BOTection: Bot Detection by Building Markov Chain Models of Bots Network Behavior

Bushra A. Alahmadi
University of Oxford, UK
bushra.alahmadi@cs.ox.ac.uk

Enrico Mariconti
University College London, UK
enrico.mariconti.14@ucl.ac.uk

Riccardo Spolaor
University of Oxford, UK
riccardo.spolaor@cs.ox.ac.uk

Gianluca Stringhini
Boston University, USA
gian@bu.edu

Ivan Martinovic
University of Oxford, UK
ivan.martinovic@cs.ox.ac.uk

ABSTRACT

Botnets continue to be a threat to organizations, thus various machine learning-based botnet detectors have been proposed. However, the capability of such systems in detecting new or unseen botnets is crucial to ensure its robustness against the rapid evolution of botnets. Moreover, it prolongs the effectiveness of the system in detecting bots, avoiding frequent and time-consuming classifier re-training. We present *BOTection*, a privacy-preserving bot detection system that models the bot network flow behavior as a Markov Chain. The Markov Chains state transitions capture the bots' network behavior using high-level flow features as states, producing content-agnostic and encryption resilient behavioral features. These features are used to train a classifier to first detect flows produced by bots, and then identify their bot families. We evaluate our system on a dataset of over 7M malicious flows from 12 botnet families, showing its capability of detecting bots' network traffic with 99.78% F-measure and classifying it to a malware family with a 99.09% F-measure. Notably, due to the modeling of general bot network behavior by the Markov Chains, *BOTection* can detect traffic belonging to unseen bot families with an F-measure of 93.03% making it robust against malware evolution.

KEYWORDS

Malware; Botnet; Network Security; Malware Detection

ACM Reference Format:
Bushra A. Alahmadi, Enrico Mariconti, Riccardo Spolaor, Gianluca Stringhini, and Ivan Martinovic. 2020. BOTection: Bot Detection by Building Markov Chain Models of Bots Network Behavior. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS'20), June 1–5, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3320269.3372202

1 INTRODUCTION

Botnets are the source of many Internet threats such as information or credit card theft (e.g. Aurora [5]), network service disruption through DDoS (e.g. DDoS on Estonia [22]), email spam (e.g. Geodo), ClickFraud (e.g. ClickBot), and spreading malware (e.g. Zeus). 10,263 malware botnet controllers (C&C) were blocked by *Spamhaus Malware Labs* in 2018 alone, an 8% increase from the number of botnet C&Cs seen in 2017.[1] Cybercriminals are actively monetizing botnets to launch attacks, which are evolving significantly and require more effective detection mechanisms capable of detecting those which are new or unseen.

Botnets rely heavily on network communications to infect new victims (propagation), to communicate with the C&C server, or to perform their operational task (e.g. DDoS, spam, ClickFraud). Hence, network-based botnet detectors have been an active research area aiming to either detect the C&C server and its communications [8, 15, 33], detect the infected machines (bots) [1, 13, 14], or detect the botmaster [16]. Although previous work that considers network communication patterns exists [1, 6, 13, 14, 33], as identified by Vormayr et al. [34], these either require unencrypted network traffic (e.g. [13]), multiple bot infections on the network (e.g. [13–15]), active propagation through scanning (e.g. [14]), or do not consider local bots attacking local targets (e.g. [14]). Importantly, some proposals lack a classifier's performance evaluation in detecting unseen bots — *bots not used in training the classifier.*[2] This is crucial in determining the classifiers' performance over time in detecting new bot families, and whether costly re-training of classifiers is needed.

Botnets tend to launch their attacks in bursts [11, 29], meaning they send multiple network connections in a short amount of time. This provides an opportunity to model the bot network behavior as a sequence of flows for bot detection. Recent work [19, 24] shows the effectiveness of building Markov Chains from the sequence of Android API calls for malware detection that can maintain its detection capabilities over time. Similarly, modeling bot network connections as a Markov Chain not only can provide insights on bot behavior, but can be used to build machine learning classifiers capable of detecting new bot families.

In this paper, we draw on the strengths of previous work and investigate the efficiency of applying Markov Chains to represent bot network communications sequence. We first explore the bursty nature of bots, to determine how frequently bots send network traffic. We then explore the discrepancies between bot network connections and those produced by benign applications. Bot network

ASIA CCS '20, June 1–5, 2020, Taipei, Taiwan
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6750-9/20/06…$15.00
https://doi.org/10.1145/3320269.3372202

[1] Spamhaus Botnet Threat Report 2019 - https://www.deteque.com/app/uploads/2019/02/Spamhaus-Botnet-Threat-Report-2019.pdf

[2] A comparative analysis of related work is given in Section 6.

traffic is then modeled using Markov Chains to explore its strength in capturing bots' network communication behavior during its C&C interactions, propagation stage, and attack operations.

To evaluate the effectiveness of applying Markov Chains for bot network connections, we propose *BOTection*, a novel system that detects infected hosts (bots) on a network by monitoring their network communication. We consider all types of malware-infected hosts that communicate with a command and control server (e.g. ransomware). In designing *BOTection*, we consider its real-world application by using light-weight high-level network flow features, enabling large-scale network analysis and resiliency against encrypted traffic. *BOTection* models are topology and protocol independent, enabling the detection of any type of bot network activity. *BOTection* does not require multiple bot infections on the network for detection. Instead, it focuses on detecting the various operational, propagational, and C&C bot communications, thus rendering it capable of detecting individual bots on a network. To foster further research, we open-source the implementation of our system. [3]

Our experimental results demonstrate that Markov Chains' state transitions capture the discrepancies between how bots use network communications to launch their attacks compared to normal use by benign applications. Hence, it can be effectively used to train supervised machine learning classifiers to distinguish between benign and bot network connections (bot detection). Importantly, the state transition discrepancies between bots and benign traffic led to *BOTection* detecting network communications belonging to unseen bots that launch similar attacks to known bots with 93% F-measure. Moreover, our system maintained its detection capabilities over time, even detecting malware emerging three years later.

We considered a real world application of *BOTection*: once a malicious network connection is detected, a security analyst's first action is to determine if that connection belongs to a known malware family (bot family classification). *BOTection* is capable of inferring from a bot connection the specific known malware family it belongs to with a 99% F-measure. This assists the analyst in choosing the best deterrence mechanism for a fast incident response.

2 BOTECTION SYSTEM DESIGN

BOTection system first detects a bot's network activity, then classifies such activity to a particular bot family. In designing *BOTection*, we consider its use in the real world. For example, organizations may deploy middle-boxes that intercept their network communications, thus allowing the deployment of content-based detection mechanisms. Alternatively, some organizations outsource their security monitoring to third parties (*Managed Security Service Provider - MSSP*), due to lack of cybersecurity expertise or to avoid high security investments [23]. However, these organizations require communications privacy, thus content inspection technologies are discouraged. Therefore, using non-privacy invasive features is crucial to foster *BOTection's* adoption in scale.

Considering the limitations of existing approaches, we identify five main design goals that the *BOTection* system has to fulfill: ① resilience to obfuscation/encryption by avoiding deep packet inspection (Section 2.1), ② privacy preservation (Section 2.2), ③ independence of network topology and protocol (Section 2.2), ④ high detection and family classification accuracy (Section 4.2, 4.1), and ⑤ capability of detecting unseen bots by capturing general bot network behavior (Section 4.2). We show the *BOTection* system in Figure 1, describing in this section its modules and how design goals were considered.

2.1 Network Flow Reassembly

Previous work (e.g. [27, 33]) used a high-level representation of the network (NetFlow) that are easier to obtain than full network dumps (for privacy concerns) and ensuring resiliency to encryption. Similarly, *BOTection* reassembles the network communications of the bot traces to *flows*, extracting content-agnostic features — fulfilling design goal ①. Hence, data required for supervised machine learning training are accessible, which is critical for the adoption of the system at scale.

A *flow* is a sequence of packets from a source host/port to a destination host/port that belongs to a unique TCP/UDP session. Thus, all packets in a flow are either going to (or coming from) the same destination IP address/port. We use the *Zeek Network Analysis Framework* (previously known as Bro)[4] to perform the network flow reassembly. Zeek generates 27 statistical and behavioral logs about the network communication as well as the application level protocols and exchanged payload of each network flow. As input, *Zeek* takes the captured PCAP network traces and generates a number of logs.[5] *BOTection* utilizes features provided in *conn.log*, one of the logs generated by Zeek.

Before building the Markov Chains, we split the traffic to windows of flows of length *n*. Specifically, in network security applications, the length of the flow window determines how often network flows are sampled for detection. Usually, in such systems, the traffic is separated to *time* windows, meaning the system reports detections in a fixed amount of time (e.g. 5 min). Bots are known to send traffic in bursts [11, 29], thus instead we split the *conn.log* to sub-logs each containing *n* number of flows (bursts of *n*-flows).

2.2 Connection States Extraction

To build a Markov Chain, we need to first identify its states. Hence, we use as states the behavioral features provided by Zeek in the *conn.log*, some which were previously used for malware family classification [4]. We build two types of Markov Chain models: one using conn_state as states and one that is a combination of conn_state, service, and protocol. It is worth noting that we avoid using statistical features that may be susceptible to malware evolution [10]. We describe the two type of states used in the following.

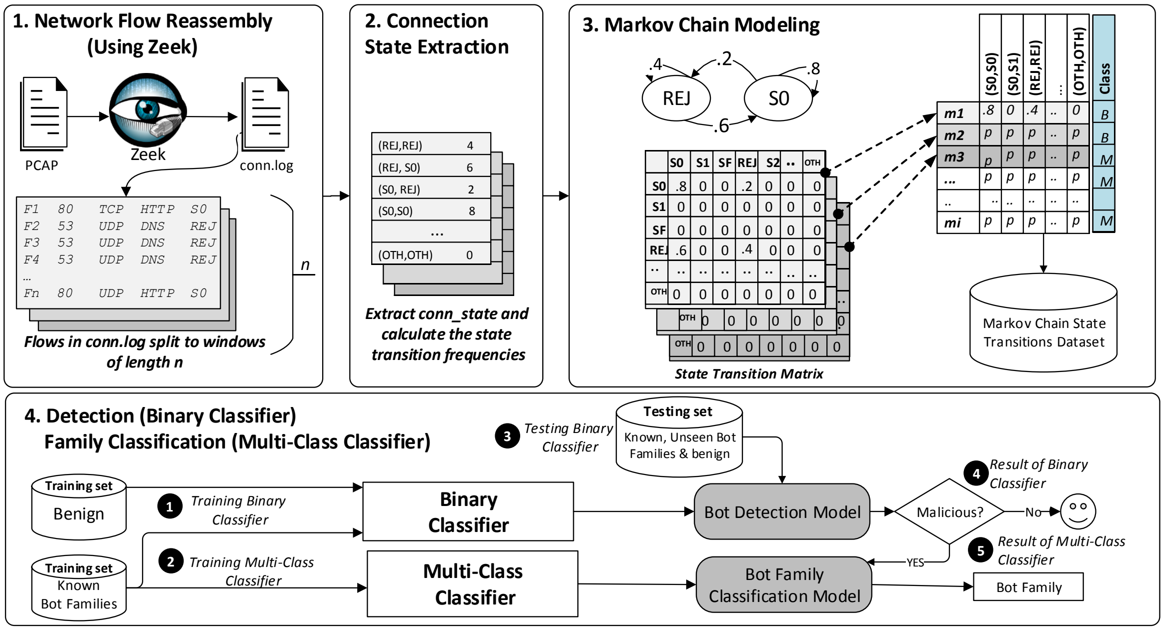
Conn_state — Zeek represents the network flow connection state (*conn_state*) using 13 different states, described in Table 1. *Conn_state* depicts the status of a bot's communication flows. For example, if a bot sends a TCP scan that was rejected by the receiver, the connection state will be *REJ*.

Protocol, Service, and *Conn_state* (PSC) — PSC is a composite of the *conn_state* attribute, connection network protocol (e.g. TCP/UDP/ICMP) and application protocol or service (e.g. DNS). An example of such a state is $(udp|dns|S0)$.

[3] https://github.com/balahmadi-Ox/botection

[4] The Zeek Network Security Monitor - https://www.zeek.org

[5] This step is not required when the network trace is pre-assembled using Zeek.


1. Network Flow Reassembly (Using Zeek)
PCAP
Zeek
conn.log
n
Flows in conn.log split to windows of length n
2. Connection State Extraction
Extract conn_state and calculate the state transition frequencies
3. Markov Chain Modeling
REJ
S0
.4
.2
.8
.6
State Transition Matrix
Markov Chain State Transitions Dataset
4. Detection (Binary Classifier) Family Classification (Multi-Class Classifier)
Training set
Benign
Training set
Known Bot Families
1 Training Binary Classifier
2 Training Multi-Class Classifier
Binary Classifier
Multi-Class Classifier
3 Testing Binary Classifier
Testing set
Known, Unseen Bot Families & benign
Bot Detection Model
Malicious?
No
YES
4 Result of Binary Classifier
5 Result of Multi-Class Classifier
Bot Family Classification Model
Bot Family

```
F1   80    TCP   HTTP   S0
F2   53    UDP   DNS    REJ
F3   53    UDP   DNS    REJ
F4   53    UDP   DNS    REJ
...
Fn   80    UDP   HTTP   S0
```

(REJ,REJ)	4
(REJ, S0)	6
(S0, REJ)	2
(S0,S0)	8
...	
(OTH,OTH)	0

	S0	S1	SF	REJ	S2	..	OTH
S0	.8	0	0	.2	0	0	0
S1	0	0	0	0	0	0	0
SF	0	0	0	0	0	0	0
REJ	.6	0	0	.4	0	0	0
..	..	..	..	..	..	..	..
OTH	0	0	0	0	0	0	0

	(S0,S0)	(S0,S1)	(REJ,REJ)	..	(OTH,OTH)	Class
m1	.8	0	.4	..	0	B
m2	p	p	p	..	p	B
m3	p	p	p	..	p	M
...	p	p	p	..	p	M
..	..	..	..	..	..	
mi	p	p	p	..	p	M

Figure 1: Overview of BOTection System. (1) Convert PCAPs to logs using Zeek, then split each log to sub-logs of n flows. (2) Extract the features (e.g. *conn_state*), producing a key-value store of state transitions and their frequency. (2) Use the state transition frequency to build Markov Chain models and produce a feature vector for each sub-log. (4) Detect malicious n-flows then classify it to a bot family.

Conn_state provides a more abstracted representation of the flow while PSC provides more granularity to the flow behavior. Although the *PSC* feature might provide a more detailed representation of the network flow behavior, we need to consider the number of states for the Markov Chain model. *Conn_state* could have 13 different values, resulting in $13^2 = 169$ Markov Chain state transitions used as features. In contrast, PSC may have various combinations of letters, resulting in over 14641 state transitions.

For each type, we extract the state transitions and calculate their frequency (number of times the state transition occurred within a window n). A state transition represents the connection states of two consecutive network flows in the log. As shown in Figure 1, the connection state transition from f_1 to f_2 is $S0 \rightarrow REJ$ (for *conn_state*) and $tcp|http|S0 \rightarrow udp|dns|REJ$ (for *PSC*). We store these state transitions for each sub-log as a key-value pair (i.e. key = (S0, REJ) and the value is the frequency = 2). These state transitions are extracted from flow headers and are thus resilient to encryption and privacy preserving — fulfilling design goals (2), (3).

2.3 Markov Chain Modeling

Markov Chains model sequences of events named states $s \in S$, where S is the set of all possible states. Markov Chain models are represented as a set of nodes (states) and edges between the nodes labeled with the corresponding transition probabilities. Markov Chains are memoryless, meaning the transition from state s_i to state s_{i+1} is dependent only on the current state s_i. The transition probabilities are calculated by observing the frequencies of each transition from each state $s_l \in S$ to state $s_j \in S$, normalized by the total number of transitions that start from node s_l. The sum of all the transition probabilities of the edges starting from any node (e.g. s_l), is one. The number of transition probabilities is the square of the number of states, as each state can be connected to all other states, including itself.

As an example, in Figure 2 we illustrate the Markov Chain of network communication for the *Neris* bot family using *conn_state* as the feature type. *Neris* performs port scanning; thus the sequences start from an OTH packet to get to either S0 (attempt to establish a connection, no reply), or REJ (connection rejected), or SF, where the connection is established. The probabilities of transition are given by the normalization of the occurrences of the transitions between OTH and the other three states. The graph demonstrates the port scanning behavior of this family, where connections were attempted and sometimes were successful ($P = 0.2$), sometimes rejected (with similar probability for this sample) and in all the other cases were ignored ($P = 0.57$).

BOTection uses Markov Chains to generate the feature vectors used for classification. Using the state transition frequencies (key-value store), we compute the transition probabilities. This is represented as the state transition matrix for each sub-log. We then represent each sub-log as a vector with all possible state transitions (i.e. keys from the key-value store) representing the features (i.e. columns). The feature vectors of all sub-logs represent our dataset. For example, if the feature type is *conn_state*, then the states in the Markov Chain model are those defined in Table 1. The 13 different *conn_state* states result in 169 transitions. Hence, for each sample sub-log, the probability of transition from one state to the other in the Markov Chain represents the feature vector for that sample.

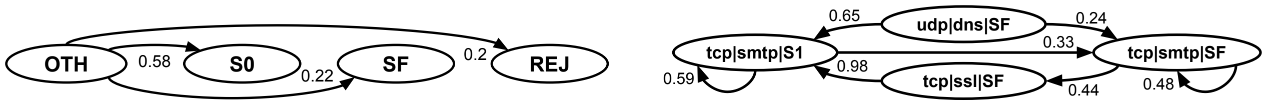


Figure 2: Markov Chain for Neris ICMP port scanning (left) and Zeus C&C (right)

Table 1: Description of the flow connection state (*conn_state*) feature obtained from Zeek *conn.log* logs.

State	Description
S0	Connection attempt seen, no reply.
S1	Connection established, not terminated.
SF	Normal establishment and termination.
REJ	Connection attempt rejected.
S2	Connection established and close attempt by originator seen (but no reply from responder).
S3	Connection established and close attempt by responder seen (but no reply from originator).
RSTO	Connection established, originator aborted (sent a RST).
RSTR	Responder sent a RST.
RSTOS0	Originator sent a SYN followed by a RST, but did not receive a SYN-ACK from the responder.
RSTRH	Responder sent a SYN ACK followed by a RST, we never saw a SYN from the originator.
SH	Originator sent a SYN followed by a FIN, we never saw a SYN ACK from the responder.
SHR	Responder sent a SYN ACK followed by a FIN, we never saw a SYN from the originator.
OTH	No SYN seen, just midstream traffic.

2.4 Classification

Using feature vectors of Markov Chain state transitions belonging to bot and benign samples, we train supervised machine learning models. Particularly, we use an ensemble classifier known to overcome over-fitting issues. Since our system carries out two subsequent tasks (i.e. bot detection and bot family classification), we adopt two different classification approaches discussed in the following.

Binary Classifier (Bot Detection Model) — Binary classification aims at partitioning the samples in the feature space in two distinct and complementary sub-spaces. The partition is done in such a way that a sub-space includes samples of a class while the complementary sub-space includes the samples of the other class. In our system, the classifier is trained using *n*-flow samples of known malware families and benign traffic. The trained model is then used to classify an *n*-flow into either the bot class (i.e. Malicious) or the benign class (i.e. not Malicious).

Multi-Class Classifier (Bot Family Classification Model) — Since our family classification task aims at discriminating between malicious flows of different bot families, we rely on a classification approach that partitions the feature space in multiple non-overlapping sub-spaces. The multi-class classifier in our system is trained using malicious *n*-flows belonging to multiple families. Thus, once an *n*-flow is classified as malicious by the *Bot Detection Model*, it is then attributed to a malware family by the *Bot Family Classification Model.*

3 EXPERIMENTAL SETUP

We implement the system as a Python 2.7 application using the *Scikit-learn* library. [6] The experiments were conducted on a 40-core processor with 128GB RAM and 20G disk space, Centos OS. We open-sourced the implementation of our system as a contribution. [7] We design the following experiments to determine BOTection's capability in modeling bot network communication, that can be used for bot detection and family classification.

Bot and Benign Communication Patterns (results in Section 4.1) — We model the bots' and benign communications as a Markov Chain to identify bots' network behavior and explore the discrepancies in bot and benign state transitions.

Bot Detection (results in Section 4.2 and Section 4.3) — We build and evaluate two bot detection classifiers to: ① detect known bots and ② detect previously unseen bots. The objective of the unseen bot classifier is to assess the classifiers' genernalylizability in detecting communication for bot families that were *not* considered in classifier training. We also attempt to answer the question '*What if a few wrong flows end up within the window n?*' Hence, we study the effect of injecting benign flows into the malicious *n*-flows on the system detection.

Bot Family Classification (results in Section 4.4) — We identify network communications that are unique to each bot family. Thus, once an *n*-flow is identified as malicious by the bot detection classifier, we evaluate the multi-class classifier's accuracy in classifying these *malicious n*-flows to a bot family.

3.1 Datasets

We use two datasets to evaluate the binary classifier (bot detection) and multi-class classifier (bot family classification). In Table 2, we show the number of malicious flows (both datasets) and benign flows (ISCX only) and the percentage of flows belonging to each family. We also show in Table 7 the bot families and the various bot attacks captured in our datasets.

ISCX Botnet Dataset — for Binary Classifier. This dataset contains malicious and benign traffic and is split into training and testing sets with 5 and 13 bots respectively, where 8 bot families in the testing set were not used to train the classifier (*unseen bots*). We use this dataset to train and evaluate the binary classifier *using malicious and benign network traces*, in detecting unseen bots. This is due to the testing set containing bot families that were not present in the training set. The dataset contains traffic from 5283 and 9896 benign hosts for training and testing respectively with 27 bot-infected hosts. The benign traffic in the dataset contains activities such as SSH, HTTP, Web browsing, World of Warcraft gaming, bit-torrent clients such as Azureus, web and email traffic

[6] Scikit-learn: machine learning in Python - https://scikit-learn.org/

[7] https://github.com/balahmadi-Ox/botection

Table 2: Datasets used for evaluation *- ISCX contains bot and benign flows used in binary classifier, MCFP contains bot traffic only for multi-class classifier.*

Dataset	#Flows	Bot Families
ISCX Botnet Dataset	Training - 6,925,812 flows - 43.92% Malicious Testing - 5,040,068 flows - 44.97% Malicious	***Training:*** Neris (12%), Rbot (22%), Virut (0.94%), Zeus (0.01%), Zeus C&C (0.01%) ***Testing:*** *Known Bots* — Neris (5.67%), Rbot (0.018%), Virut (12.80%), Zeus (0.109%),Zeus C&C (0.006%) *Unseen Bots* — Menti (0.62%), Sogou (0.019%), Murlo (0.16%), Tbot (0.283%), Zero Access (0.221%), Weasel (9.25%), Smoke Bot (0.017%), ISCX IRC Bot(0.387%)
MCFP	7,283,060 Bot flows	Miuref (2.8%), Zeus (3.5%), Geodo (3.4%) NSIS (0.3%), Bunitu (2.7%),WannaCry (0.9%), Conficker (0.8%), Zeus C&C (8.6%), Neris (2.8%), Rbot (4.5%), Virut (1.23%), Sality (20.7%), TrickBot (0.5%), Stlrat (5.3%), Donbot (0.07%), Papras (41.8%), Qvod (0.1%)

and backup and streaming media, and SMTP mimicking users' behavior. We refer the reader to [7] for a detailed description of the dataset.

Stratosphere IPS Project — for Binary Classifier. This datatset provided by *The Stratosphere Research Laboratory* [8] contains over 300 malware samples of current malware that was first seen in the wild between years 2013-2018 such as *Wannacry*, *Emotet*, and *BitCoinMiner*. This dataset is used to verify the binary classifier's performance over time, thus determining the date the samples were first in the wild was essential. We verified each samples date of "*first seen in the wild*" using VirusTotal[9] reports.

Malware Capture Facility Project (MCFP) and Stratosphere IPS Project — for Multi-class Classifier. To train our multi-class bot family classifier, we used the CTU-13 botnet traffic dataset provided by the Malware Capture Facility Project [12]. In addition, we use the samples of newer bots (e.g. *Trickbot - 2016*) and ransomware (e.g. *WannaCry - 2017*) collected by *The Stratosphere IPS Project* . The multi-class classifier classifies detected malicious traffic to a family, therefore, only *malicious* traffic (C&C communication and bot communication) is used to train and evaluate the classifier. The botnet families represented in the dataset employed various protocols (e.g. *IRC, P2P, HTTP*) and various techniques (e.g. *spam, DDoS*), as shown in Appendix - Table 7. The motivation for using this dataset is that network-traces of real botnet attacks were captured, thus generating reliable datasets for model building.

3.2 Classifier Design

For bot detection, we use a binary classifier which classifies an observation as malicious or benign. For bot family classification, we use a Multi-Class classifier, meaning an observation is classified into one of multiple bot family classes. For both approaches we use Random Forest classifiers, an ensemble method that builds a strong classifier relying on the consensus among classification results from a number of weak learners (i.e. estimators), thus overcoming overfitting issues of an individual learner. In our work, we consider a Random Forest composed of 101 decision trees as weak learners.

To address the class imbalance in the MCFP dataset used for bot family classification, we apply a method that balances the training set by associating a weight to samples of a specific class according to the number of samples that class includes. Hence, it associates a higher weight to samples of under-represented classes. In the Scikit-learn library, Random Forest classifier handles imbalanced classification problems by setting the parameter $class\_weight = balanced$.

Window Flow Size. The window flow size represents the number of flows in each sample in our dataset. For each sample *conn.log* in our dataset, we split the log into a number of sub-logs of size n. Therefore when the window size $n = 15$, that means each observation in our final dataset contains bursts of 15 flows (15-flows for short). We show the cumulative number of flows generated by a bot-infected host per bot family during the first 7 minutes in Figure 3. All bots generate 15 or more flows within less than 170 seconds, then continuously send over 15 flows each second. This confirms our system design hypothesis that bots send traffic in bursts. To explore the ideal window size, we experiment with various configurations for the window n, where $n = 10, 15, 20, 25, 30, 35$.

Classifier Performance. To assess the performance of the known bots binary classifier and multi-class classifier, we apply stratified 10-fold cross-validation. For the unseen bot classifier, we use the training set to train the classifier and evaluate using the testing set containing samples of bot families not considered in training. We employ the evaluation measures of *Precision* and *Recall* to evaluate classifiers' performance. For the binary classifier, we also measure the *False Positive Rate* (FPR) and the *False Negative Rate* (FNR).

4 RESULTS

4.1 Bot and Benign Communication Patterns

To understand the discrepancies of benign and bot network flows, we visualize the average Markov Chain *conn_state* state transitions in Figure 4 for bot and benign samples. Benign state transitions are concentrated on a few state transitions compared to bots, which are more diverse. There are similarities in both, which is due to how the Internet works and similarities in the traffic. For example, although both have a high transition probability for $OTH \rightarrow OTH$, the diversity of the states in a bot's n-flow makes it detectable. Bots generate more unique connection state sequence behavior that benign traffic does not attempt.

Bot Traffic Over Time. We show the overall amount of traffic by each bot family in Table 2 over time in Appendix - Figure 12. The majority of the bot families constantly produce a significant amount of network traffic during the different phases of its malicious operations. To understand the various communications a bot

[8] https://stratosphereips.org/

[9] www.virustotal.com

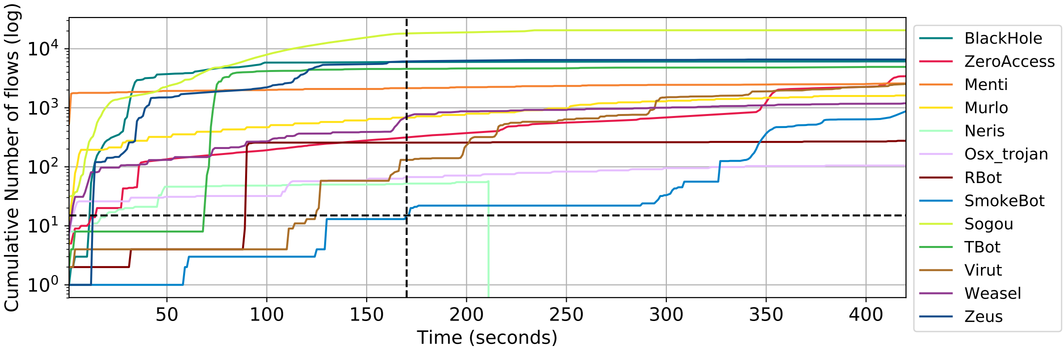


Figure 3: Cumulative number of flows (log scale) generated by each bot family every second over a time frame of 7 minutes.

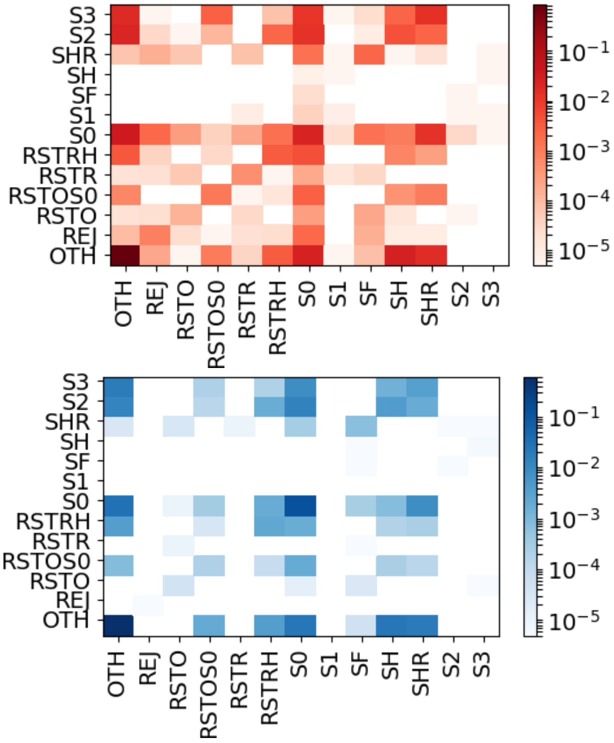


Figure 4: Average connection state transitions for network communications of bot (red-upper) and benign (blue-bottom) samples in ISCX dataset.

performs during propagation, C&C communication, and other attack operations we modeled the communication as a Markov Chain. We discuss the most prominent bot network communication found in the MCFP dataset. In order to simplify the Markov Chain for visualization, we show only the state transitions with probability $p > 0.1$.

C&C Communication. We show Zeus's C&C communication Markov Chain in Figure 2. Zeus is known to initiate the communication to its C&C by sending a *GET* message, receiving a reply with the encrypted configurations. Zeus then sends encrypted stolen information via a *POST* message [9]. This can be seen in the Markov Chain with the HTTP connection with state *SF* followed by an SSL connection with $p = 0.44$. Following the SSL state, the bot establishes a connection with the C&C ($p = 0.98$) that is terminated by the bot.

SMTP Spam. Multiple bots in our dataset send email spam, specifically *Neris*, *Sality* and *Geodo*. For brevity, we show the average of all the Markov Chains for the Geodo bot performing email spam in Figure 11. We found 29, 575 Simple Mail Transfer Protocol (SMTP) flows in our dataset for Geodo. Overall, there are 2 types of SMTP connections found. One was not terminated (SF), whilst the second was terminated by a *FIN* bit sent by both ends (S1). These two types were captured in the Markov Chain model, showing the next state after a completed SMTP connection is a DNS connection ($P = 1$), whilst an unterminated connection will either (1) remain in the same state ($P = 0.41$), (2) receive a SYN Ack followed by a FIN from destination $(tcp| - |SHR)(P = 0.27)$, (3) make a DNS request ($P = 0.32$).

Port Scanning. We were also able to identify ICMP port scanning (Figure 2), where an ICMP connection flow with *conn_state OTH* is followed by a UDP flow with states *S*0, or a TCP connection with *conn_state SF* or *REJ*. Bot families *Rbot*, *Neris*, *Virut*, *Qvod*, and *Donbot* all perform port scanning and have shown to have similar state transitions.

DDoS. We are also able to observe two types of DDoS attack, TCP and ICMP DDoS in *Rbot*. DDoS was observed as multiple connection requests sent to a single host, where these requests could be an ICMP request or TCP request. For ICMP DDoS, the bot establishes an ICMP request which has a *conn_state OTH*, whilst TCP DDoS connections have *conn_state REJ*.

4.2 Performance of Binary Classifier

Classifier Performance. We show the performance of the binary classifiers in detecting known and unseen bots in Figure 5, with the flows' *conn_state* and *PSC* as Markov Chain states. When analyzing the known bots, the flow window had no effect on the classifier performance and the classifier performed well (99% F-measure). However, when we try to identify unseen bots the detection performance varies based on the window size: larger window sizes allow a better identification in the *PSC* while smaller ones work slightly better for *conn_state*. Overall, *conn_state* works best for the goal of detecting network connections belonging to unseen bots. Specifically, for detecting unseen bots, *conn_state* had the best performance with a detection rate of 93.03%, when $n = 15$. In contrast, using PSC as a feature yielded a better accuracy (91.06%) when ($n = 35$).

We also report the FPR and FNR for each classifier in Table 3. The Unseen Classifier performed best when $n = 10, n = 15$ with 0.4% and 5.87% FPR respectively. The FPs originated from only a small number of hosts and for specific cases. We discuss this in more detail in Section 5.1.

Classifier Performance Based on the Type of Feature. Clearly, *conn_state* performance was best in detecting known (F-measure= 99.78%), and unseen bots (F-measure= 93.03%) compared to PSC where detection efficiency was high for unseen bots only when the burst window size was above 20.

Classifier Performance When Injecting Benign Flows. We explore the effect of injecting random benign flows b into the bot 15-flow (n = 15) at random locations on the detection performance. In practice, this means that we inject states that are randomly selected among the most frequent *conn_states* in benign traffic, as reported in Section 4.1 and shown in Figure 4. For example, *REJ conn_state* was not injected in this experiment as it rarely occurs in benign traffic.

In this experiment, we injected benign flows (i.e. b random benign *conn_states*), in the malicious 15-flows. For example, for 15-flow of malicious flows, we can inject 2 benign flows (b = 2), meaning that the other 13 flows are malicious ($m = 13$), where ($n = b + m$). We conducted experiments with various configurations of b and m. For each configuration, we repeated the experiment 40 times, and obtained the average of number of detected malicious flows by the *known bot classifier*.

We show the results of this experiment in Figure 6. The x-axis represents the number of benign flows injected b. For example, when less than 4 benign flow was injected ($b = 4$, $m = 11$), the classifier was capable of detecting 80% of the 15-flows even after injecting a random state into the sequence. This may be due to the diversity of the states in a bot n-flow, as discussed in 4.1.

4.3 Classifier Performance Over Time

Machine learning detectors need to maintain their performance over time to facilitate real world application. This is crucial to prevent the costly re-training of classifiers and ensuring its detection capability to new malware. As such, we conduct an experiment using current malware from the Stratosphere IPS dataset.

In training and testing the classifier, we take into consideration the *Temporal Training Consistency*, where all samples in the training are temporally precedent to samples in the testing set [25]. Thus we split the test set, depending on the year the malware was "*first seen*", which was verified from *VirusTotal* reports.

We show the results of training the binary classifier (n = 15) in Table 4. Interestingly, the classifier's performance over time is 86% when trained with samples appearing before 2015. However, the classifier performance increases to 98% when trained with samples appearing after 2015. These results show that the binary classifier generalizes well enough to not be affected by the temporal bias problem defined in [25]. This means that our classifier (i) can detect traffic from previously unseen bot families, and (ii) is robust over time, thus it does not need to be frequently re-trained.

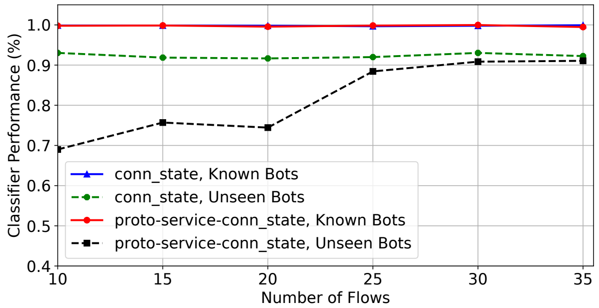


Figure 5: Binary classifier F-measure for detecting known/unseen bots for each state type.

Table 3: False Positive Rate (FPR), False Negative Rate (FNR) of the known and unseen classifiers for each value n.

Classifier	n	conn_state		PSC	
		FPR	*FNR*	*FPR*	*FNR*
Unseen Bots	10	0.004	0.111	0.353	0.256
	15	0.060	0.096	0.386	0.055
	20	0.100	0.074	0.407	0.056
	25	0.093	0.073	0.175	0.043
	30	0.079	0.065	0.142	0.030
	35	0.097	0.067	0.142	0.026
Known Bots	10	0.003	0.000	0.002	0.003
	15	0.003	0.001	0.001	0.002
	20	0.004	0.001	0.001	0.008
	25	0.008	0.001	0.003	0.001
	30	0.004	0.001	0.000	0.000
	35	0.001	0.000	0.002	0.009

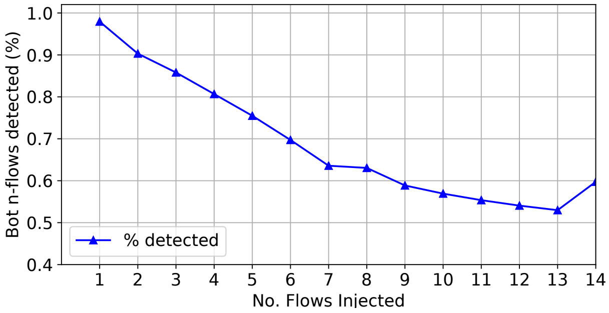


Figure 6: Percentage of bot 15-flows detected when we inject b benign flows in the sequence.

4.4 Performance of Multi-Class Classifier

Markov Chain Models. Due to space limitations, we show the Markov Chain state transitions for the *conn_state* feature in Figure 7 for some bot families. Highlighting only the most important state transitions for each bot family, we show in the figure only state transitions with probability > 0.6. We notice that connection transition $SF \longrightarrow SF$ and $S0 \longrightarrow S0$ are seen in most bot connections. Having an $SF \longrightarrow SF$ is expected, as bots will typically have normal connections.

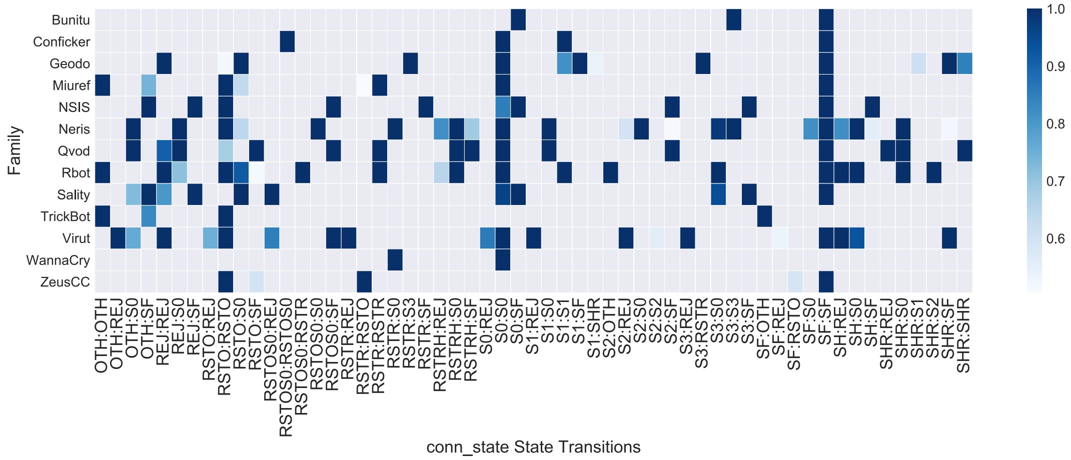


Figure 7: Markov Chain Model Transition Probabilities of the *conn_state* feature for each botnet family. For brevity, we show only transition probabilities greater than 0.6. For example, Virut had a high transition probability between states S1 and REJ.

Table 4: Binary Classifier Performance: F1 (F-measure), Precision (P) and Recall (R) over time with *Temporal Training Consistency*. We train the classifier using bot samples first seen in the years precedent of the samples in the testing set.

Training set	Testing set											
	2015			*2016*			*2017*			*2018*		
	F1	P	R	F1	P	R	F1	P	R	F1	P	R
Pre 2015	.86	.89	.87	.87	.89	.87	.86	.89	.87	.86	.89	.87
Pre 2016	-	-	-	.987	.987	.987	.983	.983	.983	.982	.982	.982
Pre 2017	-	-	-	-	-	-	.981	.982	.981	.981	.981	.981
Pre 2018	-	-	-	-	-	-	-	-	-	.989	.989	.989

Classifier Performance. We show in Figure 9 the performance of the multi-class bot classifier for each of our connection state types. When the flow window size is at least 20, the scores are stable and close to 100. Our classifiers performs well, reaching 98% F-Measure score for bot family classification when $n = 20$, reaching its maximum 99.09% when $n = 35$. The *conn_state* feature performs very well independently from the window size value, while the other feature sets had these performances with $n \geqslant 20$. We show the classifier's performance in more detail per bot family when $n = 20$ in Figure 8. As discussed in 3.2, we take precautions to ensure that the classifier is not impacted by the imbalanced training set. This is shown in Figure 8, as the classifier maintained a high precision per bot family. We observe that the *Geodo* bot is the only family presenting an F-Measure score less than 90%, with some connections misclassified to either *Bunitu* or *Stlrat*.

Classifier Performance Based on Type of Feature. *Conn_state* performed the best across the various window sizes, however, we find that PSC might provide a more complete representation of a bot's behavior. For example, we observed for *Virut* that state transitions from $S0$ to $S0$ for TCP flows were of a spam attack. However, when the bot used a UDP connection for DNS services ($udp|dns|S0 \longrightarrow udp|dns|S0$), it indicated that it is sending DNS queries. While these differences are transparent to the *conn_state* features, they are shown by the PSC. The email spam was confirmed when we uncovered that the destination port was port 25. Hence, in certain attacks, destination port, service, and protocol are crucial to distinguish types of attacks with similar state transitions.

5 DISCUSSION

5.1 Understanding Classifiers' Errors

Binary Classifier: False Positives. We focus our discussion here on the performance of the binary classifier (for unseen bots). The classifier misclassified 20561 (5.87%) benign n-flows of the testset as malicious. In a real world setting, a defender (e.g. analyst) needs to know which are the infected hosts, in order to remove them from the network. The process of investigating FPs is known to be a tedious tasks for analysts [29]. However, we found that the falsely-labeled flows originated from only a small number of hosts ($FPR > 0.1$) — 36 out of 1164 benign hosts, as shown in Figure 10.

For example, benign host IP = 10.0.0.254 had the highest percentage of FPs, where 87% of its benign traffic was flagged as malicious. The misclassified n-flows had either only *conn_state* OTH or S0 — a state transition that we found to be high in both benign and malicious (Figure 4). The FPR could be improved by white-listing connection states that are highly present in benign and malicious flows or n-flows containing only one kind of state (e.g. only OTH). Therefore, although the FPR might look high, investigating these false positives is a rather reasonable load for a security defender.

Binary Classifier: False Negatives. We attempt to understand the misclassification of bot flows to benign of the *unseen* binary classifier. In Table 5, we show the number of false negatives for each bot family, where $n = 15$. Over the $136,288$ bot 15-flows in our testing dataset, 8124 were misclassified as benign. Interestingly, *Virut* bot produced the highest number of False Negatives, although the bot family was considered in the training set. 99.87% of the misclassification for *Virut* are 15-flows with three connection state transitions (4 $RSTRH \longrightarrow S0$, 4 $S0 \longrightarrow RSTRH$, and 7 $S0 \longrightarrow S0$). These misclassified connection states represent *Virut's* IRC communication and spam behavior. *Virut* sends IRC communication to port 6667 with $conn\_state = RSTRH$ followed by sending spam on port

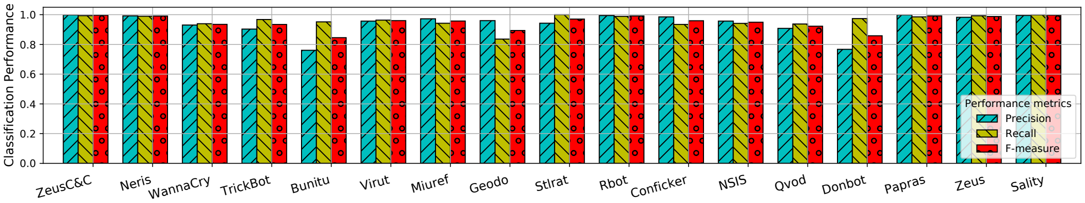


Figure 8: Multi-Class classifiers' performance per bot family (n=15).

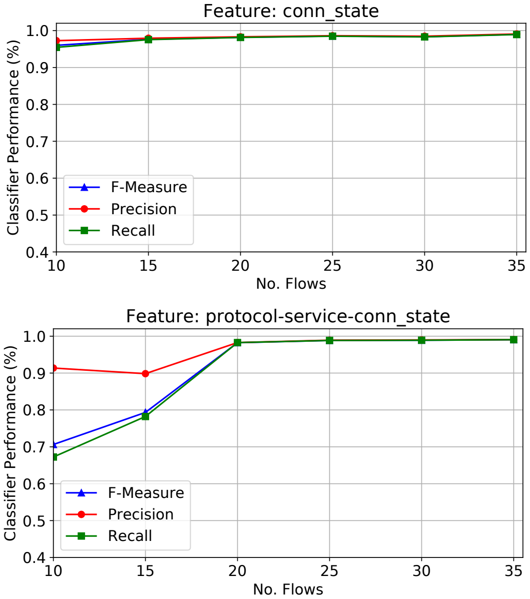


Figure 9: Multi-Class classifiers' performance for each Markov Chain state feature type.

25 with $conn\_state = S0$. Such state transitions were not seen in the training set for *Virut* and other bot families, while present in 57% of the testing set explaining the misclassification. On examining these misclassified flows, we found that in our training set, Virut attempts to establish an IRC connection by sending packets to a server that either rejects the connection (REJ or $S0$) or accepts the connection (SF). In contrast, the testing set contains unsuccessful IRC connections ($S0$) to the server followed by a $RSTRH$ response from the server (meaning the connection was timed out). Hence, this is not a result of Virut changing its behavior but was a result of the server delay in its reply and connection timing out.

Multi-Class Classifier. We attempt to investigate the misclassification (when $n = 20$) per bot family. *Geodo* had the lowest F-Measure score of 84%, with some 20-flows classified to *Bunitu* or *Sltrat*. Most flows misclassified to *Bunitu* are connections with only state transition $SF \longrightarrow SF$. This explains the misclassification, as the classifier learned that 20 consecutive SF flows is a behavior of *Bunitu*, resulting in the low classification accuracy for *Geodo*. However, as $SF \longrightarrow SF$ state transition represents a normal network connection, n-flows with only connection state SF should be whitelisted. Similarly, *Geodo* flows with only $S0 \longrightarrow S0$ transitions are classified as *Sltrat*, which has the same flows.

5.2 Lessons Learned

Emerging bots are stealthy and may seek a passive propagation approach (e.g. *spear phishing*), avoiding noisy active propagation (e.g. *scanning*) that some previous bot life-cycle detection approaches (e.g. [1–3, 13]) expect. In Figures 2 and 11 we showed the Markov Chain for various bot communications such as C&C communications, ICMP scanning and email spam respectively. Each bot attack resulted in a different Markov Chain representing that attack. These models are not only valuable in understanding bot behavior but may be used to determine bot attack stages and whether bots do actually follow the *bot life-cycle*.

Although bots share similarities in their Markov Chains, there are still differences that make it possible to distinguish between the network traffic of different bot families. *BOTection* was able to classify a n-flow to its bot family with 99% accuracy — fulfilling design goal ④. Each bot family had a set of uniquely identifiable Markov Chain state transitions (Figure 7). In fact, bots may operate the same attack in different ways. For example, Figure 11 shows the behavior of two spam families, *Geodo* and *Sality*. They both successfully use the TCP SMTP protocol ($tcp|smtp|SF$) and perform DNS queries through UDP ($udp|dns|SF$), but they use them in different ways. When *Geodo* successfully sends an email, it always goes back to the DNS queries ($P = 1$). However, *Sality* is often sending many emails in a row as state $tcp|smtp|SF$ has 0.64 probability of returning to itself in the next step of the chain. Another example is the Markov Chains for *Papras* that shows TCP DNS requests, while normally DNS is sent over UDP. This may be due to the DNS response or request data size exceeding the size of a single packet. *Papras* flows in our dataset contain DGA communication; this unique DNS request behavior is peculiar of *Papras* DGA behavior that is captured by our system.

We explored two types of states used in the Markov Chain models. Both *conn_state* and PSC are effective in detecting bot communications, but PSC provides a higher level of granularity. For example, a flow with *conn_state SF* might be sent using either UDP or TCP, and might be an SSL communication, a DNS request, or an HTTP request, each representing a different flow behavior. However, PSC has 404 state transitions compared to $conn\_state$ ($13^2 = 169$) increasing the dimensionality thus the computation needed for classifier training. We highlight the importance of evaluating the system performance in detecting bots not used when training the classifier. Our system was able to detect known bots with an F-measure of 99% and new bots with an F-measure of 93% — fulfilling design goals ④ and ⑤.

Bot Markov Chain models are different to benign ones as bots have more diverse state transitions, as seen in Figure 4. Hence, *BOTection* was able to capture bot behavior and recognize it in flows

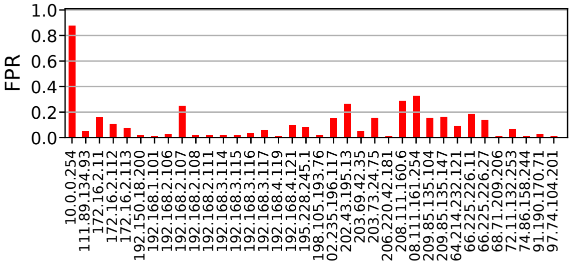


Figure 10: Host IPs that produced the highest FPR for the binary classifier (*FPR > 0.1%*)

of families not included in the training set. Markov Chain models are robust to bot evolution due to the abstraction of network communication to finite states, thus less susceptible to the introduction of new states. Applying *conn_state* in the Markov Chain models ensures that no additional states could be introduced.

We evaluated our binary classifier's performance over time. Following recommendations provided in [25] to overcome the temporal bias problem, we trained the classifier with samples first seen in the years preceding the test samples, complying with the *temporal training consistency*. We also ensured that samples in the test sets have been seen for the first time in the same year. As we reported in Table 4, our method maintained its performance over time with a 98% F-measure. This ensures that a classifier can detect new bots without frequent and time-consuming re-training.

For binary classification, we used the ISCX dataset that contains network flows labeled (malicious/benign) according to the host generating them. We studied the effect of mixed traffic on *BOTection's* performance by injecting a benign flow in the malicious 15-flow. *BOTection* was able to still detect 80% of the malicious *n*-flow even when four benign flows were injected into the bots 15-flow. As shown in Figure 3, bots have a "bursty attacks" nature that can be seen in network traces [11, 29]. We observed that bots in our dataset generated 15 or more flows within less than 3 minutes from their first execution. Such bots also sent a significant amount of flows every second during their various attack stages (see Appendix - Figure 12). Hence, *BOTection* is capable of detecting a bot-infected host during its early communications as detecting a single burst can lead to its discovery.

5.3 Evasion and Limitations

Bots may evade detection by reducing the number of network communications or increasing the time interval between them. Our detection method relies on a bot's bursts of network flows, thus the detection of a stealthier bot may be a challenge. Moreover, fewer flows may affect the reliability of the Markov Chain, as the statistical models can change. Nevertheless, we argue that most bot operational communications (e.g. *scanning, spam, DDoS*) inevitably generate communication bursts. In particular, a spam bot sends a high number of SMTP packets in a few seconds, while a DDoS attack misuses specific packets. Those operations result in network communication behaviors that are different from benign user activities. Thus, a bot would not be able to carry out its operational communications while attempting to evade detection at the same time. Covert channels [35] can be used by bots (e.g. encoding the data in TCP/IP header bits). Another evasion technique is imitating patterns of benign communications. These patterns can be replicated only to some extent (e.g. sending flows with *conn_state* = SF) and for specific particular activities (e.g. a spam bot mimicking normal traffic results in sending fewer emails). Although these evasion techniques are already implemented by some samples in the wild, their evasion efforts might introduce constraints on their behavior, affecting their full functionality [31].

Table 5: Number of flows, 15 window length samples, and number of bots per family in the ISCX testing dataset, and the number of False Negatives (FN) for each bot family and benign traffic.

Family	Total # Flows	# 15-flow window	# Hosts	# FN
IRC Attack	773370	51558	8	107
Menti	13024	1628	1	8
Murlo	81135	5409	1	68
Neris	402000	2680	1	28
Osx_trojan	375	25	1	0
Rbot	116505	7767	1	10
SmokeBot	1755	117	1	0
Sogou	20625	1375	1	1
TBot	5205	347	4	5
Virut	437550	29170	1	7726
Weasel	508080	33872	2	165
ZeroAccess	19440	1296	2	9
Zeus	15660	1044	4	9
Benign	**2,948,550**	**196570**	**1164**	**16100**

BOTection uses states that are protocol dependent, thus an attacker might attempt to evade the detection by changing the protocol (e.g. from TCP to UDP). However, changing the protocol limits bot capabilities. A bot would not be able to carry out DDoS attacks with a specific method protocol different from the original one (e.g. ICMP ping of death, UDP Flood, SYN packets on TCP handshake). ClickFraud uses HTTP, thus it is dependent on the TCP protocol. Despite SMTP using TCP, UDP is also used in rare benign applications so can be easily detected by a rule-based system. Bots are known to have similar attack objectives, thus attempting to change its network behavior will be detected as its new behavior may be similar to another family's attack behavior.

As with other machine learning based approaches, a bot might try to evade *BOTection* via adversarial learning techniques [17]. It might also try to mimic the patterns of multiple families to impact the classification. However, the main objective of malware family classification is to identify the one that exhibits the most similar network behavior to assist the defender in deploying proper mitigation. In the case of a new bot family, one solution is to classify such malicious traffic to a "New Family" class using the confidence of multi-class Random Forest classifiers applied previously in [32]. Thus, classifications with a confidence level below a threshold will not be attributed to a known family but as "New Family".

In evaluating our system, we considered its performance in detecting traffic generated by bot families that were not included in the training set. Our system was able to detect such traffic with a 93% F-measure due to these bot families exhibiting similar attacks

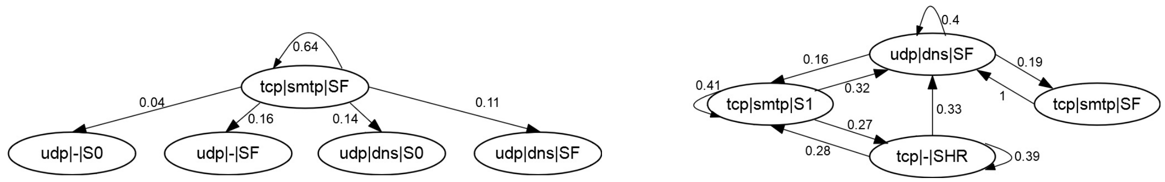


Figure 11: Markov Chain for Sality (*right*) and Geodo (*left*) spam behavior (only state transition probabilities, where $p > 0.1$).

(e.g. DDoS) to families that are present in the training set. Hence, unseen bots that use novel techniques (i.e. new types of attacks that use protocols in a unique way) may not be detectable in our system (e.g. zero-days).

6 RELATED WORK

In reviewing the literature, we consider infected host (bot) detection and classification of malware family using network flow and behavior modeling via Markov Chains. We refer the reader to a comprehensive survey on botnet detectors [16].

Markov Chain Modeling. Mariconti et al. in [19, 24] used Markov Chains to fingerprint malicious and benign behaviors in terms of sequences of Android API calls. In [18] Markov Chains were used to distinguish between network packages generated by targeted malware samples and generic ones. Abaid et al. [2] applied Markov Chain, modeling the botnet's infection sequence, to identify behavior that may lead to an attack. However, they do not consider detection of unseen bots and rely on content-based inspection systems (e.g. Snort) to generate the Markov Chain states which do not tolerate encrypted traffic. Mcgrew et al. [20] identified intra-flow features, modeled as a Markov Chain, that could be used for machine learning classifiers to identify threats for encrypted traffic. However, they did not evaluate the classifier's performance over time in detecting new malware families.

Malware Family Classification. Malware family classification using network flow clustering was proposed by Perdisci et al. [26] to derive HTTP network behavior similarities of HTTP-based malware, building malware detection models. Similarly, Firma [28] applied clustering to generate network signatures for malware network traffic detection. Mohaisen et al. [21] classified malware to families through behavioral analysis of malware network behavior by applying n-gram document analysis to train supervised classification models. AlAhmadi et al. [4], propose a system that classifies network flow sequences to a malware family using Bro/Zeek generated features. However, the feature generation process has a high computation complexity in comparison to *BOTection.*

Bot Detection. *BotHunter* [13] explored the detection of the bot network activities using infection dialog to detect the multiple stages of the botnet lifecycle. As discussed in [34], *BotHunter* is only effective on unencrypted network traffic and requires multiple bot infection on the network to detect bot activity. Similarly, *BotMiner* [14] requires multiple bot infections for detection and does not consider active bot propagation through scanning [34]. However, it does improve on *BotHunter* by not requiring unencrypted traffic. In [6], the authors extended the infection dialog proposed by *BotHunter* [13] to include passive propagation mechanism such as spam. Abaid et al. [1] analyzed botnet network behavior and identified those that are synchronized across multiple bots. However, they only focused on the detection of spam and port scanning.

ProVeX [30] is a system that focused on the detection of C&C using encrypted communications by focusing on network messages and extracting the C&C protocol semantics. *BotSniffer* [15] proposed a network anomaly based botnet detection system to detect malware (in particular botnets) C&C communication using protocols such as IRC and HTTP. The approach utilizes spatial-temporal correlation and statistical algorithms to detect strong synchronization of hosts' network traffic. The authors hypothesized that botnet C&C communication has a certain pattern that is a result of pre-programmed activities, thus hosts infected with a similar botnet will have similar network patterns that are different from benign hosts. However, this approach requires multiple infected bots on the same network. DISCLOSURE [8] also aimed at detecting C&C communication, improving detection accuracy by including blacklists of known IP addresses and domain names of C&C servers. However, detection of unseen bots was not evaluated [34].

BotFinder [33] is a botnet detection system that monitors network traffic for C&C communication to detect bot-infected hosts. Compared to *BotSniffer* [15], the approach detects individually infected hosts and therefore does not require correlating network activities of multiple hosts. Moreover, it does not rely on payload information but high-level network data such as *NetFlow* in the detection, thus applicable also for encrypted botnet traffic. However, BotFinder uses statistical time-related features (e.g. *time interval)* of the network flows assuming that bots use constant time intervals between C&C communications. However such time-related features are affected by the network quality (i.e. speed) and may vary in future unseen malware variants [10]. In contrast, *BOTection* uses behavioral features that are resistant against such variations, as proven by its effectiveness in detecting unseen bots.

In Table 6, we provide a comparative analysis on *BOTection* and previous work on bot detection. Specifically, we compare based on the type of detector, its resiliency to encryption, single bot detection capability and its evaluation of the detection performance of unseen bots. To ensure a fair and systematic evaluation, we provide only an empirical comparison of the detection results to previous work that use same dataset. For example, Zao et al. [36] used the ISCX dataset for bot detection yielding a true positive rate of over 90% and a false positive rate under 5%. In addition, they evaluated their system in detecting unseen bots, which, although yielding a high detection rate, had a FPR (*specifically for Weasel*) of 82%. A high FP increases the burden on analysts to filter out these false positives, thus is not adoptable in real world settings. In contrast, *BOTection* was able to achieve a high detection accuracy whilst having a low FPR generated from only a few number of hosts. AlAhmadi et

Table 6: Comparison of previous Bot Detection Approaches and proposed system BOTection

Approach	Type of Detector	Encryption Resiliency	Single Bot Detection	Unseen Bots
BotSniffer [15]	C&C	N	N	N
BotFinder [33]	C&C	Y	Y	N
DISCLOSURE [8]	C&C	Y	Y	N
BotHunter [13]	Operation	N	N	N
BotMiner [14]	Operation	Y	N	Y
Abaid et. al [1]	C&C	N	N	Y
Abaid et. al [2]	Operation	N	Y	N
BOTection	**C&C Operation**	Y	Y	Y

al. [4] used the CTU-13 dataset for malware family classification. In comparison, *BOTection* uses a light-weight feature with a higher classification accuracy than 96%.

7 CONCLUSION AND FUTURE WORK

We present *BOTection*, a novel topology and protocol-independent system capable of detecting and classifying flows to a bot family with a 99% accuracy. Importantly, *BOTection* models capture similar network behavior, thus are capable of detecting unseen bot network behavior with 93% accuracy without re-training the classifier. We evaluated our approach using the network communication of various bots performing propagation, C&C communication and attacks. Our results demonstrate the effectiveness of using Markov Chains to understand bot network behavior for better detection. Previous work [1, 6, 13] on botnet detection focused on detecting bot activities by modeling the bot activities using bot life-cycle, mostly using the life-cycle proposed in [13]. In the future, we plan on using the *BOTection* Markov Chain models of the various bot attack stages to evaluate whether the previously proposed bot life-cycle is still valid nowadays.

8 ACKNOWLEDGMENTS

Bushra A. AlAhmadi is supported by the Ministry of Higher Education in the Kingdom of Saudi Arabia and King Saud University. This work was supported by a grant from Mastercard.

REFERENCES

[1] Zainab Abaid, Mohamed Ali Kaafar, and Sanjay Jha. 2017. Early Detection of In-the-Wild Botnet Attacks by Exploiting Network Communication Uniformity: An Empirical Study. *IFIP Networking* (2017).

[2] Zainab Abaid, Dilip Sarkar, Mohamed Ali Kaafar, and Sanjay Jha. 2016. The Early Bird Gets the Botnet: A Markov Chain Based Early Warning System for Botnet Attacks. In *Proc. of IEEE LCN.*

[3] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. 2006. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *Proc. of ACM IMC.*

[4] Bushra A AlAhmadi and Ivan Martinovic. 2018. MalClassifier: Malware family classification using network flow sequence behaviour. In *Proc. of IEEE eCrime.*

[5] M Antonakakis, C Elisan, D Dagon, G Ollmann, and E Wu. 2010. The Command Structure of the Operation Aurora Botnet: History, Patterns, and Findings. *Atlanta, GA: Damballa, Inc* (2010).

[6] Ayesha Binte Ashfaq, Zainab Abaid, Maliha Ismail, Muhammad Umar Aslam, Affan A Syed, and Syed Ali Khayam. 2016. Diagnosing bot infections using Bayesian inference. *Journal of Computer Virology and Hacking Techniques* (2016).

[7] Elaheh Biglar Beigi, Hossein Hadian Jazi, Natalia Stakhanova, and Ali A Ghorbani. 2014. Towards effective feature selection in machine learning-based botnet detection approaches. In *Proc. of IEEE CNS.*

[8] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. 2012. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proc. of ACM ACSAC.*

[9] Hamad Binsalleeh, Thomas Ormerod, Amine Boukhtouta, Prosenjit Sinha, Amr Youssef, Mourad Debbabi, and Lingyu Wang. 2010. On the analysis of the Zeus botnet crimeware toolkit. In *Proc. of ACM PST.*

[10] Z Berkay Celik, Robert J Walls, Patrick McDaniel, and Ananthram Swami. 2015. Malware traffic detection using tamper resistant features. In *MILCOM 2015-2015 IEEE Military Communications Conference*. IEEE, 330–335.

[11] Paul Dokas, Levent Ertoz, Vipin Kumar, Aleksandar Lazarevic, Jaideep Srivastava, and Pang-Ning Tan. 2002. Data mining for network intrusion detection. In *Proc. of NSF NGDM.*

[12] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. 2014. An empirical comparison of botnet detection methods. *Computers & Security* 45 (2014), 100–123.

[13] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. 2007. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation.. In *Proc. of USENIX Security.*

[14] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. 2008. BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection.. In *Proc. of USENIX Security.*

[15] Guofei Gu, Junjie Zhang, and Wenke Lee. 2008. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proc. of USENIX Security.*

[16] Sheharbano Khattak, Naurin Rasheed Ramay, Kamran Riaz Khan, Affan A Syed, and Syed Ali Khayam. 2014. A taxonomy of botnet behavior, detection, and defense. *IEEE COMST* 16, 2 (2014), 898–924.

[17] Daniel Lowd and Christopher Meek. 2005. Adversarial learning. In *Proc. of ACM SIGKDD.*

[18] Enrico Mariconti, Jeremiah Onaolapo, Gordon Ross, and Gianluca Stringhini. 2016. What's your major threat? On the differences between the network behavior of targeted and commodity malware. In *Proc. of IEEE ARES.*

[19] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. 2017. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *Proc. of NDSS.*

[20] David McGrew and Blake Anderson. 2016. Enhanced telemetry for encrypted threat analytics. In *Proc. of IEEE ICNP*. 1–6.

[21] Aziz Mohaisen, Andrew G West, Allison Mankin, and Omar Alrawi. 2014. Chatter: Classifying malware families using system event ordering. In *Proc. of IEEE CNS.*

[22] J Nazario. 2008. Political DDoS: Estonia and beyond. In *Proc. of USENIX security.*

[23] Jon Oltsik. 2015. *SOC-as-a-service for Midmarket and Small Enterprise Organizations.* Technical Report. The Enterprise Strategy Group.

[24] Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. 2019. MaMaDroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM TOPS* 22, 2 (2019), 14.

[25] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *Proc. of USENIX Security.*

[26] Roberto Perdisci, Wenke Lee, and Nick Feamster. 2010. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces.. In *Proc. of USENIX NSDI.*

[27] Michal Piskozub, Riccardo Spolaor, and Ivan Martinovic. 2019. MalAlert: Detecting Malware in Large-Scale Network Traffic Using Statistical Features. *ACM SIGMETRICS Performance Evaluation Review* 46, 3 (2019), 151–154.

[28] M Zubair Rafique and Juan Caballero. 2013. Firma: Malware clustering and network signature generation with mixed network behaviors. In *Proc. of RAID.*

[29] Elias Raftopoulos and Xenofontas Dimitropoulos. 2011. Detecting, validating and characterizing computer infections in the wild. In *ACM IMC.*

[30] Christian Rossow and Christian J Dietrich. 2013. Provex: Detecting botnets with encrypted command and control channels. In *DIMVA*. 21–40.

[31] Elizabeth Stinson and John C Mitchell. 2008. Towards Systematic Evaluation of the Evadability of Bot/Botnet Detection Methods. *Proc. of USENIX WOOT* (2008).

[32] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2018. Robust smartphone app identification via encrypted network traffic analysis. *IEEE TIFS* 13, 1 (2018), 63–78.

[33] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. 2012. BotFinder: Finding Bots in Network Traffic Without Deep Packet Inspection. In *ACM CoNEXT.*

[34] Gernot Vormayr, Tanja Zseby, and Joachim Fabini. 2017. Botnet Communication Patterns. *IEEE COMST* 19, 4 (2017), 2768–2796.

[35] Sebastian Zander, Grenville Armitage, and Philip Branch. 2007. A survey of covert channels and countermeasures in computer network protocols. *IEEE COMST* 9, 3 (2007), 44–57.

[36] David Zhao, Issa Traore, Bassam Sayed, Wei Lu, Sherif Saad, Ali Ghorbani, and Dan Garant. 2013. Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security* 39 (2013), 2–16.

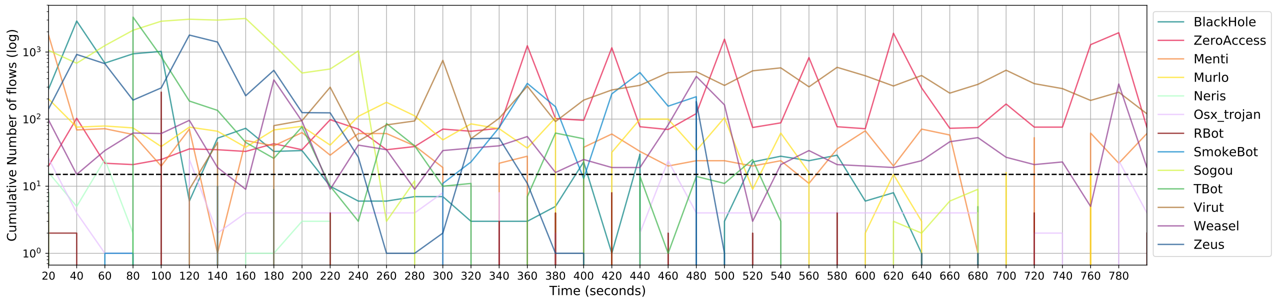


Figure 12: Number of flows generated by each malware family every 20 seconds.

A APPENDIX

A.1 Background on Bot Network Communications

Network communication is essential for botnets to carry out their attacks, as a bot receives commands from the C&C server, propagates, and executes the received commands. We discuss bot network communication during different stages, which serves as a baseline of what bot detectors aim to detect.

Rallying. As a first step, rallying [16] process consists of a new bot that notifies its presence to the C&C server. To do so, a bot could contact a domain hard-coded in its binary (which is vulnerable to blacklists), or use a *Domain Generation Algorithm* (DGA) (e.g. *Zeus, Bunitu, Sality*). A bot may also use fast-fluxing (e.g. *Bunitu*) to add resiliency to the domain generation process.

Propagation. Botnets aim to recruit new bots by using active (e.g. scanning) or passive (e.g. phishing emails) propagation. In active propagation, the bot exhibits a worm-like behavior infecting new victims via known vulnerabilities (e.g. *Duqu 2.0*) [16]. Some bots scan the network before the infection (e.g. *Conficker, WannaCry, Sality*) by identifying reachable hosts via ICMP echo requests or by port scanning to detect vulnerable services (e.g. *Neris, Rbot, Virut, Donbot, Qvod*) [34]. In passive propagation, bots propagate through less noisy means such as infected removable media, phishing emails, and drive-by download (e.g. *Zeus*). Although network activities due to passive approaches can be detected, correlating these activities with the actual bot infection is difficult since propagation and infection could occur at different times [34].

Operation. A bot could be used to carry out various types of attacks that have a different impact in terms of network communications [34]. For example, bots (e.g. *Rbot, Stlrat, Papras*) can launch Distributed Denial of Service attacks (DDoS) by sending huge amounts of requests (*usually TCP/UDP HTTP requests*), overloading systems, and making a network resource unavailable. Bots (e.g. *Geodo, Neris, Sality, Stlrat*) can also send unsolicited emails (i.e. spam) via Simple Mail Transport Protocol (SMTP). *ClickFraud* is an attack that uses bots (e.g. *Miuref, Neris*) to access pay-per-click ads via HTTP/S requests, thus defrauding advertisers. Bots can also collect network information (e.g. *Duqu 2.0*, *Phatbot*), log-in information harvesting (e.g. *Storm*), self-updating (e.g. *Zeus*, *Phatbot*), bitcoin mining (e.g. *Miner*), receiving C&C instructions (e.g. *Stuxnet*), or network traffic modification (e.g. *Miner*).

A.2 Bot traffic over time

We show the overall amount of traffic by each bot family in Table 2 over time in Figure 12. In particular, the y-axis has a logarithmic scale and each data-point consists of the traffic in an interval of 20 seconds. We can notice that the majority of the bot families constantly produce a significant amount of network flows. This means that a bot is more likely to be detected in different phases of its malicious operations since our method has available an abundant quantity of *n*-flows for the classification.

A.3 Bots' Network Communications Dataset

In Table 7, we show the bot families attack network communications in the dataset used to train the multi-class classifier.

Table 7: Bot Network communication used to train the multi-class classifier*(PS: Port Scanning, NS: Network Scanning, FF: Fast-Fluxing, CF: ClickFraud)*.

Family	**Propagation**		**C&C**			**Operational**		
	PS	*NS*	*DGA*	*FF*	*CC*	*DDoS*	*Spam*	*CF*
Miuref								✓
Zeus			✓					
Geodo							✓	
Bunitu			✓	✓	✓			
WannaCry		✓						
Conficker		✓			✓			
Zeus C&C			✓		✓			
Neris	✓						✓	✓
Rbot	✓					✓		
Virut	✓						✓	
Sality		✓	✓				✓	
Stlrat						✓		
Donbot	✓							
Papras			✓			✓		
Qvod	✓							