Machine Learning for Cybersecurity: Network-based Botnet Detection Using Time-Limited Flows

Author: Stephanie Ding Mentor: Julian Bunn
California Institute of Technology
Email: sding@caltech.edu

September 26, 2017

Abstract

Botnets are collections of connected, malware-infected hosts that can be controlled by a remote attacker, and are one of the most prominent threats in cybersecurity as they can be used for a wide variety of purposes, such as denial-of-service attacks, spam or bitcoin mining. We propose a two-stage detection method, using supervised and unsupervised machine learning techniques to distinguish between botnet and non-botnet network traffic. In the first stage, we examine network flow records generated over limited time intervals, which provide a concise, but partial summary of the complete network traffic profile, and classify flows as malicious or benign based on a set of extracted statistical features. In the second stage, we perform clustering on internal hosts involved in previously identified malicious communications to determine which hosts are most likely to be botnet-infected. Using existing datasets, we demonstrate the feasibility of our method and implement a proof-of-concept, real-time detection system that aggregates the results of multiple classifiers to identify infected hosts.

1 Introduction

In the twenty-first century, networked devices have become an integral part of any business or organization, supporting an extensive number of applications and services such as access to the World Wide Web, multimedia sharing, file storage, or instant messaging and email. The growing number and complexity of networked devices means that they are increasingly being targeted by cybercriminals, who exploit vulnerable devices for malicious purposes – particularly due to the advent of the Internet of Things (IoT), which represent new potential attack vectors as modern malware is now capable of taking over devices such as surveillance cameras and 'smart' household devices with built-in network capabilities. One of the common uses for compromised devices is to integrate them into a botnet – a collection of connected hosts ("bots"

or "zombies") infected with malware that allows them to be controlled by a remote host (the "botmaster"). Botnets are powerful assets for attackers due to their versatility as they can be employed for a wide variety of purposes, such as Distributed Denial-of-Services (DDoS) attacks, email phishing and spam, and, with the advent of cryptocurrencies in recent years, distributed bitcoin mining, making them one of the most prominent threats in the cybersecurity field.

Presently, botnets are responsible for the majority of online email spam, identity theft, phishing, fraud, ransomware and denial-of-service attacks, and are substantial sources of damage and financial loss for organizations and business. Taking action to neutralize the potential impact and harmful behaviors of botnets have become essential steps in almost all malware mitigation strategies, resulting in the need for rapid and effective identification of botnet infections.

1.1 Background and related work

Historically, botnet detection was achieved through setting up honeypots or honeynets – security mechanisms that appear to contain data and are a legitimate part of some network, but are in fact isolated systems designed to detect and/or counteract attempts at intrusion into the network – and developing specific signatures for various types of botnets in order to defend against future attacks of the same type. These signatures are then used for payload analysis techniques such as deep packet inspection (DPI), which requires individually inspecting every packet transmitted on the network and matching for malicious packet signatures. While payload inspection techniques usually achieve a high level of identification accuracy, they are extremely limited for several reasons:

1. *Cost and speed.* The development of large-scale honeypots is a significant time and economic investment, and multiple honeypots may be required to capture a variety of botnet signatures. Furthermore, inspecting every packet sent through the network is a time and resource-intensive process due to the sheer volume of packets that must be processed. This makes packet inspection techniques unsuitable for real-time detection as analysis is usually performed after an attack has occurred, making early prevention and mitigation of attacks difficult.
2. *Privacy concerns.* Detailed analysis at the packet level often exposes private information sent by network users, and is an undesirable violation of user privacy.
3. *Vulnerable to obfuscation and zero-days.* Signature-based detection methods are not versatile and rely on matching for specific existing patterns that are known. This means they cannot be generalized to new and emerging types of botnet attacks, and can be bypassed by bots that utilize encryption or obfuscation to conceal their communications.

Network-centric, traffic analysis-based schemes have become increasingly popular as an alternative to signature-based detection schemes as they do not suffer from the same limitations of payload inspection techniques. These methods often focus on examining network flows, which is conventionally defined as a sequence of packets over some period of time, grouped by source IP, source port, destination IP, destination port, and protocol – essentially a summary of a

communication channel between two hosts. The underlying assumption of flow-based network analysis is that botnet traffic is distinguishable from regular network traffic in some manner, which can be determined through statistical features irrespective of individual packet contents. This makes a flow-based approach less susceptible to encryption or obfuscation techniques, as well as vastly reducing the amount of data that needs to be processed. Furthermore, many types of bots may exhibit similar patterns of behavior despite having different signatures, making a flow-based approach more generalizable.

In recent years, there has also been an increase in the amount of literature on the topic of applying machine learning for automated botnet detection using network flows. Strayer et al. [1] was one of the first to demonstrate the use of supervised machine learning to identify IRC botnets, successfully classifying TCP flows with low ($< 3\%$) false positive and negative rates although it models TCP as the primary communication channel of botnet traffic. Masud et al. [2] similarly was able to detect botnet traffic using machine learning using a flow-based approach and performing classification on host-level forensic and deep packet inspection in order to differentiate between benign and botnet traffic. Saad et al. [3] proposed a new approach to identify traffic that comprises the C&C stage of the botnet life cycle and applied machine learning to this subset of network traffic in order to detect P2P botnets, identifying both host-based and flow-based traffic features. Camelo et al. [4] presented a new method of identifying botnet activity by appropriating features from several data feeds such as DNS domain responses, live communication directed to C&C servers, and performing machine learning on a graph representation of the data, allowing them to identify botnets as singly-connected components of known malicious servers (domains) and infected machines (IPs) to a reasonable degree of accuracy. The success of these methods confirms that botnet traffic exhibits certain characteristics and communication patterns that can be exploited using classification techniques.

2 Approach

Many existing flow-based detection approaches analyze flows between two hosts in its entirety, which is not always feasible as a flow may span several hours to several days. We propose a detection approach that examines flows generated over short time windows (300 seconds), which is short enough such that analysis and detection can occur in relatively close to real time, while still being long enough to capture interesting characteristics of network traffic. To avoid treating the same flow split across multiple time windows as completely discrete entities (and thus losing the temporal characteristics of the flow data), we generate the windows such that each window contains 150 seconds of overlap with the previous window.

For each window, we extract a set of flow features from each flow and then apply a two-stage machine learning process to determine infected network hosts. In stage 1, we utilize supervised learning by training a classifier on existing datasets to perform binary classification and determine which flows are likely to be botnet flows. In stage 2, we apply unsupervised learning and cluster the hosts involved in the identified botnet flows into two clusters, benign

and anomalous, based on a separate set of host-based features. We detect whether a host is infected or not by examining a sequence of windows and analyzing the evolution of the host's cluster membership over time. Finally, to demonstrate the potential real-world applications of our detection method, we build a proof-of-concept application designed for use in a network monitoring situation and illustrate how our detection scheme can be applied for real-time detection.

3 Method

3.1 System overview

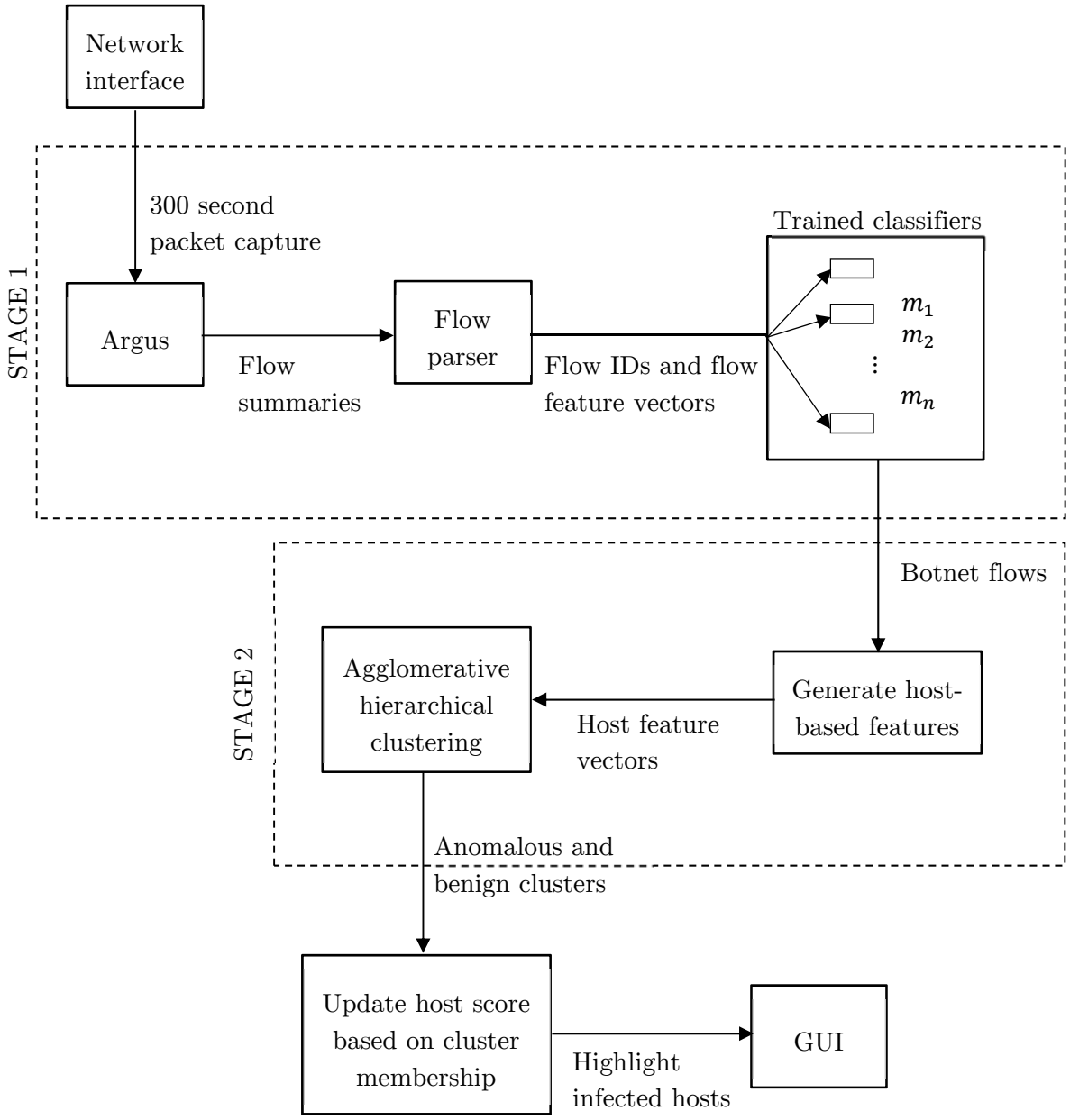


Figure 3.1.1. Overview of proposed method in a detection system.

3.2 Datasets

The dataset used is the CTU-13 dataset [5] which is a publicly available, labelled dataset developed by researchers at the Czech Technical University containing thirteen separate scenarios of mixed botnet, background and normal traffic. In each scenario, the researchers captured network traffic of the entire university network at an edge router for a period of time, during which a botnet infection was simulated on one or several networked virtual machines by running a specific type of malware. Each scenario contains a packet capture (.pcap) of the botnet traffic only, and a truncated .pcap of the full packet capture which included the complete headers of each packet but removed packet payloads to protect the anonymity of network users. The full packet capture was also processed using the utility Argus (the Audit Record Generation and Utilization System) [6] to generate bidirectional network flow summaries (in .binetflow format), with a limited number of features output for each flow. Each .binetflow file was labelled by the researchers to identify particular communications as either a botnet flow, a normal flow, or a background flow.

We examined seven of the thirteen scenarios – scenarios 5, 6, 7, 9, 11, 12 and 13 – with a particular emphasis on scenario 9 as it was one of the larger captures and contained the most infected hosts. Although we also applied the detection scheme to scenario 10, the results were not included in this report due to the capture being incomplete – the authors reported that the truncated .pcap in that scenario had been unexpectedly terminated early due to a lack of disk space. [7] The other unused scenarios featured a repeat of malware in the selected scenarios, but were less desirable due to their length or the number of infected hosts. Nevertheless, the selected scenarios cover a diverse range of botnets that vary in the number of infected network hosts, protocols, and malicious actions, and are representative of a large majority of behaviors found in modern botnets. Figure 3.2.1 describes each scenario in further detail:

Scenario	Botnet name	Infected hosts	Capture duration (hrs)	Protocol	Behaviors and characteristics
5	Virut	1	11.63	HTTP	Spam, port scan, web proxy scanner.
6	Menti	1	2.18	HTTP	Port scan, proprietary C&C, RDP.
7	Sogou	1	0.38	HTTP	Connects to Chinese hosts.
9	Neris	10	5.18	IRC	Spam, click fraud, port scanning. Bitcoin miner.
11	Rbot	3	0.26	IRC	ICMP DDoS.
12	NSIS.ay	3	1.21	P2P	Synchronization.
13	Virut	1	16.36	HTTP	Spam, port scan, captcha and web mail.

Figure 3.2.1. Details of selected scenarios in the CTU-13 dataset.

3.3 Stage 1: Supervised learning (binary classification)

3.3.1 Flow extraction and feature selection

A feature is a characteristic of a flow over a period of time, which may either be extracted directly from packet headers (for example, source and destination IP) or calculated from the packet captures (such as packet interarrival time). A variety of flow exporter utilities exist that have the capability to process packet capture files and extract similar flow features. During early stages of the project, we experimented using a variety of flow exporters such as NETMATE [8], Tranalyzer2 [9], and CICFlowMeter [10].We ultimately decided to utilize Argus for flow generation as it was the tool used by the authors of the CTU-13 dataset, and the varying behaviors of flow exporters often resulted in other tools generating flows that differed from the network flow file provided in the scenarios.

Argus is capable of generating bidirectional network flow data generator with detailed statistics about each flow, including reachability, load, connectivity, duration, rate, jitter and other metrics. 40 features were selected to describe each flow, based on domain knowledge and some assumptions about the behavior of botnets. (See Appendix A for the full list of features extracted with Argus, including descriptions.)

The features that comprise the standard 5-tuple (saddr, sport, daddr, dport, proto) were used to define the flow ID for each flow. Almost all features are numeric in nature, with the exception of two categorical values direction and state, which were mapped to discrete integer values as shown in figures 3.3.2 and 3.3.3.

For a specified packet capture, Argus outputs a tab-delimited text file where each line contains the selected features for a particular flow, facilitating an easy process of parsing the flows into array format for further processing. Features in the flow ID were not included in the feature vectors used to train the classifiers, as we wish to select a set of network-agnostic, universal features that can be applied to traffic from any network, and different networks may use different IP and port numbers. Thus, each flow is identified by its flow ID and represented numerically by a feature vector of the remaining 40 features, which was then used to train the classifiers (See Appendix B for the relative importances of each feature, and section 4.2.4 for a comparison of classifier performance when trained on a reduced feature space).

3.3.4 Training models on limited time intervals

For each scenario, a training dataset was generated with a ratio of 1:10 botnet to non-botnet flows. This ratio was chosen due to the highly imbalanced nature of the captures, which contained significantly more background and normal flows (around 90% to 97% of the entire dataset) compared to botnet flows (around 0.15% to 8.11%). We found the ratio of 1:10 to be sufficiently similar to the real datasets while containing adequate samples of botnet flows to obtain good classifier accuracy (see section 4.2.3 for a brief comparison on training set balance and classifier accuracy).

Direction	Value
->	0
?>	1
<-	2
<?	3
<->	4
<?>	5

Figure 3.3.2. Numerical mapping for categorical feature 'direction'.

State	Value
STA	0
RST	1
CON	2
FIN	3
INT	4
ECO	5
URHPRO	6
URP	7
RED	8
REQ	9
URN	10
URH	11
ACC	12
RSP	13
ECR	14
TXD	15
NNS	16
URFIL	17
NRS	18
CLO	19
URF	20
URO	21
SRC	22
DCE	23
URNPRO	24

Figure 3.3.3. Numerical mapping for categorical feature 'state'.

As the captures vary in length, some contain significantly more flows than others. On shorter captures, a training dataset with 1,000 botnet flow samples and 10,000 non-botnet flow samples was generated, while on longer datasets a training dataset containing 10,000 botnet flow samples and 100,000 botnet flow samples was generated. These datasets were produced using the following procedure:

1. Using Wireshark [11], split the truncated packet capture of the complete traffic into windowed, sequential .pcap files of 300 seconds, with 150 seconds overlap between each window.

2. Using Argus, generate .binetflows for each of the windowed .pcaps, extracting the 40 features as listed in section 3.3.2.
3. Using the labelled .binetflow included in each scenario, build an initial profile of the network traffic by parsing the flows within the file. The traffic profile consists of three unique sets of flow IDs – normal, background and botnet – that are used for further processing.
4. For each window, parse the associated .binetflow file to obtain an array of flow IDs and a corresponding array of flow feature vectors.
5. Randomly sample an equal number of non-botnet flows from every window using reservoir sampling. A flow is defined as a non-botnet flow if its flow ID is present in either the normal or the background flow ID set in the previously generated traffic profile. Assign these flows a ground truth label of 0.
6. Similarly, randomly sample an equal number of botnet flows from windows when the botnet is active, using reservoir sampling. A flow is defined as a botnet flow if its flow ID is present in the botnet flow ID set in the previously generated traffic profile. Assign these flows a ground truth label of 1.
7. Concatenate the botnet and non-botnet flow feature vectors into a single array and convert all NaN, invalid or empty values in vectors to 0 to produce the final array of training examples (x). Similarly, concatenate the ground truth labels to produce the final array of expected outputs (y).
8. Zip x and y and randomly shuffle the training set, ensuring that each vector or row in the x array continues to correspond to the correct output in the y array.

Using the training dataset, a random forest classifier was trained with 100 estimators to perform binary classification on flow feature vectors generated over 300 second windows, i.e. output 0 if it determines a feature vector to be representative of a non-botnet flow, or output 1 if it determines a feature vector to be representative of a botnet flow. Random forests are an ensemble learning method which operates by constructing a collection of decision trees, each fit on a random subset of features or samples of the entire dataset, and produces an aggregated result. The choice of random forests as the classification algorithm for stage 1 was due to its robustness and high accuracy in initial tests, making it suitable for our intended purpose (see section 4.2.1 for results of initial tests and comparisons with other supervised learning algorithms).

3.4 Stage 2: Unsupervised learning (clustering)

Stage 2 of the detection process applied unsupervised learning on the results of stage 1, and examined host-based characteristics, rather than flow-based characteristics. In each window, the classifier was first used to identify potential botnet flows. An internal host is defined as being involved in botnet communications if it is either the source IP or the destination IP of a flow predicted by the classifier as being a botnet flow. For all hosts involved in botnet communications, seven host-based features were computed:

1. The total number of predicted botnet flows that the host is involved in.

2. The total number of outgoing packets from the host involved in botnet flows.
3. The total number of incoming packets from the host involved in botnet flows.
4. The total number of incoming bytes from the host involved in botnet flows.
5. The total number of outgoing bytes from the host involved in botnet flows.
6. The total number of unique destination ports the host is communicating with.
7. The total number of unique destination IPs the host is communicating with.

These features were standardized such that the range of values for each feature had a mean of 0 and a standard deviation of 1 (using scikit-learn's StandardScaler) and were then used to form a feature vector for each host. Agglomerative hierarchical clustering was applied on the host feature vectors, using the standard Euclidean distance as the distance metric between vectors and utilizing Ward linkage. Agglomerative clustering is a 'bottom-up' method of cluster analysis, in which each data point begins in its own cluster, but are progressively merged into other clusters higher up in the hierarchy. Ward linkage, also known as minimum variance criterion, ensures that at each merging step, the pair of clusters that leads to the minimum increase in total inter-cluster variance is chosen.

We make the heuristic assumption that the majority of hosts on a network will not be infected and only a small minority are infected, which was true for the datasets examined, and label the smaller cluster as the anomalous/botnet cluster and the larger cluster as the benign/non-botnet cluster. For each window, the hosts in the botnet cluster are assumed to be infected during that timeframe of the window; however often a single window is insufficient to determine the presence of botnet activity and examining the temporal nature of a host's cluster membership is a more accurate indicator of botnet activity. (See section 4.3 for a demonstration of our detection scheme applied in a network monitoring scenario.)

4 Evaluation

4.1 Results

4.1.1. Stage 1

Due to the imbalanced nature of the dataset, accuracy is not a good predictor of classifier performance as a high accuracy can be achieved simply by classifying the majority of flows as non-botnet flows due to there being significantly higher numbers of non-botnet flows compared to botnet flows. Thus, in stage 1, the performance of the binary classifier was evaluated with metrics such as the true positive rate and true negative rate instead. A true positive is defined as a flow vector with a botnet flow ID, during a time window when the botnet is running, that is classified as a botnet flow. Similarly, a true negative is a botnet flow vector with a non-botnet flow ID that is correctly classified as a non-botnet flow.

A separate classifier was trained for each scenario examined, and the performance of the classifier was evaluated on the specific dataset it was trained on. The following table describes the number of flow vectors extracted from each dataset:

Scenario	Unique botnet flow IDs	Unique normal flow IDs	Unique background flow IDs	Number of windows	Average number of flows per window	Total botnet flow vectors	Total non-botnet flow vectors
5	856	4631	113667	13	32310	2326	417704
6	4621	7238	394056	52	48911	10960	2532403
7	44	1666	103950	9	47119	308	423766
9	93438	27749	1100291	125	67714	435076	8029131
11	8155	613	91436	7	45301	8855	308249
12	1972	7448	265712	30	40159	9234	1195532
13	32866	27183	943512	393	18136	98997	7028311

Figure 4.1.1. Statistics about flow vectors extracted from each scenario.

True positive rate was calculated as $TPR = \frac{TP}{TP+FN}$ and true negative rate was calculated as $TNR = \frac{TN}{TN+FP}$. The total TPR/TNR was calculated on the sum total of the true positives and true negatives across all windows in a dataset, while for the average TPR/TNR reflects the average of all individual TPR/TNR values calculated on each window separately. Note that in many scenarios the botnet is not active for the first few windows; thus, the true positive rate is only calculated during time windows when the botnet is running.

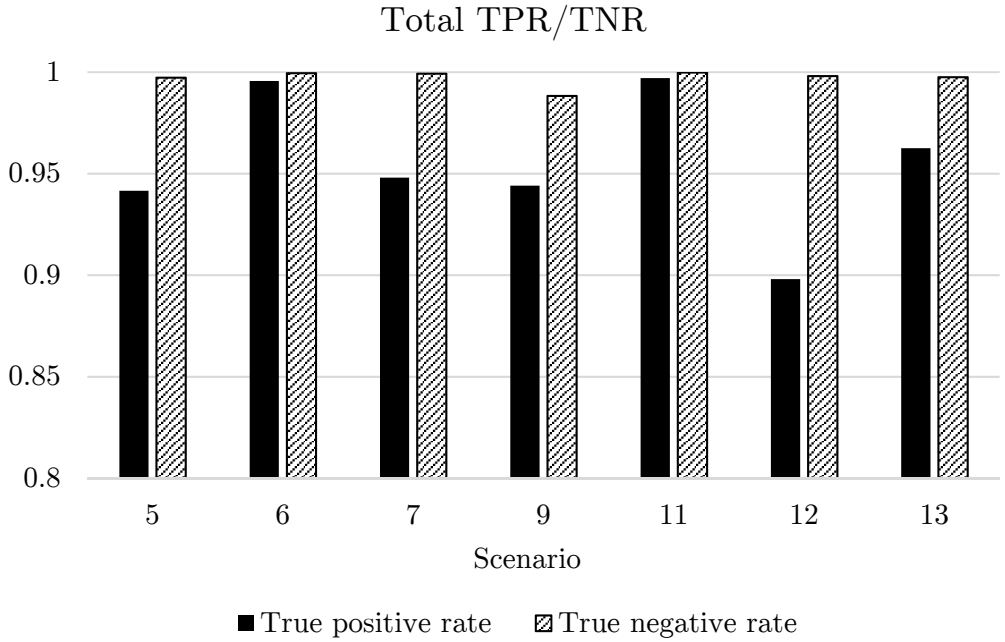


Figure 4.1.2. Total TPR/TNR across all scenarios examined.

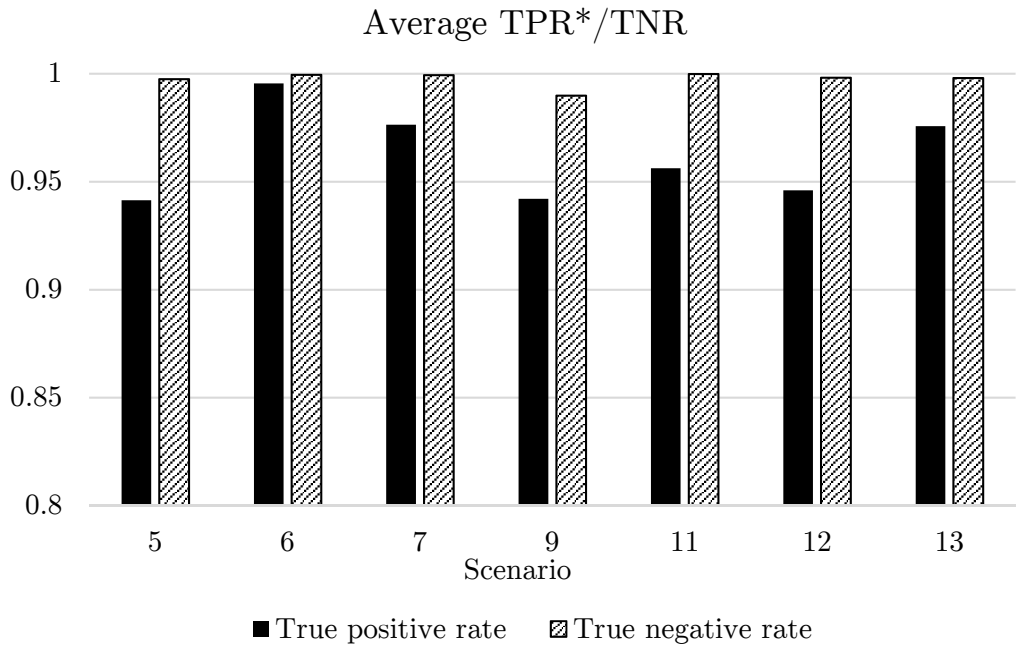


Figure 4.1.3. Average TPR*/TNR across all scenarios examined
***calculated only during windows when the botnet is running

4.1.2 Stage 2

The performance evaluation of stage 2 reflects the number of local IP addresses that are correctly clustered – a true positive in this context refers to a botnet host that is clustered into the anomalous cluster during any window where the botnet is running, while a true negative is a non-infected host clustered into the benign cluster. A false positive is any host clustered into the anomalous cluster during a window when the botnet is not running, or a non-infected host clustered into the anomalous cluster during windows when the botnet is running. Similarly, a false negative is a botnet host clustered into the benign cluster during a window when the botnet is running. A false negative is a botnet host that was not present in the anomalous cluster during a window when the botnet is running – either as a result of being clustered incorrectly into the majority cluster, or its flows not being identified by the classifier as botnet flows (and thus it would be absent from both clusters as it was not identified as a host involved in botnet communications).

Due to the temporal nature of the windowed captures, Figure 4.1.4 depicts graphs of the true positive rate, false positive rate and false negative rate across windows for each dataset over all time windows in each scenario. Note that some scenarios (5, 6, 7, 13) only have a single infected host, so if multiple false positives occur this is shown as a false positive rate of >1.

Figure 4.1.5 also lists the total true positives, true negatives, false positives and false negatives for each scenario, which is a sum of the respective values across all time windows. The total TPR/FPR is also calculated for each scenario and summarizes the results:

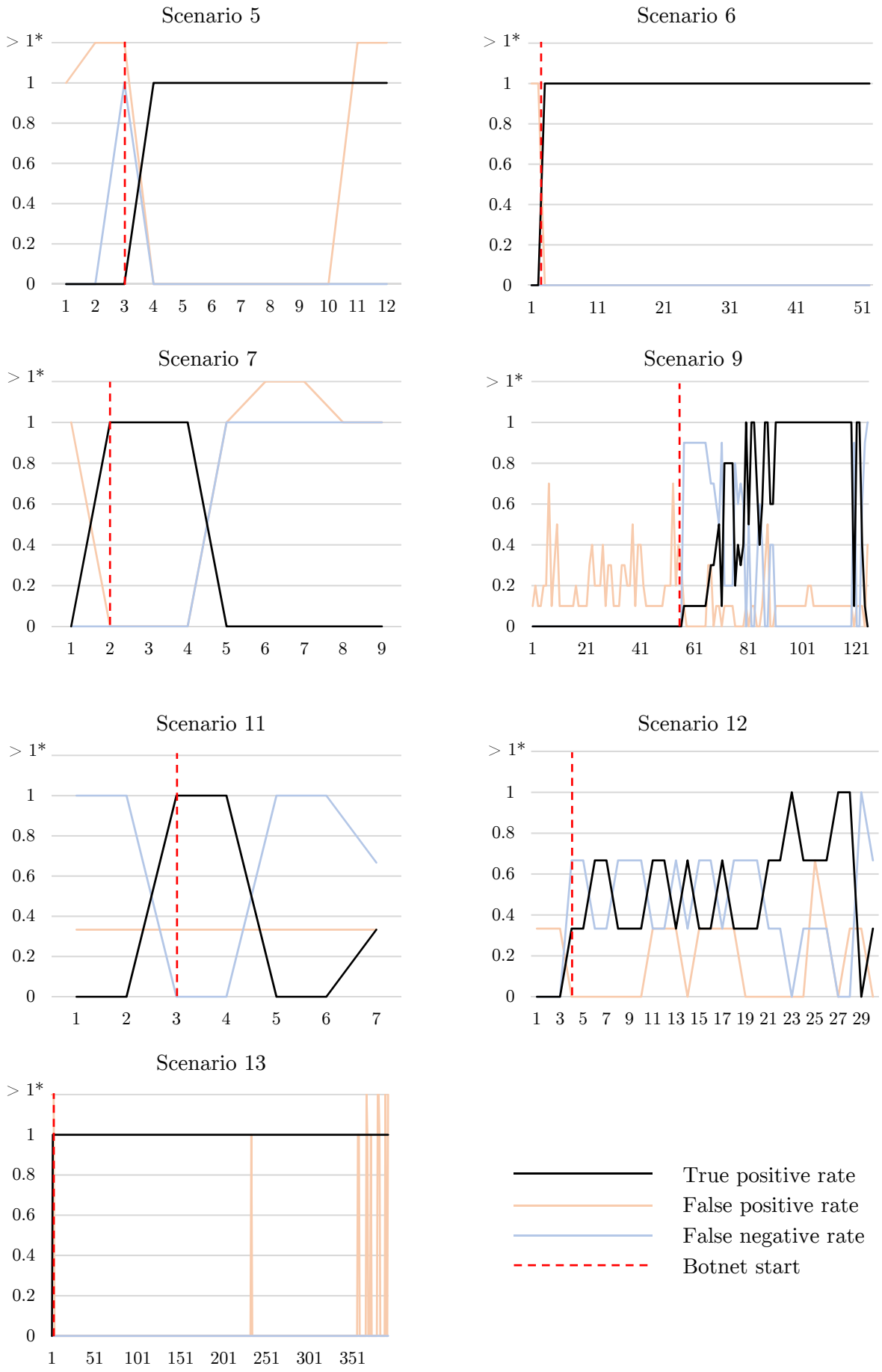


Figure 4.1.4. Graphs of stage 2 TPR/FPR/FNR over time windows.

Scenario	Total IPs	TP	FP	FN	TN	TPR	TNR
5	252	9	11	1	231	90.00%	95.45%
6	513	50	4	0	459	100.00%	99.14%
7	173	3	8	2	160	60.00%	95.24%
9	5549	427	151	161	4810	72.62%	96.96%
11	59	11	7	3	38	78.57%	84.44%
12	643	44	12	31	556	58.67%	97.89%
12	3471	392	50	0	3029	100.00%	98.38%

Figure 4.1.5. Summary of total confusion matrix results for each scenario.

4.2 Discussion

The classifier performance in stage 1 demonstrates that the selected 40 features are capable of distinguishing between botnet and non-botnet flows to a high degree of accuracy ($> 89\%$), even if the flows are only generated over fixed-time windows and provide a limited perspective of the entire network traffic. The true negative rate is consistently high, but this is likely a reflection of the nature of the training set, in which there are significantly more non-botnet flows.

The classifier for scenario 12 had a significantly lower total TPR when compared to other scenarios. This may have been due to scenario 12 featuring a P2P botnet, which, unlike a traditional botnet with a single command & control server, communicates with a list of trusted servers (including other infected machines) in a decentralized manner. The traffic is likely to be less structured and have greater variance, appearing more similar to background traffic and thus making the classification task more difficult. In contrast, scenarios 6 and 11, which are a traditional C&C botnet and an ICMP DDoS botnet respectively, are much more likely to have botnet traffic that is more structured and distinguishable from regular network traffic, and thus the classifier trained for these scenarios have better performance.

In stage 2, generally, true infected hosts will be consistently clustered into the anomalous cluster after the botnet begins running, while non-infected hosts (false positives) are not consistently identified as such across consecutive windows. The detection scheme was highly effective on scenarios 5, 6 and 13, which each contained one infected host, and maintained a TPR of 1.0 throughout the duration of the botnet's execution. Scenarios 7 and 11 also featured a TPR rate of 1.0 when the bot begins running, but the detection rate drops over subsequent windows. The performance of the detector may have been affected by the relatively short duration of these captures. Scenario 9 features a more realistic example of botnet traffic on a network, as it contains ten infected hosts. After the botnet begins running, a gradual increase in the detection rate is observed, reaching 1.0 during later stages of the botnet's execution.

Scenario 12 contained 3 infected hosts. The TPR for scenario 12 fluctuated between 0.33 and 0.66 during the majority of the botnet's execution, indicating that 1 or 2 of the 3 infected hosts were consistently detected. This may have been due to the lower accuracy of the scenario 12 classifier in stage 1, which impacts the number of flows identified as botnet flows. It is likely that some botnet flows were incorrectly classified by the classifier as benign, resulting in a host not being identified as involved in botnet communications.

4.2.1 Comparison of supervised learning algorithms

We initially tested a variety of supervised learning algorithms (Naïve Bayes, Support Vector Machines, Decision Trees, Random Forests) to explore the differences in classifier performance and to confirm our hypothesis that the selected 40 features facilitated distinction between botnet and non-botnet flows. For each dataset, these tests were performed on flows generated over the entire duration of the capture rather than over 300 second windows, using a random 30% of the entire dataset for training and the remaining 70% for testing. Figure 4.2.1 lists the number of total flows, botnet flows, and the size of the training and testing datasets for each scenario used in the test:

Scenario	Total flows	Botnet flows	Training flows	Testing flows
8	71298	1666	23189	49909
9	916824	152804	275049	641777
11	476	2856	856	2000
12	78696	13116	23608	55088
13	400902	66817	120270	280632

Figure 4.2.1. Information about the datasets used in comparing classifier performance.

Figure 4.2.2 shows the performance of various supervised learning algorithms applied to the same training datasets and evaluated on the same testing datasets:

While these tests were not particularly robust, the results obtained indicated that support vector machines (SVMs) were time consuming to train, and thus were excluded from the tests. The results of these tests also demonstrated that Naïve Bayesian classifiers had significantly lower classification accuracy than the other algorithms, and while decision trees produced high classification accuracy, the algorithm was prone to overfitting. Random forests produced similarly high classification accuracy, but are more powerful models than decision trees, being able to limit overfitting without substantially increasing error. This suggested that random forests would be the most effective supervised learning algorithm for stage 1 of our detection process.

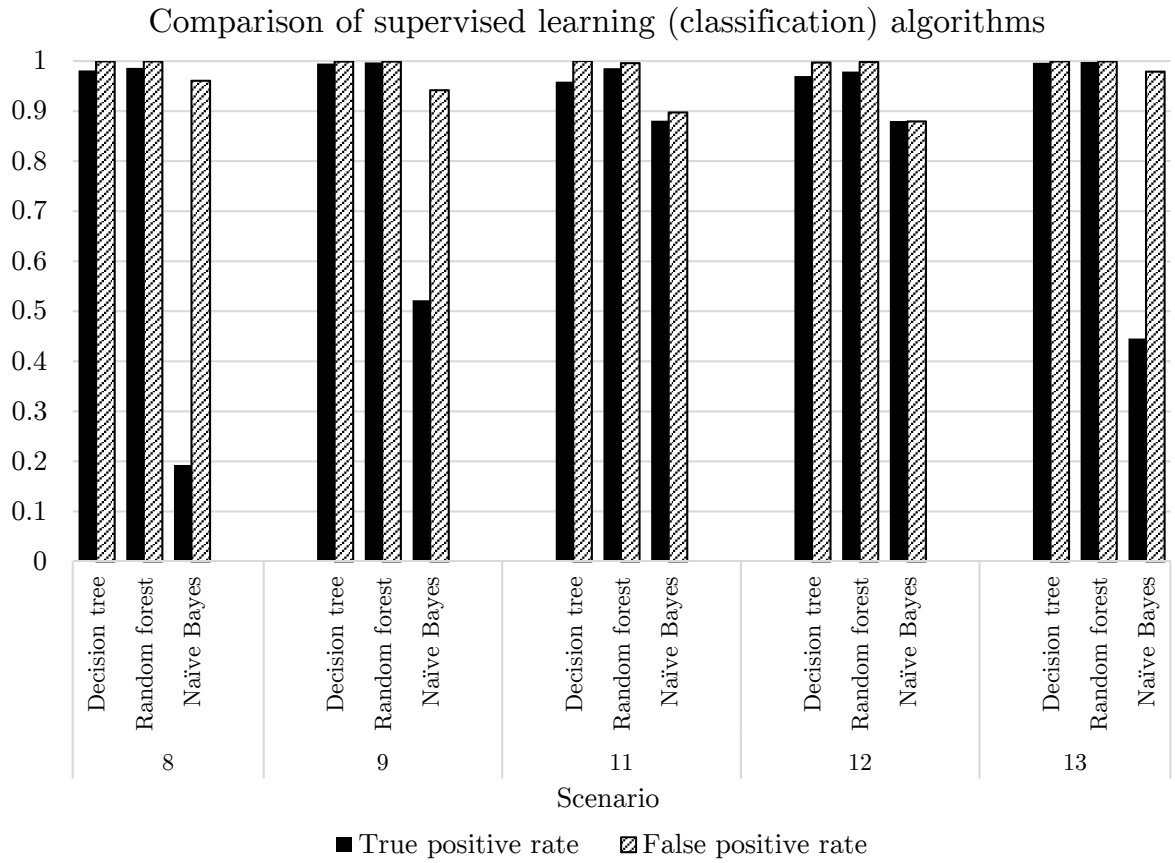


Figure 4.2.2. Performance of various supervised learning algorithms, compared.

4.2.3 Effect of dataset balance on classifier performance

In each of the provided scenarios, there is a significantly higher number of non-botnet flows when compared to botnet flows. Thus, generating a training dataset becomes difficult as the number of botnet flows compared to non-botnet flows included in the training set will influence the performance of the model. During initial testing stages, we trained another random forest classifier on scenario 9, using a completely balanced dataset containing 10,000 botnet flows and 10,000 non-botnet flows.

In the following figures, definitions of average TPR/FPR and total TPR/FPR are consistent with previous definitions in section 4.1.1:

	1:10 training set	1:1 training set
Average TPs	3286	6264
Average FPs	752	2029
Average FNs	195	41
Average TNs	63481	62205
Average TPR	94.21%	99.32%
Average TNR	98.98%	96.77%

Figure 4.2.2. Comparison of the average metrics between the two datasets.

	1:10 training set	1:1 training set
Total TPs	410758	432220
Total FPs	93992	253565
Total FNs	24318	2856
Total TNs	7935139	7775574
Total TPR	94.41%	99.34%
Total TNR	98.83%	96.84%

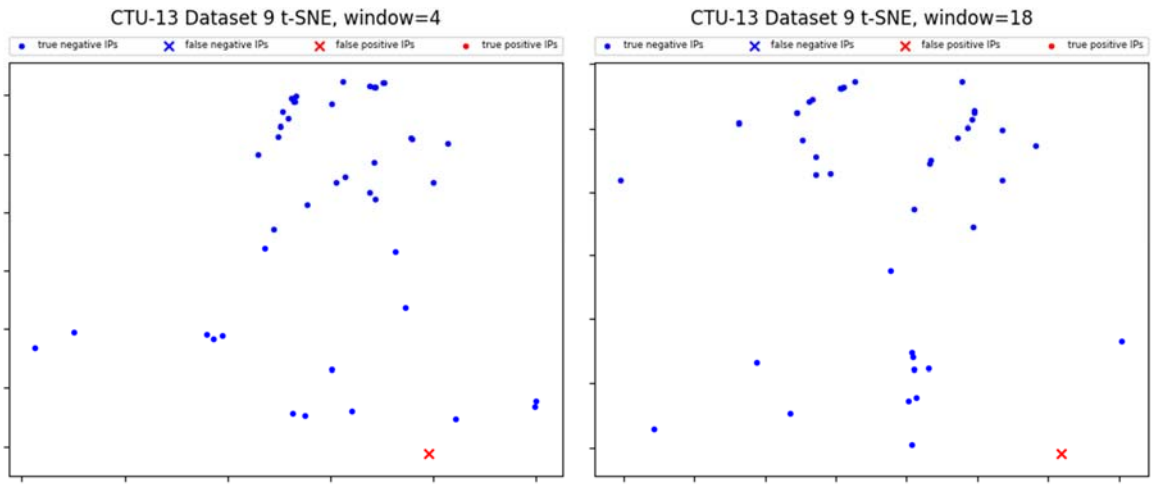
Figure 4.2.3. Comparison of the total metrics between the two datasets.

The model trained on the balanced dataset increased in the number of false positives by over twice the amount of the model trained on the 1:10 dataset, but also reduced false negatives by almost ten times. These results illustrate that it is possible to alter model performance and obtain increased TPR at the cost of more false positives, or vice-versa.

4.2.4 Cluster visualization using t-SNE

t-SNE, or t-distributed stochastic neighbor embedding, is a machine learning algorithm for dimensionality reduction, embedding data points in a higher dimensional space into two or three dimensions for visualization [12]. We utilize t-SNE to reduce the 7-dimensional host feature space to 2 dimensions and visualize the data points on a scatter plot in order to visualize the clustering process for scenarios 9 and 10, which contain 10 infected hosts each.

In the following plots, red dots denote true positives, which are botnet hosts correctly clustered into the anomalous cluster, and blue dots denote true negatives, which are non-botnet hosts correctly clustered into the benign cluster. Red crosses indicate false positives, which are non-botnet hosts incorrectly clustered in the anomalous cluster, and blue crosses denote false negatives, which are infected hosts incorrectly clustered in the benign cluster.



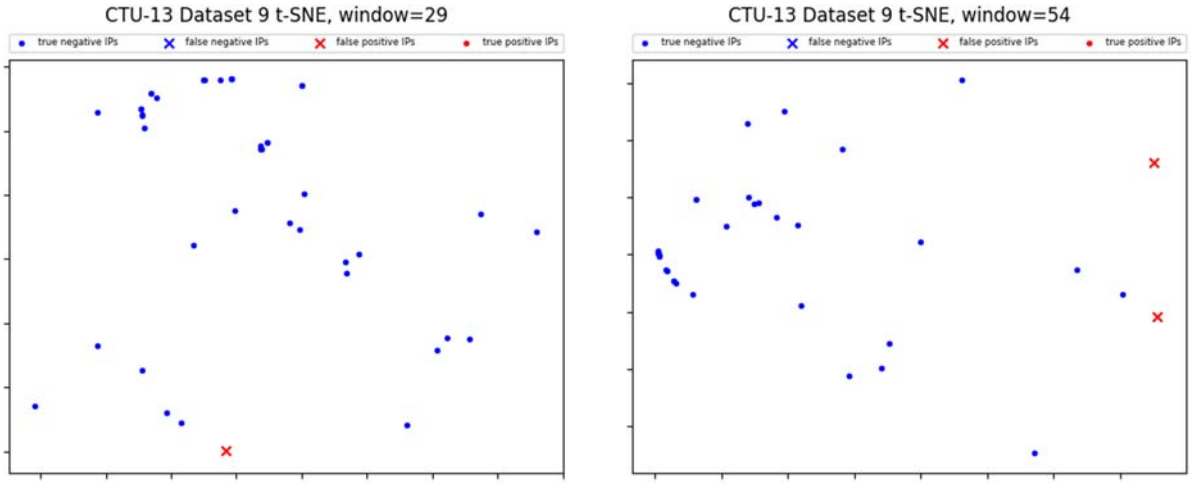


Figure 4.2.4. t-SNE of host feature vectors on some windows in scenario 9 when the botnet was *inactive*. A lack of structure in the underlying data is observed and the anomalous cluster consists of a few isolated edge points.

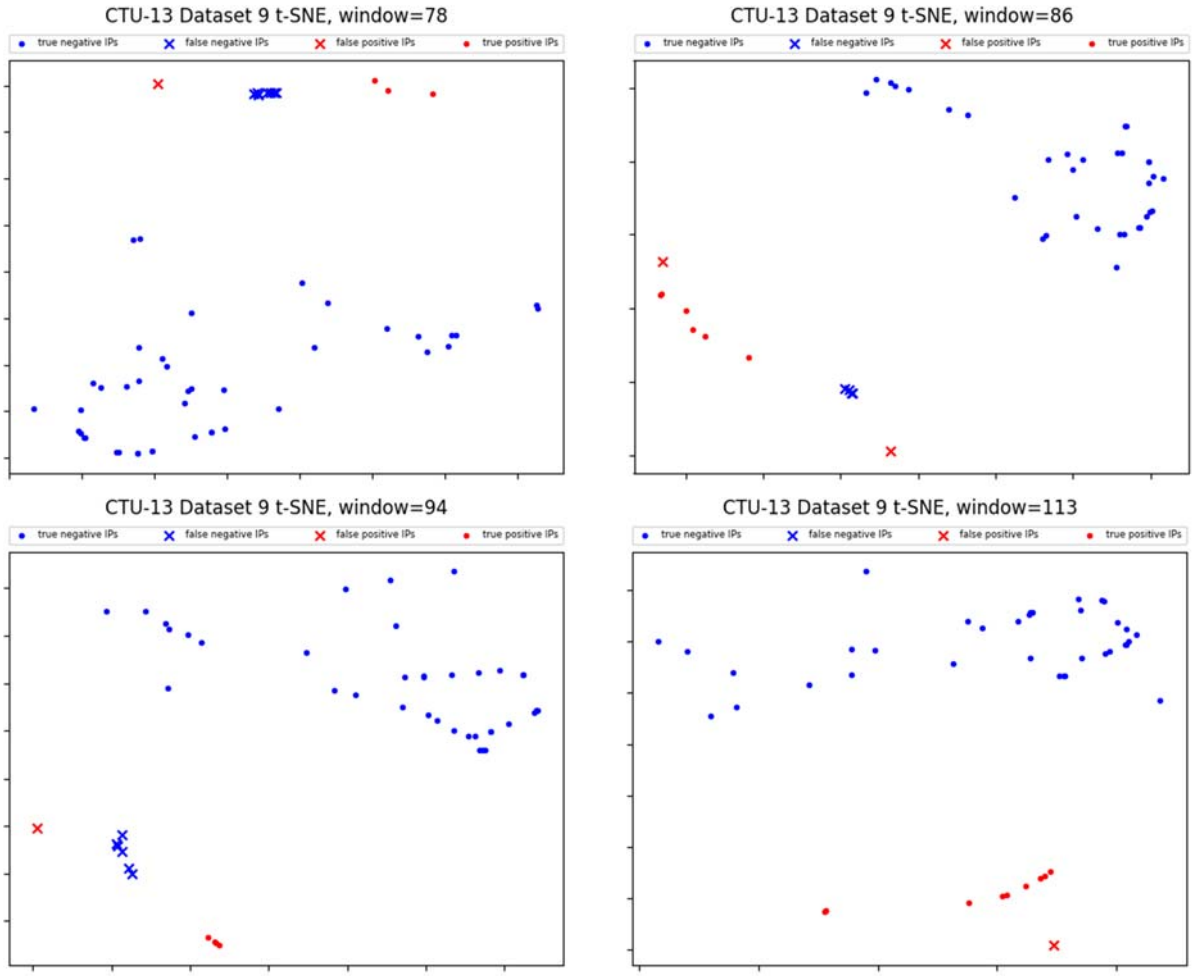


Figure 4.2.5. t-SNE of host feature vectors on some windows in scenario 9 when the botnet was *active*. Separation between points corresponding to botnet and non-botnet hosts is observed, despite the clustering algorithm failing to identify the correct clusters in all cases

From the t-SNE plots, it can be seen that during windows when the botnet is not running, there is no apparent structure to the points and the anomalous cluster largely consists of a few isolated points that are furthest from the remaining points. However, during windows when the botnet is active, the plots clearly indicate some degree of separation between points corresponding to botnet hosts and points corresponding to non-botnet hosts.

This was also observed with dataset 10 – although the classifier and clustering process produced extremely poor results, the t-SNE plots demonstrated a clear separation between botnet and non-botnet hosts, as seen in figure 4.2.3:

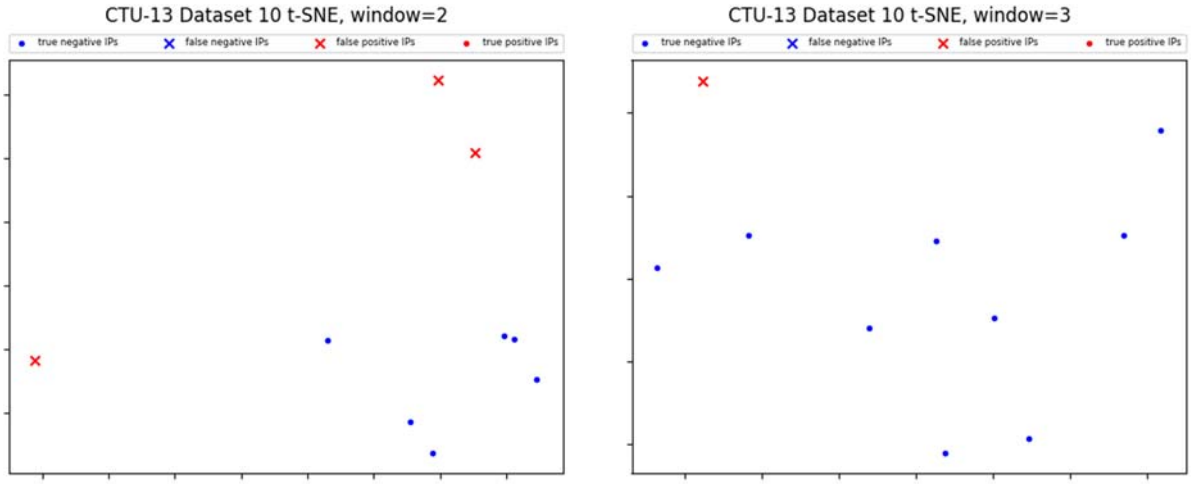
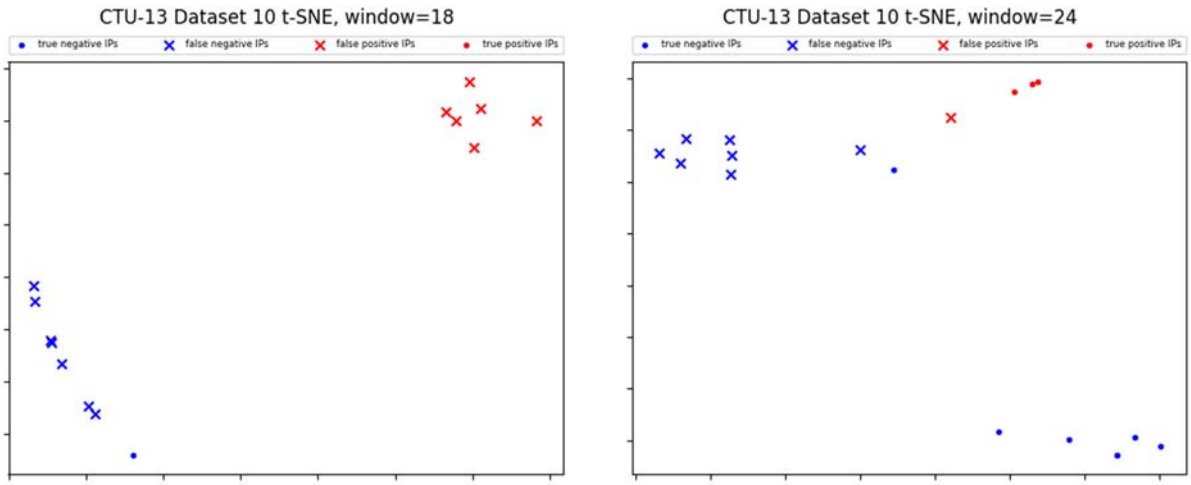


Figure 4.2.6. t-SNE of host feature vectors on some windows in scenario 10 when the botnet was *inactive*.

These plots suggest an underlying structure within the data, implying that although the current clustering algorithm does not correctly cluster the infected hosts in all windows, other clustering methods may be able to identify the structure and separate infected and non-infected hosts with a higher degree of accuracy.



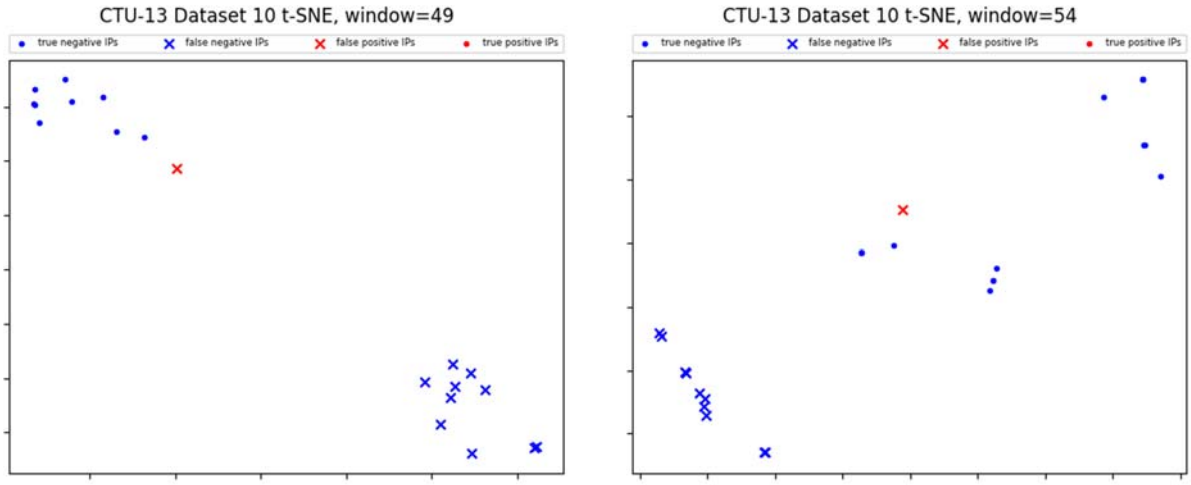


Figure 4.2.7. t-SNE of host feature vectors on some windows in scenario 10 when the botnet was *active*. Although the clustering algorithm produced incorrect clusters, the intended clusters of botnet and non-botnet hosts are observable.

4.3 Building a proof-of-concept detection and network monitoring system

To demonstrate the potential practical applications of our detection method in a real-world network monitoring scenario, we built a graphical application using PyQt4 [13] and pyqtgraph [14] that operates continuously on 300 second windows of network traffic.

The detector utilizes multiple trained classifiers that each detects a different type of botnet, and aggregates their outputs to identify botnet flows. Currently, all 7 classifiers that trained on each of the CTU-13 scenarios are used concurrently; as these scenarios contain a variety of different bots and we believe that they should be sufficient for identifying the majority of present-day botnet activity. However, the application is also extensible and supports the additional or removal of models, allowing users to customize detection for their purposes or to add new models for emerging botnet types.

In each 300 second window, flows are extracted and each model performs classification on the flows, corresponding to stage 1 of our detection process A flow is considered to be a potential botnet flow if it is classified as malicious by any one of the classifiers, and all such flows are then processed in stage 2 of the detection process, in which host-based features are generated and host-based clustering is applied. Each host is assigned an instantaneous binary score of 0 if it was clustered in the normal cluster, or an instantaneous score of 1 if it was in the anomalous cluster. The instantaneous score in that window is then used to update the host's overall score, which is an exponentially weighted moving average (EWMA) of the host's past instantaneous scores, using the parameter $\alpha = 0.3$. The host's overall score reflects its likelihood of being an infected host.

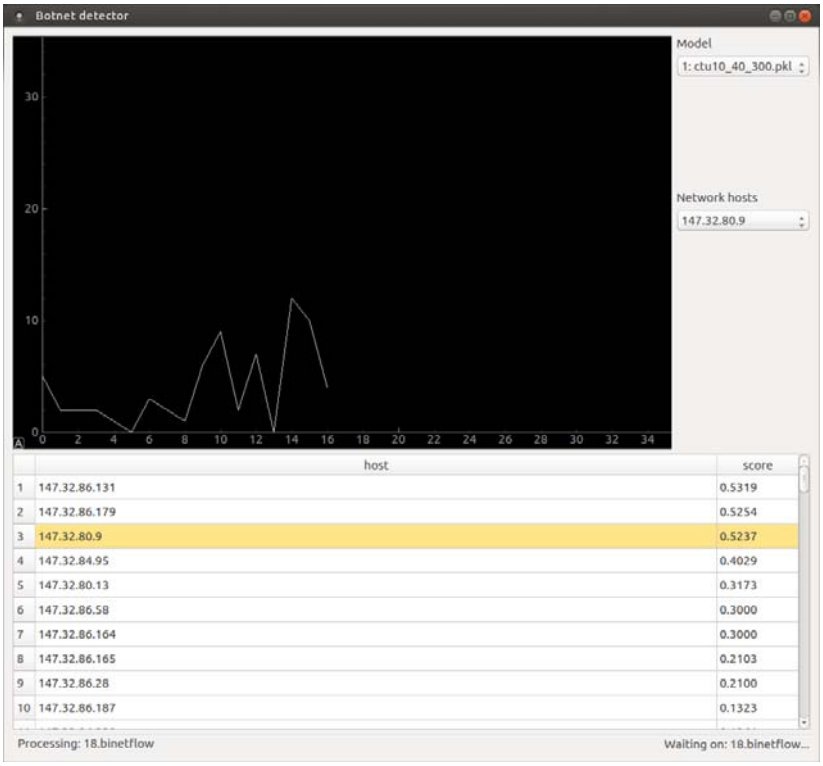


Figure 4.3.1. botd, the machine-learning based **bot**net **d**etector.

EWMA is a method of smoothing time series data that assigns exponentially decreasing weights to older data to emphasize on newer, immediate points and their neighborhood. The equation for EWMA is:

$$s_t = \alpha \cdot x_t + (1 - \alpha) \cdot s_{t-1}$$

where $0 < \alpha < 1$ is the smoothing factor (a value of α closer to 1 reduces the smoothing), s_t is the new overall host score, x_t is the instantaneous score at the current time, and s_{t-1} is the previous overall score. EWMA was used to assign scores to hosts due to the observation that true infected hosts are consistently clustered into the anomalous cluster across multiple windows, while non-infected hosts may occasionally appear in the anomalous cluster as false positives but fail to maintain anomalous cluster membership continuously. This scoring method ensures that true infected hosts will maintain a high score over time, while false positives may obtain an instantaneous score of 1 in some window, but the value will quickly decay over subsequent windows.

The interface of our detector provides two main functions. The upper graph area allows users to filter by selected model and selected internal host IP to view a graph of the number predicted botnet flows over time by the selected model associated with the selected IP. This was based on the observation that when a botnet begins running on a network, a noticeable increase in botnet activity should occur, which should be detected by the models. Although a number of false positives may occur in any window, resulting in non-botnet flows being misclassified as botnet flows, the number of false positives for a model remains a relatively

constant number and appears as a standard noise signal, as shown in figures 4.3.2 and 4.3.3. In contrast, graphs of true infected hosts display a noticeable increase in botnet flow for a specific model, as shown in figures 4.3.3 and 4.3.4.

The lower part of the interface provides a tabular view of all internal hosts and their associated score. We apply a simple heuristic to highlight a host in red if it maintains a score $>$ 0.9 for the past three consecutive windows, and mark it as an active bot. If a host was historically marked active, but is not currently active, it will be highlighted in yellow. All other hosts appear white by default. The combination of the host list and the graph provides a rapid visual indicator of anomalous behavior on a network, prompting users to check potentially suspicious hosts for botnet activity and terminate the associated machines before substantial damage or malicious activity can be carried out.

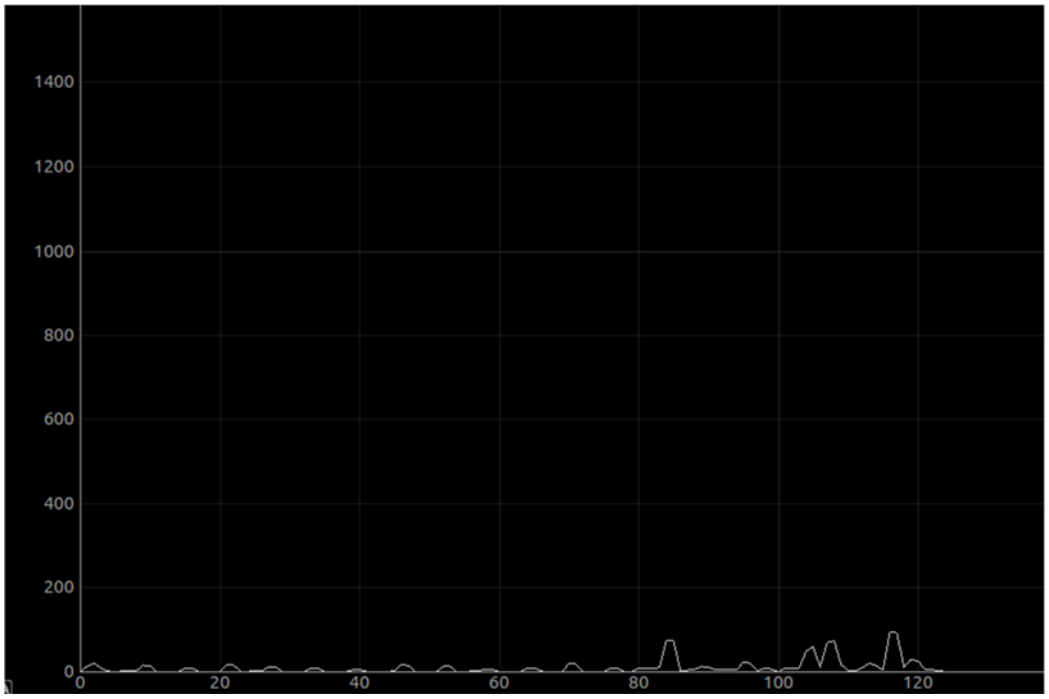


Figure 4.3.2. Number of predicted botnet flows by classifier trained on scenario 9 over time associated with 147.32.84.68, a background host in scenario 9.

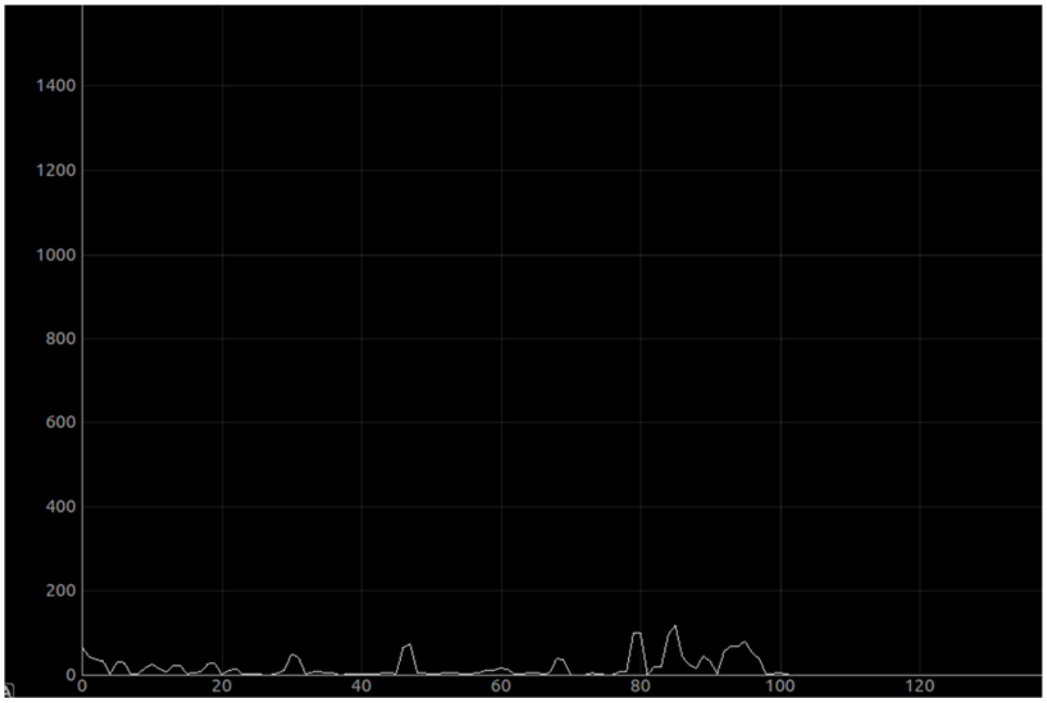


Figure 4.3.3. Number of predicted botnet flows by classifier trained on scenario 9 over time associated with 147.32.85.30, a background host in scenario 9.

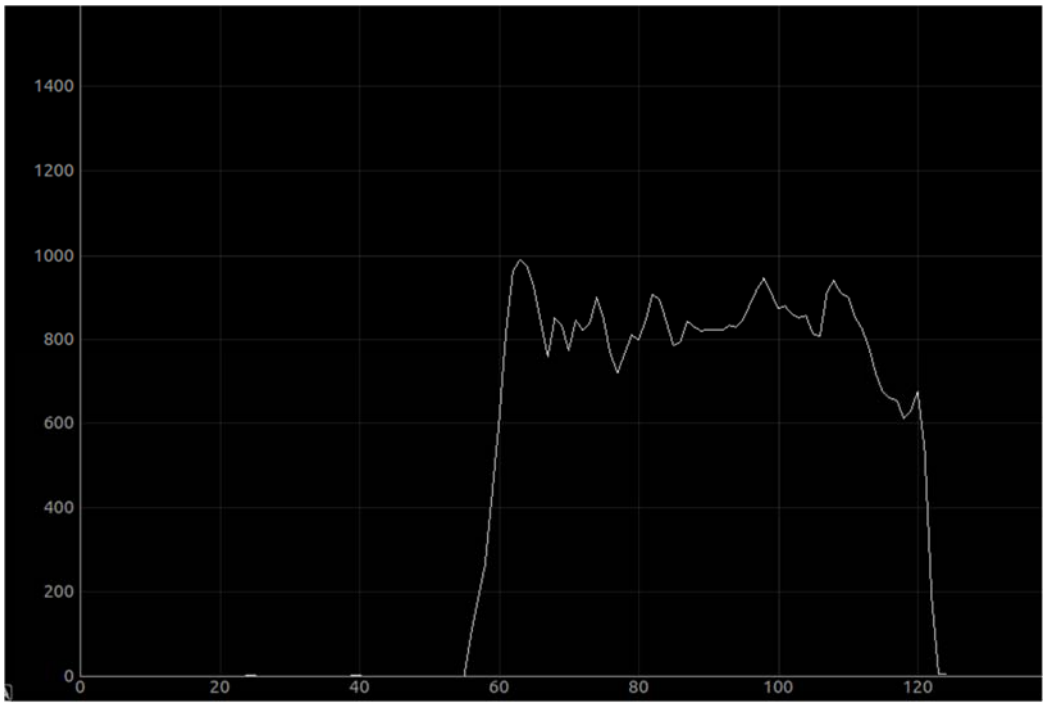


Figure 4.3.4. Number of predicted botnet flows by classifier trained on scenario 9 over time associated with 147.32.84.165, an infected host in scenario 9.

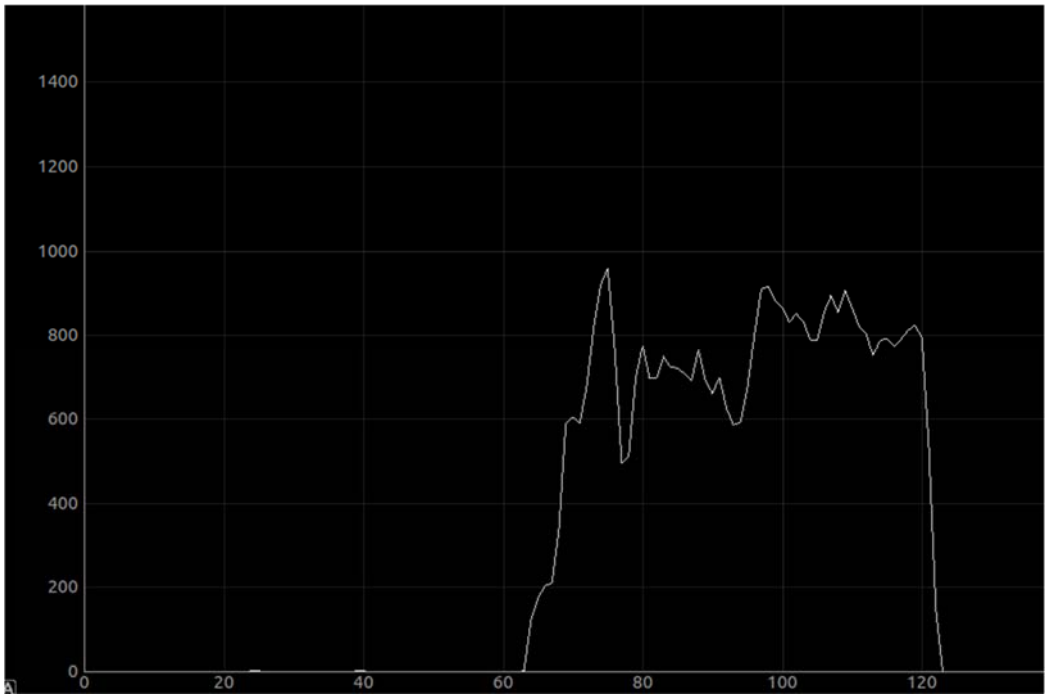
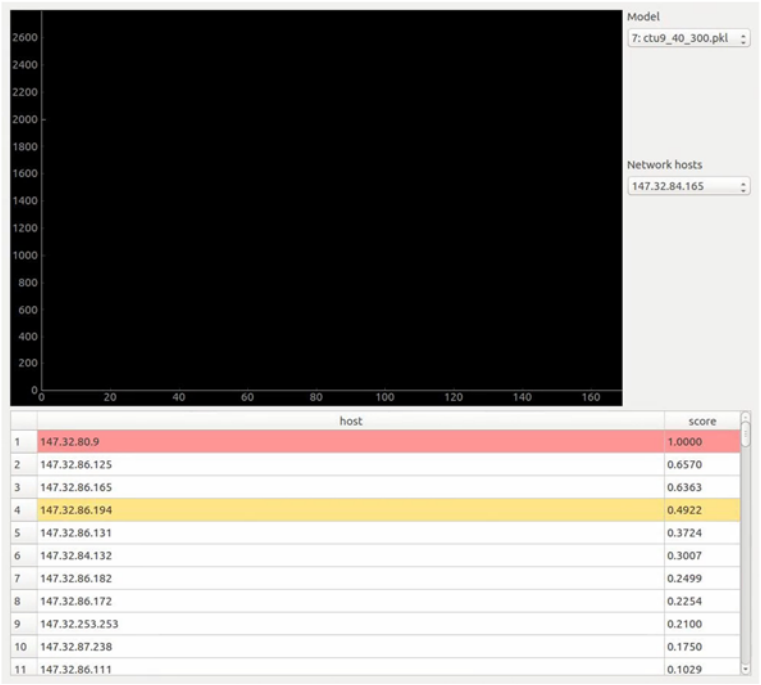


Figure 4.3.5. Number of predicted botnet flows by classifier trained on scenario 9 over time associated with 147.32.84.191, a background host in scenario 9.

The following screenshots demonstrate the usage of botd to detect botnet activity in CTU-13 scenario 9:



	host	score
1	147.32.80.9	1.0000
2	147.32.86.125	0.6570
3	147.32.86.165	0.6363
4	147.32.86.194	0.4922
5	147.32.86.131	0.3724
6	147.32.84.132	0.3007
7	147.32.86.182	0.2499
8	147.32.86.172	0.2254
9	147.32.253.253	0.2100
10	147.32.87.238	0.1750
11	147.32.86.111	0.1029

Figure 4.3.6. A window from before the botnet begins running. Various hosts are identified as potentially infected, but these are false positives and their score rapidly decays over subsequent windows.

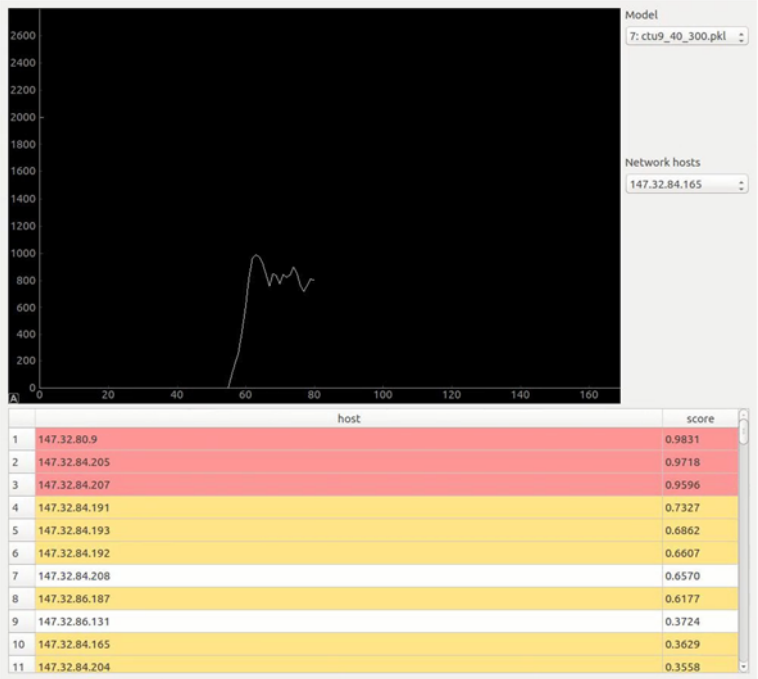


Figure 4.3.7. A window from the early stages of the botnet infection. True infected hosts are beginning to be identified as active bots.

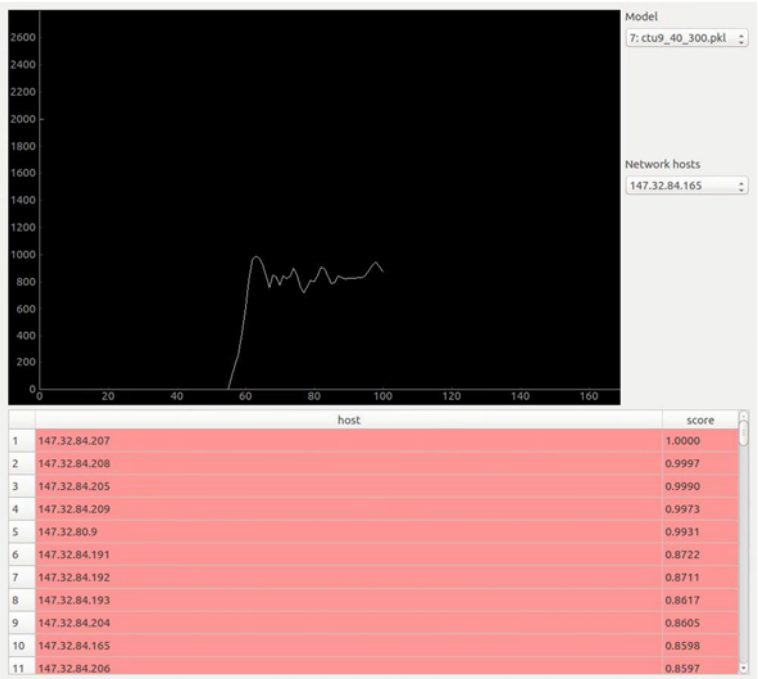


Figure 4.3.8. A window from a later stage of the infection, where the botnet has been active for some time and botnet traffic is at a peak. All infected hosts are identified as active bots.

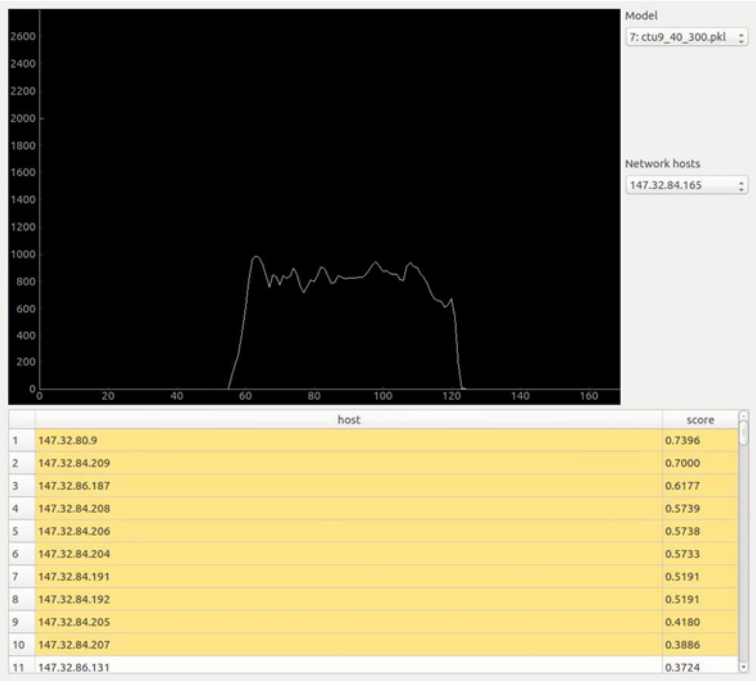


Figure 4.3.9. A window from after the botnet is terminated. Previous identified hosts remain highlighted as a visual indicator for users to check for anomalies on the selected hosts.

5 Conclusion

The results of this project show that it is possible to distinguish between botnet and non-botnet network flows to a high degree of accuracy (> 0.89 TPR), even if the flows are generated over limited time windows and provide an incomplete representation of the complete network traffic profile. This makes time-limited flows suitable for the purpose of real-time detection. Furthermore, the two-stage process of classification and clustering is able to effectively identify infected hosts for several classes of malware. We demonstrate the practical applications of this method method building a prototype real-time detection system and testing it on CTU scenario 9, successfully identifying all 10 true infected hosts with a minimal number of false positives.

Although our current clustering method is not able to produce the ideal clusters correctly in all windows, t-SNE visualization of the host data points indicates strong separability between the botnet hosts and non-botnet hosts. This has implications for further research in this area as clustering could be improved by using different algorithms in order to detect the underlying structure.

One limitation of our work is that the classifiers are trained on existing botnet data, making our detection method potentially vulnerable to new and emerging types of botnets which may have different traffic patterns. Furthermore, as we rely on statistical features of flows for classification, attackers may evade detection through varying these characteristics if they are known. Additionally, our criterion for labelling the normal and anomalous cluster is presently only a heuristic that is valid for the datasets we have been examining, and is not

true of all networks. Developing a more robust method of identifying normal and anomalous clusters based on intra-cluster variance is necessary to generalize this detection scheme to other types of botnets.

Appendix A

The following table lists all flow features extracted with Argus:

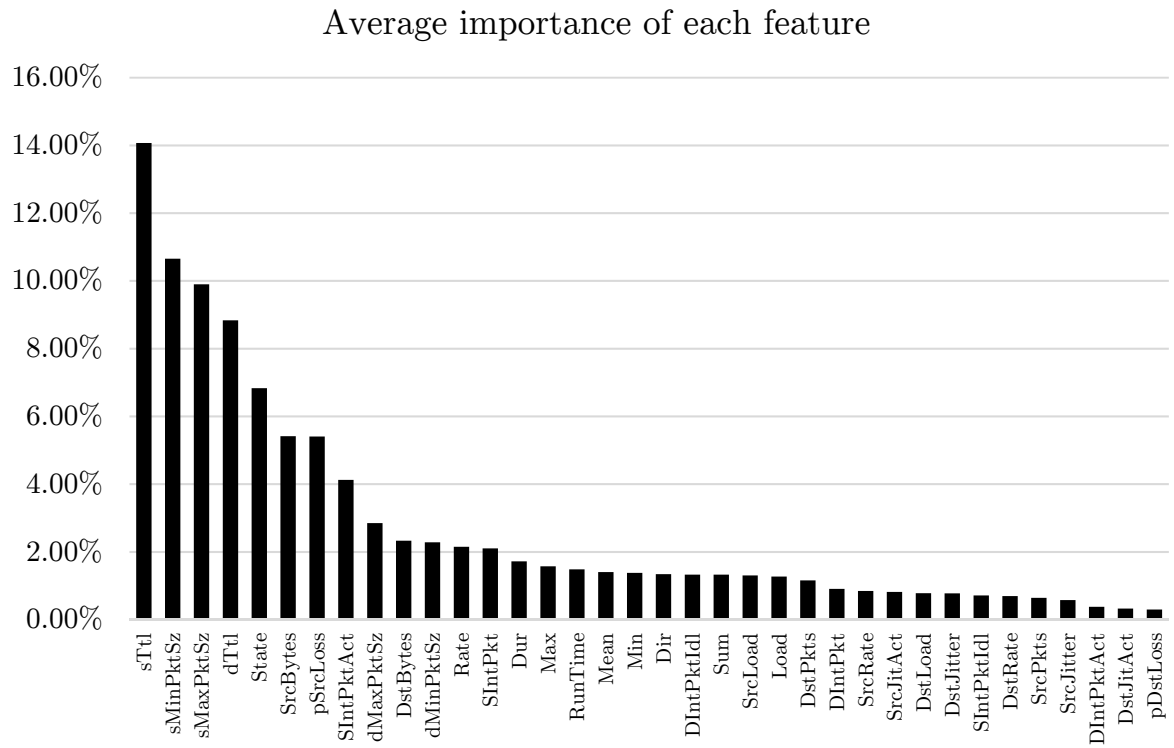
Argus field name	Feature description
srcid	argus source identifier.
stime	record start time
ltime	record last time.
flgs	flow state flags seen in transaction.
seq	argus sequence number.
smac, dmac	source or destination MAC addr.
soui, doui	oui portion of the source or destination MAC addr
saddr, daddr	source or destination IP addr.
proto	transaction protocol.
sport, dport	source or destination port number.
stos, dtos	source or destination TOS byte value.
sdsb, ddsb	source or destination diff serve byte value.
sco, dco	source or destination IP address country code.
sttl, dttl	src -> dst (sttl) or dst -> src (dttl) TTL value.
sipid, dipid	source or destination IP identifier.
smpls, dmpls	source or destination MPLS identifier.
spkts, dpkts	src -> dst (spkts) or dst -> src (dpkts) packet count.
sbytes, dbytes	src -> dst (sbytes) or dst -> src (dbytes) transaction bytes.
sappbytes, dappbytes	src -> dst (sappbytes) or dst -> src (dappbytes) application bytes.
sload, dload	source or destination bits per second.
sloss, dloss	source or destination pkts retransmitted or dropped.
sgap, dgap	source or destination bytes missing in the data stream. Available after argus-3.0.4
dir	direction of transaction
sintpkt, dintpkt	source or destination interpacket arrival time (mSec).
sintdist, dintdist	source or destination interpacket arrival time distribution.
sintpktact, dintpktact	source or destination active interpacket arrival time (mSec).

sintdistact, dintdistact	source or destination active interpacket arrival time (mSec).
sintpktidl, dintpktidl	source or destination idle interpacket arrival time (mSec).
sintdistidl, dintdistidl	source or destination idle interpacket arrival time (mSec).
sjit, djit	source or destination jitter (mSec).
sjitact, djitact	source or destination active jitter (mSec).
sjitidle, djitidle	source or destination idle jitter (mSec).
state	transaction state
suser, duser	source or destination user data buffer.
swin, dwin	source or destination TCP window advertisement.
svlan, dvlan	source or destination VLAN identifier.
svid, dvid	source or destination VLAN identifier.
svpri, dvpri	source or destination VLAN priority.
srng, erng	start or end time for the filter timerange.
stcpb, dtcpb	source or destination TCP base sequence number
tcprtt	TCP connection setup round-trip time, the sum of 'synack' and 'ackdat'.
synack	TCP connection setup time, the time between the SYN and the SYN_ACK packets.
ackdat	TCP connection setup time, the time between the SYN_ACK and the ACK packets.
tcpopt	The TCP connection options seen at initiation. The tcpopt indicator consists of a fixed length field, that reports presence of any of the TCP options that argus tracks.
inode	ICMP intermediate node.
offset	record byte offset infile or stream.
spktsz, dpktsz	histogram for the source (spktsz) or destination (dpktsz) packet size distribution
smaxsz, dmaxsz	maximum packet size for traffic transmitted by the source (smaxsz) or destination (dmaxsz).
sminsz, dminsz	minimum packet size for traffic transmitted by the source or destination.
dur	duration of a flow
rate, srate, drate	packets per second
trans	aggregation record count.
runtime	total active flow run time. This value is generated through aggregation, and is the sum of the records duration.
mean	average duration of aggregated records.
stddev	standard deviation of aggregated duration times.

sum	total accumulated durations of aggregated records.
min	minimum duration of aggregated records.
max	maximum duration of aggregated records.
pkts	total transaction packet count.
bytes	total transaction bytes.
appbytes	total application bytes.
load	bits per second.
loss	pkts retransmitted or dropped.
ploss	percent pkts retransmitted or dropped.
sploss, dploss	percent source or destination pkts retransmitted or dropped.
abr	ratio between sappbytes and dappbytes

Appendix B

The following chart illustrates the relative importances of the selected features as determined by the random forest classifier, averaged across all scenarios.



6 References

[1] W. T. Strayer, D. Lapsely, R. Walsh and C. Livadas, "Botnet detection based on network behaviour," *Advances in Information Security,* vol. 36, pp. 1-24, 2008.

[2] M. Masud, T. Al-khateeb, L. Khan, B. Thuraisingham and K. Hamlen, "Flow-based identification of botnet traffic by mining multiple log files," in *First International Conference on Distributed Framework and Applications*, 2008.

[3] S. Saad, I. Traore and A. Ghorbani, "Detecting P2P botnets through network behavior analysis and machine learning," *2011 Ninth Annual International Conference on Privacy, Security and Trust,* 2011.

[4] P. Camelo, J. Moura and L. Krippahl, "CONDENSER: A Graph-Based Approach for Detecting Botnets," 2014.

[5] S. Garcia, M. Grill, H. Stiborek and A. Zunino, "An empirical comparison of botnet detection methods," *Computers and Security Journal,* vol. 45, pp. 100-123, 2014.

[6] QoSient, LLC., "Argus: Auditing network activity," 1 June 2016. [Online]. Available: https://qosient.com/argus/index.shtml. [Accessed 22 September 2017].

[7] S. Garcia, "CTU-Malware-Capture-Botnet-51 or Scenario 10 in the CTU-13 dataset.," 05 May 2017. [Online]. Available: https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-51/. [Accessed 22 September 2017].

[8] S. Zander and C. Schmoll, "netmate-flowcalc," 4 December 2016. [Online]. Available: https://github.com/DanielArndt/netmate-flowcalc. [Accessed 22 September 2017].

[9] S. Burschka, B. Dupasquier and A. Fiaux, "Tranalyzer," 23 June 2017. [Online]. Available: https://tranalyzer.com/. [Accessed 22 September 2017].

[10] A. H. Lashkari, G. Draper-Gil, M. S. Mamu and A. A. Ghorbani, "Characterization of Tor Traffic Using Time Based Features," in *3rd International Conference on Information System Security and Privacy*, Porto, 2017.

[11] Wireshark Foundation, "Wireshark," 29 August 2017. [Online]. Available: https://www.wireshark.org/. [Accessed 22 September 2017].

[12] L. v. d. Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research,* vol. 9, pp. 2579-2605, 2008.

[13] Riverbank Computing Limited, "PyQt4 Download," 2016. [Online]. Available: https://www.riverbankcomputing.com/software/pyqt/download. [Accessed 22 September 2017].

[14] L. Campagnola, "PyQtGraph," 2012. [Online]. Available: https://github.com/pyqtgraph/pyqtgraph. [Accessed 22 September 2017].

7 Acknowledgements

I would like to acknowledge my mentor, Dr. Julian Bunn for giving me the opportunity to participate in the Caltech Summer Undergraduate Research Fellowship (SURF) program. Thank you so much for your endless patience, continued support, insight and guidance throughout this project.

I would also like to thank Bruce Nickerson and his family for the honor of being named as the J. Weldon Green SURF Fellow for 2017. Thank you for your commitment to supporting aspiring undergraduate researchers at Caltech – this project was only made possible by your generous support and contribution towards the SURF program.

Finally, I would also like to thank Arun Viswanathan and Dr. K. Mani Chandy for their assistance and input on technical aspects of this project.