Detection of botnet traffic using machine learning

Niranjan Hegde
Amrita Center for Cyber Security Systems and Networks
Amrita University
Kollam, India
niranjanhegdems@gmail.com

Gilad Gressel
Amrita Center for Cyber Security Systems and Networks
Amrita University
Kollam, India
giladgressel@gmail.com

***Abstract*—Botnet is malware where the software installed in the system connects to a command server to get instructions to do malicious activities. This type of malware can stay dormant in your system for long time. The malware authors can use it to hack into another systems or conduct ddos attacks. In this paper, we looked at different machine learning models that can be used to detect botnet traffic. We extracted the netflow using NetMate from the pcap files and trained our models to detect botnet traffic. We found that RF had 92% which was the highest among trained machine learning model namely SVM,Decision Tree and MLP. At the end, we created another dataset to see how our best model would predict it. Our trained RF model had 91.5% accuracy when tested on the new dataset.**

***Index Terms*—Malware, Botnet, Traffic analysis, Machine learning, Network**

I. Introduction

Botnet is one of the malware which works in tandem with other machines infected with same botnet to perform activities such as smtp spam,DDOS attack etc. Usually, the botnets are created using exploit kits such Underminer Exploit Kit which are basically frameworks to write malware or exploit scripts. These frameworks often provide C2 infrastructure support. This is used by the malware authors to create botnet and launch an attack against the victim or an organization.

Generally,when a botnet is infecting the machine, it establishes a connection to a control server to receive further instructions. The control server now knows how to contact the infected machine and ask it to execute instructions. An organisation usually has a huge network of computer systems. It is really difficult to use Deep packet inspection to find out the botnet traffic. But it has been observed that botnet often use dns resolution to connect to the control server. So, by passively monitoring the dns, organisation can be alerted about the botnet infection. However this method fails if the botnet is using direct ip address to connect to the control server.

In this work, we aim to create an ip-agnostic botnet detection machine learning model. The machine learning model relies upon the statistical features extracted by the NetMate to detect if a particular flow belongs to a particular botnet. We used CTU-13 scenario which consists of different scenarios. We train the machine learning models and check their performances. We convert the pcap files into flow records. A flow record is a statistical data on a network flow. A network flow is a conversation between 2 machines.

II. Literature Review

Futamura (2009) [13] proposed the use of flow records to detect worm.In the paper he mentioned that using flow records, it was able to detect an unknown worm traffic in the network. Garca et al. (2014) [1] compared BClus,CAMNEP and BotHunter methodology of detecting botnet by analysing the network packets. They created the dataset CTU-13 which had 13 scenarios - each describing various actions that can be performed by a botnet.

E. Niglar et al. (2014) [3] analysed flow based features which were further divided into 4 groups: packet-based, time-based,byte-based and behaviour-based. They used a C4.5 decision tree classifier as a model to train on the botnet dataset which was created from various sources such as ISOT,ICX and malware network samples from malware capture project. They found that on a less diverse dataset, the accuracy was 99% but on a diverse dataset, the accuracy dropped to 68%. In this paper, we will be attempted to check the performance of our trained model on a diverse dataset. K Shanti and D.Seenivasan (2014) [4] proposed a methodology in which features such as source ip and port ,destination ip and port, protocol , ICMP packets unreachable,size of the first packet flow etc were all extracted from wireshark software itself. They then used these features to train J48 decision tree classifier and Naive bayes classifier to get a true positive rate of 86.69% and 78.54% respectively. In the paper, we will be following the similar methodology however we will be using NetMate to extract features from pcap files instead of wireshark software.

Dmitri et al. (2015) [5] trained naive bayes classifier, Random Forest classifier and J48 decision tree using 925 features extracted from pcap files. Those features were extracted by analysing all the network and can be grouped into 4 categories: conversational analysis,flow analysis,session analysis and transactional analysis. The authors aim was to train classifier in such a way that it can detect unknown malware families and made sure that their classifier works even in different network environment. Therefore authors extracted data from different sandboxes such as academic sandbox,virustotal sandbox, Emerging threats sandbox etc. While training their models, the authors followed left out one malware family and then used it for testing. The authors found that all the classifier were robust against network environment and had over 75% accuracy in detecting the unknown malware.

In this paper, our goal is same however we will be training our model on all known families of botnet and then checks its performance.

Long et al. (2016) [6] proposed a flow based clustering approach in detecting botnet traffic. The authors used clusters to assign labels to the flows which was extracted from ISOT dataset using Tranalyzer tool. Once the labels are assigned by the clusters, the authors tried 3 techniques: Random Forest with K-means, Random Forest with plural voting which is taking the most popular label for the flow and Random Forest with soft voting which means the mean of all the labels. Their accuracy score was 99.3% in all cases expect random forest and soft voting being 99.37%. In this paper, we are using NetMate tool to extract flows instead of Tranalyzer tool.

Azab et al. (2016) [7] proposed using Classification of Network Information Flow Analysis (CONIFA) framework to train classifier in such a way that it can detect newer versions of botnet. The authors extracted features using Wireshark and NetMate tool which they then reduced it to 9 features using Correlation-based Feature Selection (CFS). Those 9 features are Mean Packets Forward, Largest Packet Forward, Standard Deviation inter-arrival Time Forward, Minimum inter-arrival Time Backward, Mean inter-arrival Time Backward, Standard Deviation inter-arrival Time Backward, Minimum flow Active, Maximum flow Idle and Number Push Backward. The authors however focussed solely on detecting the newer build of Zeus malware by training their classifier C4.5 Decision tree on the older build of Zeus Malware.Using this approach, they had a 0.8 f1-score when tested using a newer build of Zeus malware.

Hammerschmidt et al. (2016) [8] proposed the use of freshness as a concept to detect new changes in the dataset of botnet and ensure that their probabilistic deterministic finite automata model can detect the newer botnet versions. They used scenario 11 and scenario 10 of CTU-13 botnet dataset along with the synthetic dataset to create clusters using freshness concept. They found that it was able to discriminate between malicious and benign traffic with an error rate of 6%. Le et al. (2016) [9] used unsupervised machine learning technique called Self-organising map (SOM) on ISOT and CTU-13 dataset. They used flow records from CTU-13 and used tranalayzer to extract flow records from ISOT. Overall,they had a 99.78% in detection of botnet. In this paper, we will be using supervised machine learning approach.

Kant et al. (2017) [10] used Convolutional Neural Network (CNN) to detect botnet using network flow and compared it against other machine learning models such as Support Vector Machine (SVM),Naive Bayes and Random Forest. They found that CNN to the highest accuracy of 93.31%. Chen et al. (2017) [11] used a Tree-Shaped Deep Neural Network (TSDNN) with Quantity Dependent Backpropogation (QDBP) to detect botnet traffic flow. They created dataset by using network traffic samples of various malware families such as trojan,ransomware etc. and extracted features which can be divided into 3 groups: connection records, network packet payloads and flow behaviour records. Using these features set, they achieved an accuracy of 99.89%. They also found that just by training on the 5% of dataset, they could get 95% accuracy score.

Harnmetta and Ngamsuriyaroj (2018) [12] used decision tree classifier to classify exploit-kit traffic. They used BRO Ids which is now known as ZEEK ids to generate HTTP,DNS and FILES logs from the network traffic of exploit-kits. They further extracted features from it which can be grouped into content-specific features such as HTML script, connection-specific features such as number of HTTP connection and interaction-specific feature such as webpage transition. After training their classifier, they got 97.74% accuracy score in classifying exploit-kit traffic. In this paper, we will be using NetMate to extract features instead of BRO IDS.

M. Yeo et al. (2018) [14] used CTU-13 dataset and selected a few botnet samples to create a dataset. They then used NetMate to extract features from it to train Random Forest classifier, Support Vector Machine, Multi-layer perceptron and Convolutional Neural Network (CNN). They found that Random Forest had over 93% accuracy score. In this paper, we will be replicating their work and build a better model.

AlAhmadi and Martinovic (2018) [15] proposed MalClassifier which used CTU-13 dataset and pcaps from Stratosphere IPS project to create a dataset. They extracted features using BRO ids to generate conn.log that contains fields such as history,port etc. This is then converted to text and associated with malware. MalClassifier then further mines it using n-flows with different values of n. The author achieved 96% f1-score in classify malicious traffic flow from benign flow.

Abraham et al. (2018) [16] compared Logistic Regression, Naive Bayes, Support Vector Machine, Random Forest and Neural Networks in detection of botnet traffic. The authors created the dataset by collecting pcap files of botnet such as Trickbot and using Bro logs to extract the flow records. The authors found that Random Forest was the best model with an F1 score of 0.99 or 1.00 in case of all other botnet except bunitu where it had 0.81. They further combined logistic regression with random forest classifier to create an ensemble model that further improved the F1-score of bunitu to 0.90.

III. Methodology

A. Dataset

Dataset used here are the pcap files collected from CTU-13 dataset and pcap files containing normal traffic.

1) CTU-13 Dataset:

a) Origin: We use CTU-13 dataset [2] which can be downloaded from stratosphere ips project servers. This dataset contains the following malware: Nerris, Virut, Murlo, NSIS, Menti, RBot and Sogou. The creators of the dataset have executed these malware in an Windows XP system and have captured network thus generated in a pcap file format.

b) Feature Extraction: In [14] authors selected Nerris, Virut, Murlo, NSIS and RBOT to develop their machine learning.They created the dataset by extracting flow records from the pcap files using NetMate. The tool NetMate is an open-source network flow exporter. Tool requires the use of rule files in order to extract flows from the pcap files. However,

TABLE I
COMPARISON OF THE FLOW RECORDS OBTAINED USING NETMATE

Malware Name	No. of Malware flows mentioned in the paper	No. of Malware flows from obtained by me
Nerris	40961	6163
Nerris	20941	4740
Rbot	53518	2591
Rbot	5160	29
Virut	1802	420
Murlo	12254	2364
Nerris	184987	42267
NSIS	2143	6921
Virut	4000	5354

the authors have not described their rule file which they used with NetMate. We have used the default rule files available with the tool. Table I shows the comparison between the number of flows extracted by the authors and the number of flows extracted by us. The number of features extracted by NetMate tool is 45.

Authors used the following classifiers: Random Forest(RF), Support Vector Machine (SVM), Multi-layer Perceptron (MLP) and Convolutional Neural Network(CNN). They implemented all except CNN using sckit-learn python package. They used keras package to implement CNN. In this paper, we will be implementing their model along with the parameters mentioned by them.

c) Preprocessing: In [14], authors have removed duplicates from our dataset to ensure that dataset remains unbiased. Authors have randomly selected 2000 samples from each class to ensure that there is no class imbalance. We have also randomly selected 2000 samples from each botnet. Authors further used the following features in their dataset as shown in Table II. We kept those features in the dataset.

2) Normal Dataset:

a) Origin: To create a normal dataset, we downloaded pcap files from Stratosphere's mixed malware captures that contains both normal and infected pcap and Stratosphere's normal captures. We used pcap files from CTU-Normal-7, CTU-Normal-12, Mixed-Capture-1 and Mixed-Capture-2 [17]. These particular captures were used because they provided clean pcap files with normal user activities. We believe that using these captures represents the user activity as close as possible.

b) Feature extraction: We used NetMate tool to extract flow records from these pcap files. We got 16328 flow records with 45 features.

c) Preprocessing: Just like we did with CTU-13 botnet dataset, we kept the features used by the authors and dropped the remaining features. We also sampled only 2000 flow records from the normal dataset to ensure that there is no class imbalance. We removed the duplicates from the dataset.

B. Experiment Run

1) Reproducing work: The authors used z-score to normalize their dataset while training their dataset. In [14] authors mentioned the following parameters to their model:

TABLE II
FEATURES IN THE DATASET

No.	Feature Name	Feature Description
1	total_fpackets	Total packets in the source to destination
2	total_fvolume	Total bytes in the source to destination
3	total_bpackets	Total packets in the destination to source
4	total_bvolume	Total bytes in the destination to source
5	min_fpktl	Size of the smallest packet in the source to destination (in bytes)
6	mean_fpktl	Mean size of packets in the source to destination (in bytes)
7	max_fpktl	Size of the largest packet in the source to destination (in bytes)
8	std_fpktl	Standard deviation of mean of the packets in the source to destination (in bytes)
9	min_bpktl	Size of the smallest packet in the destination to source (in bytes)
10	mean_bpktl	Mean size of packets in the destination to source (in bytes)
11	max_bpktl	Size of the largest packet in the destination to source (in bytes)
12	std_bpktl	Standard deviation from the mean of the packets in the destination to source (in bytes)
13	min_fiat	Minimum amount of time between two packets in the source to destination (in microseconds)
14	mean_fiat	Mean amount of time between two packets in the source to destination (in microseconds)
15	max_fiat	Maximum amount of time between two packets in the source to destination (in microseconds)
16	std_fiat	Standard deviation of mean time between two packets in the source to destination (in microseconds)
17	min_biat	Minimum amount of time between two packets in the destination to source (in microseconds)
18	mean_biat	Mean amount of time between two packets in the destination to source (in microseconds)
19	max_biat	Maximum amount of time between two packets in the destination to source (in microseconds)
20	std_biat	Standard deviation of mean time between two packets in the destination to source (in microseconds)
21	duration	Duration of the flow (in microseconds)
22	min_active	Minimum amount of time that the flow was active before going idle (in microseconds)
23	mean_active	Mean amount of time that the flow was active before going idle (in microseconds)
24	max_active	Maximum amount of time that the flow was active before going idle (in microseconds)
25	std_active	Standard deviation of mean time a flow was active before going idle (in microseconds)
26	min_idle	Minimum time a flow was idle before becoming active (in microseconds)
27	mean_idle	Mean time a flow was idle before becoming active (in microseconds)
28	max_idle	Maximum time a flow was idle before becoming active (in microseconds)
29	std_idle	Standard Deviation of mean time a flow was idle before becoming active (in mircoseconds)
30	sflow_fpackets	Average number of packets in a sub flow in the source to destination
31	sflow_fbytes	Average number of bytes in a sub flow in the source to destination
32	sflow_bpackets	Average number of packets in a sub flow in the destination to source
33	sflow_bbytes	Average number of packets in a sub flow in the destination to source
34	total_fhlen	Total bytes used in headers in the source to destination.
35	total_bhlen	Total bytes used in headers in the destination to source

- Random Forest Classifier: Estimators- 128, Max Features- 16 and Criterion- Gini.
- Multi-Layer Perceptron: Hidden layer- 5,Activation- Relu and Optimizer - Adam
- Support Vector Machine: C- 1.0 and Kernel function- rbf

We divide our dataset which contains both botnet and normal network traffic in a 80-20 split of training and testing dataset. This details was not mentioned by the authors. Our model should classify flow records as normal or belonging to a certain botnet. We fit the model on the training dataset and tested it on the testing dataset to ensure we get the performance of the model when it has not seen the complete flow record of the classes.

We used precision, recall and f1-score as metrics. Precision (Eq. 1) helps in determining the number of correct predictions made overall. Recall (Eq.2) helps in determining how many of the positive predictions made were correct. F1-score (eq. 3) basically gives the both precision and recall to give us a final score. By using these metrics, we can identify which class is being misclassified and false positives by the model. We also use a confusion matrix to see which class is being misclassified as another class.

$$Recall = \frac{True\,Positives}{True\,Positives + False\,Negatives} \tag{1}$$

$$Precision = \frac{True\,Positives}{True\,Positives + False\,Positives} \tag{2}$$

$$F1\,score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

True Positives are the correctly predicted positive values.
True Negatives are the correctly predicted negative values.
False Positives are the wrongly predicted positive values.
False Negatives are the wrongly predicted negative values.

2) Improving performance: We use the 3 models: Random Forest Classifier, SVM and MLP. We use the validation curve to find the best parameter values which gives us the maximum performance. We train the models in the same fashion the authors have trained but with different parameters. We use the same metrics as earlier.

For Random Forest Classifier,parameters remain unchanged. For SVM,we increase the value of C which decreases the strength of regularization which boosts the performance of our model. For MLP, increasing the number of hidden layer and epoch which determines how well a model can learn, we have increased the performance.

IV. Results and Analysis

The results obtained from our experiment are different from the authors. As per [14] authors had over 83% in precision,recall and f1-score for all the models. When we use the same model with same parameters, we go very different results. Table III shows the result of random forest classifier model. Table IV shows the result of support vector machine(SVM). Table V shows the result of MLP.

TABLE III
Results of RF

	Precision	Recall	F1-score
Normal	0.975490	0.985149	0.980296
Nerris	0.788835	0.761124	0.774732
RBot	0.964824	0.972152	0.968474
NSIS	0.975871	0.981132	0.978495
Virut	0.783455	0.807018	0.795062
Murlo	0.987437	0.972772	0.980050

TABLE IV
Results of SVM

	Precision	Recall	F1-score
Normal	0.637450	0.792079	0.706402
Nerris	0.636029	0.405152	0.494993
RBot	0.970190	0.906329	0.937173
NSIS	0.853659	0.849057	0.851351
Virut	0.573585	0.761905	0.654467
Murlo	0.977654	0.866337	0.918635

Based on the results, it seems like the botnet traffic of nerris and virut is causing confusion. Their flow records are being misclassified by the classifier. One of the possible reason would be that the executable used by the CTU-13 is also reported as either nerris or virut by virustotal. This misclassification was also seen in [9] and [11] where they used deep learning techniques to identify different botnet traffic. Also, we got very different results than the one mentioned in [14]. Possible reasons could be our use of NetMate tool with a default rule file. The authors did not mentioned their rule which caused difference in flow records obtained from pcap files. This could have led to a different dataset and hence the decreased performance of the model given by the authors.

Fig.1, Fig.2 and Fig.3 shows that all the 3 models are misclassifying the nerris and virut flow records. In fact, out of three models, Random Forest Classifier is working the best. When we trained RF,MLP and SVM , we found that we found the parameters used by the authors were the best for RF but there are better parameters for MLP and SVM which improved

TABLE V
Results of MLP

	Precision	Recall	F1-score
Normal	0.617312	0.670792	0.642942
Nerris	0.566308	0.370023	0.447592
RBot	0.963255	0.929114	0.945876
NSIS	0.818414	0.862534	0.839895
Virut	0.525054	0.604010	0.561772
Murlo	0.778271	0.868812	0.821053

TABLE VI
Comparison of our trained model vs authors' model (F1-score))

	Our trained model	Authors model
RF	0.90735032091949	0.90735032091949
SVM	0.78573994514041	0.76368325271054
MLP	0.82978226289146	0.72226082188938

the performance.

Table VI shows the performance of our model vs authors model using f1-score as the metric. If we compare confusion matrix in Fig.2 and Fig.4 , we can see that the classification between virut and nerris has improved. Also, the overall misclassification has improved. If we compare confusion matrix in Fig.3 and Fig.5, we see that our trained model out performance the authors' MLP model.

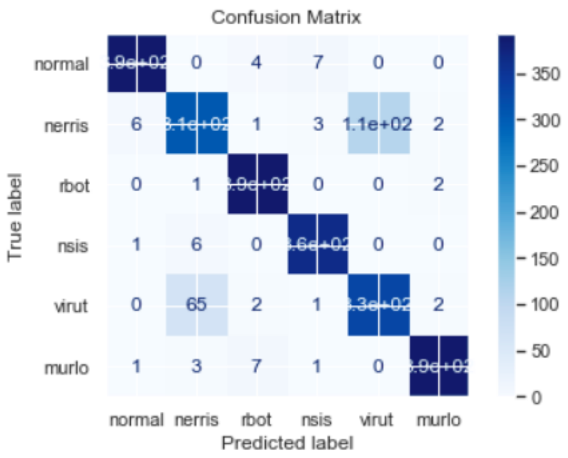

Fig. 1. Confusion Matrix when RF is used

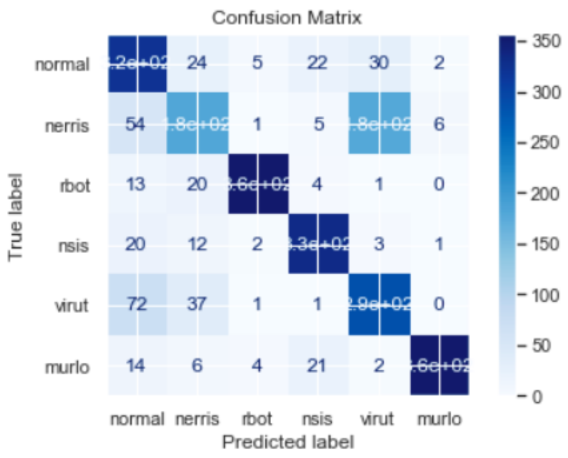

Fig. 2. Confusion Matrix when SVM is used

REFERENCES

[1] S. Garca, M. Grill, J. Stiborek, and A. Zunino, An empirical comparison of botnet detection methods, Comput. Secur., vol. 45, pp. 100123, Sep. 2014, doi: 10.1016/j.cose.2014.05.011.

[2] Garca et al. - 2014 - An empirical comparison of botnet detection method.pdf. .

[3] E. Biglar Beigi, H. Hadian Jazi, N. Stakhanova, and A. A. Ghorbani, Towards effective feature selection in machine learning-based botnet detection approaches, in 2014 IEEE Conference on Communications and Network Security, Oct. 2014, pp. 247255, doi: 10.1109/CNS.2014.6997492.

[4] K. Shanthi and D. Seenivasan, Detection of botnet by analyzing network traffic flow characteristics using open source tools, in 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, India, Jan. 2015, pp. 15, doi: 10.1109/ISCO.2015.7282353.

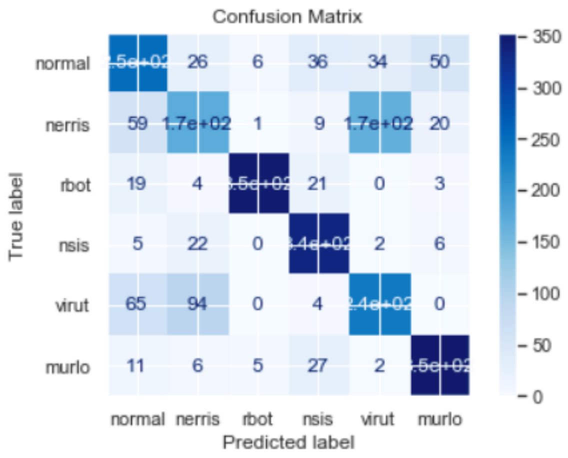

Fig. 3. Confusion Matrix when MLP is used

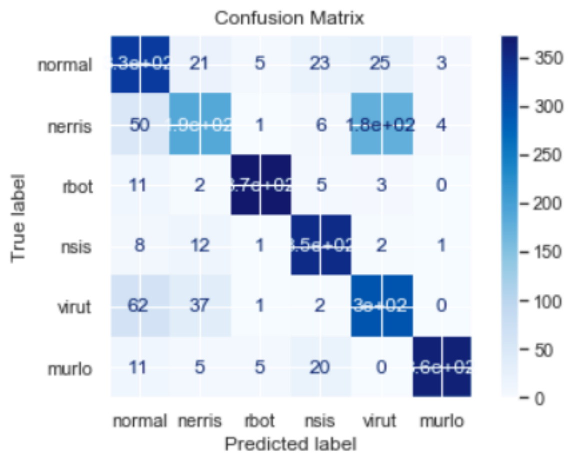

Fig. 4. Confusion Matrix when our trained SVM is used

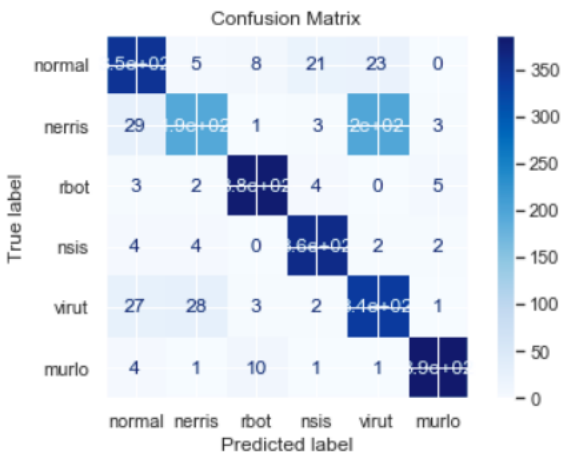

Fig. 5. Confusion Matrix when our trained MLP is used

[5] D. Bekerman, B. Shapira, L. Rokach, and A. Bar, Unknown malware detection using network traffic classification, in 2015 IEEE Conference on Communications and Network Security (CNS), Florence, Italy, Sep. 2015, pp. 134142, doi: 10.1109/CNS.2015.7346821.

[6] L. Mai, Y. Kim, D. Choi, N. Khac, T. V. Phan, and M. Park, Flow-Based Consensus Partitions for Botnet Detection, p. 3, 2016.

[7] A. Azab, M. Alazab, and M. Aiash, Machine Learning Based Botnet Identification Traffic, in 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, Aug. 2016, pp. 17881794, doi: 10.1109/TrustCom.2016.0275.

[8] C. Hammerschmidt, S. Marchal, R. State, and S. Verwer, Behavioral clustering of non-stationary IP flow record data, in 2016 12th International Conference on Network and Service Management (CNSM), Montreal, QC, Canada, Oct. 2016, pp. 297301, doi: 10.1109/CNSM.2016.7818436.

[9] D. C. Le, A. Nur Zincir-Heywood, and M. I. Heywood, Data analytics on network traffic flows for botnet behaviour detection, in 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, Greece, Dec. 2016, pp. 17, doi: 10.1109/SSCI.2016.7850078.

[10] V. Kant, Er. M. Singh, and N. Ojha, An efficient flow based botnet classification using convolution neural network, in 2017 International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, Jun. 2017, pp. 941946, doi: 10.1109/ICCONS.2017.8250603.

[11] Y.-C. Chen, Y.-J. Li, A. Tseng, and T. Lin, Deep learning for malicious flow detection, in 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Montreal, QC, Oct. 2017, pp. 17, doi: 10.1109/PIMRC.2017.8292316.

[12] S. Harnmetta and S. Ngamsuriyaroj, Classification of Exploit-Kit behaviors via machine learning approach, in 2018 20th International Conference on Advanced Communication Technology (ICACT), Feb. 2018, pp. 468473, doi: 10.23919/ICACT.2018.8323798.

[13] K. Futamura, Simple security using flow data, in 2009 18th Annual Wireless and Optical Communications Conference, Newark, NJ, USA, May 2009, pp. 14, doi: 10.1109/WOCC.2009.5312784.

[14] M. Yeo et al., "Flow-based malware detection using convolutional neural network," 2018 International Conference on Information Networking (ICOIN), Chiang Mai, 2018, pp. 910-913, doi: 10.1109/ICOIN.2018.8343255

[15] B. A. AlAhmadi and I. Martinovic, MalClassifier: Malware family classification using network flow sequence behaviour, in 2018 APWG Symposium on Electronic Crime Research (eCrime), San Diego, CA, May 2018, pp. 113, doi: 10.1109/ECRIME.2018.8376209.

[16] B. Abraham, A. Mandya, R. Bapat, F. Alali, D. E. Brown, and M. Veeraraghavan, A Comparison of Machine Learning Approaches to Detect Botnet Traffic, in 2018 International Joint Conference on Neural Networks (IJCNN), Jul. 2018, pp. 18, doi: 10.1109/IJCNN.2018.8489096.

[17] Normal capture accessed at https://www.stratosphereips.org/datasets-normal on May 27 2020.