Detecting Botnets Through Deep Learning and Network Flow Analysis

Ji An Lee and Fabio Di Troia

Abstract Botnet attacks pose a serious threat to the Internet infrastructure and its users. Botnets are operated through a command and control (C&C) channel which uniquely distinguishes it from other typical malware threats. The C&C server sends commands to the botnets to execute malicious activities using common Internet protocols, such as Hypertext transfer (HTTP), and Internet Relay Chat (IRC). Since these protocols are common, detecting botnet activities has been a challenge. This paper proposes an approach to identify the IP addresses of C&C servers and infected hosts in a network, without prior knowledge of the addresses or the type of the botnet. The approach is based on the observation that there are unique patterns in the communication between C&C server and bots which could be used to distinguish botnets from the background traffic. Regular botnet activities such as orchestrated attacks, heartbeat signals, or periodic distribution of commands are the main causes that produce such patterns. Deep learning techniques are applied on the extracted patterns to classify potential botnet traffics. The results show this pattern-based botnet detection technique is able to achieve high classification accuracy with low false positive rate.

1 Introduction

A botnet is a collection of machines that have been intentionally infected with malware to carry out various scams and cyber-attacks on the Internet without the authorization of the machines' owners. Once infected, these machines are remotely controlled by a botmaster through communication channels using standard networking protocols. At the core of the botnet are Command-and-Control (C&C) servers that act as headquarters for botnet communication [22]. Cybercriminals use C&C servers to distribute new commands to bots as well as receive execution results. Some of the malicious activities carried out by the bots include identity

J. A. Lee · F. Di Troia (✉)
San Jose State University, San Jose, CA, USA
e-mail: fabio.ditroia@sjsu.edu

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

M. Stamp et al. (eds.), *Artificial Intelligence for Cybersecurity*, Advances in Information Security 54, https://doi.org/10.1007/978-3-030-97087-1_4

theft, security breaches, distribution of SPAM emails, fraudulent financial scams, and perpetrated DDoS (distributed denial of service) attacks [22]. Potentially, any computer machine connected to the Internet has the possibility to become a compromised bot, thus, the impact of a botnet is estimated to cause severe damage. Many studies have been conducted to effectively detect botnet activities and protect machines from botnets. Despite these efforts, botnet attacks continue to pose a serious threat to the Internet infrastructure due to its constantly evolving nature [29]. Some of the previously explored botnet detection techniques include honeypot, passive anomaly analysis, and network traffic based classification [7, 17]. Among these three categories, network traffic based botnet classification is of particular interest for our work. By analyzing botnet behavior, some distinctive traits of botnet traffic may be recognized to help identify botnet activities. For instance, botnets are required to connect with the C&C servers to provide status updates and receive new commands. This unique characteristic suggests that botnets need to periodically communicate with C&C servers to be able to function properly. Using this information, the signs of periodic traffic may serve as a strong indicator for botnet activity. Furthermore, even more features can be specified by reviewing botnet behavior and network traffic for the purpose of botnet detection. The goal of this paper is to propose a deep learning model that detects botnet activities in a network by analyzing its packet captures. This paper tries to find answers to the following problems:

1. Given a dataset that consists of botnet, normal, and background traffic, is it possible to train a deep learning model that accurately classifies botnet traffic?
2. In real-life scenarios, botnets generate a significantly lower proportion of network traffic than non-botnet traffic. How should the dataset imbalance issue be addressed?
3. What are the key features of network traffic that is required to train the deep learning model?

The structure of the remaining Sections of this paper is as follows: Sect. 2 covers background information on the topics covered in this paper. Section 3 analyzes the relevant work on the same domain. Section 4 explains the key details about the CTU-13 dataset used in this project. Section 5 describes the methodology followed in this paper, the specific implementation details for feature extraction, and the deep learning model construction and evaluation. Section 6 summarizes the key findings and reports the overall project result.

2 Background

This section discusses the background domain which this paper is based on. It mainly focuses on botnets, autocorrelation analysis, and deep neural networks.

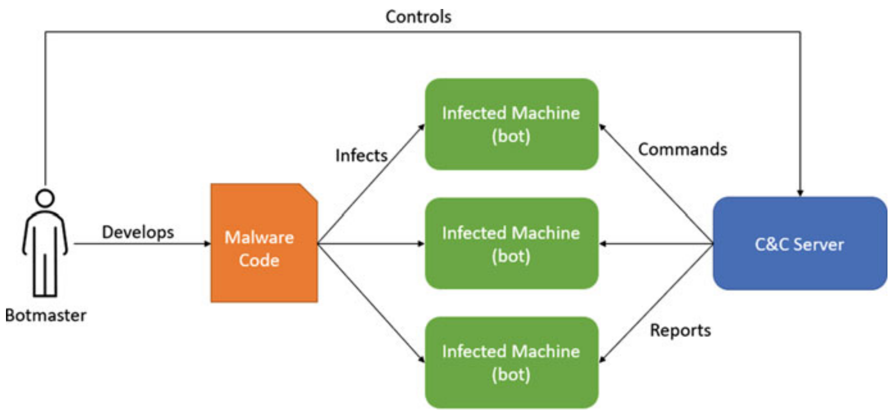


Fig. 1 Overview of the general botnet architecture

2.1 Introduction to Botnets

The term ‘botnet’ is a compound word from ‘robot’ and ‘network’. It refers to a network of compromised machines that works for a cybercriminal to perform malicious activities over the Internet. Initially, the size of botnets was roughly a few hundreds. However, with the advance of Internet technologies and computing power, the number of bots that comprise a botnet have significantly increased to a few hundreds of thousands [10]. Using this massive network of bots, hackers conduct illegal activities such as personal data theft, server attacks, and distribution of malware to infect more machines [18]. Botnets are controlled by a masterbot through Command and Control (C&C) servers [8]. This control server plays a critical role in distributing commands to the botnets and keeping a list of which botnets are active and inactive. Figure 1 illustrates the architecture of a general botnet system. Botmaster develops a malware program and infects machines through the Internet. The set of infected machines are then operated by a C&C server which is directly controlled by the botmaster.

There are four types of known C&C architectures as shown in Fig. 2. With the direct architecture, botmasters directly infect and control the botnet. However, with the possibility to trace the botmaster from the bots and the limited scaling, it lost popularity within the cybercriminal society. Centralized architecture is identical to the architecture shown in Fig. 2 and was discussed in [16]. Contrary to the direct architecture, the centralized architecture’s bots do not lead traces directly to the botmaster. P2P or decentralized architecture evades the single point failure issue by enabling communication between all nodes in the network. A hybrid architecture is an expansion of the P2P network that enables large scaling of the number of bots that a botmaster can operate.

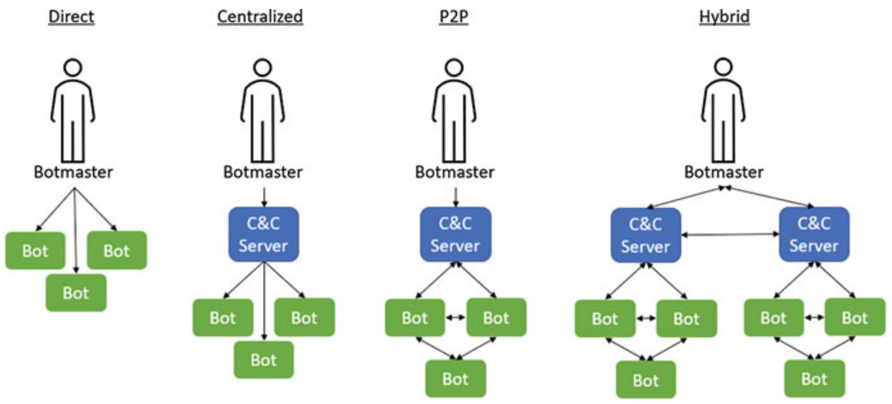
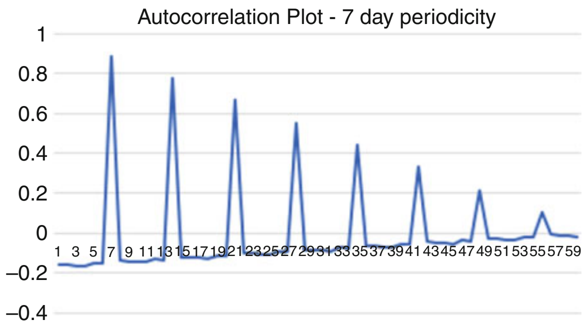


Fig. 2 The four types of known C&C architectures

Fig. 3 Autocorrelation plot of a periodic signal of lag 7



2.2 *Autocorrelation Analysis*

Given a dataset that consists of observations of a phenomenon at different points in time, autocorrelation analysis seeks for patterns over the time series. The term autocorrelation refers to the degree of similarity between a given time and a time-shifted version of itself. For instance, if it rained heavily every Monday, then the autocorrelation analysis would find the periodicity of the rain as seven days.

Using the rainy day example, an autocorrelation plot can be constructed. In Fig. 3, the x-axis is lag and the y-axis is the value of the autocorrelation function. The plot shows a peak every seven lags: 7, 14, 21, 28, and more. This means that the original input shows a repeating pattern of seven days. Similar to this example, the high peaks in the autocorrelation plot are important when using the autocorrelation tool. An autocorrelation value is considered significant if the value exceeds the threshold, otherwise known as the confidence interval (CI). The formula to calculate the CI is shown here

$$CI_{AC_k} = [AC_k - 1.96 \times \frac{AC_{SE,k}}{\sqrt{N}}, AC_k + 1.96 \times \frac{AC_{SE,k}}{\sqrt{N}}]$$

where AC_i is the autocorrelation estimate at lag i and N is the number of time steps in the sample. More details can be found in [11].

2.3 Deep Neural Networks

Deep learning is a subset of machine learning where the structure is constructed theoretically similar to living brains. It is commonly referred to as an artificial neural network (ANN). Two common examples are convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The main difference between the two types of networks is that CNN uses the connectivity patterns between the internal neurons while RNN uses time-series information that is strongly correlated to the order and the neighboring input data. Due to this difference, we use CNN rather than RNN for the deep neural network section of the implementation. There are three types of layers that make up the CNN: convolutional layer, pooling layer, and fully-connected (FC) layer. The convolutional layer is the first layer in the network that is used to extract various features from the input data. This phase consists of mathematical computations of convolution between the input data and a $K * K$ filter. This filter slides through the input array and produces a feature map which provides information about different qualities of the dataset. The pooling layer is generally used to decrease the size of the feature map to increase computational efficiency. In particular, the process of max pooling is the operation of selecting the largest element in the feature map. Lastly, the fully connected layer consists of weights and biases of the neurons and this information is used to connect the neurons between the FC layers. The components of such network are described in Fig. 4.

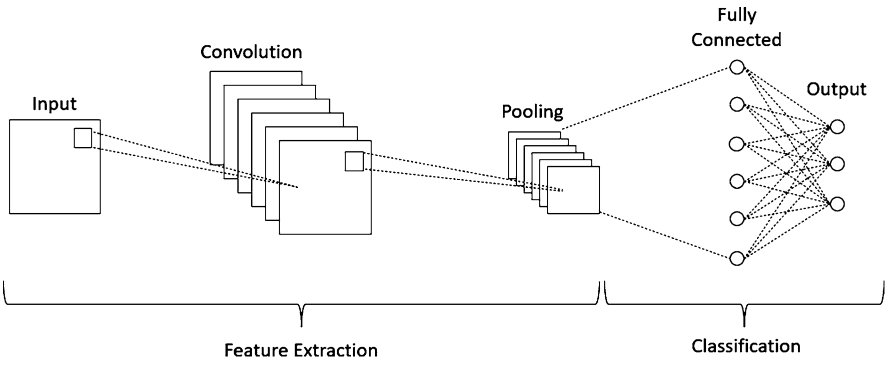


Fig. 4 Components of a convolutional neural network

3 Related Work

In the research field of malware detection, static and dynamic malware analysis have gained popularity over the recent years [21]. While static malware analysis focuses on using signature-based approaches such as file fingerprinting and virus scanning, dynamic malware analysis focuses on analyzing the behavior-based features of the malware samples [17]. Since the focus of static analysis is on the structure of the executable itself, the analysis can be performed without running the actual executable file. Although this is a cost-efficient approach, static analysis becomes vulnerable to malware threats that only reveal themselves during run-time. Dynamic analysis, on the other hand, detects malware by monitoring program activities rather than the program structure itself. Using a behavior-based approach, activities like network communication, API calls, system calls and system resource usages are analyzed. With dynamic analysis, the intention is to understand the working mechanism of a program and use this information to flag any suspicious program behavior. Due to this characteristic, dynamic analysis is resilient and flexible to more sophisticated and obfuscated types of malware. An example of a study that relies on dynamic analysis is [2], where the authors detect sniffing programs within a network. Sniffing is a type of network attack where an attacker tries to seek for vulnerabilities in a network by gathering as much information as possible about the targeted network. Sniffing is conducted by software programs called Sniffers that passively analyzes the incoming and outgoing traffic in a network. Due to this passive behavior, detecting Sniffers has been a challenge. The paper also suggests a measurement-based approach to pinpoint hosts running Sniffers by flooding the network with packets and comparing the round-trip response time among hosts. The findings of this study show that monitoring the behavior of programs can serve as an accurate indicator for finding malicious hosts. As cyber criminals and their malicious programs become more innovative and creative, the mission to detect malware paved the way for a hybrid model that employs static and dynamic analysis in conjunction. In [3], a particular use case of hybrid model is explained where dynamic analysis is used during the training phase and static analysis is used in the scoring phase. Another form of a hybrid model is to use static analysis to inspect network packet data while using machine learning to monitor network traffic to pick up malicious network communications. In [9], the authors utilized a hybrid a malware analysis model to extend the work in [2]. A sniffing detection method that uses network traffic probed with machine learning techniques is proposed. According to the authors, this paper was the first to apply machine learning for the purpose of Sniffer detection. The detection method in the paper used ICMP and HTTP for traffic probing. In addition, features like CPU load and variable period lengths were used for performance evaluation. This extended paper achieved a comparable outcome with the best results obtained in [2]. The work in [12] proposes a botnet detection approach using mining of network flow characteristics. Given a network flow dataset, four features were extracted: the ratio of incoming packets, the ratio of outgoing packets, original packet length, and the ratio of bot-response packets. These features were used by the naïve bayesian classifier to achieve 99% accuracy and 96.9% F-measure performances. However, the authors concluded that the four features were insufficient to accurately represent botnet communication patterns and additional features needed to be identified to further improve the accuracy of the model. In [19], the authors utilized 29 different features of various network protocols and its payload data to detect botnet activity in a network. Using a large number of features, the main focus of this paper was to establish a connection between a host's periodic communication patterns and botnet activity. The author pointed out that since botnets inevitably produce periodic network traffic while communicating with the C&C servers, this characteristic will be a strong indicator of botnet infestation. The datasets used in [19] are network captures that consist of only malware and botnets traffic. Autocorrelation analysis on the features extracted from the dataset was processed and the authors concluded the paper by presenting autocorrelation plots that show signs of periodic behavior of botnet traffic. The work in [19] opens the possibility to utilize the trait of periodic behavior in botnet traffic to detect infected hosts. However, the dataset used in [19] is limited to only botnet traffic where normal and background traffic are absent. The question of whether the same approach will work for real-world network traffic dataset is yet to be answered. Our paper extends the work of [19] by incorporating periodicity as one of the features used to train the deep learning model for botnet traffic classification. In contrast with the previous research, our work uses a network flow dataset that includes a mixture of botnet, normal, and background traffic to prove its efficacy in real-world botnet attack.

4 Dataset

The CTU-13 dataset was collected in 2011 by researchers at CTU University in Czech Republic for the purpose of generating a large capture of botnet traffic mixed with both normal and background traffic captures [6]. The thirteen captures that comprise CTU-13, also referred to as scenarios, are collected using seven different real botnet samples. While the dataset is now aging, it is not less representative of modern botnet attacks. For example, the Virut botnet was identified recently after being considered eradicated for many years [4]. A complete description of the seven botnet samples is provided in Table 1. The distinguishing feature of the CTU-13 dataset is that each packet has been manually examined and labeled as either botnet, normal, or background traffic. From examining the percentage of botnet traffic in all thirteen scenarios, botnet traffic makes up only a small percentage of overall traffic. This imbalance, however, accurately simulates real-world botnet infection and will be used as input without artificial manipulation for the purpose of this research.

For each scenario, the three types of network traffic were captured in a Packet Capture (pcap) file. From processing each pcap file, information such as NetFlows and WebLogs were also obtained. While pcap files carry detailed information,

Table 1 Types of CTU-13 botnet samples and its description

Botnet Type	Scenario Nr.	Description
Neris	1, 2, 9	Neris uses HTTP-based communication with the C&C servers. The infected botnets' main activities include click-fraud and distribution of SPAM emails
Rbot	3, 4, 10, 11	Rbot uses IRC-based communication with the C&C servers. Common with most IRC type malwares, the botnet is controlled by the botmaster through a pre-configured IRC server
Virut	5, 13	Virut uses HTTP-based communication with its C&C servers. Main activities of the infected hosts perform distribution of SPAM emails and unauthorized file downloads
Menti	6	Menti uses IRC-based communication with its C&C servers to scan SMTP servers
Sogou	7	Sogou uses unencrypted HTTP-based communication to connect with the C&C servers. Its malicious activities include downloading binary files and compressing them without authorization
Murlo	8	Murlo uses IRC-based communication with the C&C servers to carry out orders such as downloading executable files and scanning vulnerable local network ports
NSIS.ay	12	NSIS.ay uses P2P-based communication with the C&C

analyzing NetFlow files is the primary interest of our paper as it contains core information about traffic as well as its class labels.

4.1 CTU-13 Dataset Features

In a bi-directional NetFlow dataset, fifteen categories, listed in Table 2, are used to describe a network traffic. The dataset is initially sorted by StartTime in ascending order and, by using software programs that support csv parsing like Excel, each category can be filtered to selectively show rows of particular interest. For instance, to search for traffic generated by botnet activities, filtering the Label category for "Botnet" keyword would bring up all relevant rows.

The CTU-13 dataset has been distributed by Stratosphere Lab through their website and is open to the public for research or educational purposes [13].

5 Proposed Methodology

The goal of this project is to detect botnet activity in a given network by examining the incoming and outgoing network traffic data. As illustrated in Fig. 5, the proposed

Table 2 Explanation of features that comprise the CTU-13 network flow dataset

StartTime	StartTime represents the absolute timestamp in which the row has been recorded and is formatted as hh:mm:ss.
Dur	Dur is duration of the corresponding event in seconds for each row
Proto	There are 15 protocol is categorized in the Proto feature: ‘tcp’, ‘udp’, ‘rdp’, ‘rtp’, ‘pim’, ‘icmp’, ‘ipx/spx’, ‘arp’, ‘igmp’, ‘rarp’, ‘unas’, ‘udt’, ‘esp’, ‘ipv6’, ‘ipv6-icmp’
SrcAddr	Source IP address in ipv4 format
Sport	Port number at the Source
Dir	Direction of the network flow, represented as ‘->’, ‘<-’, ‘<->’, ‘<?>’, ‘*who*’, ‘<?’, ‘?>’
DstAddr	Destination IP address in ipv4 format
Dport	Port number at the destination
State	This feature describes the transaction state according to the protocol and has 230 unique values
sTos	Source Type of Service (0,1,2,...,192, NaN)
dTos	Destination Type of Service (0,1,2,...,192, NaN)
TotPkts	Total number of packets transmitted
TotBytes	Total number of bytes transmitted
SrcBtytes	Number of bytes transmitted from source to destination
Label	Three unique labels to describe the transaction (normal, background, botnet)

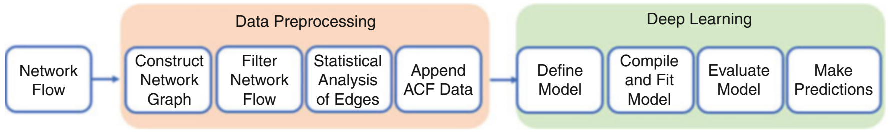


Fig. 5 Overview of the proposed botnet detection mechanism

implementation consists of two essential phases. The first is the data processing phase where the network flow records are rearranged and filtered so that only essential information is left behind for the second deep learning phase. During the first phase, a network graph which consists of nodes and edges is created to show the interconnections between hosts. The final output of the first phase is an array that stores the communication statistics of each edge in the graph, autocorrelation, as well as the label. The last phase is the deep learning stage where a deep learning model is defined, compiled, and fitted to be able to predict botnet activity in a network.

5.1 Data Preprocessing Phase

Infected botnets need to regularly connect with C&C servers to provide status updates and receive new orders. Due to this unique characteristic, the communication log between the botnets and the servers inevitably exhibits periodic patterns that can be used to signal signs of botnet activity. For instance, if there is a host in a local network connecting to an outer network host every n seconds, then this may be evidence that the local network host is an infected botnet that is sending out heartbeat signals to the C&C server. To detect signs of periodicity more efficiently, the original network flow dataset will be filtered to remove excess data.

5.1.1 Filtering Network Flow

Table 3 shows an example of an unaltered CTU-13 network flow record. Initially, there are 15 features that help describe each network transaction between a source and a destination. The explanation for each feature is described in Table 2. Among the 15 features, only 10 are of primary importance, namely, StartTime, Dur, Proto, SrcAddr, DstAddr, State, TotPkts, TotBytes, SrcBytes, and Label. According to previous research on botnets [24], the most frequent protocols used between a botnet and its C&C server are TCP, UDP, HTTP, and ICMP. The communication states that are important for these protocols are CON, URP, and FSPA_FPSA. The state CON indicates Connected in UDP, URP as Urgent Pointer in UDP and FSPA encompassing all flags (FIN, SYN, PUSH, ACK) in TCP. From the original CTU-13 dataset, the rows without Proto as UDP, TCP, HTTP, or ICMP will be filtered out, and, of the remaining rows, only those with connection state CON, URP, or FSPA_FPSA will be kept. This process of removing irrelevant transactions will significantly increase overall compute accuracy and efficiency as well as reduce computational costs.

In this paper, all 13 network flows from the CTU-13 dataset have been used as input, and they all followed the filtering process described in this Section. Table 4 shows an example of the filtered output of the loaded CTU-13 dataset.

5.1.2 Constructing Network Graph

The network graph, also known as network diagram, is useful for understanding the network's physical and logical connection status. It enables viewers to have a visual representation of the network to gain an overall picture of network topology and data flow. By reassembling the result of the previous section's rows of network log into a network graph, the networking nodes and executed transactions will be easily viewable for analysis. A proper network graph would follow a similar topology shown in Fig. 6, where nodes would represent host machines and edges would preserve records of all transactions between two nodes.

Table 3 Example of original CTU-13 network flow records

Protocol	SrcAddr	Dir	DstAddr	State	TotPkts	TotBytes	SrcBytes	Label
udp	147.32.84.165	<->	147.32.80.9	CON	2	203	64	From-Botnet-V46-UDP-DNS
tcp	1.114.187.73	->	147.32.84.229	FSPAC_FSPA	22	1772	951	Background-TCP-Established
udp	147.32.84.13	<->	82.208.56.89	CON	96	8640	4320	To-Normal-V46-UDP-NTP-server

Table 4 Example of a filtered network flow dataset

Proto	SrcAddr	DstAddr	State	TotPkts	TotBytes	SrcBytes	Label
udp	151.51.231.119	147.32.86.44	CON	32,055	8,577,197	8,575,244	Background-UDP-Established
udp	147.32.84.229	66.56.30.27	CON	2	567	95	Background-UDP-Established
udp	147.32.84.229	187.126.122.175	CON	2	567	95	Background-UDP-Established
udp	147.32.84.229	81.234.199.128	CON	2	567	95	Background-UDP-Established
udp	86.52.158.60	147.32.84.229	CON	13	999	422	Background-UDP-Established
…	…	…	…	…	…	…	…

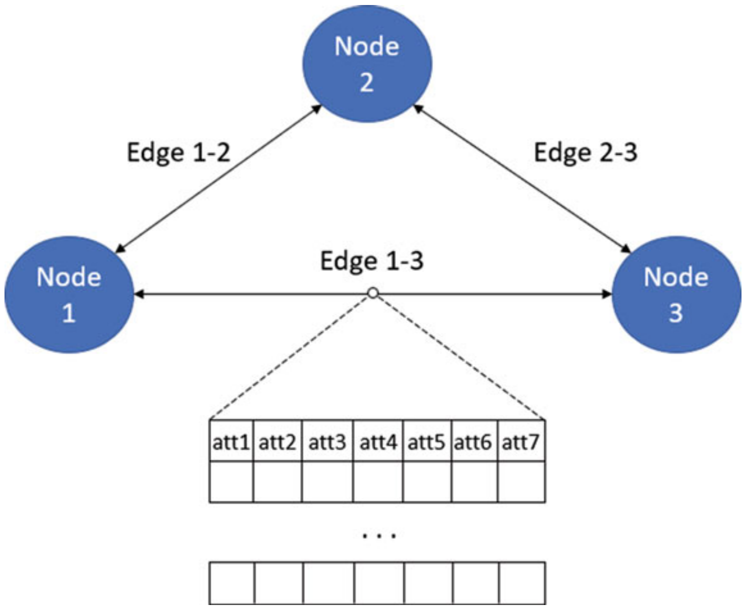


Fig. 6 Basic network graph architecture that consist of nodes and edges

Algorithm 1 describes the pseudo-code for constructing a network graph from a network flow dataset. In essence, the algorithm iterates through an array of network flow records and creates an non-repetitive node in the network graph by looking at the source and destination of the record. Once the nodes exist, the transaction information described in the row is also recorded in the edge between the two nodes on the network graph. The algorithm terminates when all rows of the array have been processed and returns the completed network graph.

Figure 7 is a visualized network graph that clearly shows the connection between each communicating node. Even though this graph was constructed with only a small subset of the filtered CTU-13 dataset to reduce visualization complexity, it still resembles the network graph that would result when the filtered CTU-13 dataset were fully used. The features that are stored in the edges would consist of protocol, duration, total bytes, total packets, state, timestamp, and label.

5.1.3 Statistical Analysis of Edges

The procedures prior to this section has been to remove irrelevant information from the original dataset and rearrange existing information into a graph structure to efficiently analyze data. In this section, a new array is created to store the statistics computed from information recorded in the edges. To remove unsubstantial transactions, only the edges that have more than four rows will be used. Each row

Algorithm 1: Pseudo-code for constructing network graph

```
   Input: Network flow data stored in an array (Array)
   Output: Network Graph (G) consisting of nodes and edges
1  for each row in Array do
2      Node1 = row.SourceIP
3      Node2 = row.DestinationIP
4      if Node1 not in Graph G then
5          add Node1 to G
6      if Node2 not in Graph G then
7          add Node2 to G
8      if edge does not exist between Node1 and Node2 then
9          create a new edge between Node1 and Node2
10     append row's flow attributes to the edge between Node1 and Node2
11 return G
```

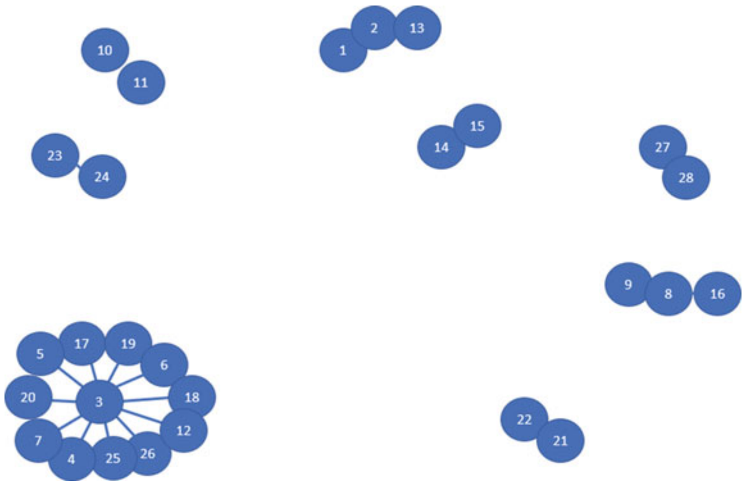


Fig. 7 Example of a network graph plotted from the filtered CTU-13 dataset

of the new array corresponds to a comprehensive summary of transactions between two nodes. Every row consists of twenty six columns: Duration (6), Total Bytes (6), Total Packets (6), Timestamp (6), ACF (1), and Label (1). In particular, the columns that belong to duration, total bytes, total packets, and timestamp are each filled with six statistics, namely, mean, median, standard deviation, minimum, maximum, and range. These numbers are calculated from the information stored at an edge.

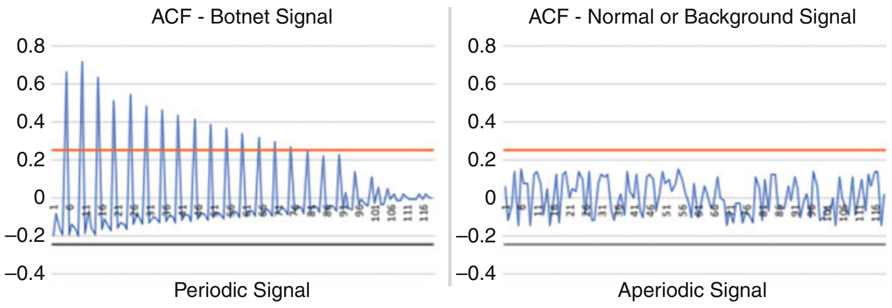


Fig. 8 ACF plot for periodic and aperiodic signals

5.1.4 Autocorrelation Analysis

A key goal of this project is to leverage the periodic communication behavior of the C&C network to detect botnet activities. To achieve this goal, the autocorrelation function (ACF) is used to calculate the periodicity of transactions. To explain with an example, Fig. 8 shows the autocorrelation plot for both periodic and aperiodic signals. In the periodic signal, the autocorrelation value peaked above the upper bound of the confidence interval 15 times. The autocorrelation plot for the aperiodic signal did not see any peaks that passed the confidence interval. In this scenario, the input used for classification would be 15 for periodic signals and 0 for aperiodic signals. A relatively high value of count indicates a strong periodic signal, which would also imply occurrence of cyclic botnet activities.

The final output of the data preprocessing phase should be a single array N which consists of 25 features (X) and a label (y) for each row. To train a deep learning model for classification, this array N will be horizontally split (column-wise) into two arrays: features $N[0:25]$ and labels $(N[25:])$.

5.2 *Deep Learning Phase*

In machine learning, an input dataset should initially be divided into two categories, that is, training and testing. This separation procedure is important to prevent the model from overfitting while accurately evaluating the model performance [28]. However, before randomly sampling the data into two datasets, a critical characteristic of the original CTU-13 dataset needs to be considered. In fact, the CTU-13 dataset has a highly imbalanced botnet to non-botnet network traffic ratio. As imbalance classification may lead to a biased and misleading deep learning model, using random sampling to divide the dataset is not considered to be an appropriate technique [14].

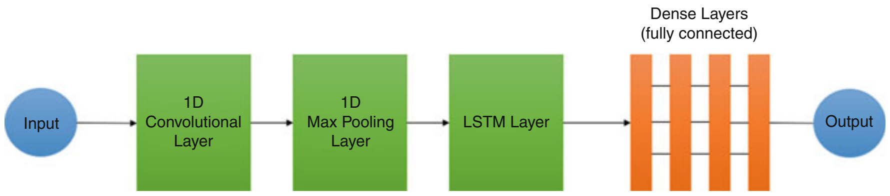


Fig. 9 Overview of deep neural network architecture

5.2.1 Stratified K-Fold Cross Validation

Among various sampling methods, the Stratified K-fold technique [14] performs well with imbalanced datasets. When the sampler divides the dataset into training and testing sets, the percentage of labels that constitutes the original dataset is maintained. For instance, if the botnet to non-botnet label in the original dataset is $1 : 20$, both training and testing sets would also keep the same ratio after being split. This ensures botnet traffic to be properly represented while the model is being trained and tested with minimum sampling error or bias.

Both testing and training sets compete to have the maximum number of samples to achieve best learning results. The best validation result would come from having more samples in the test set. However, this inherently triggers a trade-off of having less items in the training set. A solution to this dilemma is to use cross validation technique [15]. In K-fold cross validation, a dataset is partitioned into K sets of equal size and K separate learning experiments are executed. For each learning experiment, a non-repeating partition is selected as a test set while the remaining $K - 1$ partitions are used as train sets. Once all K learning experiments are complete, the performances and test results are averaged. For this project, $K(3)$-fold cross validation, supported by the scikit learn API, was used for a more accurate assessment of the learning model.

5.2.2 Define, Compile, and Fit the Neural Network

The main components of the neural network are the convolution layer, max pooling layer, LSTM layer, and fully-connected layers. In the first convolution layer, a convolution kernel takes a training dataset and extracts hidden features and establishes a relationship between the input dataset and generated features. After the one dimensional convolution layer, max pooling is applied to reduce dimensions in the data to reduce computation overhead. LSTM layer takes the output from the max pooling layer and enables sequential connection among the dataset before feeding it to the dense layers. Finally, the output layer performs classification prediction to label botnet traffic. An overview of the proposed deep network is illustrated in Fig. 9.

5.2.3 Model Evaluation

To evaluate the proposed model, metrics like overall accuracy, precision, recall, and F-measure will be considered in conjunction with four additional performance metrics: true positive (TP), true negative (TN), false positive (FP), and false negative (FN).

In this paper, the botnet traffic will be considered as a positive label and non-botnet traffic will have a negative label. The overall accuracy refers to the number of correctly predicted labels over the total number of samples. Precision is the proportion of true positives over the sum of all positive labels. Recall is the proportion of true positives in the number of all the correctly labeled samples. F1 measure is the weighted mean of precision and recall, with its values ranging from zero to one.

6 Results

The proposed approach was implemented using deep learning models in the Keras Python library with TensorFlow deep learning engine. The implementation involved a combination of $1D$ convolutional network, max pooling, LSTM, and fully-connected layers. A more detailed configuration of these models is provided in Table 6. The order and combination of these layers was selected through multiple rounds of testing various configurations to optimize the performance of the classification result.

With the layers setting in Table 6, the deep neural network repeatedly trained and validated CTU13 input data separated by the type of malware. A total of seven different types of malware, each unique with its own communication pattern, were tested. The performance metrics of each malware type is shown in Table 5. According to the result, the proposed model performed best to accurately classify botnet traffic with the Rbot and Murlo malware type, both achieving over 0.9985 accuracy for the test dataset. While the performance results of different malware types have low variance, the reason for Rbot and Murlo having high detection

Table 5 Classification result of the deep neural network implementation

Malware type	CTU13 scenario #	Accuracy—train	Accuracy—test	Precision	Recall	F1-score
Neris	1, 2, 9	1	0.9976	0.985	0.986	0.9855
Rbot	3, 4, 10, 11	1	0.9985	0.999	0.988	0.9935
Virut	5, 13	1	0.9974	0.982	0.981	0.9815
Menti	6	0.99	0.9747	0.986	0.985	0.9855
Sogou	7	0.98	0.9981	0.982	0.987	0.9845
Murlo	8	0.98	0.9987	0.998	0.981	0.9894
NSIS.ay	12	0.99	0.9905	0.997	0.985	0.9910

Table 6 Classification result of the deep neural network implementation

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 19,64)	256
max_pooling1d_1 (MaxPooling1)	(None, 9,64)	0
lstm_1 (LSTM)	(None, 32)	12,416
dense_4 (Dense)	(None, 64)	2112
dense_5 (Dense)	(None, 64)	4160
dense_6 (Dense)	(None, 64)	4160
dense_7 (Dense)	(None, 1)	65

Table 7 Comparison of performance metrics with respect to other studies

Research paper	Method	Features	Dataset	Performance metrics
Torres et al. [25]	Recurrent neural network	Size, duration, periodicity	CTU-13	Accuracy: 0.970
				False positive rate: 0.0372
Wang et al. [27]	Social communication detection	Network flow-based	CTU-13	Recall: 0.026
				Precision: 0.80
				Fl-score: 0.088
Chen et al. [20]	Decision tree	Network flow-based	CTU-13	Accuracy: 0.936
				False positive rate: 0.3
Nagarajan [19]	Periodicity in network flow	Periodicity in pcap data	CTU-13	Fl-score: 0.09
Vishwakarma [26]	Data balancing and machine learning techniques	Network flow-based	CTU-13	Accuracy: 0.98
This paper 2021	Pattern-based network flow feature extraction	Statistical network flow-based	CTU-13	Accuracy: 0.9936
				Precision: 0.9898
				Recall:0.9847
				Fl-score: 0.9872

accuracy may be due to availability of a larger input data. The trend in Table 5 shows that the malware type with a larger number of network flows achieved relatively higher performance results. This implies that, with the increase of the number of network flows, the model will perform even better.

The performance results of this paper are compared to those of relevant research papers in Table 7. Using a pattern-based approach and analyzing the features statistically, this paper recorded 0.9936 accuracy, 0.9898 precision, 0.9847 recall, and 0.9872 F1-score. Compared to the results taken from [19, 20, 25, 27], and [26], the proposed approach achieved overall high performance. This comparison result shows that a pattern-based approach enables high detection accuracy while maintaining low false positive rate.

7 Conclusions

In this paper, a novel botnet detection approach is proposed using a pattern-based classification technique. The approach begins by filtering the input network flow to focus on traffic that uses TCP, UDP, and ICMP protocols. Information presented in the filtered network flow is rearranged to enable an intuitive understanding of the network traffic. By leveraging the network graph, features like duration, total bytes exchanged, total packets, timestamp and autocorrelation count were extracted. This approach can be used for all types of botnet architectures and does not require any prior knowledge about the botnet type or C&C server IP address. The proposed approach has been tested with network flow datasets that consist of botnet, normal, and background traffic, to show that detecting botnet traffic in real-life scenarios is possible. A deep neural network was designed to process the statistical features that have been extracted from the network graph. Using the CNN architecture, a classifier for botnet traffic has been created and the statistical features were fed to the model for training and testing. The performance results were compared to the metrics found in relevant research papers to confirm that the proposed approach outperformed those of previous works. The presented method is applicable to various types of botnet families to identify malicious actors in a real-life network environment with high accuracy. For future work, a realistic networked environment can be recreated to simulate real-life implementation. In this case, we expect a considerable increase in background noise, however, we believe that the technique proposed in this paper can still obtain promising results. Other machine learning techniques can be tested relying on our selected features. For instance, Hidden Markov models (HMM) [23], profile hidden Markov models [5], and support vector machines [1] can obtain interesting results in this particular scenario.

References

1. Robert Berwick. An idiots guide to support vector machines (svms). http://web.mit.edu/6.034/wwwbob/svm.pdf, 2003. [Online; accessed August 2021].
2. Krzysztof Cabaj, Marcin Gregorczyk, Wojciech Mazurczyk, Piotr Nowakowski, and Piotr undefinedórawski. Sniffing detection within the network: Revisiting existing and proposing novel approaches. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ARES '19, New York, NY, USA, 2019. Association for Computing Machinery.

3. Anusha Damodaran, Fabio Di Troia, Corrado Visaggio, Thomas Austin, and Mark Stamp. A comparison of static, dynamic, and hybrid analysis for malware detection. *J Comput Virol Hack Tech*, 13:1–12, 2017.
4. NHS Digital. Virut botnet. https://digital.nhs.uk/cyber-alerts/2018/cc-2829, 2020.
5. Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. Biological sequence analysis: probabilistic models of proteins and nucleic acids, 1998.
6. S. García, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *Comput. Secur.*, 45:100–123, September 2014.
7. Ibrahim Ghafir and Vaclav Prenosil. Blacklist-based malicious ip traffic detection. In *2015 Global Conference on Communication Technologies (GCCT)*, pages 229–233, 2015.
8. Ibrahim Ghafir, Vaclav Prenosil, Mohammad Hammoudeh, Thar Baker, Sohail Jabbar, Shehzad Khalid, and Sardar Jaf. Botdet: A system for real time botnet command and control traffic detection. *IEEE Access*, 6:38947–38958, 2018.
9. Marcin Gregorczyk, Piotr Żórawski, Piotr Nowakowski, Krzysztof Cabaj, and Wojciech Mazurczyk. Sniffing detection based on network traffic probing and machine learning. *IEEE Access*, 8:149255–149269, 2020.
10. Nabil Hachem, Yosra Ben Mustapha, Gustavo Gonzalez Granadillo, and Herve Debar. Botnets: Lifecycle and taxonomy.
11. Box-Steffensmeier Janet, Freeman John, Hitt Matthew, and Pevehouse Jon. *Time Series Analysis for the Social Sciences*. Cambridge University Press, New York, 2014.
12. G. Kirubavathi and R. Anitha. Botnet detection via mining of traffic flow characteristics. *Computers & Electrical Engineering*, 50:91–101, 2016.
13. Stratosphere Lab. The CTU-13 Dataset. https://www.stratosphereips.org/datasets-ctu13/. [Online; accessed August 2021].
14. Victoria López, Alberto Fernández, and Francisco Herrera. On the importance of the validation technique for classification with imbalanced datasets: Addressing covariate shift when data is skewed. *Inf. Sci.*, 257:1–13, February 2014.
15. M. Lorbach, E.I. Kyriakou, R. Poppe, E.A. van Dam, L.P.J.J. Noldus, and R.C. Veltkamp. Learning to recognize rat social behavior: Novel dataset and cross-dataset application. *Journal of neuroscience methods*, 300:166–172, 2018.
16. Pavan Roy Marupally and Vamsi Paruchuri. Comparative analysis and evaluation of botnet command and control models. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 82–89, 2010.
17. H. S. Nair and V. Ewards. A study on botnet detection techniques. *Mathematical Problems in Engineering*, 2, 2012.
18. Emmanuel C. Ogu, Olusegun A. Ojesanmi, Oludele Awodele, and ‘Shade Kuyoro. A botnets circumspection: The current threat landscape, and what we know so far. *Information*, 10(11), 2019.
19. Nagarajan Prathiba, Di Troia Fabio, Austin Thomas, and Stamp Mark. Autocorrelation analysis of financial botnet traffic. In *2nd International Workshop on Formal Methods for Security Engineering (ForSE 2018), in conjunction with the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*, ICISSP 2018, 2018.
20. Chen Ruidong, Niu Weina, Zhang Xiaosong, Zhuo Zhongliu, and Lv Fengmao. An effective conversation-based botnet detection method. *Mathematical Problems in Engineering*, 2017:166–172, 2017.
21. Rami Sihwail, K. Omar, and K. A. Z. Ariffin. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *International Journal on Advanced Science, Engineering and Information Technology*, 8:1662–1671, 2018.
22. SéRgio S. C. Silva, Rodrigo M. P. Silva, Raquel C. G. Pinto, and Ronaldo M. Salles. Botnets: A survey. *Comput. Netw.*, 57(2):378–403, February 2013.
23. Mark Stamp. A revealing introduction to hidden markov models. https://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf, 2021. [Online; accessed August 2021].
24. Manoj Rameshchandra Thakur, Divye Raj Khilnani, Kushagra Gupta, Sandeep Jain, Vineet Agarwal, Suneeta Sane, Sugata Sanyal, and Prabhakar S. Dhekne. Detection and prevention of

botnets and malware in an enterprise network. *Int. J. Wire. Mob. Comput.*, 5(2):144–153, May 2012.
25. Pablo Torres, Carlos Catania, Sebastian Garcia, and Carlos Garcia Garino. An analysis of recurrent neural networks for botnet detection behavior. In *2016 IEEE Biennial Congress of Argentina (ARGENCON)*, pages 1–6, 2016.
26. Anand Ravindra Vishwakarma. Network traffic based botnet detection using machine learning, master's project, 2020.
27. Jing Wang and Ioannis Ch. Paschalidis. Botnet detection based on anomaly and community detection. *IEEE Transactions on Control of Network Systems*, 4(2):392–404, 2017.
28. Suleiman Y. Yerima, Mohammed K. Alzaylaee, Annette Shajan, and Vinod P. Deep learning techniques for android botnet detection. *Electronics*, 10(4), 2021.
29. Xing Ying, Shu Hui, Zhao Hao, Li Dannong, and Guo Li. Survey on botnet detection techniques: Classification, methods, and evaluation. *Mathematical Problems in Engineering*, 2021, 2021.