CALIFORNIA STATE UNIVERSITY SAN MARCOS

THESIS SIGNATURE PAGE

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

THESIS TITLE: Botnet Classification using Convolutional Neural Networks

AUTHOR: Dhatri Sai Kumar Reddy Gudipelly

DATE OF SUCCESSFUL DEFENSE: 05/03/2023

THE THESIS HAS BEEN ACCEPTED BY THE THESIS COMMITTEE IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN
COMPUTER SCIENCE.

Nahid Ebrahimi Majd	*Nahid Ebrahimi Majd*	05/05/2023
THESIS COMMITTEE CHAIR	SIGNATURE	DATE
Sreedevi Gutta	*Sreedevi Gutta*	05/07/2023
THESIS COMMITTEE MEMBER	SIGNATURE	DATE
Name of Committee Member		
THESIS COMMITTEE MEMBER	SIGNATURE	DATE
Name of Committee Member		
THESIS COMMITTEE MEMBER	SIGNATURE	DATE

Botnet Classification using Convolutional Neural Networks

Abstract:

A "botnet" is a type of malware that allows hackers to directly manage a large number of infected systems to perform damaging actions such as spamming, Distributed Denial of Service (DDoS) attacks, and data theft. Botnets must be identified and classified in order to keep computer networks secure. This research investigates the use of convolutional neural networks (CNNs) for botnet classification, and we assess the performance of several CNN models on a set of annotated network traffic samples. We used the N-BaIoT Dataset for this study, which contains data from various forms of Mirai and BashLite assaults. These approaches employ multiclass classification using models like 1D-CNN and CNN-LSTM with 91% accuracy for both the models. Most of the studies use different models for different devices we use one model for all the devices.

Keywords: botnets, convolutional neural networks, machine learning, network traffic, cyber security

Table of Contents

1. Introduction

1.1 Problem statement

Botnets are networks of misbehaving computers or devices controlled by a single person, typically a cybercriminal or hacker, without the owners' knowledge or consent [1],[2]. These infected computers or devices are referred to as "bots" or "zombies," and they are usually infested with malware such as Trojans, viruses, or worms. Once a device has been infected and connected to a botnet, the botnet operator can remotely manage it to do a variety of harmful actions such as launching DDoS attacks,

sending spam emails, distributing malware, stealing sensitive information, and mining cryptocurrencies. The operator who manages the botnets is known as the botmaster. These are often generated by a process known as "botnet herding," in which the botnet operator infects and takes control of a large number of computers or devices through various approaches such as social engineering, phishing, and exploiting vulnerabilities. Once the botnet has been built, the operator can remotely command and manage the bots to perform a variety of malicious tasks. Thus, Botnets are a significant threat to the security and stability of the internet, and they can cause significant harm to individuals, organizations, and entire industries. Here are some of the Botnets attacks in real-time:

WannaCry Ransomware: This attack was carried out using the EternalBlue exploit on vulnerable Windows systems. To conduct the attacks a system called EternalBlue Exploits was used. This system was primarily used by National Security Agency(NSA) to perform security operations [3].

VPNFilter Botnet: The VPNFilter botnet, which was discovered in 2018, infected a lot of routers and network-attached storage(NAS) devices. The botnet was capable of stealing sensitive information, launching DDoS attacks, and even destroying infected devices [4].

Trickbot Trojan: A highly modular botnet that first came into existence in 2016 was primarily used for banking fraud, such as stealing online banking credentials and conducting fraudulent wire transfers [5].

Mirai Botnet: This Botnet was discovered in 2016 and is responsible for the largest DDoS attacks. It infected hundreds of thousands of IoT devices, such as routers, IP cameras, and DVRs, and used them to launch massive DDoS attacks that disrupted large portions of the internet [6].

Bashlite Botnet: This Botnet was discovered in 2015 and exploited a flaw in the bash shell used to launch distributed denial of service attacks [7,8].

Emotet Trojan: This is a very sophisticated botnet that was discovered in 2014, and is used for distributing other types of malware, such as banking trojans and ransomware.

1.2 Purpose of the Study and Motivation:

Botnets are a significant threat to the security and stability of the internet, so the development of effective techniques to handle them is of utmost importance This includes identifying them and mitigating them. In recent times they have been many studies on various machine-learning techniques that could be used in the detection of Botnets. But, after detecting the botnets we need to identify which class does it belongs to, know its history of origin, and what it could have done. For, this we propose a Classification system using convolutional neural networks and compare them with other machine-learning techniques. Traditional botnet detection methods frequently rely on signature-based approaches, which are limited to detecting only known botnets. A CNN-based categorization system, on the other hand, has the ability to detect new and previously unknown botnets by studying their traffic patterns, even if they do not match any known signatures.

1.3 Research Methodology:

To prepare a model that assesses whether there is a botnet in a network or not. The requirement is to have a dataset that has information about botnets. Then, a model needs to be made by using the botnet dataset by partitioning it into training, validation, and testing. The training set is used to train the model whereas the validation set is used to get the best-fit model for the dataset and for results, we use the testing dataset. The models are made using the research strategy of Convolutional Neural Networks a technique in deep learning algorithms that are vastly used for object detection, image classification, and so on. Once the model is made, verifying with other machine learning models is done using the metrics like accuracy, precision, recall, and F1 score.

1.4 Research Scope

This research is used to propose a method that can be effective in detecting botnet attacks better. For mitigating them on the internet. There are a lot of machine learning techniques that have been successful in detecting botnets. This is a procedure of making a model that is very rarely used in network traffic.

2. Related Work:

In this section, we mention some of the previous works related to botnet detections like binary classification and multi-class classification. There are many studies that exist for botnet detection using various datasets. But very few use the approach of Deep learning models in classifying the attacks. In [9] the researchers examined the AutoEncoder method with the softmax classifier on the KDD99 dataset and showed that the AutoEncoder method had an accuracy of 94.7% in detection. In [10] the authors used the Stacked NSAE method and the random forest classifier to classify the KDD99 and NSL_KDD datasets and their accuracies were 97.8% and 85.42% respectively. In [2], the researchers used the deep auto-encoder network on the N-BaIoT dataset, which is an IoT-specific dataset. They showed that with the AutoEncoder method, they reached TPR = 100% and FPR = 0.007 which is the lowest number of false alarms, and also the execution time of the algorithm was lower than other compared methods. In 11], the authors examined several deep neural networks, including CNN, RNN, and LSTM models. These are tested on the N-BaIoT dataset. Their experimental results show that the CNN method has an accuracy of 91%, the RNN method has an accuracy of 41%, and the LSTM method has an

accuracy of 62%. Whereas [12] was another research on botnet detection using Autoencoders having FPR and TPRs as their metrics. In [13] they use a botnet detection method using a negative selection algorithm and a combo of a CNN-LSTM model on the ISOT and ISCX datasets. The deep learning model had an accuracy of 99% on the detection taking place. In [14] they have built an unsupervised swarm intelligent system called Grey Wolf Optimization Algorithm that can detect the anomalies in the network and report using the hyperparameters of OCSVM (One Class Support Vector Machine). There is this specific model on [15] a kitsune NIDS model which accepts the traffic data and detects whether the incoming traffic is benign or attacked. The model was running so fast that it didn't require much memory to run and was easily able to run on Raspberry Pi devices. A CNN model, an LSTM model and a RNN model were trained and tested on the N-BaIoT dataset in [16]. It compares some of the ML techniques with the DL technique results.

3. Methodology:

3.1 Dataset:

N-BaIoT (Network-based IoT Botnet Attack Dataset)[15] is a 2019 freely available dataset of network-based IoT botnet attacks. The collection contains several attack events, including both benign and malicious network traffic generated by IoT devices. The dataset was gathered from a large-scale IoT network environment over a three-month period and contains various types of IoT devices such as cameras, routers, and sensors. N-BaIoT is a great resource for IoT security researchers and practitioners since it provides a realistic and diversified dataset for creating and testing IoT security solutions. The dataset can be used for a variety of applications, including the development of machine learning algorithms for botnet detection, the analysis of IoT botnet attack characteristics, and the evaluation of the effectiveness of existing security solutions. The inclusion of both benign and malicious network traffic in the N-BaIoT dataset is one of its important advantages, allowing for the creation of more robust and effective detection models. Furthermore, the dataset contains precise information on network traffic, such as source and destination IP addresses, port numbers, and protocols utilized, making it easier to assess the features of the assaults. The N-BaIoT dataset has 115 features and the attributes are as follows:

1. Stream aggregation:
    a. H: Stats summarizing the recent traffic from this packet's host (IP)
    b. HH: Stats summarizing the recent traffic going from this packet's host (IP) to the packet's destination host.
    c. HpHp: Stats summarizing the recent traffic going from this packet's host+port (IP) to the packet's destination host+port. Example 192.168.4.2:1242 -> 192.168.4.12:80
    d. HH_jit: Stats summarizing the jitter of the traffic going from this packet's host (IP) to the packet's destination host.
2. Time-frame (The decay factor Lambda used in the damped window): How much recent history of the stream is capture in these statistics, including L5, L3, L1, L0.1, L0.01
3. The statistics extracted from the packet stream:
    a. weight: The weight of the stream (can be viewed as the number of items observed in recent history)
    b. mean: The average of the stream

c. std: Standard deviation
d. radius: The root squared sum of the two streams' variances
e. magnitude: The root squared sum of the two streams' means
f. cov: an approximated covariance between two streams
g. pcc: an approximated covariance between two streams

The dataset has 10 classes of attacks with 1 class as ‘benign’. In this research, we consider the data of the doorbell sensors captured because the data here was balanced when considered in other case scenarios and most of the other’s models had been having good TPR rates. There are also some studies like [12,14,17] where different models were made on the dataset and we can compare them with our results.

Table 1: Dataset Attack Schema

Botnet	**Attack Type**	**Doorbell**	**Thermostat**	**Baby Monitor**	**Webcam**	**Security Camera 1**	**Security Camera 2**
Benign		88648	13113	175240	52150	160668	66113
BASHLITE	COMBO	112732	53012	58152	58669	118910	113681
	JUNK	58865	30312	28349	28305	59966	55992
	SCAN	57969	27494	27859	27698	57694	56397
	TCP	193677	95021	92581	97783	193897	186891
	UDP	209807	104791	105782	110617	208669	206700
MIRAI	ACK	102195	113285	91123	0	118551	218667
	SCAN	107685	43192	103621	0	193877	89604
	SYN	122573	116807	118128	0	127597	248194
	UDP	237665	151481	217034	0	314856	308963
	Plain UDP	81982	87368	80808	0	110466	162680
Total		1373798	835876	1098677	375222	1665151	1713882

Table 1. above illustrates the schema and the number of rows that are present to apply our models. As we have already mentioned, this is the real-time a captured from all the sensors as mentioned in [12].

3.2 Data Preprocessing:

Usually, Data Preprocessing is a step we use for making data consistent throughout the training, validation, and testing phase. We cannot make models on inconsistent data having values like null, or missing values and infinity values. To remove all these kinds of inconsistencies we follow a step called Data Cleaning. The aim of this step is to clean the data and make it consistent and readable. Most of the deep learning techniques like the data to be in numerical values for effective model creation. To this we follow a step called Data Transformation, in this step various methods like data scaling, and data encoding take place. For our research, we use Standard Scaler and Label Encoding.

The StandardScaler is a prominent data preparation tool in machine learning for scaling and standardizing dataset properties. It converts the features to have a zero mean and unit variance, which is required by many machine learning techniques. The procedure is removing each feature's mean from its values and then dividing it by its standard deviation. This produces a normalized dataset with a mean of zero and a standard deviation of one for each characteristic. When dealing with characteristics that have varying scales, the standard scaler can bring all features to a similar scale, making it easier for the machine learning algorithm to learn from them. It can also aid in the reduction of the impact of outliers and the convergence of various optimization algorithms. Many machine learning libraries, notably sci-kit learn in Python, including the StandardScaler. It is simple to apply and may be used on both training and test data by utilizing the fit_transform method. It is crucial to note, however, that the StandardScaler assumes that the data is normally distributed and may not be appropriate for non-Gaussian data. Other scaling algorithms, such as MinMaxScaler or RobustScaler, may be more appropriate in such instances.

Label encoding is a data preprocessing technique used to turn categorical data into numerical data that machine learning algorithms can handle. It is a straightforward and efficient method of encoding categorical variables, with each category allocated a distinct integer value. In label encoding, each distinct category is assigned a distinct integer value ranging from 0 to n-1, where n is the number of distinct categories in the variable. For example, if we have a categorical variable "color" with three categories (red, green, blue), label encoding would assign these categories the values (0, 1, 2), respectively. When working with ordinal categorical data having a natural order or hierarchy among the categories, label encoding is an effective strategy. However, it may not be appropriate for nominal categorical variables with no natural order of the categories.

Feature Selection is an important step that could help in dimensionality reduction and performance of the model that we make. Some of the most important techniques of The next step in data preprocessing would be dealing with redundant data. This also helps in the performance of the model. The more quality of the data the better the model. Let us see Fig.1, This is the correlation heatmap of all the features divided into sets of 20 for better evaluation of features. From this, we can say that most of the related features are something around 85 features which have a correlation threshold of more than 0.9. So, we remove all the correlated features and then select the remaining features for making the models. Here are some of the features that are highly correlated and dropped before making the model.

'HH_L0.01_magnitude','HH_L0.01_mean', 'HH_L0.1_magnitude', 'HH_L0.1_mean',
'HH_L0.1_radius','HH_L0.1_weight' 'HH_L1_covariance', 'HH_L1_magnitude',
'HH_L1_mean','HH_L1_radius', 'HH_L1_std', 'HH_L1_weight',
'HH_L3_covariance','HH_L3_magnitude', 'HH_L3_mean', 'HH_L3_pcc',
'HH_L3_radius','HH_L3_std', 'HH_L3_weight', 'HH_L5_magnitude',
'HH_jit_L0.01_mean','HH_jit_L0.01_variance', 'HH_jit_L0.01_weight', 'HH_jit_L0.1_mean',
'HH_jit_L0.1_weight','HH_jit_L1_mean', 'HH_jit_L1_weight', 'HH_jit_L3_mean',
'HH_jit_L3_variance','HH_jit_L3_weight', 'HH_jit_L5_weight','H_L0.01_mean',
'H_L0.01_variance','H_L0.01_weight', 'H_L0.1_mean', 'H_L0.1_variance',
'H_L0.1_weight','H_L1_mean', 'H_L1_variance', 'H_L1_weight', 'H_L3_mean',
'H_L3_variance','H_L3_weight', 'H_L5_mean', 'H_L5_variance',
'H_L5_weight','HpHp_L0.01_covariance', 'HpHp_L0.01_magnitude', 'HpHp_L0.01_mean',
'HpHp_L0.01_radius','HpHp_L0.01_std', 'HpHp_L0.01_weight', 'HpHp_L0.1_covariance',
'HpHp_L0.1_magnitude','HpHp_L0.1_mean', 'HpHp_L0.1_radius', 'HpHp_L0.1_std',
'HpHp_L0.1_weight','HpHp_L1_covariance', 'HpHp_L1_magnitude', 'HpHp_L1_mean',
'HpHp_L1_pcc','HpHp_L1_radius', 'HpHp_L1_std', 'HpHp_L1_weight',
'HpHp_L3_covariance','HpHp_L3_magnitude', 'HpHp_L3_mean', 'HpHp_L3_pcc',
'HpHp_L3_radius','HpHp_L3_std', 'HpHp_L3_weight', 'HpHp_L5_magnitude',
'HpHp_L5_mean','MI_dir_L0.01_mean', 'MI_dir_L0.01_variance', 'MI_dir_L0.1_mean',
'MI_dir_L0.1_variance','MI_dir_L0.1_weight', 'MI_dir_L1_mean', 'MI_dir_L1_variance',
'MI_dir_L1_weight','MI_dir_L3_mean', 'MI_dir_L3_variance','MI_dir_L3_weight'

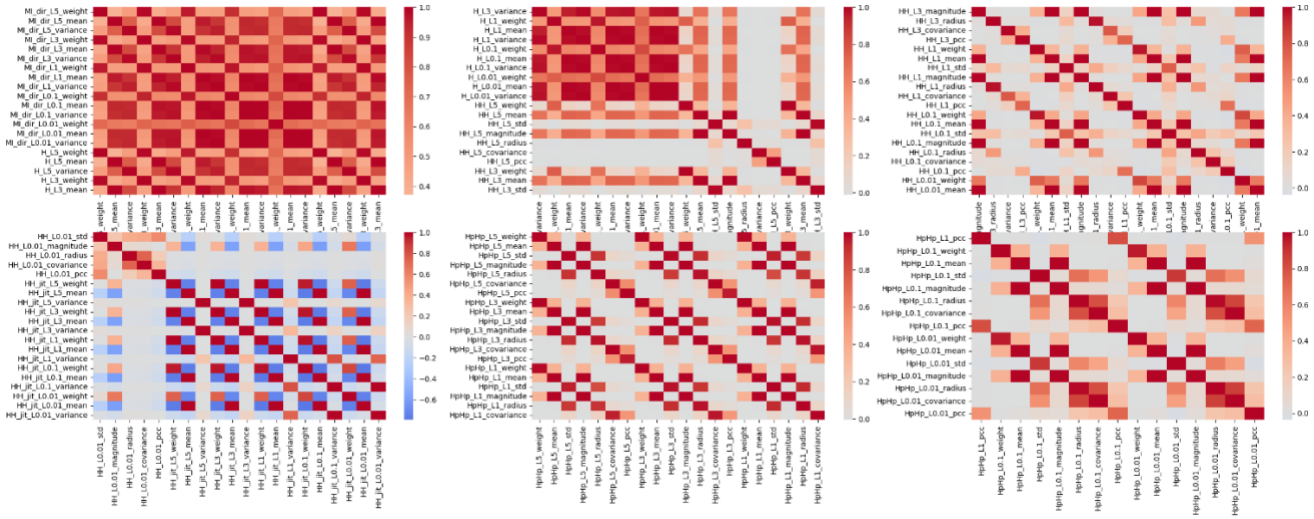


Fig 1. Features Correlation Heatmap

Fig. 2 shows the feature importance of all the features that are left after removing the correlated features. According to the scores of the method SelectKBest from scikit learn.

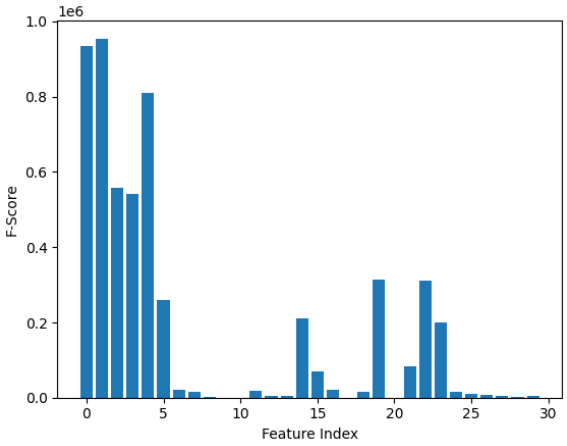


Fig 2. Feature Scores for Non-Correlated Features.

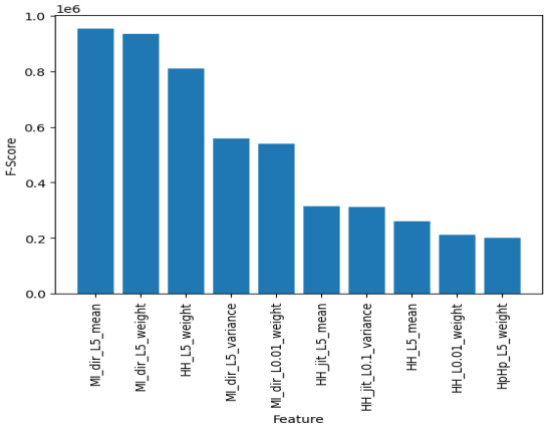


Fig 3. Top 10 features of the dataset.[Text Wrapping Break]

Yet to avoid underfitting we are considering all the 30 non-correlated features to make our models and evaluate them. The best 10 features are shown in the fig 3. as follows.

Table 2. Top 10 Features of the Dataset

Feature	***Importance***
MI_dir_L5_mean	954381.5949
MI_dir_L5_weight	933833.6769
HH_L5_weight	810192.5151
MI_dir_L5_variance	557012.233
MI_dir_L0.01_weight	540210.3354
HH_jit_L5_mean	314979.1933
HH_jit_L0.1_variance	312342.3368
HH_L5_mean	259424.909
HH_L0.01_weight	210585.8751
HpHp_L5_weight	199429.9612

Table 2. Illustrates the top ten features which were calculated by the SelectKBest method and their importance values respectively.

In this dataset, “H” refers the source host IP and “MI” refers to the source MAC and IP. Other features are “HH”, which refers to the source to destination IP; “HH_jit”, which refers to the source to destination jitter; and “HpHp”, which refers to the source to destination port. The “mean” refers to the mean packet size in the stream. The “L” refers to the time frame of the stream. The number after “L” is the decay factor Lambda used in the damped window. For example, L5 indicates a short time frame and L0.01 indicates a long time frame.

The top 10 features show that the “H_L0.01_mean” impacts the model the most. This feature refers to the stream by its source host IP and holds the mean packet size for a long time frame history of the stream. We observe that the top 10 features are either “H” or “MI”, which means the features that refer to the stream by its source host IP or source MAC and IP have higher

impact than the features that refer to the stream by its source to destination IP, jitter, or port. Also, we observe that the mean packet size for a long time frame history (L0.01) has a higher impact than a short history (L0.1 or L1). We also observe that mean packet size has higher impact than variance.

3.3 Proposed Approach:

We propose the method of classifying the IoT botnet attacks using some of the convolutional neural networks for the doorbell device, we are training the model on statistical features that are extracted from the doorbell files. Fig. 4 provides a better understanding of what our aim is.

Table 4. History of Different Models on different datasets

Algorithm	**Standard Dataset**			**Original Dataset**							**N-BaIoT**		
Reference		[18]	[10]	[19]	[11]	[20]	[13]	[21]	[22]	[23]	[12]	[17]	[14]
ML	Random Forest									x		x	
	Decision Tree										x	x	
	BLSTM							x					
	MLP	x											
	Naïve Bayes		x	x	x								
	KNN		x		x							x	
	LDA		x		x								
	ANN		x		x								
	SVM				x								
	OCSVM												x
	IF												x
	LOF												x
DL	DLANN				x		x	x					
	LSTM	x	x	x								x	
	deep autoencoder												x
	Dense RNN												
	RNN					x		x	x				

Algorithm	Standard Dataset			Original Dataset							N-BaIoT		
Reference		[18]	[10]	[19]	[11]	[20]	[13]	[21]	[22]	[23]	[12]	[17]	[14]
	CNN											x	

As we see in the Table 4, above we can illustrate that, The standard Datasets are the datasets that are used for intrusion detection like KDD Cup, NSL-KDD etc., these datasets are used mostly on all kinds of devices. The Original Dataset is where the authors have made their own data and test their models on the data. Thus, we can say that, they are very few CNNs that have been designed on the dataset so far. Our model has a botnet dataset, botnet training models, and botnet detection models. The botnet dataset consists of four sub-datasets of N-BaIoT. We select devices that include all 10 attack samples as described in the table. We use the Ennio doorbell for our research. As botnet training models, we use most widely used ML and DL algorithms. We employed not only 3 types of Ml models (Naïve Bayes, Decision tree and Random Forest) but also two types of DL models (CNN, and a CNN-LSTM model). Our models are used for multi class classification. The multi-class classification deals all kinds of attacks that are in the dataset with benign cases.

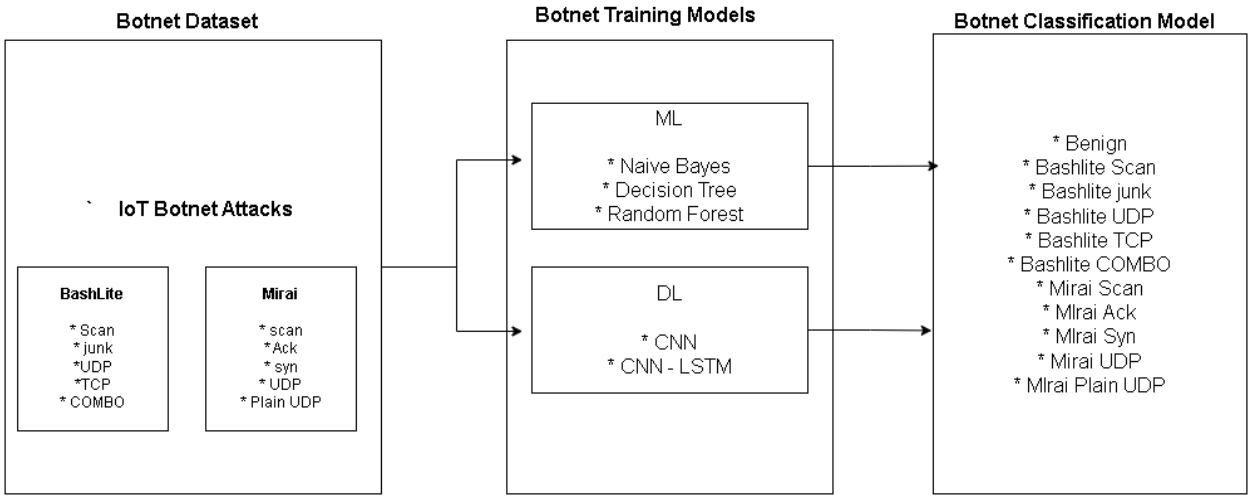


Fig 4. Proposed Model

In order to create a matrix of characteristics, including packet size, orientation, and payload content, we preprocess the incoming data. varying CNN designs, including varying convolutional and pooling layer combinations, followed by one or more fully linked layers, are the subject of our experiments. Additionally, we investigate the application of transfer learning to pre-trained CNN models for malware categorization. The Proposed system pipeline is as follows in the figure. In the phase of Data Collection, we select the data the model is to be made on as we have selected N-BaIoT dataset where the data was collected from the doorbell sensors.

The next step is data preprocessing where we transform, standardize and perform various operations on data such that it helps in making effective models. So, to transform the data we use techniques like Data Scaling and Data Encoding.

Data Scaling: Data scaling refers to the process of transforming the numerical values of a dataset to a similar scale to improve the performance of machine learning models. Machine learning algorithms often rely on numerical values, and if these values are not scaled, the algorithms may not be able to make accurate predictions. There are several methods of scaling data, including: a)Min-Max Scaling: This method scales the data so that the minimum value is 0 and the maximum value is 1. b) Standardization: This method scales the data so that it has a mean of 0 and a standard deviation of 1. c) Log Transformation: This method applies the natural logarithm function to the data to reduce the range of values and make the distribution more normal. d) Robust Scaling: This method scales the data using the interquartile range (IQR) to make the data less sensitive to outliers. The choice of scaling method depends on the characteristics of the dataset and the requirements of the machine learning algorithm being used. It is important to choose an appropriate scaling method to avoid introducing bias or noise in the data.

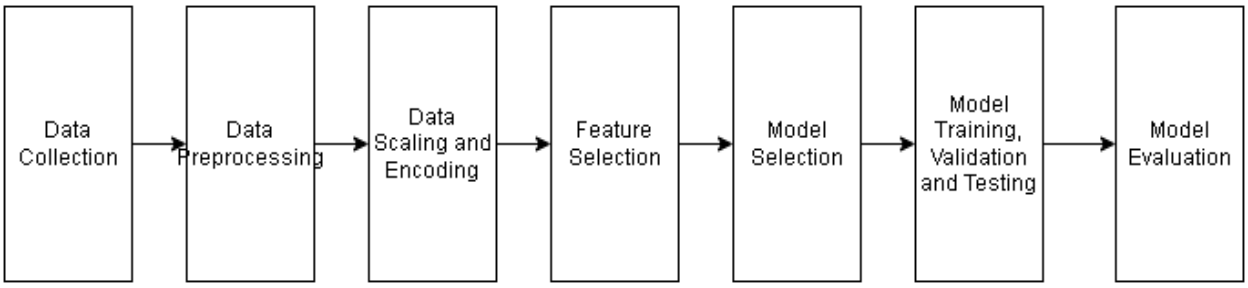
Data Encoding: Data encoding is the process of transforming categorical data into a numerical format that can be used as input to a machine learning algorithm. Categorical data represents information about the qualities or attributes of an object or entity, such as color, shape, or category, and is often represented as text labels. There are different techniques for encoding categorical data, including: a) Label encoding: In this technique, each unique category is assigned a numerical value. For example, if there are three categories, they could be encoded as 0, 1, and 2. This technique can be used for ordinal data, where there is a natural ordering to the categories. B) One-hot encoding: In this technique, a binary vector is used to represent each category. Each vector has a length equal to the number of categories, and the value 1 is placed in the position corresponding to the category, while all other positions are 0. For example, if there are three categories, the first category could be represented as [1, 0, 0], the second as [0, 1, 0], and the third as [0, 0, 1]. This technique is useful for nominal data, where there is no natural ordering to the categories. C)Binary encoding: This technique is similar to one-hot encoding, but uses fewer dimensions by representing each category as a binary number. For example, if there are three categories, they could be encoded as 00, 01, and 10. D) Count encoding: In this technique, each category is assigned a numerical value based on its frequency in the dataset. For example, a category that appears frequently might be assigned a higher value than one that appears less frequently. E) Target encoding: In this technique, each category is assigned a numerical value based on the target variable. For example, if the target variable is a binary classification, the mean of the target variable for each category can be used as the numerical value. This technique can be useful when there is a strong relationship between the categorical variable and the target variable. The choice of encoding technique depends on the nature of the data and the machine learning algorithm being used. It is important to choose an encoding technique that preserves the information in the data and does not introduce bias or artifacts that could affect the performance of the algorithm.

Table 5. The classes are encoded as follows

Class label	Encoded Label
benign	0
gafgyt combo	1
gafgyt junk	2
gafgyt scan	3
gafgyt tcp	4
gafgyt udp	5
mirai ack	6
mirai scan	7
mirai syn	8
mirai udp	9
mirai udpplain	10

The labels having ‘gafgyt’ are attacks made by Bashlite Botnet where ‘mirai’ is by Mirai Botnet.

\



Full Text:
Data Collection pointing to Data Prepocessing pointing to Data Scaling and Encoding pointing to Feature Selection pointing to Model Selection pointing to Model Training, Validationa nd Testing pointing to Model Evaluation

Fig 5. Process of achieving Proposed System.

Fig 5. above gives us some visual clarity on how we are achieving our goal. After Data preprocessing and training the data with the dataset and then testing, there is one step that we use to get the best model it is called Model Validation with a validation dataset. For the validation step, we use cross_val_score from sklearn and then select the best model using the randomized grid search method on parameters.

3.4 Machine Learning and Deep Learning Techniques:

The Various Machine Learning and Deep Learning Techniques that are used in our research are as follows: (1) Bernoulli Naïve Bayes, (2) Decision Tree Classifier, (3) Random Forest Classifier

3.4.1 Machine Learning Models

Bernoulli Naïve Bayes: Bernoulli Naive Bayes is generally intended for binary classification issues, however, it can be modified for multiclass classification by employing a "one-vs-all" strategy. The algorithm in this approach creates a distinct binary classifier for each class, with the positive class consisting of instances from that class and the negative class consisting of instances from all other classes combined. Based on the training data, the algorithm calculates the probability of each feature being present in each class during training. The method identifies traits that have a higher likelihood of being present in that class than in all other classes combined for each class. These characteristics are then utilized to construct a binary classifier for that class. During prediction, the method uses the associated binary classifier to compute the likelihood of each data point belonging to each class. As the predicted class, the algorithm selects the class with the highest likelihood.

Decision Tree Classifier: A Decision Tree is a supervised machine-learning technique that is commonly used to solve categorization problems. The algorithm builds a tree-like model of decisions and their potential outcomes, which is then used to classify fresh data points. The tree is built by recursively separating the data based on feature values so that the resulting data subsets have similar values to the target variable (i.e., the class label). Based on some criterion, such as the Gini index or information gain, the algorithm chooses the feature that gives the best split. The method finds the feature that gives the optimal split at each node of the tree and divides the data into two or more subsets depending on the potential values of that feature. The operation is repeated until the subsets at each node are homogeneous in terms of the target variable, or until some stopping requirement (such as a maximum depth or a minimum number of samples) is fulfilled. The procedure begins at the root node of the tree and follows the branches based on the values of the features in the new data point until it reaches a leaf node, which corresponds to a predicted class label.

Extra Tree Classifier: This is a type of ensemble learning technique that is based on decision tree algorithms. The key idea is to randomize the construction of decision trees, in order to reduce overfitting and improve accuracy. Extra trees also create a forest of decision trees, but the splitting thresholds at each node are selected randomly for being based on the gini index and information gain. Specifically, for each feature at each node, the algorithm selects a random threshold within the range of that feature and chooses the threshold that provides the best split.

Random Forest Classifier: This is a type of ensemble learning algorithm that is also based on the decision tree algorithms. The key idea behind a random forest is to construct a forest of decision trees, where each tree is constructed using a random subset of the training data and a random subset of the features. The algorithm works as follows: 1. Random subset of the training data is selected with the replacement, 2. Each tree in the model is trained by a random subset of features., 3. We construct multiple trees using the above steps, 3. During prediction, the algorithm aggregates the predictions of all decision trees in the forest, using a majority voting scheme for classification problems.

Convolutional Neural Networks:

A Convolutional Neural Network is a type of deep learning technique which are highly used for image and video processing applications. This is a class of feedforward AI that is it goes only in one direction from input to output layer only. As the CNNs are designed to specifically work with grid like structures. We can also use them for network traffic as they are large in number and can take a lot of time to assess for other algorithms.

LSTM is an abbreviation for Long Short-Term Memory. It is a recurrent neural network architecture that can detect long-term dependencies and learning patterns in sequential data. LSTMs have a memory cell that can store information for a long time and control the flow of information into and out of the cell via input, forget, and output gates. These gates enable the network to selectively forget or remember data from past time steps, making them ideal for applications such as speech recognition, language modeling, and time series prediction.

Model 1:

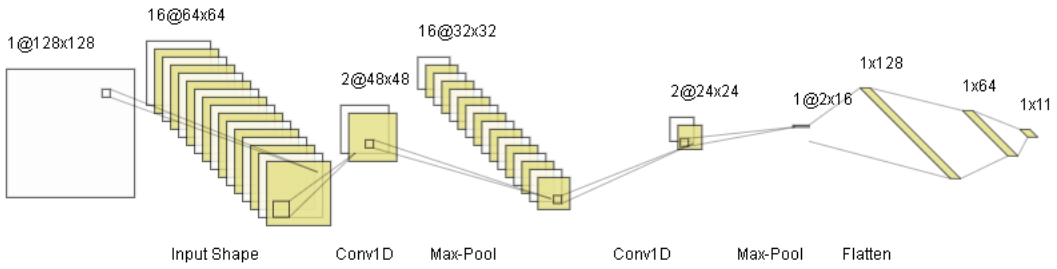


Fig 6. CNN Model 1

The model takes input in the form of 1D time-series data, and the output is a prediction of the class label. The first convolutional layer has 6 filters and a kernel size of 5, and the activation function used is ReLU. This is followed by a max pooling layer with pool size 2.The second convolutional layer has 16 filters and a kernel size of 5, again with ReLU activation, followed by another max pooling layer with pool size 2. The output from the convolutional layers is then flattened to a 1D vector, which is fed into the first fully connected layer with 128 neurons and ReLU activation. This is followed by another fully connected layer with 64 neurons and ReLU activation. Finally, the output layer has num_classes neurons with SoftMax activation, which produces the probability distribution over the class labels.

Model 2:

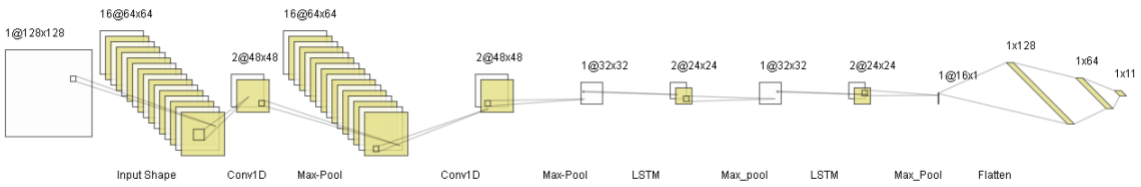


Fig 7. CNN Model 2

The model takes as input 1D sequences of data (with shape (X_train_cnn.shape[1], 1)) and consists of the following layers: A 1D convolutional layer with 64 filters, a kernel size of 5, and a ReLU activation function. A max pooling layer with a pool size of 2. Another 1D convolutional layer with 32 filters, a kernel size of 5, and a ReLU activation function. Another max pooling layer with a pool size of 2. An LSTM layer with 32 units and a ReLU activation function, which returns sequences of vectors. Another max pooling layer with a pool size of 2. Another LSTM layer with 16 units, which also returns sequences of vectors. Another max pooling layer with a pool size of 2. A flatten layer to convert the output from the LSTM layers into a 1D vector. Two fully connected (dense) layers with 128 and 64 units respectively, both with ReLU activation functions. A final dense layer with a SoftMax activation function, which produces the output probabilities for each class. This architecture combines both convolutional and recurrent layers to process the input sequences and learn features that are relevant for the classification task. The CNN layers perform local feature extraction, while the LSTM layers capture temporal dependencies between the features. The max pooling layers down sample the feature maps to reduce the dimensionality of the data and avoid overfitting. The final dense layer produces the output probabilities for each class based on the learned features.

3.4.2 System requirements:

All the models are made in colab pro an online data analysis and machine learning tool that allows executable python code. An NVIDIA A100 GPU is used. A CPU with 83.5 GB is been allotted for each model in the study. With an additional RAM of 40GB on GPU and a memory of 166.8GB on the Disk. The CPU speed in Colab Pro version is 2.30GHz. More information on colab has been given in [25].

Chapter 4: Analysis of Results

4.1 Evaluation Metrics:

We employ a collection of annotated network traffic examples, including both benign and botnet family-specific traffic. We create training, validation, and test groups from the information. We use the validation set to validate our models and the training set to fine-tune their hyperparameters. Through the use of measures like accuracy, and loss score, we assess the success of our models. We contrast the performance of our CNN models with that of SVMs and RFs, two other machine-learning techniques frequently employed for classifying botnets. For all of the methods, we utilize the same dataset and preprocessing procedures. In order to improve each algorithm's efficiency, we also test out various hyperparameters.

The Dataset we selected has been widely used in research on botnet detection and classification, using many machine-learning models like decision trees, random forests, and neural networks. So, this thesis will be comparing some machine learning models with the new CNN models. The evaluation metrics that are going to be used are precision, recall and F1-score for the machine learning models and f1- score for the CNN models. Usage of classification reports and confusion matrix are done to represent the efficiency of the models made.

Confusion Matrix: Confusion Matrix is a matrix of actual vs. predicted values for a set of data, where each row represents the instances of a particular class, and each column represents the instances predicted to belong to that class. Cij is equal to the number of observations known to be in class ’i’ and predicted to be in class ’j’. Confusion matrix whose i-th row and j-th column entry indicates the number of samples with true label being i-th class and predicted label being j-th class. Table 1 shows our confusion matrix, where for class A

1. TP = True Positive, An instance of class A was correctly predicted to be in class A.

2. FP = False Positive, An instance of another class was incorrectly predicted to be in class A.

3. FN = False Negative, An instance of class A was incorrectly predicted to be in another class.

4. TN = True Negative, An instance of another class was correctly predicted to be in its own class.

Similarly, the other metrics that we use to measure the performance:

1. Precision: Precision for class A measures the proportion of instances that were correctly classified to be in class A among the total number of instances that were classified to be in class A, either correctly or incorrectly. It is measured by equation 1.

$$Precision = \frac{TP}{TP+FP} \quad (1)$$

2. Recall: Recall for class A measures the proportion of instances that were correctly classified to be in class A among the total number of instances that were in class A, either correctly or incorrectly classified. It is measured by equation 2.

$$Recall = \frac{TP}{TP+FN} \tag{2}$$

3. F1 score: F1 score for class A measures the 'Harmonic mean' of precision and recall for class A. It is an overall measure of the quality of a classifier's predictions. It is usually the metric of choice because it captures both precision and recall. It is measured by equation 3.

$$F1\ score = \frac{2\times(precision \times recall)}{precision + recall} \tag{3}$$

If we express it in terms of True Positive (TP), False Positive (FP), and False Negative (FN), we get equation 4.

$$F1\ score = \frac{TP}{TP + \frac{1}{2}(FP+FN)} \tag{4}$$

The above three metrics evaluate how well our model is predicting each class in the dataset. However, they do not give us a single performance indicator that allows us to compare a model against other models. The metrics introduced below give us such metric.

4. Global Accuracy: Global Accuracy measures the proportion of instances that were correctly classified among the total number of predictions. It is measured by equation 5.

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \tag{5}$$

5. Micro-Averaged F1 Score: We can compute global precision and recall scores from the sum of FP, FN, TP, and TN counts across classes. Then we can use these global precision and recall scores to compute a global F1 score as their Harmonic Mean. This F1 score is known as the micro-averaged F1 score.

$$Micro-Averaged\ F1\ score = \frac{2\times(global\ precision \times global\ recall)}{global\ precision + global\ recall} \tag{6}$$

Micro averaging computes a global average F1 score by counting the sums of the True Positives (TP), False Negatives (FN), and False Positives (FP). We first sum the respective TP, FP, and FN values across all classes and then plug them into the F1 equation to get our micro F1 score.

$$Micro-Averaged\ F1\ score = \frac{global\ TP}{global\ TP + \frac{1}{2}(global\ FP+global\ FN)} \tag{7}$$

This is a good metric for a balanced dataset, where there are almost equal number of instances in each class. However, if the dataset is highly imbalanced, this metric is heavily influenced by abundant classes in the dataset. If the classifier performs very well on majority classes (with high number of instances) and poorly on minority classes (with low number of instances), this metric will still be high.

6. Macro-Averaged F1 Score: The macro-averaged F1 score (aka macro F1 score or macro avg) is computed using the arithmetic mean (aka unweighted mean) of all the per-class F1 scores. This method treats all classes equally regardless of their support values.

$$Macro - Averaged\ F1\ score = \frac{\sum_{i=1}^{n}(F1\ score\ of\ class\ i)}{n} \quad (8)$$

If the dataset is highly imbalanced, macro-averaging is to be preferred over micro-averaging because it weighs each of the classes equally and is not influenced by the number of examples of each class.

7. Weighted-Averaged F1 score: The weighted-averaged F1 score (aka weighted average) is calculated by taking the mean of all per-class F1 scores while considering each class's support. Support refers to the number of actual instances of the class in the dataset. The 'weight' refers to the proportion of each class's support relative to the sum of all support values.

$$Weighted - Averaged\ F1\ score = \frac{\sum_{i=1}^{n}(F1\ score\ of\ class\ i * Support\ of\ class\ i)}{n} \quad (9)$$

8. Micro-Averaged Precision: is calculated as precision of global values.

$$Micro - Averaged\ Precision = \frac{global\ TP}{global\ TP + global\ FP} \quad (10)$$

9. Macro-Averaged Precision: is calculated as an average pf precisions of all classes.

$$Macro - Averaged\ Precision = \frac{\sum_{i=1}^{n}(Precision\ of\ class\ i)}{n} \quad (11)$$

10. Weighted-Averaged Precision: is also calculated based on Precision per class but takes into account the number of samples of each class in the data as well.

$$Weighted - Averaged\ Precision = \frac{\sum_{i=1}^{n}(Precision\ of\ class\ i * Support\ of\ class\ i)}{n} \quad (12)$$

11. Micro-Averaged Recall: is calculated as recall of global values.

$$Micro - Averaged\ Precision = \frac{global\ TP}{global\ TP + global\ FN} \quad (13)$$

12. Macro-Averaged Recall: is calculated as an average pf recalls of all classes.

$$Macro - Averaged\ Recall = \frac{\sum_{i=1}^{n}(Recall\ of\ class\ i)}{n} \quad (14)$$

13. Weighted-Averaged Recall: is also calculated based on Recall per class but takes into account the number of of each class in the data as well.

$$Weighted - Averaged\ Recall = \frac{\sum_{i=1}^{n}(Recall\ of\ class\ i * Support\ of\ class\ i)}{n} \quad (15)$$

14. Loss Function: A loss function is a function that compares the true and the predicted output values and shows how good the neural network models are. There are many kinds of loss functions like (1) binary cross-entropy, (2) categorical cross-entropy, Mean Absolute error, and Mean Squared error. As we are dealing with CNNs and multi-class classification Categorical cross-entropy would be feasible.

$$\text{CE loss} = -{}^{1}/_{n} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \cdot \log(p_i)_j \quad (16)$$

15. Fowlkes_Mallows_score: This is said to be a geometric mean between precision and recall. It measues the similarity of two clustering of a set of points.

$$FMS = TP/(\sqrt{(TP + FP)(TP + FN)})$$

4.2 Results:

Machine Learning Results:

Let us dive into the results of the ML methods. The table shows the hyperparameters of the models used.

Table 5. List Of Parameters used to make the models

Model	Hyper parameters
Naïve Bayes	'alpha': 1.0, 'binarize': 0.0, 'class_prior': None, 'fit_prior': True, 'force_alpha': 'warn'
Decision Tree	'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_state': 100, 'splitter': 'best'
Extra Tree	'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'sqrt', 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_state': 100, 'splitter': 'random'
Random Forest	'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': 100, 'verbose': 0, 'warm_start': False
1d-CNN	Kernel_size = 5, activation = 'relu', pool_size = 2,optimizer = 'adam', learning_rate = 0.001
CNN+LSTM	Kernel_size = 5, activation = 'relu', pool_size = 2,optimizer = 'adam', learning_rate = 0.001

Table 5. gives us the parameters we have selected using hyperparameter tuning with respect to validation scores for all the mentioned algorithms above.

Table 6. Results of the models we performed.

Model Type	Model	Attacks	Precision	Recall	F1-Score	Support
ML	NB	Benign	0.96	0.83	0.89	51117
		Bashlite Combo	0.44	0.64	0.53	51065
		Bashlite junk	0.62	0.73	0.67	51035
		Bashlite Scan	0.77	0.64	0.7	51026
		Bashlite tcp	0.37	1	0.54	50780
		Bashlite udp	0	0	0	51230
		Mirai ack	0.55	0.48	0.51	51143

Model Type	Model	Attacks	Precision	Recall	F1-Score	Support
		mirai scan	0.62	0.56	0.59	51002
		mirai syn	0.71	0.04	0.08	51015
		mirai udp	0.46	0.54	0.5	51008
		mirai udpplain	0.82	0.76	0.79	50824
	Dtree	Benign	1	1	1	51117
		Bashlite Combo	1	1	1	51065
		Bashlite junk	1	1	1	51035
		Bashlite Scan	1	1	1	51026
		Bashlite tcp	0.5	1	0.66	50780
		Bashlite udp	0.93	0	0	51230
		Mirai ack	1	1	1	51143
		mirai scan	1	1	1	51002
		mirai syn	1	1	1	51015
		mirai udp	1	1	1	51008
		mirai udpplain	1	1	1	50824
	RF	Benign	1	1	1	51117
		Bashlite Combo	1	1	1	51065
		Bashlite junk	1	1	1	51035
		Bashlite Scan	1	1	1	51026
		Bashlite tcp	0.5	1	0.66	50780
		Bashlite udp	0.98	0	0	51230
		Mirai ack	1	1	1	51143
		mirai scan	1	1	1	51002
		mirai syn	1	1	1	51015
		mirai udp	1	1	1	51008
		mirai udpplain	1	1	1	50824
	Extra tree	Benign	1	1	1	51117

Model Type	Model	Attacks	Precision	Recall	F1-Score	Support
		Bashlite Combo	1	1	1	51065
		Bashlite junk	1	1	1	51035
		Bashlite Scan	1	1	1	51026
		Bashlite tcp	0.84	1	0.91	50780
		Bashlite udp	1	0.81	0.89	51230
		Mirai ack	1	1	1	51143
		mirai scan	1	1	1	51002
		mirai syn	1	1	1	51015
		mirai udp	1	1	1	51008
		mirai udpplain	1	1	1	50824
DL	1D-CNN	Benign	1	1	1	51117
		Bashlite Combo	0.93	0.78	0.85	51065
		Bashlite junk	0.81	0.94	0.87	51035
		Bashlite Scan	1	1	1	51026
		Bashlite tcp	0.5	1	0.66	50780
		Bashlite udp	0.73	0	0	51230
		Mirai ack	0.99	0.99	0.99	51143
		mirai scan	0.98	1	0.99	51002
		mirai syn	1	0.99	0.99	51015
		mirai udp	0.96	0.91	0.93	51008
		mirai udpplain	0.91	0.96	0.93	50824
	CNN-LSTM	Benign	1	1	1	51117
		Bashlite Combo	0.93	0.87	0.90	51065
		Bashlite junk	0.88	0.94	0.91	51035

Model Type	Model	Attacks	Precision	Recall	F1-Score	Support
		Bashlite Scan	1	1	1	51026
		Bashlite tcp	0.5	1	0.66	50780
		Bashlite udp	0.7	0	0	51230
		Mirai ack	1	1	1	51143
		mirai scan	1	1	1	51002
		mirai syn	1	1	1	51015
		mirai udp	0.96	0.93	0.94	51008
		mirai udpplain	0.93	0.96	0.95	50824

Table 6. illustrates the performance of all the models we have performed and seeing the results we can say that most of the models have performed good on all attacks other than the Bashlite TCP. Most of the Bashlite TCP have been predicted as Bashlite UDP which has caused discrepancies in the model. But, there is one model in machine learning that we have performed the Extra tree ML Classifier has a 100% positives on all kinds of attacks that have been predicted. The DL classifier also failed in classifying BASHLITE TCP and BASHLITE SCAN. In the rest of the cases, they have performed well. The Naïve Bayes Model was not good for the dataset we have selected with reporting the least of 57% accuracy, whereas the highest would-be extra tree with a 98% and the next would be the Random Forest and Decision Tree Classifiers with 91% and follows CNN models having 88% and 87%.accuracies.

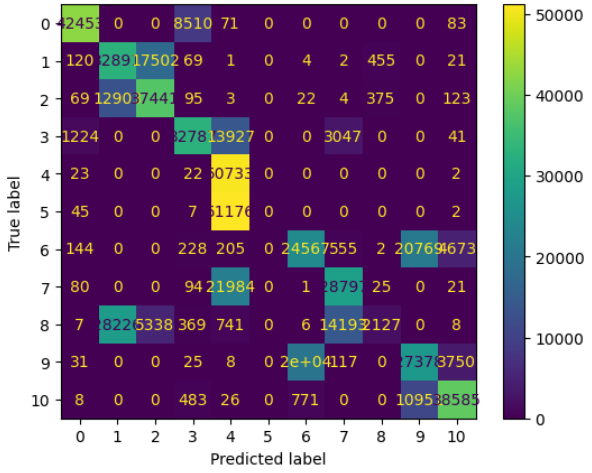


Fig 8. Confusion Matrix for Naïve Bayes

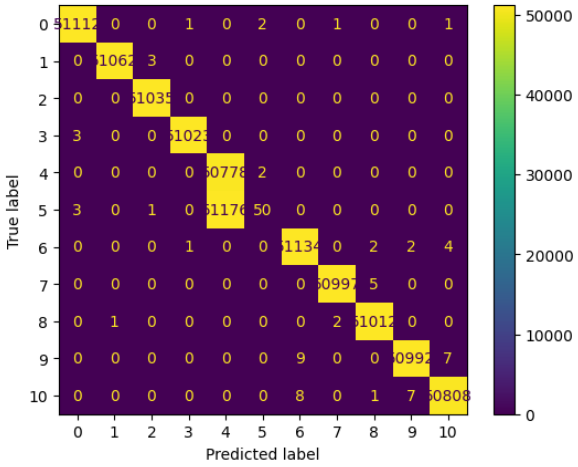
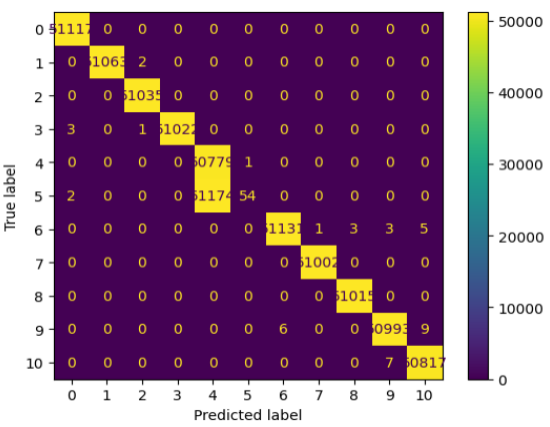


Fig. 9 Confusion Matrix for Decision Tree



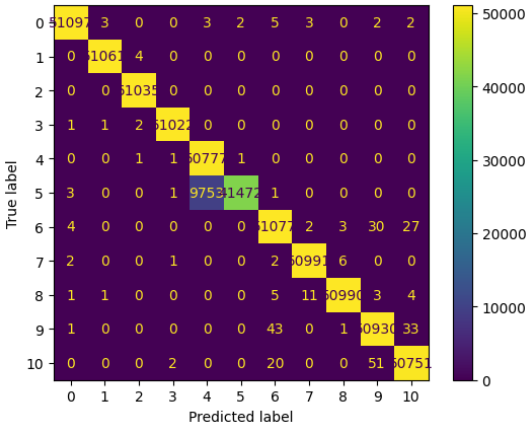
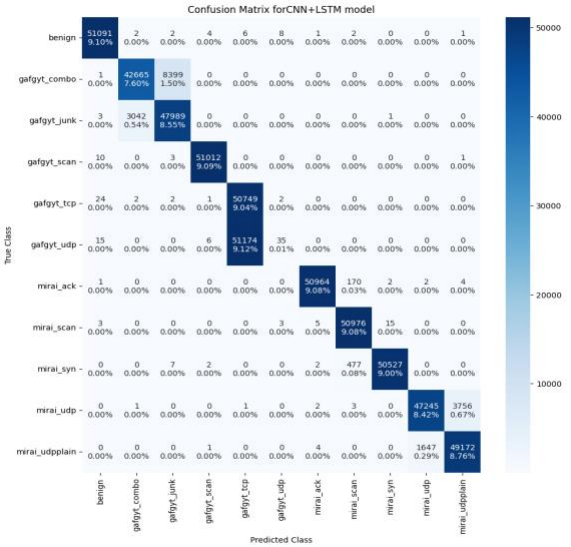


Fig. 10 Confusion Matrix for Random Forest Fig.11 Confusion Matrix for Extra Tree

From Figures 8,9,10 and 11 we can confirm that the model's performance on BASHLITE TCP and BASHLITE UDP cases are not accurate. As in most of the models the BASHLITE UDP is predicted as BASHLITE TCP. Only The Extra tree Classifier with its randomness in training has provided a better prediction on BASHLITE UDP cases.



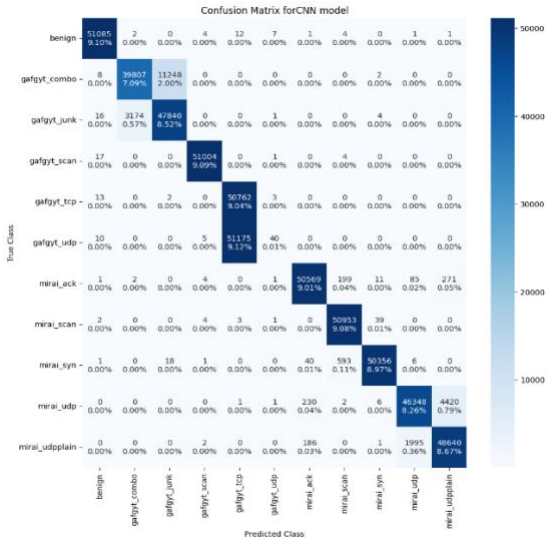


Fig.12 Confusion Matrix for CNN + LSTM model Fig.13 Confusion Matrix for 1D-CNN

From Figures 12, 13 we can illustrate that both the models have underperformed in predicting Bashlite UDP. But, Bashlite TCP has been predicted to be good. But there was some confusion between these two as well as the bashlite COMBO has 11000+ predictions as bashlite junk.

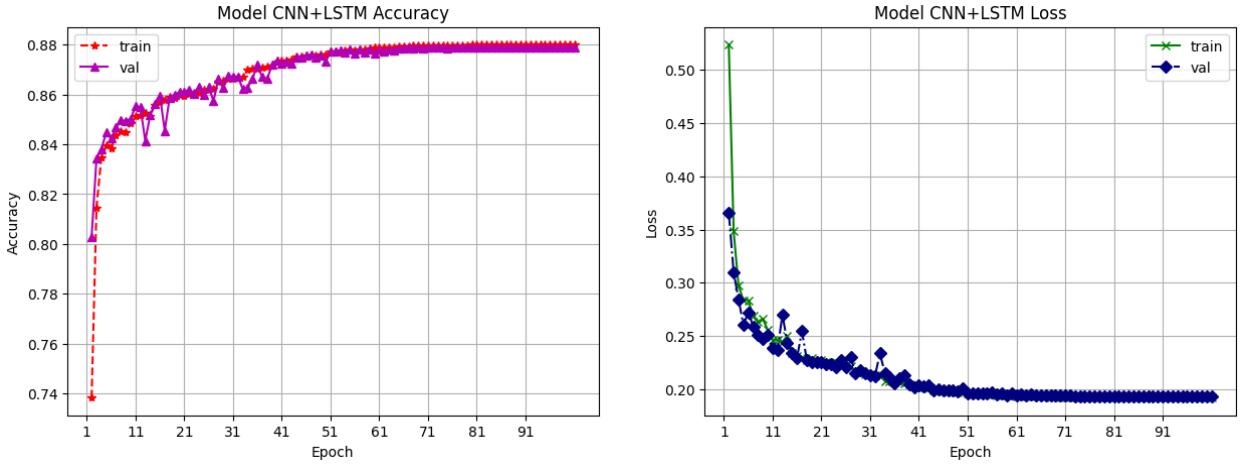


Fig 14. Train vs Validation Curve for CNN+LSTM model

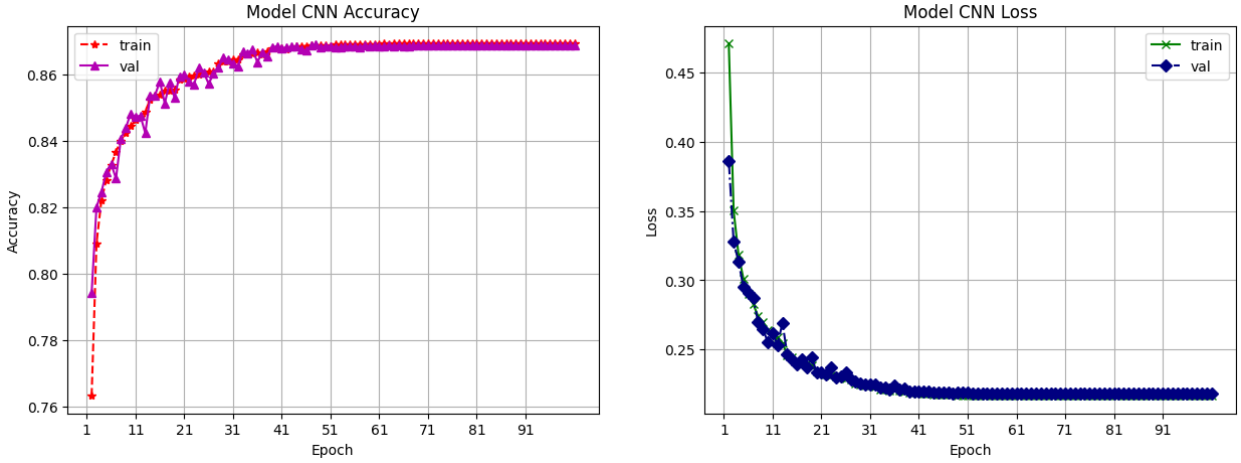
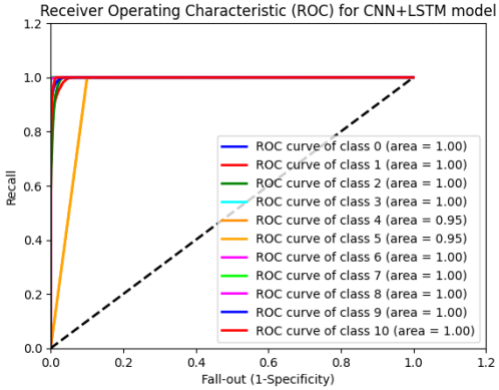
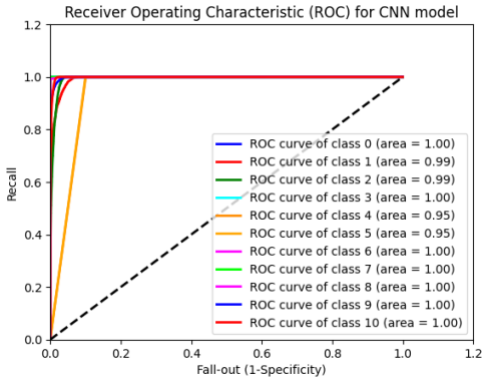


Fig 15. Train vs Validation Curve for 1D-CNN model

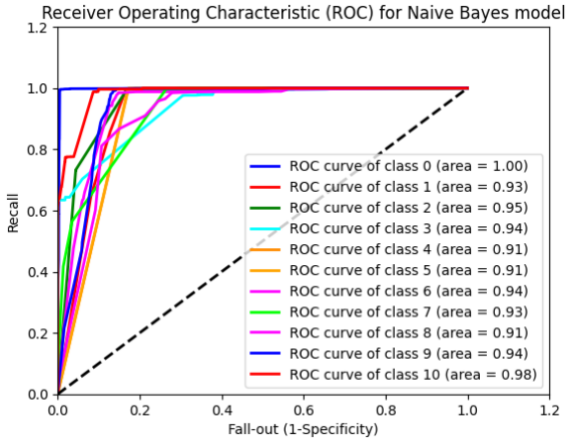
Figures 14 & 15 show the performance of the DL models for all the epochs.



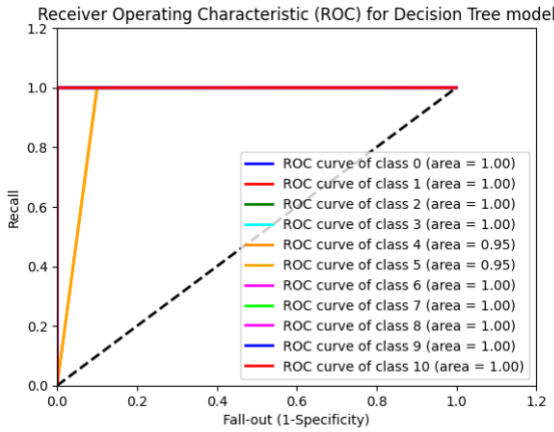
(A). ROC curve for CNN+LSTM



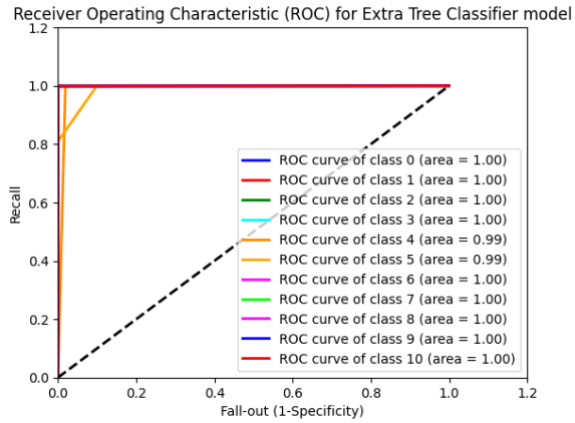
(B) ROC curve for 1d – CNN



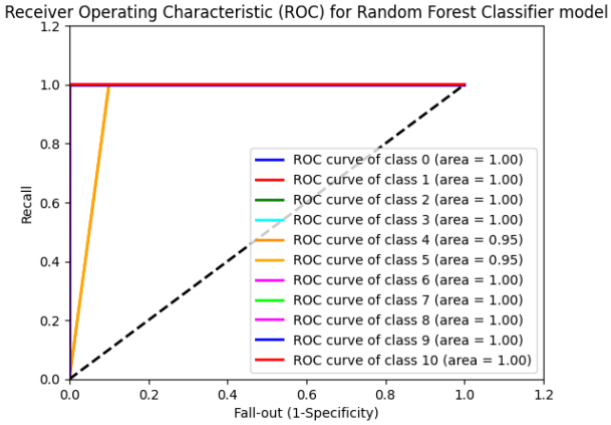
(C). ROC Curve for Naïve Bayes



(D). ROC Curve for Decision Tree



(E). ROC Curve for Random Forest



(F). ROC Curve for Extra Forest

Fig 16. ROC Curves

Fig 16. Shows the ROC curves, the roc curves are used to plot the TPR against FPR, where TPR is True Positive Rate and FPR is False Positive Rate. In fig.16, We see the ROC curve is plotted between Recall and Fall-Out (1- Specificity). In ROC curves TPR is also called Recall and False Rates are also called as 1-Specificity. In ROC graphs, the diagonal divides the space into good classification area and bad Classification area. If the graph is above the diagonal then it is said to be a good classification model. Otherwise, it is a bad Classification. Most of our curves are above the diagonal and if the curve is at y =1. Then it is said to be the perfect classification. We have got the Extra Tree classifier being the most consistent model of all as it has a whole of 100% accuracy. We had two CNN models trained and tested. Our designed CNN model has been good in classifying most of the results except the BASHLITE TCP and BASHLITE UDP where its precision and recall values are lower than others. It had predicted the BASHLITE UDP values as BASHLITE TCP. But for other attacks, the model has predicted spot-on results. So, the CNN models have the same similarity in predicting attacks of BASHLITE UDP AS TCP just like the random forest and decision tree Machine learning models. But, It had also some confusion in predicting BASHLITE COMBO as BASHLITE JUNK is illustrated in fig 12 & 13.. But in further scenarios taking some feature engineering methods may help our model be even better. And finally, the results of all models are shown as follows.

Metrics	Naïve Bayes	Decision Tree	Random Forest	Extra Tree	CNN+LSTM	1D-CNN
Micro-Averaged F1 Score	0.57	0.91	0.91	**0.98**	0.88	0.87
Macro-Averaged F1 Score	0.53	0.88	0.88	**0.98**	0.85	0.84
Weighted-Averaged F1 score	0.53	0.88	0.88	**0.98**	0.85	0.84
Micro-Averaged precision	0.573	0.947	0.952	**0.984**	0.905	0.891
Macro-Averaged precision	0.566	0.908	0.908	**0.982**	0.879	0.868
Weighted-Averaged Precision	0.573	0.947	0.952	**0.984**	0.905	0.891
Micro-Averaged recall	0.566	0.909	0.909	**0.982**	0.879	0.868
Macro-Averaged recall	0.566	0.909	0.908	**0.982**	0.879	0.868
Weighted-Averaged recall	0.566	0.909	0.908	**0.982**	0.879	0.868
Loss	1.02	0.13	0.15	**0.06**	0.193	0.21
FMS	0.502	0.919	0.919	**0.967**	0.867	0.862
Train Time	1.73s	38.41s	471.35s	**2.58s**	6120s	1506s
Test time	0.27s	0.08s	11.38s	**0.135s**	68s	37s

Table 7. Results of all models

[Text Wrapping Break]

By seeing Table 7. we can state that the extra tree classifier has the highest accuracy. So, going with the CNNs with having an issue on BASHLITE TCP and UDP attacks can be working better for other cases. When we compare the test time 1D-CNN outperforms CNN-LSTM on time taken to train and test. But in case of accuracy and loss CNN+LSTM is a slightly better CNN model. We have also trained the DL models having an extra layer of Convolutional layer and maxpooling layer for 1D_CNN to check if there would be any better results but we had same results as the model shown in Fig 6. Similarly we have tried the same thing for CNN+LSTM model by adding a layer of Convolutional Layer and an LSTM layer and compared the results with the previous model. As there was no change in the result we have went ahead with the model in Fig 7.

In the research, we could have used the autoencoders model, but our aim was to make a general model with good results, so we have not thought of using them. When compared with other papers like [17] which have used the N-BaIoT Dataset we can say that our model has performed better when compared to individual attacks we have 100% TPR when we go for individual attacks like Mirai scan, Mirai ack, bashlite udp etc. When compared with the bashlite junk, our predicting attacks like BASHLITE SCAN, MIRAI ACK, MIRAI SCAN and MIRAI UDPPlain have been more accurately predicted than the DL models of [17]. And when we consider the LSTM model, our model has outperformed [17]. We have an accuracy of 88% while theirs is 62%.

In [17], Models are made on different datasets like for each device models are being made to predict the attacks. So, For CNN models made on a) Doorbell dataset has an accuracy of 91%, while b) Baby Monitor has an accuracy of 91% as well, c) Security camera has an accuracy of 85% and d) Webcam has an accuracy of 82%. Similarly, LSTM models have a) Doorbell 62%, b) Baby Monitor 54%, c) Security Camera 25% and d) Webcam 43% respectively. All the ML and DL models in [17] are made on separate datasets. While our models are made on One Dataset as a whole. Similarly, [14] and [12] have also trained models on datasets separately rather than training them as one large dataset.

4.3 CONCLUSION

We have developed some of the ML and DL models to classify the Botnet Attacks. The framework consisted of a dataset, a few training models, and a classification model. The dataset used was N-baIoT which has 10 types of attacks generated by MIRAI and BASHLITE botnets, including TCP, UDP, and ACK. The ML models developed were Naïve Bayes, Random Forest, and Decision Tree. The DL models used have a 1d-CNN and a CNN-LSTM model. Through this Research, we state that the use of Convolutional neural networks on traffic data can help in presenting better models. The CNN models have worked better than most of the ML models we have performed. But the only issue the CNN models were having is with the TCP and UDP attacks being predicted wrong. This might be because when we are doing correlation, the features that are important for UDP or UDP plain might be dropped before training the model and rest of the attacks that have been performed by botnets are like TCP or parts of TCP attacks.

So, making model for UDP separately would give an effective model. In the future, we will try to improve the models that could help predict those things well.

REFERENCES:

[1] J. Leonard, S. Xu and R. Sandhu, "A Framework for Understanding Botnets," 2009 International Conference on Availability, Reliability and Security, Fukuoka, Japan, 2009, pp. 917-922, doi: 10.1109/ARES.2009.65.

[2] Feily, Maryam, Alireza Shahrestani, and Sureswaran Ramadass. "A survey of botnet and botnet detection." 2009 Third International Conference on Emerging Security Information, Systems and Technologies. IEEE, 2009.

[3] Josh Frulinger, Contributing writer, CSO. 'WannaCry explained: A perfect ransomware storm', Online, Aug 2022. [Online]. Available: https://www.csoonline.com/article/3227906/wannacry-explained-a-perfect-ransomware-storm.html (Accessed April 2023)

[4] Stephan Hilt, Fernando Merces, 'VPNFIlter Two Years Later: Routers Still Compromised', Online, Jan 2021. [Online]. AVailable:https://www.trendmicro.com/en_us/research/21/a/vpnfilter-two-years-later-routers-still-compromised-.html (Accessed April 2023)

[5] Lucian Constantin, CSO Senior Writer, 'TrickBot explained: A multi-purpose crimeware tool that haunted businesses for years', Online, Dec 2020. [Online]. Available: https://www.csoonline.com/article/3600457/trickbot-explained-a-multi-purpose-crimeware-tool-that-haunted-businesses-for-years.html

[6] M.Antonakakis,T.April,M.Bailey,M.Bernhard,E.Bursztein, J.Cochran,Z.Durumeric,J.A.Halderman,L.Invernizzi,M.Kallitsis, D.Kumar,C.Lever,Z.Ma,J.Mason,D.Menscher,C.Seaman, N.Sullivan,K.Thomas,andY.Zhou,"Understandingthemiraibotnet," inProc.ofUSENIXSecuritySymposium,2017.

[7] K.Angrishi,"Turninginternetofthings(iot)intointernetofvulnerabilities(iov): Iotbotnets,"2017.

[8] BASHLITE, 'wikipedia.org', Online, April 2023. [Online]. Available:https://en.wikipedia.org/wiki/BASHLITE (Accessed April 2023)

[9] Yunsheng Fu, Fang Lou, Fangzhi Meng, Zhihong Tian, Hua Zhang, and Feng Jiang. An intelligent network attack detection method based on rnn. In 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC), pages 483–489. IEEE, 2018.

[10] Soodeh Hosseini, Ali Emamali Nezhad, and Hossein Seilani. Botnet detection using negative selection algorithm, convolution neural network and classification methods. Evolving Systems, 13(1):101–115, 2022.

[11] Furqan Alam, Rashid Mehmood, Iyad Katib, and Aiiad Albeshri. Analysis of eight data mining algorithms for smarter internet of things (iot). Procedia Computer Science, 98:437–442, 2016.
[12] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. IEEE Pervasive Computing, 17(3):12–22, 2018.
[13] Elike Hodo, Xavier Bellekens, Andrew Hamilton, Pierre-Louis Dubouilh, Ephraim Iorkyase, Christos Tachtatzis, and Robert Atkinson. Threat analysis of iot networks using artificial neural network intrusion detection system. In 2016 International Symposium on Networks, Computers and Communications (ISNCC), pages 1–6. IEEE, 2016.
[14] Amaal Al Shorman, Hossam Faris, and Ibrahim Aljarah. Unsupervised intelligent system based on one class support vector machine and grey wolf optimization for iot botnet detection. Journal of Ambient Intelligence and Humanized Computing, 11:2809–2825, 2020.
[15] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. arXiv preprint arXiv:1802.09089, 2018.
[16] Kashif Naveed (2020). N-BaIoT Dataset to Detect IoT Botnet Attacks [Dataset].
[17] Jiyeon Kim, Minsun Shim, Seungah Hong, Yulim Shin, and Eunjung Choi. Intelligent detection of iot botnets using machine learning and deep learning. Applied Sciences, 10(19):7009, 2020.
[18] Nathan Shone, Tran Nguyen Ngoc, Vu Dinh Phai, and Qi Shi. A deep learning approach to network intrusion detection. IEEE transactions on emerging topics in computational intelligence, 2(1):41–50, 2018.
[19] Xiaofeng Xie, Di Wu, Siping Liu, and Renfa Li. Iot data analytics using deep learning. arXiv preprint arXiv:1708.03854, 2017.
[20] Hamed HaddadPajouh, Ali Dehghantanha, Raouf Khayami, and KimKwang Raymond Choo. A deep recurrent neural network-based approach for internet of things malware threat hunting. Future Generation Computer Systems, 85:88–96, 2018.
[21] Christopher D McDermott, Farzan Majdani, and Andrei V Petrovski. Botnet detection in the internet of things using deep learning approaches. In 2018 international joint conference on neural networks (IJCNN), pages 1– 8. IEEE, 2018.
[22] Olivier Brun, Yonghua Yin, Erol Gelenbe, Y Murat Kadioglu, Javier Augusto-Gonzalez, and Manuel Ramos. Deep learning with dense random neural networks for detecting attacks against iot-connected home environments. In Security in Computer and Information Sciences: First International ISCIS Security Workshop 2018, Euro-CYBERSEC 2018, London, UK, February 26-27, 2018, Revised Selected Papers 1, pages 79–89. Springer International Publishing, 2018.
[23] Yair Meidan, Michael Bohadana, Asaf Shabtai, Martin Ochoa, Nils Ole Tippenhauer, Juan Davis Guarnizo, and Yuval Elovici. Detection of unauthorized iot devices using machine learning techniques. arXiv preprint arXiv:1709.04647, 2017.
[24] Fahimeh Farahnakian and Jukka Heikkonen. A deep auto-encoder based approach for intrusion detection system. In 2018 20th International Conference on Advanced Communication Technology (ICACT), pages 178–183. IEEE, 2018.
[25] Colab Pro: IS it Worth the Money?, Dario Radecic, May 20,2020. https://towardsdatascience.com/colab-pro-is-it-worth-the-money-32a1744f42a8 [Accessed on May, 2023]
[25] 1D CNN LSTM Model: https://colab.research.google.com/drive/1g-sO6ZcFOe-oQ0uf3QRJx139xC7cYF_I?usp=sharing
[26] 1D CNN Model

https://colab.research.google.com/drive/1vjIP-QdnF8PMpF-X8foziSpLv_Ek_voJ?authuser=3#scrollTo=Wd0zoeahu7hT