Machine Learning-Based Malware Detection: A Comparative Study of Classifiers

Dr G.N.V Vibhav Reddy[1], A. Pallavi[2] , N. Ramya[2], B.Shekar Reddy[2], K. Hemanth[2]

[1]Professor, [2]UG Student, [1,2] Department of Computer Science and Engineering(DS)

Sree Dattha Group of Institutions, Sheriguda, Hyderabad, Telangana

ABSTRACT

In today's interconnected world, the widespread adoption of Internet of Things (IoT) devices has brought forth a host of conveniences and opportunities. However, this technological revolution has also opened the door to a new type of malware attacks, with attackers exploiting vulnerabilities in IoT devices to compromise user privacy, disrupt critical services, and wreak havoc. Traditional security measures have proven inadequate to combat the evolving complexity of these malware attacks, necessitating a more advanced and adaptive approach. This urgency has given rise to the development of a Machine Learning Model for Malware attack Detection and Classification in IoT Environments (ML-IoT-CD). In addition, the need for a robust cybersecurity solution in IoT environments has become paramount due to the increasing reliance on these devices for critical applications. Existing intrusion detection systems and conventional security measures often lack the scalability and agility needed to keep pace with rapidly evolving attack techniques. As a result, there is a pressing demand for an intelligent, automated, and proactive cyber defense mechanism capable of real-time detection and classification of emerging malware attacks. The ML-IoT-CD model aims to fulfill this need by harnessing the power of machine learning algorithms to analyze vast amounts of data generated by IoT devices. By doing so, it can effectively distinguish between legitimate and malicious activities, thereby bolstering the security posture of IoT ecosystems.

Keywords: IoT Security, Malware Detection, Machine Learning, Intrusion Detection, Cybersecurity, Real-time Classification, ML-IoT-CD, Anomaly Detection, Attack Mitigation, Threat Intelligence.

1.Introduction

1.1 Overview

The general idea of the Internet of Things (IoT) is to allow for communication between human-to-thing or thing-to-thing(s). Things denote sensors or devices, whilst human or an object is an entity that can request or deliver a service [1]. The interconnection amongst the entities is always complex. IoT is broadly acceptable and implemented in various domains, such as healthcare, smart home, and agriculture. However, IoT has a resource constraint and heterogeneous environments, such as low computational power and memory. These constraints create problems in providing and implementing a security solution in IoT devices. These constraints further escalate the existing challenges for IoT environment. Therefore, various kinds of attacks are possible due to the vulnerability of IoT devices. IoT-based botnet attack is one of the most popular, spreads faster and create more impact than other attacks. In recent years, several works have been conducted to detect and avoid this kind of attacks [2]–[3] by using novel approaches. Hence, a plethora of relevant of relevant models, methods, and etc. have been introduced over the past few years, with quite a reasonable number of studies reported in the research domain. Many studies are trying to protect against these botnet attacks on the IoT environment. However, there are many gaps still existing to develop an effective detection mechanism. An intrusion detection system (IDS) is one of the efficient ways to deal with attacks. However, the traditional IDSs are often not able to be deployed for the IoT environments due to the resource constraint problem of these devices. The complex cryptographic mechanisms cannot be embedded in many IoT devices either for the same reason. There are mainly two kinds of IDSs: the anomaly and misuse approaches. The misuse-based, also called the signature-based, approach, is based on the attacks' signatures, and they can also be found in most public IDSs, specifically Suricata [4]. Formally, the attacker can easily circumvent the signature-based approaches, and these mechanisms cannot guarantee to detect the unknown attacks and the variances of known attacks. The anomaly-based systems are based on normal data and can support to identify the unknown attacks. However, the different nature of IoT devices is being faced with the difficulty of collecting common normal data. The machine learning-based detection can guarantee detection of not only the known attacks and their variances. Therefore, we proposed a machine learning-based botnet attack detection architecture. We also adopted a feature selection method to reduce the demand for processing resources for performing the detection system on resource constraint devices. The experiment results indicate that the detection accuracy of our proposed system is high enough to detect the botnet attacks. Moreover, it can support the extension for detecting the new distinct kinds of attacks.

2.Literature Survey

Soe et al. [5] adopted a lightweight detection system with a high performance. The overall detection performance achieves around 99% for the botnet attack detection using three different ML algorithms, including artificial neural network (ANN), J48 decision tree, and Naïve Bayes. The experiment result indicated that the proposed architecture can effectively detect botnet-based attacks, and also can be extended with corresponding sub-engines for new kinds of attacks.Ali et al. [6] outlined the existing proposed contributions, datasets utilised, network forensic methods utilised and research focus of the primary selected studies. The demographic characteristics of primary studies were also outlined. The result of this review revealed that research in this domain is gaining momentum, particularly in the last 3 years (2018-2020). Nine key contributions were also identified, with Evaluation, System, and Model being the most conducted. Irfan et al. [7] classified the incoming data in the IoT, contain a malware or not. In this research, this work under sample the dataset because the datasets contain imbalance class. After that, this work classified the sample using Random Forest. This work used Naive Bayes, K-Nearest Neighbor and Decision Tree too as a comparison. The dataset that has been used in this research are from UCI Machine Learning Depository's Website. The dataset showed the data traffic from the IoT Device in a normal condition and attacked by Mirai or Bashlite.Shah et al. [8] presented a concept called 'login puzzle' to prevent capture of IoT devices in a large scale. Login puzzle is a variant of client puzzle, which presented a puzzle to the remote device during the login process to prevent unrestricted log-in attempts. Login puzzle is a set of multiple mini puzzles with a variable complexity, which the remote device is required to solve before logging into any IoT device. Every unsuccessful log-in attempt increases the complexity of solving the login puzzle for the next attempt. This paper introduced a novel mechanism to change the complexity of puzzle after every unsuccessful login attempt. If each IoT device had used login puzzle, Mirai attack would have required almost two months to acquire devices, while it acquired them in 20 h.

Tzagkarakis et al. [9] presented an IoT botnet attack detection method based on a sparsity representation framework using a reconstruction error thresholding rule for identifying malicious network traffic at the IoT edge coming from compromised IoT devices. The botnet attack detection is performed based on small-sized benign IoT network traffic data, and thus we have no prior knowledge about malicious IoT traffic data. We present our results on a real IoT-based network dataset and show the efficacy of proposed technique against a reconstruction error-based autoencoder approach.Meidan et al. [10] proposed a novel network-based anomaly detection method for the IoT called N-BaIoT that extracts behavior snapshots of the network and uses deep autoencoders to detect anomalous network traffic from compromised IoT devices. To evaluate the method, this work infected nine commercial IoT devices in our lab with two widely known IoT-based botnets, Mirai and BASHLITE. The evaluation results demonstrated the proposed methods ability to detect the attacks accurately and instantly as they were being launched from the compromised IoT devices that were part of a botnet. Popoola et al. [11] proposed the federated DL (FDL) method for zero-day botnet attack detection to avoid data privacy leakage in IoT-edge devices. In this method, an optimal deep neural network (DNN) architecture is employed for network traffic classification. A model parameter server remotely coordinates the independent training of the DNN models in multiple IoT-edge devices, while the federated averaging (FedAvg) algorithm is used to aggregate local model updates. A global DNN model is produced after several communication rounds between the model parameter server and the IoT-edge devices. The zero-day botnet attack scenarios in IoT-edge devices are simulated with the Bot-IoT and N-BaIoT data sets. Hussain et al. [12] produced a generic scanning and DDoS attack dataset by generating 33 types of scans and 60 types of DDoS attacks. In addition, this work partially integrated the scan and DDoS attack samples from three publicly available datasets for maximum attack coverage to better train the machine learning algorithms. Afterwards, this work proposed a two-fold machine learning approach to prevent and detect IoT botnet attacks. In the first fold, this work trained a state-of-the-art deep learning model, i.e., ResNet-18 to detect the scanning activity in the premature attack stage to prevent IoT botnet attacks. While, in the second fold, this work trained another ResNet-18 model for DDoS attack identification to detect IoT botnet attacks. Abu et al. [13] proposed an ensemble learning model for botnet attack detection in IoT networks called ELBA-IoT that profiles behavior features of IoT networks and uses ensemble learning to identify anomalous network traffic from compromised IoT devices. In addition, this IoT-based botnet detection approach characterizes the evaluation of three different machine learning techniques that belong to decision tree techniques (AdaBoosted, RUSBoosted, and bagged). To evaluate ELBA-IoT, we used the N-BaIoT-2021 dataset, which comprises records of both normal IoT network traffic and botnet attack traffic of infected IoT devices. Alharbi et al. [14] proposed Gaussian distribution used in the population initialization. Furthermore, the local search mechanism was followed by the Gaussian density function and local-global best function to achieve better exploration during each generation. Enhanced BA was further employed for neural network hyperparameter tuning and weight optimization to classify ten different botnet attacks with an additional one benign target class. The proposed LGBA-NN algorithm was tested on an N-BaIoT data set with extensive real traffic data with benign and malicious target classes. The performance of LGBA-NN was compared with several recent advanced approaches such as weight optimization using Particle Swarm Optimization (PSO-NN) and BA-NN. Ahmed et al. [15] proposed a model for detecting botnets using deep learning to identify zero-day botnet attacks in real time. The proposed model is trained and evaluated on a CTU-13 dataset with multiple neural network designs and hidden layers. Results demonstrated that the deep-learning artificial neural network model can accurately and efficiently identify botnets.

3.PROPOSED SYSTEM

3.1 Detecting cyberattacks using a combination of data preprocessing techniques, such as Standard Scaling, and a RFC classifier is a common approach in cybersecurity. Figure 4.1 shows a malware-attack attack detection system model.

Step 1: Preprocessing: Gather a dataset of network traffic or system logs, where each data point is labeled as either a normal activity or a malware-attack. Preprocess the data to make it suitable for training a RFC classifier. This may include handling missing values, encoding categorical variables, and scaling numerical features.

Step 2: Standard Scaling: Extract relevant features from the dataset. Common features for malware-attack detection may include network traffic statistics, log event patterns, and more. Feature selection or dimensionality reduction techniques can be applied if the dataset has many features. Use Standard Scaling to standardize the numerical features in the dataset. Standard Scaling is preferred over standard scaling when dealing with data that may have outliers. Standard Scaling scales the features in a way that is less affected by extreme values, making it a suitable choice for cybersecurity datasets.

Step 3: Split the Data: Divide your dataset into training, validation, and test sets. A common split might be 80% for training, and 20% for testing.

Step 4: RFC Classifier: Design and build an RFC classifier.

Step 5: Training: Train the RFC classifier using the training dataset. During training, monitor performance on the validation set to avoid overfitting and adjust hyperparameters accordingly.

Step 6: Evaluation: Evaluate the trained model on the test dataset to assess its performance. Common evaluation metrics for malware-attack detection include accuracy, precision, recall, F1-score, and confusion matrix.

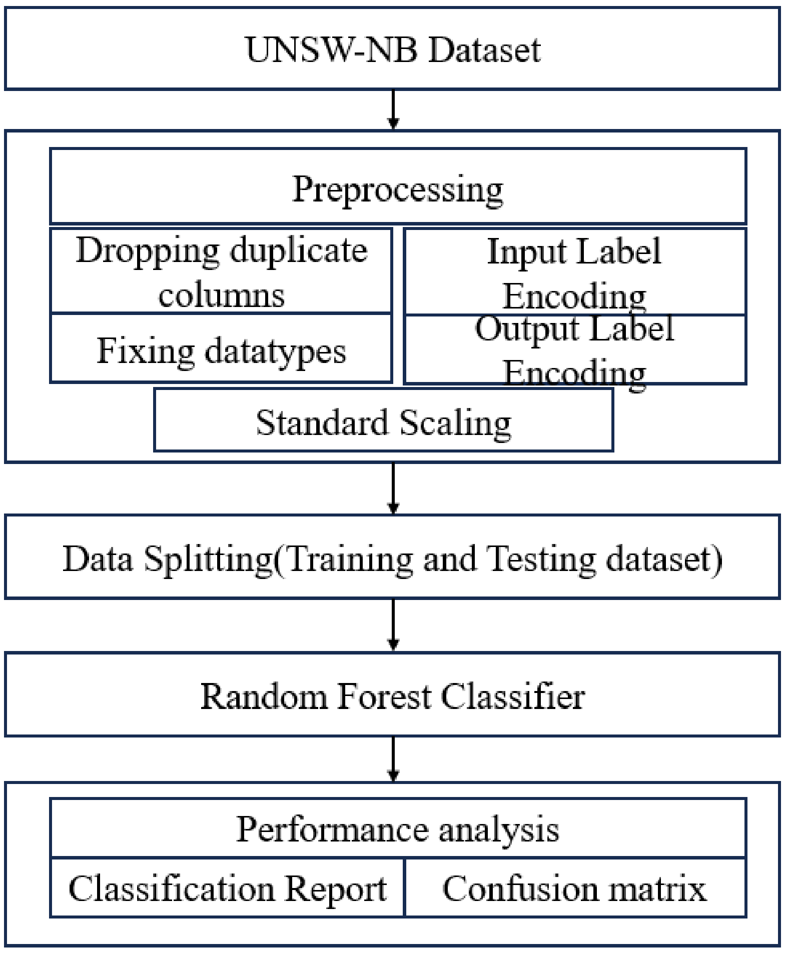


Fig. 1: Block diagram of proposed system.

3.2 Data Preprocessing

Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So, for this, we use data pre-processing task.

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

One-Hot Encoding: Categorical variables are one-hot encoded to convert them into a numerical format suitable for machine learning models. The code uses the pd.get_dummies() function to create binary columns for each category within categorical variables. This transformation allows machine learning algorithms to work with categorical data effectively.

3.3 Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

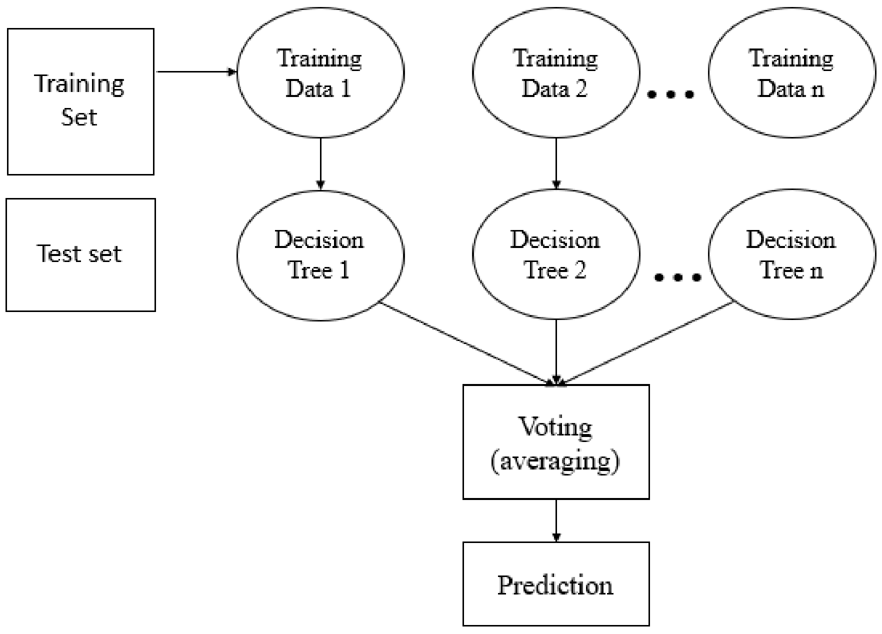


Fig. 2: Random Forest algorithm.

3.1.1 Random Forest algorithm

Step 1: In Random Forest n number of random records are taken from the data set having k number of records.

Step 2: Individual decision trees are constructed for each sample.

Step 3: Each decision tree will generate an output.

Step 4: Final output is considered based on Majority Voting or Averaging for Classification and regression respectively.

3.1.2 Important Features of Random Forest

- **Diversity**- Not all attributes/variables/features are considered while making an individual tree, each tree is different.
- **Immune to the curse of dimensionality**- Since each tree does not consider all the features, the feature space is reduced.
- **Parallelization**-Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.
- **Train-Test split**- In a random forest we don't have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.
- **Stability**- Stability arises because the result is based on majority voting/ averaging.

3.1.3 Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random Forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Below are some points that explain why we should use the Random Forest algorithm

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

3.1.4 Types of Ensembles

Before understanding the working of the random forest, we must look into the ensemble technique. Ensemble simply means combining multiple models. Thus, a collection of models is used to make predictions rather than an individual model. Ensemble uses two types of methods:

Bagging– It creates a different training subset from sample training data with replacement & the final output is based on majority voting. For example, Random Forest. Bagging, also known as Bootstrap Aggregation is the ensemble technique used by random forest. Bagging chooses a random sample from the data set. Hence each model is generated from the samples (Bootstrap Samples) provided by the Original Data with replacement known as row sampling. This step of row sampling with replacement is called bootstrap. Now each model is trained independently which generates results. The final output is based on majority voting after combining the results of all models. This step which involves combining all the results and generating output based on majority voting is known as aggregation.

Boosting– It combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy. For example, ADA BOOST, XG BOOST.

4.RESULT AND DISCUSSION

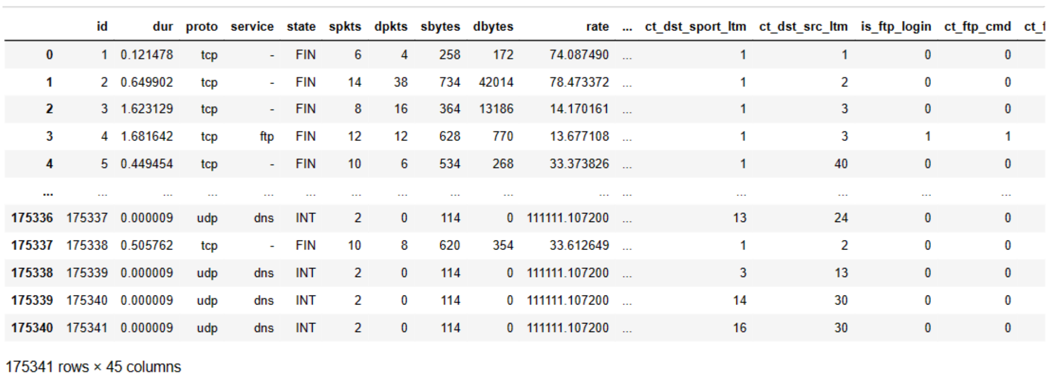
4.1 Dataset Description

The UNSW-NB15 dataset is a network traffic dataset that is commonly used for malware-attack detection system evaluation. Here's a brief description of each column in the dataset:

- **id**: A unique identifier for each record in the dataset.
- **dur**: Duration of the connection in seconds. Represents the time elapsed for the connection.
- **proto**: The transport layer protocol used in the connection, such as TCP, UDP, ICMP, etc.
- **service**: The network service on the destination, indicating the type of service being accessed (e.g., HTTP, FTP).
- **state**: The connection state, indicating the status of the connection (e.g., FIN, CON, INT).
- **spkts**: The count of packets sent from the source to the destination.
- **dpkts**: The count of packets sent from the destination to the source.
- **sbytes**: The number of source-to-destination bytes in the connection.
- **dbytes**: The number of destination-to-source bytes in the connection.
- **rate**: Data transfer rate, representing the average number of bits transferred per second.
- **sttl**: Source time to live, indicating the remaining time to live of the source in the connection.
- **dttl**: Destination time to live, indicating the remaining time to live of the destination in the connection.
- **sload**: Source bits per second, representing the data load on the source side.
- **dload**: Destination bits per second, representing the data load on the destination side.
- **sloss**: Source packets retransmitted or lost during the connection.
- **dloss**: Destination packets retransmitted or lost during the connection.
- **sinpkt**: Source inter-packet arrival time, indicating the time between consecutive packets from the source.
- **dinpkt**: Destination inter-packet arrival time, indicating the time between consecutive packets from the destination.
- **sjit**: Source jitter, representing the variability in inter-packet arrival times on the source side.
- **djit**: Destination jitter, representing the variability in inter-packet arrival times on the destination side.
- **swin**: Source TCP window size, indicating the size of the TCP window on the source side.
- **stcpb**: Source TCP base sequence number, indicating the initial sequence number of the connection on the source side.
- **dtcpb**: Destination TCP base sequence number, indicating the initial sequence number of the connection on the destination side.
- **dwin**: Destination TCP window size, indicating the size of the TCP window on the destination side.
- **tcprtt**: TCP connection setup round-trip time, representing the time taken for the TCP connection setup.

- **synack**: Time taken for the TCP connection setup (SYN-ACK phase).
- **ackdat**: Time taken for the TCP connection setup (ACK-DAT phase).
- **smean**: Mean of the source payload data size, representing the average size of data sent from the source.
- **dmean**: Mean of the destination payload data size, representing the average size of data received at the destination.
- **trans_depth**: Connection transaction depth, indicating the depth of the transaction in the connection.
- **response_body_len**: Length of the response body, indicating the size of the response payload.
- **ct_srv_src**: Number of connections to the same service as the current connection in the past two seconds.
- **ct_state_ttl**: Number of connections with the same source TTL (Time to Live) value.
- **ct_dst_ltm**: Number of connections with the same destination IP address in the past two seconds.
- **ct_src_dport_ltm**: Number of connections with the same source port to the same destination port in the past two seconds.
- **ct_dst_sport_ltm**: Number of connections with the same destination port to the same source port in the past two seconds.
- **ct_dst_src_ltm**: Number of connections with the same source and destination IP addresses in the past two seconds.
- **is_ftp_login**: Binary indicator of whether the FTP login was successful or not.
- **ct_ftp_cmd**: Number of FTP commands carried in the connection.
- **ct_flw_http_mthd**: Number of HTTP methods carried in the connection.
- **ct_src_ltm**: Number of connections with the same source IP address in the past two seconds.
- **ct_srv_dst**: Number of connections with the same source and destination service in the past two seconds.
- **is_sm_ips_ports**: Binary indicator of whether the source and destination ports are the same.
- **attack_cat**: The category of the attack, such as DoS (Denial of Service), Probe, R2L (Unauthorized access from a remote machine), U2R (Unauthorized access to privileged local resources).
- **label**: Binary label indicating normal (0) or malicious (1) activity in the network connection.

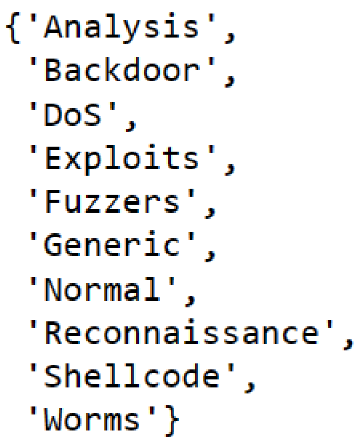
4.2 Results and Description



	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	...	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd	ct_f
0	1	0.121478	tcp	-	FIN	6	4	258	172	74.087490	...	1	1	0	0	
1	2	0.649902	tcp	-	FIN	14	38	734	42014	78.473372	...	1	2	0	0	
2	3	1.623129	tcp	-	FIN	8	16	364	13186	14.170161	...	1	3	0	0	
3	4	1.681642	tcp	ftp	FIN	12	12	628	770	13.677108	...	1	3	1	1	
4	5	0.449454	tcp	-	FIN	10	6	534	268	33.373826	...	1	40	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
175336	175337	0.000009	udp	dns	INT	2	0	114	0	111111.107200	...	13	24	0	0	
175337	175338	0.505762	tcp	-	FIN	10	8	620	354	33.612649	...	1	2	0	0	
175338	175339	0.000009	udp	dns	INT	2	0	114	0	111111.107200	...	3	13	0	0	
175339	175340	0.000009	udp	dns	INT	2	0	114	0	111111.107200	...	14	30	0	0	
175340	175341	0.000009	udp	dns	INT	2	0	114	0	111111.107200	...	16	30	0	0	

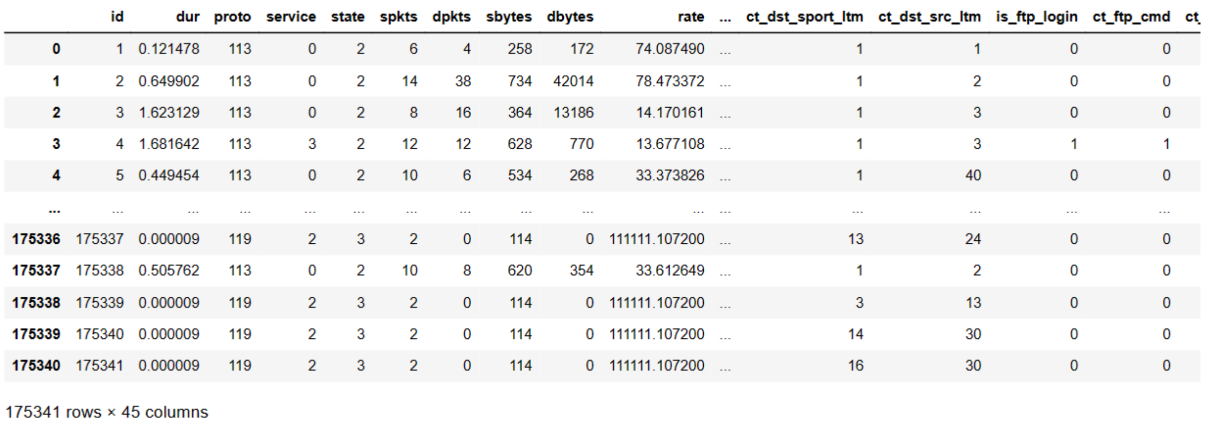
175341 rows × 45 columns

Figure 10.1. Sample dataset.



```
{'Analysis',
 'Backdoor',
 'DoS',
 'Exploits',
 'Fuzzers',
 'Generic',
 'Normal',
 'Reconnaissance',
 'Shellcode',
 'Worms'}
```

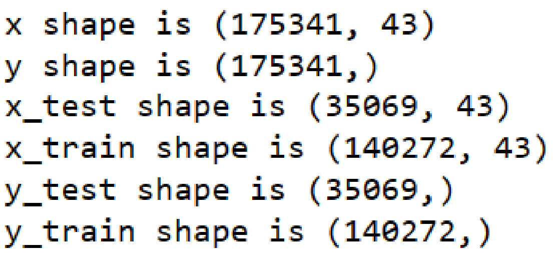
Fig. 3: Attack classes in dataset.



	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	...	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd	ct_
0	1	0.121478	113	0	2	6	4	258	172	74.087490	...	1	1	0	0	
1	2	0.649902	113	0	2	14	38	734	42014	78.473372	...	1	2	0	0	
2	3	1.623129	113	0	2	8	16	364	13186	14.170161	...	1	3	0	0	
3	4	1.681642	113	3	2	12	12	628	770	13.677108	...	1	3	1	1	
4	5	0.449454	113	0	2	10	6	534	268	33.373826	...	1	40	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
175336	175337	0.000009	119	2	3	2	0	114	0	111111.107200	...	13	24	0	0	
175337	175338	0.505762	113	0	2	10	8	620	354	33.612649	...	1	2	0	0	
175338	175339	0.000009	119	2	3	2	0	114	0	111111.107200	...	3	13	0	0	
175339	175340	0.000009	119	2	3	2	0	114	0	111111.107200	...	14	30	0	0	
175340	175341	0.000009	119	2	3	2	0	114	0	111111.107200	...	16	30	0	0	

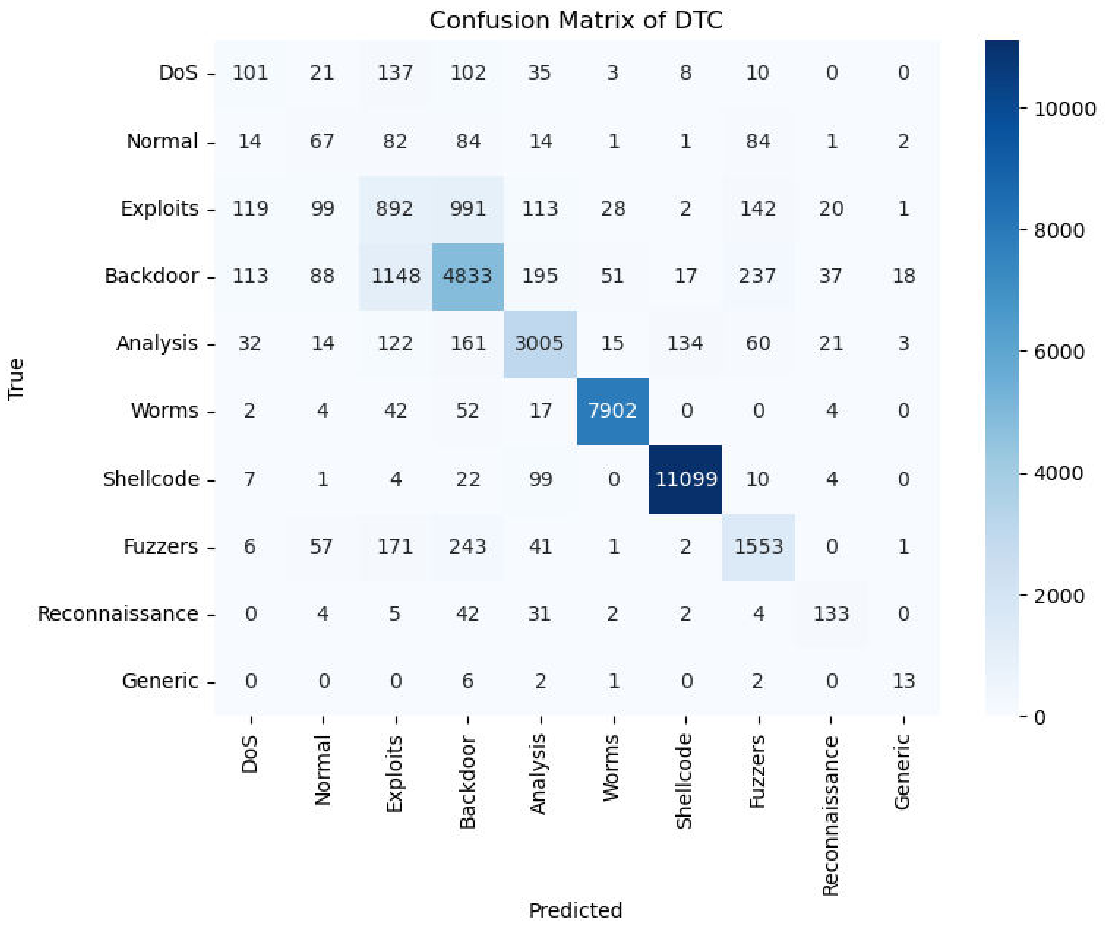
175341 rows × 45 columns

Fig. 4: Dataset after label encoding.



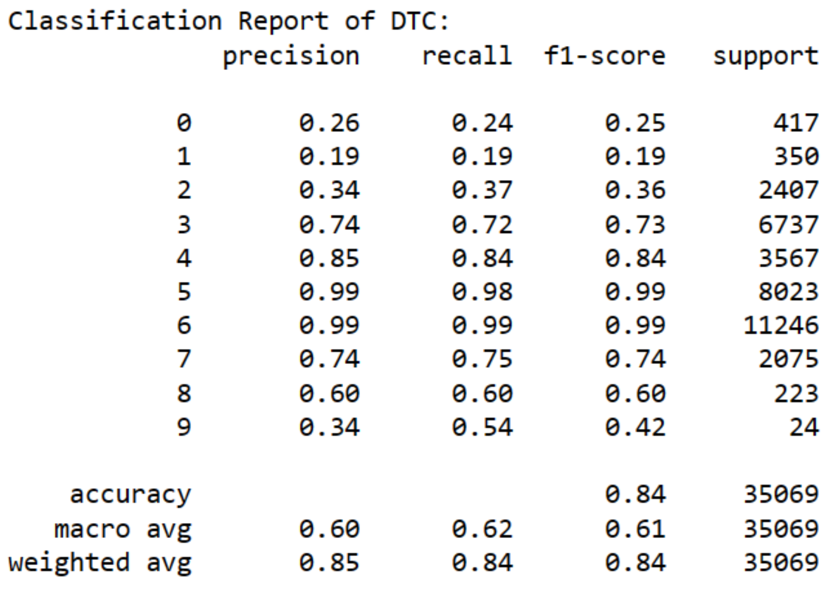
```
x shape is (175341, 43)
y shape is (175341,)
x_test shape is (35069, 43)
x_train shape is (140272, 43)
y_test shape is (35069,)
y_train shape is (140272,)
```

Fig. 5: Train and Test sizes in dataset.



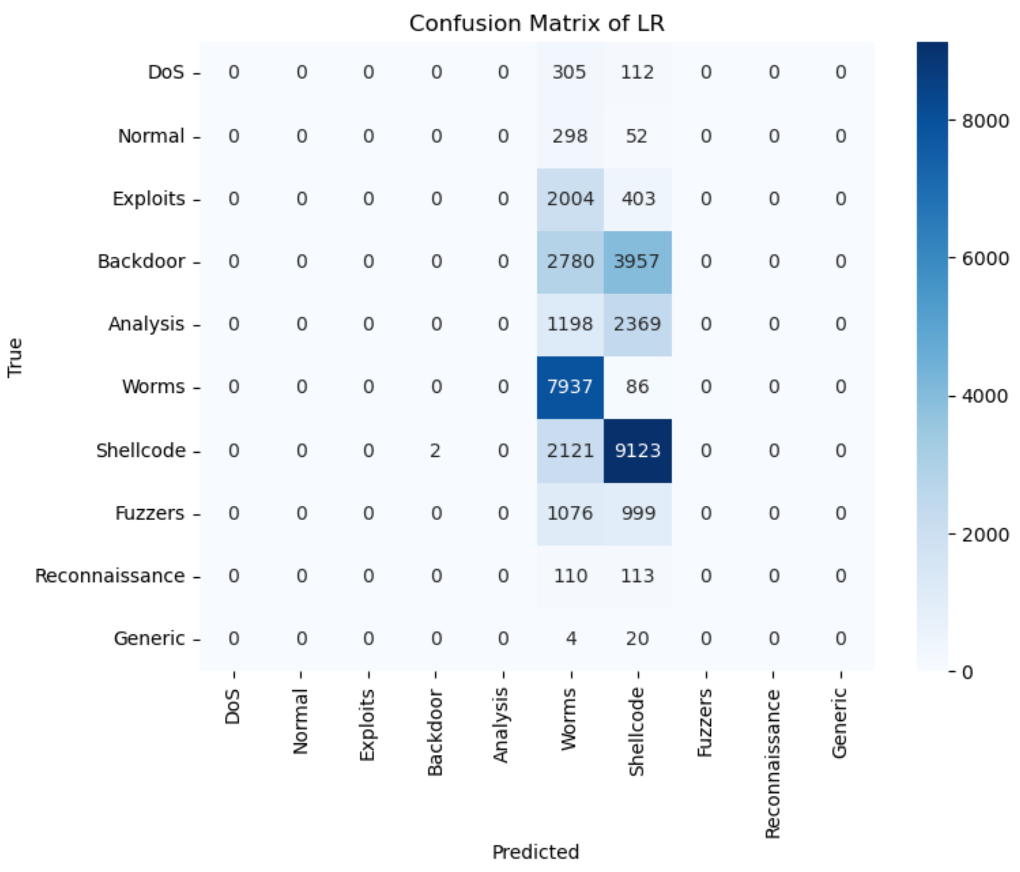
True \ Predicted	DoS	Normal	Exploits	Backdoor	Analysis	Worms	Shellcode	Fuzzers	Reconnaissance	Generic
DoS	101	21	137	102	35	3	8	10	0	0
Normal	14	67	82	84	14	1	1	84	1	2
Exploits	119	99	892	991	113	28	2	142	20	1
Backdoor	113	88	1148	4833	195	51	17	237	37	18
Analysis	32	14	122	161	3005	15	134	60	21	3
Worms	2	4	42	52	17	7902	0	0	4	0
Shellcode	7	1	4	22	99	0	11099	10	4	0
Fuzzers	6	57	171	243	41	1	2	1553	0	1
Reconnaissance	0	4	5	42	31	2	2	4	133	0
Generic	0	0	0	6	2	1	0	2	0	13

Fig. 6: Existing DTC confusion matrix.



```
Classification Report of DTC:
              precision    recall  f1-score   support

           0       0.26      0.24      0.25       417
           1       0.19      0.19      0.19       350
           2       0.34      0.37      0.36      2407
           3       0.74      0.72      0.73      6737
           4       0.85      0.84      0.84      3567
           5       0.99      0.98      0.99      8023
           6       0.99      0.99      0.99     11246
           7       0.74      0.75      0.74      2075
           8       0.60      0.60      0.60       223
           9       0.34      0.54      0.42        24

    accuracy                           0.84     35069
   macro avg       0.60      0.62      0.61     35069
weighted avg       0.85      0.84      0.84     35069
```

Fig. 7: Existing DTC classification report.



True \ Predicted	DoS	Normal	Exploits	Backdoor	Analysis	Worms	Shellcode	Fuzzers	Reconnaissance	Generic
DoS	0	0	0	0	0	305	112	0	0	0
Normal	0	0	0	0	0	298	52	0	0	0
Exploits	0	0	0	0	0	2004	403	0	0	0
Backdoor	0	0	0	0	0	2780	3957	0	0	0
Analysis	0	0	0	0	0	1198	2369	0	0	0
Worms	0	0	0	0	0	7937	86	0	0	0
Shellcode	0	0	0	2	0	2121	9123	0	0	0
Fuzzers	0	0	0	0	0	1076	999	0	0	0
Reconnaissance	0	0	0	0	0	110	113	0	0	0
Generic	0	0	0	0	0	4	20	0	0	0

Fig. 8: Existing LRC confusion matrix.

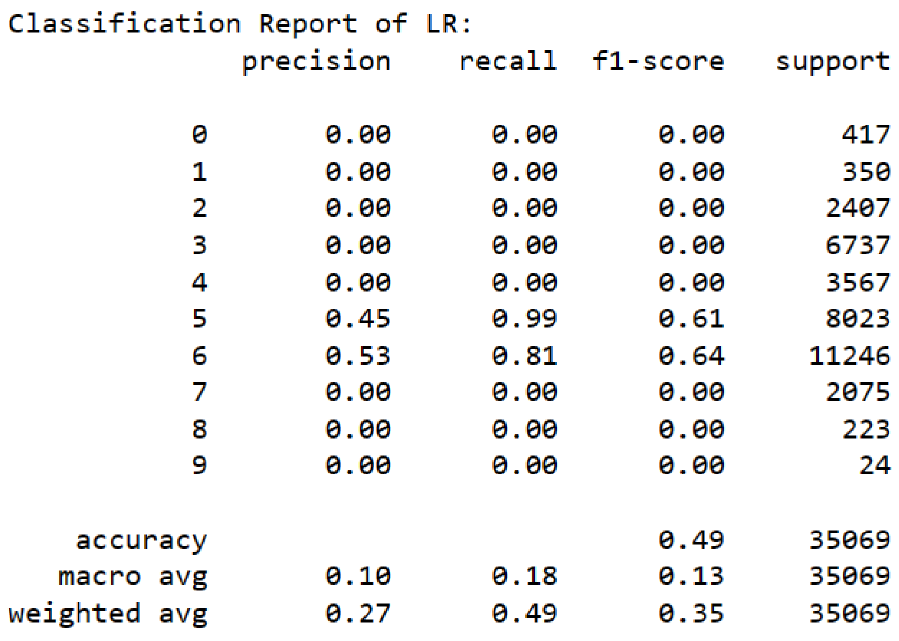
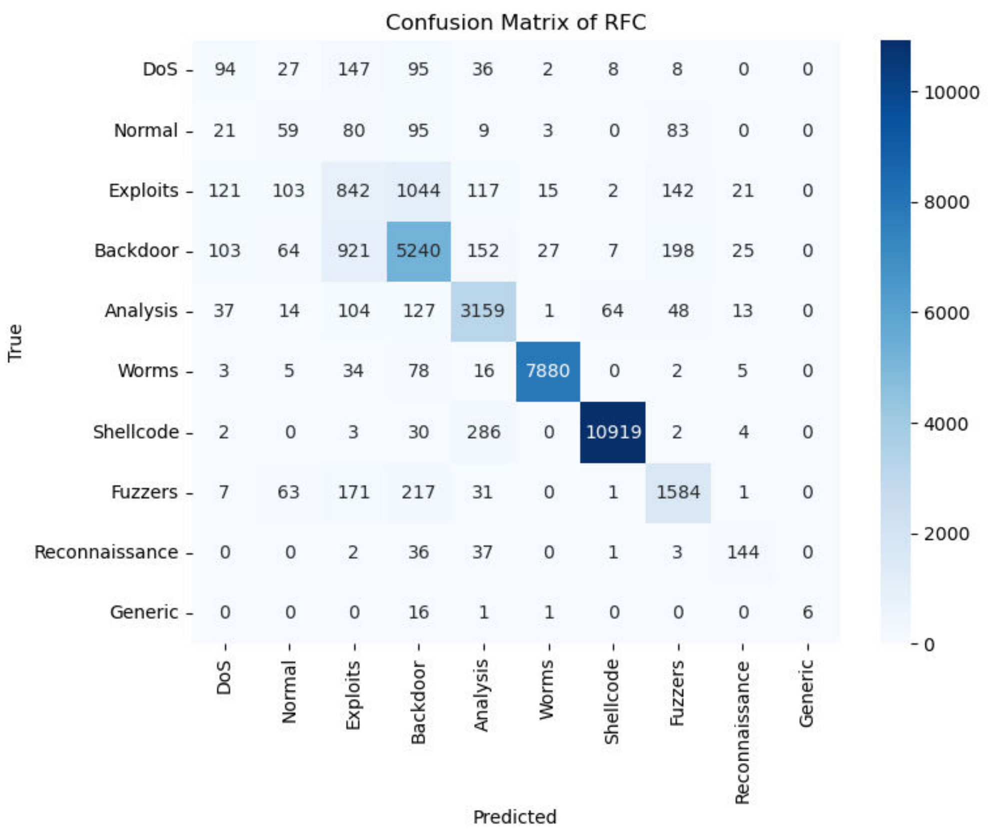
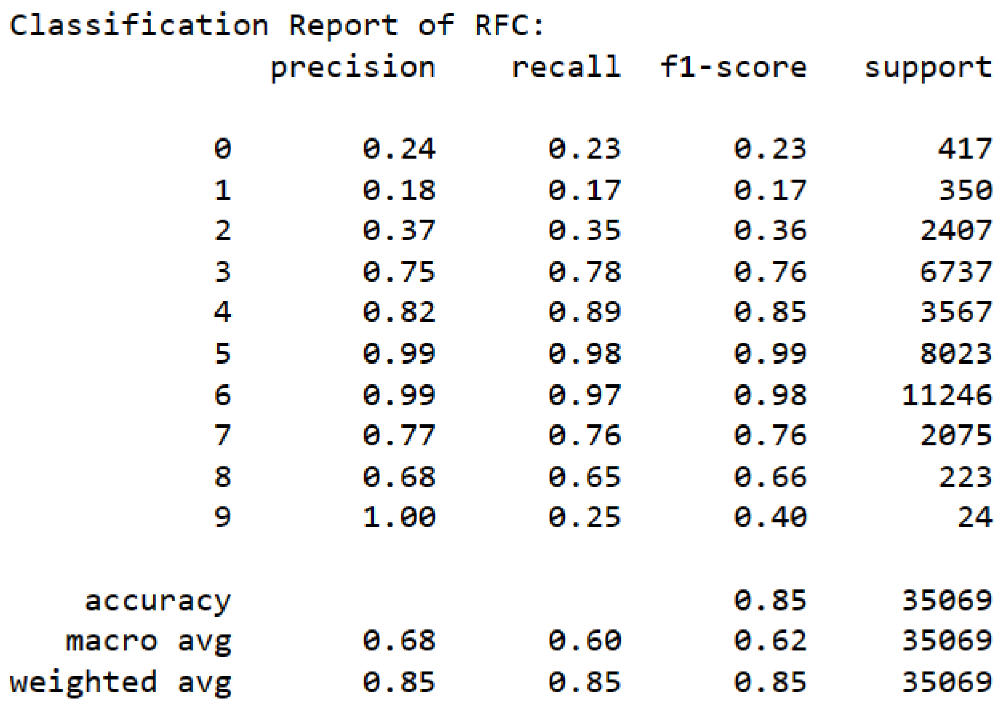

```
Classification Report of LR:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       417
           1       0.00      0.00      0.00       350
           2       0.00      0.00      0.00      2407
           3       0.00      0.00      0.00      6737
           4       0.00      0.00      0.00      3567
           5       0.45      0.99      0.61      8023
           6       0.53      0.81      0.64     11246
           7       0.00      0.00      0.00      2075
           8       0.00      0.00      0.00       223
           9       0.00      0.00      0.00        24

    accuracy                           0.49     35069
   macro avg       0.10      0.18      0.13     35069
weighted avg       0.27      0.49      0.35     35069
```

Fig. 8: Existing LRC classification report.



	DoS	Normal	Exploits	Backdoor	Analysis	Worms	Shellcode	Fuzzers	Reconnaissance	Generic
DoS	94	27	147	95	36	2	8	8	0	0
Normal	21	59	80	95	9	3	0	83	0	0
Exploits	121	103	842	1044	117	15	2	142	21	0
Backdoor	103	64	921	5240	152	27	7	198	25	0
Analysis	37	14	104	127	3159	1	64	48	13	0
Worms	3	5	34	78	16	7880	0	2	5	0
Shellcode	2	0	3	30	286	0	10919	2	4	0
Fuzzers	7	63	171	217	31	0	1	1584	1	0
Reconnaissance	0	0	2	36	37	0	1	3	144	0
Generic	0	0	0	16	1	1	0	0	0	6

Fig. 9: Proposed RFC confusion matrix.



```
Classification Report of RFC:
              precision    recall  f1-score   support

           0       0.24      0.23      0.23       417
           1       0.18      0.17      0.17       350
           2       0.37      0.35      0.36      2407
           3       0.75      0.78      0.76      6737
           4       0.82      0.89      0.85      3567
           5       0.99      0.98      0.99      8023
           6       0.99      0.97      0.98     11246
           7       0.77      0.76      0.76      2075
           8       0.68      0.65      0.66       223
           9       1.00      0.25      0.40        24

    accuracy                           0.85     35069
   macro avg       0.68      0.60      0.62     35069
weighted avg       0.85      0.85      0.85     35069
```

Fig. 10: Proposed RFC classification report.

5.CONCLUSION

The approach of using standard scaling and a RFC classifier for malware-attack detection is a promising one. It leverages advanced machine learning techniques to identify malicious activities in network traffic or system logs. By preprocessing the data effectively and training a RFC model, it is possible to achieve accurate and timely detection of cyber threats. However, it's important to note that the effectiveness of such a system depends on various factors, including the quality and diversity of the training data, the design of the RFC architecture, and the continuous monitoring and updating of the model.

REFERENCES

[1] S. Dange and M. Chatterjee, "Iot botnet: The largest threat to the iot network" in Data Communication and Networks, Cham, Switzerland:Springer, pp. 137-157, 2020.

[2] J. Ceron, K. Steding-Jessen, C. Hoepers, L. Granville and C. Margi, "Improving IoT botnet investigation using an adaptive network layer", Sensors, vol. 19, no. 3, pp. 727, Feb. 2019.

[3] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, et al., "N-baiot-network-based detection of iot botnet attacks using deep autoencoders", IEEE Pervas. Comput., vol. 17, no. 3, pp. 12-22, 2018.

[4] Shah, S.A.R.; Issac, B. Performance comparison of intrusion detection systems and application of machine learning to Snort system. Futur. Gener. Comput. Syst. 2018, 80, 157–170.

[5] Soe YN, Feng Y, Santosa PI, Hartanto R, Sakurai K. Machine Learning-Based IoT-Botnet Attack Detection with Sequential Architecture. Sensors. 2020; 20(16):4372. https://doi.org/10.3390/s20164372

[6] I. Ali et al., "Systematic Literature Review on IoT-Based Botnet Attack," in IEEE Access, vol. 8, pp. 212220-212232, 2020, doi: 10.1109/ACCESS.2020.3039985.

[7] Irfan, I. M. Wildani and I. N. Yulita, "Classifying botnet attack on Internet of Things device using random forest", IOP Conf. Ser. Earth Environ. Sci., vol. 248, Apr. 2019.

[8] Shah, T., Venkatesan, S. (2019). A Method to Secure IoT Devices Against Botnet Attacks. In: Issarny, V., Palanisamy, B., Zhang, LJ. (eds) Internet of Things – ICIOT 2019. ICIOT 2019. Lecture Notes in Computer Science(), vol 11519. Springer, Cham. https://doi.org/10.1007/978-3-030-23357-0_3

[9] C. Tzagkarakis, N. Petroulakis and S. Ioannidis, "Botnet Attack Detection at the IoT Edge Based on Sparse Representation," 2019 Global IoT Summit (GIoTS), Aarhus, Denmark, 2019, pp. 1-6, doi: 10.1109/GIOTS.2019.8766388.

[10] Y. Meidan et al., "N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders," in IEEE Pervasive Computing, vol. 17, no. 3, pp. 12-22, Jul.-Sep. 2018, doi: 10.1109/MPRV.2018.03367731.

[11] S. I. Popoola, R. Ande, B. Adebisi, G. Gui, M. Hammoudeh and O. Jogunola, "Federated Deep Learning for Zero-Day Botnet Attack Detection in IoT-Edge Devices," in IEEE Internet of Things Journal, vol. 9, no. 5, pp. 3930-3944, 1 March1, 2022, doi: 10.1109/JIOT.2021.3100755.

[12] F. Hussain et al., "A Two-Fold Machine Learning Approach to Prevent and Detect IoT Botnet Attacks," in IEEE Access, vol. 9, pp. 163412-163430, 2021, doi: 10.1109/ACCESS.2021.3131014.

[13] Abu Al-Haija Q, Al-Dala'ien M. ELBA-IoT: An Ensemble Learning Model for Botnet Attack Detection in IoT Networks. Journal of Sensor and Actuator Networks. 2022; 11(1):18. https://doi.org/10.3390/jsan11010018

[14] Alharbi A, Alosaimi W, Alyami H, Rauf HT, Damaševičius R. Botnet Attack Detection Using Local Global Best Bat Algorithm for Industrial Internet of Things. Electronics. 2021; 10(11):1341. https://doi.org/10.3390/electronics10111341

[15] Ahmed, A.A., Jabbar, W.A., Sadiq, A.S. et al. Deep learning-based classification model for botnet attack detection. J Ambient Intell Human Comput 13, 3457–3466 (2022). https://doi.org/10.1007/s12652-020-01848-9