

ANALYSIS AND DETECTION OF MALICIOUS IoT TRAFFIC

Advait Kunte, Gargi Prabhugaonkar, Tejal Bijwe, Keerthana Sambasivam

Guide: Dr. Yuan Cheng

CSC- 255 Computer Networks

FALL 2019

California State University, Sacramento

TABLE OF CONTENTS

1. Abstract.....	3
2. Introduction.....	3
3. Motivation.....	4
4. Related work	5
5. Datasets	6
5.1. Dataset evaluation	6
5.2. Dataset description	6
5.3. Dataset processing	7
6. Malicious IoT traffic detection using deep neural networks.....	9
6.1. Design	9
6.2. Implementation.....	10
6.3. Results.....	13
7. Firewall	16
7.1. Examples	17
8. Lessons learnt	22
9. Future work.....	22
10. Conclusion	22
11. References.....	23
12. Implementation Code	23

1. ABSTRACT

The acquisition of Internet of Things (IoT) technologies is increasing and thus IoT is shifting from hype to reality. The growth of the Internet of Things establishes various challenges in monitoring network growth in the market. Experts have enumerated several risks inflicted by IoT devices on organizations. Due to this issue, there is a requirement of an intelligent process that can detect the IoT devices trustworthy or not. We present a Deep-neural network technique to categorize the malicious IoT devices from the datasets. The input data preprocessing to the deep neural network is in the form of .csv file by converting the objects to float type. After processing the input to identify the best feature out of it, lasso regression is used to identify the true features of the output. Finally, test data is processed to repeat the data. The results are promising with an accuracy as high as 86.57% on the test set records. Additionally, we imposed a firewall to completely block the malicious IoT devices using pfsense.

Keywords: *Internet of Things, Deep neural network, firewall.*

2. INTRODUCTION

The Internet of Things is expanding worldwide, providing benefits in every aspect of our lives. At the same time, the IoT is also exposed to a wide range of security vulnerability issues.

Our project focuses on analyzing network traffic and segregating IoT devices from other devices. The next step is focusing on identifying malicious traffic from the extracted IoT devices. After detecting the malicious traffic from the IoT devices, firewall policies are built to strengthen network consisting of IoT devices. The main purpose of introducing the firewall policies is to block the IoT devices based on malicious traffic.

3. MOTIVATION

Anything connected to the internet is vulnerable to a multitude of cyber-attacks. According to research, most IoT devices are insecure. i.e. 48% of U.S Companies with IoT devices have experienced breaches. Likewise, there is no enough efforts are made towards securing these IoT devices cheap without security. This shows that IoT devices can be taken over to form botnets or to leak private data like ‘Mirai Botnet’.

Lucian Constantin. (2019) *Zscaler reports: CSO United states*, a network security firm, found that 40% of the IoT devices present on enterprise networks do not have their traffic encrypted. This traffic is susceptible to man-in-the-middle attacks (MitM) [1]. Due to which the attacker who has gained access to the local network can then compromise a local router. The main reason is to intercept the traffic to deliver malicious updates or steal data. Zscaler found that many IoT devices have security issues as they do not have automatic updates and these updates are required to be deployed manually. They have found around 6000 transactions per quarter have occurred because of the malware infections [2]. Additionally, the consumer-grade IoT devices and even the devices used in enterprise networks have very low or no security at all.

Therefore, the need of the hour is to have the IoT traffic be analyzed to scan any shadow devices and have the policy to prevent such devices from connecting to critical networks. The policy should prevent the network from being attacked even if a single device is compromised. Considering the current risk around IoT devices, we intend to devise a policy that would block malicious traffic into IoT devices to protect the network from being compromised.

4. RELATED WORK

4.1. A Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis

This paper contributes to identifying IoT devices by analyzing the traffic and classifying the data is the goal of this project. The method used in this paper is extracting the network features through the prefix of MAC addresses. This strategy is used to find the manufacturer of each device [3]. Our work complements these, as most are focused on analyzing the datasets, identifying the IoT traffic by classifying the network features. In contrast, we focus on the Firewall policy which can lead to blocking the malicious IoT traffic.

4.2. Classification of Application Traffic Using TensorFlow Machine Learning

Most recent work by Jee. Tae Park has closely examined the approach of using TensorFlow Machine Learning to classify IoT devices. The traffic classification system algorithms are as follows. The collection of application traffic is captured with MS Network Monitor 3.4 with the appropriate traffic. The next process is extracting all the learning features from the collected traffic. Then training all the features is done with test set and train set. Finally, it is normalized with the ML SoftMax regression [4]. After the classification, FloFex system structure which is used as the extraction system for learning feature. The implemented work only classifies the application traffic. But our work extends these results by developing the technique for identifying the IoT devices and feature extraction by MAC address.

5. DATASETS

Datasets are a collection of data representing the flow of network routing which corresponds with the feature. Features are the network model relevant property channel id, source IP, the weight of the stream, etc. These data set entity is defined in tabular form in which every column and row represents a particular attribute.

5.1. DATASET EVALUATION

For the dataset in this project, we use the UCI Machine Learning Repository for identifying and analyzing the IoT devices. For identifying and analyzing the data we used Machine learning. In that initially, we decided to use a pcap file which is API for captures network traffic.

In this, we decided to capture raw networking data from the typical network traffic flow. But there is one issue regarding the pcap files is that in machine learning we perform the operation only on file with extension .csv (comma-separated values) format. So, either converting pcap file to csv file was in option or directly accessing the csv file. As the solution to this problem UCI Machine Learning Repository provides csv dataset which we used in this project.

5.2. DATASET DESCRIPTION

The dataset which we used is based on the real traffic data which gathers from 8 commercial IoT devices. These 8 devices are specified in Fig.5.2 as mentioned all kinds of IoT devices are used in this project.

We used this dataset because our modeling corresponding to the neural network model and this dataset contains the feature need for that.



Fig 5.2. Devices used

5.3. DATASET PROCESSING

Given datasets were complex and vast, so we combine a certain amount of data from each dataset into one as mentioned in Fig 5.3.a.

		Train Set			Test Set				
Devices			Benign	Malicious			Benign	Malicious	
				Bashlite	Mirai			Bashlite	Mirai
1	Danmini_Doorbell/	Y	10000	5000	5000	N			
2	Ecobee_Thermostat/	Y	10000	5000	5000	N			
3	Philips_B120N10_Baby_Monitor/	Y	10000	5000	5000	N			
4	Provision_PT_737E_Security_Camera/	Y	10000	5000	5000	N			
5	Samsung_SNH_1011_N_Webcam/	Y	10000	10000	0	N			
6	Provision_PT_838_Security_Camera/	Y	10000	5000	5000	N			
7	SimpleHome_XCS7_1002_WHT_Security_Camera/	N				Y	10000	5000	5000
8	SimpleHome_XCS7_1003_WHT_Security_Camera/	N				Y	10000	5000	5000
		120000	60000	35000	25000	40000	20000	10000	10000

Fig 5.3.a: Numeric representation of data taken from each device.

We distinguish benign data with malicious data, in that two types of attacks are defined Bashlite and Mirai.

Bashlite attack infects Linux systems for launching DDoS attack which contains features like scan-Scanning the network for vulnerable devices, Junk-which sends spam data, UDP-UDP flooding, TCP-TCP flooding and COMBO-that sends spam data and opening a connection to a specified IP address and port. Mirai attack basically targets IoT devices by attacking its source code. As in Bashlite this also contains features as scan, Ack-Ack flooding, Syn-Syn flooding, UDP, UDP plain: UDP flooding with fewer options, optimizing for higher PPS.

In the dataset, the total number of rows is 150000 and feature 116. In which we trained at 120000 i.e 70% of the dataset which is 6 devices specifies in Fig 1 and test on 40000 data i.e 30% which are 2 devices same as specified in Fig 1 of the total dataset. But at the end of processing, only 6 features are selected that are closes to the output. This feature is described in Fig 5.3.b.

	HH_L3_weight	MI_dir_L1_mean	HH_L1_std	HH_jit_L3_weight	HH_L0.1_mean	HH_L3_std
0	1.000000	60.000000	0.000000	1.000000	60.00000	0.000000
1	1.000032	354.000000	2.072641	1.000032	346.61980	0.067642
2	1.912156	360.091968	6.356005	1.912156	352.01884	5.994038
3	1.000000	337.000000	0.000000	1.000000	337.00000	0.000000
4	1.000000	193.165753	0.000000	1.000000	60.00000	0.000000

Fig 5.3.b.: Features used

As per the meaning of the feature following are the description of the dataset features:

H: Specifies Source IP

MI: Specifies Source MAC-IP

HH: Defines Channel where data is operating

HH_jit: Channel jitter summarize the jitter of the traffic from source IP to destination IP

L0.1, L1, L3: The statistics extracted from the packet stream

weight: the weight of the stream

6. MALICIOUS IOT TRAFFIC DETECTION USING DEEP NEURAL NETWORK

Machine learning techniques can use patterns available in data to train systems. It allows building learning models that can make decisions independently. By iteratively exposing these models to new data, they adapt to generate results that are greater in accuracy and precision [5].

Deep learning is a type of machine learning method based on artificial neural networks. A deep neural network is a type of deep learning architecture that constitutes of interconnected nodes. These nodes function like the neurons present in the human brain. They recognize hidden patterns in data to identify anomalies and learn by continuous improvement. [5, 6].

A deep neural network consists of an input layer, intermediate hidden layers, and an output layer. The input layer accepts the data to pass it to the hidden layers. The nodes in these layers connect through a system of weighted connections. The activation functions use the products of weight and input to determine the progress of the signal to generate the outcome from the output layer. The hidden layers process the data by assigning weights to the incoming coefficient data values. The output layer allows for obtaining the result [5].

6.1. DESIGN

The dataset shared by UCI repository reflected patterns between attack and benign data [8]. Therefore, we decided to implement a deep neural network to identify attack data with high accuracy. The idea was to train the machine learning model and test this model by using a test set created after preprocessing the existing datasets provided by the UCI machine learning repository [8]. We decided to select IoT traffic from different IoT devices to train the model on a diverse dataset.

We planned to train the machine learning model based on a limited set of features that were closer to the final desired outcome- to determine if data is malicious or not. The available dataset for attack and benign data contained values that spread over a wide range. To identify the best features to train the machine learning model, those features which would help the model to identify attack data more accurately had to be determined.

Shrinking values could help us identify a few features that we intended to train our machine learning model [9]. Shrinkage allows shrinking extreme values towards the mean. By identifying benign data as '0' and attack data as '1', we planned to identify those features that had values closer to '0' and '1'.

Lasso Regression is a type of linear regression that shrinks data values towards the mean value [9]. It is a commonly used technique for variable selection or parameter elimination. We decided to use this technique to identify and select only those features that were closer to the output. Training a deep neural network using features identified by Lasso regression helps to reduce model complexity and reduces over-fitting issues [10]. Over-fitting means that the learning model also learns the noise in the data set. This issue can impact the performance of the machine learning model. Therefore, feature selection is a crucial step in the process.

As a final step, we planned to use the selected features from the training data set to train the machine learning model. To test this model, we decided to create a test data set containing data from various IoT devices. After pre-processing the test data set like the train data set, we decided to test the designed model for accuracy. Determining the iteration cycles for deep neural networks is crucial. Fewer cycles can underfit the test and train data set. Too many iterations can degrade the performance caused by overfitting [12]. Early stopping is used to resolve this issue. Therefore, we decided to train the model only to the point before the performance degrades.

6.2. IMPLEMENTATION

The implementation of the Malicious Traffic Detection Model using Deep Neural consists of three main steps as depicted below-

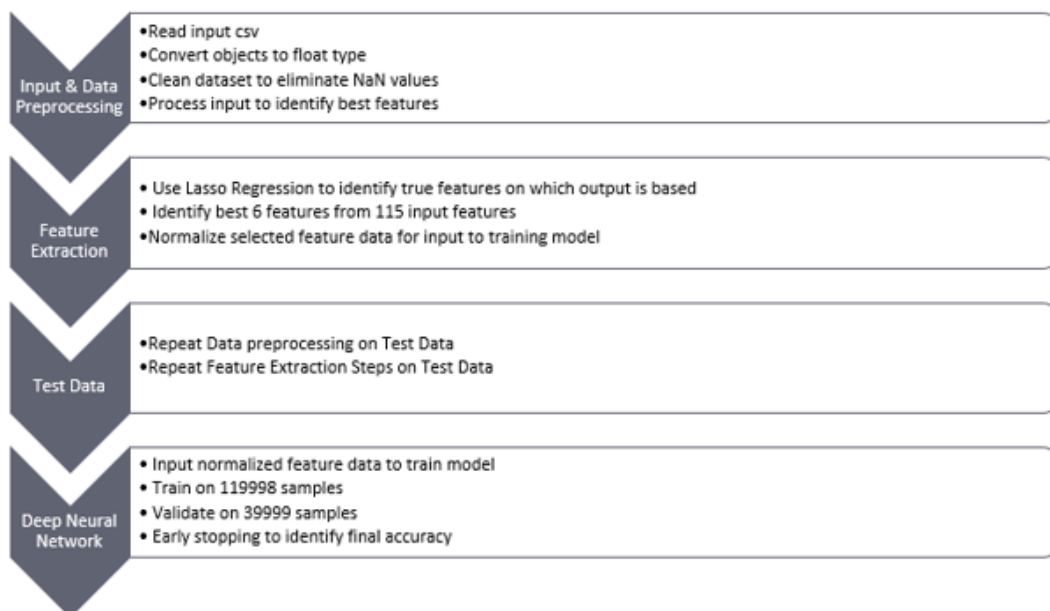


Fig 6.2.a: Implementation steps for Malicious IoT Traffic Detection using Deep Neural Networks

6.2.1. INPUT DATA PRE-PROCESSING

The input to the deep neural network consists of two .csv files- Train.csv containing over 120,000 rows and Test.csv containing 40,000 rows of malicious and benign IoT traffic data. Our preprocessing constituted converting the Object data type to Float data type, eliminating records of data containing non-numerical values. It is a crucial step, as the accuracy of the model depends on the quality of data.

Data-Preprocessing

```
data = data.apply(pd.to_numeric, errors='coerce') # converted object to floats.

#check for nan,infinite values
def clean_datasets(dframe):
    assert isinstance(dframe, pd.DataFrame), ""
    dframe.dropna(inplace=True)
    indices_to_keep = ~dframe.isin([np.nan, np.inf, -np.inf]).any(1)
    return dframe[indices_to_keep].astype(np.float64)

import numpy as np

data = clean_datasets(data)
```

Fig 6.2.1.a: Steps to clean data sets

6.2.2. FEATURE EXTRACTION USING LASSO REGRESSION

As described earlier, we used Lasso Regression to shrink and select values that were closet to the desired output of '0' indicating benign data and '1' indicating attack data.

```
%matplotlib inline
from IPython.display import display

def report_coef_value(names,coef,intercept):
    r = pd.DataFrame( { 'coef': coef, 'positive': coef>=0 }, index = names )
    r = r.sort_values(by=['coef'])
    display(r)
    print("Intercept value is: {}".format(intercept))
    r['coef'].plot(figsize=(20,10),kind='barh', color=r['positive'].map({True: 'b', False: 'r'}))

import numpy as np
from sklearn import metrics
import sklearn
from sklearn.linear_model import Lasso

# Create linear regression
ls_regressor = Lasso(alpha=0.1)

# Fit/train LASSO
ls_regressor.fit(xtrain,ytrain)
# Predict
pred = ls_regressor.predict(xtest)

# Measure RMSE error. RMSE is common for regression.
score = np.sqrt(metrics.mean_squared_error(pred,ytest))
print("Final score (RMSE): {}".format(score))

names = list(data.columns.values)

report_coef_value(
    names,
    ls_regressor.coef_,
    ls_regressor.intercept_)
```

Fig 6.2.2.a: Steps to clean data sets

6.2.3. TEST DATA PRE-PROCESSING

After selecting the desired features identified by Lasso regression, we repeat the same steps on the test data set as the input to the machine learning model needs to be consistent. Additionally, as a final step, data from both the files need to be Normalized before training and testing the deep neural network.

```
newdata = data[['HH_L3_weight', 'MI_dir_L1_mean', 'HH_L1_std', 'HH_jit_L3_weight', 'HH_L0.1_mean', 'HH_L3_std']]

newdata.head()
```

	HH_L3_weight	MI_dir_L1_mean	HH_L1_std	HH_jit_L3_weight	HH_L0.1_mean	HH_L3_std
0	1.000000	80.000000	0.000000	1.000000	80.00000	0.000000
1	1.000032	354.000000	2.072841	1.000032	348.81980	0.087642
2	1.912158	380.091988	8.358005	1.912158	352.01884	5.994038
3	1.000000	337.000000	0.000000	1.000000	337.00000	0.000000
4	1.000000	193.185753	0.000000	1.000000	80.00000	0.000000

```
nor_xtrain = StandardScaler()
x_train = nor_xtrain.fit_transform(newdata.iloc[:, :].values)

tdata=tdata[['HH_L3_weight', 'MI_dir_L1_mean', 'HH_L1_std', 'HH_jit_L3_weight', 'HH_L0.1_mean', 'HH_L3_std']]
```

Fig 6.2.3.a: Normalize input data to Deep Neural Network

6.2.4. DEEP NEURAL NETWORK IMPLEMENTATION

Finally, the steps indicated below describe the steps that build the desired deep neural network model consisting of three hidden layers and their respective activation functions. We use early stopping to train the model until the accuracy begins to stay constant.

Using Tensorflow for fully connected neural network

```
from sklearn import metrics
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint

model = Sequential()

#Deep Neural network
model.add(Dense(3, input_dim = 6, activation='relu')) #hidden 1
model.add(Dense(3, activation='relu')) #hidden 2
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics = ['accuracy'])
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')

x_test=np.asarray(x_test)

x_train=np.asarray(x_train)

ytrain= np.asarray(ytrain)

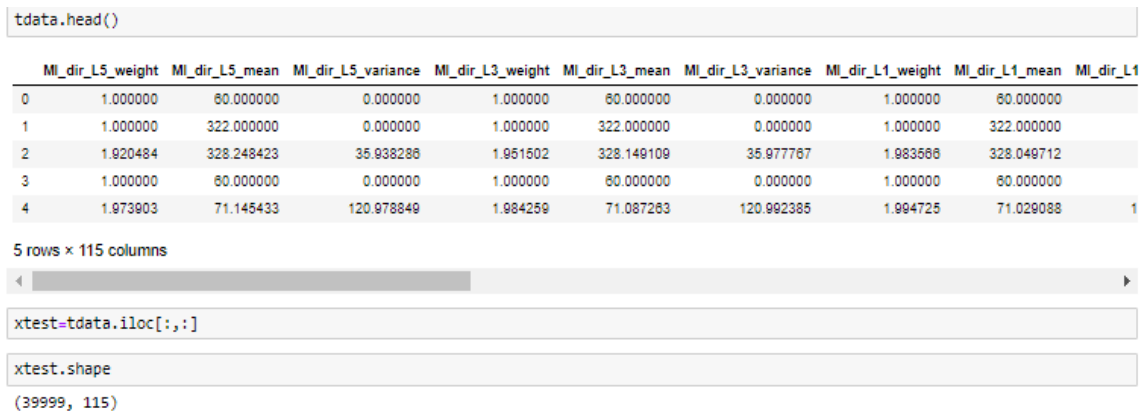
ytest=np.asarray(ytest)

import os
model.fit(x_train, ytrain, validation_data=(x_test,ytest), callbacks=[monitor], verbose=2, epochs=10)
```

Fig 6.2.4.a: Deep Neural Network Implementation

6.3. RESULTS

By implementing the function to clean the data for eliminating non-numerical values, we were able to obtain 119,998 rows containing clean data from 120,000 initial data rows. Similarly, cleaning the test data set using same operation, the outcome consisted of 39,999 clean test data rows from 40,000 rows from original test file.



```
tdata.head()
```

	MI_dir_L5_weight	MI_dir_L5_mean	MI_dir_L5_variance	MI_dir_L3_weight	MI_dir_L3_mean	MI_dir_L3_variance	MI_dir_L1_weight	MI_dir_L1_mean	MI_dir_L1
0	1.000000	80.000000	0.000000	1.000000	80.000000	0.000000	1.000000	80.000000	
1	1.000000	322.000000	0.000000	1.000000	322.000000	0.000000	1.000000	322.000000	
2	1.920484	328.248423	35.938286	1.951502	328.149109	35.977767	1.983566	328.049712	
3	1.000000	80.000000	0.000000	1.000000	80.000000	0.000000	1.000000	80.000000	
4	1.973903	71.145433	120.978849	1.984259	71.087263	120.992385	1.994725	71.029088	1

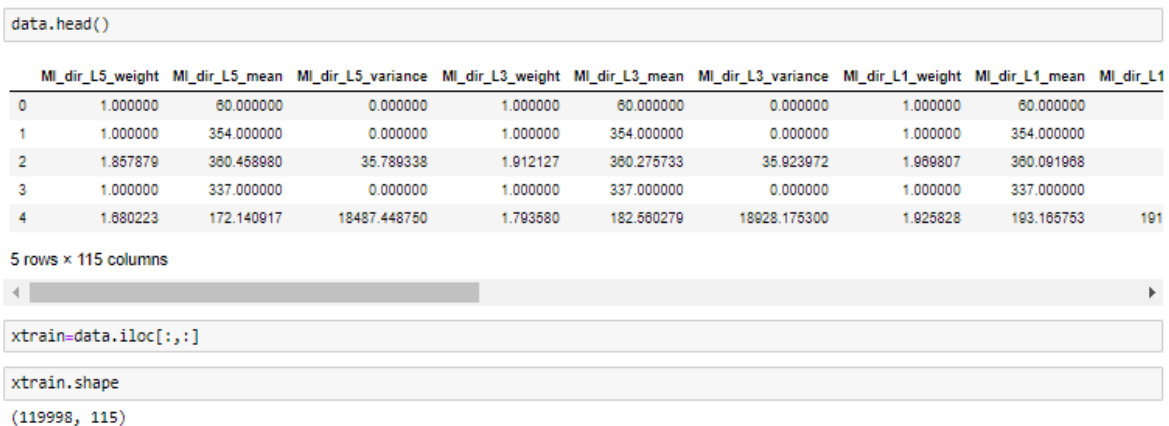
5 rows × 115 columns

```
xtest=tdata.iloc[:,:]
```

```
xtest.shape
```

(39999, 115)

Fig 6.3.a: Result: Cleaning Train Data



```
data.head()
```

	MI_dir_L5_weight	MI_dir_L5_mean	MI_dir_L5_variance	MI_dir_L3_weight	MI_dir_L3_mean	MI_dir_L3_variance	MI_dir_L1_weight	MI_dir_L1_mean	MI_dir_L1
0	1.000000	80.000000	0.000000	1.000000	80.000000	0.000000	1.000000	80.000000	
1	1.000000	354.000000	0.000000	1.000000	354.000000	0.000000	1.000000	354.000000	
2	1.857879	380.458980	35.789338	1.912127	380.275733	35.923972	1.969807	380.091968	
3	1.000000	337.000000	0.000000	1.000000	337.000000	0.000000	1.000000	337.000000	
4	1.680223	172.140917	18487.448750	1.793580	182.560279	18928.175300	1.925828	193.185753	191

5 rows × 115 columns

```
xtrain=data.iloc[:,:]
```

```
xtrain.shape
```

(119998, 115)

Fig 6.3.b: Result: Cleaning Test Data

The output from Lasso Regression helped to identify 6 best features that were closer to the desired value of 0 and 1. We selected three features closer to 0 and three closer to 1.

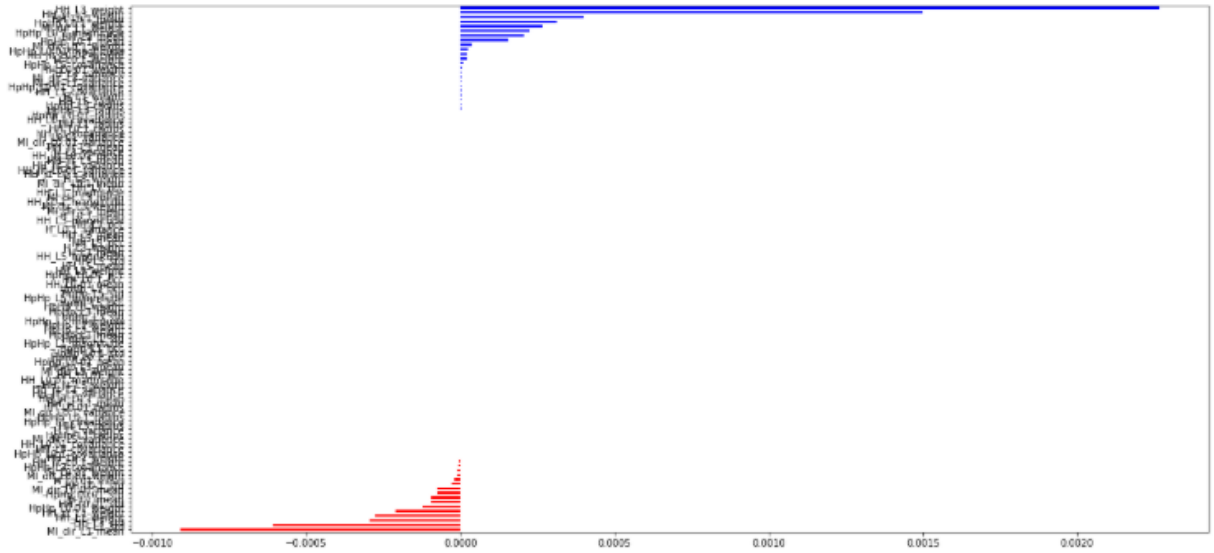


Fig 6.3.c: Lasso Regression to select best features closer to desired outcome

We selected 6 best features for our train and test data set. The final input dimensions of train data set were - [119,998 X 6] and test data set matrix- [39,999 X 6]. The input was normalized before training the machine learning model.

```
tdata=tdata[['HH_L3_weight', 'MI_dir_L1_mean', 'HH_L1_std', 'HH_jit_L3_weight', 'HH_L0.1_mean', 'HH_L3_std']]
```

```
tdata.head()
```

	HH_L3_weight	MI_dir_L1_mean	HH_L1_std	HH_jit_L3_weight	HH_L0.1_mean	HH_L3_std
0	1.000000	60.000000	0.000000	1.000000	60.000000	0.000000
1	1.000058	322.000000	3.789325	1.000058	333.059988	0.152442
2	1.951558	328.049712	6.238371	1.951558	333.350705	5.998515
3	1.000000	60.000000	0.000000	1.000000	60.000000	0.000000
4	1.000000	71.029088	0.000000	1.000000	82.000000	0.000000

```
nor_xtest = StandardScaler()
x_test = nor_xtest.fit_transform(tdata.iloc[:, :].values)
```

Fig 6.3.d: Dimensions of the final matrix before Normalization

```

pred = model.predict(x_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ytest, pred.round())
cm

array([[15356, 4644],
       [ 724, 19275]], dtype=int64)

from sklearn.metrics import accuracy_score
print("Accuracy: ", accuracy_score(ytest, pred.round()))

Accuracy: 0.8657966449161228

from sklearn.metrics import classification_report
print(classification_report(ytest, pred.round()))

```

	precision	recall	f1-score	support
0.0	0.95	0.77	0.85	20000
1.0	0.81	0.96	0.88	19999
accuracy			0.87	39999
macro avg	0.88	0.87	0.86	39999
weighted avg	0.88	0.87	0.86	39999

Fig 6.3.e: Final Output of Deep Neural Network Model

After training the model using the pre-processed and normalized train and test data respectively, it identified 19,275 records as attack data from 20,000 records present in the test data set. Similarly, 19,275 records were identified as benign from 20,000 benign records. However, 724 records were falsely identified as attack and 4,644 attack records were identified as benign.

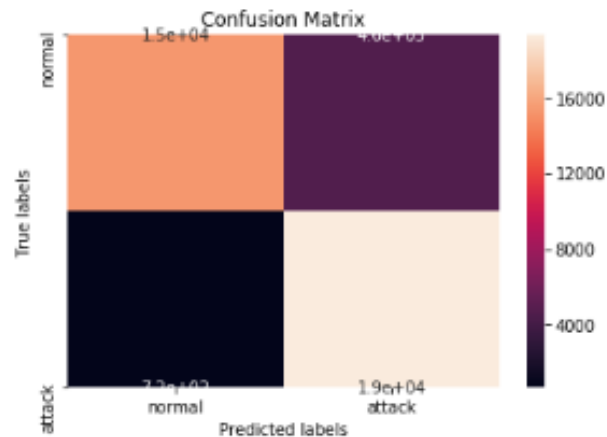


Fig 6.3.f: Confusion Matrix describing the output

The final accuracy of the model after early stopping at the 6th Epoch was determined to be 86.57%.

7. FIREWALL

As a basic level security measure, we are using a firewall with two subnets. The subnets are not connected initially. The first subnet is dedicated to regular devices like desktop, laptops, etc. The second subnet is where all the IoT devices are added. When we categorize the devices as IoT or non-IoT devices, we add them to their respective subnets. The network architecture is shown in Fig 7.1. Using the pfsense[13] firewall, we connect the two subnets and WAN using these specific rules

- Allow traffic from LAN1 to WAN
- Allow traffic from IoT1 to WAN
- Allow traffic from LAN1 to IoT1
- Reject traffic from IoT1 to LAN1
- Allow traffic from LAN1 to LAN1
- Reject traffic from IoT1 to IoT1

These rules allow both the LAN1 network devices and the IoT1 network devices to connect to the internet. Generally, the IoT devices do not need to locally (internal network) initiate a connection to other devices. They keep their ports open for other devices to connect to them. Therefore, we do not allow IoT1 network devices to connect to any local devices, either from the LAN1 network or from the IoT1 network. LAN1 network can connect locally to other devices, either to LAN1 or to the IoT1 network. There are two reject rules, which drop packets originating from IoT1 and destination being LAN1 or IoT1. This set of rules set up a network for basic security against malicious IoT devices on the same network. This rule also makes sure that IoT devices continue to function as expected.

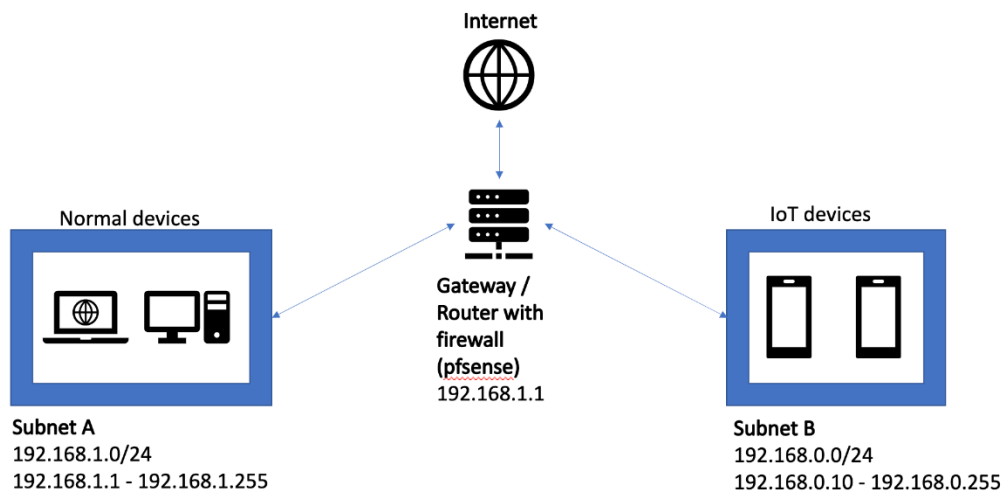


Fig 7.1. Network architecture

7.1 EXAMPLES

The firewall rules are shown in the screenshot below (Fig 7.1.a)

	States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description
✓	0 / 0 B	*	*	*	LAN Address	80	*	*		Anti-Lockout Rule
✓ ⚙️	2 / 122.97 GiB	IPv4 *	192.168.1.0/24	*	*	*	LoadBalancing	none		LAN to Any
👉	0 / 7.21 MiB	IPv4 *	IoT_Devices	*	LAN address	*	*	none		IoT to LAN
✓ ⚙️	0 / 10.35 GiB	IPv4 *	IoT_Devices	*	*	*	BSNL_PRIORITY	none		IoT to Internet

Fig 7.1.a Pfsense firewall rules

The first rule is an inbuilt rule, to make sure that the user doesn't lock themselves out of the firewall. The second rule allows LAN network to access any network. The third rule drops all packets from IoT network to the LAN network. The fourth rule allows access to the internet for the IoT network.

The Fig 7.1.a shows the devices on the network. We can see the regular devices like laptops and cellphones are connected to the LAN network and the IoT devices like the Chromecast and Raspberry Pi are connected to the IoT network.

Status / DHCP Leases									
Leases									
	IP address	MAC address	Client Id	Hostname	Description	Start	End	Online	Lease Type
⊕	192.168.1.107	34		iPhone-5S		2019/12/01 22:31:51	2019/12/02 22:31:51	online	active
⊕	192.168.1.120	90		iPhone		2019/12/01 20:17:30	2019/12/02 20:17:30	offline	active
⊕	192.168.1.105	24		Linksys		2019/12/02 02:08:44	2019/12/02 14:08:44	online	active
⊕	192.168.1.121	54		iPhone		2019/12/01 13:38:38	2019/12/02 13:38:38	offline	active
⊕	192.168.1.132	24		Linksys		2019/12/01 23:36:06	2019/12/02 11:36:06	online	active
⊕	192.168.1.108	80				2019/12/01 22:30:34	2019/12/02 10:30:34	offline	active
⊕	192.168.1.109	94		OnePlus3		2019/12/01 20:17:57	2019/12/02 08:17:57	offline	active
⊕	192.168.1.128	30				2019/12/01 18:17:04	2019/12/02 06:17:04	offline	active
⊕	192.168.1.123	a0				2019/12/01 15:01:59	2019/12/02 03:01:59	offline	active
👤	192.168.0.11	b8	raspberrypi	raspberrypi	raspberrypi	n/a	n/a	online	static
👤	192.168.0.21	38	Chromecast	Chromecast	Chromecast	n/a	n/a	offline	static
👤	192.168.1.30	64	_OnePlus_6	_OnePlus_6	_OnePlus_6	n/a	n/a	offline	static
👤	192.168.1.31	98	Advait-MBP	Advait-MBP	Advait-MBP	n/a	n/a	offline	static

Fig 7.1.b

To verify the rules, we run a series of ping test from various hosts.

7.1.1. EXAMPLE 1 (IOT1 TO LAN1)

```

root@raspbx:~# ifconfig eth0 && ping 192.168.0.1 -c 5
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.11 netmask 255.255.254.0 broadcast 192.168.1.255
    inet6 fe80::acbc:3aff:71a:60f7 prefixlen 64 scopeid 0x20<link>
    ether b8:27:eb:4f:01:21 txqueuelen 1000 (Ethernet)
    RX packets 436585 bytes 56456839 (53.8 MiB)
    RX errors 0 dropped 5026 overruns 0 frame 0
    TX packets 80480 bytes 21162159 (20.1 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.

--- 192.168.0.1 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4180ms

root@raspbx:~#
```

Fig 7.1.1

As we see the screenshot in Fig 7.1.1, the raspberry pi pings the pfsense router (which is on LAN network) and all the packets are dropped.

7.1.2. EXAMPLE 2 (LAN1 TO LAN1)

```
[2.4.4-RELEASE][admin@pfSense.localdomain]/root: ping -c 5 192.168.1.105
PING 192.168.1.105 (192.168.1.105): 56 data bytes
64 bytes from 192.168.1.105: icmp_seq=0 ttl=64 time=0.447 ms
64 bytes from 192.168.1.105: icmp_seq=1 ttl=64 time=0.348 ms
64 bytes from 192.168.1.105: icmp_seq=2 ttl=64 time=0.356 ms
64 bytes from 192.168.1.105: icmp_seq=3 ttl=64 time=0.357 ms
64 bytes from 192.168.1.105: icmp_seq=4 ttl=64 time=0.358 ms

--- 192.168.1.105 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.348/0.373/0.447/0.037 ms
```

Fig 7.1.2

In the screenshot Fig. 7.1.2, the pfSense router pings another host on the LAN network and gets a response.

7.1.3. EXAMPLE 3 (LAN1 TO IOT1)

```
[2.4.4-RELEASE][admin@pfSense.localdomain]/root: ping -c 5 192.168.0.11
PING 192.168.0.11 (192.168.0.11): 56 data bytes
64 bytes from 192.168.0.11: icmp_seq=0 ttl=64 time=0.781 ms
64 bytes from 192.168.0.11: icmp_seq=1 ttl=64 time=0.742 ms
64 bytes from 192.168.0.11: icmp_seq=2 ttl=64 time=0.731 ms
64 bytes from 192.168.0.11: icmp_seq=3 ttl=64 time=0.749 ms
64 bytes from 192.168.0.11: icmp_seq=4 ttl=64 time=0.741 ms

--- 192.168.0.11 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.731/0.749/0.781/0.017 ms
```

Fig 7.1.3

In fig 7.1.3 We ping the raspberry pi from the pfSense router and get a success response.

7.1.4. EXAMPLE 4 (IOT1 TO WAN/INTERNET)

```
root@raspbx:~# ifconfig eth0 && ping 8.8.8.8 -c 5
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.11 netmask 255.255.254.0 broadcast 192.168.1.255
    inet6 fe80::acbc:3aff:71a:60f7 prefixlen 64 scopeid 0x20<link>
    ether b8:27:eb:4f:01:21 txqueuelen 1000 (Ethernet)
    RX packets 437728 bytes 56565684 (53.9 MiB)
    RX errors 0 dropped 5026 overruns 0 frame 0
    TX packets 80828 bytes 21206156 (20.2 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=53 time=12.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=53 time=11.9 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=53 time=12.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=53 time=12.0 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=53 time=11.9 ms

--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 11.900/12.064/12.301/0.192 ms
```

Fig 7.1.4

In Fig 7.1.4 we see that the raspberry pi can connect to the internet by pinging the Google's DNS server.

7.1.5. EXAMPLE 5 (LAN1 TO WAN/INTERNET)

```
[2.4.4-RELEASE][admin@pfSense.localdomain]/root: ping -c 5 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=58 time=7.043 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=58 time=7.047 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=58 time=7.257 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=58 time=7.259 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=58 time=7.085 ms

--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 7.043/7.138/7.259/0.099 ms
```

Fig 7.1.5

Finally, we see that the pfsense router can access the internet.

These examples show us that the IoT devices continue to function normally, and the network is a bit more secure, should any IoT devices start to misbehave, they would not be able to attack any devices in the local network.

8. LESSONS LEARNT

The major limitation of using datasets of network captures differs from using the real devices. One of the main challenges that are to categorize the IoT devices based on the vendors. The major lesson learned from the MAC address prefix and OUI lookup tool is the datasets gives only the Manufacturer. The vendor categorization cannot be done using the MAC address. Using a network consisting of a non-IoT device would make the detection model weak. Hence, we used dataset facts to filter out non-IoT devices. Identifying IoT devices based on protocols is not a full-proof method of categorization.

9. FUTURE WORK

Although the provided methodologies are good, we worked mainly on the datasets that capture the network. Hence the real-time analysis is required to improve the detection of traffic. The main future work of this project will work with the real devices and classify the malicious IoT devices from the datasets. The accuracy is less due to the limitations in the datasets we used. We implemented the datasets available online and used it for data pre-processing from the .csv file. In the future, the real dataset live network can enhance the performance and its accuracy level.

10. CONCLUSION

The malicious IoT traffic detection technique was successfully implemented with an accuracy of 86.57%. We were also able to successfully verify the devised firewall policy.

11. REFERENCES

- [1] <https://www.csoonline.com/article/3397044/over-90-of-data-transactions-on-iot-devices-are-unencrypted.html>
- [2] <https://www.zscaler.com/blogs/research/iot-traffic-enterprise-rising-so-are-threats>
- [3] https://cyber.bgu.ac.il/wp-content/uploads/2017/10/Meidan-et-al-2017_Profiliot-A-Machine-Learning-Approach-for-IoT-Device-Identification-Based-on-Network-Traffic-Analysis.pdf
- [4] <https://www.semanticscholar.org/paper/Classification-of-application-traffic-using-machine-Park-Shim/de0acbd3f7882a2479310050a5af4cf3f94ca4ba>
- [5] https://www.sas.com/en_us/insights/analytics/machine-learning.html
- [6] https://en.wikipedia.org/wiki/Deep_learning#Deep_neural_networks
- [7] <https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/anatomy>
- [8] <https://archive.ics.uci.edu/ml/machine-learning-databases/00442/>
- [9] <https://www.statisticshowto.datasciencecentral.com/shrinkage-estimator/>
- [10] <https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b>
- [11] <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>
- [12] <https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/>
- [13] pfsense - <https://www.pfsense.org/>
- [14] Separating IoT devices from main LAN
https://www.reddit.com/r/PFSENSE/comments/b4yfpq/advice_needed_separating_iot_devices_from_main_lan/
- [15] pfSense Firewall rules for IoT devices -
https://www.reddit.com/r/PFSENSE/comments/8dhu4t/pfsense_firewall_rules_for_iot_devices/

12. IMPLEMENTATION CODE

<https://github.com/Gargipg27/MaliciousTrafficDetection>