Data Science Project Report :: 1

DataCraft Trio: Debnath Kundu (MT22026), Pijush Bhuyan (MT22049), Snehal Buldeo (MT22074)

DATASET DESCRIPTION

This public dataset contains real-time network traffic data recorded from nine commercial IOT devices that were infected with common botnet malware - BASHLITE and MIRAI to carry out ten types of network-based attacks.

Dataset URL - UCI Repository [detection_of_IoT_botnet_attacks_N_BaIoT]

Types of IOT devices - Thermostat, Baby Monitor, Webcam, Doorbells, and Security Cameras.

Types of Attacks -

1. **BASHLITE MALWARE -**
    a. **Scan** - Scanning the network for vulnerable devices
    b. **Junk** - Sending spam data packets
    c. **UDP** - Flooding the network with UDP packets
    d. **TCP** - Flooding the network with TCP packets
    e. **Combo** - Sending spam data and opening a connection to a specified IP address and port
2. **MIRAI MALWARE -**
    a. **Scan** - Automatic scanning for vulnerable devices
    b. **Ack** - Flooding the network with Ack packets
    c. **Syn** - Flooding the network with Syn packets
    d. **UDP** - Flooding the network with UDP packets
    e. **UDP Plain** - UDP flooding with fewer options, optimized for higher PPS

Dataset Size -

Device Name	Device Type	Benign Traffic Instances	BASHLITE Traffic Instances					MIRAI Traffic Instances				
			Scan	Junk	UDP	TCP	Combo	Scan	Ack	Syn	UDP	UDP Plain
Damini	Doorbell	49548	29849	29068	105874	92141	59718	107685	102195	122573	237665	81982
Ennio	Doorbell	39100	28120	29797	103933	101536	53014	-	-	-	-	-
Ecobee	Thermostat	13113	27494	30312	104791	95021	53012	43192	113285	116807	151481	87368
Philips B120N/10	Baby Monitor	175240	27859	28349	105782	92581	58152	103621	91123	118128	217034	80808

Provision PT-737 E	Security Camera	62154	29297	30898	104011	104510	61380	96781	60554	65746	156248	56681
Provision PT-838	Security Camera	98514	28397	29068	104658	89387	57530	97096	57997	61851	158608	53785
Simple Home XCS7-1002-WHT	Security Camera	46585	27825	28579	103720	88816	54283	45930	111480	125715	151879	78244
Simple Home XCS7-1003-WHT	Security Camera	19528	28572	27413	102980	98075	59398	43674	107187	122479	157084	84436
Samsung SNH 1011 N	Webcam	52150	27698	28305	110617	97783	58669	-	-	-	-	-
Total Instances		555932	255111	261789	946366	859850	515156	537979	643821	733299	1229999	523304

Total Benign Instances	=	555,932	(0.55M)
Total BASHLITE Instances	=	2,838,272	(2.84M)
Total MIRAI Instances	=	3,668,402	(3.67M)
Total Instances	**=**	**7,062,606**	**(7.06M)**

<u>Dataset Features</u> -

This dataset contains **115 network traffic features** over a temporal window of varying context sizes which capture the behavioural snapshot af the hosts and protocols that communicated with each packet. These features are captured as recording **23 standard features** over **5 varying sizes of time windows** of the most recent 100ms, 500ms, 1.5sec, 10sec, and 1min context. These 23 features can be grouped into 5 categories as illustrated in the table below -

Feature Category	**Statistical Feature**	**Aggregated by**	**Total No of features**
Packet size (outbound only)	Mean, Variance	Source IP, Source MAC-IP,Channel, Socket	8
Packet count	Number	Source IP, Source MAC-IP,Channel, Socket	4
Packet jitter (time difference between packet arrivals)	Number, Mean, Variance	Channel	3
Packet size (both outbound and inbound)	Magnitude, Radius, Covariance, Correlation coefficient	Channel, Socket	8

Features Space -

-- Stream aggregation:

1) Source IP - The source IP is used to track the host as a whole. It contains stats summarizing the recent traffic from this packet's host (IP).
2) Source MAC-IP - The source MAC-IP adds the capability to distinguish between traffic originating from different gateways and spoofed IP addresses. It contains stats summarizing the recent traffic from this packet's host (IP + MAC)
3) Channel - It contains stats summarizing the recent traffic going from this packet's host (IP) to the packet's destination host.
4) Socket - The sockets are determined by the source and destination TCP or UDP port numbers can identify each unique connection between a pair of devices. For example, all of the traffic sent from 192.168.1.12:1234 to 192.168.1.50:80 (traffic flowing from one socket to another). It contains stats summarizing the recent traffic going from this packet's host+port (IP) to the packet's destination host+port.

-- Time-frame (The decay factor Lambda used in the damped window):

1. How much recent history of the stream is capture in these statistics
2. L5, L3, L1, L0.1 and L0.01

-- The statistics extracted from the packet stream:

1. weight: The weight of the stream (can be viewed as the number of items observed in recent history)
2. mean: ...
3. std: ...
4. radius: The root squared sum of the two streams' variances
5. magnitude: The root squared sum of the two streams' means
6. cov: An approximated covariance between two streams

EXISTING ANALYSIS

Earlier related research that deals with botnets does not yet refer to the IoT, as the IoT concept has experienced tremendous growth only very recently. In 2011, research was done on detecting P2P (Peer-to-Peer) botnets during the pre-attack phase. In 2013 a group of researchers in India again explored P2P botnet detection with neural networks, the authors proposed and implemented a novel hybrid framework for detecting P2P botnets in live network traffic by integrating Neural Networks with Bayesian Regularization. Others tried different approaches, like not searching for particular patterns in overall traffic flow itself, but, for example, only in DNS queries.

IoT botnet research has been prominently on the scene more recently. In 2015, there was a proposal for a packet inspection algorithm that would have to be installed on the IoT device and inspect the network packet payload. Later works, however, recognized that this host-based approach would be challenging to implement in a natural environment due to the resource constraints and many differences between vendors. And so in 2018, a team from Ben-Gurion University at Negev described how botnet attack traffic can be detected based on network flow data using **autoencoders**, a particular type of neural network **[1]**; in addition to that, they also released a public dataset of Mirai and BASHLITE1 botnet traffic data, that will be used in this work.

A team at Aberdeen modeled IoT botnet attacks as a sequence using **recurrent neural networks** **[2]**. The developed Bidirectional Long Short Term Memory Recurrent Neural Network (**BLSTM-RNN)** detection model is compared to an **LSTM-RNN** for detecting *four attack vectors used* by the ***Mirai*** botnet and evaluated for accuracy and loss. The paper demonstrates that although the bidirectional approach adds overhead to each epoch and increases processing time, it proves to be a better progressive model over time. The positive results demonstrate the effectiveness of the novel application of deep learning for botnet detection in the IoT. By focusing detection at the packet level, and using text recognition on features normally discarded, they have demonstrated that the limitations of existing specification or flow based detection methods, can be overcomed with DL techniques.

A couple of studies have been conducted at TalTech: one showed how to detect anomalous traffic using a combination of feature selection techniques together with unsupervised machine learning methods like Local Outlier Factor, One-class SVM and Isolation Forest, while another focused on the exploration of **dimensionality reduction** and methods like Decision Tree and k-Nearest Neighbors classification **[3].** Application of the entire machine learning work-flow has allowed to dramatically reduce the dimensions of the feature-set without any considerable loss on detection accuracy. Their proposed model is based on one common classifier, instead of forming a classifier for each IoT devices, which makes it attractive from the deployment and online usage viewpoints in core networks. Finally, the results of decision tree classifier may be readily interpreted by the cyber security analysts and converted to the rules for signature-based intrusion detection systems which are used widely in many organisations.

Our approach to data cleaning/preparation:

1. We plan to analyze and compare the plots for malignant and benign instances for different context windows.
2. Given, our hypothesis is specific to a device, we plan to visualize the above mentioned plots for particular devices.

PROPOSED HYPOTHESES

Challenges faced while preparing the dataset for analysis:

1. **Dimensionality Reduction Issue:** The dataset consists of 115 features (23 features recorded over 5 sets of time context windows). It is difficult to plot & visualize patterns in this huge feature space. So, we need to perform dimensionality reduction.
2. **Sampling the dataset:** Given that the data is recorded over multiple combinations of devices, brands & attacks; it is a challenge to sample the data for testing our proposed hypothesis.

Some of hypothesis that we anticipate in the data are:

- **On the basis of Anomaly Detection ML/DL Models -**
    1. Models trained to detect botnet attacks on various brands of security cameras are equally effective in detecting anomalies in webcams.
    2. Models trained to detect botnet attacks on various brands of security cameras are not effective in detecting anomalies in thermostats and baby monitors.
- **On the basis of features of the dataset -**
    1. The packet flow and jitter time changes when devices are infected compared to benign traffic.
    2. Scan based mirai and bashlite attacks have similar flow of packets on various devices.
    3. UDP flooding based mirai and bashlite attacks have similar flow of packets on various devices.
    4. UDP flooding based mirai and bashlite attacks have similar packet jitter time on various devices.
    5. TCP flooding attacks have similar packet flow on various security camera brands.
    6. TCP flooding attacks have packet jitter on various security camera brands.
    7. The packet flow and packet jitter of various types of attacks varies with the type of device

Features or labels that we want to learn from the data using ML/DL models:

1. Perform **Dimensionality Reduction using AutoEncoder** based methods & check the performace of ML/DL classification models on the reduced dataset.
2. Perform cross device model evaluation.
   (Existing analysis does not includes any of the above proposed experiments.)

References:

[1] N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders | IEEE Journals & Magazine

[2] Botnet Detection in the Internet of Things using Deep Learning Approaches | IEEE Conference Publication

[3] Dimensionality Reduction for Machine Learning Based IoT Botnet Detection | IEEE Conference Publication

APPENDIX

1. **Distribution of botnet classes for a device**

Link : https://www.kaggle.com/code/mohamedahmedae/iot-simple-tensorflow-federated-learning

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
```

```python
benign_df = pd.read_csv('../input/nbaiot-dataset/5.benign.csv')

g_c_df = pd.read_csv('../input/nbaiot-dataset/5.gafgyt.combo.csv')
g_j_df = pd.read_csv('../input/nbaiot-dataset/5.gafgyt.junk.csv')
g_s_df = pd.read_csv('../input/nbaiot-dataset/5.gafgyt.scan.csv')
g_t_df = pd.read_csv('../input/nbaiot-dataset/5.gafgyt.tcp.csv')
g_u_df = pd.read_csv('../input/nbaiot-dataset/5.gafgyt.udp.csv')
m_a_df = pd.read_csv('../input/nbaiot-dataset/5.mirai.ack.csv')
m_sc_df = pd.read_csv('../input/nbaiot-dataset/5.mirai.scan.csv')
m_sy_df = pd.read_csv('../input/nbaiot-dataset/5.mirai.syn.csv')
m_u_df = pd.read_csv('../input/nbaiot-dataset/5.mirai.udp.csv')
m_u_p_df = pd.read_csv('../input/nbaiot-dataset/5.mirai.udpplain.csv')
```

```python
benign_df['type'] = 'benign'
m_u_df['type'] = 'mirai_udp'
g_c_df['type'] = 'gafgyt_combo'
g_j_df['type'] = 'gafgyt_junk'
g_s_df['type'] = 'gafgyt_scan'
g_t_df['type'] = 'gafgyt_tcp'
g_u_df['type'] = 'gafgyt_udp'
m_a_df['type'] = 'mirai_ack'
m_sc_df['type'] = 'mirai_scan'
m_sy_df['type'] = 'mirai_syn'
m_u_p_df['type'] = 'mirai_udpplain'
```

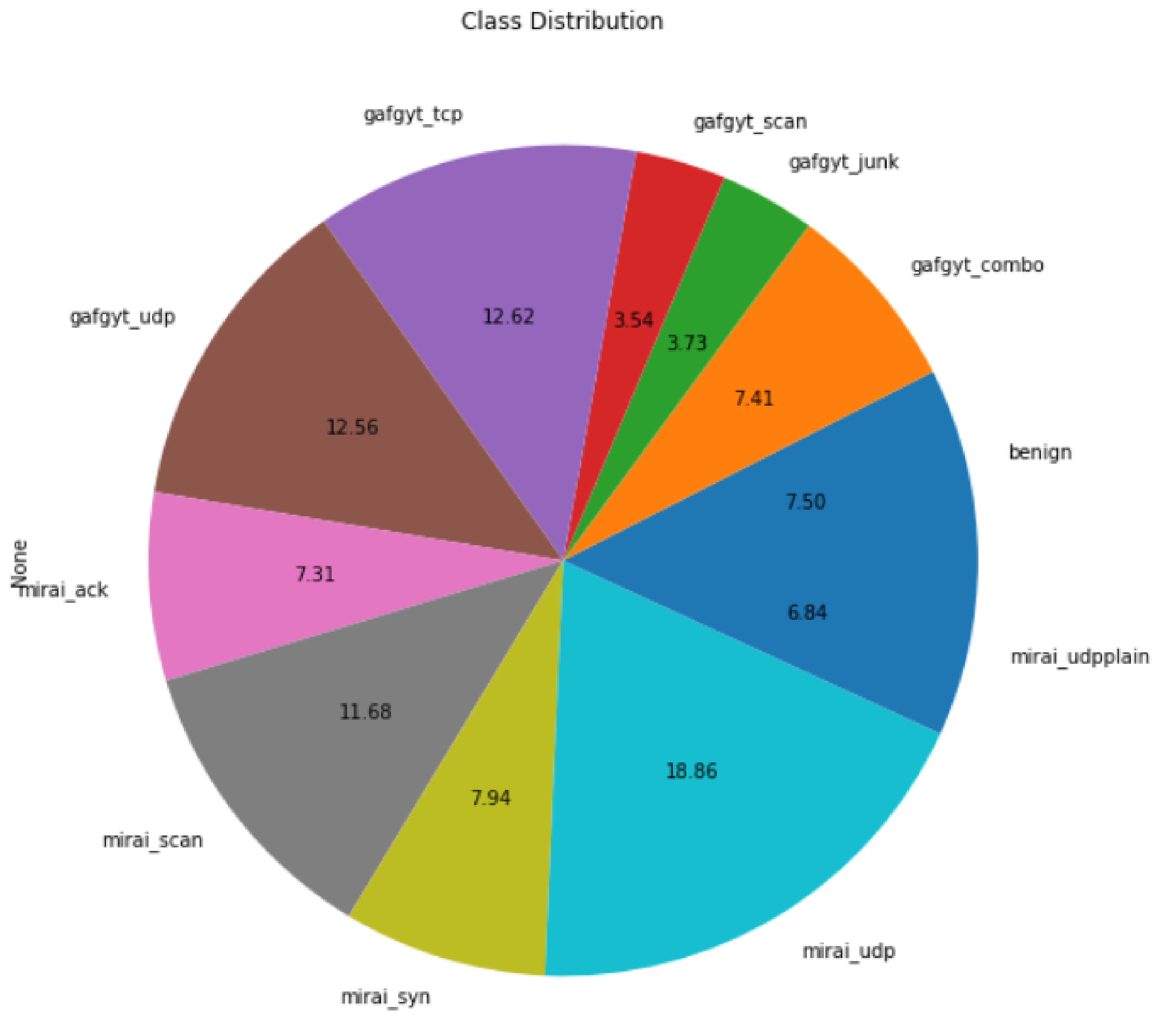
```python
df = pd.concat([benign_df, m_u_df, g_c_df,
                 g_j_df, g_s_df, g_t_df,
                 g_u_df, m_a_df, m_sc_df,
                 m_sy_df, m_u_p_df],
                 axis=0, sort=False, ignore_index=True)
```

```
Python
df["type"].value_counts()
```

```
mirai_udp         156248
gafgyt_tcp        104510
gafgyt_udp        104011
mirai_scan         96781
mirai_syn          65746
benign             62154
gafgyt_combo       61380
mirai_ack          60554
mirai_udpplain     56681
gafgyt_junk        30898
gafgyt_scan        29297
Name: type, dtype: int64
```

```
Python
from matplotlib import pyplot as plt

plt.title("Class Distribution")
df.groupby("type").size().plot(kind='pie', autopct='%.2f', figsize=(20,10))
```

```
Out[7]:
        <AxesSubplot:title={'center':'Class Distribution'}, ylabel='None'>
```


Class Distribution
gafgyt_tcp
12.62
gafgyt_scan
3.54
gafgyt_junk
3.73
gafgyt_combo
7.41
benign
7.50
mirai_udpplain
6.84
mirai_udp
18.86
mirai_syn
7.94
mirai_scan
11.68
mirai_ack
7.31
gafgyt_udp
12.56
None

2. “N-BaIoT - 2D and 3D Scatter Plots”

Link : https://www.kaggle.com/code/mkashifn/n-baiot-2d-and-3d-scatter-plots/notebook

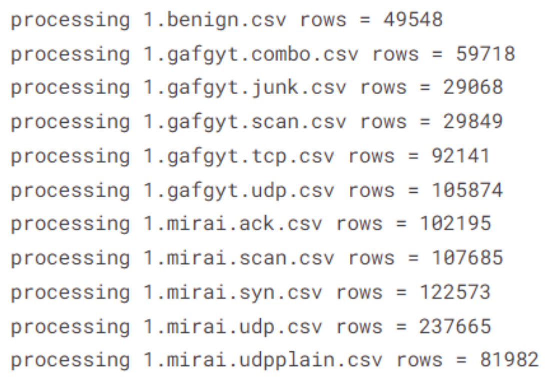
In this study, the author has generated 2-dimensional and 3-dimensional graphs following Principal Component Analysis (PCA) as a technique for dimensionality reduction. These graphical representations are specifically designed to accentuate the discrimination between benign and malware instances in the dataset.

Following are a few instances of these plots. We could not copy all of the data from the file here, so we will be attacking a link for the same.

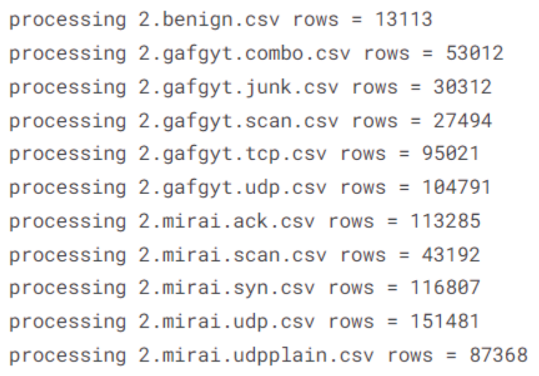
```python
# Utility functions

########################################################################
# Walk through input files
def print_input_files():
    # Input data files are available in the "../input/" directory.
    for dirname, _, filenames in os.walk('/kaggle/input'):
        for filename in filenames:
            print(os.path.join(dirname, filename))

########################################################################
# Dump text files
def dump_text_file(fname):
    with open(fname, 'r') as f:
        print(f.read())

########################################################################
# Dump CSV files
def dump_csv_file(fname, count=5):
    # count: 0 - column names only, -1 - all rows, default = 5 rows max
    df = pd.read_csv(fname)
    if count < 0:
        count = df.shape[0]
    return df.head(count)

########################################################################
# Dataset related functions
ds_nbaiot = '/kaggle/input/nbaiot-dataset'
dn_nbaiot = ['Danmini_Doorbell', 'Ecobee_Thermostat', 'Ennio_Doorbell', 'Philips_B120N10_Baby_Monitor',
'Provision_PT_737E_Security_Camera', 'Provision_PT_838_Security_Camera', 'Samsung_SNH_1011_N_Webcam',
'SimpleHome_XCS7_1002_WHT_Security_Camera', 'SimpleHome_XCS7_1003_WHT_Security_Camera']

def fname(ds, f):
    if '.csv' not in f:
        f = f'{f}.csv'
    return os.path.join(ds, f)

def fname_nbaiot(f):
    return fname(ds_nbaiot, f)
```

```python
def get_nbaiot_device_files():
    nbaiot_all_files = dump_csv_file(fname_nbaiot('data_summary'), -1)
    nbaiot_all_files = nbaiot_all_files.iloc[:,0:1].values
    device_id = 1
    indices = []
    for j in range(len(nbaiot_all_files)):
        if str(device_id) not in str(nbaiot_all_files[j]):
            indices.append(j)
            device_id += 1
    nbaiot_device_files = np.split(nbaiot_all_files, indices)
    return nbaiot_device_files

def get_nbaiot_device_data(device_id):
    if device_id < 1 or device_id > 9:
        assert False, "Please provide a valid device ID 1-9, both inclusive"
    device_index = device_id -1
    device_files = get_nbaiot_device_files()
    device_file = device_files[device_index]
    df = pd.DataFrame()
    y = []
    for i in range(len(device_file)):
        fname = str(device_file[i][0])
        df_c = pd.read_csv(fname_nbaiot(fname))
        rows = df_c.shape[0]
        print("processing", fname, "rows =", rows)
        y_np = np.ones(rows) if 'benign' in fname else np.zeros(rows)
        y.extend(y_np.tolist())
        df = pd.concat([df.iloc[:,:].reset_index(drop=True),
                    df_c.iloc[:,:].reset_index(drop=True)], axis=0)
    X = df.iloc[:,:].values
    y = np.array(y)
    return (X, y)

def get_nbaiot_devices_data():
    devices_data = []
    for i in range(9):
        device_id = i + 1
        (X, y) = get_nbaiot_device_data(device_id)
        devices_data.append((X, y))
    return devices_data
#print_input_files()

def plot_scatter_nbaiot_device(device_data, device_id):
    if device_id < 1 or device_id > 9:
        assert False, "Please provide a valid device ID 1-9, both inclusive"
    device_index = device_id-1
    print("scatter plot for", dn_nbaiot[device_index])
    (X, y) = device_data
    X_std = StandardScaler().fit_transform(X)
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X_std)
    data_X = X_pca[:,0]
    data_Y = X_pca[:,1]
    data_Z = y
    data = np.column_stack((data_X, data_Y, data_Z))
    plot_3d_scatter(data)

########################################################################
# Visualization related functions
def plot_3d_histogram(data):
    cols = data.shape[1]
    if cols < 2:
        assert False, 'The number of columns should be 2'
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    X = data[:,0]
    Y = data[:,1]
    bins = 10
    hist, xedges, yedges = np.histogram2d(X, Y, bins=bins, range=[[0, bins*0.6], [0, bins*0.6]])

    # Construct arrays for the anchor positions of the bars.
    xpos, ypos = np.meshgrid(xedges[:-1] + 0.25, yedges[:-1] + 0.25, indexing="ij")
    xpos = xpos.ravel()
    ypos = ypos.ravel()
    zpos = 0

    # Construct arrays with the dimensions for the 16 bars.
    dx = dy = 0.5 * np.ones_like(zpos)
    dz = hist.ravel()

    cmap = cm.get_cmap('cool')
    max_height = np.max(dz)
    min_height = np.min(dz)
    rgba = [cmap((k-min_height)/max_height) for k in dz]
    ax.bar3d(xpos, ypos, zpos, dx, dy, dz, zsort='average', color=rgba)

    plt.show()

def plot_3d_surface(data, func):
    cols = data.shape[1]
    if cols < 2:
        assert False, 'The number of columns should be 2'
    X = data[:,0]
    Y = data[:,1]
    X, Y = np.meshgrid(X, Y)
    Z = func(X, Y)
    #print(Z.shape)
    ax = plt.axes(projection='3d')
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='viridis', edgecolor='none')
    ax.set_title('surface');
```

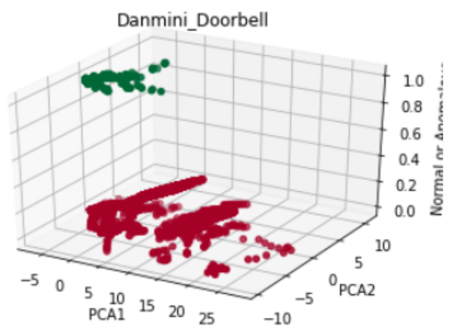
```python
def plot_3d_scatter(data, title=None, xlabel=None, ylabel=None, zlabel=None):
    cols = data.shape[1]
    if cols < 3:
        assert False, 'The number of columns should be 3'
    X = data[:,0]
    Y = data[:,1]
    Z = data[:,2]
    ax = plt.axes(projection='3d')
    ax.scatter(X, Y, Z, c = Z, cmap='RdYlGn')
    ax.set_title(title);
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_zlabel(zlabel)
    plt.show()

def plot_2d_scatter(data, title=None, xlabel=None, ylabel=None, handles=None):
    cols = data.shape[1]
    if cols < 3:
        assert False, 'The number of columns should be 3'
    X = data[:,0]
    Y = data[:,1]
    Z = data[:,2]
    ax = plt.axes()
    scatter = ax.scatter(X, Y, c = ['green' if z > 0.5 else 'red' for z in Z], cmap='RdYlGn')
    ax.set_title(title);
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    plt.legend(handles=handles)
    plt.show()
```

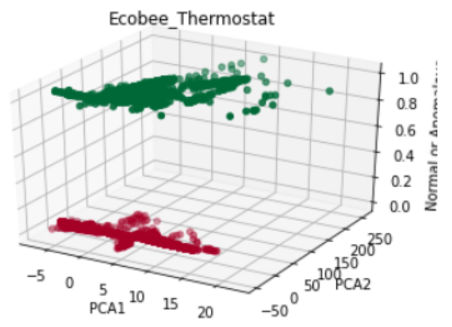


```
processing 1.benign.csv rows = 49548
processing 1.gafgyt.combo.csv rows = 59718
processing 1.gafgyt.junk.csv rows = 29068
processing 1.gafgyt.scan.csv rows = 29849
processing 1.gafgyt.tcp.csv rows = 92141
processing 1.gafgyt.udp.csv rows = 105874
processing 1.mirai.ack.csv rows = 102195
processing 1.mirai.scan.csv rows = 107685
processing 1.mirai.syn.csv rows = 122573
processing 1.mirai.udp.csv rows = 237665
processing 1.mirai.udpplain.csv rows = 81982
```



```
processing 2.benign.csv rows = 13113
processing 2.gafgyt.combo.csv rows = 53012
processing 2.gafgyt.junk.csv rows = 30312
processing 2.gafgyt.scan.csv rows = 27494
processing 2.gafgyt.tcp.csv rows = 95021
processing 2.gafgyt.udp.csv rows = 104791
processing 2.mirai.ack.csv rows = 113285
processing 2.mirai.scan.csv rows = 43192
processing 2.mirai.syn.csv rows = 116807
processing 2.mirai.udp.csv rows = 151481
processing 2.mirai.udpplain.csv rows = 87368
```





These are the plots for two different devices, namely, **Danmini Doorbell** and **Ecobee Thermostat.**