Configuration Manual

MSc Research Project
MSc Cybersecurity

Samuel Avwerosughene Arhore
Student ID: x21134502

School of Computing
National College of Ireland

Supervisor: Dr Vanessa Ayala-Rivera

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. Samuel Avwerosughene Arhore…………………………………………

Student ID: ………………x21134502………………………………………………………….……

Programme: ………MSc Cyber Security…………………………… **Year:** ……2022/2023…….

Module: …………MSc Research Project/Internship……………………………….………

Lecturer: ……………Dr Vanessa Ayala-Rivera…………………………………….………

Submission Due Date: ………………15th December 2022……………………………….………

Project Title: ……………Intrusion Detection in IoT using Machine Learning….……

Word Count: …………530………… **Page Count:** ………………20………….…….………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ………… Samuel Avwerosughene Arhore ………………………………

Date: ……………… 14 December 2022…………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Samuel Avwerosughene Arhore
x21134502

1 Introduction

A detailed description of the steps and processes involved in detecting intrusions in IoT systems is provided in this Configuration Manual. For the replication of the experimental setup, this document describes all the necessary software settings and tools.

2 System Specification

The configuration of the system used in this project is:

- Operating System: MacOS Ventura 13.0
- Processor: Apple M2
- Hard drive: 256GB
- RAM: 8GB

3 Software Tools

This project was implemented using the following software tools:

- Jupyter Notebook
- Python
- Anaconda Navigator

3.1 Software Installation

A detailed description of the steps taken in installing the tools is presented here.

- Python 3.11.1 can be downloaded and installed from https://www.python.org/downloads/macos/
- Visit Anaconda's website to download and install it

4. Implementation

In order to implement this project, the following Python libraries were used:

- Scikit-Learn
- Numpy

- Seaborn
- Pandas
- Matplotlib

Implementation on the Network Intrusion Detection dataset:

1. For all analyses, visualizations, and models in this paper, Python libraries were imported

```
In [1]: #importing required libraries

        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.metrics import accuracy_score
        from xgboost import XGBClassifier
        from sklearn.svm import SVC
        from sklearn.metrics import plot_confusion_matrix
        from sklearn import model_selection
```

Fig 1: Importing the libraries

2. The data was loaded on the notebook

```
In [2]: #importing the dataset to build model

        df_train = pd.read_csv("Train_data.csv")
```

Fig 2: Importing the dataset

```
In [4]: #checking the columns in the dataframe
        df_train.columns

Out[4]: Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
               'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
               'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',
               'su_attempted', 'num_root', 'num_file_creations', 'num_shells',
               'num_access_files', 'num_outbound_cmds', 'is_host_login',
               'is_guest_login', 'count', 'srv_count', 'serror_rate',
               'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
               'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
               'dst_host_srv_count', 'dst_host_same_srv_rate',
               'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
               'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
               'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
               'dst_host_srv_rerror_rate', 'class'],
              dtype='object')
```

Fig 3: Checking the Dataset columns

```
In [6]: #Checking for null values in the dataframe
        df_train.isnull().sum(axis = 0)

Out[6]: duration                       0
        protocol_type                  0
        service                        0
        flag                           0
        src_bytes                      0
        dst_bytes                      0
        land                           0
        wrong_fragment                 0
        urgent                         0
        hot                            0
        num_failed_logins              0
        logged_in                      0
        num_compromised                0
        root_shell                     0
        su_attempted                   0
        num_root                       0
        num_file_creations             0
        num_shells                     0
        num_access_files               0
        num_outbound_cmds              0
        is_host_login                  0
        is_guest_login                 0
        count                          0
        srv_count                      0
        serror_rate                    0
        srv_serror_rate                0
        rerror_rate                    0
        srv_rerror_rate                0
        same_srv_rate                  0
        diff_srv_rate                  0
        srv_diff_host_rate             0
        dst_host_count                 0
        dst_host_srv_count             0
        dst_host_same_srv_rate         0
        dst_host_diff_srv_rate         0
        dst_host_same_src_port_rate    0
        dst_host_srv_diff_host_rate    0
        dst_host_serror_rate           0
        dst_host_srv_serror_rate       0
        dst_host_rerror_rate           0
```

Fig 4: Checking for null values in the Dataset

3. Visualization of the data

```
In [14]: class_count.value_counts().plot.pie(autopct = '%2.f')

Out[14]: <AxesSubplot:ylabel='class'>
```

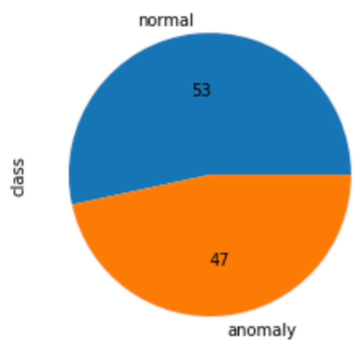

Fig 5: Data Visualization of the percentage of the anomalous packet and normal packet

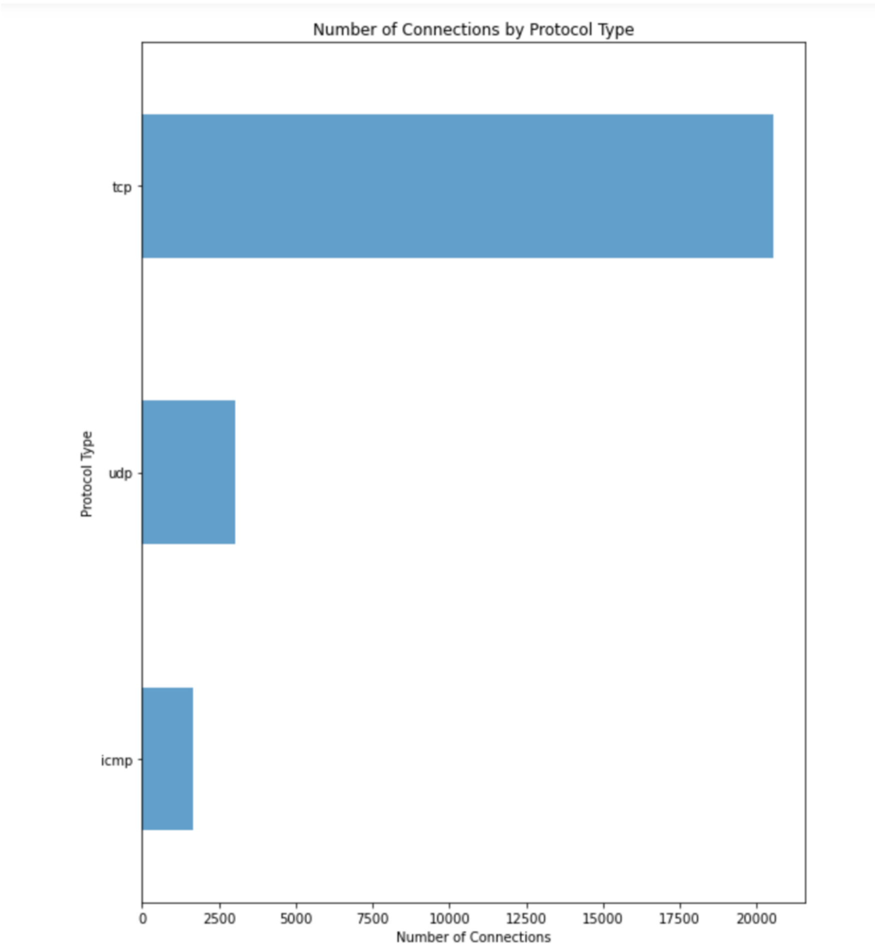

Fig 6: Data Visualization of the number of connection by the protocol type

4. Label Encoding the categorical features in the dataset to numerical values

In [17]:
```
#Label encoding the categorical features in the dataset

from sklearn.preprocessing import LabelEncoder

enc = LabelEncoder()
df_train['flag'] = enc.fit_transform(df_train['flag'])
df_train['service'] = enc.fit_transform(df_train['service'])
df_train['protocol_type'] = enc.fit_transform(df_train['protocol_type'])
#df_train['class'] = enc.fit_transform(df_train['class'])
```

Fig 7: Label Encoding

```
In [20]: #setting the flag for the target feature
df_train['class'].replace(['normal', 'anomaly'],
                          [0, 1], inplace=True)
```

Fig 8: Label Encoding

5. The following showed the use of correlation coefficient for feature Selection

```
In [22]: #Pearson correlation Index on initial dataframe
df_train.corr()['class']›‹

Out[22]: duration                       0.050901
protocol_type                 -0.283653
service                        0.270494
flag                          -0.651309
src_bytes                      0.005743
dst_bytes                     -0.010949
land                           0.000605
wrong_fragment                 0.097625
urgent                         0.006743
hot                           -0.012839
num_failed_logins              0.000028
logged_in                     -0.688084
num_compromised               -0.018620
root_shell                    -0.018579
su_attempted                  -0.025851
num_root                      -0.019753
num_file_creations            -0.018322
num_shells                    -0.013454
num_access_files              -0.036999
num_outbound_cmds                   NaN
is_host_login                       NaN
is_guest_login                -0.038662
count                          0.578790
srv_count                      0.002370
serror_rate                    0.649952
srv_serror_rate                0.647817
rerror_rate                    0.256858
srv_rerror_rate                0.256152
same_srv_rate                 -0.749237
diff_srv_rate                  0.193528
srv_diff_host_rate            -0.120649
dst_host_count                 0.368828
dst_host_srv_count            -0.719292
dst_host_same_srv_rate        -0.692212
dst_host_diff_srv_rate         0.238170
dst_host_same_src_port_rate    0.092974
dst_host_srv_diff_host_rate    0.062928
```

Fig 9: Feature Selection

```
In [23]: #dropping the negative features and features with weights over 0.5
df_train = df_train.drop(['num_compromised', 'root_shell','su_attempted','num_root', 'num_file_creations', 'num_shel
   'is_guest_login', 'srv_count', 'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'duration_minutes',
```

Fig 10: Removing the negative and heavy features

```
In [31]: #Setting the train test split to 70:30 percent of the dataframe
train_features, test_features, train_labels, test_labels = train_test_split(X,y, test_size=0.4, random_state=41)
```

Fig 11: Data is split into 70% training and 30% testing

6. The following show the different machine learning algorithms used in my model

```
##Random Forest Algorithm
```

```
In [32]: #Importing randomforest classifier and fitting it on the data

         from sklearn.ensemble import RandomForestClassifier
         clf = RandomForestClassifier()

         clf.fit(train_features, train_labels)

Out[32]: ▾ RandomForestClassifier
         RandomForestClassifier()
```

```
In [33]: %%time
         clf = RandomForestClassifier()
         clf.fit(train_features, train_labels)
         pred_clf= clf.predict(test_features)

         CPU times: user 380 ms, sys: 4.04 ms, total: 384 ms
         Wall time: 385 ms
```

```
In [34]: pred_train = clf.predict(train_features)
         pred_test = clf.predict(test_features)


         RFCaccuracy_train = (accuracy_score(pred_train,train_labels)) * 100
         RFCaccuracy_test = (accuracy_score(pred_test,test_labels)) * 100
```

```
In [35]: #Random forest model evaluation
         print('Training Accuracy is ' + str(round(RFCaccuracy_train, 4)) + '%')
         print('Test Accuracy is ' +  str(round(RFCaccuracy_test, 4)) + '%')

         Training Accuracy is 99.7155%
         Test Accuracy is 99.4046%
```

Fig 12: Random Forest Model

```
In [36]: #Classification Report for random forest
         print('Accuracy Score')
         print(accuracy_score(test_labels, pred_test),'\n')

         print('Precision Score')
         print(precision_score(test_labels, pred_test,average = None),'\n')

         print('Confusion Matrix')
         array = confusion_matrix(test_labels, pred_test)
         columns = ['Normal','Anomaly']
         print(pd.DataFrame(array,columns = columns, index = columns),'\n')

         print('Classification Report')
         print(classification_report(test_labels, pred_test),'\n')

         Accuracy Score
         0.9940458469782674

         Precision Score
         [0.99275362 0.99552716]

         Confusion Matrix
                  Normal  Anomaly
         Normal     5343       21
         Anomaly      39     4674

         Classification Report
                       precision    recall  f1-score   support

                    0       0.99      1.00      0.99      5364
                    1       1.00      0.99      0.99      4713

             accuracy                           0.99     10077
            macro avg       0.99      0.99      0.99     10077
         weighted avg       0.99      0.99      0.99     10077
```

Fig 13: Classification Report for Random Forest

```
In [38]: from IPython.display import Image
```

```
In [39]: %%time
         xgb = XGBClassifier(random_state=0)
         xgb.fit(train_features, train_labels)
         predict_xgb = xgb.predict(test_features)

         CPU times: user 1.85 s, sys: 268 ms, total: 2.11 s
         Wall time: 305 ms
```

```
In [40]: pred_train_xgb=xgb.predict(train_features)
         # predict test set
         pred_test_xgb=xgb.predict(test_features)
```

```
In [41]: XGBaccuracy_train = (accuracy_score(pred_train_xgb,train_labels)) * 100
         XGBaccuracy_test = (accuracy_score(pred_test_xgb,test_labels)) * 100
```

```
In [42]: #train_features_balanced, train_labels_balanced
         print('Training Accuracy is ' + str(round(XGBaccuracy_train, 4)) + '%')
         print('Test Accuracy is ' +  str(round(XGBaccuracy_test, 4)) + '%')

         Training Accuracy is 99.6957%
         Test Accuracy is 99.4443%
```

Fig 14: XGBoost Model

```
In [44]: print('Accuracy Score')
         print(accuracy_score(test_labels, pred_test_xgb),'\n')

         print('Precision Score')
         print(precision_score(test_labels, pred_test_xgb,average = None),'\n')

         print('Confusion Matrix')
         array = confusion_matrix(test_labels, pred_test_xgb)
         columns = ['Normal','Anomaly']
         print(pd.DataFrame(array,columns = columns, index = columns),'\n')

         print(confusion_matrix(test_labels, pred_test_xgb))
         print(classification_report(test_labels, pred_test_xgb))

         Accuracy Score
         0.9944427905130495

         Precision Score
         [0.99330855 0.99574196]

         Confusion Matrix
                  Normal  Anomaly
         Normal     5344       20
         Anomaly      36     4677

         [[5344   20]
          [  36 4677]]
                       precision    recall  f1-score   support

                    0       0.99      1.00      0.99      5364
                    1       1.00      0.99      0.99      4713

             accuracy                           0.99     10077
            macro avg       0.99      0.99      0.99     10077
         weighted avg       0.99      0.99      0.99     10077
```

Fig 15: Classification Report for XGBoost

```
In [45]: ##SVM
```

```
In [46]: clfsvm = SVC(kernel = 'rbf', random_state = 0)

         clfsvm.fit(train_features, train_labels)

         pred_trainsvm = clfsvm.predict(train_features)
         pred_testsvm = clfsvm.predict(test_features)
```

```
In [47]: svmaccuracy_train = (accuracy_score(pred_trainsvm,train_labels)) * 100
         svmaccuracy_test = (accuracy_score(pred_testsvm,test_labels)) * 100
```

```
In [48]: #SVM model evaluation
         print('Training Accuracy is ' + str(round(svmaccuracy_train, 4)) + '%')
         print('Test Accuracy is ' +  str(round(svmaccuracy_test, 4)) + '%')

         Training Accuracy is 53.953%
         Test Accuracy is 53.5973%
```

Fig 16: SVM Model

```
In [49]: #Classification Report for SVM
         print('Accuracy Score')
         print(accuracy_score(test_labels, pred_testsvm),'\n')

         print('Precision Score')
         print(precision_score(test_labels, pred_testsvm,average = None),'\n')

         print('Confusion Matrix')
         array = confusion_matrix(test_labels, pred_testsvm)
         columns = ['Normal','Anomaly']
         print(pd.DataFrame(array,columns = columns, index = columns),'\n')

         print('Classification Report')
         print(classification_report(test_labels, pred_testsvm),'\n')

         Accuracy Score
         0.5359730078396349

         Precision Score
         [0.53437937 0.76056338]

         Confusion Matrix
                  Normal  Anomaly
         Normal     5347       17
         Anomaly    4659       54

         Classification Report
                       precision    recall  f1-score   support

                    0       0.53      1.00      0.70      5364
                    1       0.76      0.01      0.02      4713

             accuracy                           0.54     10077
            macro avg       0.65      0.50      0.36     10077
         weighted avg       0.64      0.54      0.38     10077
```

Fig 17: Classification report for SVM

Implementation on the IoT Device Network Logs dataset:

1. Import the libraries for all analyses, visualizations, and models in this paper, Python libraries were imported

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.metrics import accuracy_score
        from xgboost import XGBClassifier
        from sklearn.svm import SVC
        from sklearn.metrics import plot_confusion_matrix
        from sklearn import model_selection
```

Fig 18: Importing libraries to be used

2. The data was loaded on the notebook

```
In [2]: df =  pd.read_csv('Preprocessed_data.csv')
```

Fig 19: Importing the dataset to the notebook

3. Data analysis and visualization

```
In [3]: df.head()
Out[3]:
```

	frame.number	frame.time	frame.len	eth.src	eth.dst	ip.src	ip.dst	ip.proto	ip.len	tcp.len	tcp.srcport	tcp.dstport
0	1	123722736684743	54	87971959760497	167275820076079	192168035	1921680121	6.0	40.0	0.0	49279.0	80.0
1	2	123722736773147	62	87971959760497	167275820076079	192168035	1921680121	6.0	48.0	0.0	56521.0	80.0
2	3	123722736824792	62	167275820076079	87971959760497	1921680121	192168035	6.0	48.0	0.0	80.0	56521.0
3	4	123722736836228	54	167275820076079	87971959760497	1921680121	192168035	6.0	40.0	0.0	80.0	49279.0
4	5	123722749684991	54	87971959760497	167275820076079	192168035	1921680121	6.0	40.0	0.0	56521.0	80.0

Fig 20: Top 5 records of the dataset

```
In [4]: df['normality'].value_counts()

Out[4]: 1    82285
        4    79052
        0    79035
        5    79032
        2    79020
        3    79002
        Name: normality, dtype: int64
```

Fig 21: Checking the evenness of the target variable

```
In [32]: df['normality'].value_counts().plot.pie(autopct = '%1.2f%%')
Out[32]: <AxesSubplot:ylabel='normality'>
```

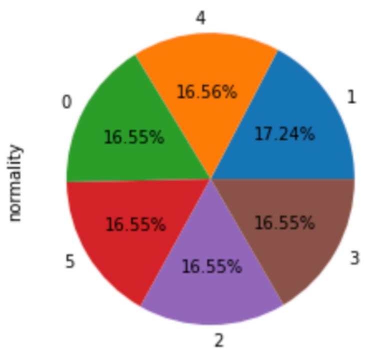

Fig 21: Visualization of the target variable in a pie chart

4. Label Encoding the different attacks into numerical figures

```
In [20]: def normality_label(df):
             if df['normality'] == 0:
                 return 'normal'
             elif df['normality'] == 1:
                 return 'wrong setup'
             elif df['normality'] == 2:
                 return 'ddos'
             elif df['normality'] == 3:
                 return 'data type probing'
             elif df['normality'] == 4:
                 return 'scan attack'
             else:
                 return 'man in the middle'
```

Fig 22: Label Encoding

```
In [15]: df['normality_label'].value_counts()

Out[15]: wrong setup          82285
         scan attack          79052
         normal               79035
         man in the middle    79032
         ddos                 79020
         data type probing    79002
         Name: normality_label, dtype: int64
```

Fig 23: Visual representation of the Labels Encoded

5. Using the correlation coefficient for feature Selection

```
In [14]: df.corr()['normality']

Out[14]: frame.number   -0.021738
         frame.time     -0.036645
         frame.len      -0.473391
         eth.src         0.141900
         eth.dst         0.187007
         ip.src         -0.154404
         ip.dst         -0.172763
         ip.proto       -0.760546
         ip.len         -0.540205
         tcp.len        -0.397025
         tcp.srcport    -0.459901
         tcp.dstport    -0.426248
         Value           0.000068
         normality       1.000000
         Name: normality, dtype: float64
```

Fig 24: Feature Selection

```
In [15]: #dropping some of the high negative features
         df_filtered = df.drop(['frame.number', 'frame.time', 'frame.len', 'ip.proto', 'ip.len', 'tcp.len', 'tcp.srcport', 't
```

Fig 25: Dropping some high negative features

```
In [16]: df_filtered.corr()['normality']

Out[16]: eth.src       0.141900
         eth.dst       0.187007
         ip.src       -0.154404
         ip.dst       -0.172763
         normality     1.000000
         Name: normality, dtype: float64
```

Fig 26: The used features in the model

```
In [22]: #Setting the train test split to 70:30 percent of the dataframe
         train_features, test_features, train_labels, test_labels = train_test_split(X,y, test_size=0.4, random_state=41)
```

Fig 27: Splitting the data into 70% test and 30% train

6. The following show the different machine learning algorithms used in my model

```
In [23]: #Importing randomforest classifier and fitting it on the data

         from sklearn.ensemble import RandomForestClassifier
         clf = RandomForestClassifier()

         clf.fit(train_features, train_labels)

         pred_train = clf.predict(train_features)
         pred_test = clf.predict(test_features)


         RFCaccuracy_train = (accuracy_score(pred_train,train_labels)) * 100
         RFCaccuracy_test = (accuracy_score(pred_test,test_labels)) * 100
```

```
In [45]: %%time
         clf = RandomForestClassifier()
         clf.fit(train_features, train_labels)
         pred_clf= clf.predict(test_features)

         CPU times: user 5.38 s, sys: 121 ms, total: 5.5 s
         Wall time: 5.74 s
```

```
In [24]: #Random forest model evaluation
         print('Training Accuracy is ' + str(round(RFCaccuracy_train, 4)) + '%')
         print('Test Accuracy is ' +  str(round(RFCaccuracy_test, 4)) + '%')

         Training Accuracy is 83.1094%
         Test Accuracy is 82.999%
```

Fig 28: Random forest model

```
Accuracy Score
0.8299898937535019

Precision Score
[0.99540542 0.67978856 0.9765913  0.65592551 0.9937931  0.99996828]

Confusion Matrix
                   Normal  Wrong Setup   DDOS  Data type Probing  Scan Attack  \
Normal              15382        15279    381                447          198
Wrong Setup             0        32793      0                  0            0
DDOS                    0            0  15895              15922            0
Data type Probing       0          168      0              31205            0
Scan Attack            71            0      0                  0        31702
Man in the Middle       0            0      0                  0            0

                   Man in the Middle
Normal                             0
Wrong Setup                        0
DDOS                               0
Data type Probing                  0
Scan Attack                        1
Man in the Middle              31527

Classification Report
              precision    recall  f1-score   support

           0       1.00      0.49      0.65     31687
           1       0.68      1.00      0.81     32793
           2       0.98      0.50      0.66     31817
           3       0.66      0.99      0.79     31373
           4       0.99      1.00      1.00     31774
           5       1.00      1.00      1.00     31527

    accuracy                           0.83    190971
   macro avg       0.88      0.83      0.82    190971
weighted avg       0.88      0.83      0.82    190971
```

Fig 29: Classification Report of the Random Forest algorithm

```
In [28]: from IPython.display import Image
```

```
In [44]: %%time
xgb = XGBClassifier(random_state=0)
xgb.fit(train_features, train_labels)
predict_xgb = xgb.predict(test_features)
```

```
CPU times: user 1min 33s, sys: 9.88 s, total: 1min 42s
Wall time: 13.7 s
```

```
In [30]: pred_train_xgb=xgb.predict(train_features)
# predict test set
pred_test_xgb=xgb.predict(test_features)
```

```
In [31]: XGBaccuracy_train = (accuracy_score(pred_train_xgb,train_labels)) * 100
XGBaccuracy_test = (accuracy_score(pred_test_xgb,test_labels)) * 100
```

```
In [32]: #train_features_balanced, train_labels_balanced
print('Training Accuracy is ' + str(round(XGBaccuracy_train, 4)) + '%')
print('Test Accuracy is ' +  str(round(XGBaccuracy_test, 4)) + '%')
```

```
Training Accuracy is 83.1094%
Test Accuracy is 82.999%
```

Fig 30: XGBoost Model

```
Accuracy Score
0.8299898937535019

Precision Score
[0.99540542 0.67978856 0.9765913  0.65592551 0.9937931  0.99996828]

Confusion Matrix
                   Normal  Wrong Setup   DDOS  Data type Probing  Scan Attack  \
Normal              15382        15279    381                447          198
Wrong Setup             0        32793      0                  0            0
DDOS                    0            0  15895              15922            0
Data type Probing       0          168      0              31205            0
Scan Attack            71            0      0                  0        31702
Man in the Middle       0            0      0                  0            0

                   Man in the Middle
Normal                             0
Wrong Setup                        0
DDOS                               0
Data type Probing                  0
Scan Attack                        1
Man in the Middle              31527

[[15382 15279   381   447   198     0]
 [    0 32793     0     0     0     0]
 [    0     0 15895 15922     0     0]
 [    0   168     0 31205     0     0]
 [   71     0     0     0 31702     1]
 [    0     0     0     0     0 31527]]
              precision    recall  f1-score   support

           0       1.00      0.49      0.65     31687
           1       0.68      1.00      0.81     32793
           2       0.98      0.50      0.66     31817
           3       0.66      0.99      0.79     31373
           4       0.99      1.00      1.00     31774
           5       1.00      1.00      1.00     31527

    accuracy                           0.83    190971
   macro avg       0.88      0.83      0.82    190971
weighted avg       0.88      0.83      0.82    190971
```

Fig 31: Classification Report of the XGBoost algorithm

```
In [36]: df_filtered_svm = df_filtered.sample(frac = .1)
```

Fig 32: Setting only 10% of the dataset to be used

```
In [39]: clfsvm = SVC(kernel = 'rbf', random_state = 2)

         clfsvm.fit(train_features_svm, train_labels_svm)

Out[39]: SVC
         SVC(random_state=2)

In [40]: pred_trainsvm = clfsvm.predict(train_features_svm)
         pred_testsvm = clfsvm.predict(test_features_svm)

In [41]: svmaccuracy_train = (accuracy_score(pred_trainsvm,train_labels_svm)) * 100
         svmaccuracy_test = (accuracy_score(pred_testsvm,test_labels_svm)) * 100

In [42]: #SVM model evaluation
         print('Training Accuracy is ' + str(round(svmaccuracy_train, 4)) + '%')
         print('Test Accuracy is ' +  str(round(svmaccuracy_test, 4)) + '%')

         Training Accuracy is 67.4568%
         Test Accuracy is 67.7505%
```

Fig 33: SVM Model

```
Accuracy Score
0.6775054979579014

Precision Score
[0.75       0.6440404  0.65913758 0.60351148 0.98490186 0.67234806]

Confusion Matrix
                  Normal  Wrong Setup  DDOS  Data type Probing  Scan Attack  \
Normal                63         1544    37                 55           30
Wrong Setup            0         3188     0                  0            0
DDOS                   0            0  1605               1622            0
Data type Probing      0           35     0               3128            0
Scan Attack           19            0   793                378         1957
Man in the Middle      2          183     0                  0            0

                  Man in the Middle
Normal                         1437
Wrong Setup                       0
DDOS                              0
Data type Probing                 0
Scan Attack                      24
Man in the Middle              2998

Classification Report
              precision    recall  f1-score   support

           0       0.75      0.02      0.04      3166
           1       0.64      1.00      0.78      3188
           2       0.66      0.50      0.57      3227
           3       0.60      0.99      0.75      3163
           4       0.98      0.62      0.76      3171
           5       0.67      0.94      0.78      3183

    accuracy                           0.68     19098
   macro avg       0.72      0.68      0.61     19098
weighted avg       0.72      0.68      0.61     19098
```

Fig 34: Classification Report for the SVM algorithm