National College of Ireland

Configuration Manual

MSc Research Project
MSc Cybersecurity

Pradeep Prakash
Student ID: X21215413

School of Computing
National College of Ireland

Supervisor: Evgeniia Jayasekera

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Pradeep Prakash
Student ID:	21215413
Programme:	MSc in Cybersecurity **Year:** 2022-2023
Module:	MSc Research Project
Lecturer:	Evgeniia Jayasekera
Submission Due Date:	14-Aug-2023
Project Title:	Enhanced Security for Insecure Systems within Zero trust Architecture

Word Count:1625 Page Count: 15

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: Pradeep Prakash

Date: 14-08-2023

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

Pradeep Prakash
Student ID: 21215413

1 Introduction

This Configuration Manual consists of the fundamental environment setup requirements, and which also consists of python modules that are required to carryout this project. The main agenda is to create a kind of Zero trust security framework with the help of docker and intend to introduce the trained models into this simulated environment and observe their efficiency in detecting anomalies in real-time. This would also allow to test the robustness of these models in a controlled but realistic network environment.

2 Hardware Requirements

Operating System: Windows 11, Kali Linux OS
RAM: 16GB
Processor: Intel Core i5
Storage: 512GB SSD
System Type: 64-bit operating system

3 Software Requirements

Anaconda Navigator
Jupyter Notebook or Google Colab
Python 3.6.3 version
GNS3
Python libraries like: keras, scikit learn and tensorflow

Python programming was used in this research for implementation of the ML model and also Jupyter notebook was used for this research. Many python libraries were utilized in order to accomplish this research work and its analysis will be explained in the subsequent section of this report.

Anaconda navigator is also used which has the Jupyter package within it. As it is useful for executing and debugging the python code. I have used Anaconda Navigator 64-bit version on my windows 11 system. Anaconda navigator has been downloaded from the link mentioned below. (*Anaconda Navigator* , no date)
https://docs.anaconda.com/free/navigator/install/

4 List of Python Libraries Installed

The following python libraries were used and installed in the research implementation environment with the help of python standard command called pip.

Keras: It is a python-based deep learning API that runs on top of the machine learning platform called TensorFlow. (*Keras Library*, no date)

TensorFlow: An open-source software library for efficient numerical computation is called TensorFlow. Because of its flexible design, computation may be quickly deployed among a range of platforms (CPUs, GPUs, and TPUs), from desktop machines to server clusters to mobile and peripheral devices. (*Tensorflow Library*, no date)

Scikit-learn: Based on SciPy, the Python machine learning package scikit-learn has been made available under the 3-Clause BSD license. (*Scikit Library*, no date)

5 Dataset Description

In this research project CICIDS2017 Dataset has been used to train the machine learning model and this dataset consists of large network flows that was captured for span of 10days.The selected dataset has variety of traffic which includes benign traffic also it has wide range of different attacks, and those attacks can be classified into the following categories:

1. Denial of Service
2. Intrusions attacks
3. Malware
4. Botnets
5. Web Attacks
6. Zero-day attacks

Selected dataset is huge and widespread which makes the representation of real-world traffic in a superior manner. The dataset which has been selected for carrying out the implementation of research work has helped to test the proposed ML model on a wide range of scenarios.
Link for the dataset is as follows:
https://www.kaggle.com/datasets/devendra416/ddos-datasets

Snapshot of different attack scenarios that was performed in this research implementation is as follows:

1. Denial of Service:

As per the need of the project, this is mentionable that the hping3 has been utilised in order to perform the tasks associated with the DDOS attack. In that aspect, the hping3 is very much useful in terms of the fact that this can perform the flooding attack and can transmit an unusual amount of TCP packets at a time.

```
┌──(kali㉿kali)-[~]
└─$ sudo su
[sudo] password for kali:
┌──(root㉿kali)-[/home/kali]
└─# hping3 -c 15000 -d 120 -S -w 64 -p 80 --flood --rand-source 192.168.6.133
HPING 192.168.6.133 (eth0 192.168.6.133): S set, 40 headers + 120 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.6.133 hping statistic ---
1759283 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Figure 1: DOS attack

The TCP syn flooding that has been done in the previous step. As per the figure, this can be seen that the red colour is responsible for showcasing the result related to the attack and unusual TCP attacks.

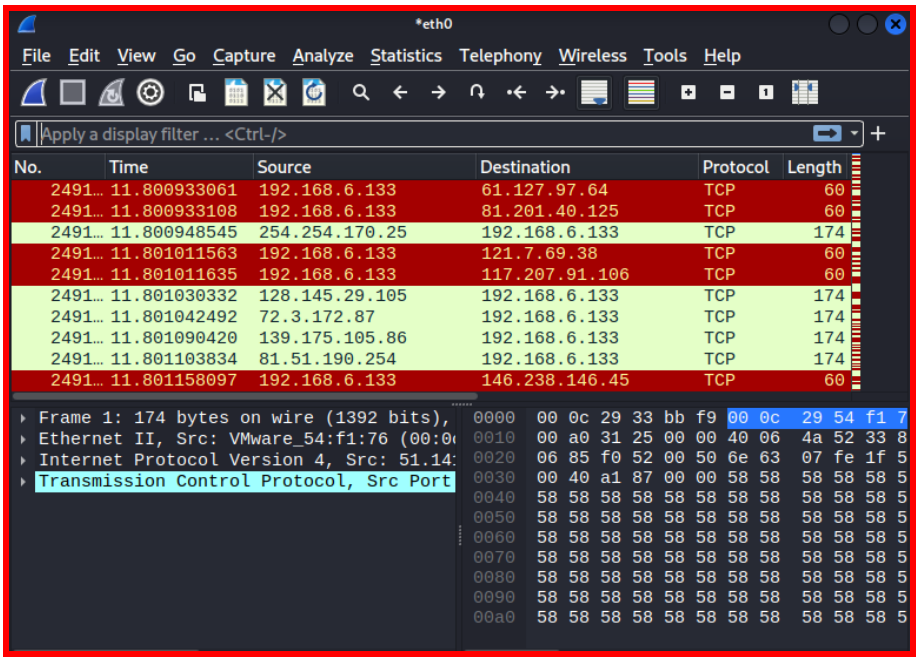

Figure 2: Wireshark capturing TCP unusual data packets.

2. Intrusion Attacks:

Nmap has been utilised with the need to check the open ports. Before starting the attack, this is important to check the open ports. Hydra tool has been utilised with the goal of performing a Brute force attack. This is an attack that is capable to break the password of the target machine.

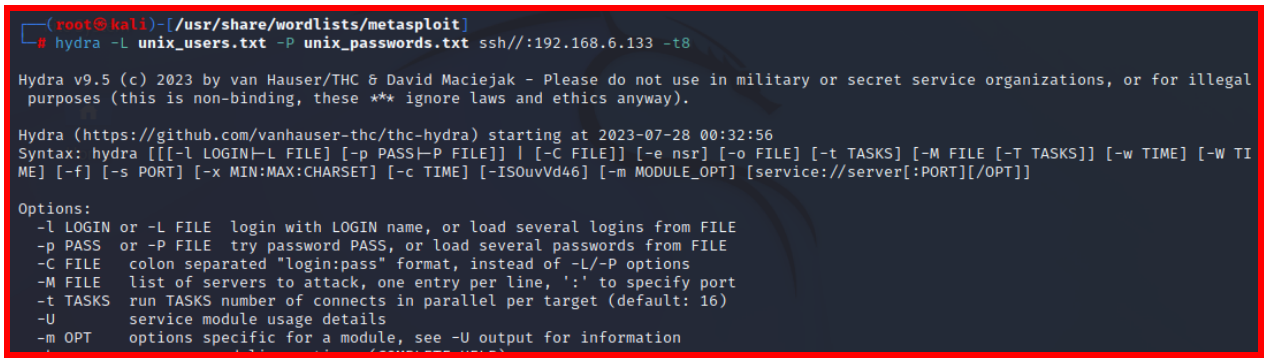

Figure 3: Brute Force Attack.

Brute force has been accomplished and it has made 15 login tries and finally has cracked the password and the user id of the targeted machine.

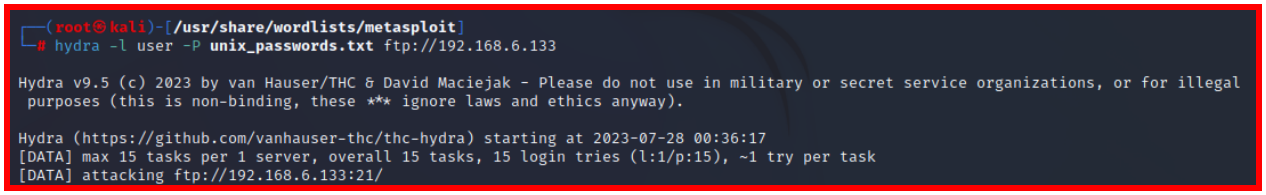

Figure 4: Accomplishment of Brute force attack.

3. Malware

The Lynis tool has been utilised to perform malware checking. In that aspect, the above command has been written in the kali Linux platform.

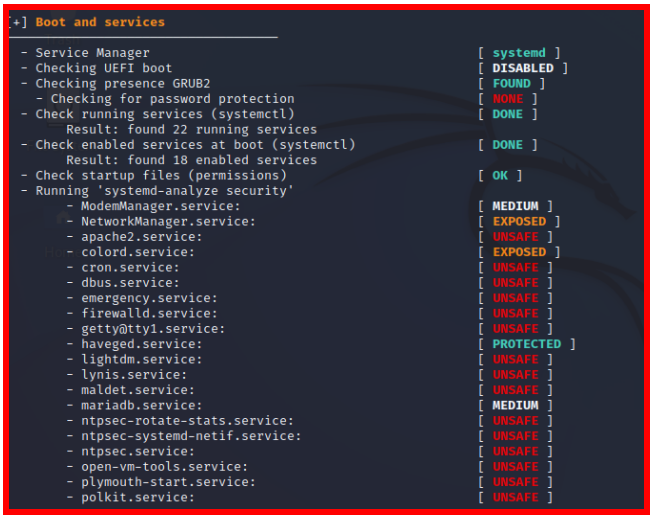

Figure 5: Identification of Unsafe Programs.

Figure 6: Final Report.

4. Botnet

Rkhunter has been utilised to perform botnet checking. Once the anomalies get detected it provides the warning.

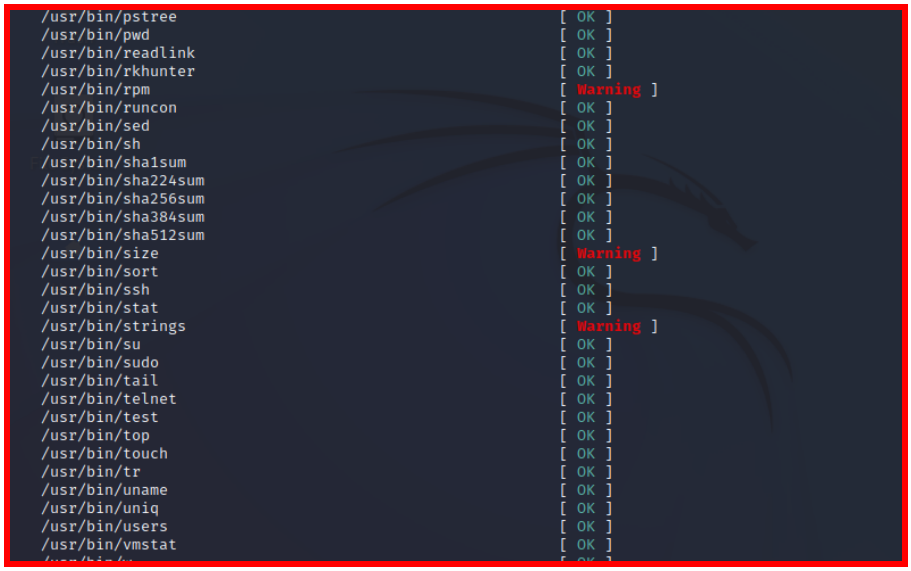

Figure 7: Warnings Provided by RKhunter.

5. Web Based Attacks:

1) Backdoors
2) DOS
3) Worms
4) Exploits
5) Fuzzers

Shell SQL Injection was performed on the application running in docker. SQLmap has been utilised in order to perform the vulnerability checking depending on the need of the project.

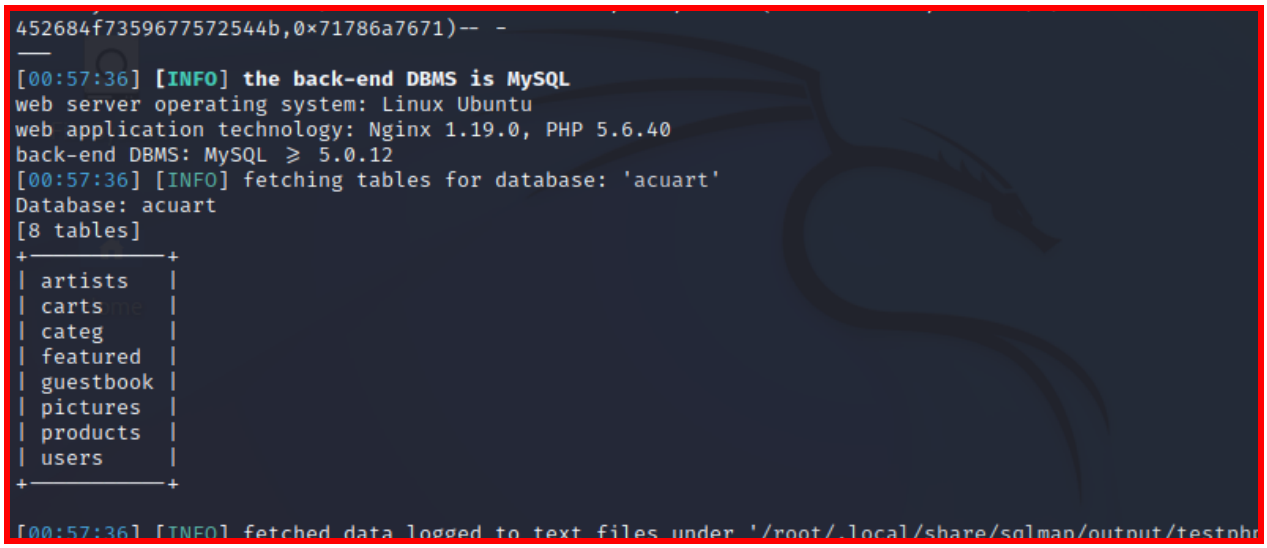

Figure 8: Available Tables.

Figure 9: Available Columns.

Depending on the need, the dumping has been done where the dumping has been done inside a CSV file.

Figure 10: Output of dumping data.

Creation of Deep Learning Layer:

The whole deep learning layer is built using tensorflow and Keras library which are available in python language. Below is snap of the code which is used in google colab to perform the implementation of the ML model. And remaining model which we are going to use in this implementation will be imported using SKlearn.

```
[ ]  !pip install tensorflow==2.10.0
     !pip install keras==2.10.0
     !pip install h5py==3.7.0
```

Figure 11: Deep Learning dependencies

Data importing, Cleaning and Preprocessing:

In this step around one lakh data is being imported from the dataset which are belonging to only class DOS and benign and at the same time we are skipping the unwanted rows from the dataset and a final dataset is created with the selected target variables. Below snapshot shows the final dataset along with some attributes like Src IP, Dst IP and many more.

	Unnamed: 0	Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean	Fwd Pkt Len Std	Bwd Pkt Len Max	Bwd Pkt Len Min	Bwd Pkt Len Mean	Bwd Pkt Len Std	Flow Byts/s	Flow Pkts/s	Flow IAT Mean	Fl
0	624	192.168.4.118-203.73.24.75-4504-80-6	192.168.4.118	4504	203.73.24.75	80	6	12/06/2010 08:34:32 AM	3974862	29	44	86.0	59811.0	86.0	0.0	2.965517	15.969799	1460.0	0.0	1359.340909	372.027190	15068.950821	18.365417	55206.416667	195
1	625	192.168.4.118-203.73.24.75-4504-80-6	192.168.4.118	4504	203.73.24.75	80	6	12/06/2010 08:34:36 AM	63	1	1	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.000000	31746.031746	63.000000	
2	626	192.168.4.118-203.73.24.75-4505-80-6	192.168.4.118	4505	203.73.24.75	80	6	12/06/2010 08:34:36 AM	476078	2	6	86.0	3037.0	86.0	0.0	43.000000	60.811183	1460.0	0.0	506.166667	740.224403	6559.849436	16.803969	68011.142857	110
3	627	192.168.4.118-203.73.24.75-4505-80-6	192.168.4.118	4505	203.73.24.75	80	6	12/06/2010 08:34:37 AM	151	2	1	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.000000	19867.549669	75.500000	
4	628	192.168.4.118-203.73.24.75-4506-80-6	192.168.4.118	4506	203.73.24.75	80	6	12/06/2010 08:34:37 AM	472507	2	5	73.0	1050.0	73.0	0.0	36.500000	51.618795	1050.0	0.0	210.000000	469.574275	2376.684367	14.814595	78751.166667	118

Figure 12: Final combined selected dataset.

Count Plot:

In this step we are checking whether the final dataset is balanced with same number of selected target variable entries. The output of this step is shown below:

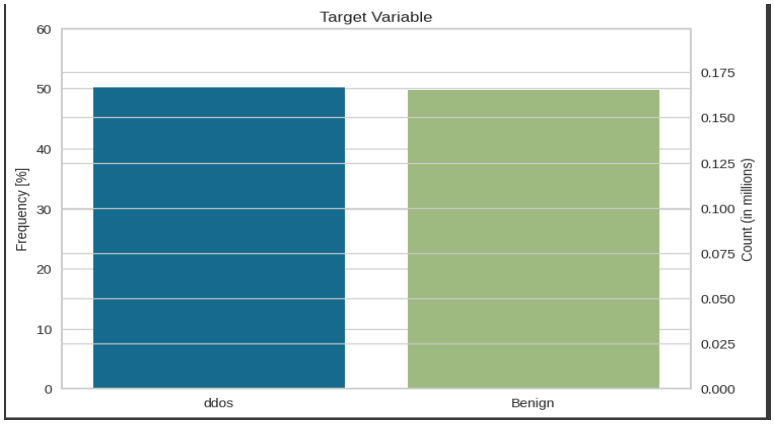

Figure 13: Count Plot.

Feature Importance:
The feature Importance represents which features have more impact on the selected target variables. Below graph shows the top 20 features and its impact level.

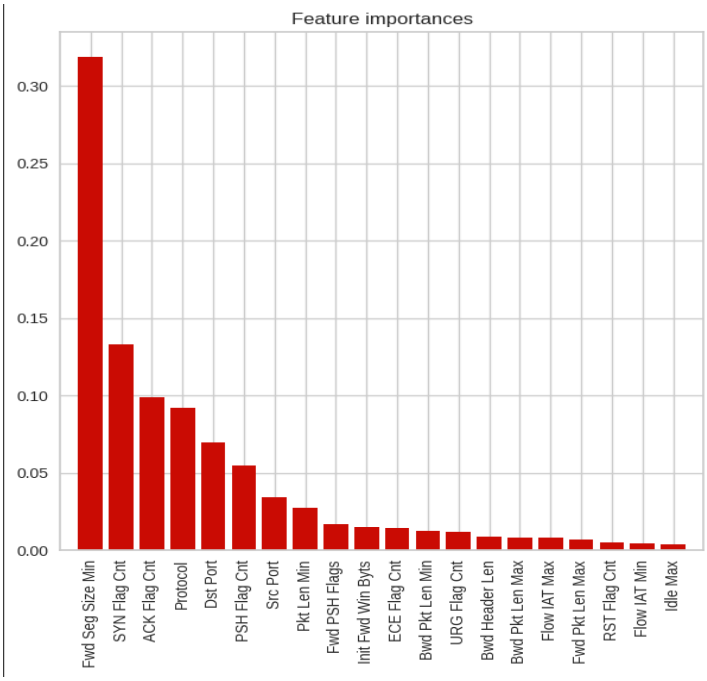

Figure 14: Feature Importance Graph.

6 Training and testing Summary of ML models

- **Training and testing:**

In this step the dataset is split into training and testing categories in which 60% of data is being used for training and remaining 40% is being used for testing purpose.

```
[ ]  X = df_final.drop(['Label','Unnamed: 0','Flow ID','Timestamp','Src IP','Dst IP'],1)
     y=df_final['Label']

     #Convert Str to int in Labes
     y.replace({"ddos": 1, "Benign": 0}, inplace=True)

     X_train, X_test, y_train, y_test =train_test_split(X, y ,test_size=0.4)
```

Figure 15: Training and testing.

- **Random forest Classifier:**

Accuracy of this model and classification report is as follows:

```
[ ]  #Random Forest
     random_forest_model = RandomForestClassifier(n_estimators=2, criterion='gini', max_depth=1)
     rf = random_forest_model
     rf.fit(X_train,y_train)
     y_pred = rf.predict(X_test)

[ ]  #accuracy score
     print("Accuracy Score: ",accuracy_score(y_test,y_pred))
     #confusion Matrix
     matrix =confusion_matrix(y_test, y_pred)
     class_names=[0,1]
     fig, ax = plt.subplots()
     tick_marks = np.arange(len(class_names))
     plt.xticks(tick_marks, class_names)
     plt.yticks(tick_marks, class_names)
     sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
     ax.xaxis.set_label_position("top")
     plt.tight_layout()
     plt.title('Confusion matrix', y=1.1)
     plt.ylabel('Actual label')
     plt.xlabel('Predicted label')
     plt.show()
     #Classification Report
     print(classification_report(y_test, y_pred))

     Accuracy Score:  0.9421901474741136
```

Figure 16: Accuracy of the model.

	precision	recall	f1-score	support
0	0.91	0.97	0.94	39632
1	0.97	0.91	0.94	40043
accuracy			0.94	79675
macro avg	0.94	0.94	0.94	79675
weighted avg	0.94	0.94	0.94	79675

Figure 17: Classification report.

- **XGBOOST Classifier:**

Accuracy of this model and classification report is as follows:

```
[ ]  #Xgboost Classifier
     xgboost_model = XGBClassifier(max_depth=1, n_estimators=2)
     xg = xgboost_model
     xg.fit(X_train,y_train)
     y_pred = xg.predict(X_test)

[ ]  #accuracy score
     print("Accuracy Score: ",accuracy_score(y_test,y_pred))
     #confusion Matrix
     matrix =confusion_matrix(y_test, y_pred)
     class_names=[0,1]
     fig, ax = plt.subplots()
     tick_marks = np.arange(len(class_names))
     plt.xticks(tick_marks, class_names)
     plt.yticks(tick_marks, class_names)
     sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
     ax.xaxis.set_label_position("top")
     plt.tight_layout()
     plt.title('Confusion matrix', y=1.1)
     plt.ylabel('Actual label')
     plt.xlabel('Predicted label')
     plt.show()
     #Classification Report
     print(classification_report(y_test, y_pred))

     Accuracy Score:  0.9570881706934421
```

Figure 18: Accuracy of XGBOOST.

	precision	recall	f1-score	support
0	1.00	0.91	0.95	39632
1	0.92	1.00	0.96	40043
accuracy			0.96	79675
macro avg	0.96	0.96	0.96	79675
weighted avg	0.96	0.96	0.96	79675

Figure 19: Classification Report.

➢ **CNN model:**

Accuracy of this model, Classification and both accuracy, loss report Training and Validation is as follows:

```
[ ]  #prediction
     y_pred = model.predict(X_test)
     y_pred = np.argmax(y_pred,axis=1)
     #performance
     print ("CNN:Accuracy : ", accuracy_score(y_test_new,y_pred)*100)

     2490/2490 [==============================] - 4s 2ms/step
     CNN:Accuracy :  88.43802949482271
```

Figure 20: Accuracy of CNN model.

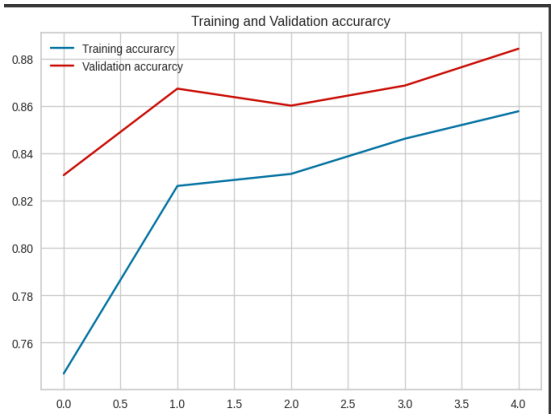

Figure 21: Training and Validation accuracy.

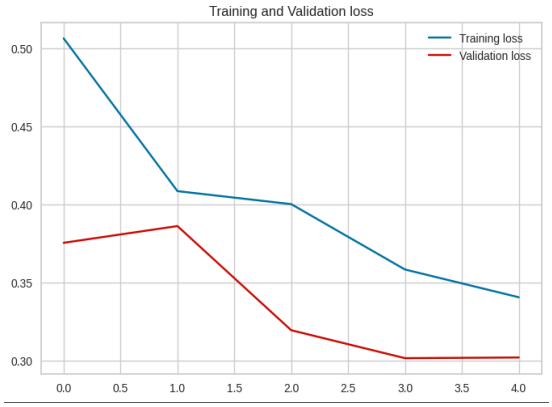

Figure 22: Training and Validation loss.

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

         Yes       0.94      0.82      0.88     39632
          No       0.84      0.95      0.89     40043

    accuracy                           0.88     79675
   macro avg       0.89      0.88      0.88     79675
weighted avg       0.89      0.88      0.88     79675
```

Figure 23: Classification report.

➢ LSTM model:

Accuracy of this model, Classification and both accuracy, loss report Training and Validation is as follows:

```
[ ]  y_pred = model.predict(X_test1)
     y_pred = np.argmax(y_pred,axis=1)
     print ("LSTM:Accuracy : ", accuracy_score(y_test_new,y_pred)*100)

     2490/2490 [==============================] - 7s 3ms/step
     LSTM:Accuracy :  93.16724192030122
```

Figure 24: Accuracy of LSTM model.

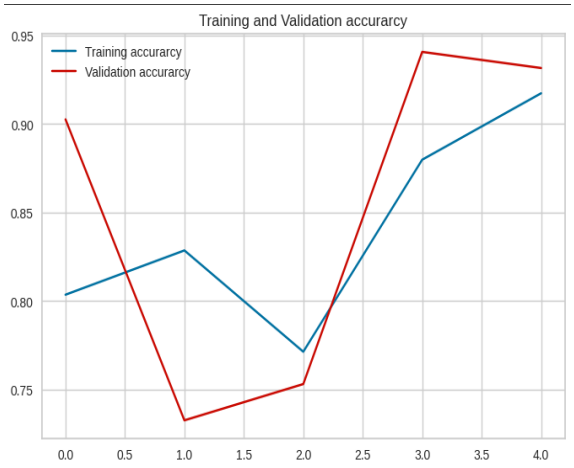

Figure 25: Training and Validation accuracy.

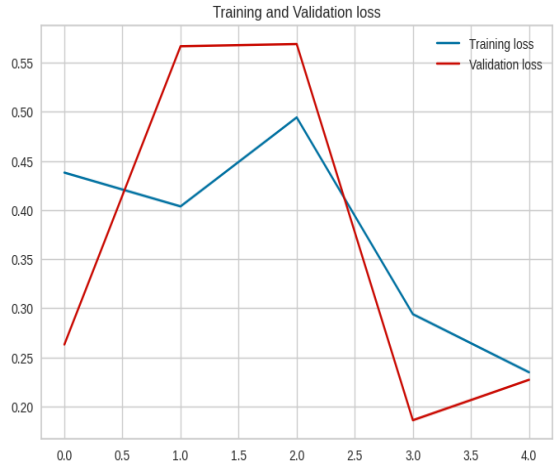

Figure 26: Training and Validation loss.

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

         Yes       0.98      0.88      0.93     39632
          No       0.89      0.98      0.94     40043

    accuracy                           0.93     79675
   macro avg       0.94      0.93      0.93     79675
weighted avg       0.94      0.93      0.93     79675
```

Figure 27: Classification report.

- **BILSTM Model:**

Accuracy of this model, Classification and both accuracy, loss report Training and Validation is as follows:

```
[ ]  y_pred = model.predict(X_test1)
     y_pred = np.argmax(y_pred,axis=1)
     print ("BILSTM:Accuracy : ", accuracy_score(y_test_new,y_pred)*100)

     2490/2490 [==============================] - 11s 4ms/step
     BILSTM:Accuracy :  95.8205208660182
```

Figure 28: Accuracy of BILSTM model.

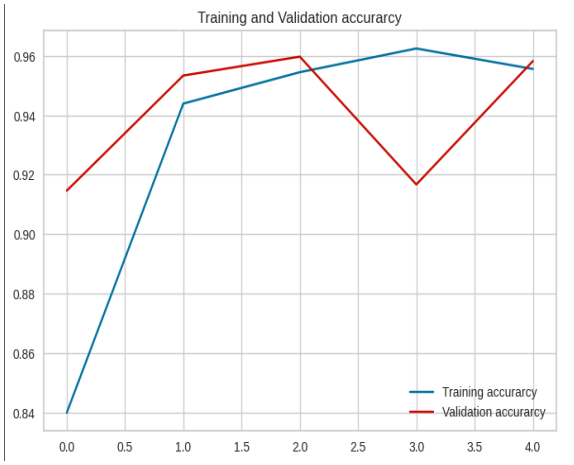

Figure 29: Training and Validation accuracy.

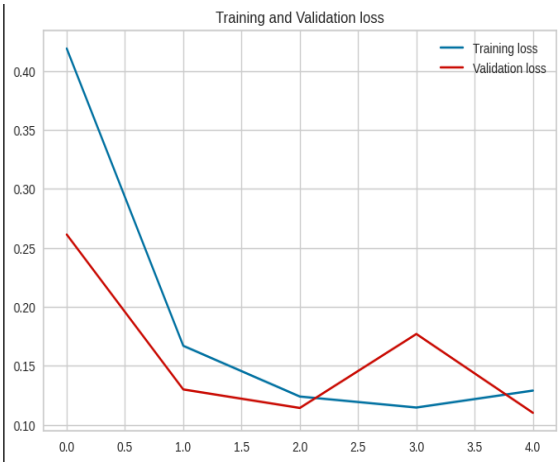

Figure 30: Training and Validation loss.

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

         Yes       0.97      0.95      0.96     39632
          No       0.95      0.97      0.96     40043

    accuracy                           0.96     79675
   macro avg       0.96      0.96      0.96     79675
weighted avg       0.96      0.96      0.96     79675
```

Figure 31: Classification Report.

- **BILSTM with Attention Mechanism:**

Accuracy of this model, Classification and both accuracy, loss report Training and Validation is as follows:

```
[ ]  y_pred = model.predict(X_test1)
     y_pred = np.argmax(y_pred,axis=1)
     print ("BILSTM with Attention:Accuracy : ", accuracy_score(y_test_new,y_pred)*100)

     2490/2490 [==============================] - 11s 4ms/step
     BILSTM with Attention:Accuracy :  99.25698148729212
```

Figure 32: Accuracy of the BILSTM attention mechanism.

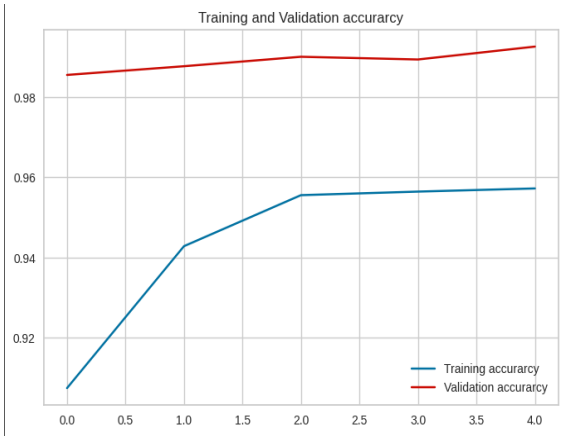

Figure 33: Training and Validation accuracy.

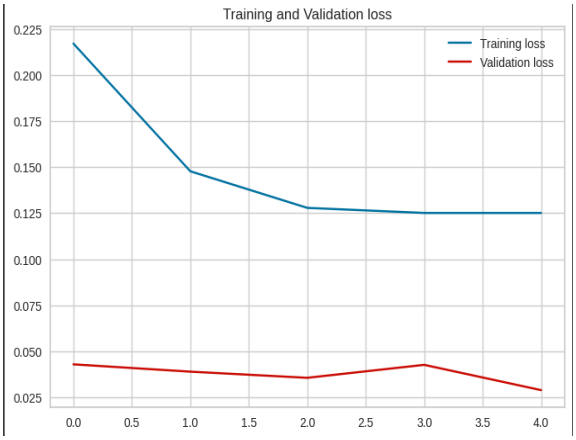

Figure 34: Training and Validation loss.

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

         Yes       1.00      0.99      0.99     39632
          No       0.99      1.00      0.99     40043

    accuracy                           0.99     79675
   macro avg       0.99      0.99      0.99     79675
weighted avg       0.99      0.99      0.99     79675
```

Figure 35: Classification report.

References

Anaconda Navigator (no date). Available at: https://docs.anaconda.com/free/navigator/index.html (Accessed: 1 August 2023).

Keras Library (no date). Available at: https://pypi.org/project/keras/ (Accessed: 1 August 2023).

Scikit Library (no date). Available at: https://pypi.org/project/scikit-learn/ (Accessed: 1 August 2023).

Tensorflow Library (no date). Available at: https://pypi.org/project/tensorflow/ (Accessed: 1 August 2023).