Configuration Manual

MSc Academic Internship

Cyber Security

Shiron Shine Yesudas

Student ID: x22175504

School of Computing

National College of Ireland

Supervisor: Michael Pantridge

Video Link: https://www.youtube.com/watch?v=nW_3VJhM7fE

Introduction

The following paper demonstrates how to build a deep learning-based intrusion detection system (IDS) utilizing ANN and GRU models.
It is based on the findings of the project "Safeguarding Financial Transaction: A Machine learning perspective on online payment network security".

SYSTEM PREREQUISITES:

HARDWARE REQUIREMENTS

- System Processor : Intel i3/ AMD A5
- Hard Disk : 500 GB.
- Ram : 8 GB / 12 GB.
- Any Desktop / Laptop system with the above configuration or higher level.

SOFTWARE REQUIREMENTS

- **Operating system:** Windows 10 / 11 (64 bits OS)
- **Programming Language : Python 3**
- **Framework: Anaconda**
- **Libraries: Keras, TensorFlow, OpenCV**
- **IDE : JupyterNotebook / Pycharm**
- **TOOLS**
  - Visual Code ++
  - Anaconda Navigator
  - Jupyter Notebook
  - Tensor flow, Keras
  - Matplotlip, SKLearn
  - Numpy, Pandas

- **PROGRAMMING LANGUAGES**

  Python 3.4.1

DATASET LINK:

https://data.world/dradar/cicids2017

- CICIDS2017 stands for Canadian Institute for Cybersecurity Intrusion Detection Evaluation Dataset.
- It serves as a benchmark dataset for evaluating intrusion detection systems (IDS).
- The dataset is openly accessible to the general public, facilitating study and assessment.
- It includes tagged network traffic captures (PCAPs) and network flow statistics.
- PCAPs and flow statistics reveal both benign and malicious network activity.
- The primary objective is to assess IDS efficiency in detecting and mitigating cyber attacks.
- With large dimensions, the dataset allows for extensive testing of IDS.
- Reflecting real-world traffic patterns, CICIDS2017 provides a realistic environment for assessment.
- Researchers can utilize it to train and evaluate IDS algorithms in real-world scenarios.
- Analyzing network traffic patterns aids in developing more effective IDS.
- It serves cybersecurity research by offering a significant resource for testing and enhancing IDS.
- By fostering the development of more capable IDS, it contributes to improved cybersecurity posture.
- The dataset includes instances of benign network traffic and common cyber-attacks.
- Sized at 7.92 MB, it is easy to download and study.
- Distributed under a Public Domain license, it allows free use, modification, and redistribution.
- Organized as a single file with 79 columns representing various network traffic aspects.
- Unlike standard table designs, CICIDS2017 lacks tables, potentially requiring additional preparation for analysis.

- Its purpose is to provide real-world examples of network activity for cybersecurity research and analysis.
- Accessible nature and comprehensive coverage make it essential for academics, analysts, and practitioners to understand cybersecurity risks, develop intrusion detection models, and enhance network security.

Steps to segregate and normalize data in preparation for model utilization:

Data Segregation:

Data segregation is essential for the security by partitioning data into distinct components with unique access controls. This tiered strategy diminishes the risk of unauthorized access, data breaches, and leaks, fostering a more secure information environment.

Compliance mandates in sectors like banking, healthcare, and personal data handling underscore the necessity of data segregation. These regulations frequently mandate data isolation to safeguard privacy and ensure proper data handling, aiding enterprises in sidestepping legal and financial repercussions.

Data classification enhances the efficacy of pinpointing, evaluating, and retrieving specific information, particularly within vast datasets. This heightened accessibility to relevant data holds the promise of refining decision-making processes and overall operational efficiency.

By upholding data in its intended state, data segregation fosters data integrity. It curtails the likelihood of errors and data quality issues while mitigating unintended alterations or corruption through controlled data partitioning.

Data Separation: Data segregation represents a systematic approach to internal data management within a company. It optimizes data handling, accessibility, and security, addressing security concerns, aiding compliance endeavors, and culminating in a sturdier and more efficient information infrastructure.

DATA NORMALIZATION:

Data Normalization

```
In [35]: scaler = MinMaxScaler()
         scaler = scaler.fit(X.values)
         scaled_X = scaler.transform(X.values)

         df = pd.DataFrame(data=scaled_X,columns=X.columns)
         df['Label'] = y.values.ravel()
         df.head()
```

Figure 1: Data Normalization

- Utilize the MinMaxScaler function for scaling numerical features in the dataset to a specified range.
- Benefit from scaled features for algorithms reliant on feature scaling.
- Scaling facilitates faster convergence and improved performance of algorithms on the data.

STEPS TO SET UP THE ANN AND GRU MODELS:

1. ARTIFICIAL NEURAL NETWORK (ANN):

- An Artificial Neural Network (ANN) is like a computer brain inspired by how our brains work. It's made of fake neurons connected together in layers: input, hidden, and output. These connections can change to learn patterns from data.

- Each neuron's output is calculated using an activation function. When we give data to the network to make predictions, it's called forward propagation. And when the network adjusts its connections based on mistakes, it's called backpropagation. This process repeats until the network gets good at predicting.

- ANNs are popular because they're flexible and work well in many areas like recognizing images, understanding language, and solving problems. Deep learning uses deep neural networks with lots of hidden layers, making ANNs good at handling complex information, so they're used a lot in today's AI systems.

Algorithm-1 : ArtificialNeuralNetwork

```
In [10]: from tensorflow.keras import Sequential
         from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization
         from tensorflow.keras.backend import clear_session
```

```
In [11]: ann_model = Sequential()
         ann_model.add(Input(shape=(X_train.shape[1],)))
         ann_model.add(Dense(32, activation="relu"))
         ann_model.add(Dropout(0.5))
         ann_model.add(Dense(64, activation="relu"))
         ann_model.add(Dropout(0.5))
         ann_model.add(Dense(128, activation='relu'))
         ann_model.add(Dense(5, activation='softmax'))
         ann_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 2: Importing the necessary sequential feature

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                992

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 64)                2112

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 128)               8320

 dense_3 (Dense)             (None, 5)                 645

=================================================================
Total params: 12,069
Trainable params: 12,069
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Model Summary for ANN

2. GATED RECURRENT UNIT (GRU):

GRUs are a special type of neural network that help to remember patterns in data that comes in a sequence. They fix some problems that normal neural networks have in keeping track of these patterns. They were made to be better than another type of network called LSTM. GRUs use gates to control how information flows through the network. These gates help the network decide what information to keep and what to forget, so it can understand long sequences better without getting stuck. GRUs are good at tasks like understanding languages and analyzing feelings in text. Because of how they work, they're really useful for understanding and predicting sequences of events, which is important in deep learning.

Algorithm: 2 Gated Recurrent Unit

In [26]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Flatten,Dense,Dropout,GRU,BatchNormalization
```

In [27]:
```
gru_model = Sequential()
gru_model.add(GRU(units=200, return_sequences=True, input_shape=(x_train.shape[1],x_train.shape[2])))
gru_model.add(GRU(units=200, return_sequences=True))
gru_model.add(Flatten())
gru_model.add(Dropout(0.4))
gru_model.add(Dense(256, activation='relu'))
gru_model.add(Dense(5, activation='softmax'))
gru_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 4: Importing the necessary sequential features

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 30, 200)           121800

 gru_1 (GRU)                 (None, 30, 200)           241200

 flatten (Flatten)           (None, 6000)              0

 dropout (Dropout)           (None, 6000)              0

 dense (Dense)               (None, 256)               1536256

 dense_1 (Dense)             (None, 5)                 1285

=================================================================
Total params: 1,900,541
Trainable params: 1,900,541
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Model Summary for GRU

STEPS TO EVALUATE THE MODELS USING ACCURACY AND LOSS METRICS, CLASSIFICATION REPORT, AND CONFUSION MATRIX:

1. ARTIFICIAL NEURAL NETWORK (ANN)

1.1 CLASSIFICATION REPORT– ANN

This is a classification report on a machine learning model's performance on a dataset with five classes: BENIGN, Brute Force, DDoS, DoS, and PortScan. Precision, recall, and f1-score values for each class are presented, demonstrating the model's ability to properly identify instances of that class. The overall accuracy of the model is 95%, with high precision and recall for the majority of classes, demonstrating its utility. Notably, the Brute Force and DDoS classes have excellent precision and recall, whereas DoS has far lower precision and recall. The macro and weighted averages illustrate the general balanced performance across classes. In summary, the model achieves an impressive 95% accuracy when distinguishing between different types of network data.

```
In [18]: class_labels = ['BENIGN', 'Brute Force', 'DDoS', 'DoS', 'PortScan']
```

Figure 6: The featured class labels

```
In [20]: print(classification_report(y_true=true_labels,y_pred=ann_pred, target_names=class_labels))
```

	precision	recall	f1-score	support
BENIGN	0.92	0.90	0.91	997
Brute Force	0.97	1.00	0.98	1047
DDoS	0.97	1.00	0.98	993
DoS	0.98	0.90	0.94	1003
PortScan	0.94	0.97	0.95	1077
accuracy			0.95	5117
macro avg	0.95	0.95	0.95	5117
weighted avg	0.95	0.95	0.95	5117

Figure 7: Classification Report for ANN

1.2 CONFUSION MATRIX-ANN

The table here shows how well a computer model did at figuring out different types of things in a group of 5117 examples. For example, it got 899 cases right in one group called "BENIGN" but got 98 cases wrong. In another group called "Brute Force", it got 1044 right and only 3 wrong. It did pretty well in a group called "DDOS", getting 990 right and 3 wrong. In another group called "DOS", it got 907 right and 96 wrong. Finally, in a group called "PortScan", it got 1045 right and 32 wrong. This table helps us see how good the model is at putting things in the right groups, but it also shows where it made mistakes. It seems like the model is better at figuring out things that aren't what we're looking for, but it still makes some mistakes, especially in certain groups.

```
In [21]: plt.figure(figsize=(10,5))
         plt.rcParams['font.size'] = 15
         sns.heatmap(data=confusion_matrix(y_true=true_labels, y_pred=ann_pred),
                     annot=True,
                     cmap=plt.cm.Blues,
                     fmt = "5d",
                     xticklabels=class_labels,
                     yticklabels=class_labels,
                     cbar=False)
         plt.title(label="Confusion Matrix")
         plt.xticks(rotation=90)
         plt.yticks(rotation=0)
         plt.show()
```

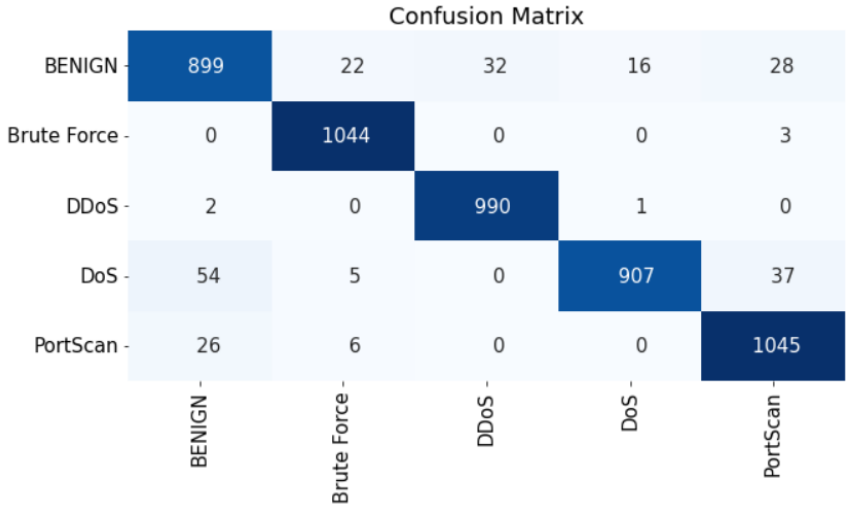


Figure 8: Confusion Matrix for ANN

1.3 ACCUARY PLOT GRAPH -ANN

An accuracy plot shows how well a model works overtime or in different situations. On the graph, you usually see the number of times the model learns (epochs or iterations) on the bottom line and how accurate it is on the up-and-down line. This picture helps us understand how well the model learns and how good it is at solving problems it hasn't seen before. If the line goes up, the model is getting better. But if it stays flat or goes up and down a lot, there might be problems. Accuracy

charts help us see how well our model is doing during training, find out if it's learning too much from the data (overfitting), or not enough (underfitting), and decide how to make it better. Looking at these graphs helps scientists and programmers make models work better, making them more reliable and useful.

```
In [13]: history=ann_model.fit(x=X_train.values,y=y_train,batch_size=64,epochs=10,validation_data=(X_test.values,y_test))

Epoch 1/10
320/320 [==============================] - 1s 2ms/step - loss: 1.0366 - accuracy: 0.5925 - val_loss: 0.4536 - val_accuracy: 0.9
074
Epoch 2/10
320/320 [==============================] - 1s 2ms/step - loss: 0.5100 - accuracy: 0.8394 - val_loss: 0.2614 - val_accuracy: 0.9
134
Epoch 3/10
320/320 [==============================] - 1s 2ms/step - loss: 0.3731 - accuracy: 0.8873 - val_loss: 0.2186 - val_accuracy: 0.9
250
Epoch 4/10
320/320 [==============================] - 1s 2ms/step - loss: 0.3306 - accuracy: 0.8980 - val_loss: 0.1949 - val_accuracy: 0.9
291
Epoch 5/10
320/320 [==============================] - 0s 2ms/step - loss: 0.2943 - accuracy: 0.9061 - val_loss: 0.1732 - val_accuracy: 0.9
279
Epoch 6/10
320/320 [==============================] - 1s 2ms/step - loss: 0.2721 - accuracy: 0.9130 - val_loss: 0.1620 - val_accuracy: 0.9
355
Epoch 7/10
320/320 [==============================] - 1s 2ms/step - loss: 0.2460 - accuracy: 0.9216 - val_loss: 0.1491 - val_accuracy: 0.9
384
Epoch 8/10
320/320 [==============================] - 1s 2ms/step - loss: 0.2359 - accuracy: 0.9258 - val_loss: 0.1322 - val_accuracy: 0.9
369
Epoch 9/10
320/320 [==============================] - 1s 2ms/step - loss: 0.2224 - accuracy: 0.9313 - val_loss: 0.1375 - val_accuracy: 0.9
457
Epoch 10/10
320/320 [==============================] - 1s 2ms/step - loss: 0.2156 - accuracy: 0.9331 - val_loss: 0.1233 - val_accuracy: 0.9
420
```

Figure 9: Train the values in x and y

```
with plt.style.context(style='bmh'):
    plt.figure(figsize=(18,8))
    plt.plot(history.history["accuracy"],label="accuracy",marker="o",markersize=10)
    plt.plot(history.history["val_accuracy"],label="val_accuracy",marker="*",markersize=10)
    plt.title(label="accuracy plot-graphs")
    plt.xlabel(xlabel='Epochs')
    plt.ylabel(ylabel='Accuracy')
    plt.xticks(range(0,10))
    plt.legend()
    plt.show()
```

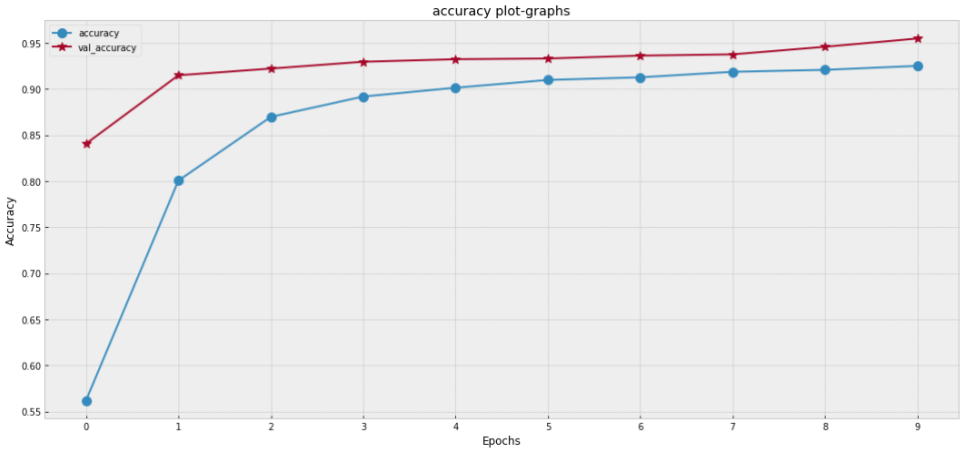


Figure 10: Accuracy plot Graph

1.4 LOSS PLOT GRAPH – ANN

A loss plot graph shows how a machine learning model's performance changes over time. It uses the x-axis to show time or steps and the y-axis to show how well the model is doing (lower is better). The goal is to make the model perform better by minimizing this value. When the graph goes down, it means the model is getting better. If it suddenly goes up, it might mean there's a problem like the model learning too much from the training data or having trouble converging. Checking the loss plot regularly helps us understand how the model is learning and fix any issues. This helps us make the model more accurate and reliable by adjusting things like hyperparameters or its structure.

```
plt.figure(figsize=(18,8))
plt.plot(history.history["loss"],label="loss",marker="o",markersize=10)
plt.plot(history.history["val_loss"],label="val_loss",marker="*",markersize=10)
plt.title(label="loss plot-graphs")
plt.xlabel(xlabel='Epochs')
plt.ylabel(ylabel='Loss')
plt.xticks(range(0,10))
plt.legend()
plt.show()
```

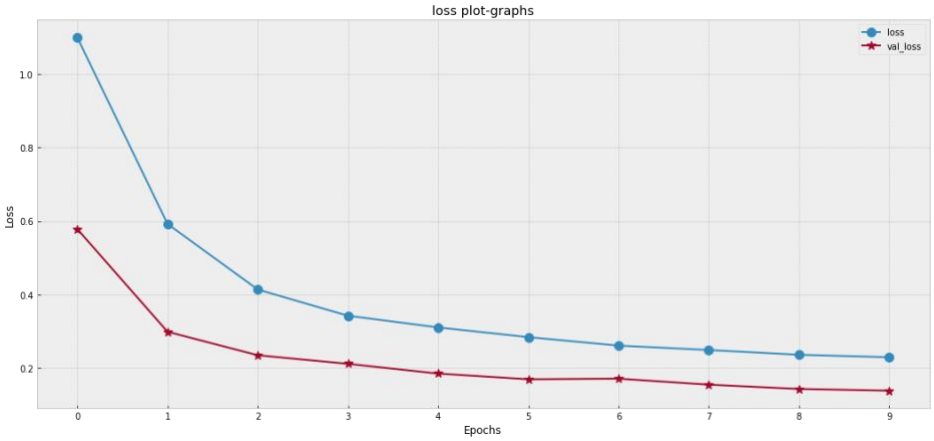


Figure 11: Loss plot Graph

2. GATED RECURRENT UNIT (GRU)

2.1 CLASSIFICATION REPORT– GRU

This report checks how well a model works on a set of data that has five different types of things in it: BENIGN, Brute Force, DDoS, DoS, and PortScan. The model is really accurate, with an overall score of 99%. Each type of thing in the data has really good scores too, which means the model can recognize each type really well. BENIGN is especially perfect, and the others are almost perfect, showing the model works really well. The model's good performance is confirmed by some other scores too, which show it can find things right across all the types. So basically, the model is super good at recognizing things in the data, getting a 99% score overall.

```
In [35]: print(classification_report(y_true=true_labels,y_pred=gru_pred, target_names=class_labels))
```

	precision	recall	f1-score	support
BENIGN	0.98	0.96	0.97	1026
Brute Force	0.99	1.00	0.99	1032
DDoS	0.98	1.00	0.99	1008
DoS	1.00	0.99	0.99	1006
PortScan	0.99	1.00	0.99	1045
accuracy			0.99	5117
macro avg	0.99	0.99	0.99	5117
weighted avg	0.99	0.99	0.99	5117

Figure 12: Classification report for GRU

2.2 CONFUSION MATRIX FOR GRU:

The confusion matrix shows how well the GRU model performed on a dataset with 5117 samples. It correctly predicted 966 BENIGN events but got 31 wrong. For Brute Force, it got 1044 out of 1044 right, with only three mistakes. In the DDOS category, it got 991 right and 2 wrong. In DOS, it got 999 right and 4 wrong. For PortScan, it got 1071 right and 6 wrong. This matrix helps us see where the model is good and where it needs improvement. It's good at identifying most cases correctly, but it still makes some mistakes, especially in BENIGN and PortScan categories. This tells us there's room to make it more accurate overall.

```
In [36]: plt.figure(figsize=(10,5))
         plt.rcParams['font.size'] = 15
         sns.heatmap(data=confusion_matrix(y_true=true_labels, y_pred=gru_pred),
                     annot=True,
                     cmap=plt.cm.Blues,
                     fmt = "5d",
                     xticklabels=class_labels,
                     yticklabels=class_labels,
                     cbar=False)
         plt.title(label="Confusion Matrix")
         plt.xticks(rotation=90)
         plt.yticks(rotation=0)
         plt.show()
```

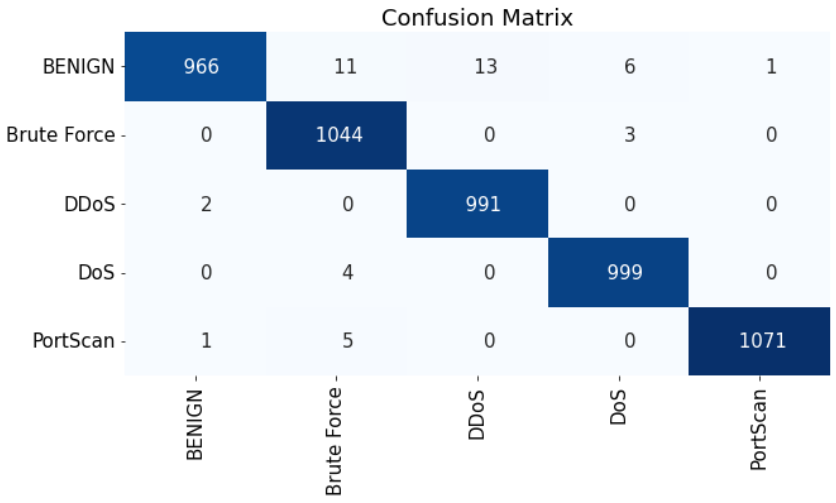


Figure 13: confusion matrix for GRU

2.3 ACCURACY PLOT GRAPH FOR GRU :

An accuracy plot shows how well a model works as time goes on or when you change its settings. The horizontal line shows the number of times the model learns, while the vertical line shows how accurate it is. It helps quickly see how well the model is learning and making predictions. This picture is important in machine learning because it helps keep track of progress, find if the model is fitting too much or too little, and decide how to make it work better for specific tasks.

```
In [29]: history=gru_model.fit(x=x_train,y=y_train,batch_size=32,epochs=10,validation_data=(x_test,y_test))

         Epoch 1/10
         640/640 [==============================] - 43s 63ms/step - loss: 0.3571 - accuracy: 0.8849 - val_loss: 0.2028 - val_accuracy:
         0.9429
         Epoch 2/10
         640/640 [==============================] - 41s 63ms/step - loss: 0.1745 - accuracy: 0.9427 - val_loss: 0.1212 - val_accuracy:
         0.9539
         Epoch 3/10
         640/640 [==============================] - 40s 63ms/step - loss: 0.1352 - accuracy: 0.9533 - val_loss: 0.0931 - val_accuracy:
         0.9595
         Epoch 4/10
         640/640 [==============================] - 40s 63ms/step - loss: 0.1065 - accuracy: 0.9658 - val_loss: 0.0852 - val_accuracy:
         0.9646
         Epoch 5/10
         640/640 [==============================] - 39s 60ms/step - loss: 0.0831 - accuracy: 0.9738 - val_loss: 0.0569 - val_accuracy:
         0.9789
         Epoch 6/10
         640/640 [==============================] - 38s 59ms/step - loss: 0.0714 - accuracy: 0.9767 - val_loss: 0.0578 - val_accuracy:
         0.9810
         Epoch 7/10
         640/640 [==============================] - 38s 59ms/step - loss: 0.0573 - accuracy: 0.9812 - val_loss: 0.0515 - val_accuracy:
         0.9810
         Epoch 8/10
         640/640 [==============================] - 38s 59ms/step - loss: 0.0482 - accuracy: 0.9844 - val_loss: 0.0549 - val_accuracy:
         0.9824
         Epoch 9/10
         640/640 [==============================] - 37s 58ms/step - loss: 0.0484 - accuracy: 0.9858 - val_loss: 0.0641 - val_accuracy:
         0.9828
         Epoch 10/10
         640/640 [==============================] - 38s 60ms/step - loss: 0.0429 - accuracy: 0.9877 - val_loss: 0.0484 - val_accuracy:
         0.9881
```

Figure 14: train the values x and y

```
with plt.style.context(style='fivethirtyeight'):
    plt.figure(figsize=(18,8))
    plt.plot(history.history["accuracy"],label="accuracy",marker="o",markersize=10)
    plt.plot(history.history["val_accuracy"],label="val_accuracy",marker="*",markersize=10)
    plt.title(label="accuracy plot-graphs")
    plt.xlabel(xlabel='Epochs')
    plt.ylabel(ylabel='Accuracy')
    plt.xticks(range(0,10))
    plt.legend()
    plt.show()
```

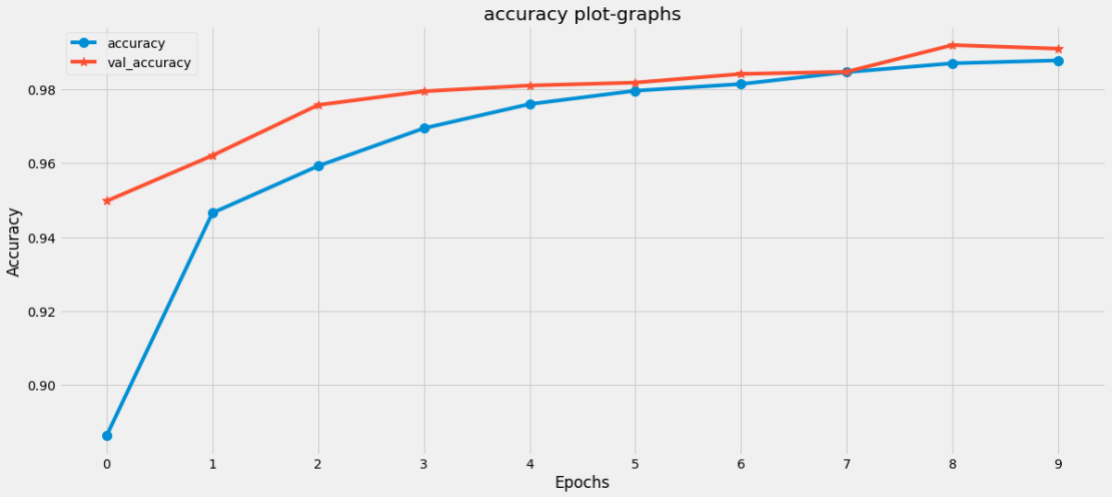


Figure 15: Accuracy plot Graph

2.4 LOSS PLOT GRAPH FOR GRU

A loss plot shows how a machine learning model's mistakes change over time. It uses a graph where the horizontal line shows how many times the model has been trained (iterations or epochs), and the vertical line shows how many mistakes it's making (loss values). The goal of training is to make fewer mistakes, so when the line goes down, it means the model is getting better. If the line suddenly goes up, it might mean there's a problem, like the model is learning too much from the training data and not enough from other data (overfitting). Checking the loss plot regularly is important to understand how the model is learning and to make it better at its job.

```
plt.figure(figsize=(18,8))
plt.plot(history.history["loss"],label="loss",marker="o",markersize=10)
plt.plot(history.history["val_loss"],label="val_loss",marker="*",markersize=10)
plt.title(label="loss plot-graphs")
plt.xlabel(xlabel='Epochs')
plt.ylabel(ylabel='Loss')
plt.xticks(range(0,10))
plt.legend()
plt.show()
```

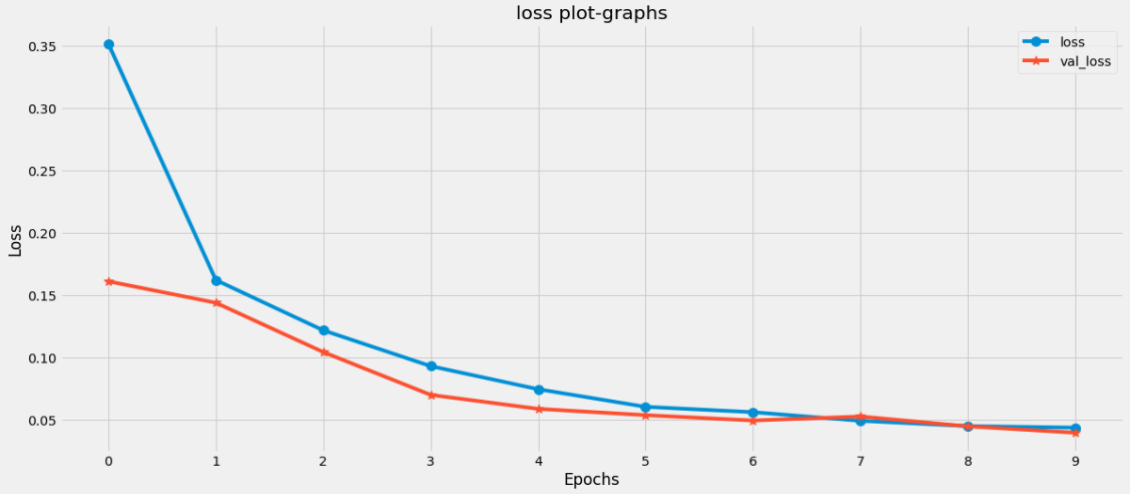


Figure 16: Loss plot Graph for GRU

TESTING Phase

1.Server side validation:

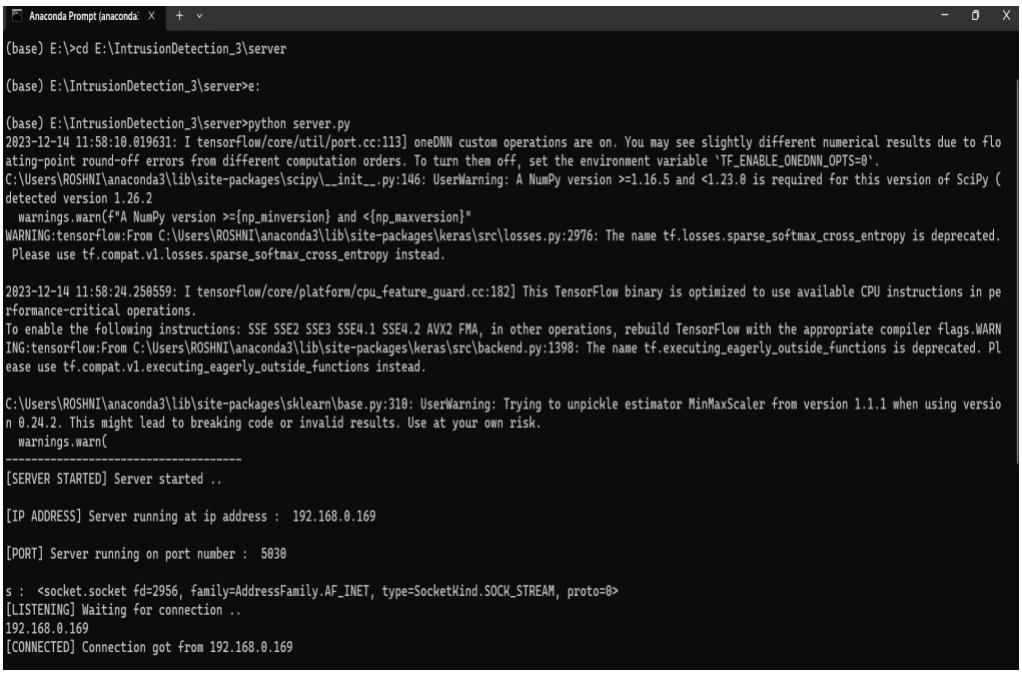


Figure 17: Getting connection from the client side

```
[RECEIVING] Receiving file from client

[PREDICTION] Predicting...

1/1 [==============================] - 1s 988ms/step
1/1 [==============================] - 0s 79ms/step
0
Benign
1.0
result_ :  Benign
[RESULT] Predicted result is  Benign
-----------------------------------
```

Figure 18:result of the Normal file

```
192.168.0.169
[CONNECTED] Connection got from 192.168.0.169

[RECEIVING] Receiving file from client

[PREDICTION] Predicting...

1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 40ms/step
2
DDoS
0.8952705
result_ :  DDoS
[RESULT] Predicted result is  DDoS
-----------------------------------

Email sent successfully
```

Figure 19:Prediction result of the attacked file

2. CLIENT SIDE TESTING :

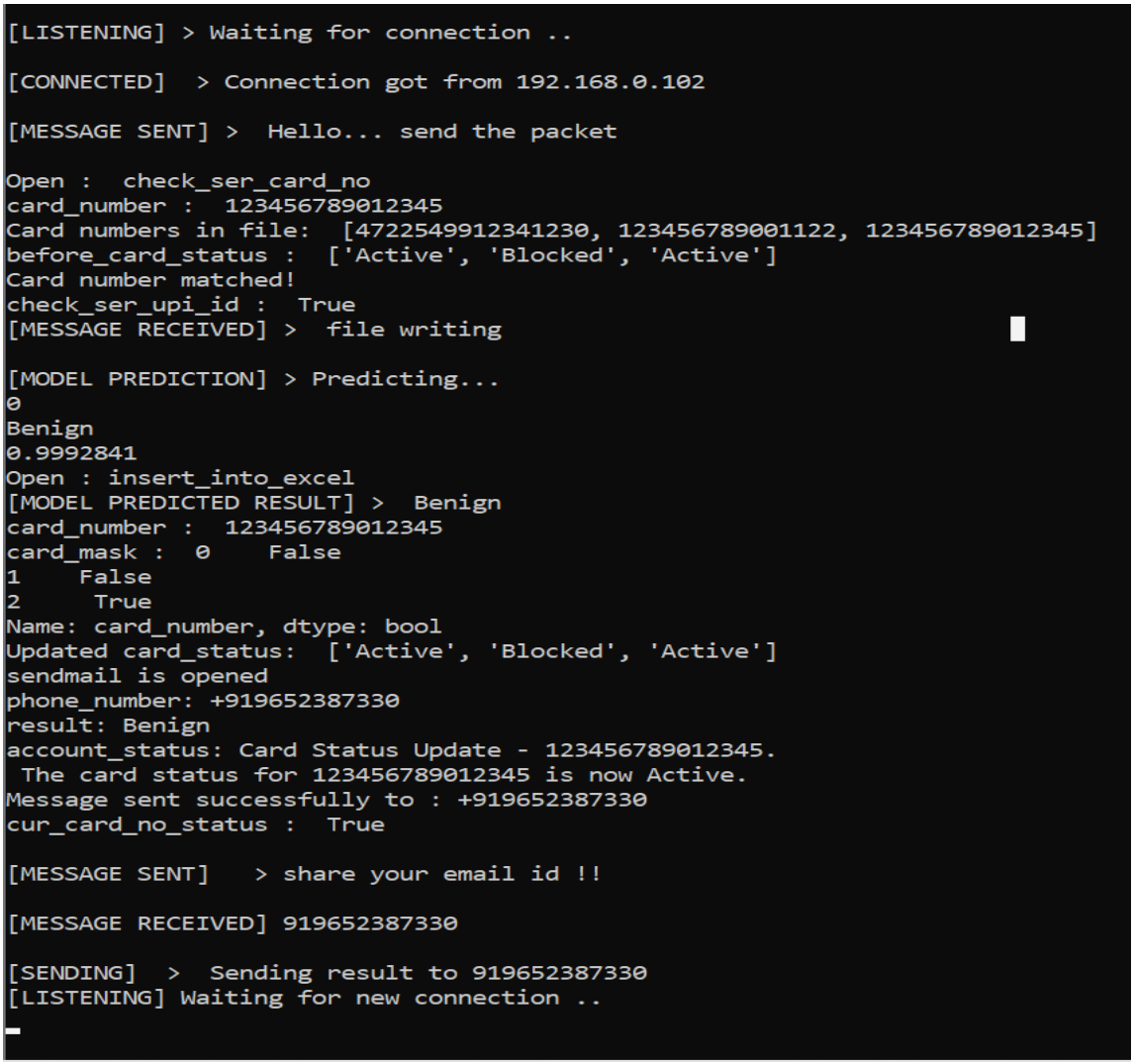


Figure 20:Client side statements

```
[LISTENING] Waiting for new connection ..

[CONNECTED]  > Connection got from 192.168.0.102

[MESSAGE SENT] >  Hello... send the packet

Open :  check_ser_upi_id
upi_id :  rajesh@ybl
upi id in file:  ['rajan@oksbi', 'shiva@gml', 'rajesh@ybl']
before_upi_status:  ['Active', 'Blocked', 'Active']
upi id matched!
check_ser_upi_id :  True
[MESSAGE RECEIVED] >  file writing

[MODEL PREDICTION] > Predicting...
1
Brute Force
0.9966185
Open : insert_into_excel
[MODEL PREDICTED RESULT] >  Brute Force
upi_id :  rajesh@ybl
upi_mask :  0    False
1    False
2     True
Name: upi_id, dtype: bool
Updated upi_id_status:  ['Active', 'Blocked', 'Blocked']
sendmail is opened
phone_number: +919652387330
result: Brute Force
account_status: UPI ID Status Update - rajesh@ybl.
 The UPI ID status for rajesh@ybl is now Blocked.
Message sent successfully to : +919652387330
upi_id_status :  False
Account is Blocked.

[LISTENING] Waiting for new connection ...
```

Figure 21:Importing files from libraries and used secret key to encrypt data

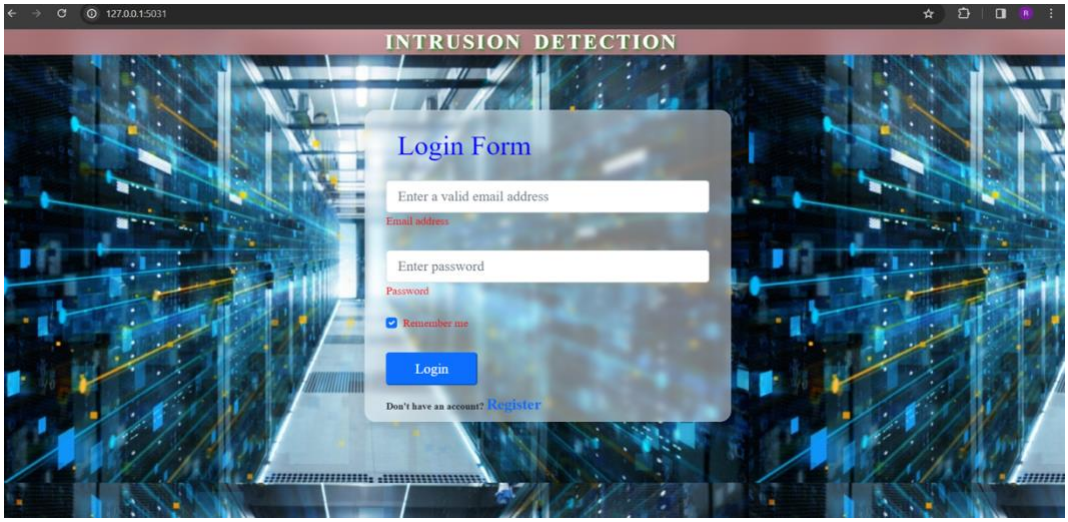


Figure 22: Login form of the forntend process

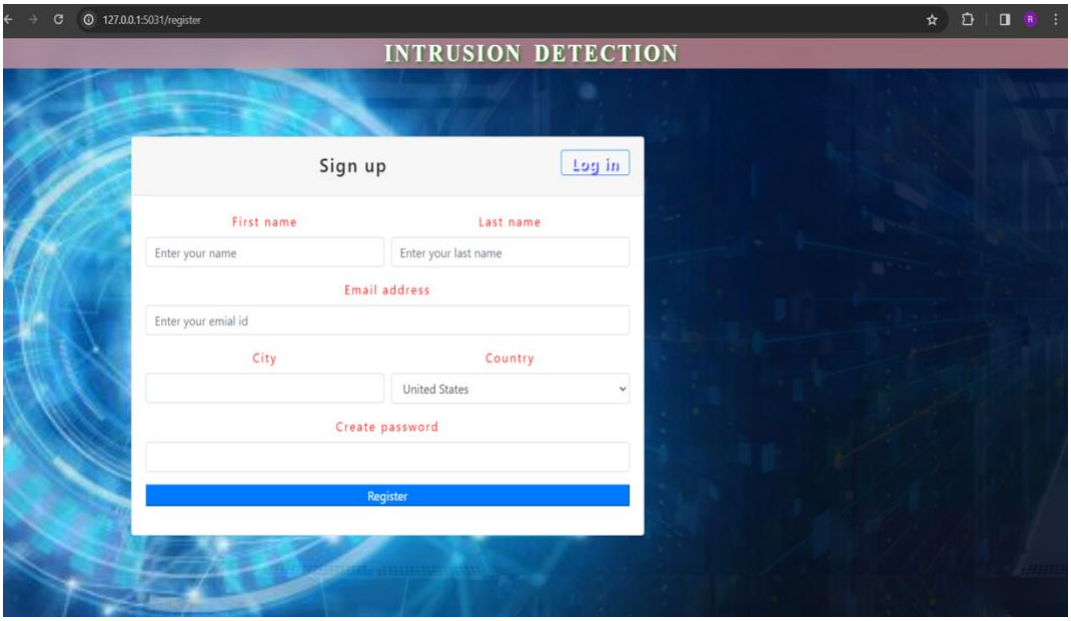
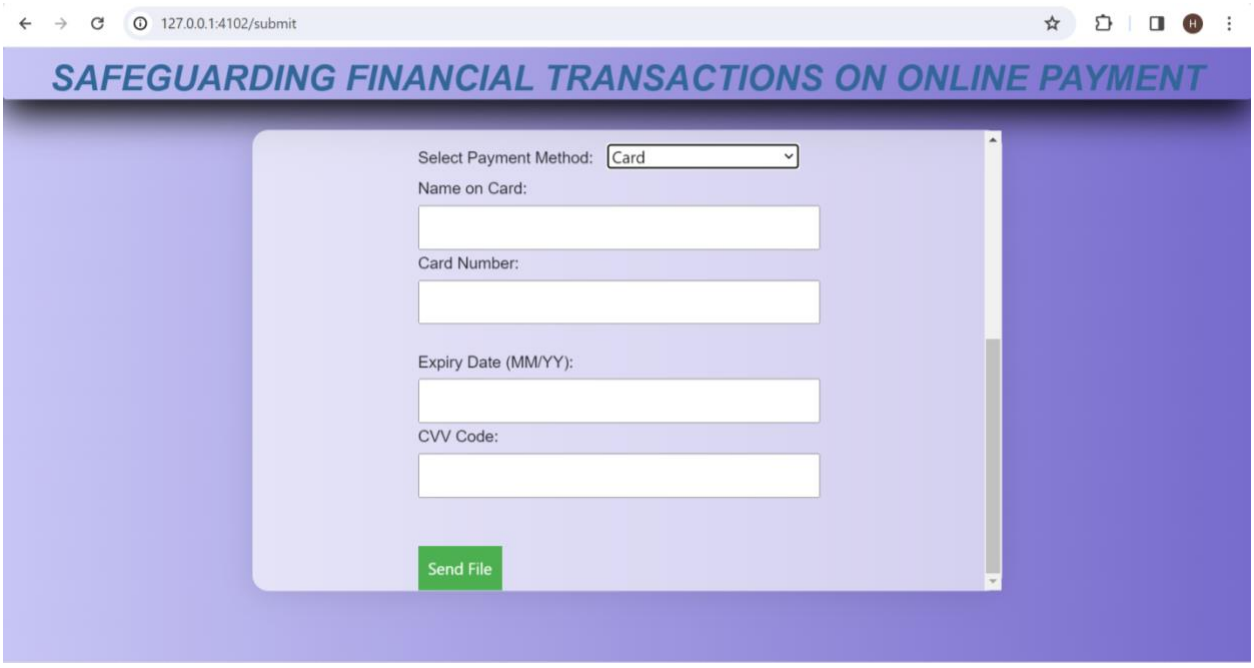


Figure 23: Registration form



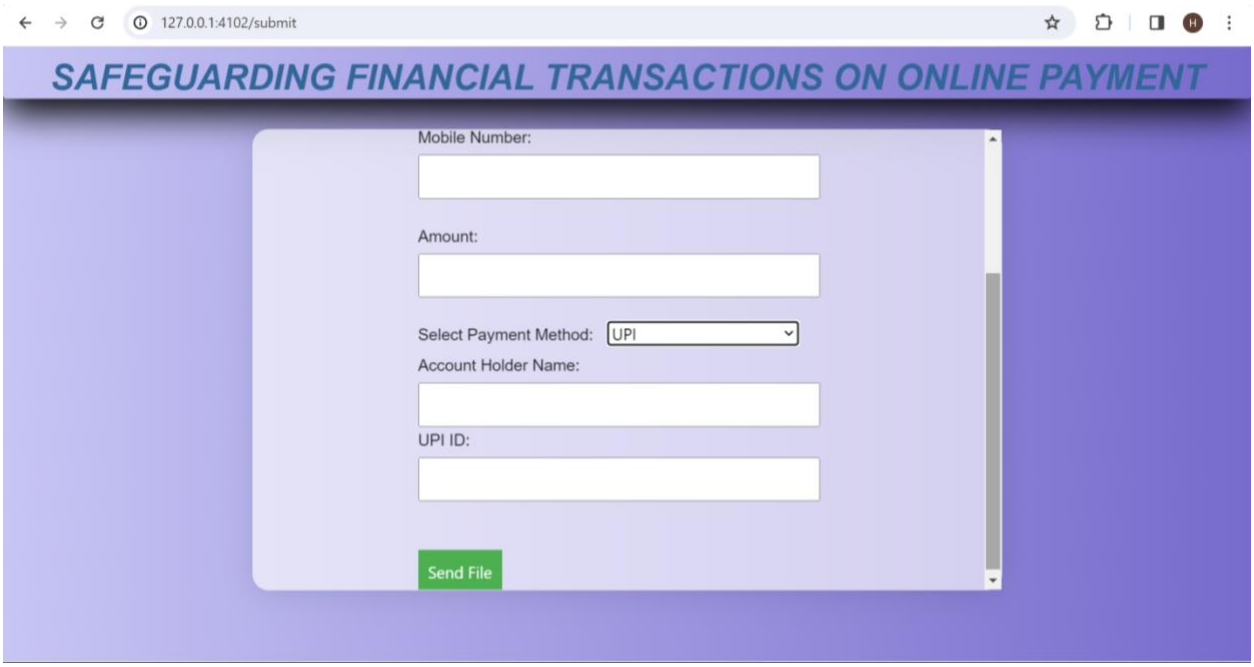


Figure 24: Displays the all clients

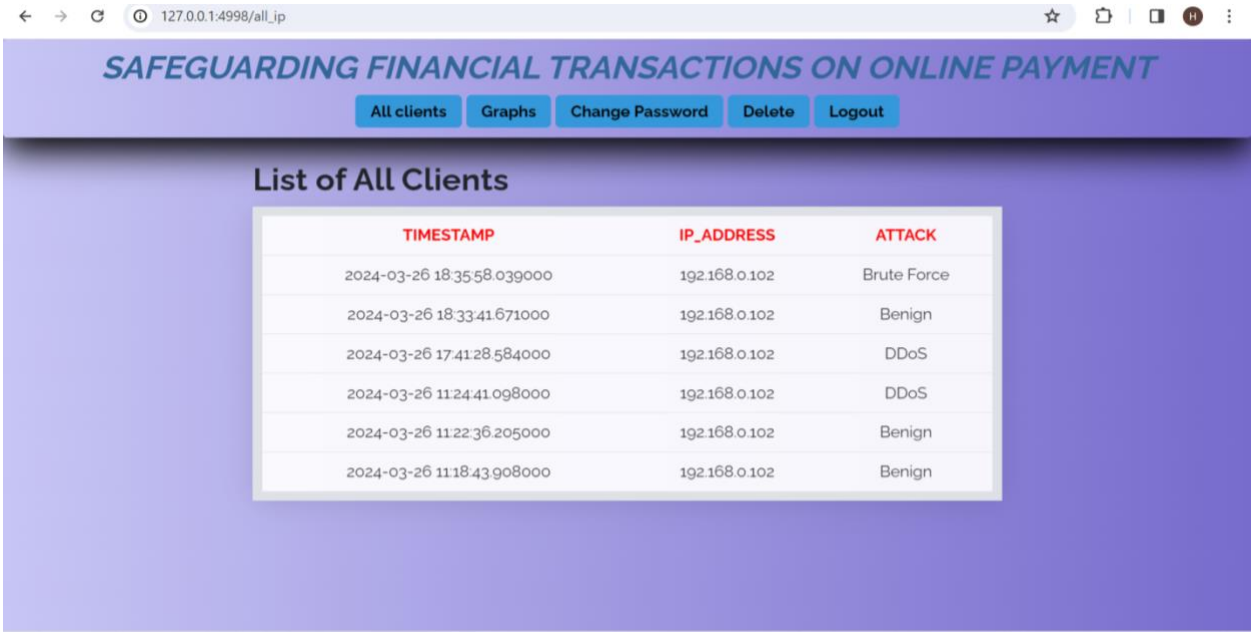


Figure 25: Blocked clients

RESULT ANALYSIS:

This study made intrusion detection systems (IDS) better by using advanced deep learning methods like artificial neural networks (ANN) and Gated Recurrent Units (GRU). They tested these methods with lots of different intrusion types using a dataset called CICIDS2017. The results showed that deep learning can greatly improve how accurately and quickly IDS can detect intrusions. The ANN and GRU models they trained did really well, with accuracy rates of 85.47% and 79.10% in spotting five different types of intrusions.

Artificial Neural Network(ANN):

```
In [18]: class_labels = ['BENIGN', 'Brute Force', 'DDoS', 'DoS', 'PortScan']
```

Accuracy Score

```
In [19]: ann_model_accuracy=accuracy_score(y_true=true_labels,y_pred=ann_pred)
         print("Validation accuracy of ArtificialNeuralNetwork model is {:.2f}%".format(ann_model_accuracy*100))

         Validation accuracy of ArtificialNeuralNetwork model is 94.20%
```

Gated Recurrent Unit (GRU):

```
In [34]: gru_model_accuracy=accuracy_score(y_true=true_labels,y_pred=gru_pred)
         print("Validation accuracy of GatedRecurrentUnit model is {:.2f}%".format(gru_model_accuracy*100))

         Validation accuracy of GatedRecurrentUnit model is 98.81%
```