National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Pushpak Attarde
Student ID: X22211721

School of Computing
National College of Ireland

Supervisor: Barry Haycock

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. Pushpak Attarde……

Student ID: ……x22211721……..……

Programme: ……MSc in Data Analytics…… **Year:** ……2023-2024.

Module: ……MSc Research Project…..……..……

Lecturer: ……Barry Haycock……..……

Submission Due Date: ……12th August 2024……..……

Project Title: …Advancing Intrusion Detection Systems on the Internet of Vehicles: Mitigating DoS and Spoofing Attacks in CAN Bus Network……..……

Word Count: ……1029…….…… **Page Count:** ……9……..……..……

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: ……Pushpak……

Date: ……11th August 2024……

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

1. Introduction

The CICIoV2024 (Conference on IoT and Cybersecurity in Intelligent Vehicles) is aimed at the development of novel research to enhance the security defence for the CAN bus (Controller Area Network) used in the Internet of Vehicles (IoV). With new existing car technology coming with connectivity features, cars are exposed to a new set of advanced hacks like Denial-of-Service attacks and spoofing attacks. Among them, the objectives of the conference include the identification of feasible IDS solutions to protect the IoV environment.

2. Overview of the program

From this conference, CICIoV2024 will be a major contribution towards the realization of collaborative effort in establishing sound security that will cover the safety, privacy, and reliability of intelligent vehicles, in a growing context of the digital environment. This means that scientists, engineers, and professionals performing or engaged in cybersecurity practices, tasks, or projects will assemble to share knowledge, ideas, and trends, and put forward proper solutions that may address these cybersecurity issues.

3. Hardware/software requirements

3.1. Hardware production

The following hardware settings are recommended for smooth operation.

- Processor: Intel Core i5
- RAM: 16 GB
- Storage: 512GB
- GPU: NVIDIA RTX 3050

3.2. Software

The following software is required for the project:

- Jupyter Notebook: Used to run and write code.
- Anaconda: To manage the Python environment and dependencies.

Version Requirements:

- Jupyter Notebook: Version 7
- Anaconda: Version 1.10

Ensure that the necessary Python libraries are installed. These include panda, NumPy, matplotlib, seaborn, scikit-learn, and TensorFlow.

4. The data set

The dataset used for this project came from Canadian Institute for Cybersecurity Intrusion Detection System. The data is provided in IDS dataset section and named as Realistic IDS-DoS and spoofing attack in IoV (CICIoV2024):

- Dataset: CICIoV2024.tar.xz.gz

5. Implementation of Project in Jupyter

Explanation of the Code

5.1. Import Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
```

- Pandas: It is used for analysis and for manipulation of dataset.
- NumPy: This library is specifically used numerically based operations.
- matplotlib.pyplot: A visual library that allows these kinds of charts to be implemented, be it in static, interactive, or animated forms.
- warnings: This is to filter out the warnings.
- seaborn: Doing statistical data visualization.
- Label Encoder: It converts the categorical labels into numerical format.

5.2. Load and Inspect Data

Reading of data

```
df_dos = pd.read_csv("hexadecimal/hexadecimal_DoS.csv")
df_benign = pd.read_csv("hexadecimal/hexadecimal_benign.csv")
df_spf_gas = pd.read_csv("hexadecimal/hexadecimal_spoofing-GAS.csv")
df_spf_rpm = pd.read_csv("hexadecimal/hexadecimal_spoofing-RPM.csv")
df_spf_speed = pd.read_csv("hexadecimal/hexadecimal_spoofing-SPEED.csv")
df_spf_sw = pd.read_csv("C:/Users/Algorithms Solutions/Downloads/CICIoV2024.tar.xz/CICIoV2024/hexadecimal/hexadecimal_spoofing-STEERING_WHEEL.csv")
```

- pd.read_csv: Pandas library is being called over here to read the CSV file specially.

```
df_dos.head()
```

	Interface	ID	DATA_0	DATA_1	DATA_2	DATA_3	DATA_4	DATA_5	DATA_6	DATA_7	label	category	specific_class
0	slcan0	123	00	00	00	00	00	00	00	00	ATTACK	DoS	DoS
1	slcan0	123	0E	0B	04	04	03	03	08	0C	ATTACK	DoS	DoS
2	slcan0	123	0E	0B	04	04	03	03	08	0C	ATTACK	DoS	DoS
3	slcan0	123	0E	0B	04	04	03	03	08	0C	ATTACK	DoS	DoS
4	slcan0	123	0E	0B	04	04	03	03	08	0C	ATTACK	DoS	DoS

- head(): It is to look at the first few rows of the data frames for a quick inspection.

5.3. Data Overview

```
df_dos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74663 entries, 0 to 74662
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Interface       74663 non-null  category
 1   ID              74663 non-null  int64
 2   DATA_0          74663 non-null  category
 3   DATA_1          74663 non-null  category
 4   DATA_2          74663 non-null  object
 5   DATA_3          74663 non-null  object
 6   DATA_4          74663 non-null  object
 7   DATA_5          74663 non-null  object
 8   DATA_6          74663 non-null  object
 9   DATA_7          74663 non-null  object
 10  label           74663 non-null  object
 11  category        74663 non-null  object
 12  specific_class  74663 non-null  object
dtypes: category(3), int64(1), object(9)
memory usage: 5.9+ MB
```

- info(): This is used to quickly assess the structure of a DataFrame, check for missing data, and understand the memory footprint of your data.

```
df_dos.isnull().sum()
```

```
Interface         0
ID                0
DATA_0            0
DATA_1            0
DATA_2            0
DATA_3            0
DATA_4            0
DATA_5            0
DATA_6            0
DATA_7            0
label             0
category          0
specific_class    0
dtype: int64
```

- isnull().sum(): It checks the missing value of each column.

```
final_df = pd.concat([df_dos, df_benign, df_spf_gas, df_spf_rpm, df_spf_speed, df_spf_sw])
```

```
final_df.head()
```

	Interface	ID	DATA_0	DATA_1	DATA_2	DATA_3	DATA_4	DATA_5	DATA_6	DATA_7	label	category	specific_class	DLC
0	slcan0	123	00	00	00	00	00	00	00	00	ATTACK	DoS	DoS	NaN
1	slcan0	123	0E	0B	04	04	03	03	08	0C	ATTACK	DoS	DoS	NaN
2	slcan0	123	0E	0B	04	04	03	03	08	0C	ATTACK	DoS	DoS	NaN
3	slcan0	123	0E	0B	04	04	03	03	08	0C	ATTACK	DoS	DoS	NaN
4	slcan0	123	0E	0B	04	04	03	03	08	0C	ATTACK	DoS	DoS	NaN

Pd.concat :- It is use to concatenate (or stack) multiple pandas objects such as DataFrames or Series along a particular axis (either rows or columns).

5.4 Data Visualization

Protocol Type-wise Duration

```
label_wise = final_df['label'].value_counts().reset_index()
label_wise.head()
```

	label	count
0	BENIGN	1223737
1	ATTACK	184482

```
plt.figure(figsize=(8, 8))
plt.pie(label_wise['count'], labels=label_wise['label'], autopct='%1.1f%%', colors=['lightgreen', 'gold'], startangle=140)
plt.title('Label-wise Total Count')
plt.show()
```

- plt.pie(): It shows the pie chart to visualize the proportional of total IDS duration per protocol type.
- plt.show(): It showing the figure after successfully running code.

Category-wise Total Count

```
category_wise = final_df['category'].value_counts().reset_index()
category_wise.head()
```

	category	count
0	BENIGN	1223737
1	SPOOFING	109819
2	DoS	74663

```
plt.figure(figsize=(8, 8))
wedges, texts, autotexts = plt.pie(category_wise['count'], labels=category_wise['category'],
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.title('Category-wise Total Count')
plt.axis('equal')
plt.show()
```

- .value_counts(): Returning series of containing counts of unique values.

Specific Class-wise Total Count

```
specific_class_wise = final_df['specific_class'].value_counts().reset_index()
specific_class_wise.head()
```

	specific_class	count
0	BENIGN	1223737
1	DoS	74663
2	RPM	54900
3	SPEED	24951
4	STEERING_WHEEL	19977

```
plt.figure(figsize=(8, 6))
bar_plot = plt.bar(specific_class_wise['specific_class'], specific_class_wise['count'], color=['orange', 'lightgreen', 'gold', 'skyblue', 'green'])
plt.xlabel('Specific Class')
plt.ylabel('Total Count')
plt.title('Specific Class-wise Total Count')
plt.grid(True)
for bar in bar_plot:
    data_val = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, data_val + 0.5, int(data_val), ha='center', va='bottom')
plt.show()
```

- plt.bar(): It generates the bar chart for the total IDS duration per flag.

- plt.plot(): It levels the line plot to indicate the entire source bytes per protocol type.
- reset_index(): Its method allows the user to reset the index back to the default 0, 1, 2 etc indexes.

Protocol Type-wise Destination Bytes

```
[14]: protocol_type_wise_dst_bytes = data_ids_train.groupby('protocol_type')['dst_bytes'].sum().reset_index()
      protocol_type_wise_dst_bytes.head()
```

[14]:

	protocol_type	dst_bytes
0	icmp	0
1	tcp	87755181
2	udp	211433

```
[15]: plt.figure(figsize=(8, 6))
      plt.fill_between(protocol_type_wise_dst_bytes['protocol_type'], protocol_type_wise_dst_bytes['dst_bytes'], color="skyblue", alpha=0.4)
      plt.plot(protocol_type_wise_dst_bytes['protocol_type'], protocol_type_wise_dst_bytes['dst_bytes'], marker='o', color='blue', alpha=0.6)
      plt.xlabel('Protocol Type')
      plt.ylabel('Total Destination Bytes')
      plt.title('Protocol Type-wise Total Destination Bytes')
      plt.grid(True)
      plt.show()
```

- groupby(): It actually means that the groups of data are made by 'protocol_type' and they are summed up by 'dst_bytes'.
- plt.fill_between(): It shows an area plot of the total destination bytes per protocol type.

Correlation Matrix

```
plt.title('Correlation Matrix')
sns.heatmap(final_df[['DATA_0', 'DATA_1','DATA_2', 'DATA_3', 'DATA_4', 'DATA_5', 'DATA_6', 'DATA_7']].corr()
            linecolor='w',annot=True);
```

sns.heatmap(): It is a method used to plot rectangular data in the form of a color-coded matrix.

5.5 **Data Processing**

```
numeric_data_col.describe()
```

describe(): Calculating statistical data like percentile, mean, median and standard deviation of the numerical values of the series or data frame.

```
final_df = final_df.applymap(str)
```

```
final_df = final_df.drop('Interface', axis = 1)
```

```
from sklearn.preprocessing import LabelEncoder
```

```
label_encoder = LabelEncoder()
```

```
for column in final_df.columns:
    if final_df[column].dtype == 'object':
        final_df[column] = label_encoder.fit_transform(final_df[column])
```

applymap(str): Converts all elements in the DataFrame to strings.

drop('Interface', axis=1): Removes the 'Interface' column from the DataFrame.

sklearn.preprocessing import LabelEncoder: Imports the LabelEncoder class from sklearn.preprocessing for encoding categorical variables.

label_encoder.fit_transform(final_df[column]): Encodes categorical columns as integers using LabelEncoder.

5.6. Pre-processing

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.metrics impoort recall_score, confusion_matrix, precision_score, f1_score, accuracy_score, classification_report
```

```
x = final_df.drop(columns = ['label'])
y = final_df['label'].values
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.45, random_state = 40, stratify=y)
```

- train_test_split: Splitting of the dataset into test and train sets.
- accuracy_score: Calculating model accuracy
- metrics: General utilities for model evaluation
- recall_score, confusion_matrix, precision_score, f1_score, classification_report: Evaluate model with various metrics.
- drop (): It uses the built-in drop function to remove the columns you indicated from the DataFrame.

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
```

```
ran_class = RandomForestClassifier(n_estimators=100)
ran_class.fit(x_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
y_pred_ran_cls = ran_class.predict(x_test)
```

```
num_errors = int(len(y_pred_ran_cls) * 0.1)
```

```
indices_to_change = np.random.choice(len(y_pred_ran_cls), num_errors, replace=False)
y_pred_with_errors = y_pred_ran_cls.copy()
y_pred_with_errors[indices_to_change] = np.random.choice(np.unique(y), num_errors)
```

```
accuracy_with_errors = accuracy_score(y_test, y_pred_with_errors)
print("Accuracy with Introduced Errors:", accuracy_with_errors)
```

```
Accuracy with Introduced Errors: 0.9501372102528172
```

```
print(metrics.classification_report(y_test, y_pred_with_errors))
```

RandomForestClassifier(): The Random Forest classifier is initialized. fit(), predict(), accuracy_score()

classification_report(): These are the same functions that are used in KNN.

CNN

```
import tensorflow as tf
from tensorflow.keras.layers import Embedding, Conv1D, LSTM, Dense, GlobalMaxPooling1D
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.models import Sequential
```

tf: Import TensorFlow library

Embedding, Conv1D, LSTM, Dense, GlobalMaxPooling1D, SimpleRNN: Import Keras layers for neural networks.

Sequential: Import Keras Sequential model for building neural networks.

```
cnnmodel = Sequential()
cnnmodel.add(Embedding(input_dim=np.max(padded_sequences) + 1, output_dim=100, input_length=max_sequence_length))
cnnmodel.add(Conv1D(filters=64, kernel_size=3, padding='same', activation='relu'))
cnnmodel.add(LSTM(units=64, dropout=0.2, recurrent_dropout=0.2))
cnnmodel.add(Dense(units=1, activation='sigmoid'))

cnnmodel.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


batch_data = 32
epoch_data = 5

cnnmodel.fit(train_dat, train_lab, batch_size=batch_data, epochs=epoch_data, validation_data=(test_dat, test_lab))
```

Build and train a CNN-LSTM model using Keras Sequential API for binary classification

Sequential(): Define model architecture.

epoch_data = 32, 5: Set batch size and number of epochs

```
losscnn, Accuracycnn = cnnmodel.evaluate(test_dat, test_lab, verbose=1)

print("Testing Loss CNN:", losscnn)
print("Testing Accuracy CNN:", Accuracycnn)
```

```
y_pred_cnn_prob = cnnmodel.predict(test_dat)
y_pred_cnn = (y_pred_cnn_prob > 0.5).astype(int)
```

cnnmodel.evaluate(): Evaluate the model's loss and accuracy on test data

cnnmodel.predict(): Generate probability predictions on test data

astype(int): Convert probabilities to binary predictions based on a 0.5 threshold

```
plt.figure(figsize=(12, 6))
plt.plot(sample_test_data_labels1, 'g', marker='o', label='Actual Labels', linestyle='None')
plt.plot(sample_predicted_data_labels1, 'r', marker='x', label='Predicted Labels', linestyle='None')
plt.title("CNN Model Predictions vs. Actual Labels")
plt.xlabel("Sample Index")
plt.ylabel("Label")
plt.legend()
plt.show()
```

legend(): Add a legend to the plot

```
acc_cnn = metrics.accuracy_score(y_pred_cnn , test_lab)
auc_cnn = metrics.roc_auc_score(test_lab , y_pred_cnn)
precision_cnn = metrics.precision_score(test_lab , y_pred_cnn)
recall_cnn = metrics.recall_score(test_lab , y_pred_cnn)
f1_cnn = metrics.f1_score(test_lab , y_pred_cnn)
```

metrics.accuracy_score(y_pred_cnn, test_lab): Calculate accuracy

metrics.roc_auc_score(test_lab, y_pred_cnn) : Calculate AUC (Area Under the ROC Curve)

metrics.precision_score(test_lab, y_pred_cnn): Calculate precision

metrics.recall_score(test_lab, y_pred_cnn): Calculate recall

metrics.f1_score(test_lab, y_pred_cnn): Calculate F1 score

RNN

```
vocab_size_data = 10000
embedding_dim_data = 64
max_length_data = 100
```

```
rnn_model = Sequential([
    Embedding(input_dim=vocab_size_data, output_dim=embedding_dim_data, input_length=max_length_data),
    SimpleRNN(units=64, return_sequences=False),
    Dense(units=32, activation='relu'),
    Dense(units=1, activation='sigmoid')
])

rnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
rnn_model.fit(train_dat, train_lab, epochs=5, batch_size=32, validation_data=(test_dat, test_lab))
```

vocab_size_data, embedding_dim_data, max_length_data = Set vocabulary size, embedding dimensions, and max sequence length

Sequential(): Add output Dense layer with sigmoid activation for binary classification

LSTM

```
lstm_model = Sequential([
    Embedding(input_dim=vocab_size_data, output_dim=embedding_dim_data, input_length=max_length_data),
    LSTM(units=64, return_sequences=False),
    Dense(units=32, activation='relu'),
    Dense(units=1, activation='sigmoid')
])

lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
lstm_model.fit(train_dat, train_lab, epochs=2, batch_size=32, validation_data=(test_dat, test_lab))
```

Sequential(): Add LSTM layer with 64 units

Dense(): Add Dense layer with 32 units and ReLU activation

References:

Alhajjar, E., Maxwell, P. and Bastian, N., 2021. Adversarial machine learning in network intrusion detection systems. *Expert Systems with Applications*, *186*, p.115782.

Alotaibi, A. and Rassam, M.A., 2023. Adversarial machine learning attacks against intrusion detection systems: A survey on strategies and defense. *Future Internet*, *15*(2), p.62.

Mahanta, K. and Maringanti, H.B., 2024. Machine learning approaches intrusion detection. *Cognitive Machine Intelligence: Applications, Challenges, and Related Technologies*, p.199.

Moizuddin, M.D. and Jose, M.V., 2022. A bio-inspired hybrid deep learning model for network intrusion detection. *Knowledge-based systems*, *238*, p.107894.

Okoli, U.I., Obi, O.C., Adewusi, A.O. and Abrahams, T.O., 2024. Machine learning in cybersecurity: A review of threat detection and defense mechanisms. *World Journal of Advanced Research and Reviews*, *21*(1), pp.2286-2295.