

Configuration Manual

MSc Research Project
MSc Cybersecurity

Elsamma Joshy
Student ID: 23171847

School of Computing
National College of Ireland

Supervisor: Arghir Nicolae Moldovan

**National College of Ireland
MSc Project Submission Sheet
School of Computing**



Student

Name: Elsamma Joshy

Student ID: 23171847

Programme: MSc Cybersecurity

Year: 2024-25

Module: MSc Research Project

Lecturer: Arghir Nicolae Moldovan

Submission

Due Date: 18/12/2024

Project Title: Comparing Zero Trust Model with traditional network and Machine Learning enhancement in OpenZiti

Word Count: 1422 **Page Count:** 11

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: Elsamma Joshy

Date: 18/12/2024

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

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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

Configuration Manual

Elsamma Joshy
Student ID:23171847

1 Introduction

This configuration manual provides a guide to setting up the environment for implementing and evaluating Zero Trust Network Access (ZTNA) and Virtual Private Network (VPN) models using machine learning. It includes instructions for installing the required Python modules necessary for data collection and performance analysis. The manual focuses on key network metrics such as throughput, jitter, and latency to evaluate the efficiency and reliability of both networks. The goal is to compare ZTNA and VPN to determine which provides superior performance in various network conditions and enhance the network using machine learning.

Hardware Requirement

Operating System: Windows 10 or above

RAM: 16GB

Processor: Intel Core i3 9th gen or above or AMD Ryzen 3 3rd gen or above

Storage: 256GB SSD (Solid State Drive)

System Type: 64-bit Operating System

Software Requirement

Google Colab

Python 3.6 version

Python Libraries like seaborn, keras, pandas, scikit learn and matplotlib

Python programming is used in this research project to implement Machine Learning model to evaluate the datasets. Google Colab is used as a cloud-based platform for running and executing the python code. In this project, many python libraries were used to evaluate and visualize the models.

2 Python Libraries

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

a) Scikit learn:

Scikit-learn is a versatile Python library offering efficient tools for machine learning tasks such as classification, regression, clustering, dimensionality reduction, preprocessing, and model evaluation, making it suitable for building and validating predictive models.

b) Matplotlib:

Matplotlib is a powerful library for creating detailed visualizations, including line plots, bar charts, histograms, scatter plots, and more, allowing users to customize and export their plots in various formats effectively.

c) XGBOOST:

XGBoost is a cutting-edge gradient-boosting library, designed for building fast, scalable, and accurate machine learning models, with features like regularization, missing value handling, and GPU support for improved performance and model precision(Farook et al., 2022).

d) Pandas:

Pandas is a widely used Python library for data manipulation, cleaning, aggregation, and analysis, offering tools to work seamlessly with structured datasets like tables, spreadsheets, and time-series, enabling efficient and intuitive data workflows.

3 Dataset Used

The UNSW-NB15 dataset contains realistic network traffic generated using the IXIA PerfectStorm tool, including normal and malicious data. It features 49 attributes and nine attack types, offering a diverse and comprehensive representation of real-world network traffic for training and testing intrusion detection systems. It contains network traffic data, which includes both benign (normal) traffic and malicious traffic. The data is structured in the form of network flow records, each representing a connection or session between two devices on a network. The dataset provides information about various attributes of these network connections, categorized as follows:

1. Benign (Normal) Traffic :

This includes data representing typical, non-malicious network activity. It consists of regular browsing, file transfers, email exchanges, and other standard network services under typical conditions. These records provide a baseline for distinguishing between normal and malicious traffic.

2. Malicious (Attack) Traffic:

This data represents different types of malicious activities, each categorized into specific attack classes. The malicious traffic is generated by a wide range of attack types to simulate realworld cyber threats. These attacks are categorized into the following types:

A) **Denial of Service (DoS)**: Attacks designed to overwhelm a system, such as flooding traffic or resource exhaustion.

B) **Intrusion Attacks**: Attacks aimed at unauthorized access to systems, including exploits and brute-force attempts.

C) **Malware**: Malicious software designed to disrupt or harm systems, including viruses, worms, and Trojans.

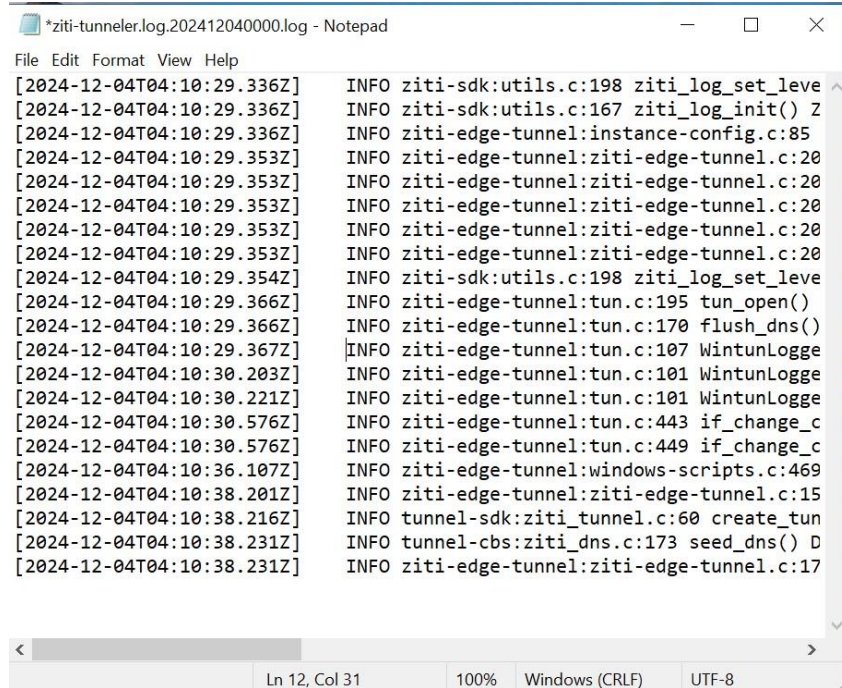
D) **Botnets**: Traffic generated by networks of compromised devices used for malicious purposes, often in coordinated attacks.

E) **Web Attacks**: Attacks targeting web servers, including SQL injection, cross-site scripting (XSS), and other vulnerabilities.

F) **Zero-Day Attacks**: Exploits that take advantage of previously unknown vulnerabilities,

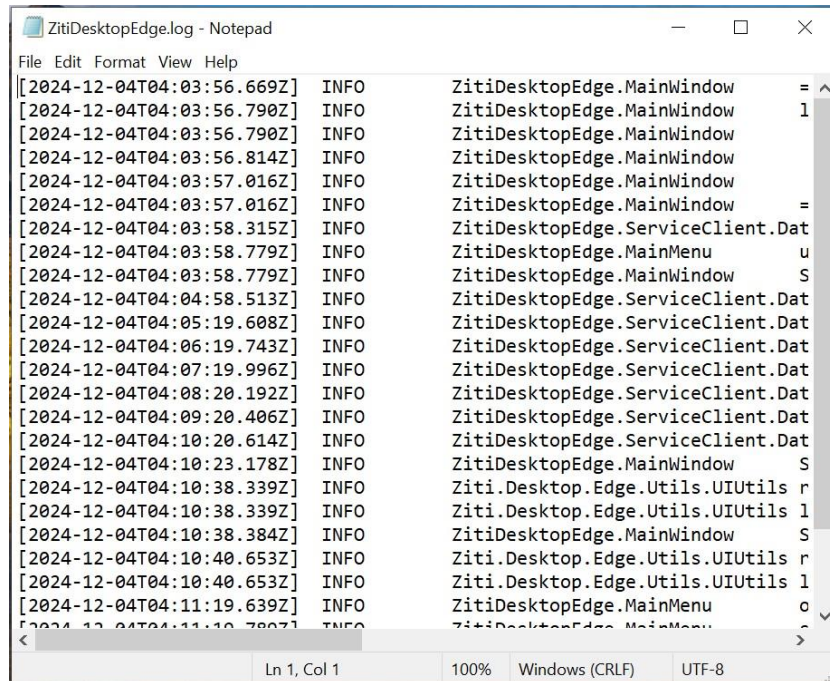
often used in sophisticated attacks.

Snapshots of the Dataset



```
*ziti-tunneler.log.202412040000.log - Notepad
File Edit Format View Help
[2024-12-04T04:10:29.336Z] INFO ziti-sdk:utils.c:198 ziti_log_set_leve
[2024-12-04T04:10:29.336Z] INFO ziti-sdk:utils.c:167 ziti_log_init() Z
[2024-12-04T04:10:29.336Z] INFO ziti-edge-tunnel:instance-config.c:85
[2024-12-04T04:10:29.353Z] INFO ziti-edge-tunnel:ziti-edge-tunnel.c:20
[2024-12-04T04:10:29.353Z] INFO ziti-edge-tunnel:ziti-edge-tunnel.c:20
[2024-12-04T04:10:29.353Z] INFO ziti-edge-tunnel:ziti-edge-tunnel.c:20
[2024-12-04T04:10:29.353Z] INFO ziti-edge-tunnel:ziti-edge-tunnel.c:20
[2024-12-04T04:10:29.353Z] INFO ziti-edge-tunnel:ziti-edge-tunnel.c:20
[2024-12-04T04:10:29.354Z] INFO ziti-sdk:utils.c:198 ziti_log_set_leve
[2024-12-04T04:10:29.366Z] INFO ziti-edge-tunnel:tun.c:195 tun_open()
[2024-12-04T04:10:29.366Z] INFO ziti-edge-tunnel:tun.c:170 flush_dns()
[2024-12-04T04:10:29.367Z] INFO ziti-edge-tunnel:tun.c:107 WintunLogge
[2024-12-04T04:10:30.203Z] INFO ziti-edge-tunnel:tun.c:101 WintunLogge
[2024-12-04T04:10:30.221Z] INFO ziti-edge-tunnel:tun.c:101 WintunLogge
[2024-12-04T04:10:30.356Z] INFO ziti-edge-tunnel:tun.c:443 if_change_c
[2024-12-04T04:10:30.576Z] INFO ziti-edge-tunnel:tun.c:449 if_change_c
[2024-12-04T04:10:36.107Z] INFO ziti-edge-tunnel:windows-scripts.c:469
[2024-12-04T04:10:38.201Z] INFO ziti-edge-tunnel:ziti-edge-tunnel.c:15
[2024-12-04T04:10:38.216Z] INFO tunnel-sdk:ziti_tunnel.c:60 create_tun
[2024-12-04T04:10:38.231Z] INFO tunnel-cbs:ziti_dns.c:173 seed_dns() D
[2024-12-04T04:10:38.231Z] INFO ziti-edge-tunnel:ziti-edge-tunnel.c:17
```

Fig 1: OpenZITI dataset



```
ZitiDesktopEdge.log - Notepad
File Edit Format View Help
[2024-12-04T04:03:56.669Z] INFO ZitiDesktopEdge.MainWindow =
[2024-12-04T04:03:56.790Z] INFO ZitiDesktopEdge.MainWindow 1
[2024-12-04T04:03:56.790Z] INFO ZitiDesktopEdge.MainWindow
[2024-12-04T04:03:56.814Z] INFO ZitiDesktopEdge.MainWindow
[2024-12-04T04:03:57.016Z] INFO ZitiDesktopEdge.MainWindow
[2024-12-04T04:03:57.016Z] INFO ZitiDesktopEdge.MainWindow =
[2024-12-04T04:03:58.315Z] INFO ZitiDesktopEdge.ServiceClient.Dat
[2024-12-04T04:03:58.779Z] INFO ZitiDesktopEdge.MainMenu u
[2024-12-04T04:03:58.779Z] INFO ZitiDesktopEdge.MainWindow S
[2024-12-04T04:04:58.513Z] INFO ZitiDesktopEdge.ServiceClient.Dat
[2024-12-04T04:05:19.608Z] INFO ZitiDesktopEdge.ServiceClient.Dat
[2024-12-04T04:06:19.743Z] INFO ZitiDesktopEdge.ServiceClient.Dat
[2024-12-04T04:07:19.996Z] INFO ZitiDesktopEdge.ServiceClient.Dat
[2024-12-04T04:08:20.192Z] INFO ZitiDesktopEdge.ServiceClient.Dat
[2024-12-04T04:09:20.406Z] INFO ZitiDesktopEdge.ServiceClient.Dat
[2024-12-04T04:10:20.614Z] INFO ZitiDesktopEdge.ServiceClient.Dat
[2024-12-04T04:10:23.178Z] INFO ZitiDesktopEdge.MainWindow S
[2024-12-04T04:10:38.339Z] INFO Ziti.Desktop.Edge.Utils.UIUtils r
[2024-12-04T04:10:38.339Z] INFO Ziti.Desktop.Edge.Utils.UIUtils l
[2024-12-04T04:10:38.384Z] INFO ZitiDesktopEdge.MainWindow S
[2024-12-04T04:10:40.653Z] INFO Ziti.Desktop.Edge.Utils.UIUtils r
[2024-12-04T04:10:40.653Z] INFO Ziti.Desktop.Edge.Utils.UIUtils l
[2024-12-04T04:11:19.639Z] INFO ZitiDesktopEdge.MainMenu o
[2024-12-04T04:11:19.780Z] INFO ZitiDesktopEdge.MainMenu
```

Fig 2: OpenVPN dataset

These figures are the sample data from the dataset which is used for evaluating which configuration is the best, which is either OpenZITI or OpenVPN. The dataset consist of date, timestamp, protocol used, type of configuration, and so on.

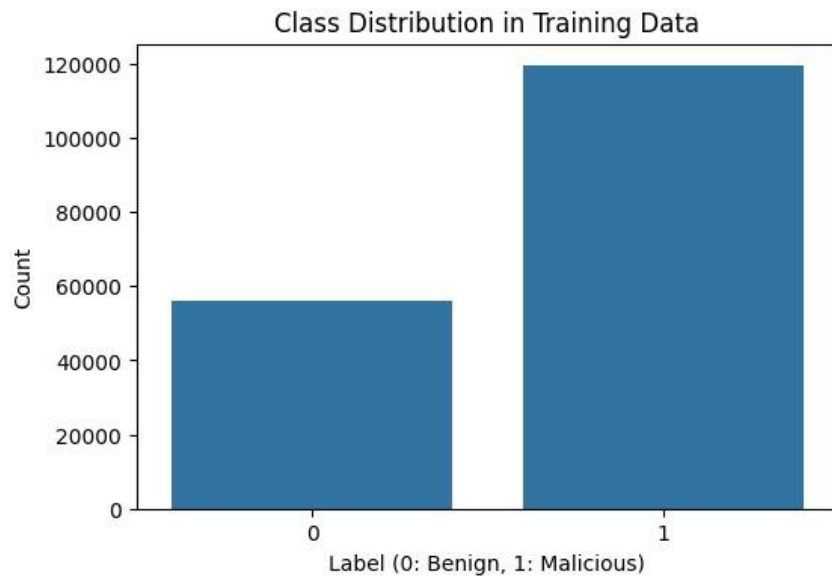


Fig 3: Classification of Data

This part of the code is focused on visualizing the distribution of **benign** (normal) and **malicious** (attack) network traffic within the training dataset. It uses the seaborn and matplotlib libraries to create a bar chart (count plot) showing how many instances of each class are present (Gunuganti, 2023).

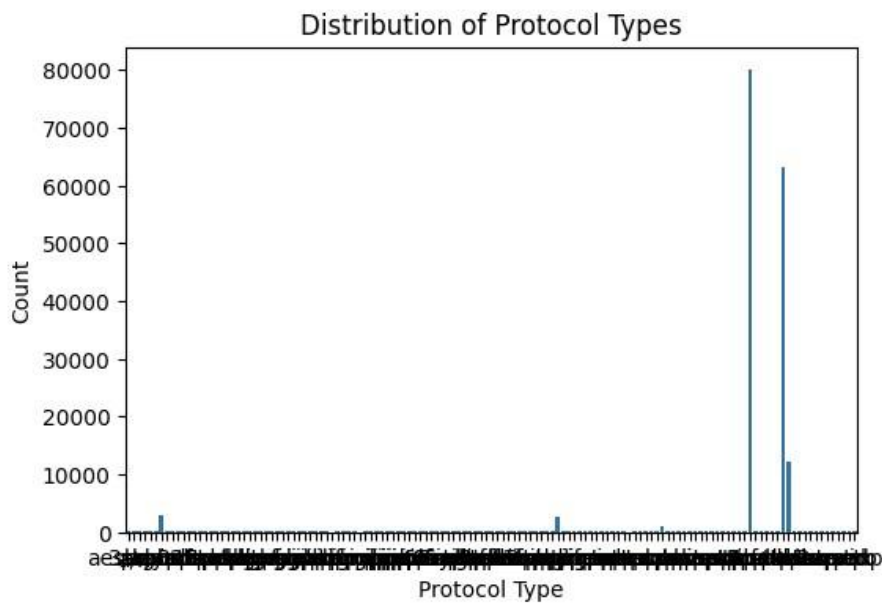


Fig 4: Types of Protocols found in the dataset

This code snippet is creating a bar plot to visualize the distribution of different protocol types within the training dataset (train_data). It's essentially showing how many times each protocol appears in the network traffic data.

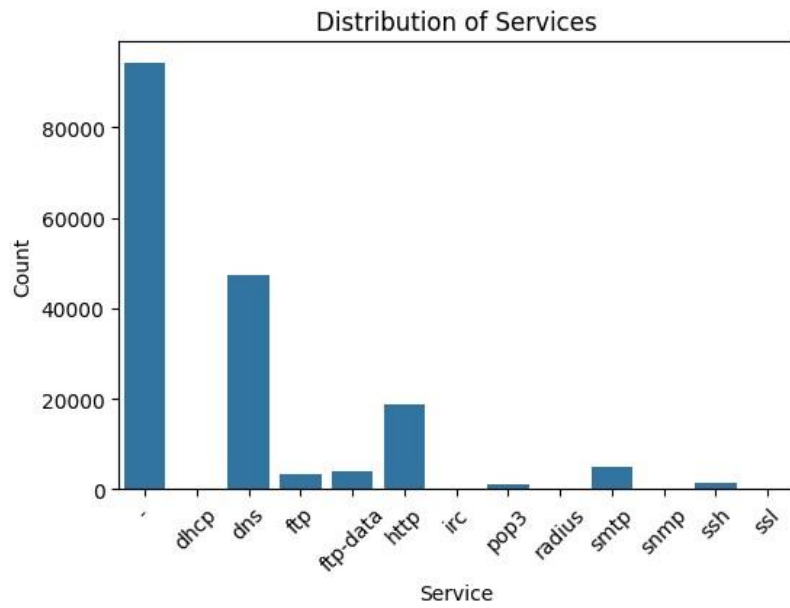


Fig 5: Types of Services found in Dataset

This code snippet is designed to visualize the distribution of the service feature within the train_data dataset. It essentially shows how many times each distinct service appears in the dataset

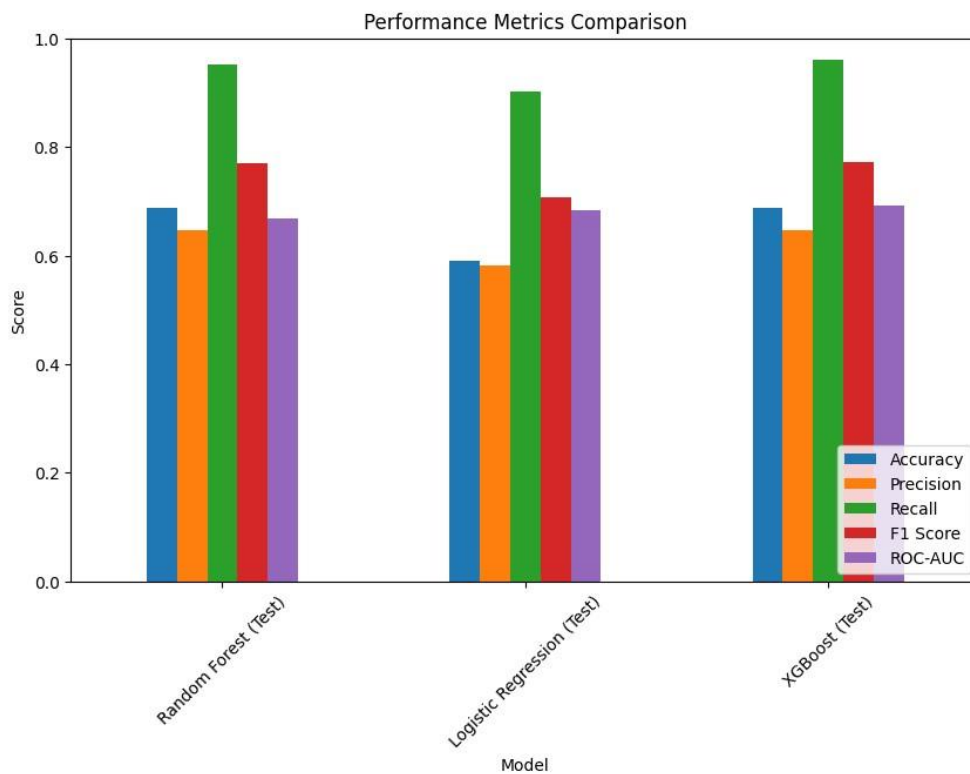


Fig 6: Performance Metric Comparison between Models

The image presents a comparison of performance metrics for three different machine learning models: Random Forest, Logistic Regression, and XGBoost. Each model is evaluated across five metrics: Accuracy, Precision, Recall, F1-Score, and ROC-AUC.

Training and Testing Summary of the Machine Learning Models

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Step 1: Load the Data
train_data = pd.read_parquet('UNSW_NB15_training-set.parquet')
test_data = pd.read_parquet('UNSW_NB15_testing-set.parquet')

# Step 2: Data Exploration (optional - you already have this)

# Step 3: Preprocessing
X_train = train_data.drop('label', axis=1)
y_train = train_data['label']
X_test = test_data.drop('label', axis=1)
y_test = test_data['label']

# Encode categorical features
categorical_cols = X_train.select_dtypes(include=['object']).columns
le = LabelEncoder()
for col in categorical_cols:
    X_train[col] = le.fit_transform(X_train[col])
    X_test[col] = le.transform(X_test[col])

# Normalize numeric features
numeric_cols = X_train.select_dtypes(include=['int64', 'float64']).columns
scaler = StandardScaler()
X_train[numeric_cols] = scaler.fit_transform(X_train[numeric_cols])
X_test[numeric_cols] = scaler.transform(X_test[numeric_cols])
```

Fig 7: Python code for importing libraries and storing the dataset into variables

The code loads the training and testing datasets from Parquet files, separates features and labels into `X_train`, `y_train`, `X_test`, and `y_test`. It encodes categorical features using `LabelEncoder` and normalizes numeric features with `StandardScaler` to ensure proper scaling, improving compatibility with machine learning models and their performance.

```

# Normalize numeric features
numeric_cols = X_train.select_dtypes(include=['int64', 'float64']).columns
scaler = StandardScaler()
X_train[numeric_cols] = scaler.fit_transform(X_train[numeric_cols])
X_test[numeric_cols] = scaler.transform(X_test[numeric_cols])

# Step 4: Train the Models
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train[numeric_cols], y_train)

lr_clf = LogisticRegression(random_state=42)
lr_clf.fit(X_train[numeric_cols], y_train)

xgb_clf = XGBClassifier(n_estimators=100, random_state=42)
xgb_clf.fit(X_train[numeric_cols], y_train)

```

Fig 8: Training the Dataset using Models

The code normalizes numeric features by scaling them with StandardScaler, ensuring they have a mean of 0 and standard deviation of 1. Then, it trains three machine learning models: **Random Forest**, **Logistic Regression**, and **XGBoost**, using the normalized data and their corresponding labels to predict outcomes effectively(Haddon, 2021).

```

# Random Forest - Test Set
y_pred_rf = rf_clf.predict(X_test[numeric_cols])
print("\nRandom Forest - Test Set:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf)}")
print(f"Precision: {precision_score(y_test, y_pred_rf)}")
print(f"Recall: {recall_score(y_test, y_pred_rf)}")
print(f"F1 Score: {f1_score(y_test, y_pred_rf)}")
print(f"ROC-AUC: {roc_auc_score(y_test, rf_clf.predict_proba(X_test[numeric_cols]))[:, 1])}")
print("\nClassification Report for Random Forest:")
print(classification_report(y_test, y_pred_rf))

# Logistic Regression - Test Set
y_pred_lr = lr_clf.predict(X_test[numeric_cols])
print("\nLogistic Regression - Test Set:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_lr)}")
print(f"Precision: {precision_score(y_test, y_pred_lr)}")
print(f"Recall: {recall_score(y_test, y_pred_lr)}")
print(f"F1 Score: {f1_score(y_test, y_pred_lr)}")
print(f"ROC-AUC: {roc_auc_score(y_test, lr_clf.predict_proba(X_test[numeric_cols]))[:, 1])}")
print("\nClassification Report for Logistic Regression:")
print(classification_report(y_test, y_pred_lr))

# XGBoost - Test Set
y_pred_xgb = xgb_clf.predict(X_test[numeric_cols])
print("\nXGBoost - Test Set:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_xgb)}")
print(f"Precision: {precision_score(y_test, y_pred_xgb)}")
print(f"Recall: {recall_score(y_test, y_pred_xgb)}")
print(f"F1 Score: {f1_score(y_test, y_pred_xgb)}")
print(f"ROC-AUC: {roc_auc_score(y_test, xgb_clf.predict_proba(X_test[numeric_cols]))[:, 1])}")
print("\nClassification Report for XGBoost:")
print(classification_report(y_test, y_pred_xgb))

```

Fig 9: Python code to display the evaluation of each model

The code normalizes numeric features by scaling them with StandardScaler, ensuring they have a mean of 0 and standard deviation of 1. Then, it trains three machine learning models: **Random Forest**, **Logistic Regression**, and **XGBoost**, using the normalized data and their corresponding labels to predict outcomes effectively.

```

Random Forest - Test Set:
Accuracy: 0.6869625419035126
Precision: 0.6467489983643703
Recall: 0.9507632577428748
F1 Score: 0.769828440784832
ROC-AUC: 0.6672391685009814

Classification Report for Random Forest:

```

	precision	recall	f1-score	support
0	0.86	0.36	0.51	37000
1	0.65	0.95	0.77	45332
accuracy			0.69	82332
macro avg	0.75	0.66	0.64	82332
weighted avg	0.74	0.69	0.65	82332

Fig 10: Random Forest Classification Report

The output shows the performance metrics for a Random Forest model evaluated on the test set. The accuracy is 68.7%, with precision of 64.7%, recall of 95.1%, and F1 score of 77%. The classification report further breaks down precision, recall, and F1 score for both classes, indicating a strong performance in detecting class 1 (malicious) traffic.

```

Logistic Regression - Test Set:
Accuracy: 0.5902079385900987
Precision: 0.5826595365418895
Recall: 0.9013279802347128
F1 Score: 0.7077786535246888
ROC-AUC: 0.6840734425416328

Classification Report for Logistic Regression:

```

	precision	recall	f1-score	support
0	0.63	0.21	0.31	37000
1	0.58	0.90	0.71	45332
accuracy			0.59	82332
macro avg	0.61	0.56	0.51	82332
weighted avg	0.61	0.59	0.53	82332

Fig 11: Logistic Regression Classification Report

The Logistic Regression model's overall accuracy is 59%, meaning it correctly predicts whether a patient has a disease or not in 59% of cases. It's better at identifying patients with the disease (recall of 90%) but less accurate at correctly identifying healthy patients (precision of 58%). The F1-score of 71% balances precision and recall, while the ROC-AUC of 68% measures how well the model distinguishes between classes.

```

XGBoost - Test Set:
Accuracy: 0.6885900986250789
Precision: 0.6462011314199172
Recall: 0.9600502955969293
F1 Score: 0.7724638581482237
ROC-AUC: 0.6928534174892267

Classification Report for XGBoost:

```

	precision	recall	f1-score	support
0	0.88	0.36	0.51	37000
1	0.65	0.96	0.77	45332
accuracy			0.69	82332
macro avg	0.76	0.66	0.64	82332
weighted avg	0.75	0.69	0.65	82332

Fig 12: XGBOOST Classification Report

The XGBoost model's overall accuracy is 69%, meaning it correctly predicts whether a patient has a disease or not in 69% of cases. It's better at identifying patients with the disease (recall of 96%) but less accurate at correctly identifying healthy patients (precision of 65%). The F1-score of 77% balances precision and recall, while the ROC-AUC of 69% measures how well the model distinguishes between classes.

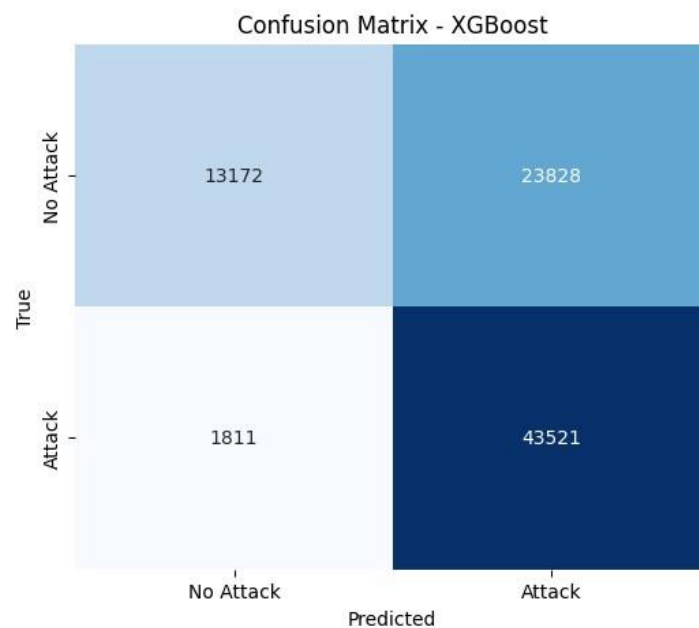


Fig 13: XGBOOST Confusion Matrix

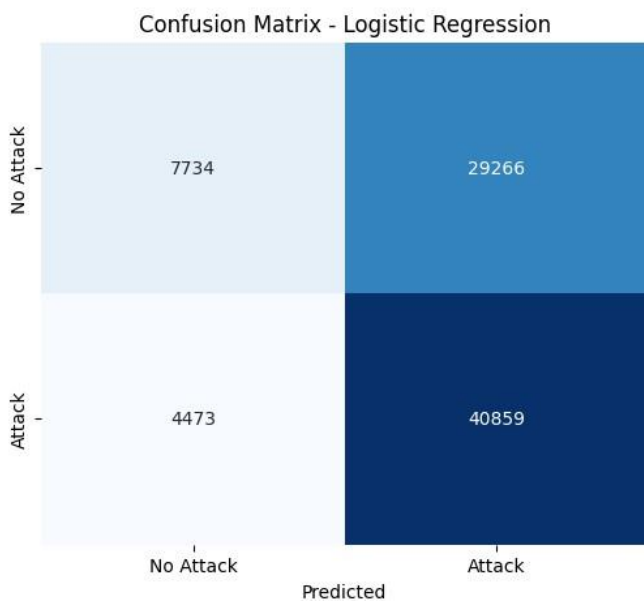


Fig 14: Logistic Regression Confusion Matrix

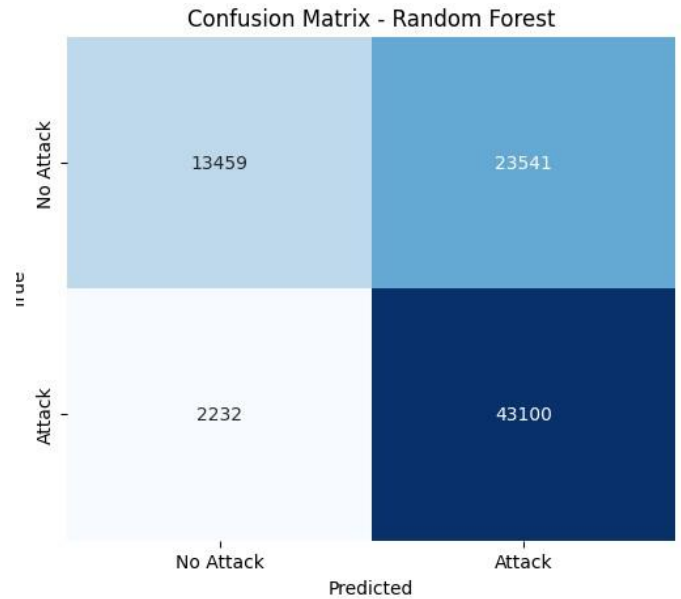


Fig 15: Random Forest Confusion Matrix

The XGBoost model has the best overall performance with an accuracy of 69%, outperforming both Logistic Regression (59%) and the other XGBoost model (68%). All models struggle with false negatives (incorrectly predicting benign traffic as malicious traffic), but XGBoost models are better at identifying malicious traffic (high recall).

References

- Farook, M., Macklin, T., Ahmadinia, A., & Tyagi, S. (2022). *THE PROJECT HAS BEEN ACCEPT (vlonqmmqd Fqroo/< Zero Trust Evolution and Transforming Enterprise Security Sanjay Kak THE PROJECT HAS BEEN ACCEPTED BY THE PROJECT COMMITTEE IN SCIENCE IN CYBERSECURITY.*
- Gunuganti, A. (2023). Citation: Gunuganti A. Identity Based-Zero Trust. *J Artif Intell Mach Learn & Data Sci*, 2023(2), 492–497. <https://doi.org/10.51219/JAIMLD/anvesh-gunuganti/133>
- Haddon, D. A. E. (2021). Zero Trust networks, the concepts, the strategies, and the reality. *Strategy, Leadership, and AI in the Cyber Ecosystem: The Role of Digital Societies in Information Governance and Decision Making*, 195–216. <https://doi.org/10.1016/B978-0-12-821442-8.00001-X>