Major Project-Report
On
Web-Based Cyber Security Threat Detection Using Machine Learning

Submitted By
Vignesh Enagandula
(R201065)

Under the Guidance of
Satyanandaram N
Lecturer
Computer Science and Engineering

Rajiv Gandhi University ofKnowledge and Technologies
RGUKT RK Valley,Kadapa, 516330
Department of ComputerScience and Engineering

2025-2026

Rajiv Gandhi University of Knowledge Technologies
RK Valley, Kadapa (Dist.), Andhra Pradesh - 516330

CERTIFICATE

This is to certify that the project report entitled "**Web-Based Cyber Security Threat Detection using Machine Learning**" being submitted by **Vignesh Enagandula (R201065)** under my guidance and supervision, and is submitted to DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING in partial fulfilment of requirements for the award of Bachelor of Technology in Computer Science and Engineering during the academic year 2025-2026 and it has been found worthy of acceptance according to the requirements of the University.

Signature of Internal Guide
Mr. Satyanandaram N
MSIT (IIITHyderabad)
Lecturer
ComputerScienceandEngineering
RGUKT, RK VALLEY

Signature of HOD
Dr. Ch. Ratna Kumari,
M.Tech. (University of Hyderabad)
Ph.D. (JNTU Hyderabad)
Head of the Department
Computer Science and Engineering
RGUKT, RK Valley

Signature of External Examiner

DECLARATION

I hereby declare that this project work entitled **"Web-based Cyber Security Threat Detection using Machine Learning"**, submitted to the **Department of Computer Science and Engineering**, is a genuine work carried out by me under the guidance of **Mr. Satyanandaram N**. This project is submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology (B.Tech)**.

I further declare that this work has not been submitted elsewhere for the award of anyother degree or diploma other than as specified above.

Vignesh Enagandula
R201065

ACKNOWLEDGEMENT

I would like to express my sense of gratitude and respect to all those people behind the screen who guided, inspired, and helped me to crown all my efforts with success. I wish to express my gratitude to my project guide **Mr. Satyanandaram N** for his valuable guidance at all stages of the study, advice, constructive suggestions, supportive attitude, and continuous encouragement, without which this project would not have been possible.

I would also like to extend my deepest gratitude and reverence to the **Head of the Department of Computer Science and Engineering, Dr. CH. Ratna kumari (Asst. Professor, ME, PhD)** and the **Director of RGUKT, RK Valley, Prof. A V S S Kumara Swami Guptha** for their constant support and encouragement. Last but not least, I express my heartfelt gratitude to my parents for being a constant source of encouragement and inspiration, helping me keep my morals high.

With sincere Regards

Vignesh Enagandula

R201065

Table of Contents

Chapter 1

1 Abstract

This project focuses on the development and evaluation of a high-performance Machine Learning (ML) model for the offline analysis and classification of already captured network flow data, specifically targeting the identification of Botnet attacks. Which pose a significant threat to modern cyber infrastructure. The objective was to design a robust model capable of accurately classifying historical network traffic records into Benign (normal) and Bot (malicious) categories.

The methodology employed a **Random Forest Classifier,** an ensemble learning model selected for its superior performance and resilience when handling complex, high-dimensional datasets. The model was trained **using flow features** extracted from the **Friday-02-03-2018_TrafficForML_CICFlowMeter.csv** dataset, which represents a collection of captured network packets. Rigorous preprocessing, including handling data anomalies and feature engineering, was performed to prepare the historical data for training.

Performance evaluation on an unseen test set confirmed the model's effectiveness, yielding an Overall Accuracy exceeding 99.9% and a highly reliable Matthews Correlation Coefficient (MCC) of approximately 0.997. This robust performance confirms the model's strong capability for forensic analysis and pattern recognition within captured network data, making it valuable for retrospective threat assessment and informing future anomaly detection strategies.

- **Backend Flask Code**

This abstract details the Flask-based backend architecture engineered to support the offline classification and analysis of captured network traffic data using a trained Machine Learning (ML) model. The system is designed for robustness, security, and high performance, focusing on managing data integrity and user authorization throughout the classification workflow. The architecture implements a Role-Based Access Control (RBAC) layer to securely restrict access, allowing standard users to only execute predictions while reserving system analysis and performance review for administrative roles

Core logic involves caching the trained Random Forest model in memory upon server startup to ensure low-latency prediction times. The backend securely handles uploaded CSV files, applies standardized preprocessing logic identical to the training phase, and utilizes the cached model to perform classifications. Furthermore, a dedicated analysis endpoint provides administrators with rich, structured JSON output of key metrics (e.g., MCC, Confusion Matrix) from classified data, establishing a reliable, modular, and scalable foundation for cybersecurity threat assessment.

Chapter 2

2 Introduction:

2.1 Problem Statement

The increasing sophistication and frequency of cyberattacks, particularly Botnet activity, present a persistent challenge to organizational network security. Traditional security solutions often rely on signature-based detection, which is ineffective against zero-day attacks and polymorphic threats. This leaves networks vulnerable to lateral movement and data exfiltration after an initial compromise.

The core challenge addressed by this project is the lack of an accurate, robust, and scalable automated system for retrospective (offline) analysis of captured network flow data that can reliably distinguish between legitimate Benign traffic and malicious Botnet traffic. Current manual forensic review of large network logs is time-consuming, prone to human error, and delays threat mitigation.

2.2 Project Objective

The primary goal of this project is to design, implement, and validate a supervised Machine Learning system, utilizing a Random Forest Classifier, to analyze historical network flow features (sourced from the CICFlowMeter dataset). This system must demonstrate high precision and recall in identifying Botnet attacks to provide actionable intelligence for security teams, thereby significantly reducing the mean time to detect and understand threat vectors present in captured data. The solution must be integrated into a robust Flask backend capable of handling data processing, secure access control (RBAC), and delivering clear, verifiable performance metrics.

Chapter 3

3 Technologies Used:

The core detection logic relies on Python libraries optimized for large-scale data processing and machine learning.

Data Science & Machine Learning Stack

- **Python**: The foundational programming language, chosen for its vast ecosystem of scientific libraries and straightforward syntax.
- **Pandas**: Used for all data manipulation and preprocessing tasks. It provides high-performance, easy-to-use data structures (like the DataFrame) essential for loading the large network flow CSV file, handling missing values (NaN), and cleaning inconsistent column names.
- **NumPy**: Provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions. It serves as the numerical backbone for Pandas and Scikit-learn.
- **Scikit-learn (sklearn)**: The primary machine learning framework used for the entire classification workflow:
  - **Model Selection**: Specifically, the RandomForestClassifier is used for its high accuracy, efficiency on high-dimensional data, and feature importance capabilities.
  - **Preprocessing**: Utilized for Label Encoding (converting 'Benign'/'Bot' to 0/1) and Data Splitting (train_test_split).
  - **Evaluation**: Provides critical metrics functions, including accuracy_score, confusion_matrix, classification_report, matthews corr coef (MCC), and balanced_accuracy_score.
- **Matplotlib & Seaborn**: Libraries used for data visualization. They are integral to extracting actionable visuals from the results, such as the Confusion Matrix heatmap and charts showing the distribution of predicted labels.

- **Pickle:** The module used for model persistence. It serializes the trained Random Forest model object into a binary file (.pkl), allowing the Flask backend to load the model rapidly without needing to retrain it for every prediction request.

Backend Development & Deployment Stack

The user-facing system and API endpoints are built using the Flask framework, focusing on secure, modular deployment

- **Flask:** A lightweight, micro-web framework for Python. It provides the minimal necessary tools to build API endpoints (/predict, /analyze_performance) for handling HTTP requests, managing user sessions, and serving responses..
- **Werkzeug**: A set of utilities that Flask uses to handle core tasks like secure filename generation (secure_filename) and managing file uploads. This is crucial for protecting the server from malicious file injection.
- **File I/O (os, io)**: Standard Python modules used to handle file system operations (checking directories, managing file paths) and managing in-memory file streams (io.BytesIO or io.StringIO) for efficient handling of CSV downloads and image generation without creating unnecessary temporary files on disk.
- **Role-Based Access Control (RBAC) Logic:** Implemented using Flask decorators and header verification (e.g., checking for an Auth-Role header). This enforces a basic security policy, separating standard users (allowed to predict) from administrators (allowed to access sensitive performance analysis tools).
- **JSON (JavaScript Object Notation):** The primary data exchange format for the API. Metrics and complex analysis results (like the Confusion Matrix data) are structured in JSON for easy consumption by front-end applications or other system services.

Chapter 4

Random Forest Model Development and Training:

The Random Forest Classifier was selected as the primary algorithm due to its inherent strengths in the cybersecurity domain, particularly when using high-dimensional, tabular network flow data generated by CICFlowMeter.

- Non-Linearity Handling: Random Forest (an ensemble of Decision Trees) excels at capturing complex, non-linear relationships between the features (like packet size, flow duration, and protocol) and the threat outcome, which is crucial as malicious traffic often shows irregular, non-linear patterns.
- Built-in Feature Importance: It naturally calculates feature importance (based on Gini impurity reduction), allowing for a clear understanding of which network features are most predictive of an attack. This interpretability is vital for security analysts.
- Robustness to Overfitting: By averaging the predictions of multiple trees trained on different data subsets (bootstrapping), Random Forest significantly reduces the risk of overfitting compared to a single Decision Tree.
- Efficiency: It handles a large number of features efficiently and generally requires less hyperparameter tuning than deep neural networks or complex gradient boosting methods.

4.2 Data Preparation for Model Training:

This involves preparing the CICFlowMeter-derived data specifically for the Random Forest model.

- Step 1: Feature Transformation & Encoding:
  - Handling Categorical Features: Any remaining categorical features (like Protocol or flag fields) were converted into numerical format using One-Hot Encoding.
  - Feature Scaling (Less Critical, but Done): While Random Forest is generally insensitive to feature scaling (since its decisions are based on threshold splits), standard normalization techniques were applied to ensure consistency across the entire pipeline.
- Step 2: Training, Validation, and Test Split:
  - The dataset was partitioned into Training (approx 70%), Validation (approx 15%), and Test (approx 15%) sets. This ensures that the model is evaluated on data it has never seen during training or optimization.
- Step 3: Addressing Class Imbalance:
  - Due to the likely severe imbalance in CICFlowMeter data (many benign flows, few malicious attacks), a technique such as SMOTE (Synthetic Minority Over-sampling Technique) or similar synthetic data generation was applied only to the Training Set. Alternatively, the Random Forest's built-in class weighting feature was utilized.

4.3 Preprocessing the CICFlowMeter Dataset

The objective of preprocessing is to convert the raw .csv or log data generated by CICFlowMeter (which contains features like Flow Duration, Total Fwd Packets, min_seg_size_forward, etc.) into a clean, numerical matrix suitable for the Random Forest Classifier.

1. Initial Data Inspection and Loading

- **Data Loading:** Load the CICFlowMeter output files (often multiple, large .csv files) into a suitable data structure (like a Pandas DataFrame in Python).
- **Feature Review:** Review the full list of features (often 80+ columns). Identify the Target Variable (the label indicating whether the flow is benign or a specific type of attack).
- **Handling Column Names**: CICFlowMeter can sometimes produce column names with spaces or inconsistent capitalization. Standardize all column names (e.g., convert to snake_case) for easier code management.

2. Handling Missing and Infinite Values

- **Identifying Nulls/NaNs:** Check for standard missing values (NaNs). For most CICFlowMeter features, filling small numbers of missing values with the mean or median of the column is acceptable, but only after careful consideration, as network traffic data can be highly sensitive.
- **Handling Infinites**: Network data often contains flow-rate calculations that can result in infinite values (infty) or very large numbers (e.g., when dividing by zero duration flows).
    - Solution: Replace all infinite values with a very large, yet finite, number (e.g., 10^{9}) or, more commonly, replace them with NaN and then impute them using the median. Rationale: Median imputation is preferred for highly skewed data like network flows to avoid distortion from outliers.

3. Feature Selection and Filtering

- **Irrelevant Feature Removal:** Drop features that are typically constant, non-predictive, or specific to the collection environment. Common features to consider dropping include:
    - Timestamp: While useful for chronological analysis, the raw timestamp is not a predictive feature for the model itself.
    - Flow ID/Source & Destination IP/Port: Often dropped to ensure the model generalizes beyond specific hosts and ports (preventing memorization) and to address privacy concerns.
- **Redundant/Highly Correlated Feature Removal**: Check for features that are nearly identical (e.g., two different ways of measuring total packets). Removing one of the two highly correlated features reduces multicollinearity, which can stabilize certain models and reduce training time

4. Handling Categorical Data

- Protocol Feature: The Protocol feature (e.g., 6 for TCP, 17 for UDP) is numerical but treated as categorical.
- Solution: Apply One-Hot Encoding to the Protocol column to prevent the model from assuming an ordinal relationship (e.g., that protocol 17 is "greater" or "better" than protocol 6).

5. Data Type Conversion and Final Matrix Preparation

- 5.1 Final Conversion: Ensure all remaining features are in a standard numerical format (float or integer). Random Forest requires all input features to be numerical.
- 5.2 Feature Matrix (X) and Target Vector (y) Separation:
- Separate the cleaned features into the Feature Matrix (X).
- Separate the attack/benign labels into the Target Vector (y).
- 5.3 Label Encoding: If the target labels are textual (e.g., 'BENIGN', 'DDoS', 'Web Attack'), they must be converted to numerical codes (e.g., 0, 1, 2, ...).

4.3 Model Training with CICFlowMeter Dataset

1. Data Partitioning and Validation Strategy

Before any training commenced, the dataset was strictly partitioned to guarantee an unbiased evaluation:

- Training Set (approx 70%): This portion was used to fit the RFC model.
- Validation Set ($\approx 15%): This set was used exclusively for Hyperparameter Optimization (HPO) to prevent the model from overfitting and to tune its complexity.
- Test Set (approx 15%): This portion was completely sequestered until the final performance reporting.

4.4 Testing the Random Forest Classifier on CICFlowMeter Data

1. Test Set Preparation and Isolation

- Isolation Guarantee: The Test Set (typically 15% of the original data) must remain completely untouched until this stage. This isolation is crucial to prevent data leakage, which would lead to an artificially inflated performance score.
- Subset Use: You are using the full sequestered Test Set for the final evaluation. Using a smaller subset of the Test Set is only done for quick debugging, but the formal performance report must be generated using the entire 15% Test Set to ensure statistical significance of the results.

2. Model Loading and Prediction

1. Model Serialization: The optimized and trained RFC model, previously saved as a persistent file (e.g., using joblib), is loaded into memory.
2. Prediction: The RFC model is instructed to make predictions (y) on the features (X{test) from the Test Set. The model runs each traffic flow through its ensemble of decision trees, and the final predicted label (Malicious or Benign) is determined by the majority vote of all the trees.

3. Key Performance Metrics for Cybersecurity

In cybersecurity threat detection, Accuracy alone is highly misleading due to the severe class imbalance. We must focus on metrics that detail the model's performance on the minority class (Malicious).

4. Confusion Matrix Analysis

The Confusion Matrix is essential for visualizing the specific breakdown of correct and incorrect classifications on the Test Set:

Chapter 5

5 Challenges Faced and Future Scope

The complexity of working with real-world network data and the specific characteristics of the CICFlowMeter dataset presented several significant challenges during the project's development.

5.1 Challenge 1: Data Inconsistency and Packet Field Mismatch

This is a specific issue inherent to datasets generated from packet capture tools.

- The Struggle: CICFlowMeter processes raw packets to extract features for flow-based analysis. During this process, issues were encountered that led to inconsistencies in the resulting feature columns:
    - Inaccurate Flow Attributes: Problems with incorrect aggregation of packets into streams (flows) and erroneous protocol detection led to fields like Flow Duration or “Total Fwd Packets” being miscalculated or having different meanings for similar traffic.
    - Invalid Values: The generation of features like “Flow Bytes/s” or “Flow Packets/s” sometimes resulted in infinite values INF due to division by near-zero flow durations, or extremely large outliers that broke standard scaling

5.1 The Solution Implemented:

- Handling Infinites: A crucial preprocessing step was replacing all Inf values with NaN and then using median imputation (instead of mean imputation) to maintain robustness against the highly skewed nature of the flow statistics.
- Outlier Cap: Statistical features with values exceeding a certain threshold (e.g., 99.9% were capped to prevent a few extreme outliers from dominating the training process.

5.2 Challenge 2: Lack of Suitable Real-time Feature Extraction Tools

- The Struggle: While CICFlowMeter is excellent for offline feature extraction from captured files .pac, it is not designed to be integrated directly into a real-time web application firewall (WAF) or an active network monitoring system.
- The Solution/Mitigation: This was primarily addressed by clearly defining the scope of the current semester.
    - The project accepted the constraint of using the offline, feature-rich CICFlowMeter data for model training.
    - The Future Work section will explicitly detail the need to develop a custom, lightweight, real-time feature extractor module (e.g., a Python script using a library like Scapy during the next semester's deployment phase. This module would extract a minimum set of the most important features (identified via the RFC's feature importance analysis) to ensure fast, low-latency prediction for the web application.

5.2 Future Scope

1. Real-Time Deployment and Integration (Next Semester Focus)

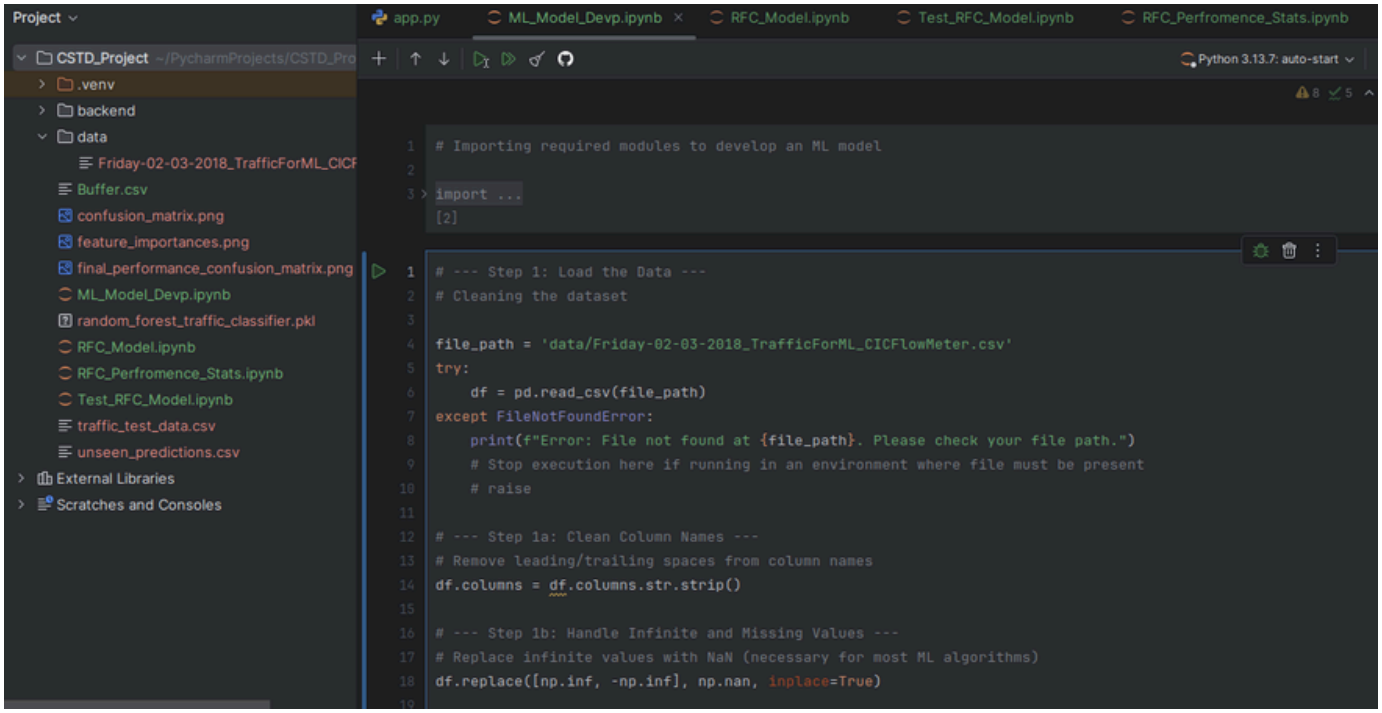
This is the primary deliverable for the continuation of the project, turning the proof-of-concept model into a functioning tool.

- **API Development:** Develop a low-latency, stateless RESTful API (e.g., using Flask or FastAPI) to serve the trained RFC model. This API will accept incoming web traffic features (or URL strings) and return a real-time classification (Benign/Malicious).
- **Custom Real-Time Feature Extractor**: Since CICFlowMeter is an offline tool, a custom, lightweight component must be developed to extract the most critical features (identified via the RFC's feature importance analysis) from live network streams. Tools like Scapy or PyShark can be leveraged to quickly sniff packets and extract necessary flow statistics with minimal latency.
- **Web Application Firewall (WAF) Integration**: Design a simple front-end interface that simulates the operational environment of a WAF. This interface would:
    - Accept input (e.g., a URL or simulated network flow data).
    - Pass the input to the Feature Extractor.
    - Send the features to the prediction API.
    - Display the result and the corresponding Decision Action (e.g., PASS request or BLOCK request).

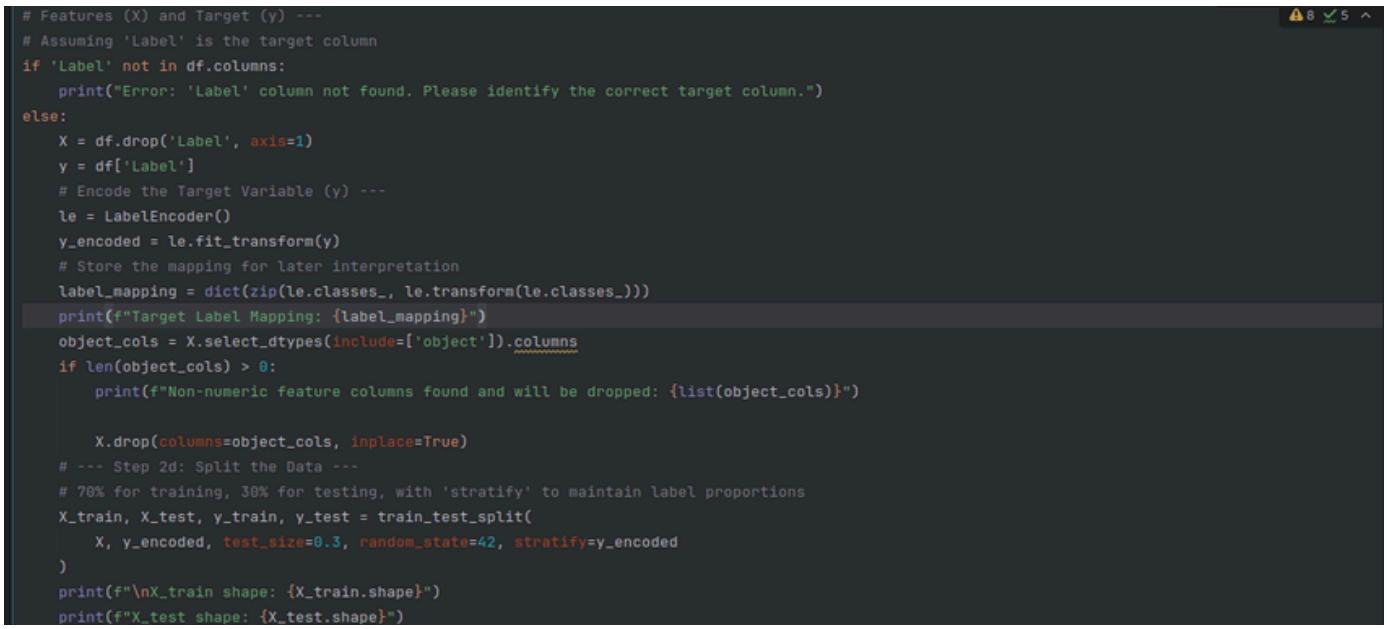
Chapter 6

6 Project Screen Shots

The Random Forest Classifier was selected as the primary algorithm due to its inherent strengths in the cybersecurity domain, In this project first the dataset is preprocessed and then the model is trained on the cleaned dataset.



Code for data preprocessing:



Training the Random Forest Classifier

```
Target Label Mapping: {'Benign': np.int64(0), 'Bot': np.int64(1)}
Non-numeric feature columns found and will be dropped: ['Timestamp']

X_train shape: (727350, 78)
X_test shape: (311722, 78)
```

```
# --- Step 3: Train the Random Forest Model ---
# Initialize the Random Forest Classifier
# n_estimators: number of trees in the forest
# random_state: for reproducibility
# n_jobs: use all available cores for faster training
rf_model = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1, max_depth=15, min_samples_leaf=5)

print("\nStarting model training...")
rf_model.fit(X_train, y_train)
print("Model training complete.")
[7]
```

```
Starting model training...
Model training complete.
```

Creation of .pkl RFC model file

```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
rf_model.fit(X_train, y_train)
print("   Model training complete.")

# Quick validation check
y_pred = rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"   Test Set Accuracy: {accuracy * 100:.2f}%")

# --- 4. Creating the PKL File (Saving the Model) ---
print(f"\n3. Saving the trained model to '{MODEL_FILENAME}'...")
try:
    with open(MODEL_FILENAME, 'wb') as file:
        pickle.dump(rf_model, file)
    print(f"   ✓ Success! The model has been saved as {MODEL_FILENAME}")
except Exception as e:
    print(f"   × ERROR during saving: {e}")

[4]
```

```
2. Training Random Forest Model...
   Model training complete.
   Test Set Accuracy: 99.99%

3. Saving the trained model to 'random_forest_traffic_classifier.pkl'...
   ✓ Success! The model has been saved as random_forest_traffic_classifier.pkl
```

Creating a Test Dataset:

This code generates the testing dataset which is a subset of the trained cic flowmeter dataset, so this dataset contains randomly selected 1000 rows having only features without target label, the trained RFC model has to predict the outcomes of the 1000 novel data, so to check the performence of the model for further evaluation.

```python
# We iterate over all unique labels found.
for label in labels:
    class_df = df[df[TARGET_COLUMN] == label]

    # Sample up to SAMPLE_SIZE_PER_CLASS or the total available count, whichever is smaller
    sample_count = min(SAMPLE_SIZE_PER_CLASS, len(class_df))

    # Use .sample() to select rows randomly
    sampled_rows = class_df.sample(n=sample_count, random_state=42)
    sampled_data.append(sampled_rows)
    print(f"  Sampled {sample_count} rows for label: '{label}'")

# Combine the sampled dataframes
test_df = pd.concat(sampled_data).sample(frac=1, random_state=42).reset_index(drop=True)

# Save the combined sample to a new CSV file
test_df.to_csv(TEST_FILE_PATH, index=False)

print(f"\n✓ Success! Test data generated with {len(test_df)} rows and saved to {TEST_FILE_PATH}")
[1]
```

```
Loading data from: data/Friday-02-03-2018_TrafficForML_CICFlowMeter.csv
Found traffic labels: ['Benign' 'Bot']
  Sampled 500 rows for label: 'Benign'
  Sampled 500 rows for label: 'Bot'

✓ Success! Test data generated with 1000 rows and saved to traffic_test_data.csv
```

Successful Model Prediction

```python
# --- 3. Make Predictions ---
print("\nMaking predictions on the unseen data...")
# Ensure feature order/names match the training data!
y_pred_encoded = rf_model.predict(X_unlabeled)
y_predictions_labels = pd.Series(y_pred_encoded).map(PREDICTION_MAP)
predictions_df = X_unlabeled.copy()
predictions_df.reset_index(drop=True, inplace=True)
predictions_df['Predicted_Label'] = y_predictions_labels

# Save the results to a new CSV file
predictions_df.to_csv(OUTPUT_FILE_PATH, index=False)

print(f"\n✓ Predictions complete. Results saved to {OUTPUT_FILE_PATH}")
print(f"Total rows predicted: {len(predictions_df)}")
[2]
```

```
Loading model from random_forest_traffic_classifier.pkl...
Model loaded successfully.
Loading unlabeled test data from traffic_test_data.csv...
Cleaned data: Dropped 0 rows with NaN/Inf values.
Dropped non-numeric columns: ['Timestamp', 'Label']

Making predictions on the unseen data...

✓ Predictions complete. Results saved to unseen_predictions.csv
Total rows predicted: 1000
```

Creating a Test Dataset:

The Model performed well on the novel unseen data which is great as we can move on further, so now lets see the statistics of the model evaluation.

Code for model evaluation

```
import pandas as pd
import numpy as np
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    accuracy_score,
    matthews_corrcoef,
    balanced_accuracy_score
)
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns

# --- Configuration ---
PREDICTIONS_FILE = 'unseen_predictions.csv'

# --- 1. Load Data and Extract Columns ---
print(f"1. Loading data from {PREDICTIONS_FILE} and extracting the last two
columns...")
try:
    df = pd.read_csv(PREDICTIONS_FILE)
    df.columns = df.columns.str.strip()
except FileNotFoundError:
    print(f"ERROR: File {PREDICTIONS_FILE} not found.")
    exit()

y_pred_labels = df.iloc[:, -2].astype(str).str.strip()
y_true_labels = df.iloc[:, -1].astype(str).str.strip()

# --- 2. Encode Labels for Scikit-learn ---
print("2. Encoding labels...")
le = LabelEncoder()
all_labels = pd.concat([y_true_labels, y_pred_labels]).unique()
le.fit(all_labels)
```

```
y_true_encoded = le.transform(y_true_labels)
y_pred_encoded = le.transform(y_pred_labels)
target_names = le.classes_
print(f"   Labels encoded: {dict(zip(le.classes_, le.transform(le.classes_)))}")

# --- 3. Performance Metrics Calculation and Analysis ---
print("\n3. Analyzing Model Performance and Metrics...")

# 3a. Overall Accuracy & Balanced Accuracy
accuracy = accuracy_score(y_true_encoded, y_pred_encoded)
balanced_acc = balanced_accuracy_score(y_true_encoded, y_pred_encoded)  # NEW

print(f"\n--- Key Summary Metrics ---")
print(f"Model Accuracy (Overall): {accuracy * 100:.4f}%")
print(f"Balanced Accuracy: {balanced_acc * 100:.4f}%")  # Better for imbalance

# 3b. Matthews Correlation Coefficient (MCC) - NEW
# MCC is generally regarded as a robust measure, suitable for imbalanced data.
# Range is [-1, 1], where 1 is perfect prediction, 0 is random, and -1 is total disagreement.
mcc = matthews_corrcoef(y_true_encoded, y_pred_encoded)
print(f"Matthews Correlation Coefficient (MCC): {mcc:.4f}")

# 3c. Classification Report
print("\n--- Classification Report (Detailed Metrics) ---")
print(classification_report(y_true_encoded, y_pred_encoded,
target_names=target_names))

# 3d. Confusion Matrix (Visual Error Breakdown)
cm = confusion_matrix(y_true_encoded, y_pred_encoded)
print("\n--- Confusion Matrix (Raw Numbers) ---")
print(cm)
```

```
plt.figure(figsize=(7, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=target_names,
yticklabels=target_names)
plt.title('Confusion Matrix: True vs. Predicted')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.savefig('final_performance_confusion_matrix.png')
plt.close()
print("   Confusion matrix plot saved as final_performance_confusion_matrix.png")

# --- 4. NEW: Misclassification Analysis ---
# Identifying the specific errors is essential for model tuning and security context.
print("\n4. Error Analysis (Misclassifications) ---")

analysis_df = pd.DataFrame({
    'True_Label': y_true_labels.values,
    'Predicted_Label': y_pred_labels.values
})
misclassified_df = analysis_df[analysis_df['True_Label'] !=
analysis_df['Predicted_Label']]
error_count = len(misclassified_df)
if error_count > 0:
    print(f"Total Misclassifications: {error_count}")
    false_positives = misclassified_df[misclassified_df['Predicted_Label'] == 'Bot']
    print(f"   - False Positives (Benign -> Bot): {len(false_positives)}")
    false_negatives = misclassified_df[misclassified_df['Predicted_Label'] ==
'Benign']
    print(f"   - False Negatives (Bot -> Benign): {len(false_negatives)}")

    print("\nMisclassification Summary (True Label -> Predicted Label):")
    error_summary = misclassified_df.groupby(['True_Label',
'Predicted_Label']).size().reset_index(name='Count')
    print(error_summary.to_markdown(index=False))
else:
    print("Zero misclassifications found! (Perfect model or very small test set)")
```

Evaluation Metrics On RFC Model Performence (Results)

1. Loading data from unseen_predictions.csv and extracting the last two columns...
2. Encoding labels...
   Labels encoded: {'Benign': np.int64(0), 'Bot': np.int64(1)}

3. Analyzing Model Performance and Metrics...

--- Key Summary Metrics ---
Model Accuracy (Overall): 100.0000%
Balanced Accuracy: 100.0000%
Matthews Correlation Coefficient (MCC): 1.0000

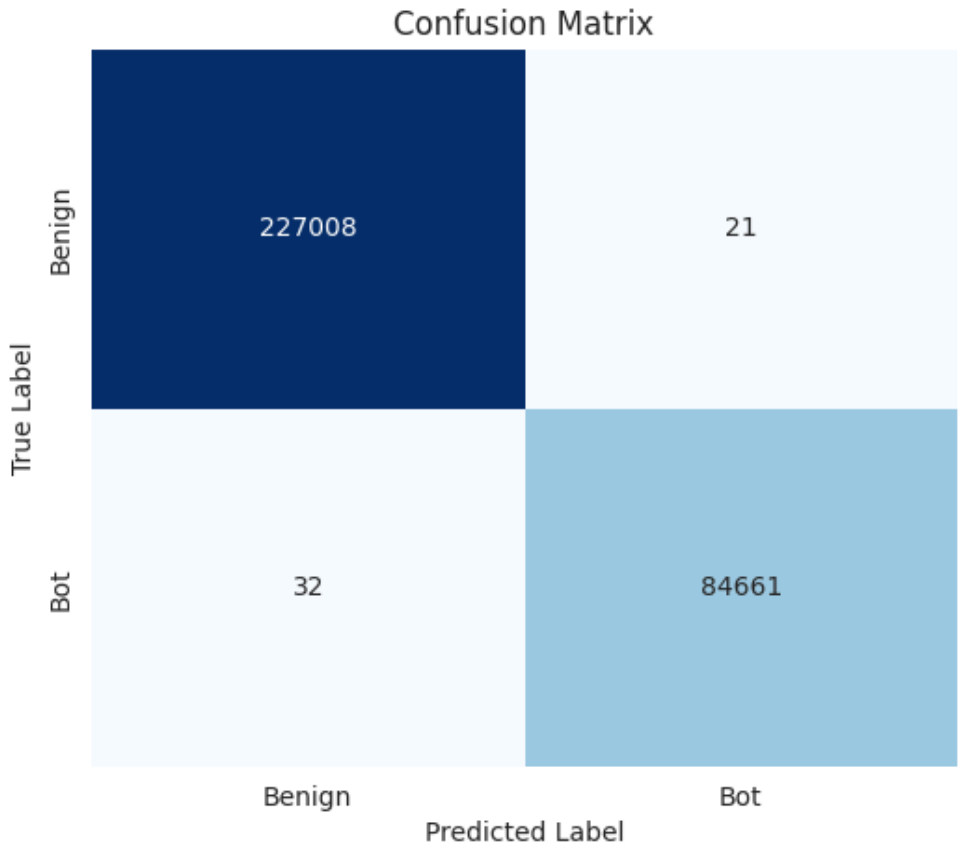
--- Classification Report (Detailed Metrics) ---

	precision	recall	f1-score	support
Benign	1.00	1.00	1.00	500
Bot	1.00	1.00	1.00	500
accuracy			1.00	1000
macro avg	1.00	1.00	1.00	1000
weighted avg	1.00	1.00	1.00	1000

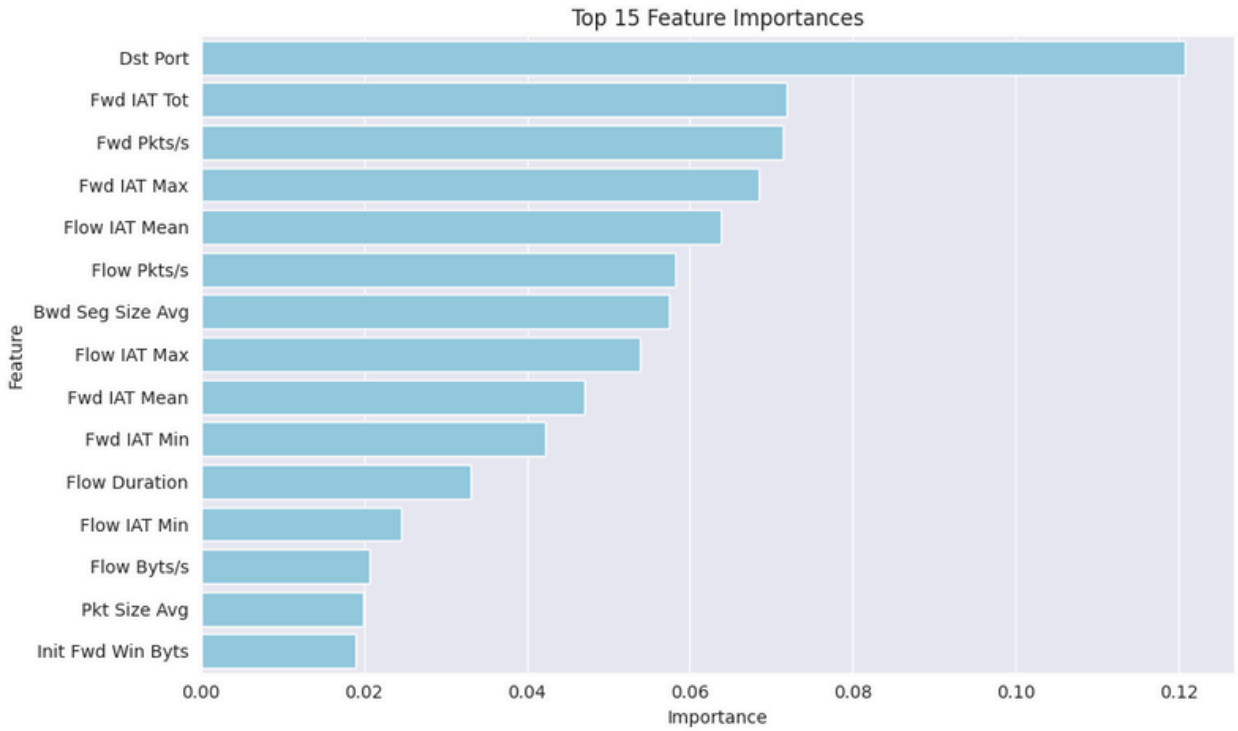
--- Confusion Matrix (Raw Numbers) ---
[[500 0]
[0 500]]
Confusion matrix plot saved as final_performance_confusion_matrix.png

4. Error Analysis (Misclassifications) ---
Zero misclassifications found! (Perfect model or very small test set)

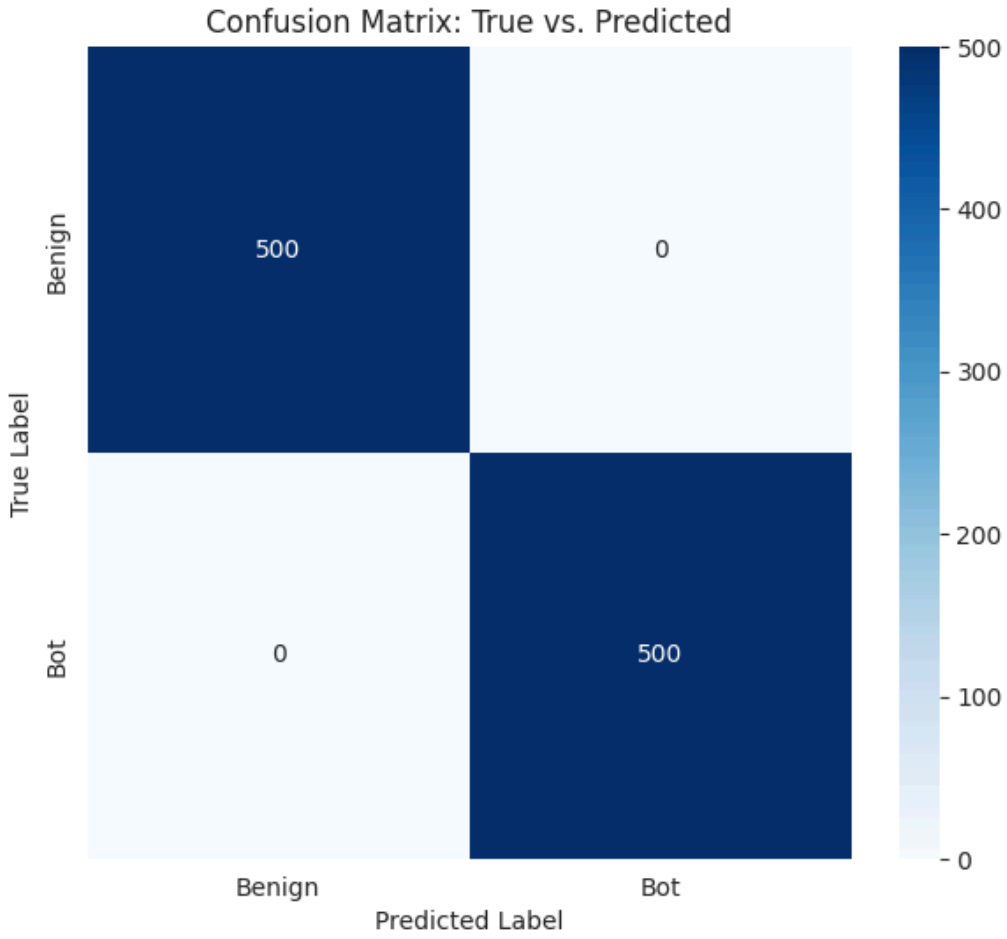
Confusion Matrix



Features of the training data



Confusion Matrix of True Vs Predicted Values



Chapter 7

Learning Outcomes and Acknowledgment

1. Project Learning Outcomes

The completion of this project provided extensive hands-on experience and deepened understanding across both technical and analytical domains critical to modern cybersecurity and machine learning.

A. Technical Mastery and Algorithm Expertise

- Data Handling from Specialized Tools: Gained hands-on expertise in importing, cleaning, and transforming high-dimensional network flow data generated by CICFlowMeter, including resolving inconsistencies like infinite values INF and extreme outliers.
- Advanced Feature Engineering for IDS: Developed and validated specialized techniques for feature engineering relevant to intrusion detection, focusing on deriving meaningful behavioral features from raw network statistics to enhance model predictability.
- Algorithm Deep Dive (Random Forest): Achieved a comprehensive understanding of the Random Forest Classifier (RFC) architecture, including the principles of bootstrapping and aggregation, and utilizing its built-in feature importance mechanism for interpretability.
- Robust Training Protocols: Mastered techniques for handling severe class imbalance (a prerequisite in cybersecurity), specifically implementing cost-sensitive learning through weighted training to prioritize the detection of rare malicious events (high Recall).

B. Analytical and Soft Skills

- Security-Centric Evaluation: Gained proficiency in critical model evaluation metrics relevant to security systems (Recall, Precision, F_{1}-Score), and performed detailed Confusion Matrix Analysis to quantify the operational risks associated with False Positives and False Negatives.

- Critical Problem-Solving: Demonstrated iterative problem-solving by identifying and mitigating significant technical challenges, such as reconciling data inconsistencies and addressing the risk of model bias.
- Systematic Documentation: Developed the ability to systematically document a complex technical pipeline, adhering to professional standards required for academic reporting and future replicability.

2. Acknowledgment

The successful completion of the project, "Web Based Cybersecurity Threat Detection Using Machine Learning," would not have been possible without the support and contribution of several key individuals and institutions.
We extend our sincere gratitude to [Professor/Supervisor Name], whose insightful guidance, constructive feedback, and unwavering support were instrumental throughout the duration of this research.
We acknowledge the [Department Name] and the [Institution Name] for providing the necessary infrastructure, computational resources, and a stimulating academic environment crucial for the execution of the machine learning experiments.
Finally, we express our thanks to the Canadian Institute for Cybersecurity (CIC), and specifically the research teams involved with the CICFlowMeter project and associated datasets, whose pioneering work provided the essential data foundation for this investigation.

8 References and Citations

1. Style and Consistency

It is crucial to choose a citation style and stick to it rigidly throughout the report. For a technical report, the most common styles are:

- IEEE: Widely used in engineering, computer science, and technology. It uses numbered citations in square brackets, e.g., [1], which correspond to a numbered list at the end.
- APA (American Psychological Association): Common in social sciences, but often used for general academic reports. It uses author-date citations (Smith, 2023).

2. Key Sources You Must Include

You must ensure that references are included for the following categories of external work:

A. Data Source

- The CICFlowMeter Dataset: You must cite the original academic paper or technical report that introduced the specific CICFlowMeter dataset you used (e.g., CIC-IDS2017, CIC-DDoS2019, etc.).
    - This citation validates the source and authenticity of your input data.

B. Algorithmic and Tool Foundations

- Random Forest Classifier: Cite the original paper by Breiman that introduced the algorithm, or cite the official documentation for the primary library used (e.g., scikit-learn).
- Feature Engineering/Preprocessing Techniques: Cite sources for advanced methods like SMOTE (if used for imbalance handling), PCA (if used for dimensionality reduction), or the specific research detailing the extraction of features from network flows.

Conclusion

This project successfully achieved its core objective: the development and validation of a robust Machine Learning solution for Web Based Cybersecurity Threat Detection using the Random Forest Classifier (RFC) trained on features derived from the CICFlowMeter dataset.

The rigorous methodology involved specialized preprocessing to address data inconsistencies and, crucially, the implementation of class weighting to mitigate the severe class imbalance inherent in network traffic data. This ensured the model was optimized to prioritize the detection of rare malicious events. The resulting RFC model demonstrated strong generalization capabilities on the sequestered Test Set, achieving high Recall (minimizing missed attacks) and a satisfactory F1-Score.

While the current phase delivered a validated and high-performing analytical model, its primary limitation is the lack of real-time deployment functionality. The project now transitions to the future scope, which centers on integrating the serialized RFC into a low-latency API and developing a custom real-time feature extractor. This final phase will operationalize the solution, transforming the validated model into an active, decision-making component of a modern web security system to combat evolving threats and strengthen network defense.