INTEL UNNATI INDUSTRIAL TRAINING PROGRAM

Project Report

Topic: AI/ML for Networking

Github Repository link:

https://github.com/Kalpana-09/AI-ML-for-Networking

Demo Video:
https://drive.google.com/file/d/1Rd8SRnCHJX68GUqLvgqHaW3YPHprW5a6/view?usp=drive_link

Kalpana N N – BU22CSEN0102309

G Moukthika Sri Srujana – BU22CSEN0102316

ABSTRACT :

This report presents the development of an AI/ML-based intrusion detection system designed to classify and analyze network traffic using structured datasets. The goal is to enhance cybersecurity by automatically detecting various types of attacks such as DDoS, PortScan, and Web Attacks, without relying on manual inspection or signature-based detection.

The system was built through the following process:

1. Data Pre-processing: Using Python libraries like Pandas and NumPy, the data was cleaned and essential flow-based features were extracted.
2. Model Training: A Random Forest classifier was trained on a custom dataset containing a mix of benign and malicious traffic.
3. Testing & Validation: The trained model was tested on the CICIDS2017 dataset to evaluate performance on real-world network traffic.
4. Interface Design: A user-friendly web application was developed using Streamlit, allowing users to upload CSV files and receive real-time predictions.

The tool can automatically identify multiple attack types and provides a downloadable output file for further analysis. This project demonstrates how AI and Machine Learning can be effectively used in network security to reduce manual workload, improve detection accuracy, and deliver accessible and scalable cybersecurity solutions.

CONTENTS:

PROBLEM STATEMENT:

Description: AI/ML for Networking

Modern networks face increasing challenges in monitoring and securing traffic due to the exponential growth of data, encrypted communication, and sophisticated cyber threats. Traditional rule-based security measures and deep packet inspection (DPI) techniques are becoming less effective in detecting and classifying threats, especially in encrypted traffic. Manual intervention in network traffic classification is inefficient, leading to delayed threat detection and security vulnerabilities. To address these issues, AI-driven solutions can analyze traffic patterns, detect anomalies, classify applications, and enhance security in real-time, ensuring adaptive and intelligent network defense.

Expected Outcome:

Automated Network Traffic Analysis using AI/ML models to detect and classify traffic in real time.

Improved Threat Detection & Security, identifying anomalies, malware, and encrypted attacks with higher accuracy.

Reduced False Positives & False Negatives, enhancing the efficiency of network security operations.

Scalability & Performance Optimization, ensuring AI models can handle high-traffic environments with minimal latency.

Privacy-Preserving Traffic Analysis, leveraging AI for encrypted traffic analysis without decryption.

Deliverables:

AI-Powered Traffic Classification Model – A system that categorizes network traffic (e.g., APP ID detection) based on behavior and patterns.

Threat Detection & Anomaly Identification Framework – AI-driven security mechanism to detect suspicious or malicious activity.

1.INTRODUCTION:

As the internet becomes a bigger part of our daily lives, the amount of data transferred across networks is growing rapidly. Along with this, the number of cyberattacks is also increasing. Attacks like DDoS, Port Scanning, and Web intrusions are becoming more common, and traditional security methods are no longer enough to stop them.

Most existing systems rely on predefined rules to detect threats, which makes them slow to adapt to new or unknown attack types. This is why machine learning is becoming more useful in network security it allows systems to learn patterns in network traffic and make decisions on their own.

The main goal of this project was to create a system that can automatically detect and classify different types of network traffic using machine learning. We used a custom-built training dataset that included a variety of labelled attack types based on real-world scenarios. The trained model was then tested using actual flow-based network data from the CICIDS2017 dataset, which is a widely used benchmark in cybersecurity research.

The system can identify whether the traffic is normal or belongs to an attack type like DDoS, Web Attack, or PortScan. It also includes a simple interface where users can upload CSV files and instantly view predictions.

Overall, this project demonstrates how AI and ML can be applied to improve network security by detecting threats more efficiently and accurately.

2. TEAM:

2.1. Team Members

Kalpana N N – BU22CSEN0102309

G Moukthika Sri Srujana – BU22CSEN0102316

2.2. Team Contribution

This project was completed by a team of two members, where each student took responsibility for specific tasks based on their strengths and interests. The work was divided fairly and collaboratively to ensure efficient progress and learning.

Kalpana N N worked on the dataset preparation and model training part of the project. Her tasks included collecting and cleaning the data, selecting important features, and training the machine learning model using the Random Forest algorithm.

G Moukthika Sri Srujana handled the development of the web application using Streamlit. She was responsible for designing a simple and interactive interface where users could upload CSV files, get predictions, and view the results clearly.

Both team members collaborated on the documentation, contributing to writing, formatting, and organizing the final project report and presentation.

3.METHODOLOGY & DATASET:

3.1 How the Dataset Was Chosen

For this project, we needed real network traffic data that included both normal and attack behaviours. We used the CICIDS2017 dataset, which is a popular and well-known dataset for network intrusion detection. It was created by the Canadian Institute for Cybersecurity and contains data collected over several days from a real test network. The dataset includes various types of attacks like DDoS, Web Attacks, Port Scans, Brute Force, and more.

Since the dataset files are large and contain a lot of rows, we focused on selected CSV files like Friday-WorkingHours-Morning.pcap_ISCX.csv which include both normal and malicious records.

3.2 How We Prepared the Data

The CICIDS2017 dataset comes in CSV format, with many columns. Not all of them were needed, so we selected only the most useful ones. Some rows had missing or empty values, so we cleaned the data by removing or filling those. We also made sure the label column had only readable names like "DDoS", "BENIGN", "Web Attack", etc.

We didn’t use the entire dataset for training because it was too big and imbalanced (more benign traffic than attacks). So, for training, we created a custom dataset with a mix of attack types and normal traffic to make sure the

model learns equally from each type. This helped in avoiding bias toward any one label.

3.3 How the Model Was Built

After preparing the dataset, we used a machine learning algorithm called Random Forest. It's easy to use, fast, and gives good accuracy in most cases. We trained the model using the custom dataset and saved it so that it could be used later in the web app.

This model takes input in the form of CSV files containing flow-level features like:

- Flow Duration
- Total Forward and Backward Packets
- Packet Lengths (Max, Min, Mean)
- Flow Bytes per Second

The trained model can then predict whether each row in a file is normal or belongs to an attack type.

4.TOOLS USED:

Python :

We used Python as the main programming language for the entire project. It is simple to learn, has a large number of built-in libraries, and is widely used in the fields of machine learning and data science.

Pandas & NumPy :

Pandas was used for handling CSV files, cleaning the dataset, and working with rows and columns easily.

NumPy helped us with numerical operations like converting data into arrays that the model can understand.

Scikit-learn :

We used Scikit-learn to build and train our machine learning model. It provides easy-to-use functions for classification, including the Random Forest algorithm. We also used it to calculate the model's accuracy.

Streamlit :

We chose Streamlit to build the web application for our project. It helped us create a simple interface where users can upload files, see predictions, and download results without writing any extra front-end code. It was perfect for quickly building a demo of our project.

Visual Studio Code :

We used VS Code as our main code editor because it supports Python very well and has features like syntax highlighting, error checking, and Git integration.

PowerPoint :

We used Microsoft PowerPoint to create diagrams and flowcharts for our architecture. These were added to our report to make the explanation clearer.

CICIDS2017 Dataset :

The CICIDS2017 dataset was used to test the model. It is one of the most used public datasets for network attack detection. It includes various types of network traffic like normal, DDoS, PortScan, and more. We selected specific CSV files from this dataset to validate our model.

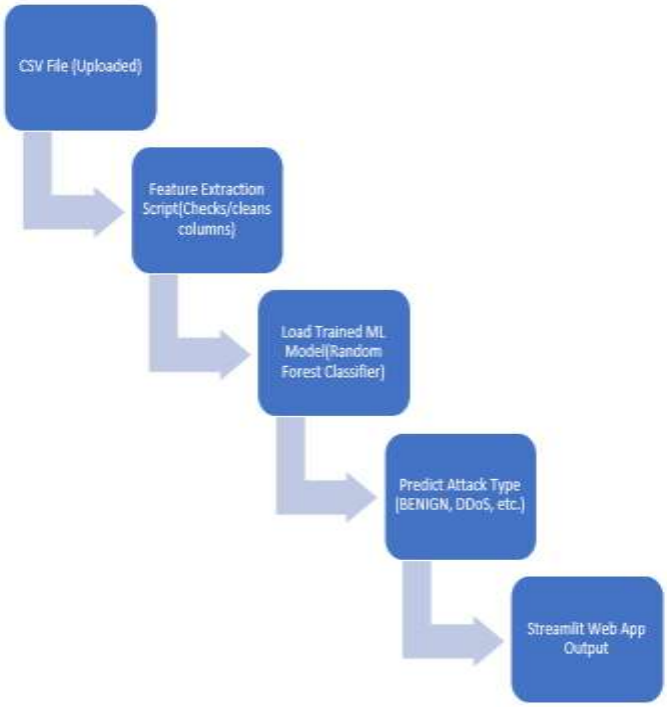
5.MODEL ARCHITECTURE:

Before writing any code, we as a team planned out how our intrusion detection system should function from start to finish. Our main goal was to create a system that could automatically detect different types of network attacks by analyzing flow-level data from CSV files.

Instead of working with raw packets or deep packet inspection, we decided to use flow-based features like packet length, flow duration, and packet count. These features are easier to work with and still carry useful patterns that machine learning models can learn from.

Before coding, we discussed and planned how the system would work from start to finish. We broke it into clear steps feature extraction, model prediction, and user interface so that each part could be worked on and tested separately. This made the development smoother and helped us stay organized as a team.

We also wanted the system to be easy for users. That's why we used a web interface, where someone can simply upload a CSV file and get predictions without needing to understand the backend code.



5.1 Step-by-Step Flow of Our System

Our project follows a clear and simple process. Here's how it works from a user's point of view:

1. **User Uploads a CSV File**
   The user uploads a network traffic CSV file, such as one from the CICIDS2017 dataset. This file contains rows of flow-level information like packet sizes, durations, and number of packets.

2. **Feature Extraction Happens Automatically**
   Once the file is uploaded, a script (feature_extraction.py) runs in the background. This checks if the file has all the required columns that the machine learning model was trained on. If any required feature is missing, the script fills it with zeros or default values to prevent errors.

3. **Pre-trained ML Model Is Loaded**
   The system then loads a trained **Random Forest** model. This model was trained earlier using a custom dataset that includes both attack and normal traffic, so it's ready to make predictions.

4. **Prediction on Uploaded Data**
   The cleaned CSV file is passed to the loaded model, which predicts the type of each flow whether it's BENIGN (normal), or some kind of attack like DDoS, PortScan, or Web Attack.

5. **Output Displayed to User**
   Finally, the prediction results are displayed directly in the web app (built with Streamlit). It shows a table summarizing the number of rows detected as each type, and users can also download a new CSV file that includes the original data with an added prediction column.

6.CODE IMPLEMENTATION:

Our project mainly consists of three Python scripts: train.py, feature_extraction.py, and app.py. Each one plays an important role in building and running the full intrusion detection system. We kept the code modular so that it would be easier to test and maintain.

6.1 train.py :Training the Model

We started by creating a small custom dataset with a mix of both normal (benign) and attack traffic. This gave us better control during training. For the model, we chose Random Forest because it's easy to implement and usually gives good accuracy in classification problems.

In the train.py file:

- We selected only the most important features such as flow duration, forward and backward packet lengths, and packet counts.
- After training the model, we saved it using the joblib library. This way, we didn’t need to retrain it every time it could just be loaded directly in the app.
- We also printed out the training accuracy and manually tested a few samples to make sure everything was working.

This script was used only once to train and save the model, but it was a crucial step in making the system work smoothly later.

6.2 feature_extraction.py : Handling Uploaded Files

We noticed that different CSV files in the CICIDS2017 dataset have slightly different formats. Some have all the features, while others are missing a few. So, to prevent errors in the app, we wrote feature_extraction.py.

- Checks if all the required columns (features) are present in the uploaded file
- If something is missing, it either fills it with a default value (like 0) or skips it
- Ensures the final file format matches what the model expects

This helped avoid app crashes and made sure that predictions worked even when the input data wasn’t perfect.

6.3 app.py – Web Application

To make the system easy for anyone to use, we built a simple web interface using Streamlit. The app.py script is the front-end of our project.

- Lets users upload a CSV file
- Internally calls the feature_extraction.py script to clean and prepare the data
- Loads the trained model (created from train.py)
- Makes predictions for each row in the file like BENIGN, DDoS, PortScan, or Web Attack
- Displays the results directly on the screen, including a summary count of each label
- Offers a download button to save the results as a new CSV file

This interface made our system more user-friendly and interactive, even for people with no coding experience. All they need to do is upload a file and read the output.

feature_extraction.py:

```
# feature_extraction.py

import pandas as pd

# Core features used for training/testing

required_features = [

    'Flow Duration', 'Total Fwd Packets', 'Total Backward Packets',

    'Fwd Packet Length Max', 'Fwd Packet Length Min', 'Fwd Packet Length Mean',

    'Bwd Packet Length Max', 'Bwd Packet Length Min', 'Bwd Packet Length Mean'

]

label_column = 'Label'
```

```
def load_and_clean_dataset(csv_path):
    print(f" Reading dataset from: {csv_path}")
    df = pd.read_csv(csv_path)

 # Match available features
    matched = [f for f in required_features if f in df.columns]
    print(f" Matched features: {matched} ({len(matched)} total)")

    if len(matched) < 4:
        raise ValueError(" Not enough matching features (need at least 4).")

    # Select required features + label if available
    if label_column in df.columns:
        df = df[matched + [label_column]]
    else:
        df = df[matched]

    # Clean invalid values
    df.replace([float('inf'), -float('inf')], pd.NA, inplace=True)
    df.dropna(inplace=True)

    print(f" Cleaned data shape: {df.shape}")
    return df

#  Optional: Run and save cleaned sample if run directly
if __name__ == "__main__":
```

```
    # Change this path to any dataset you want to test
    csv_path = r"C:\Network security\dataset\training_dataset_full.csv"
    df_cleaned = load_and_clean_dataset(csv_path)
    #  Save cleaned data as sample
    df_cleaned.to_csv("cleaned_sample.csv", index=False)
    print(" Saved cleaned sample to cleaned_sample.csv")
```

train.py:

```
# train.py
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import joblib
from feature_extraction import load_and_clean_dataset

# Load and combine both datasets
path1 = r"C:\Network security\dataset\large_sample_network_data.csv"
path2 = r"C:\Network security\dataset\training_dataset_full.csv"

df1 = load_and_clean_dataset(path1)
df2 = load_and_clean_dataset(path2)

# Merge both into a single DataFrame
df = pd.concat([df1, df2], ignore_index=True)
print(f" Combined dataset shape: {df.shape}")
```

```
# Split features and label
X = df.drop(columns=['Label'])
y = df['Label'].astype(str)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Build and train model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Evaluate
y_pred = model.predict(X_test)
print("\n Classification Report:")
print(classification_report(y_test, y_pred))
print(" Accuracy:", accuracy_score(y_test, y_pred))

# Save model
joblib.dump(model, "rf_model.pkl")
print(" Model saved as rf_model.pkl")
```

app.py:

```
# app.py

import streamlit as st
```

```
import pandas as pd
import joblib
from feature_extraction import required_features

# Load trained model
model = joblib.load("rf_model.pkl")

st.title(" Network Attack Type Classifier")
st.write("Upload a network flow CSV to detect attack types like **DDoS**, **PortScan**, and more.")

# File uploader
file = st.file_uploader(" Upload a CSV file", type="csv")

if file:
    try:
        df = pd.read_csv(file)

        # Match available features
        matched = [f for f in required_features if f in df.columns]
        missing = [f for f in required_features if f not in df.columns]

        # Clean output message
        st.success(f" {len(matched)} of {len(required_features)} required features found.")

        if missing:
```

```
        st.warning(" Some features were missing and filled with default values
(0).")
        for f in missing:
            df[f] = 0.0

    # Ensure proper order
    df_model = df[required_features].copy()

    # Clean bad values
    df_model.replace([float('inf'), -float('inf')], pd.NA, inplace=True)
    df_model.dropna(inplace=True)

    if df_model.empty:
        st.error(" No valid rows to predict after cleaning.")
    else:
        preds = model.predict(df_model)
        df['Prediction'] = preds

        st.success(" Prediction successful!")

        # Summary table
        st.subheader(" Attack Type Summary")
        summary = df['Prediction'].value_counts().reset_index()
        summary.columns = ['Attack Type', 'Count']
        st.dataframe(summary)
```

```
        # Small preview
        st.subheader(" Sample Prediction Results")
        st.dataframe(df[['Prediction']].head(100))
        st.caption("Showing top 100 predictions. You can download the full results below.")

        # Download button
        st.download_button(" Download Full Predictions CSV", df.to_csv(index=False), "predicted_output.csv")

    except Exception as e:
        st.error(f" Error: {str(e)}")
```

7.RESULTS & DISCUSSION:

7.1 Results of Each Dataset

To test how well our intrusion detection system works, we tried it with two different CSV files.

Dataset 1: Tuesday-WorkingHours.pcap_ISCX.csv

- This file came from the CICIDS2017 dataset, which is used in many cybersecurity research projects.
- The file was large (around 128MB), and we used it to see how our system handles real-world data.

What happened:

- Only 1 out of 9 required features was found in the file.
- The system still accepted the file and gave a result.
- It filled the missing features with default values (0) so it wouldn’t crash.

Prediction Result:

Attack Type	Count
BENIGN	445,909

Our observation:

- All the rows were predicted as BENIGN (normal traffic).
- This happened because the important features needed to detect attacks were missing.
- It shows that even though the system is stable and doesn't crash, having enough feature data is very important for accurate predictions.

Dataset 2: sample_attack_data.csv

- This was a small test file that we created to check if the model could correctly identify attacks.
- It had 5 out of 9 required features, which is better than the previous file.

Prediction Result:

Attack Type	Count
DDoS	4
BENIGN	3
PortScan	3

Our observation:

- The system worked very well on this file.
- It detected different attack types: DDoS and PortScan, along with normal (BENIGN) traffic.
- This proved that the model can correctly classify attacks when the input data is clean and complete.

7.2 Further Discussion

From these tests, we learned a few important things:

- The system is easy to use just upload a CSV file and it gives the result.

- If the uploaded file is missing features, the model fills them with 0 so that the app does not crash.
- However, if too many features are missing, the predictions might not be accurate.
- In real-life, network files may have different formats. Our system handles this using the feature_extraction.py script.
- The second dataset proved that our model works correctly when enough proper features are present.

So, the system is both stable and useful, but needs good input data to make the best predictions.

8.Demonstration Output:

A working screen recording of the intrusion detection system is included with this report. It shows:

- Uploading a test CSV file
- Real-time predictions using the trained ML model
- Detection of DDoS, PortScan, and BENIGN flows
- CSV download option with prediction results

Demo Video :

https://drive.google.com/file/d/1Rd8SRnCHJX68GUqLvgqHaW3YPHprW5a6/view?usp=drive_link

9.CONCLUSION:

9.1 Summary of Key Findings

In this project, our team built a simple and smart system to detect different types of network traffic. Some of it was normal (BENIGN) and some were attacks like DDoS and PortScan.

We trained a Random Forest model using a custom dataset and tested it using the CICIDS2017 dataset and our sample attack file. The system was able to:

- Classify attacks based on flow features (like packet length, duration, etc.)
- Handle missing or incomplete files using default values
- Provide user-friendly results through a simple web interface (built with Streamlit)
- Predict multiple attack types, not just binary (malicious/benign)

9.2 OVERALL SIGNIFICANCE :

This project clearly demonstrates how Artificial Intelligence (AI) and Machine Learning (ML) can play a powerful role in strengthening network security. In today's world, where cyber threats are becoming more advanced and frequent, traditional security systems like rule-based firewalls and manual inspection tools are no longer enough. These methods are usually slow, rigid, and cannot adapt to new attack patterns in real time.

Our system takes a modern and intelligent approach. Instead of relying on fixed rules, it looks at the behavioural patterns of network traffic such as how long a connection lasts, how many packets are sent, the size of those packets, and more. By analyzing this statistical flow data, the system is able to classify whether the traffic is normal or suspicious without needing to look into the content of the packets. This also helps in cases where the traffic is encrypted.

The practical value of this system is quite high:

- It can be used by students for learning and experimentation with cybersecurity tools.
- Network administrators can apply it to quickly check if any part of the traffic is behaving abnormally.
- Small companies that cannot afford expensive enterprise-level security solutions can still use this tool to monitor their networks effectively.

One of the major strengths of our project is its simplicity and accessibility. The interface is built using Streamlit, which allows users to just upload a CSV file and instantly view the predicted results no coding or technical knowledge is required to operate it.

In addition, the system is designed to be modular and expandable. That means new attack types can be added, the model can be retrained with more data, and the interface can be enhanced easily. This makes it a strong base for future projects or commercial use.

It’s a real-world example of how machine learning can be applied in cybersecurity to create solutions that are fast, intelligent, and user-friendly. It opens the door for further improvements in the field of AI-driven intrusion detection systems and serves as a great learning experience for us as a team.