INTEL UNNATI
INTERNSHIP-2025

AI/ML FOR NETWORKING

Problem Statement

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) are becoming less effective, especially with encrypted traffic.

To address these issues, AI/ML-powered solutions can:

- Analyze traffic patterns
- Detect anomalies
- Classify network applications
- Enhance real-time security

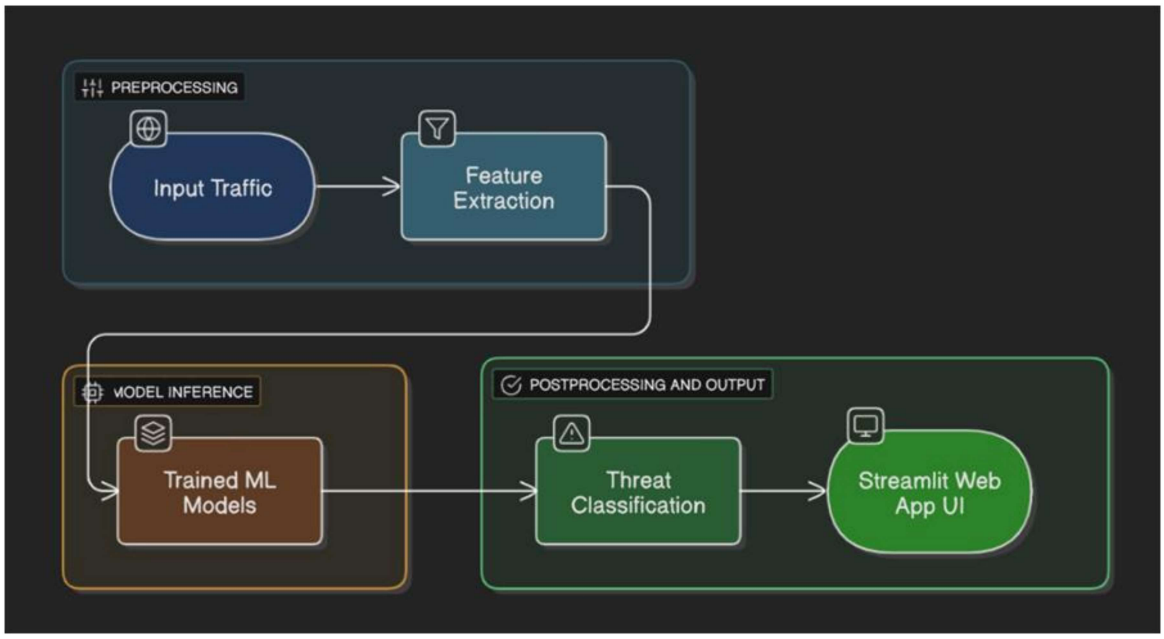
Team Members:

1. GONGADI AKHILESH – Worked on URL-based threat detection model.

2. BAVIGADDA MANI KUMAR REDDY – Built and deployed the real-time packet classification system.

3. SIRIVELLA VAMSI KRISHNA – Developed the Streamlit web interface and integrated both models into a unified UI.

System Architecture Diagram:



How the Code Works:

1. URL Classification – by Gongadi Akhilesh

- Dataset : malicious_phish.csv
- Features extracted using feature_extraction_url.py
- Model : RandomForestClassifier
- Accuracy : 98%
- Output: model/rf_url_model.pkl

2. Real-Time Packet Classification – by Bavigadda Mani Kumar

- Dataset : Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv
- Features : Flow Duration, ACK Flag Count, Fwd Packet/s, etc.
- Model : RandomForestClassifier

- Accuracy : 99%
- Output : model/realtime_rf_model.pkl

3. Streamlit UI Integration – by Sirivella Vamsi Krishna
- URL Detection: Single URL or CSV
- Real-Time Monitoring: Reads from logs/sniffer_output.txt
- Auto-refresh using streamlit_autorefresh
- Uses joblib to load models

4. Packet Sniffer – by Bavigadda Mani Kumar
- Uses pyshark to capture live packets
- Extracts length, builds feature vector
- Predicts using realtime_rf_model.pkl
- Logs to logs/sniffer_output.txt

Code Files Overview :
- train_url_model.py

-load_top_domains.py
- feature_extraction_url.py
- train_realtime_model.py

- pyshark_packet_sniffer.py
- app.py

Model Performance Summary

URL Classification Model :

```
E:\network_>python -m utils.train_url_model
Classification Report:

              precision    recall  f1-score   support

           0       0.90      0.97      0.93     85778
           1       0.93      0.79      0.86     44461

    accuracy                           0.91    130239
   macro avg       0.91      0.88      0.89    130239
weighted avg       0.91      0.91      0.91    130239

✅ Model saved to model/rf_url_model.pkl
```

Real-Time Traffic Model :

```
E:\network-threat-detector>python utils/train_realtime_model.py
📊 Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     33639
           1       0.74      0.91      0.82       431

    accuracy                           0.99     34070
   macro avg       0.87      0.95      0.91     34070
weighted avg       1.00      0.99      1.00     34070

✅ Model saved to ../model/realtime_rf_model.pkl
```

Source Code with Explanation:

train_url_model.py

```
import os
 import pandas as pd
 import joblib
 from sklearn.ensemble import RandomForestClassifier
 from sklearn.model_selection import train_test_split
 from sklearn.metrics import classification_report

from utils.feature_extraction_url import extract_features
```

```
# Load dataset
 df = pd.read_csv("malicious_phish.csv")

df = df.dropna(subset=["url", "type"])

X = [extract_features(url) for url in df["url"]]

y = [0 if label.lower() == "benign" else 1 for label in df["type"]]

X_train, X_test, y_train, y_test = train_test_split(
     X, y, test_size=0.2, random_state=42
 )

model = RandomForestClassifier(n_estimators=100, random_state=42)
 model.fit(X_train, y_train)

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

os.makedirs("model", exist_ok=True)

joblib.dump(model, "model/rf_url_model.pkl")
 print("✅ Model saved to ../model/rf_url_model.pkl")
```

This script loads the phishing URL dataset, extracts features from each URL using a separate utility, trains a Random Forest model to classify URLs as benign or malicious, and saves the model using joblib.

feature_extraction_url.py

```
import re
 import pandas as pd
 from urllib.parse import urlparse
 from .load_top_domains import load_top_domains

TOP_DOMAINS = load_top_domains("tranco_8L2QV.csv")

def extract_features(url):
     features = []

    # Parse the URL
```

```
    parsed = urlparse(url)
    hostname = parsed.netloc.lower()
    scheme = parsed.scheme.lower()

    # Basic length-based features
    features.append(len(url))                  # Full URL length
    features.append(len(parsed.netloc))        # Domain length
    features.append(len(parsed.path))          # Path length

    # Special character counts
    special_chars = ['@', '?', '-', '=', '.', '#', '%', '+', '$', '!', '*', ',', '&']
    features.extend([url.count(c) for c in special_chars])

    # Malicious keyword indicators
    keywords = ['alert', 'script', 'onerror', 'onload', 'select', 'drop', 'union', '--',
'insert']
    features.extend([1 if kw in url.lower() else 0 for kw in keywords])

    features.append(sum(c.isdigit() for c in url))
    features.append(sum(c.isupper() for c in url))

    ip_pattern = re.compile(r'^(\d{1,3}\.){3}\d{1,3}$')
    features.append(1 if ip_pattern.fullmatch(hostname) else 0)

    features.append(hostname.count('.'))

    features.append(1 if scheme == 'https' else 0)

    root_domain = hostname.split(':')[0]
    if root_domain.startswith("www."):
        root_domain = root_domain[4:]
    features.append(1 if root_domain in TOP_DOMAINS else 0)

    return features
```

This module parses each URL and extracts meaningful features like URL length, domain, path, special characters, use of HTTPS, presence of suspicious keywords, and whether it appears in the top domains list. These features are used for model training.

train_realtime_model.py

```
# utils/train_realtime_model.py
import pandas as pd
import joblib
```

```
 import os
 from sklearn.ensemble import RandomForestClassifier
 from sklearn.model_selection import train_test_split

 from sklearn.metrics import classification_report


# Load dataset
 df = pd.read_csv("../Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv")

df.columns = df.columns.str.strip()

if "Label" not in df.columns:
     print("❌ 'Label' column not found! Available columns:")
     print(df.columns.tolist())
     exit()

df.replace([float("inf"), float("-inf")], 0, inplace=True)
 df.dropna(inplace=True)

df["Label"] = df["Label"].apply(lambda x: 0 if str(x).strip().upper() == "BENIGN" else 1)

features = [
     "Flow Duration", "Total Fwd Packets", "Total Backward Packets", "Flow Bytes/s",
     "Flow Packets/s", "Fwd Packet Length Mean", "Bwd Packet Length Mean",
     "Packet Length Mean", "Packet Length Std", "PSH Flag Count", "ACK Flag Count",
     "URG Flag Count", "Fwd Packets/s", "Avg Fwd Segment Size"
 ]
 available = [col for col in features if col in df.columns]
 df = df[available + ["Label"]]

X = df.drop(columns=["Label"])
 y = df["Label"]
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestClassifier(n_estimators=50, max_depth=10, random_state=42)
 model.fit(X_train, y_train)

print("📊 Classification Report:")
 print(classification_report(y_test, model.predict(X_test)))

os.makedirs("../model", exist_ok=True)
 joblib.dump((model, X.columns.tolist()), "model/realtime_rf_model.pkl")
 print("✅ Model saved to ../model/realtime_rf_model.pkl")
```

This script uses network traffic data (CSV from CICIDS dataset), cleans and processes the data, selects relevant

features, trains a Random Forest classifier to predict if traffic is benign or malicious, and stores the trained model.

pyshark_packet_sniffer.py

```
import pyshark
 import joblib
 import pandas as pd
 import time
 import os
from datetime import datetime


model_path = os.path.join("model", "realtime_rf_model.pkl")
 rf_net_model, netflow_feature_cols = joblib.load(model_path)


os.makedirs("logs", exist_ok=True)
 log_file_path = os.path.join("logs", "sniffer_output.txt")


def predict_packet(length):
     features = {
         "Flow Duration": 1,
         "Total Fwd Packets": 1,
         "Total Backward Packets": 1,
         "Flow Bytes/s": length,
         "Flow Packets/s": 1,
         "Fwd Packet Length Mean": length,
         "Bwd Packet Length Mean": length,
         "Packet Length Mean": length,
         "Packet Length Std": 0,
         "PSH Flag Count": 0,
         "ACK Flag Count": 0,
         "URG Flag Count": 0,
         "Fwd Packets/s": 1,
         "Avg Fwd Segment Size": length
     }


    df = pd.DataFrame([features])
     for col in netflow_feature_cols:
         if col not in df.columns:
             df[col] = 0
     df = df[netflow_feature_cols]
     pred = rf_net_model.predict(df)[0]
     return "Benign" if pred == 0 else "Malicious"


def start_sniffing(interface="Wi-Fi"):
     try:
         cap = pyshark.LiveCapture(interface=interface, bpf_filter="ip")


        print(f"[INFO] Starting packet capture on interface: {interface}")
         for pkt in cap.sniff_continuously():
             try:
```

```
                length = int(pkt.length)
                label = predict_packet(length)
                timestamp = datetime.now().strftime("%H:%M:%S")
                log_line = f"[{timestamp}] {label}\n"

               # Write to log file
                with open(log_file_path, "a") as log_file:
                    log_file.write(log_line)

               print(log_line.strip())

            except Exception as e:
                print(f"[ERROR] Packet processing failed: {e}")
    except Exception as e:
        print(f"[FATAL] Failed to start sniffing: {e}")

if __name__ == "__main__":
    start_sniffing(interface="Wi-Fi")
```

This script captures live packets using PyShark, builds a minimal feature vector from the packet's size, uses the trained real-time model to classify it as benign or malicious, and logs predictions with timestamps.

App.py

```
import streamlit as st
import pandas as pd
import joblib
import time
from datetime import datetime
import threading
import queue
from utils.feature_extraction_url import extract_features

# Load models
rf_url_model = joblib.load("model/rf_url_model.pkl")
rf_net_model, netflow_feature_cols = joblib.load("model/realtime_rf_model.pkl")

# Streamlit UI
st.set_page_config(page_title="Network Threat Detector", layout="wide")
st.title("🛡️ AI-Powered Network Threat Detector")

mode = st.radio("Choose Detection Mode:", ["URL-based (SQLi/XSS/Phishing)", "Live Network
Traffic Monitoring (PyShark)"])
```

```
# 1 URL Detection
 if mode == "URL-based (SQLi/XSS/Phishing)":
     st.subheader("🔍 Detect Malicious URLs")
     option = st.radio("Choose Input Type:", ["Upload CSV", "Enter Single URL"])

    if option == "Upload CSV":
         uploaded_file = st.file_uploader("Upload a CSV file with a 'url' column",
type=["csv"])
         if uploaded_file:
             df = pd.read_csv(uploaded_file)
             if "url" in df.columns:
                 df["features"] = df["url"].apply(extract_features)
                 df["prediction"] = df["features"].apply(lambda x:
rf_url_model.predict([x])[0])
                 df["result"] = df["prediction"].apply(lambda pred: "🟢 Benign" if pred == 0
else f"🔴 {str(pred).capitalize()}")
                 st.dataframe(df[["url", "result"]])
             else:
                 st.error("CSV must contain a column named 'url'")

    else:
         url = st.text_input("Enter a URL:")
         if url:
             features = extract_features(url)
             pred = rf_url_model.predict([features])[0]
             result = "🟢 Benign" if pred == 0 else f"🔴 {str(pred).capitalize()}"
             https_used = "Yes" if features[-1] == 1 else "No"
             st.markdown(f"### Prediction: {result}")
             st.markdown(f"**HTTPS used?** {https_used}")

elif mode == "Live Network Traffic Monitoring (PyShark)":
     import os
     from streamlit_autorefresh import st_autorefresh

    st.subheader("📡 Real-time Network Packet Monitoring")
     st.info("ℹ️ Make sure `utils/pyshark_packet_sniffer.py` is running in the background and
writing to `logs/sniffer_output.txt`.")


    auto_refresh = st.checkbox("🔄 Auto-refresh every 2 seconds", value=True)
     if auto_refresh:
         st_autorefresh(interval=2000, limit=None, key="sniffer_autorefresh")


    if st.button("🧹 Clear Log"):
         try:
             open("logs/sniffer_output.txt", "w").close()
             st.success("✅ Log cleared.")
```

```
        except Exception as e:
            st.error(f"❌ Could not clear log: {e}")



    st.markdown("### 📋 Real-time Predictions")
     result_area = st.empty()

    def read_logs():
         try:
             with open("logs/sniffer_output.txt", "r") as f:
                 lines = f.readlines()
                 if not lines:
                     return ["🔎 Waiting for predictions..."]
                 return lines[-20:]
         except FileNotFoundError:
             return ["❌ Log file 'sniffer_output.txt' not found. Make sure the sniffer
script is running."]

    logs = read_logs()
     result_area.code("".join(logs))
```

This is the main Streamlit app file that provides a UI to detect malicious URLs from CSV/inputs and to monitor live network packets. It loads both the trained models and interacts with the user for real-time classification.

PROJECT VIDEO LINK:

https://drive.google.com/file/d/1GU2YGVQ49JmRmidfRoLfJ3t4MLD4xuDh/view?usp=drivesdk

GitHub:

https://github.com/Akhilesh100426/network_threat_detector