AI-Powered Adaptive Intrusion Detection and Response System (AI-ADRS)

Created by BlackPanda999 Team

July 24, 2025

1 Introduction

The AI-Powered Adaptive Intrusion Detection and Response System (AI-ADRS) is a state-of-the-art cybersecurity project that integrates artificial intelligence to detect, analyze, and respond to network intrusions in real-time. By combining machine learning, real-time packet analysis, automated threat mitigation, and cloud-based logging, AI-ADRS provides a robust solution for modern network security challenges. This document includes the project overview, implementation code, setup instructions, and future enhancements, making it ideal for sharing with your team.

2 Project Objectives

- Detect malicious network activities using AI-driven models (Random Forest and LSTM).
- Capture and analyze live network traffic in real-time using `scapy`.
- Automate threat responses, such as blocking malicious IPs.
- Log threat data to AWS S3 for persistent storage and analysis.
- Visualize threat patterns through an interactive web-based dashboard.

3 Technologies Used

- **Programming**: Python (`scapy`, `pandas`, `numpy`, `scikit-learn`, `tensorflow`, `boto3`, `plotly`, `dash`)
- **Dataset**: NSL-KDD (UNB CIC)
- **AI/ML**: Random Forest for robust classification, LSTM for sequential pattern detection
- **Visualization**: Matplotlib/Seaborn for static plots, Plotly/Dash for interactive dashboards
- **Cloud**: AWS S3 for threat logging

4 Setup Instructions

1. Install Python 3.8+ from python.org.
2. Install required libraries:

```
pip install pandas numpy scikit-learn scapy matplotlib seaborn
   tensorflow boto3 plotly dash
```

3. Download the NSL-KDD dataset from UNB CIC and save it as $\texttt{NSL}_K DD.csvintheprojectdirectory$
4. Run the Python script provided below to execute the system.

5 Code Implementation

The following Python code implements the AI-ADRS, including data processing, model training, real-time packet capture, automated response, cloud logging, and visualization.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import scapy.all as scapy
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import boto3
import plotly.express as px
from dash import Dash, dcc, html, Input, Output

# Load and preprocess NSL-KDD dataset
def load_data():
    data = pd.read_csv('NSL_KDD.csv')
    data = pd.get_dummies(data, columns=['protocol_type', 'service',
        'flag'])
    X = data.drop('label', axis=1)
    y = data['label'].apply(lambda x: 0 if x == 'normal' else 1)
    return X, y

# Preprocess data: scale features
def preprocess_data(X):
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled, scaler

# Train Random Forest model
def train_rf_model(X_train, y_train):
```

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

# Train LSTM model
def train_lstm_model(X_train, y_train, timesteps=1):
    X_train_reshaped = X_train.reshape((X_train.shape[0], timesteps,
        X_train.shape[1]))
    model = Sequential([
        LSTM(64, input_shape=(timesteps, X_train.shape[1]),
            return_sequences=False),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy',
        metrics=['accuracy'])
    model.fit(X_train_reshaped, y_train, epochs=10, batch_size=32,
        verbose=1)
    return model

# Real-time packet capture
def capture_packets(interface='eth0', count=10):
    packets = scapy.sniff(iface=interface, count=count)
    features = []
    for pkt in packets:
        if pkt.haslayer(scapy.IP):
            feature = [len(pkt), pkt[scapy.IP].ttl, pkt[scapy.IP].
                len]
            features.append(feature)
    return np.array(features)

# Automated response: block IP
def block_ip(ip_address):
    print(f"Blocking␣malicious␣IP:␣{ip_address}")
    # Example: Use iptables (Linux) or firewall rules (Windows)
    # os.system(f"iptables -A INPUT -s {ip_address} -j DROP")

# Cloud logging to AWS S3
def log_to_s3(data, bucket_name='ai-adrs-logs'):
    s3 = boto3.client('s3')
    s3.put_object(Bucket=bucket_name, Key='threat_log.txt', Body=str
        (data))

# Visualization: Confusion matrix
def plot_confusion_matrix(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d')
    plt.title('Confusion␣Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
```

```
    plt.show()

# Visualization: Interactive dashboard
def create_dashboard(X_test, y_test, y_pred):
    app = Dash(__name__)
    fig = px.scatter(x=X_test[:, 0], y=X_test[:, 1], color=y_pred,
       labels={'color': 'Prediction'})
    app.layout = html.Div([
        html.H1('AI-ADRS Threat Dashboard'),
        dcc.Graph(figure=fig)
    ])
    app.run_server(debug=True)

# Main function
def main():
    # Load and preprocess data
    X, y = load_data()
    X_train, X_test, y_train, y_test = train_test_split(X, y,
       test_size=0.3, random_state=42)
    X_train_scaled, scaler = preprocess_data(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Train and evaluate Random Forest
    print("Training Random Forest model...")
    rf_model = train_rf_model(X_train_scaled, y_train)
    y_pred = rf_model.predict(X_test_scaled)
    print("Random Forest Evaluation:")
    print(classification_report(y_test, y_pred))
    plot_confusion_matrix(y_test, y_pred)
    joblib.dump(rf_model, 'rf_model.pkl')
    joblib.dump(scaler, 'scaler.pkl')

    # Train and evaluate LSTM
    print("Training LSTM model...")
    lstm_model = train_lstm_model(X_train_scaled, y_train)
    X_test_reshaped = X_test_scaled.reshape((X_test_scaled.shape[0],
        1, X_test_scaled.shape[1]))
    y_pred_lstm = (lstm_model.predict(X_test_reshaped) > 0.5).astype
       (int)
    print("LSTM Evaluation:")
    print(classification_report(y_test, y_pred_lstm))

    # Real-time packet capture
    print("Capturing packets...")
    live_features = capture_packets()
    if live_features.size > 0:
        live_features_scaled = scaler.transform(live_features)
        live_pred = rf_model.predict(live_features_scaled)
        if 1 in live_pred:
            print("Potential threat detected in live traffic!")
            block_ip('example_malicious_ip')  # Replace with actual
```

```
                IP

    # Cloud logging
    log_to_s3({'predictions': y_pred_lstm.tolist(), 'timestamp': '
        2025-07-24'})

    # Interactive dashboard
    create_dashboard(X_test_scaled, y_test, y_pred)

if __name__ == "__main__":
    main()
```

6 How It Works

The AI-ADRS system operates as follows:

- **Data Processing**: Loads the NSL-KDD dataset, encodes categorical features, and scales numerical features for model compatibility.
- **Model Training**: Uses Random Forest for robust classification and LSTM for detecting sequential patterns in network traffic.
- **Real-Time Analysis**: Captures live packets using `scapy` and applies the trained model to detect threats.
- **Automated Response**: Blocks malicious IPs using firewall rules (e.g., iptables).
- **Cloud Integration**: Logs threat data to AWS S3 for persistent storage.
- **Visualization**: Generates static confusion matrices and an interactive web-based dashboard for threat analysis.

7 Future Enhancements

- Integrate quantum-resistant encryption for enhanced security.
- Implement zero-trust architecture for stricter access controls.
- Add real-time SMS/email alerts for immediate threat notifications.
- Extend to monitor IoT devices for broader network coverage.
- Incorporate federated learning for distributed threat detection.

8 Conclusion

The AI-ADRS project represents a cutting-edge fusion of AI and cybersecurity, offering real-time threat detection, automated responses, and advanced visualization. Its modular design and use of modern technologies make it a standout solution for network security. The included code and instructions enable your team to deploy and extend the system for real-world applications.

Thank you for exploring AI-ADRS! Let's secure the future together, Black-Panda999 Team!

Big Thank You!