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 cutting-edge cybersecurity solution that leverages artificial intelligence to protect networks from malicious activities. By integrating machine learning models (Random Forest and LSTM), real-time packet analysis with `scapy`, automated threat mitigation, cloud-based logging with AWS S3, and an interactive visualization dashboard using Plotly/-Dash, AI-ADRS offers a robust, scalable, and future-ready system. This document provides a detailed guide to understanding, setting up, and extending the project, making it ideal for your team to deploy in academic, enterprise, or research environments.

2 Project Objectives

The AI-ADRS aims to achieve the following:

- **Threat Detection**: Identify malicious network activities (e.g., DoS, probing, R2L, U2R attacks) using AI models trained on the NSL-KDD dataset.
- **Real-Time Monitoring**: Capture and analyze live network traffic using `scapy` for immediate threat detection.
- **Automated Response**: Block malicious IPs or trigger alerts to mitigate threats.
- **Cloud Integration**: Store threat logs in AWS S3 for persistent analysis and auditing.
- **Visualization**: Provide static (Matplotlib/Seaborn) and interactive (Plotly/Dash) visualizations to analyze threat patterns.

3 Use Cases

- **Enterprise Security**: Monitor corporate networks for intrusions and automate responses to minimize downtime.
- **Academic Research**: Study AI applications in cybersecurity using a real-world dataset and extensible code.

- **Portfolio Showcase**: Demonstrate expertise in AI, cybersecurity, and cloud technologies for professional or academic purposes.
- **IoT Security**: Extend the system to protect IoT networks by analyzing device traffic patterns.

4 Technologies Used

- **Programming**: Python 3.8+ with libraries: `scapy`, `pandas`, `numpy`, `scikit-learn`, `tensorflow`, `boto3`, `plotly`, `dash`.
- **Dataset**: NSL-KDD, a benchmark dataset for intrusion detection, available at UNB CIC.
- **AI/ML**: Random Forest for robust, tree-based classification; LSTM for sequential pattern detection in network traffic.
- **Visualization**: Matplotlib/Seaborn for static plots (e.g., confusion matrices); Plotly/Dash for interactive web-based dashboards.
- **Cloud**: AWS S3 for secure, persistent storage of threat logs.

5 Detailed Setup Instructions

This section provides step-by-step instructions to set up and run the AI-ADRS project, designed to be easy for all team members.

5.1 Step 1: Install Python

- Download and install Python 3.8 or higher from python.org.
- Verify installation:

```
python --version
```

- Ensure `pip` (Python package manager) is installed:

```
pip --version
```

5.2 Step 2: Set Up a Virtual Environment

- Create a virtual environment to isolate project dependencies:

```
python -m venv ai_adrs_env
```

- Activate the virtual environment:
  - Windows:

    ```
    ai_adrs_env\Scripts\activate
    ```

  - Linux/macOS:

```
source ai_adrs_env/bin/activate
```

- Confirm activation (you should see (ai$_a drs_e nv) in your terminal prompt).

5.3 Step 3: Install Dependencies

- Install required Python libraries:

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

- If errors occur, ensure `pip` is up-to-date:

```
pip install --upgrade pip
```

5.4 Step 4: Download the Dataset

- Download the NSL-KDD dataset from UNB CIC.

- Save the dataset file (e.g., `KDDTrain+.csv`) as NSL$_K DD.csv$ in your project directory. Alternatively, use as $NSL - KDD$ on $Kaggle$.

5.5 Step 5: Configure AWS S3 (Optional for Cloud Logging)

- Sign up for an AWS account at aws.amazon.com.
- Create an S3 bucket named `ai-adrs-logs` in the AWS Management Console.
- Set up AWS credentials:
  - Install the AWS CLI:

```
pip install awscli
```

  - Configure credentials:

```
aws configure
```

  - Enter your AWS Access Key ID, Secret Access Key, region (e.g., `us-east-1`), and output format (e.g., `json`).

5.6 Step 6: Prepare the Project Directory

- Create a project directory (e.g., AI$_A DRS$). $Place$ NSL$_K DD.csv$ and the Python script (ai$_a drs.py$, copi

• 5.7 Step 7: Run the Script

  - Run the Python script with elevated privileges for packet capture:

```
sudo python ai_adrs.py
```

- Replace eth0 in the capture$_p ackets function$ with your network interface ($find it using$ `ifconfig` on l

6 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. Save this as $ai_a drs.pyinyourprojectdirectory$.

```
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():
    try:
        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
    except FileNotFoundError:
        print("Error: NSL_KDD.csv not found. Please download
           from https://www.unb.ca/cic/datasets/nsl.html")
        exit(1)

# 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):
    try:
        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)
    except Exception as e:
        print(f"Error capturing packets: {e}")
        return np.array([])

# 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'):
    try:
        s3 = boto3.client('s3')
        s3.put_object(Bucket=bucket_name, Key='threat_log.txt',
           Body=str(data))
        print("Logged to S3 successfully")
    except Exception as e:
        print(f"Error logging to S3: {e}")

# 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)
    ])
    print("Starting dashboard at http://127.0.0.1:8050/")
    app.run_server(debug=True)

# Main function
def main():
    # Load and preprocess data
    print("Loading dataset...")
    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
    print("Logging to AWS S3...")
    log_to_s3({'predictions': y_pred_lstm.tolist(), 'timestamp':
        '2025-07-24'})

    # Interactive dashboard
    print("Launching interactive dashboard...")
    create_dashboard(X_test_scaled, y_test, y_pred)

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

7 How It Works

The AI-ADRS system operates through the following components:

- **Data Processing**: Loads the NSL-KDD dataset, applies one-hot encoding to categorical features (e.g., protocol type), and scales numerical features using StandardScaler for model compatibility.
- **Model Training**: Trains two models:
  - Random Forest: A tree-based classifier for robust, feature-based threat detection.
  - LSTM: A deep learning model for detecting sequential patterns in network traffic.
- **Real-Time Analysis**: Uses scapy to capture live packets, extracting features like packet length, TTL, and IP length for real-time threat detection.
- **Automated Response**: Identifies malicious IPs and simulates blocking them (e.g., via iptables on Linux).
- **Cloud Integration**: Logs threat predictions and timestamps to an AWS S3 bucket for auditing and analysis.
- **Visualization**: Generates:
  - Static confusion matrices to evaluate model performance.
  - An interactive Plotly/Dash dashboard to visualize threat patterns in a web browser.

8 Troubleshooting Common Issues

  - **FileNotFoundError for NSL$_K$DD.csv** :

– Ensure NSL$_K$DD.csvisintheprojectdirectory.Verifythefilenamematchesexactly(case−sensitive).

- **Permission Denied for Packet Capture**:

– Run the script with sudo (e.g., sudo python ai$_a$drs.py).Checkyournetworkinterfacenameusi

- **AWS S3 Errors**:
  - Verify AWS credentials are correctly configured (aws configure).
  - Ensure the S3 bucket ai-adrs-logs exists and is accessible.
- **Dependency Installation Failures**:
  - Update pip: pip install --upgrade pip.
  - Install libraries one by one to identify the problematic package.
- **Dashboard Not Loading**:
  - Access the dashboard at http://127.0.0.1:8050/ in a web browser.
  - Ensure no other process is using port 8050.

9 Future Enhancements

- **Quantum-Resistant Encryption**: Integrate post-quantum cryptographic algorithms to secure data transmission.
- **Zero-Trust Architecture**: Implement strict access controls for enhanced network security.
- **Real-Time Alerts**: Add SMS/email notifications using services like Twilio or Amazon SNS.
- **IoT Integration**: Extend packet capture to monitor IoT device traffic.
- **Federated Learning**: Enable distributed model training across multiple networks for privacy-preserving threat detection.

10 Conclusion

The AI-ADRS project combines AI and cybersecurity to deliver a powerful, scalable intrusion detection and response system. Its modular design, real-time capabilities, and advanced features like LSTM and cloud integration make it a standout solution for modern network security. The detailed setup instructions and troubleshooting tips ensure your team can easily deploy and extend the system. This project is ideal for enterprise security, academic research, or showcasing technical expertise.

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

Big Thank You!