GitHub:

https://github.com/harisharan-s

DETECTING CYBER THREADS THROUGH ANOMALY DETECTIONIN NETWORK TRAFFIC DATA

1. Problem Statement

Cyber security is increasingly critical in today's networked world. Traditional signature-based methods often fail to detect novel threats. This project aims to identify cyber threats using anomaly detection techniques on network traffic data. The focus is on predicting whether a session is an attack based on various behavioral and contextual features, transforming the problem into a binary classification task.

2. Abstract

The project applies machine learning to network traffic data to detect anomalies and identify potential cyber attacks. Using features such as login attempts, session duration, encryption type, and reputation scores, models including Logistic Regression, SVM, KNN, Decision Tree and Random Forest were trained and evaluated. The Random Forest model achieved an accuracy of 89.36%.

3. System Requirements

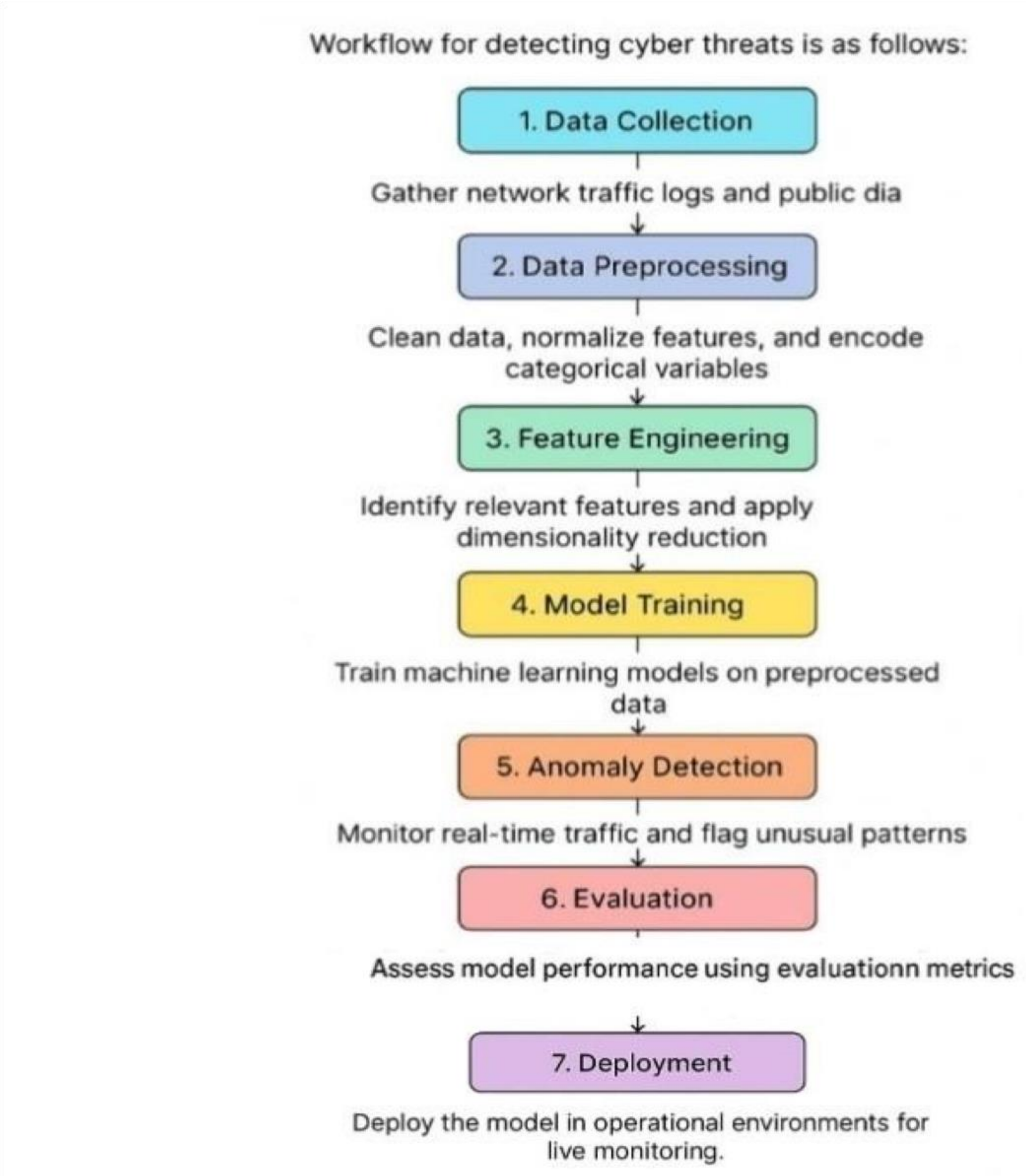
- Python 3.10+
- Libraries: pandas, numpy, matplotlib, seaborn, scikit-learn,Gradio.
- IDE: Google Colab or Jupyter Notebook
- Hardware: Minimum 4 GB RAM (8 GB recommended)

4. Objectives

The objective is to develop an accurate and interpretable ML model to classify sessions as normal or attack.

It also aims to deploy the model in a user-friendly interface to assist security teams in threat detection.

5. Flowchart of the Project Workflow



6. Dataset Description

- Rows: 9,537
- Columns: 11
- Target: attack_detected (0 or 1)
- Types: Mix of categorical and numeric
- Source: Provided dataset on network intrusion logs
- Key Features: login_attempts, session_duration, ip_reputation_score, protocol_type, encryption_used

7. Data Preprocessing

- Dropped session_id as non-informative
- One-hot encoded categorical columns
- Scaled numeric features using StandardScaler
- No missing values found

8. Exploratory Data Analysis (EDA)

- Checked distributions, correlations, and anomalies
- Observed that ip_reputation_score and failed_logins are important indicators for attack detection

9. Feature Engineering

- Created one-hot encoded features for protocol, encryption, and browser type
- Final feature set includes 20+ numerical and binary columns

-

10. Model Building

- Logistic Regression
- Random Forest Classifier
- SVM
- KNN
- Decision Tree

11. Model Evaluation

Random Forest:

- Accuracy: 89.36%
- Precision (Threat): 99.4%

Logistic Regression:

- Accuracy: 75.10%
- Precision (Threat): 75.5%

SVM

- Accuracy: 87.21%
- Precision: 94.02%

KNN

- Accuracy: 80.16%
- Precision: 85.37%

Decision Tree

- Accuracy: 83.51%
- Precision: 80.50%

12. Deployment

- Gradio Interface built for real-time threat detection

- Users input session attributes and receive predictions on attack presence
- Can be hosted locally or via Gradio's public share

13. source code

```
import pandas as pd
df = pd.read_csv('cybersecurity_intrusion_data.csv')
#Data Exploration
df.head()
df.info()
df.describe()
 df.shape
#Preprocessing
# Drop session_id
df = df.drop(columns=['session_id'])
 # One-hot encode categorical columns
 df = pd.get_dummies(df, drop_first=True)
# Check for missing values
print(df.isnull().sum())
# If any:
df.fillna(df.mean(), inplace=True)
Feature Scaling and Train-Test Split
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
X = df.drop('attack_detected', axis=1)
y = df['attack_detected']
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)

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

#Model Training (All Models)

# Logistic Regression

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(max_iter=1000)

lr.fit(X_train, y_train) lr_pred = lr.predict(X_test)

# Random Forest

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

rf.fit(X_train, y_train)

rf_pred = rf.predict(X_test)

# SVM

from sklearn.svm import SVC

svm = SVC()

svm.fit(X_train, y_train)

svm_pred = svm.predict(X_test)

# KNN

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

knn.fit(X_train, y_train)

knn_pred = knn.predict(X_test)

# Decision Tree

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

dt.fit(X_train, y_train)

dt_pred = dt.predict(X_test)
```

```
#Model Evaluation
from sklearn.metrics import accuracy_score, precision_score
print('Logistic Regression:', accuracy_score(y_test, lr_pred), precision_score(y_test, lr_pred))
print('Random Forest:', accuracy_score(y_test, rf_pred), precision_score(y_test, rf_pred))
print('SVM:', accuracy_score(y_test, svm_pred), precision_score(y_test, svm_pred))
print('KNN:', accuracy_score(y_test, knn_pred), precision_score(y_test, knn_pred))
print('Decision Tree:', accuracy_score(y_test, dt_pred), precision_score(y_test, dt_pred))
#Sample Prediction
sample = X_test[5].reshape(1, -1)
print('Prediction:', rf.predict(sample))
#Visualization: Class Distribution
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x='attack_detected', data=df)
plt.title('Attack vs Normal')
plt.show()
#Visualization: Correlation Heatmap
sns.heatmap(df.corr(), annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Feature Correlation')
plt.show()
#Visualization: Boxplots
sns.boxplot(x='attack_detected', y='ip_reputation_score', data=df)
plt.title('IP Reputation vs Attack')
plt.show()
sns.boxplot(x='attack_detected', y='failed_logins', data=df)
plt.title('Failed Logins vs Attack')
plt.show()
```

```
# Line plot for IP reputation scores of first 100 sessions
plt.plot(df['ip_reputation_score'][:100], label='IP Reputation Score')
plt.title('Line Plot of IP Reputation Score (First 100 Sessions)')
plt.xlabel('Session Index')
plt.ylabel('IP Reputation Score')
plt.legend()
plt.grid(True)
plt.show()
import seaborn as sns
import matplotlib.pyplot as plt

# KDE plot for session duration by attack class
sns.kdeplot(data=df, x='session_duration', hue='attack_detected', fill=True)
plt.title('KDE Plot of Session Duration by Attack Type')
plt.xlabel('Session Duration')
plt.ylabel('Density')
plt.show()

#Pair Plot
sns.pairplot(df[['network_packet_size', 'login_attempts', 'session_duration',
             'ip_reputation_score', 'failed_logins', 'attack_detected']],
          hue='attack_detected')
plt.suptitle('Pairplot of Selected Features', y=1.02)
plt.show()

import matplotlib.pyplot as plt
```

```
plt.hist(df['session_duration'], bins=30, color='skyblue', edgecolor='black')
plt.title('Histogram of Session Duration')
plt.xlabel('Session Duration')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

14. Future scope

- Integrate deep learning (LSTM for sequential patterns)
- Use streaming data for real-time updates
- Apply SHAP for explainability
- Collaborate with SOC teams for practical deployment

15.Team Members and Role

-Data preparation and cleaning: Adhithya.A

-Feature Engineering and EDA: Harisharan.P.S

-Anomaly Detection Modeling: Harish.K

-Results Interpretation and reporting: Prasanth.P