NETWORK ANOMALY DETECTION USING MACHINE LEARNING

ROHAN BABA SHAIK, SUMANA BOYAPALLI, HEMASAI SRI ALLA
Dept. Computer Science and Engineering
Florida Institute Of Technology
Prof. Abdullah Aydeger – aaydeger@fit.edu
Emails: rohanbaba2023@my.fit.edu, sboyapalli2023@my.fit.edu, halla2023@my.fit.edu

***Abstract*—This paper outlines a comprehensive approach to creating an advanced anomaly detection system employing machine learning (ML) techniques. By constructing two controlled virtual network environment consisting of client-server interactions, we simulate normal network traffic along with SYN flood attack using hping3 to create anomalies in network and capture the network through wireshark. Data collected from this environment serves as a basis for training and evaluating a range of machine learning models. Our methodology includes preprocessing steps to structure network data suitably for ML algorithms, feature engineering to highlight characteristics indicative of anomalous behavior, and the deployment of several supervised and unsupervised learning models to distinguish between normal and abnormal traffic patterns. To bridge the gap between theoretical ML models and practical applications, we integrate our anomaly detection system into a web framework, specifically Flask. This integration involves creating a Flask application to handle file uploads, parse data, and render model predictions. We design HTML templates to facilitate user interaction through an intuitive web interface. The backend is equipped to handle data parsing and preprocessing seamlessly, ensuring that the ML model receives well-structured input data. Upon loading and integrating the pre-trained anomaly detection model, the system efficiently processes the uploaded data and displays the prediction results on the web page, providing immediate feedback regarding network health.**

***Index Terms*—Virtual Machines, ML Algorithms, Anomalies, SYN flood Attack, Web Application, Flask, wireshark, hping3.**

I. Introduction to Network Anomaly Detection Project

Virtual Network Environment

Our project is designed to create a sophisticated network anomaly detection system that harnesses the power of machine learning to identify and flag irregularities in network traffic. We aim to accomplish this through the deployment of a virtual network environment that replicates the complexities of real-world network interactions in a secure and controlled setting.

Utilizing virtualization technologies, we construct a communication between a client machine, which pings server and with SYN flood attack to server and we capture the network through wireshark. This environment is not just a theoretical construct but a strategic platform that allows us to replicate and observe a client and server network behaviors, including those that deviate from the normal and potentially indicate cybersecurity threats.

Isolation Forest Machine learning Algorithm

In the realm of machine learning, we've chosen the isolation forest algorithm for its proficiency in identifying data points that deviate from the pattern. This model operates on the principle that anomalies are data points that are isolated when fewer conditions are required to separate them from the rest of the traffic. This characteristic makes the isolation forest particularly suitable for our project, as it enables us to detect anomalies in an efficient and effective manner, even in the vast and often chaotic datasets typical of network traffic.

As we apply this model to our virtual network, we are able to sift through complex data streams, highlighting unusual activity without the need for extensive labeling or prior definition of what constitutes normal traffic. This project, therefore, not only serves to improve network security but also advances the field of anomaly detection through practical, machine learning-driven solutions.

II. Problem Definition

The primary goal of this project is to develop an effective network anomaly detection system using machine learning techniques. The system should be capable of analyzing network traffic data and accurately identifying anomalous behavior, such as SYN flood attacks, port scans, and other potential threats. The system should also provide a user-friendly interface for real-time analysis and alerting.

III. Objectives

The key objectives of this project are:

1. Create a virtual network environment to simulate real-world network traffic scenarios.
2. Capture and preprocess network traffic data for analysis.
3. Develop a machine learning model for anomaly detection, specifically targeting SYN flood attacks and other network anomalies.
4. Integrate the anomaly detection model into a web application for user interaction and real-time analysis.
5. Evaluate the performance of the developed system and identify areas for improvement

IV. Scope

The scope of the project encompasses:

Virtual Environment Creation: Setting up a virtualized network

using industry-standard tools like VirtualBox or VMware to simulate client-server interactions.
Data Generation and Capture: Using network simulation tools like Wireshark to create a flow of network traffic that includes predefined normal and anomalous behaviors.
Machine Learning Implementation: Developing and training the isolation forest algorithm to identify network anomalies within the simulated data.
Interface Development: Creating a user-friendly interface, possibly a web application, to allow users to interact with the system and review anomalies.
Evaluation and Testing: Assessing the system's performance in detecting network anomalies against a dataset of known traffic patterns.
Project Limitations: While the project will provide a comprehensive framework for anomaly detection, it is bound by the constraints of simulated environments and may require further adaptation for live network applications.

V. System Requirements

Language: Python
Hardware: Modest laptop/desktop with 8GB RAM
Tools: VirtualBox/VMware, Wireshark, Python libraries (scikit-learn, pandas, numpy, matplotlib,Flask), Jupyter Notebook
Operating System: Ubuntu Server for VMs (lightweight Linux distribution)

VI. Literature Review

Network anomaly detection has been an active area of research in the field of cybersecurity, with various approaches and techniques proposed over the years. This section provides an overview of relevant literature and previous work in this domain.

A. Traditional Approaches

Traditional approaches to network anomaly detection have relied on rule-based systems and signature-based intrusion detection systems (IDS). Rule-based systems use predefined rules to identify known attack patterns or policy violations, while signature-based IDS systems compare network traffic against a database of known attack signatures.

However, these traditional approaches have several limitations:
- Inability to detect unknown or zero-day attacks, as they rely on predefined rules or signatures.
- Difficulty in keeping up with the ever-evolving nature of cyber threats and the constant emergence of new attack vectors.
- Reliance on manual rule or signature updates, which can be time-consuming and prone to human error.
- Difficulty in handling the complexities and high volumes of modern network traffic data.

B. Machine Learning Approaches

Machine learning techniques have emerged as a promising alternative to traditional approaches for network anomaly detection. These techniques leverage algorithms and statistical models to automatically learn patterns and anomalies from network traffic data, without relying on predefined rules or signatures.

Several machine learning algorithms and techniques have been explored in the context of network anomaly detection, including:
- Supervised learning algorithms, such as decision trees, random forests, and support vector machines (SVMs), which learn from labeled training data to classify network traffic as normal or anomalous.
- Unsupervised learning algorithms, such as k-means clustering, density-based spatial clustering of applications with noise (DBSCAN), and isolation forest, which can identify anomalies without labeled training data.
- Deep learning techniques, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which can automatically extract relevant features from raw network traffic data.
- Ensemble methods, which combine multiple machine learning models to improve overall performance and robustness.

C. Existing Systems and Tools

Several systems and tools have been developed for network anomaly detection using machine learning techniques. Some notable examples include:
- Kitsune: An open-source network anomaly detection system developed by Cisco, which uses an ensemble of online machine learning models to identify anomalies in real-time.
- OC-Kernel: An open-source library for anomaly detection based on one-class SVMs and other kernel methods.
- Zeek (formerly known as Bro): A powerful network security monitoring tool that can be extended with custom scripts and plugins for anomaly detection using machine learning techniques.

While these existing systems and tools provide valuable functionality, there is still room for improvement and exploration of new approaches tailored to specific network environments and use cases.

VII. Methodology

To achieve the objectives of our project, the following methodology will be adopted:

A. Virtual Network Environment Setup

A virtual network environment will be created using virtualization software (e.g., VirtualBox, VMware) to simulate a client-server network scenario. Two virtual machines (VMs) will be set up, one acting as the client and the other as the server. These VMs will run a lightweight Linux distribution,

such as Ubuntu Server, to facilitate network traffic generation and capture.

B. Network Traffic Generation

Both normal and anomalous network traffic patterns will be generated within the virtual network environment. Normal traffic will include HTTP requests and responses, simulating typical web browsing behavior. Anomalous traffic will include SYN flood attacks, port scans, and other known network anomalies. Tools such as hping3 and nmap will be utilized for generating anomalous traffic patterns.

C. Network Traffic Capture

The network traffic generated within the virtual environment will be captured using a network protocol analyzer, such as Wireshark. Captured traffic data will be exported to a suitable format (e.g., CSV, PCAP) for further analysis and preprocessing.

D. Data Preprocessing and Feature Engineering

The captured network traffic data will undergo preprocessing and feature engineering steps to prepare it for machine learning model training and evaluation. This may include handling missing data, encoding categorical features, normalization, and selecting or extracting relevant features that can effectively capture patterns and anomalies in the network traffic.

E. Machine Learning Model Development

Based on the preprocessed data and selected features, various machine learning algorithms and techniques will be explored for anomaly detection. This may include supervised learning algorithms (e.g., random forests, SVMs), unsupervised learning algorithms (e.g., isolation forest, clustering-based methods), and deep learning techniques (e.g., CNNs, RNNs). Model selection, hyperparameter tuning, and evaluation using appropriate metrics will be performed to identify the most suitable approach for the given network traffic data and anomaly detection requirements. For our project we choose Isolation model to detect anomalies.

F. Web Application Integration

The trained machine learning model for anomaly detection will be integrated into a web application using a suitable framework (e.g., Flask, Django), we used flask in our project. The web application will provide a user-friendly interface for uploading network traffic data, initiating the anomaly detection process, and displaying the analysis results. Users will be able to configure various parameters, such as anomaly detection thresholds, and receive alerts or notifications when anomalies are detected.

VIII. Implementation

The implementation phase will follow an iterative and incremental approach, allowing for continuous testing, evaluation, and refinement of the system. The following steps will be involved:

A. Virtual Network Environment Setup

Install and set up the two virtual network environment using virtualization software (e.g., VirtualBox, VMware) and configure the client and server VMs with the necessary network settings such as:

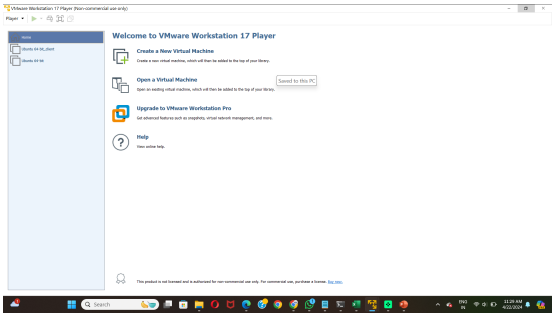

Fig. 1. VMWare for Client and Server

1) choose linux distribution for both Client and server this can be done by downloading Ubuntu ISO file from online and place in the place of operating system.
2) NAT as Network Adapter and restart the both machines.
3) Check in the terminal is the network is correctly configure or not.

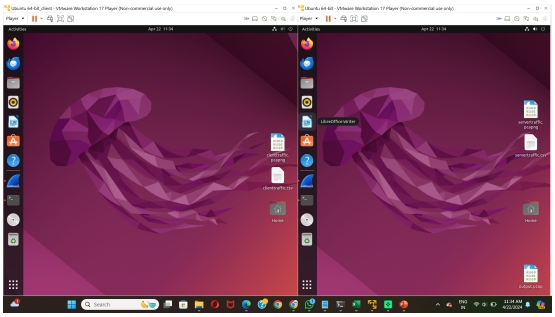

Fig. 2. Ubuntu operating system for Client and Server

4) Now we have install the wireshark and grant the permissions.
5) sudo apt-get update
6) sudo apt-get install wireshark
7) Granting permission for wireshark to capture —- sudo dpkg-reconfigure wireshark-common.
8) Check if wireshark is installed or not and also network in wireshark is correctly configure or not.

B. Network Traffic Generation and Capture

Before implementing scripts and tools for generating both normal and anomalous network traffic patterns within the virtual environment. We need to install some prerequisties:

1) For generate normal HTTP traffic from client to server sudo apt-get install Apache2
2) For simulate anomalous traffic (syn flood attacks , port scans) using tools like hping3 and nmap

3) sudo apt-get install hping3

4) sudo apt-get install curl
5) sudo apt-get install nmap
6) After installation we need to start wireshark for capturing data.
7) At the same time in terminal execute below commands.

8) Normal: ping server-ip,

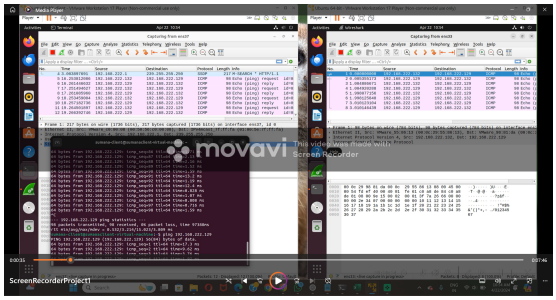

Fig. 3. Communication between client and server using curl

9) curl http://server-IP

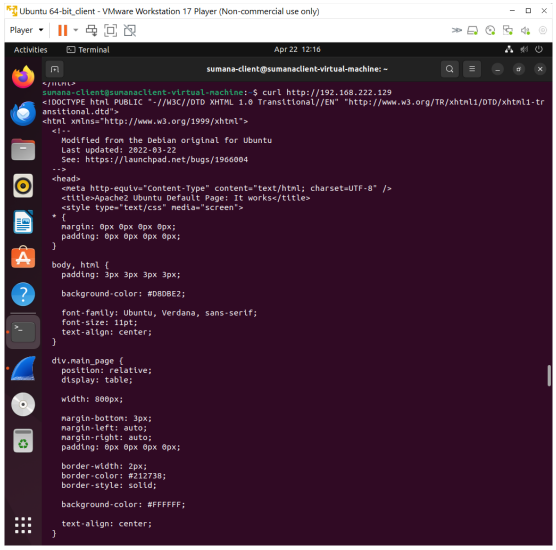

Fig. 4. Communication between client and server using curl

10) Anomalous traffic:
hping3 –S –flood –V –p 80 server-IP
11) Scanning network between client and server using nmap
nmap -p- -sT server-ip
12) Stop capturing and saving traffic data (pcap files), and export to CSV format.

C. Machine Learning Model Development

Explore and implement various machine learning algorithms and techniques for anomaly detection, such as random forests, isolation forests, clustering-based methods, and deep learning approaches. Utilize libraries like scikit-learn, TensorFlow, or

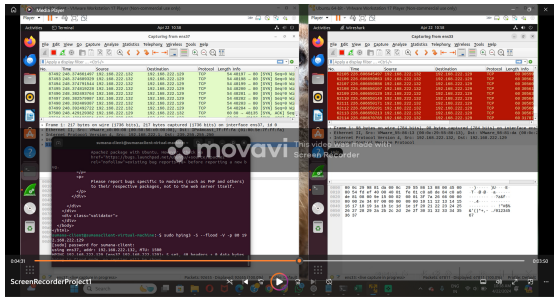

Fig. 5. Communication between client and server using hping3

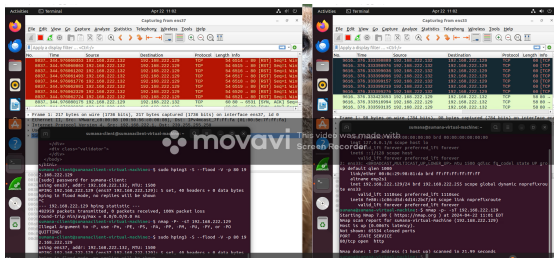

Fig. 6. Scanning network between client and server using nmap

PyTorch for model training, evaluation, and hyperparameter tuning.

1) Python Code for selecting Machine learning model:

```
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 IsolationForest
from sklearn.metrics import
 ↪  classification_report, confusion_matrix
import matplotlib.pyplot as plt
import joblib

# Load data
server_data =
pd.read_csv("C://Users//sumana//
Documents//CN//PROJECT//servertraffic.csv")
client_data =
pd.read_csv("C://Users//sumana//
Documents//CN//PROJECT//clienttraffic.csv")

# Combine data
combined_data = pd.concat([server_data,
 ↪  client_data], ignore_index=True)

# Define target based on 'Protocol'
combined_data['target_column'] =
 ↪  combined_data['Protocol'].apply(lambda x:
 ↪  0 if x in ['TCP', 'UDP'] else 1)

# Preserve original data for analysis
original_data = combined_data[['Source',
 ↪  'Destination', 'Protocol', 'Info']].copy()

# Prepare data for training
```

```
X = combined_data.drop(['Source', 'Destination', 'Protocol', 'Info', 'target_column'], axis=1)
y = combined_data['target_column']

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

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train model
model = IsolationForest(n_estimators=100, contamination=float(np.mean(y_train)))
model.fit(X_train_scaled)

# Predict anomalies
y_pred = model.predict(X_test_scaled)
y_pred = [1 if x == -1 else 0 for x in y_pred]  # Convert from Isolation Forest output to binary output

# Append original data to test set for detailed analysis
X_test.reset_index(drop=True, inplace=True)
original_data_test = original_data.loc[X_test.index]
original_data_test.reset_index(drop=True, inplace=True)
detailed_results = pd.concat([X_test, original_data_test], axis=1)
detailed_results['Anomaly'] = y_pred

# Filter and display anomalies
print("Anomalies Detected from the following file:")
print("clienttraffic.csv" if 'clienttraffic.csv' in detailed_results['Info'].values else "servertraffic.csv")
anomalies = detailed_results[detailed_results['Anomaly'] == 1]
for index, row in anomalies.iterrows():
    print("Anomalous Packet Detected:")
    print(f"Source: {row['Source']}, Destination: {row['Destination']}, Protocol: {row['Protocol']}, Info: {row['Info']}")
    # Additional logic can be added here to explain why a packet is considered anomalous

# Evaluate and save model
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
joblib.dump(model, 'isolation_forest_model.pkl')

# Plot anomaly scores
anomaly_scores = model.decision_function(X_test_scaled)
plt.hist(anomaly_scores, bins=50, density=True, alpha=0.6, color='g')
plt.xlabel('Anomaly Score')
plt.ylabel('Density')
plt.title('Distribution of Anomaly Scores')
plt.grid(True)
plt.show()
```

D. Data Preprocessing and Feature Engineering

We Develop Python scripts using Jupyter notebook for preprocessing the captured network traffic data. This may include handling missing data, encoding categorical features, normalization, and feature selection or extraction techniques.

1) Python Code for detecting anomalies::

```
import pandas as pd
from sklearn.ensemble import IsolationForest

# Load the CSV data
df = pd.read_csv("C://Users//sumana//Documents//CN/PROJECT//servertraffic.csv")

# Define a function to parse flags from Info column
def parse_syn_flag(info):
    # Assuming the info column contains strings where SYN packets are identifiable
    return int('SYN' in info and not 'ACK' in info)

# Apply function to create a new column for SYN flags
df['SYN_flag'] = df['Info'].apply(parse_syn_flag)

# Calculate the rate of SYN packets for each Source IP
syn_rate = df.groupby('Source')['SYN_flag'].mean().reset_index(

# Join the rate back to the original dataframe to get the syn_rate for each packet
df = df.join(syn_rate.set_index('Source'), on='Source')

# Since we're using unsupervised learning, there's no need for labels
# Let's fit an Isolation Forest model on the rate of SYN packets
model = IsolationForest(n_estimators=100, contamination=0.01, random_state=42)
model.fit(df[['syn_rate']])

# Predict anomalies
df['Anomaly'] = model.predict(df[['syn_rate']])

# Filter the anomalous packets
anomalous_packets = df[df['Anomaly'] == -1]

# Extract packet details for the anomalies
anomaly_details = anomalous_packets[['No.', 'Time', 'Source', 'Destination', 'Protocol', 'Length', 'Info']]
```

```

# Display the anomalous packet details
anomaly_details
# Flag as anomaly if the syn_rate exceeds a certain threshold, say 0.5
df['Anomaly'] = df['syn_rate'] > 0.5

# Now, instead of an empty dataframe, we should see sources with a syn_rate higher than 0.5
anomaly_details = df[df['Anomaly'] == True][['No.', 'Time', 'Source', 'Destination', 'Protocol', 'Length', 'Info']]
print(anomaly_details)
```

E. Web Application Development

For Developing the web application using a suitable framework (e.g., Flask, Django) for user interaction and integration with the anomaly detection model. We implementing user authentication, file upload functionality, data parsing, model integration, and result visualization components.

1) python Code for Web Application:: App.py

```
from flask import Flask, request, render_template
import pandas as pd

app = Flask(__name__)

def calculate_syn_rate(df):
    # Extract the 'SYN' and 'ACK' flags from the 'Info' column
    df['SYN_flag'] = df['Info'].str.contains('SYN') & ~df['Info'].str.contains('ACK')
    df['ACK_flag'] = df['Info'].str.contains('ACK')
    syn_rate_df = df.groupby('Source')['SYN_flag'].mean().reset_index(name='syn_rate')
    ack_rate_df = df.groupby('Source')['ACK_flag'].mean().reset_index(name='ack_rate')
    merged_df = df.merge(syn_rate_df, on='Source', how='left').merge(ack_rate_df, on='Source', how='left')
    return merged_df

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        server_ip = request.form['server_ip']
        client_ip = request.form['client_ip']
        file = request.files['file']
        df = pd.read_csv(file)

        # Calculate SYN and ACK rates
        df = calculate_syn_rate(df)

        # Flag as anomaly if the syn_rate exceeds a certain threshold, say 0.5
        df['Anomaly'] = df['syn_rate'] > 0.5

        # Filter for anomalies between the specified client and server IPs
        anomalies = df[(df['Anomaly']) & (df['Source'] == client_ip) & (df['Destination'] == server_ip)]

        # Prepare the data for display
        results = anomalies[['No.', 'Time', 'Source', 'Destination', 'Protocol', 'Length', 'Info']]

        return render_template('results.html', tables=[results.to_html(classes='data')], titles=['Anomalous Traffic'])

    return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=True)
```

index.html

```
<!doctype html>
<html>
<head><title>Upload File</title></head>
<body>
    <h2>Upload File</h2>
    <form method=post enctype=multipart/form-data>
        Server IP: <input type=text name=server_ip><br>
        Client IP: <input type=text name=client_ip><br>
        <input type=file name=file><br>
        <input type=submit value=Upload>
    </form>
</body>
</html>
```

Result.html

```
<!doctype html>
<html>
<head><title>Results</title></head>
<body>
    <h2>Anomaly Detection Results</h2>
    {% for table in tables %}
        <h3>{{titles[loop.index]}}</h3>
        {{ table|safe }}
    {% endfor %}
</body>
</html>
```

F. Testing, Evaluation and Results

Steps For execution:

1) Save the exported network files in PC and give the file path in python implementation correctly.
2) We are using Jupyter notebook as IDE, we need to first test the Python Code for selecting Machine learning model.

3) Then After selecting model, we execute the Python Code for detecting anomalies.
4) As for executing web application in PC, we create a folder called templates and place index.html and result.html files in templates folder and give files path in app.py correctly.
5) Now we need to execute the app.py for web application.

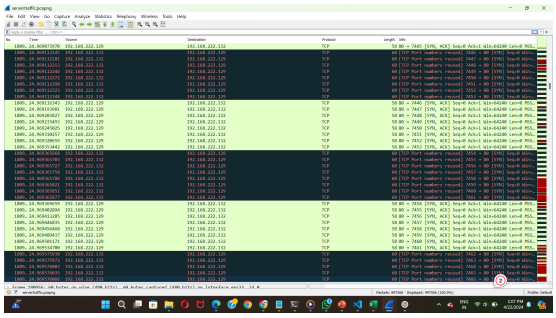

Fig. 7. SERVER TRAFFIC pcap file

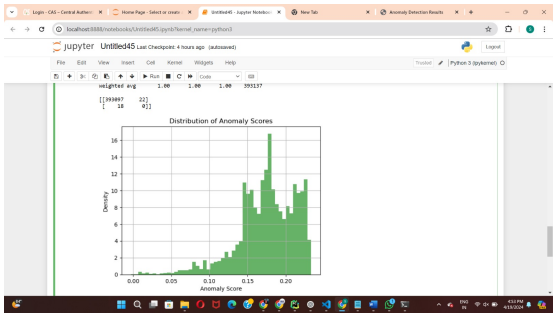

Fig. 8. Machine learning model performance– Isolation forest

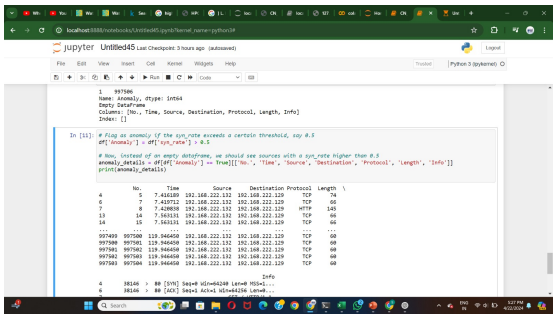

Fig. 9. Result of Anomalies packets detection

G. Future Work

While the developed network anomaly detection system aims to provide an effective solution for identifying anomalies in network traffic, there are several potential areas for future work and improvement:

1) Continuous Model Improvement: As new cyber threats and attack vectors emerge, it will be necessary to continuously update and retrain the machine learning models to ensure

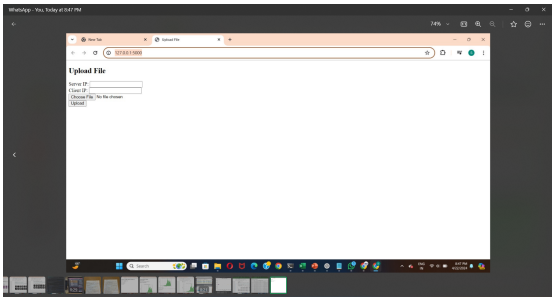

Fig. 10. Web Application for detecting anomalies

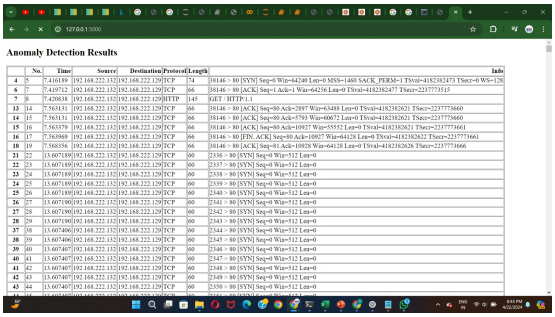

Fig. 11. Final output of detecting anomalies

accurate anomaly detection. This may involve exploring new algorithms, incorporating additional data sources, and leveraging transfer learning techniques.

2) Real-time Anomaly Detection: Implement real-time or near-real-time anomaly detection capabilities to enable prompt response and mitigation of potential threats. This may involve optimizing the system for high-performance computing environments and integrating with real-time data streaming platforms.

3) Explainable AI and Interpretability: Enhance the system's interpretability by incorporating techniques from explainable artificial intelligence (XAI) to provide insights into the decision-making process of the machine learning models. This can aid in understanding the root causes of detected anomalies and facilitate more effective incident response.

4) Automated Mitigation and Response: Explore the integration of automated mitigation and response mechanisms based on the detected anomalies. This may involve developing decision support systems or integrating with existing security information and event management (SIEM) tools.

5) Multi-tenant and Cloud-based Deployment: Extend the system's architecture to support multi-tenant deployments and cloud-based environments, enabling scalability and accessibility for organizations with distributed networks or cloud infrastructure.

6) Integration with Threat Intelligence and Cyber Threat Information Sharing: Investigate the integration of threat intelligence feeds and cyber threat information sharing platforms to enhance the system's ability to detect and respond to emerging threats and attack patterns.

H. Conclusion

The network anomaly detection system developed in this project leverages machine learning techniques to identify anomalous behavior in network traffic data, with a focus on detecting SYN flood attacks and other potential threats. By combining a virtual network environment, data preprocessing, feature engineering, and advanced machine learning algorithms, the system aims to provide an effective and user-friendly solution for enhancing network security.

Throughout the project, various challenges were encountered, including generating realistic network traffic patterns, selecting appropriate features, tuning machine learning models, and integrating the system with existing network infrastructure. However, valuable lessons were learned regarding the importance of data preprocessing, algorithm selection, and the need for continuous system updates and improvements.

While the developed system addresses the primary objectives, there is still room for further enhancements and exploration of new techniques. Future work may involve continuous model improvement, real-time anomaly detection, explainable AI, automated mitigation and response, multi-tenant and cloud-based deployments, and integration with threat intelligence and cyber threat information sharing platforms.

Overall, this project demonstrates the potential of machine learning in the domain of network anomaly detection and contributes to the ongoing efforts to enhance network security and protect against ever-evolving cyber threats.

REFERENCES

[1] Tan, Z., Jamdagni, A., He, X., Nanda, P., Liu, R. P. (2014). A system for denial-of-service attack detection based on multivariate correlation analysis. IEEE Transactions on Parallel and Distributed Systems, 25(2), 447-456.

[2] Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A. (2018). Kitsune: An ensemble of autoencoders for online network anomaly detection. In Proceedings of the Network and Distributed System Security Symposium (NDSS).

[3] Akhtar, N., Mian, A. (2018). Threat of adversarial attacks on deep learning in computer vision: A survey. IEEE Access, 6, 14410-14430.

[4] Angiulli, F., Pizzuti, C. (2002). Fast outlier detection in high dimensional spaces. In European Conference on Principles of Data Mining and Knowledge

[5] Sahay, S. K., Mehrotra, D., Moussa, A. (2021). Machine learning for network anomaly detection: A survey. arXiv preprint arXiv:2109.05528. [6] Marir, N., Wang, H., Feng, G., Li, B., Jia, M. (2018). Distributed abnormal behavior detection technique using deep belief network and ensemble SVM in Internet of Things. IEEE Internet of Things Journal, 6(2), 3597-3607. [7] Mirsky, Y., Doitshaman, T., Elovici, Y., Shabtai, A. (2018). Kitsune: An ensemble of autoencoders for online network traffic anomaly detection. arXiv preprint arXiv:1802.09089. [8] Ring, M., Wunderlich, S., Grüdl, D., Landes, D., Hotho, A. (2019). Flow-based network traffic anomaly detection. Machine Learning and Knowledge Extraction, 1(1), 15-40. [9] Niyaz, Q., Sun, W., Javaid, A. Y. (2016). A deep learning-based DDoS detection system in software-defined networking (SDN). arXiv preprint arXiv:1611.07400. [10] Sommer, R., Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In 2010 IEEE symposium on security and privacy (pp. 305-316). IEEE. [11] Yuan, X., Li, C., Li, X. (2017). DeepDefense: Identifying and detecting intelligent malicious traffic in mobile app networks. In 2017 IEEE Conference on Communications and Network Security (CNS) (pp. 1-9). IEEE.