Name: Manasa Jammalamadaka

PROJECT TITLE:
FIREWALL LOG ANALYZER WITH ANOMALY DETECTION

Firewalls are a critical component of network security, acting as the first line of defense against unauthorized access and cyber threats. However, simply having a firewall is not enough—continuous monitoring and analysis of firewall logs are essential to detect suspicious activities, prevent security breaches, and ensure compliance with security policies.

Firewall logs record all network traffic, capturing details like IP addresses, ports, protocols, and actions (allowed/blocked). Analyzing these logs is crucial for:

- **Detecting Unauthorized Access:** Identifies unusual or suspicious traffic patterns.
- **Preventing Cyber Threats:** Helps recognize port scanning, DDoS attacks, and brute-force attempts.
- **Compliance & Auditing:** Ensures adherence to security policies and regulatory requirements.
- **Incident Response:** Provides forensic evidence for investigating security breaches.

This project, Firewall Log Analyzer with Anomaly Detection, focuses on automating firewall log analysis using machine learning (Isolation Forest) to identify anomalies in network traffic. It processes firewall logs, detects potential threats, and sends real-time alerts via Telegram, enabling proactive security monitoring and threat detection.

STEPS FOR IMPLEMENTATION:

Step-1: Collect firewall logs

1.1 Identify the log sources

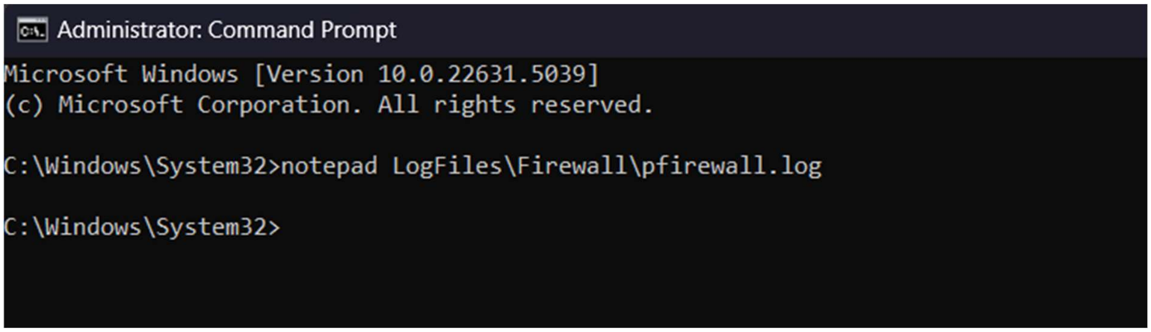
Log sources can be identified at various locations in different systems. Below mentioned is the path in Windows.

```
C:\Windows\System32\LogFiles\Firewall\pfirewall.log
```

1.2 Extract logs

- Open the command prompt
- Run it as administrator
- Use the below command to extract the log files.

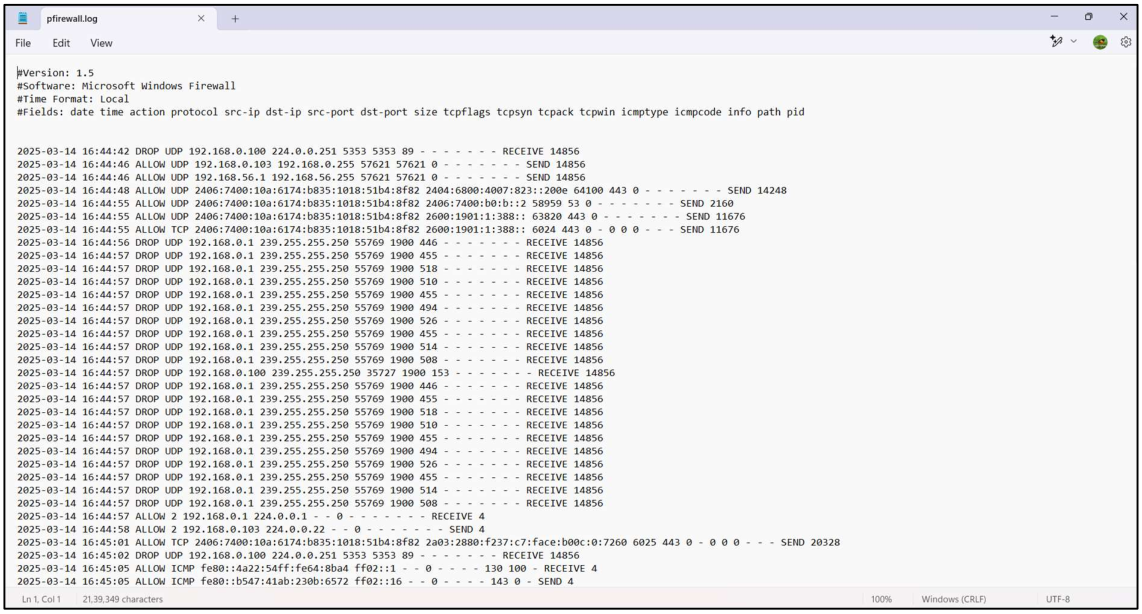
```
notepad C:\Windows\System32\LogFiles\Firewall\pfirewall.log
```



```
Microsoft Windows [Version 10.0.22631.5039]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>notepad LogFiles\Firewall\pfirewall.log

C:\Windows\System32>
```



```
#Version: 1.5
#Software: Microsoft Windows Firewall
#Time Format: Local
#Fields: date time action protocol src-ip dst-ip src-port dst-port size tcpflags tcpsyn tcpack tcpwin icmptype icmpcode info path pid


2025-03-14 16:44:42 DROP UDP 192.168.0.100 224.0.0.251 5353 5353 89 - - - - - - - RECEIVE 14856
2025-03-14 16:44:46 ALLOW UDP 192.168.0.103 192.168.0.255 57621 57621 0 - - - - - - - SEND 14856
2025-03-14 16:44:46 ALLOW UDP 192.168.56.1 192.168.56.255 57621 57621 0 - - - - - - - SEND 14856
2025-03-14 16:44:48 ALLOW UDP 2406:7400:10a:6174:b835:1018:51b4:8f82 2404:6800:4007:823::200e 64100 443 0 - - - - - - - SEND 14248
2025-03-14 16:44:55 ALLOW UDP 2406:7400:10a:6174:b835:1018:51b4:8f82 2406:7400:b0:b::2 58959 53 0 - - - - - - - SEND 2160
2025-03-14 16:44:55 ALLOW UDP 2406:7400:10a:6174:b835:1018:51b4:8f82 2600:1901:1:388:: 63820 443 0 - - - - - - - SEND 11676
2025-03-14 16:44:55 ALLOW TCP 2406:7400:10a:6174:b835:1018:51b4:8f82 2600:1901:1:388:: 6024 443 0 - 0 0 0 - - - SEND 11676
2025-03-14 16:44:56 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 446 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 455 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 518 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 510 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 455 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 494 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 526 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 455 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 514 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 508 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.100 239.255.255.250 35727 1900 153 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 446 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 455 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 518 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 510 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 455 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 494 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 526 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 455 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 514 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 DROP UDP 192.168.0.1 239.255.255.250 55769 1900 508 - - - - - - - RECEIVE 14856
2025-03-14 16:44:57 ALLOW 2 192.168.0.1 224.0.0.1 - - 0 - - - - - - - RECEIVE 4
2025-03-14 16:44:58 ALLOW 2 192.168.0.103 224.0.0.22 - - 0 - - - - - - - SEND 4
2025-03-14 16:45:01 ALLOW TCP 2406:7400:10a:6174:b835:1018:51b4:8f82 2a03:2880:f237:c7:face:b00c:0:7260 6025 443 0 - 0 0 0 - - - SEND 20328
2025-03-14 16:45:02 DROP UDP 192.168.0.100 224.0.0.251 5353 5353 89 - - - - - - - RECEIVE 14856
2025-03-14 16:45:05 ALLOW ICMP fe80::4a22:54ff:fe64:8ba4 ff02::1 - - 0 - - - - 130 100 - RECEIVE 4
2025-03-14 16:45:05 ALLOW ICMP fe80::b547:41ab:230b:6572 ff02::16 - - 0 - - - - 143 0 - SEND 4
```

Step-2: Parse and preprocess logs
2.1 Extract Relevant Fields Using Regex
Create a Python script to extract fields like:

1. Source IP
2. Destination IP
3. Protocol
4. Source Port
5. Destination Port
6. Timestamp

Note: All the libraries will be installed through command prompt.

Before writing the script, ensure that Python Package Installer (pip) is updated.
Use the following command to upgrade pip:

```
python.exe -m pip install --upgrade pip
```

Command to install pandas:

```
python -m pip install pandas
```

Script Code:

```
import time
import re
import pandas as pd

log_file_path = r"C:\Windows\System32\LogFiles\Firewall\pfirewall.log"

pattern = r"(?P<timestamp>[\d-
]+\s[\d:]+)\s(?P<action>\w+)\s(?P<protocol>\w+)\s(?P<source_ip>[\d\.:a-fA-
F]+)\s(?P<destination_ip>[\d\.:a-fA-
F]+)\s(?P<source_port>\d+)\s(?P<destination_port>\d+)"

# Function to read all logs
```

```
def read_firewall_logs():
    with open(log_file_path, "r") as file:
        logs = file.readlines()

    log_data = []
    for log_entry in logs:
        match = re.search(pattern, log_entry)
        if match:
            log_data.append(match.groupdict())

    return pd.DataFrame(log_data) if log_data else pd.DataFrame()
```

Step 3: Store Logs in a Database
3.1 Store extracted logs in CSV, SQLite, or MongoDB for easy access.
Here, logs are stored in CSV.

Code:

```
df.to_csv("firewall_logs.csv", mode='a', header=False, index=False)
```

Step 4: Anomaly Detection Using Machine Learning
4.1 Train an Unsupervised Model
Here, Isolation Forest is used to detect anomalies in traffic.
Features used:

1. Source IP (src_ip)
2. Destination IP (dst_ip)
3. Protocol (proto)
4. Source Port (src_port)
5. Destination Port (dst_port)
6. Timestamp (timestamp)

Command to install scikit-learn:

```
python -m pip install scikit-learn
```

Code:

```
# Read all logs
df = read_firewall_logs()

if not df.empty:
    # Convert categorical data to numeric for Isolation Forest
    df['protocol'] = df['protocol'].astype('category').cat.codes
    df['source_port'] = df['source_port'].astype(int)
    df['destination_port'] = df['destination_port'].astype(int)

    # Train Isolation Forest
    model = IsolationForest(contamination=0.05, random_state=42)
    model.fit(df[['protocol', 'source_port', 'destination_port']])

    df.to_csv("firewall_anomalies.csv", index=False) #Save results

    print("Log analysis completed! Anomalies saved in
'firewall_anomalies.csv'")
```

Step 5: Real-time Anomaly Detection & Alerts
5.1 Monitor Live Logs & Detect Threats
Create a script that continuously checks logs and triggers alerts when anomalies appear.

Code:

```
import time
import pandas as pd
def monitor_logs():
    print("Monitoring logs for anomalies...")
    while True:
        try:
            df = pd.read_csv("firewall_anomalies.csv")  # Read latest logs
            time.sleep(10)  # Check logs every 10 seconds
        except Exception as e:
            print(f"Error in monitoring logs: {e}")
            time.sleep(10)  # Wait before retrying

monitor_logs()  # Start monitoring after defining everything
```

Step 6: Send Alerts
6.1 Telegram Bot for Alerting
For giving alerts, we need to create a telegram bot which alerts us when an anomaly takes place.

Code:

```
import requests
# Function to send alerts
def send_alert(message):
    bot_token = "<telegram bot token"
    chat_id = "<chat id>"
    url = f"https://api.telegram.org/bot{bot_token}/sendMessage"
    data = {"chat_id": chat_id, "text": message}
    requests.post(url, data=data)
send_alert("Firewall Alert! Suspicious activity detected.")
```

For the telegram bot token:

- Open telegram and create a bot using botfather.
- Start the chat with **/newbot**
- Give a name and username for your bot.
- A long string will be generated which is to be used as bot token

For chat id:

- Open telegram and search for userinfobot.
- Start the chat by typing **/start**
- It will give the Chat ID of our telegram chat.

Step 7: Build a Web Dashboard to Display Logs & Anomalies
7.1 Create a Flask Dashboard

Code:

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

app = Flask(__name__)
```

```
def parse_firewall_log(filepath):
    """Reads `pfirewall.log`, extracts relevant fields, and marks
anomalies."""
    with open(filepath, "r") as file:
        lines = file.readlines()

    data = []
    headers = None

    for line in lines:
        line = line.strip()

        if not line or line.startswith("#"):
            continue  # Skip empty lines or comments

        if headers is None:
            headers = line.split(",")  # Extract column names
            headers.append("Anomaly")  # Add a new column for anomalies
        else:
            values = line.split(",")
            if len(values) == len(headers) - 1:
                row = dict(zip(headers[:-1], values))

                if row.get("Event") == "Blocked":
                    row["Anomaly"] = "Yes"  # Mark as an anomaly
                else:
                    row["Anomaly"] = "No"   # Mark as normal

                data.append(row)  # Append row to data

    return data

@app.route('/')
def home():
    filepath = "firewall_anomalies.csv"

    try:
        data = parse_firewall_log(filepath)  # Parse log file
        print("DEBUG DATA:", data[:5])  # Print first 5 rows for debugging
    except Exception as e:
        data = [{"Error": f"Failed to read log file: {e}"}]  # Show error
in HTML

    return render_template("index.html", data=data)

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

For index.html to display in the dashboard, make sure you create the index file in the same folder as Dashboard.

Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Firewall Anomalies Data</title>
</head>
```

```
<body>
    <h2>Firewall Anomalies Data</h2>

    {% if data and "Error" in data[0] %}
        <p style="color: red;">{{ data[0]["Error"] }}</p>
    {% else %}
        <table border="1">
            <thead>
                <tr>
                    {% for col in data[0].keys() %}
                        <th>{{ col }}</th>
                    {% endfor %}
                </tr>
            </thead>
            <tbody>
                {% for row in data %}
                    <tr>
                        {% for value in row.values() %}
                            <td>{{ value }}</td>
                        {% endfor %}
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    {% endif %}

    <h3>Debugging Output:</h3>
    <pre>{{ data }}</pre>
</body>
</html>
```

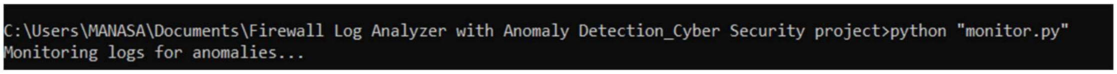
RESULT

After creating python files for Dashboard and Firewall log analyzer:

- Open the command prompt and run it as Administrator.
- Navigate to the folder in which dashboard and firewall log analyzer files are present through command prompt.

  cd path\of\the\folder
- To monitor the anomalies, run the following command in a command prompt:

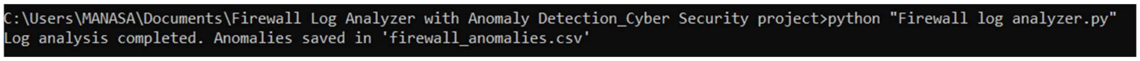
  python “monitor.py”



```
C:\Users\MANASA\Documents\Firewall Log Analyzer with Anomaly Detection_Cyber Security project>python "monitor.py"
Monitoring logs for anomalies...
```

- To save the anomalies in a .csv file, run the following:

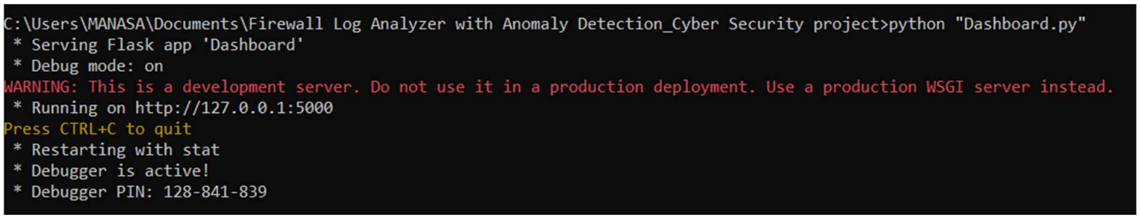
  python “Firewall log analyzer.py”



```
C:\Users\MANASA\Documents\Firewall Log Analyzer with Anomaly Detection_Cyber Security project>python "Firewall log analyzer.py"
Log analysis completed. Anomalies saved in 'firewall_anomalies.csv'
```

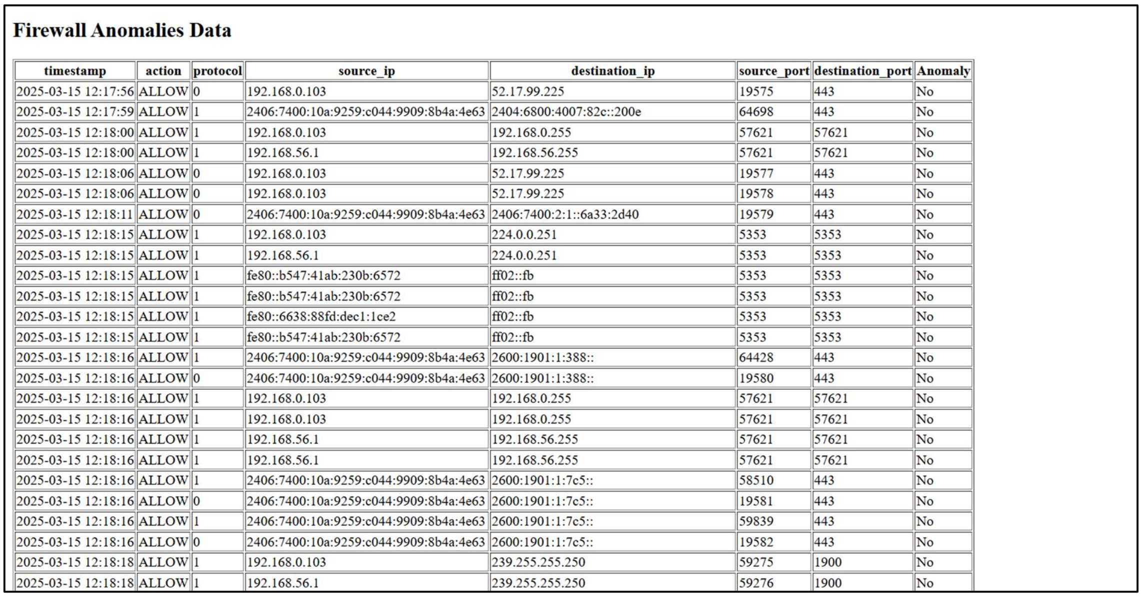
- For displaying the logs as a website, enter the following command:

  python “Dashboard.py”
- After running this command, the details of the dashboard will appear.



```
C:\Users\MANASA\Documents\Firewall Log Analyzer with Anomaly Detection_Cyber Security project>python "Dashboard.py"
 * Serving Flask app 'Dashboard'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 128-841-839
```

- Upon clicking on the link which appears, the dashboard looks like this:



Firewall Anomalies Data

timestamp	action	protocol	source_ip	destination_ip	source_port	destination_port	Anomaly
2025-03-15 12:17:56	ALLOW	0	192.168.0.103	52.17.99.225	19575	443	No
2025-03-15 12:17:59	ALLOW	1	2406:7400:10a:9259:c044:9909:8b4a:4e63	2404:6800:4007:82c::200e	64698	443	No
2025-03-15 12:18:00	ALLOW	1	192.168.0.103	192.168.0.255	57621	57621	No
2025-03-15 12:18:00	ALLOW	1	192.168.56.1	192.168.56.255	57621	57621	No
2025-03-15 12:18:06	ALLOW	0	192.168.0.103	52.17.99.225	19577	443	No
2025-03-15 12:18:06	ALLOW	0	192.168.0.103	52.17.99.225	19578	443	No
2025-03-15 12:18:11	ALLOW	0	2406:7400:10a:9259:c044:9909:8b4a:4e63	2406:7400:2:1::6a33:2d40	19579	443	No
2025-03-15 12:18:15	ALLOW	1	192.168.0.103	224.0.0.251	5353	5353	No
2025-03-15 12:18:15	ALLOW	1	192.168.56.1	224.0.0.251	5353	5353	No
2025-03-15 12:18:15	ALLOW	1	fe80::b547:41ab:230b:6572	ff02::fb	5353	5353	No
2025-03-15 12:18:15	ALLOW	1	fe80::b547:41ab:230b:6572	ff02::fb	5353	5353	No
2025-03-15 12:18:15	ALLOW	1	fe80::6638:88fd:dec1:1ce2	ff02::fb	5353	5353	No
2025-03-15 12:18:15	ALLOW	1	fe80::b547:41ab:230b:6572	ff02::fb	5353	5353	No
2025-03-15 12:18:16	ALLOW	1	2406:7400:10a:9259:c044:9909:8b4a:4e63	2600:1901:1:388::	64428	443	No
2025-03-15 12:18:16	ALLOW	0	2406:7400:10a:9259:c044:9909:8b4a:4e63	2600:1901:1:388::	19580	443	No
2025-03-15 12:18:16	ALLOW	1	192.168.0.103	192.168.0.255	57621	57621	No
2025-03-15 12:18:16	ALLOW	1	192.168.0.103	192.168.0.255	57621	57621	No
2025-03-15 12:18:16	ALLOW	1	192.168.56.1	192.168.56.255	57621	57621	No
2025-03-15 12:18:16	ALLOW	1	192.168.56.1	192.168.56.255	57621	57621	No
2025-03-15 12:18:16	ALLOW	1	2406:7400:10a:9259:c044:9909:8b4a:4e63	2600:1901:1:7c5::	58510	443	No
2025-03-15 12:18:16	ALLOW	0	2406:7400:10a:9259:c044:9909:8b4a:4e63	2600:1901:1:7c5::	19581	443	No
2025-03-15 12:18:16	ALLOW	1	2406:7400:10a:9259:c044:9909:8b4a:4e63	2600:1901:1:7c5::	59839	443	No
2025-03-15 12:18:16	ALLOW	0	2406:7400:10a:9259:c044:9909:8b4a:4e63	2600:1901:1:7c5::	19582	443	No
2025-03-15 12:18:18	ALLOW	1	192.168.0.103	239.255.255.250	59275	1900	No
2025-03-15 12:18:18	ALLOW	1	192.168.56.1	239.255.255.250	59276	1900	No

CHALLENGES & SOLUTIONS

Challenge: High Volume of Logs

- Solution: Used regex-based log parsing and Pandas for efficient data processing.

Challenge: False Positives in Anomaly Detection

- Solution: Fine-tuned Isolation Forest contamination parameter to reduce noise.

Challenge: Real-Time Monitoring

- Solution: Implemented periodic log checks and automated Telegram alerts for immediate threat notification.

Challenge: Log Format Variability

- Solution: Developed flexible parsing patterns to handle different firewall log structures.

CONCLUSION

This project successfully automated firewall log analysis and anomaly detection using machine learning. By leveraging Isolation Forest and real-time alerting, it enhances network security and enables proactive threat mitigation. Future improvements can include dashboard visualization, integration with SIEM tools, and advanced deep learning models for even more accurate anomaly detection.