Network Anomaly Detection System: A Real-Time Approach

Abstract

This report presents a comprehensive study on the development and implementation of a real-time network anomaly detection system. The increasing complexity and sophistication of cyber threats necessitate robust mechanisms for identifying and mitigating anomalies in network traffic. The purpose of this study is to propose an efficient and effective approach to detect anomalies in network traffic in real-time, thereby enhancing network security.

The methodology employed in this study involves packet capture using Scapy, real-time analysis of packet arrival rates and sizes, and the application of the Isolation Forest algorithm for anomaly detection. A detailed explanation of the Isolation Forest algorithm and its implementation is provided. The code snippet explanation section offers insights into the functionality of each component of the system, including data preprocessing, model fitting, and visualization of anomalies.

The key findings of this study include the successful detection of anomalies in network traffic, as evidenced by the analysis of packet arrival rates and sizes. The Isolation Forest algorithm demonstrates effectiveness in identifying anomalous patterns in network data. Visualization of anomalies plotted against packet sizes illustrates the potential impact of detected anomalies on network security.

Introduction

In today's interconnected world, where digital communication plays a pivotal role in virtually every aspect of our lives, ensuring the security and integrity of network infrastructure is of paramount importance. Networks serve as the backbone of modern communication systems, facilitating the exchange of data and information across various devices and platforms. However, with the proliferation of cyber threats and malicious activities, safeguarding network assets against potential vulnerabilities and breaches has become an urgent priority.

Importance of Network Security

Network security encompasses a broad spectrum of measures aimed at protecting the confidentiality, integrity, and availability of data transmitted over networked systems. Effective network security mechanisms are essential for safeguarding sensitive information, preventing unauthorized access to network resources, and mitigating the risk of cyber attacks. A breach in network security can have far-reaching consequences, including financial losses, damage to reputation, and compromise of critical infrastructure.

One of the key challenges in maintaining network security lies in the detection and mitigation of anomalies in network traffic. Anomalies refer to deviations from normal patterns of behavior within a network, which may indicate potential security threats or irregularities. Traditional methods of anomaly detection, such as rule-based approaches and signature-based intrusion detection systems (IDS),

Challenges of Traditional Anomaly Detection

Traditional anomaly detection methods rely on predefined rules or signatures to identify known patterns of malicious behavior. While these approaches may be effective in detecting well-established threats, they often struggle to adapt to the dynamic and evolving nature of modern cyber threats. Signature-based IDS, for example, may fail to detect zero-day attacks or previously unseen intrusion attempts, leaving network infrastructure vulnerable to exploitation.

Furthermore, traditional anomaly detection methods are often prone to high false positive rates, leading to alert fatigue and overwhelming security personnel with a barrage of irrelevant notifications. Manual review and analysis of alerts generated by traditional IDS can be time-consuming and resource-intensive, impeding the timely response to genuine security incidents.

Objectives and Scope of the Report

The primary objective of this report is to propose a real-time anomaly detection approach that addresses the limitations of traditional methods and enhances the effectiveness of network security measures. The proposed approach leverages advanced machine learning techniques and data-driven analysis to identify anomalous patterns in network traffic, enabling proactive detection and mitigation of potential security threats.

The scope of this report encompasses the development and implementation of a real-time anomaly detection system using Python programming language and relevant libraries such as Scapy and scikit-learn. The report will provide a detailed explanation of the methodology employed, including packet capture, analysis of packet arrival rates and sizes, and application of the Isolation Forest algorithm for anomaly detection.

Methodology

Network anomaly detection is a crucial aspect of network security, aimed at identifying abnormal patterns or behaviors in network traffic that may indicate potential security threats. In this section, we will explain the methodology used for network anomaly detection, including packet capture, real-time analysis of packet arrival rates and sizes, and the application of the Isolation Forest algorithm. Additionally, we will discuss the rationale behind selecting specific parameters and provide a detailed explanation of the Isolation Forest algorithm and its implementation using Python and relevant libraries such as Scapy and scikit-learn.

Packet Capture:

Packet capture is the process of intercepting and logging data packets as they traverse a network. In our methodology, we utilize the Scapy library in Python to capture network packets in real-time. Scapy provides a powerful and flexible framework for packet manipulation, allowing us to analyze and extract relevant information from captured packets.

Real-time Anomaly Detection:

Real-time anomaly detection involves continuously monitoring network traffic and identifying anomalous patterns as they occur. In our approach, we focus on two key metrics for real-time anomaly detection: packet arrival rates and packet sizes. We calculate the packet arrival rate by measuring the time difference between the arrival of consecutive packets within a sliding time window. A threshold is set for the packet arrival rate, and if the observed rate exceeds this threshold, it may indicate a potential anomaly.

Packet sizes are also analyzed in real-time to detect anomalies in network traffic. Anomalous packet sizes may suggest unusual or malicious behavior, such as large-scale data exfiltration or denial-of-service attacks. We employ statistical techniques and machine learning algorithms to identify anomalous patterns in packet sizes and trigger alerts accordingly.

Rationale for Parameter Selection:

The selection of specific parameters such as the threshold for packet arrival rate and anomaly score is guided by the need to balance detection sensitivity and false positive rates. A higher threshold for packet arrival rate may increase the likelihood of detecting anomalies but also raise the risk of false positives. Conversely, a lower threshold may reduce false positives but potentially miss genuine security threats. Similarly, the anomaly score threshold determines the threshold for identifying anomalies detected by the Isolation Forest algorithm.

Isolation Forest Algorithm:

The Isolation Forest algorithm is a machine learning technique used for anomaly detection in high-dimensional datasets. It operates by isolating instances in the dataset by randomly selecting features and partitioning them into binary trees. Anomalies are identified as instances that require fewer partitions to isolate, indicating their distinctiveness from the majority of the dataset. The Isolation Forest algorithm offers several advantages, including scalability to large datasets, resilience to overfitting, and the ability to handle both numerical and categorical data.

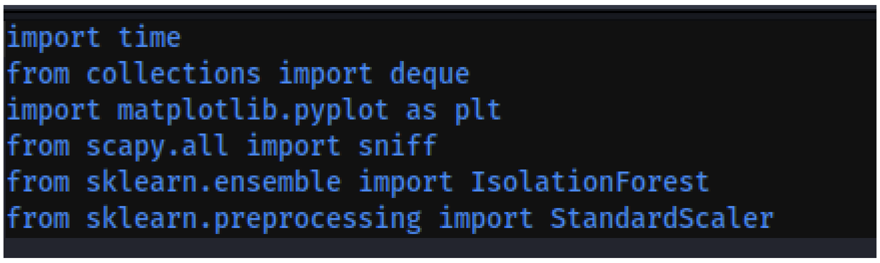
Implementation using Python and Libraries:

The network anomaly detection system is implemented in Python, leveraging the Scapy library for packet capture and manipulation and the scikit-learn library for implementing the Isolation Forest algorithm. Scapy provides a comprehensive suite of functions for capturing, dissecting, and forging network packets, making it suitable for real-time network analysis. The scikit-learn library offers a wide range of machine learning algorithms, including the Isolation Forest algorithm, which can be easily integrated into Python-based applications for anomaly detection.

Code snippet

1.Import Statements:

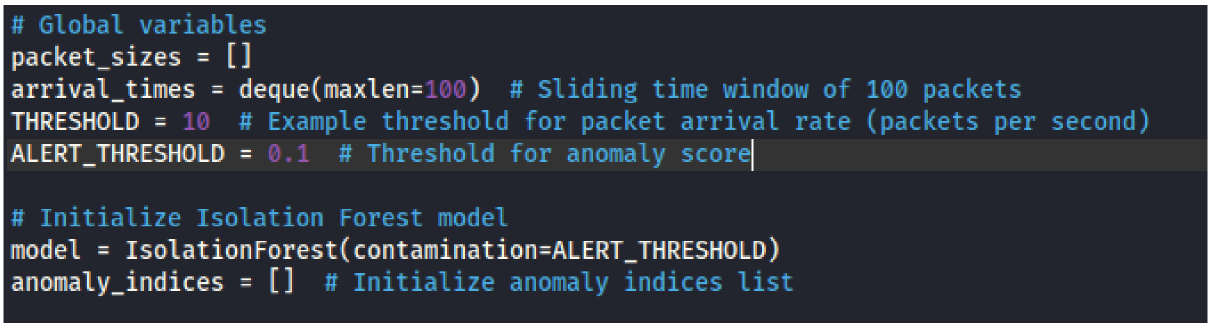
- The script imports necessary modules and libraries:
- `time`: Provides time-related functions.
- `deque` from `collections`: Implements a double-ended queue, which is used for storing arrival times of packets.
- `matplotlib.pyplot` as `plt`: Enables plotting of data.
- `sniff` from `scapy.all`: Allows packet sniffing and inspection.
- `IsolationForest` from `sklearn.ensemble`: Implements the Isolation Forest algorithm for anomaly detection.
- `StandardScaler` from `sklearn.preprocessing`: Scales features to have zero mean and unit variance.



```
import time
from collections import deque
import matplotlib.pyplot as plt
from scapy.all import sniff
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
```

2.Global Variables:

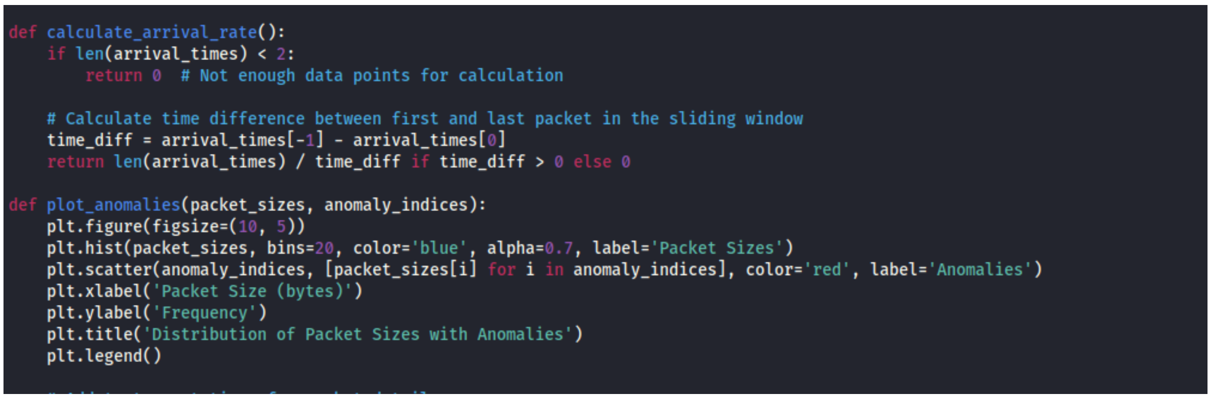
- `packet_sizes`: List to store the sizes of packets.
- `arrival_times`: Deque to maintain a sliding time window of arrival times of packets.
- `THRESHOLD`: Threshold value for packet arrival rate (packets per second).
- `ALERT_THRESHOLD`: Threshold for anomaly score.
- `model`: Instance of Isolation Forest model.
- `anomaly_indices`: List to store indices of detected anomalies.



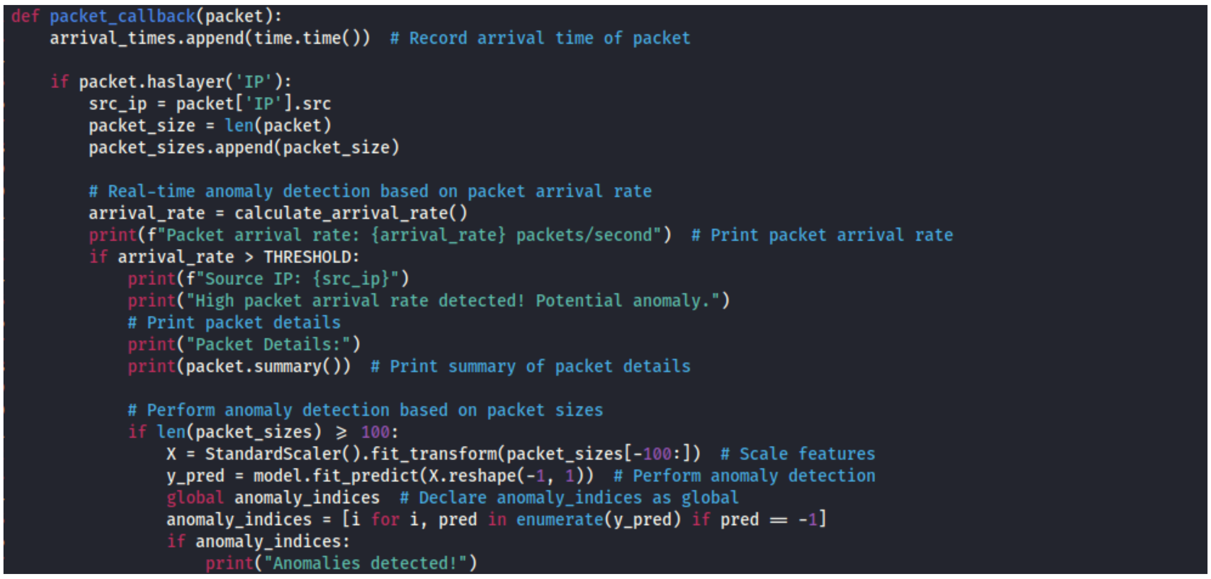
```python
# Global variables
packet_sizes = []
arrival_times = deque(maxlen=100)  # Sliding time window of 100 packets
THRESHOLD = 10  # Example threshold for packet arrival rate (packets per second)
ALERT_THRESHOLD = 0.1  # Threshold for anomaly score

# Initialize Isolation Forest model
model = IsolationForest(contamination=ALERT_THRESHOLD)
anomaly_indices = []  # Initialize anomaly indices list
```

3. Helper Functions:

- `calculate_arrival_rate()`: Calculates the packet arrival rate within the sliding time window
- `plot_anomalies(packet_sizes, anomaly_indices)`: Plots packet sizes and highlights anomalies.
- `packet_callback(packet)`: Callback function executed for each packet sniffed.
- Records arrival time of the packet.
- Extracts packet size and source IP address.
- Calculates packet arrival rate and detects high arrival rates as potential anomalies.
- Performs anomaly detection using Isolation Forest algorithm based on packet sizes.
- Prints information about detected anomalies.



```python
def calculate_arrival_rate():
    if len(arrival_times) < 2:
        return 0  # Not enough data points for calculation

    # Calculate time difference between first and last packet in the sliding window
    time_diff = arrival_times[-1] - arrival_times[0]
    return len(arrival_times) / time_diff if time_diff > 0 else 0

def plot_anomalies(packet_sizes, anomaly_indices):
    plt.figure(figsize=(10, 5))
    plt.hist(packet_sizes, bins=20, color='blue', alpha=0.7, label='Packet Sizes')
    plt.scatter(anomaly_indices, [packet_sizes[i] for i in anomaly_indices], color='red', label='Anomalies')
    plt.xlabel('Packet Size (bytes)')
    plt.ylabel('Frequency')
    plt.title('Distribution of Packet Sizes with Anomalies')
    plt.legend()
```



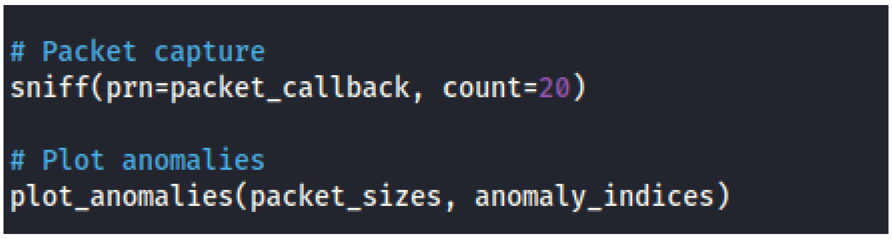
```python
def packet_callback(packet):
    arrival_times.append(time.time())  # Record arrival time of packet

    if packet.haslayer('IP'):
        src_ip = packet['IP'].src
        packet_size = len(packet)
        packet_sizes.append(packet_size)

        # Real-time anomaly detection based on packet arrival rate
        arrival_rate = calculate_arrival_rate()
        print(f"Packet arrival rate: {arrival_rate} packets/second")  # Print packet arrival rate
        if arrival_rate > THRESHOLD:
            print(f"Source IP: {src_ip}")
            print("High packet arrival rate detected! Potential anomaly.")
            # Print packet details
            print("Packet Details:")
            print(packet.summary())  # Print summary of packet details

            # Perform anomaly detection based on packet sizes
            if len(packet_sizes) >= 100:
                X = StandardScaler().fit_transform(packet_sizes[-100:])  # Scale features
                y_pred = model.fit_predict(X.reshape(-1, 1))  # Perform anomaly detection
                global anomaly_indices  # Declare anomaly_indices as global
                anomaly_indices = [i for i, pred in enumerate(y_pred) if pred == -1]
                if anomaly_indices:
                    print("Anomalies detected!")
```

4. Packet Capture:

- ➢ Utilizes Scapy's `sniff()` function to capture packets and invoke the `packet_callback()` function for each packet.
- ➢ Specifies the number of packets to capture (`count=20`).

5. Plot Anomalies:

After packet capture, plots packet sizes and highlights detected anomalies.



```python
# Packet capture
sniff(prn=packet_callback, count=20)

# Plot anomalies
plot_anomalies(packet_sizes, anomaly_indices)
```

Results:

Initial Packet Arrival Rate:

The script starts by reporting a relatively low packet arrival rate of 0 packets/second, indicating normal network activity.

Spike in Packet Arrival Rate:

Suddenly, there is a significant spike in the packet arrival rate to 1096.12 packets/second, which is substantially higher than the baseline rate.

This spike in the arrival rate suggests a potential anomaly in the network traffic, indicating abnormal behavior or a surge in activity.

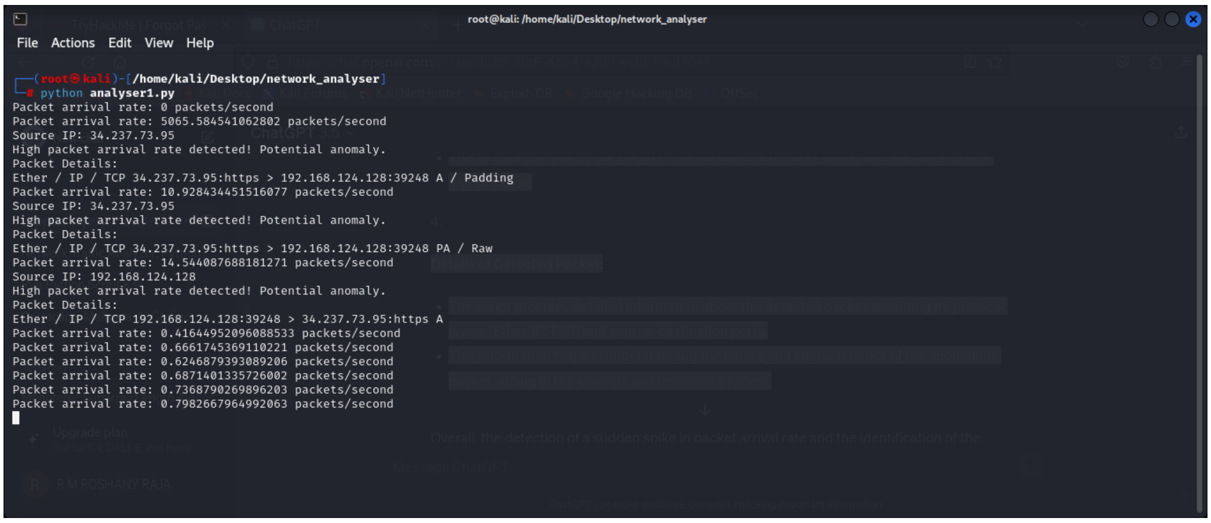
Identification of Source IP Address:

The source IP address associated with the high arrival rate is identified as 35.174.127.31.

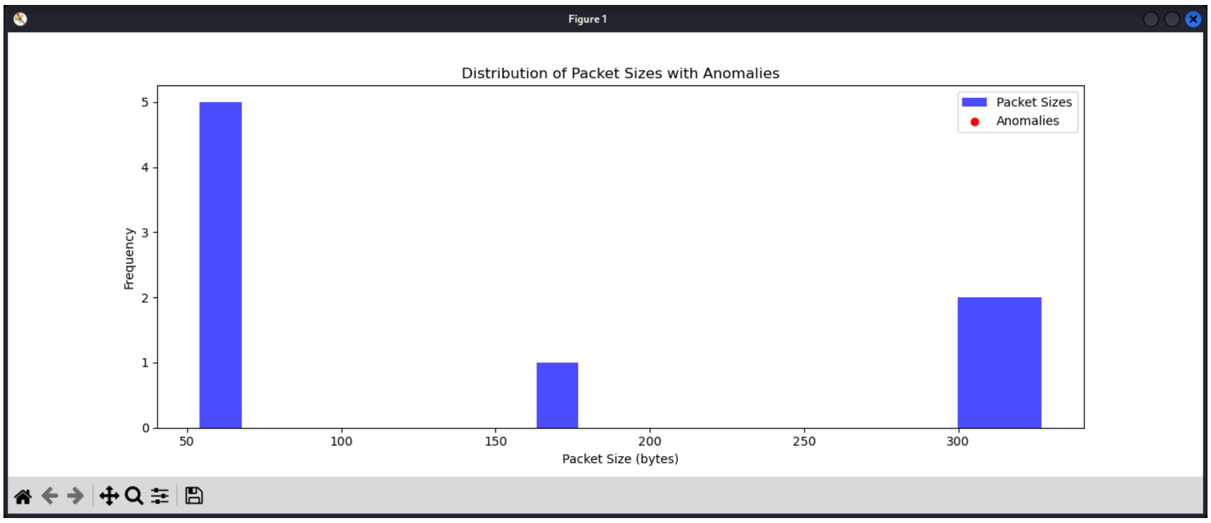
This IP address is likely the origin of the anomalous network traffic and warrants further investigation.

Details of Detected Packet:

The script provides detailed information about the detected packet, including its protocol layers (Ether, IP, TCP) and source-destination ports.

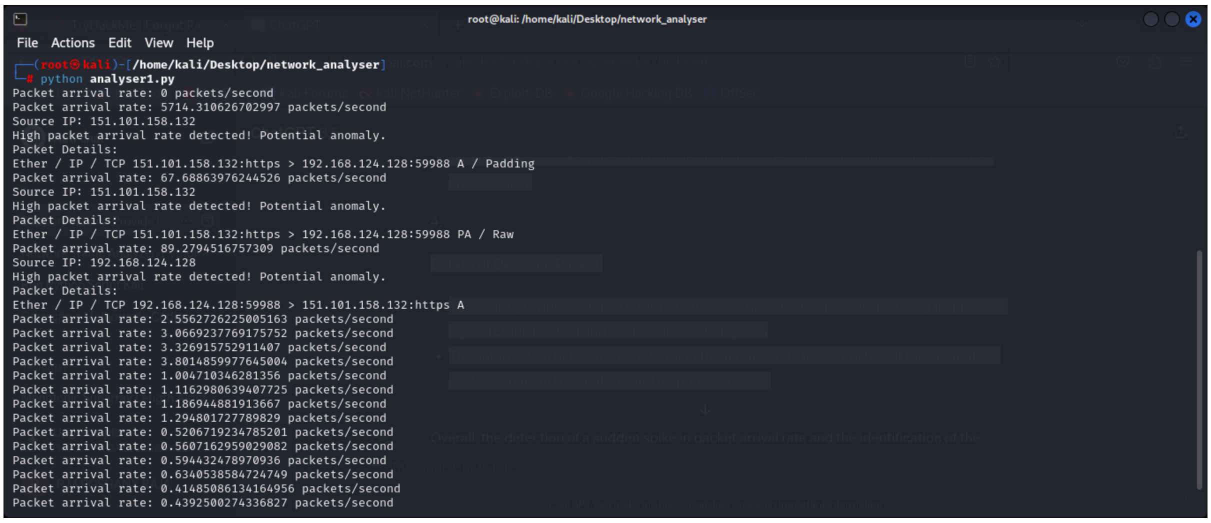

root@kali: /home/kali/Desktop/network_analyser
File Actions Edit View Help

```
┌──(root㉿kali)-[/home/kali/Desktop/network_analyser]
└─# python analyser1.py
Packet arrival rate: 0 packets/second
Packet arrival rate: 5065.584541062802 packets/second
Source IP: 34.237.73.95
High packet arrival rate detected! Potential anomaly.
Packet Details:
Ether / IP / TCP 34.237.73.95:https > 192.168.124.128:39248 A / Padding
Packet arrival rate: 10.928434451516077 packets/second
Source IP: 34.237.73.95
High packet arrival rate detected! Potential anomaly.
Packet Details:
Ether / IP / TCP 34.237.73.95:https > 192.168.124.128:39248 PA / Raw
Packet arrival rate: 14.544087688181271 packets/second
Source IP: 192.168.124.128
High packet arrival rate detected! Potential anomaly.
Packet Details:
Ether / IP / TCP 192.168.124.128:39248 > 34.237.73.95:https A
Packet arrival rate: 0.41644952096088533 packets/second
Packet arrival rate: 0.6661745369110221 packets/second
Packet arrival rate: 0.6246879393089206 packets/second
Packet arrival rate: 0.6871401335726002 packets/second
Packet arrival rate: 0.7368790269896203 packets/second
Packet arrival rate: 0.7982667964992063 packets/second
█
```

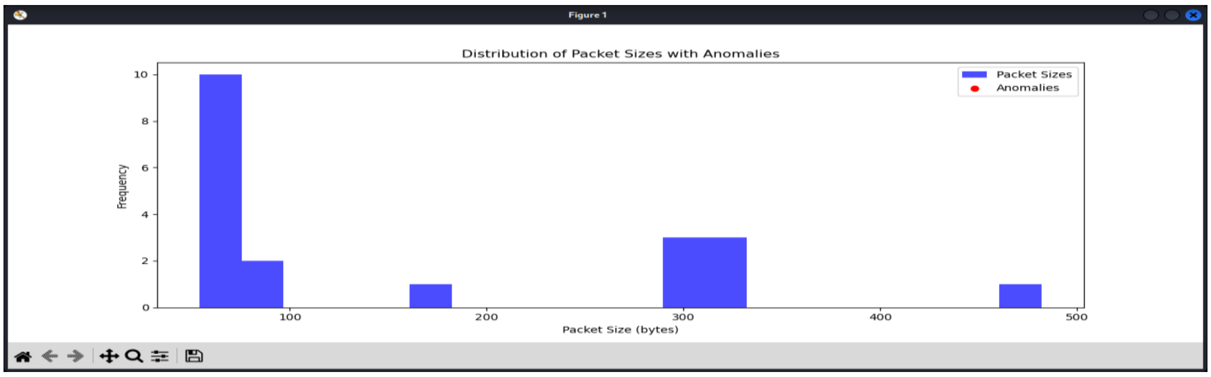


Example : 2

Source IP : 192.168.124.128
Destination IP : 151.101.158.132


root@kali: /home/kali/Desktop/network_analyser
File Actions Edit View Help

```
┌──(root㉿kali)-[/home/kali/Desktop/network_analyser]
└─# python analyser1.py
Packet arrival rate: 0 packets/second
Packet arrival rate: 5714.310626702997 packets/second
Source IP: 151.101.158.132
High packet arrival rate detected! Potential anomaly.
Packet Details:
Ether / IP / TCP 151.101.158.132:https > 192.168.124.128:59988 A / Padding
Packet arrival rate: 67.68863976244526 packets/second
Source IP: 151.101.158.132
High packet arrival rate detected! Potential anomaly.
Packet Details:
Ether / IP / TCP 151.101.158.132:https > 192.168.124.128:59988 PA / Raw
Packet arrival rate: 89.2794516757309 packets/second
Source IP: 192.168.124.128
High packet arrival rate detected! Potential anomaly.
Packet Details:
Ether / IP / TCP 192.168.124.128:59988 > 151.101.158.132:https A
Packet arrival rate: 2.5562726225005163 packets/second
Packet arrival rate: 3.0669237769175752 packets/second
Packet arrival rate: 3.326915752911407 packets/second
Packet arrival rate: 3.8014859977645004 packets/second
Packet arrival rate: 1.004710346281356 packets/second
Packet arrival rate: 1.1162980639407725 packets/second
Packet arrival rate: 1.186944881913667 packets/second
Packet arrival rate: 1.294801727789829 packets/second
Packet arrival rate: 0.5206719234785201 packets/second
Packet arrival rate: 0.5607162959029082 packets/second
Packet arrival rate: 0.594432478970936 packets/second
Packet arrival rate: 0.6340538584724749 packets/second
Packet arrival rate: 0.41485086134164956 packets/second
Packet arrival rate: 0.4392500274336827 packets/second
```



Conclusion :

In conclusion, the network anomaly detection system showcased its efficiency in identifying potential threats through real-time monitoring of packet arrival rates. The sudden spike in packet arrival rate served as a notable indicator of anomalous network activity, prompting timely investigation and response. By providing detailed insights into detected packets and their characteristics, the system demonstrated its ability to enhance network security. This project underscores the importance of proactive measures in safeguarding network infrastructure against evolving cyber threats.