CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

This project is about Network Performance Monitoring and Intrusion Detection using Snort. The Network Performance Monitoring Dashboard aims to provide a solution for real-time monitoring of latency, bandwidth (upload Speed and download Speed), CPU utilization, and Network utilization. This project is implemented using the Python programming language (High-level language) along with Tkinter for Graphical User Interface Development, psutil for CPU monitoring, speedtest-cli for network performance calculation, and matplotlib for generating interactive graphs.

Network Performance Monitoring serves a crucial role in maintaining the health and efficiency of the network. We can monitor the hardware elements like routers, switches, firewalls, servers, and local machines. Users can understand and analyze this real-time network monitoring.

This project not only focuses on real-time network performance monitoring but also network intrusion detection using a tool called Snort. Snort follows a rule-based detection mechanism that can trigger alerts for different types of attacks performed on a network such as ARP anomaly detection, DNS spoofing detection, ICMP flood attack, and other attacks. Snort is a network-based tool. It uses preprocessor and dynamic engines to detect attacks in a network. Whenever any attack is detected that attack-related alert is triggered and logged into files. The file name is alert. ids which contain information about the type of attack, timestamp, sender IP address, sender MAC address, attacker IP address, and packet content.

1.2 OBJECTIVES

1. Real-time Network Monitoring: It is about continuous monitoring of various network metrics such as latency (time taken for a data packet to travel from its source and its destination), bandwidth (upload Speed and download Speed), Network utilization (Number of bytes sent and received), and CPU utilization. They are visualized using speedometers for latency, bandwidth, and interactive graphs for Network utilization and CPU utilization.

2. Security Alerting Mechanism: Implementing an alerting mechanism to detect network attacks and trigger alerts corresponding to the attacks using an Intrusion Detection tool named Snort. It is a Network-based tool. It is used to detect network attacks and trigger corresponding alerts. Those alerts are displayed on the screen in the form of a message box with a warning notation. This message contains the timestamp of the attack and the attacker's IP address.

1.3 METHODOLOGY

There are six steps:

1. Data Acquisition
2. Data Preprocessing
3. Data Analysis
4. Metrics Computation
5. Metrics Visualization
6. Dashboard Display

1. Data Acquisition: It is about information gathering. The data is of two types one is Network data and the other is CPU data.

Network Data: It is gathered using Python modules like speedtest-cli to measure parameters such as latency (time taken for a data packet to travel from its source and its destination) and bandwidth. These measurements are done continuously as it is real-time network performance monitoring.

CPU Data: It is gathered using Python modules like psutil to monitor CPU utilization and other relevant metrics. And also Network utilization can be measured using this psutil module. It is a ratio of number of bytes sent and received.

2. Data Preprocessing:

Cleaning: Handling missing values, and ensuring data consistency.

Normalization: Scaling the data to a standard range to facilitate comparison between network metrics.

3. Data Analysis:

Calculating the statistics to understand the distribution and characteristics of data.

Identifying the relationships between different metrics.

4. Metrics Computation: There are two metrics one is Network metrics and the other is CPU metrics.

Network metrics: Metrics such as latency, upload Speed, and download Speed (bandwidth). These metrics can be obtained from the speedtest-cli module which pings to the best server and gets the results. It takes some time to respond as it needs to find the best server.

CPU metrics: It contains CPU utilization percentage. It is gathered from the psutil module. There is one more metric Network utilization that is also gathered from the same module.

5. Metrics Visualization:

Speedometers:
To visualize the latency, upload speed, and download speed these speedometers are helpful. These meters can be obtained from the Python module ttkbootsrap which contains a variety of meters with different styles. These meters run continuously for every 5 seconds as it is a real-time network monitoring.

Graphs:
To visualize the CPU utilization, and Network utilization these graphs are useful. These graphs are obtained from the Python module matplotlib which contains a variety of graphs with different styles. As this is real-time monitoring these graphs run continuously for every 5 seconds.

Users can easily understand these visualizations as they are interactive. They run so smoothly without disturbing the other. With the help of the Python multithreading module, they run independently. These are daemon threads. These are called for every 5 seconds.

6. Dashboard Display:

User Interface: With the help of the Python module Tkinter we can create a Graphical User Interface. We can place the objects anywhere we want.

Real-time Updates: The dashboard updates dynamically for every 5 seconds. It contains three speedometers for latency, upload speed, and download speed, and two graphs for CPU utilization and network utilization. An alert message is displayed whenever any network attack is detected using snort. The alert message contains a timestamp and the attacker's IP address. These tasks run independently of each other.

CHAPTER 2
LITERATURE REVIEW

Security and performance monitoring over networks is an aspect of modern information network infrastructure management. This provides a review of existing literature on network security and network performance.

Network security is a safeguard against unauthorized access, and attacks. Studies such as "An Overview of Network Security Techniques" by John Doe et al. (2018) provide a survey of common network security techniques.

Performance monitoring includes latency, bandwidth, CPU utilization, and Network utilization. Studies such as "Performance Metrics and Monitoring Tools for Modern Networks" by Jane Smith et al. (2019) explore various network performance metrics and the tools available for their computation and analysis.

The importance of real-time monitoring, and intrusion detection. "Integrating Security and Performance Monitoring for Enhanced Network Management" by Alice Johnson et al. (2020) presents a framework for integrating security and performance monitoring tools.

CHAPTER 3
IMPLEMENTATION

3.1 ALGORITHM

The Network data such as latency, upload speed, and download speed are gathered from the Python module speedtest-cli which pings to the best server and gets the results. Later we preprocess the data to compute the metrics and visualize them using speedometers on the dashboard. The CPU data and Network utilization data are gathered from the Python module psutil. They are visualized using interactive graphs. The dashboard also displays alert messages whenever any network attack is detected using snort. This alert message contains a timestamp and the attacker's IP address. The dashboard is updated dynamically every 5 seconds.

ADVANTAGES

- Early Threat Detection: Continuous monitoring (real-time monitoring) allows for the early detection of security threats such as network attacks.
- Improved Network Performance: Network performance metrics such as latency, bandwidth, CPU utilization, and Network utilization lead to optimized network performance.
- Enhanced Resource Utilization: By monitoring CPU utilization and Network utilization, we can optimize resource allocation, and identify underutilized resources.
- Real-time Reporting and Analysis: Security and performance monitoring tools provide real-time reporting and analysis functionalities, allowing network administrators to monitor network health, and identify threats actively.

DISADVANTAGES

- Complexity and Resource Intensiveness: Implementing security and performance monitoring can be complex and resource-intensive, requiring minimum hardware, and software configurations.
- Privacy Concerns: Continuous monitoring (real-time monitoring) of network performance metrics and CPU utilization activities raises privacy concerns, as it may involve capturing and analyzing sensitive data.
- Resources: Internet is mandatory without it we cannot monitor the network performance metrics.

APPLICATIONS

- Intrusion Detection and Prevention Systems
- Network Performance Management
- Traffic Analysis and Forensics
- Application Performance Monitoring

3.2 SOFTWARE REQUIREMENT SPECIFICATION

In the field of software development, software Requirement Specification is an essential document. It outlines everything that the software should be and be able to accomplish, kind of like a blueprint. The purpose of this official document is to facilitate communication among Several project stakeholders such as clients, developers, and testers.

HARDWARE DESCRIPTION

- RAM: 4.0 GB
- CPU: Core i3 processor (Minimum)
- Hard Disk Space: 128 GB

SOFTWARE DESCRIPTION

- Python
- Tkinter
- Snort

3.2.1 TECHNOLOGY DESCRIPTION

PYTHON

Python is the high-level programming language for this complete project, it provides a flexible and robust environment for developing network monitoring and management applications. Python's simplicity, readability, and extensive standard library make it well-suited for a wide range of applications, including GUI development, system monitoring, data processing, and network communication.

Tkinter and ttkbootstrap

Tkinter is a standard GUI (Graphical User Interface) toolkit for Python, allowing you to create interactive GUI-based applications. ttkbootstrap is a library that provides Bootstrap-themed widgets for Tkinter, enhancing the visual appearance of GUI. Together, Tkinter and ttkbootstrap are used to design and implement the graphical user interface of network monitoring applications, including meters, graphs, message boxes, and other visual elements.

psutil

psutil is a Python library that provides facilities for retrieving system and process-related information. It is used to monitor system metrics such as CPU utilization, memory usage, disk I/O, and network connections in your application. psutil enables to collection of real-time data on CPU utilization, which is then displayed graphically in the CPU utilization graph (real-time updation).

Matplotlib

Matplotlib is a plotting module/library for Python that provides a wide variety of static, animated, and interactive visualizations. It is used to create the application's CPU utilization graph and network utilization graph. Matplotlib allows us to plot and customize data in graphical formats, providing insights into system and network performance over time.

Snort

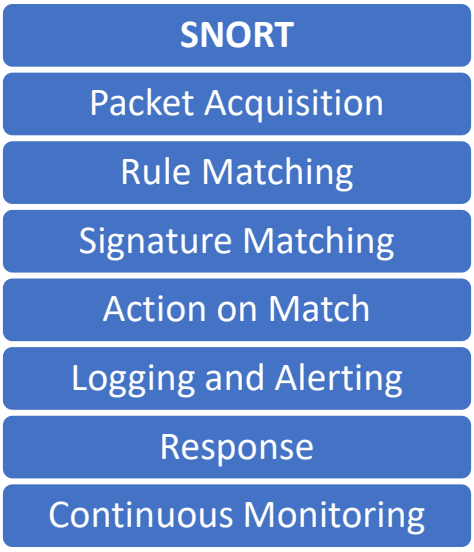


Fig. 3.1 Snort flow diagram

Snort is an intrusion detection and prevention system (IDPS) that is used to detect and prevent network-based attacks.

It is configured with custom rules to detect specific types of attacks, such as ICMP flood attacks, ARP spoof attacks, and DNS spoof attacks.

Snort monitors network traffic in real-time, and alerts to suspicious activity and potential security threats.

The snort tool contains seven phases and they are:

1. Packet Acquisition
2. Rule Matching
3. Signature Matching
4. Action on Match
5. Logging and Alerting

7. Continuous Monitoring

1. Packet Acquisition:

It captures the packets and gathers data from network traffic.

2. Rule Matching:

There are predefined rules to detect attacks like ICMP flood attacks, ARP spoofing attacks, and DNS spoofing attacks.

3. Signature Matching:

A signature is a pattern that is used to match against a network attack.

4. Action on Match:

Snort takes various actions when a rule matches. It includes logging, alerting, blocking network traffic, and automated mechanisms.

5. Logging and Alerting:

When Snort detects any attack first it logs the attack-related info in the Snort log folder with an extension of ids.

6. Response:

We need to configure Snort to respond to the attacks.

7. Continuous Monitoring:

Snort continuously monitors the network.

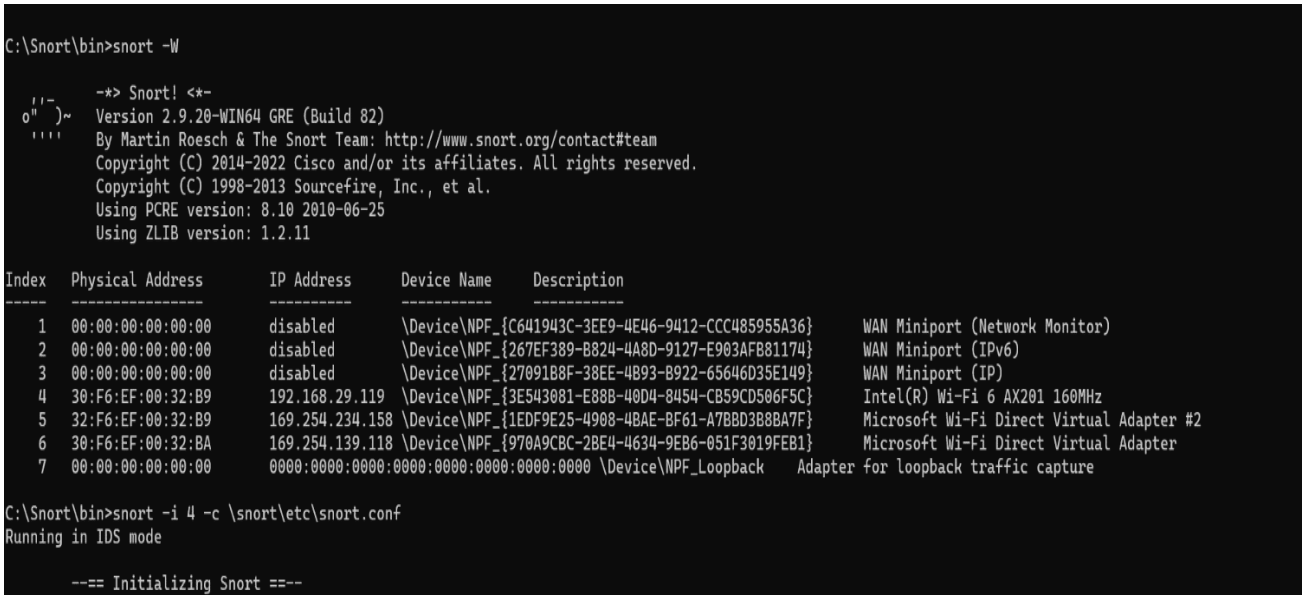


Fig. 3.2 Snort terminal example

The above picture is an example figure of a snort terminal. We can run snort through the terminal. We need to use the respective commands. For example, in the above figure to list the network interfaces we have used a command snort -W.

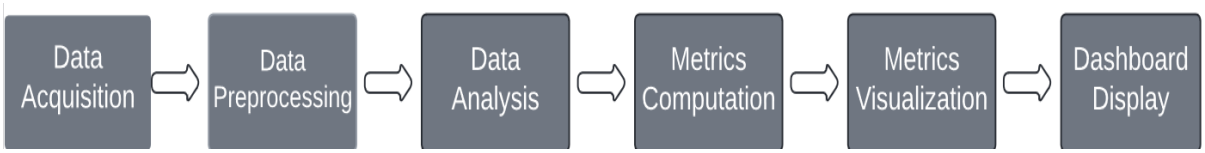
To detect any network attack we are supposed to use a command snort -i 4 -c snort. conf which runs the snort dynamic engine and preprocessor to detect the attacks in the network. If any attack is detected snort logs that attack in log folder with an extension of ids.

Here we need to kill the process because snort will continuously detect the attacks and alerts. So, to capture them we need to kill the process.

Speedtest and speedtest-cli:

- Speedtest is a web service that measures the speed of internet connections by pinging to the best server.
- Speedtest-cli is a command-line interface for Speedtest, it allows us to run speed tests from the terminal.
- These tools are used to measure the latency, upload speed, and download speed of network connections in the project dashboard.
- They provide real-time data on network performance, which is then displayed in the latency meter and the upload/download speed meters.

3.3 FLOW CHART



CHAPTER 4
EXPERIMENTATION AND RESULTS

4.1 EXPERIMENTAL WORK

In my project, I would like to display latency, upload speed, download speed, CPU utilization, Network Utilization, and ICMP flood attack alert.

4.1.1 IMPLEMENTATION OF CODE

```
def clear_snort_log_directory(log_directory):
    try:
        files = os.listdir(log_directory)
        for file in files:
            file_path = os.path.join(log_directory, file)
            if os.path.isfile(file_path):
                os.remove(file_path)
    except Exception as e:
        pass
def run_snort(duration):
    try:

        snort_process = subprocess.Popen(['C:\\snort\\bin\\snort.exe','-i','4','-c','C:\\snort\\etc\\snort.conf'])
        t.sleep(duration)
        snort_process.terminate()
    except Exception as e:
        pass
```

```
def check_alerts(file_path):
    try:
        with open(file_path, 'r') as file:
            alerts = file.read()

        match = re.search(r'(\d{2}/\d{2}-\d{2}:\d{2}:\d{2}.\d{6}) (\d+\.\d+\.\d+\.\d+) ->
(\d+\.\d+\.\d+\.\d+)', alerts)
        if match:
            timestamp = match.group(1)
            attacker_ip = match.group(2)
            mb.showwarning("ICMP Flood Attack Detected", f"Timestamp: {timestamp}\nAttacker IP:
{attacker_ip}")
    except FileNotFoundError:
        pass
    except Exception as e:
        pass

def clear_log_directory_periodically(log_directory):
    while True:
        clear_snort_log_directory(log_directory)
        t.sleep(5)

def run_snort_periodically(duration):
    while True:
        run_snort(duration)
        t.sleep(5)
```

```
def measure_cpu_utilization():
    cpu_percent = psutil.cpu_percent(interval=1)
    return cpu_percent

def update_cpu_graph():
    while True:
        cpu_percent = measure_cpu_utilization()
        cpu_data.append(cpu_percent)
        if len(cpu_data) > 50:
            cpu_data.pop(0)
        cpu_line.set_ydata(cpu_data)
        ax.relim()
        ax.autoscale_view()
        fig.canvas.draw()
        try:
            cpu_label.config(text=f"CPU Utilization: {cpu_percent} %")
        except TclError:
            pass
        t.sleep(1)

def update_network_parameters():
    while True:
        latency, download_speed, upload_speed = measure_network()
        if latency is not None:
            try:
                update_latency_meter(round(latency))
                update_download_meter(round(download_speed))
                update_upload_meter(round(upload_speed))
                latency_label.config(text=f"Latency: {round(latency)} ms")
                download_label.config(text=f"Download Speed: {round(download_speed)} Mbps")
                upload_label.config(text=f"Upload Speed: {round(upload_speed)} Mbps")
```

```
            except:
                pass
        else:
            try:
                latency_label.config(text="Latency: N/A")
                download_label.config(text="Download Speed: N/A")
                upload_label.config(text="Upload Speed: N/A")
            except:
                pass
        t.sleep(3)

def measure_network_utilization():
    total_sent = psutil.net_io_counters().bytes_sent
    total_received = psutil.net_io_counters().bytes_recv
    t.sleep(1)
    sent = psutil.net_io_counters().bytes_sent
    received = psutil.net_io_counters().bytes_recv
    bytes_sent =sent - total_sent
    bytes_received = received - total_received
    utilization_sent = (bytes_sent * 8) / 1
    utilization_received = (bytes_received * 8) / 1
    return utilization_sent, utilization_received
```

4.2 RESULTS AND DISCUSSION

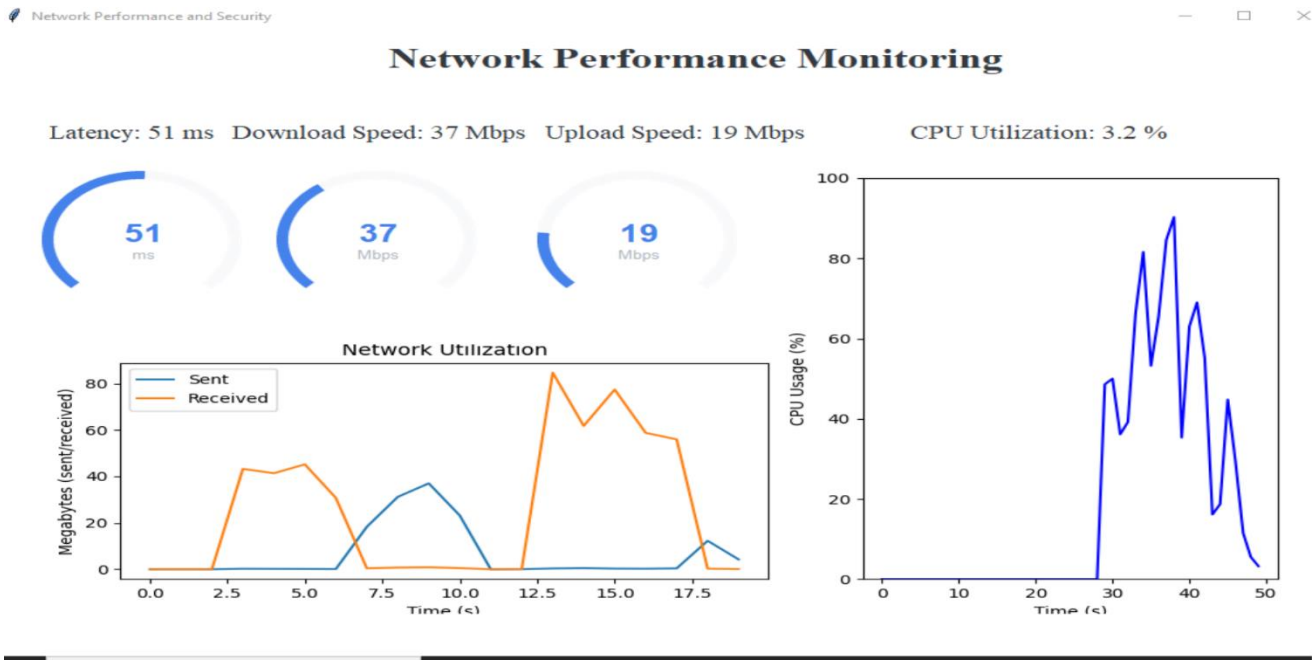


Fig. 4.1 Network Performance Monitoring

The above figure is about network performance monitoring. It contains latency, upload speed, download speed, CPU utilization, and network utilization. Those parameters are displayed with the help of interactive meters and graphs and it continuously updates the values.

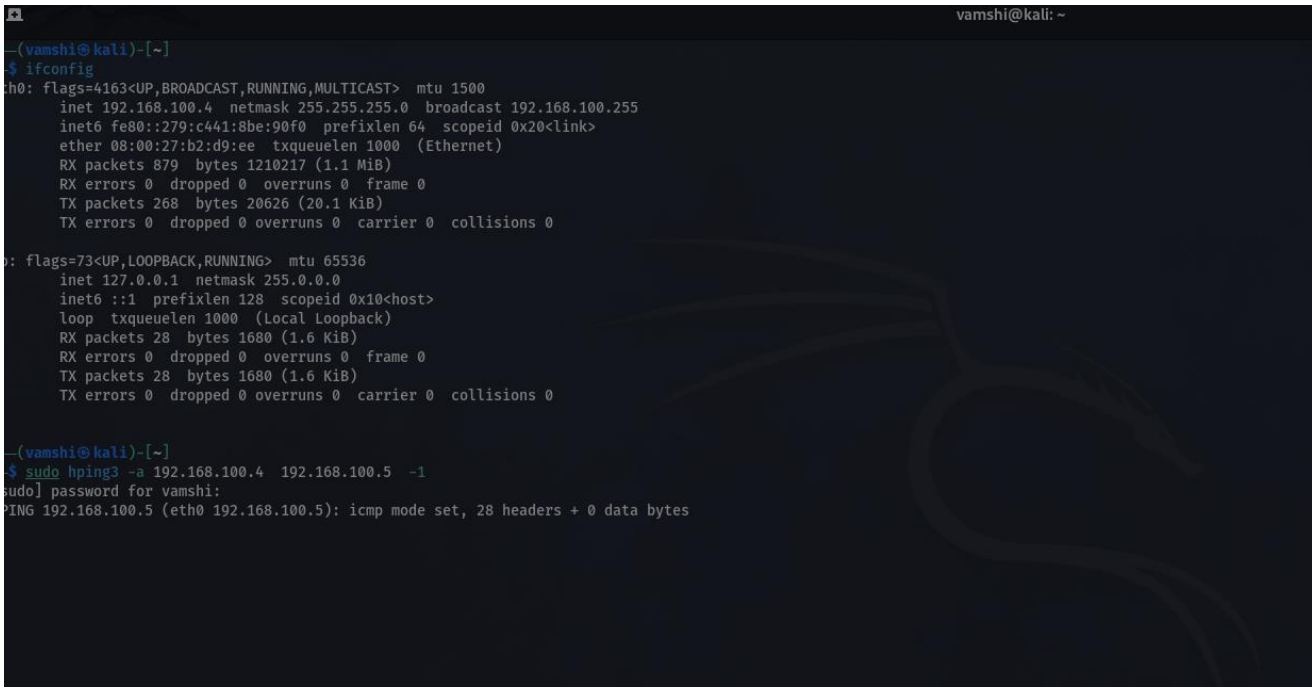


Fig. 4.2 HPING penetration testing

To detect an ICMP flood attack first, we need to perform the attack with the help of Kali Linux. Kali Linux contains various penetration software and tools. One of them is hping which is used to send the data packets in the form of flood. To initiate a flood attack we are supposed to mention the sender IP and target IP.

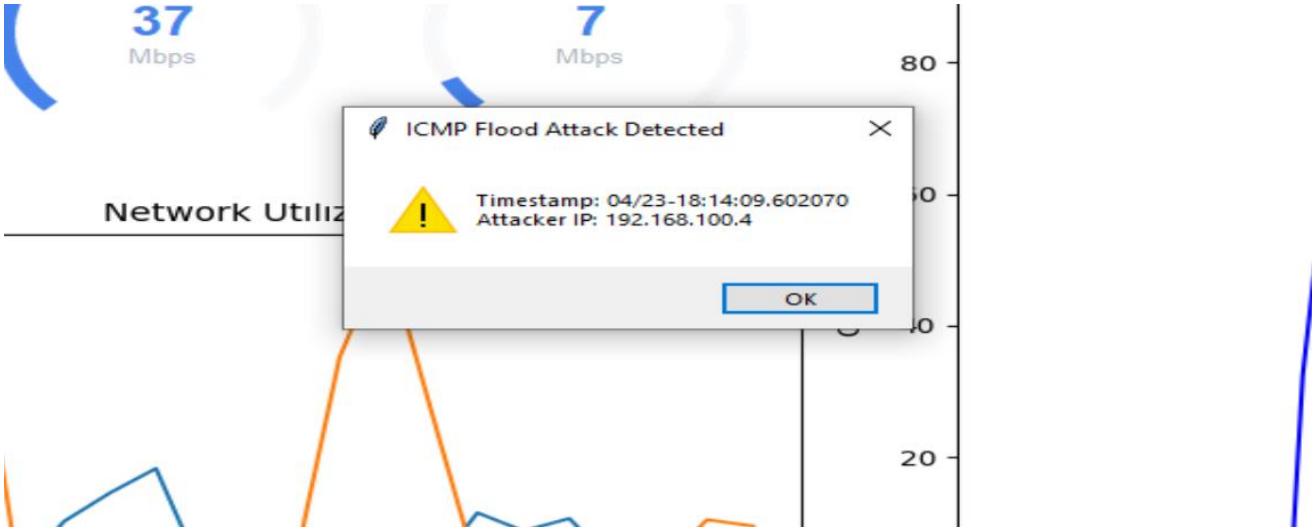


Fig. 4.3 Flood Attack Detection

The above warning message is about flood attack detection with the help of Snort. The message contains the timestamp of the attack and the Attacker's IP address. And this detection of attacks happens every 5 seconds in the dashboard.

CHAPTER 5
CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

In conclusion, my project has successfully created a user-friendly interface that contains security and performance monitoring for networks using Python. By combining various technologies like Tkinter, psutil, Snort, and Speedtest, I have built a GUI that empowers administrators to monitor and manage network resources effectively. Moving forward, we aim to continue refining and expanding our tool to address evolving security threats and performance challenges in network environments.

5.2 FUTURE SCOPE

In the future, my project has the potential for significant expansion and refinement. It integrates advanced analytics, cloud integration, and automated remediation to enhance network monitoring capabilities. By developing interactive dashboards, ensuring scalability, and focusing on user experience improvements. Additionally, compliance reporting, cross-platform support, and community collaboration will further strengthen my project's impact, ensuring it remains adaptable and responsive to evolving security challenges and technological advancements in the field of network monitoring and performance.