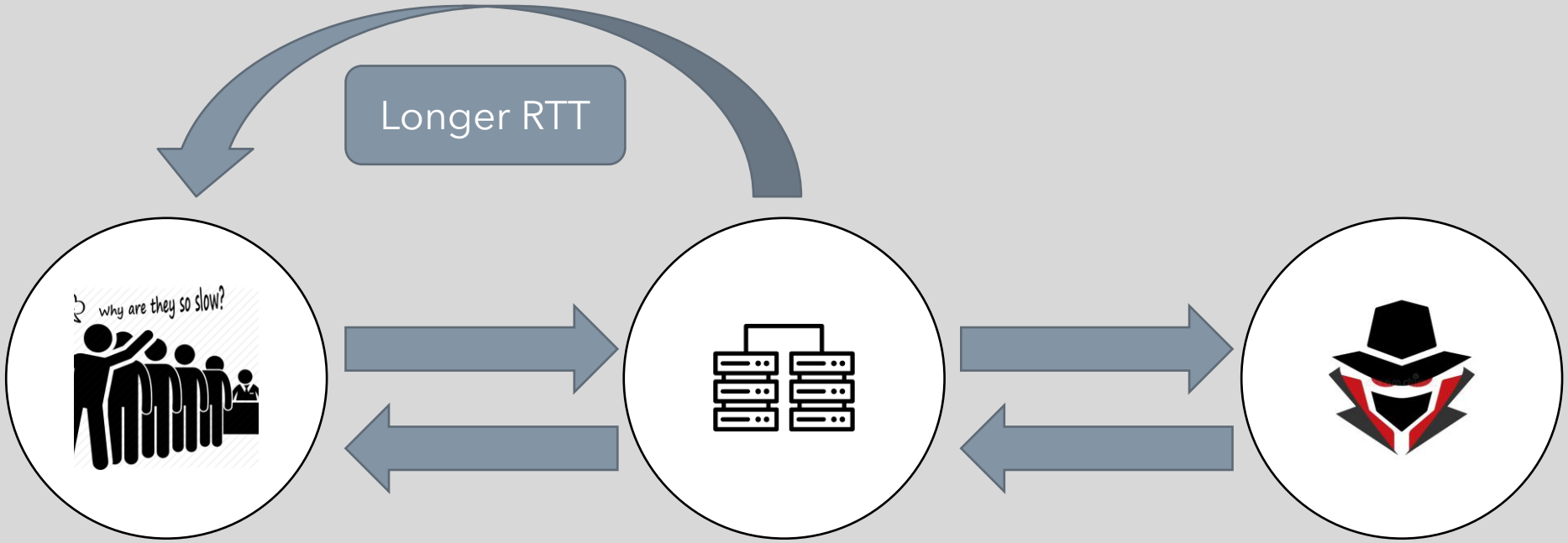
DDOS INTERFACE

Masrik Dahir, Vivian Nguyen

Distributed Denial of Service Attack

- An attempt to disrupt the normal traffic of a targeted server
- Try to impair the ability of a server to respond
- Future and current connection requests take higher Round Trip Time (RTT) to response
- As a result, legitimate users are denied access to the server
- Therefore it exhausts the system’s resources
- Massive DoS attack can crash the server completely

DoS attack in Action



NORMAL CLIENT SERVER ATTACKER CLIENT

Objectives

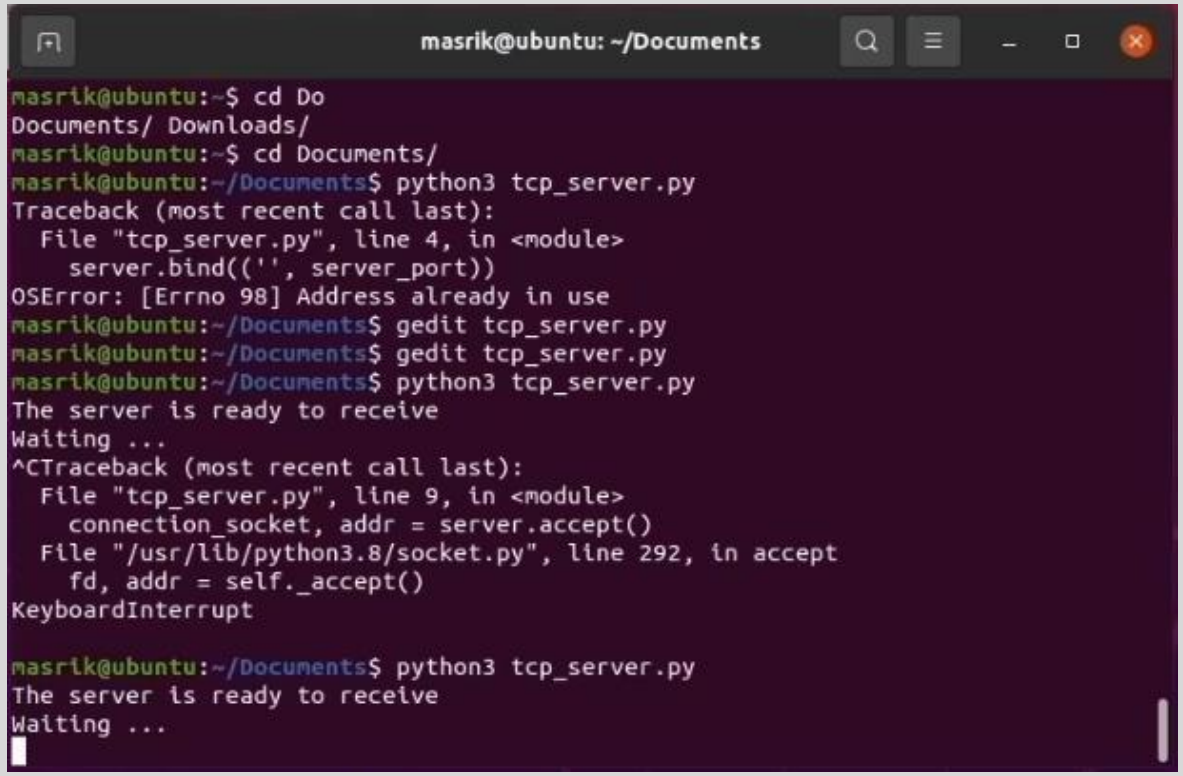
- The objective of this project is to show how we can overload a TCP/UDP server by sending an overwhelming multitude of requests from an easy-to-use interface and causing a denial of service attack.
- We will be setting up our own server to attack.

Our Approach

- We are creating our frontend interface with PyQt5 libraries in python.
- We are using MHDDoS libraries for conducting DDoS attacks on TCP and UDP servers.
- We are using the server codes from Assignment 1 of CMSC 414 and running them on a virtual machine on VMWare.
- We intend to have the server machine overloaded by attacks generated by our DDoS Interface.
- To monitor the Server resource exhaustion, we would create another interface that monitors the server RTT time with the host by every second.

TCP Server

- We start a TCP server on a Virtual Machine
  - IP: 192.168.111.136
  - PORT: 12002
- The Server authenticates with a TCP handshake
- The server receives the Request message and makes it uppercase
- The response message is sent to the Client

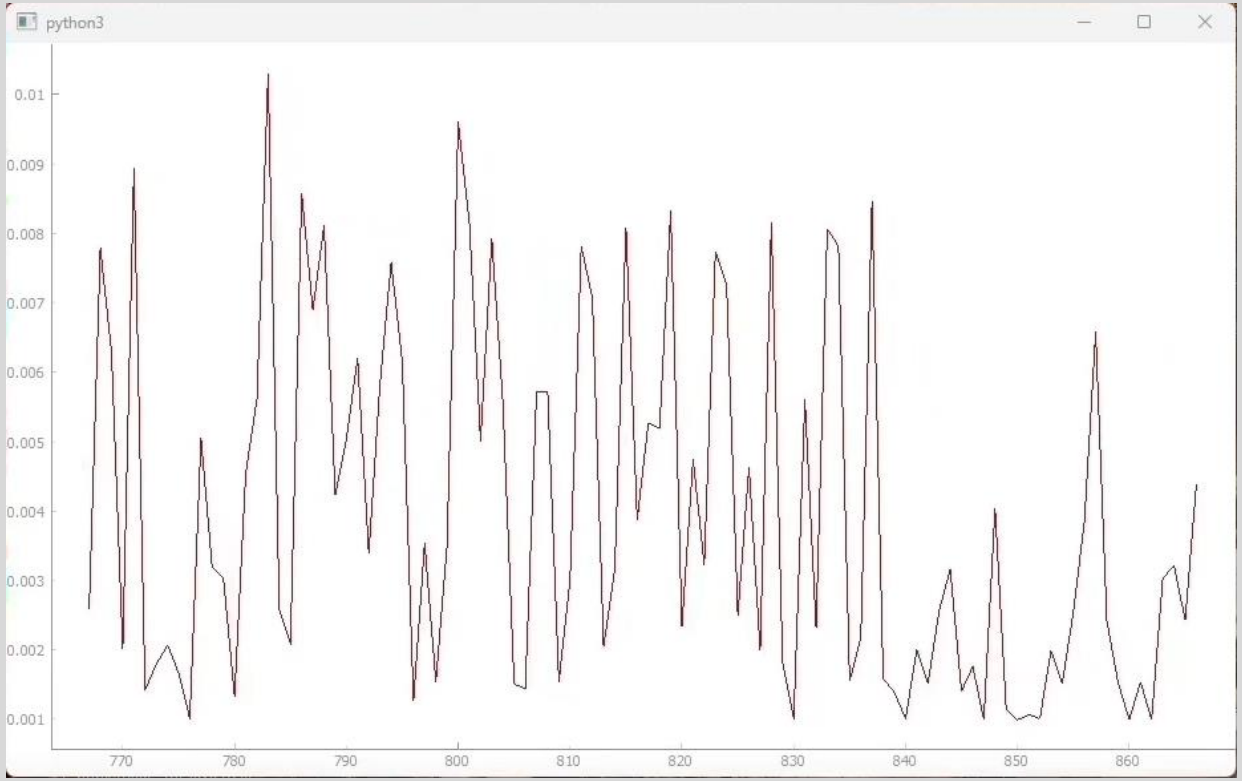


TCP Server Code

```
import socket
server_port = 12001
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('', server_port))
server.listen(1)
print ("The server is ready to receive")
while 1:
    print ("Waiting ...")
    connection_socket, addr = server.accept()
    print ("accept")
    sentence = connection_socket.recv(2048).decode(encoding = 'iso-8859-1')
    print ("Message Received: " + sentence)
    modifiedSentence = sentence.upper()
    connection_socket.send(modifiedSentence.encode())
    connection_socket.close()
```

TCP Normal Client

- We start a Normal TCP Client on the Host Machine
  - IP: 192.168.61.1
- The Client authenticates with a TCP handshake
- The Client sends a Request message every second
- The Client prints the RTT time on the console
- The Client Interface shows RTT overtime in an XY graph

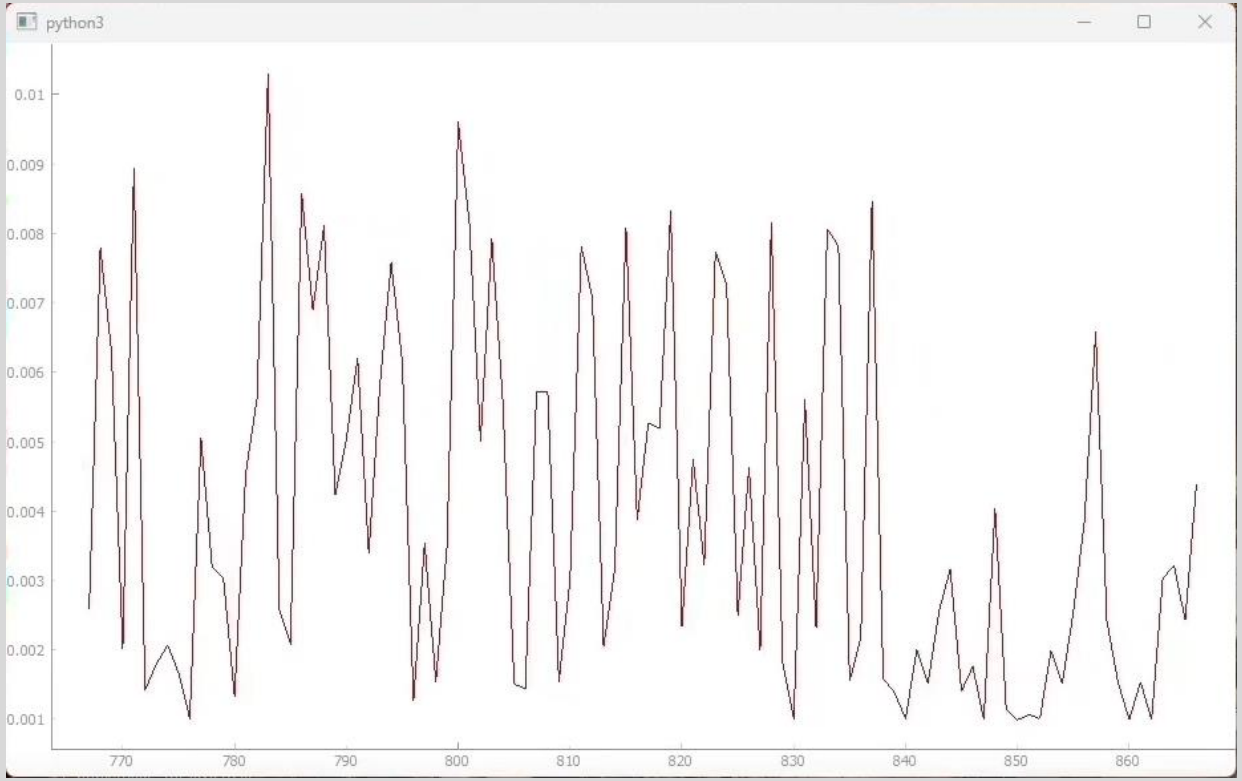


TCP Normal Client Code

```
import matplotlib.pyplot as plt
import socket
import time

server_name = '127.0.0.1'
server_port = 12001

# create a socket object
t = [0]
p = [0]
while True:
    start = time.time()
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((server_name, server_port))
    sentence = "None"
    client.send(sentence.encode())
    modifiedSentence = client.recvfrom(2048)
    print (modifiedSentence[0].decode())
    client.close()
    end = time.time()
    print("Eclipsed time: %f" %(end-start))
    time.sleep(1)

    t.append(end-start)
    p.append(p[len(p)-1] + 100)

    plt.plot(t, p)
    plt.xlabel('Time (hr)')
    plt.ylabel('Position (km)')
```

Normal Client Interface

- RTT Overtime for a Normal (legit) Client
- X-axis = time in millisecond (ms)
- Y-axis = RTT time in second (s)



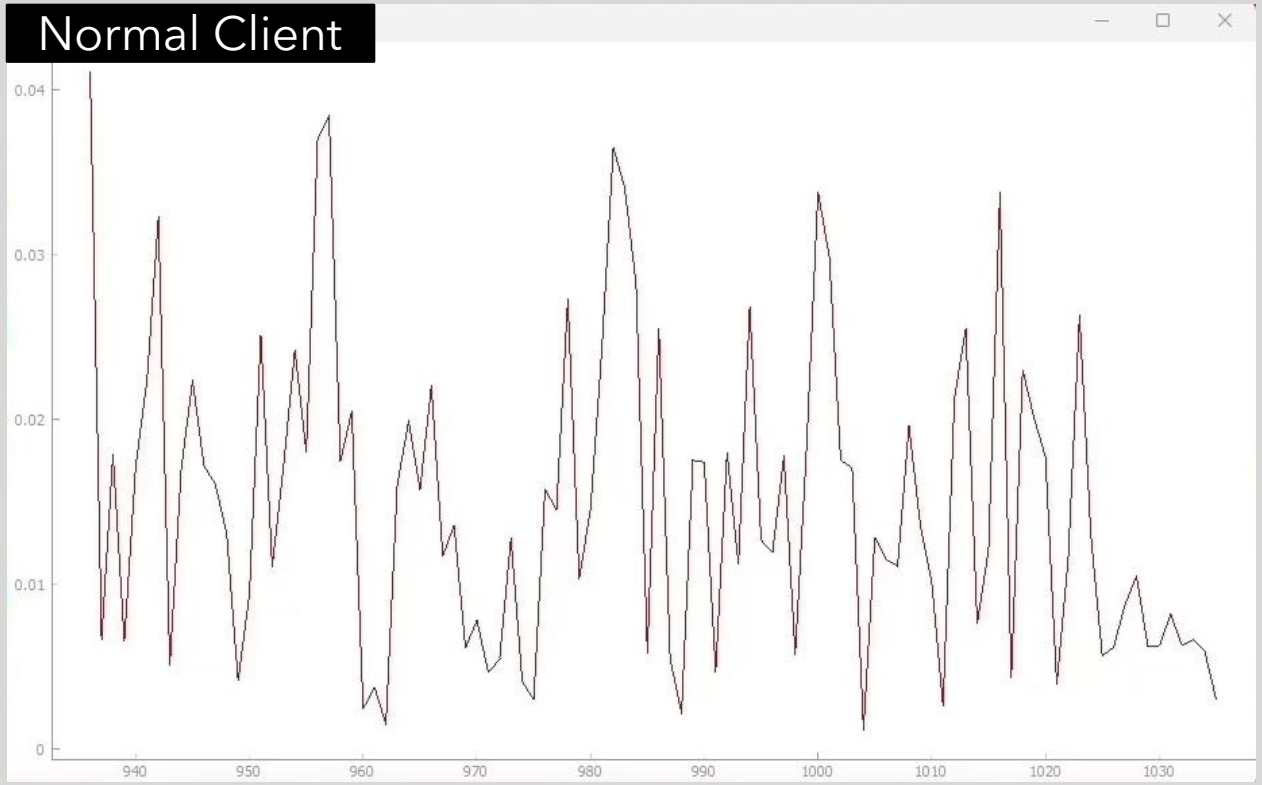
TCP Attacker Client

- We start a TCP Attacker Client on the Host Machine
  - IP: 192.168.61.1
- The Attacker Client authenticates with a TCP handshake
- The Attacker Client sends an enormous amount of TCP requests to overload the server
- The Attacker Client impedes other legitimate users (i.e., Normal Client) to access the server
- When the Attacker uses a lot of threads, the server crashes

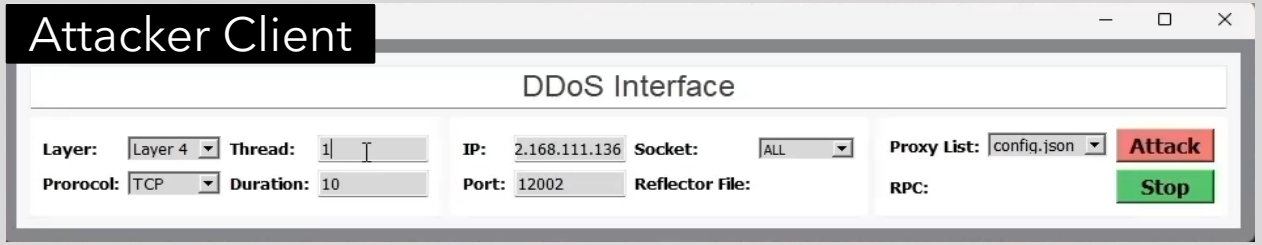
TCP Attacker Client Interface

- Layer: The layer of the protocol i.e. Layer 4
- Protocol: protocols from the selected layer i.e. TCP, UDP
- Thread: sending requests in n threads to multiply the attack intensity
- IP: the IP address of the server
- Port: the port of the server
- Socket: sending requests in the specified socket type
- Proxy List: List of proxy IP addresses

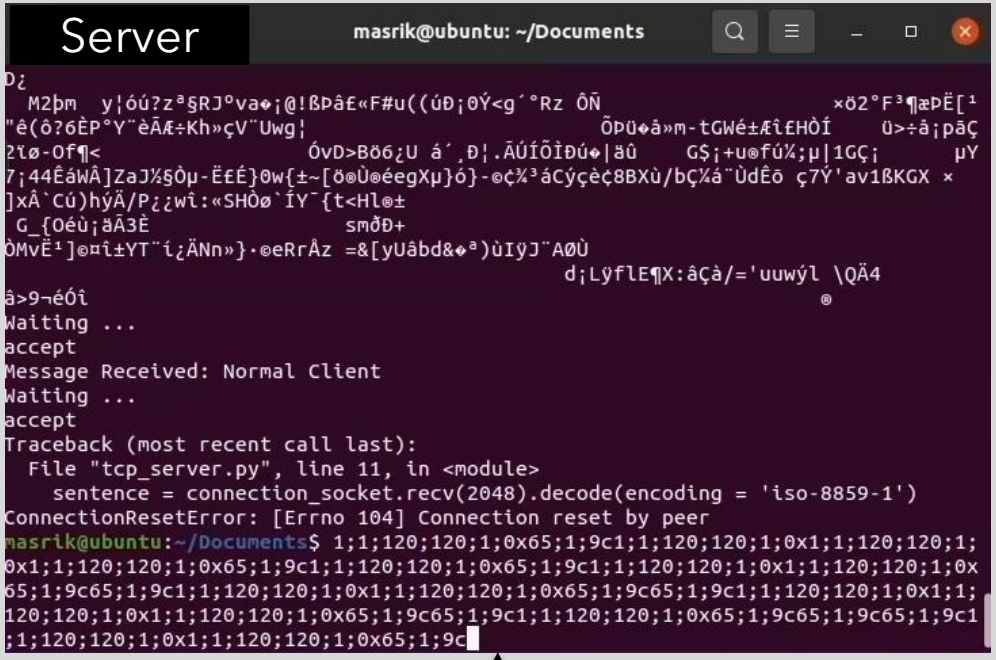
Normal Client



Attacker Client



Server



Server Crashed!

Evaluation Results

- We have evaluated our tool on a Virtual Machine TCP Server
- Our DDoS attacker interface successfully overloaded the TCP Server
- The server crashed in the end and closed all connections both with the Normal Client and the Attacker Client
- Therefore, we are successful in Denying service to legitimate clients using our tool
- As a result, DDoS Interface is a legitimate tool for identifying server vulnerabilities against DDoS attacks

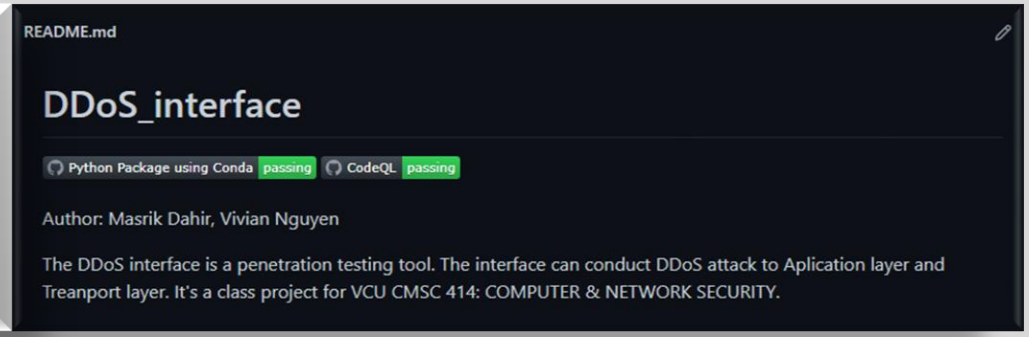
Repository

- Configuration
  - git clone https://github.com/Masrik-Dahir/DDoS_interface.git; # Cloning the Repository
  - .\requirement.bat; # Installing Required Libraries
  - python3 tcp_server.py # Starts Server
  - python3 cl.py # Starts Normal Client
  - python3 form.py # Starts Attacker Client

GitHub:

https://github.com/Masrik-Dahir/DDoS_interface/

- Programming Language: Python
- Library: PyQt5, Matplotlib



References

- Masrik-Dahir. “Masrik-Dahir/DDoS_interface.” *GitHub*, 23 Apr. 2022, github.com/Masrik-Dahir/DDoS_interface. Accessed 24 Apr. 2022.
- MatrixTM. “MatrixTM/MHDDoS: Best DDoS Attack Script Python3, (Cyber / DDos) Attack with 53 Methods.” *GitHub*, 16 Apr. 2022, github.com/MatrixTM/MHDDoS. Accessed 24 Apr. 2022.
- “PyQt5.” *PyPI*, 29 Oct. 2021, pypi.org/project/PyQt5/. Accessed 24 Apr. 2022.
- “Matplotlib — Visualization with Python.” *Matplotlib.org*, 2022, matplotlib.org/. Accessed 24 Apr. 2022.