

CSCE 5580: Computer Network

Project-2 Report

Title: Implementation and Analysis of Socket Programming and TCP SYN Flood Attack

Author: Kishan Kumar Zalavadia

1. Introduction

This paper presents the fundamental concepts and implementations of socket programming and DDoS (Distributed Denial of Service) attacks, more specifically, TCP SYN flood attacks.

This project contains two goals or aspects of networking programming: one is communicating between client-server reliability and how concurrent connections can be managed. In this section, we learn how a client can send a message to the server, and the server replies to the client upon receiving the message from the client.

The second part of the project focuses on learning and understanding/identifying the network vulnerabilities by implementing the TCP SYN flood attack. Upon implementing this DDoS attack, we will gain valuable insights into network security. The project will conclude by understanding the importance of robust sockets, how they are handled, and network monitoring for detecting potential threats.

2. Methodology

2.1 Socket Programming

2.1.1 Overview and Purpose

The main goal for this part of this project is to establish a communication channel between the client and the server on the same network by which they can communicate with each other. The client

sends a message to the server, and the server receives the message and prints it on the console.

2.1.2 Setup and Socket creation

2.1.2.1 Server Code Setup

To create a socket, the 'socket' library from Python programming language is used. The socket is bound to a specific IP address and port number, which, in our case, are 127.0.0.1 and 65525. Once the socket is created, we start listening for the incoming connections from the client. Once a client establishes the connection with the server, it will receive data from the client and display it on the console.

2.1.2.2 Client Code Setup

There is a socket created by the client, also using the 'socket' library provided in Python programming language. This socket will be connected with the server's IP address and server's port numbers, which, in our case, are 127.0.0.1 and 65525. Once the connection is established to the server, it sends a message to the server, which the server will receive and display on the console.

2.1.2.3 Platform Details

The code was developed using Python 3.13.0. It used a local environment with a loopback address of 127.0.0.1 for both the client and server. The program was developed and executed on MacOS.

2.1.2 Data transmission process

Once the server is listening to the client, the client connects to the server using the server's IP address and port number. After the connection is successfully established, the client sends a message to the server, which the server prints on the console. This process showcases a simple form of client-server communication implemented in Python programming.

Server approach: Create socket → Bind the IP address and port number → Start listening.

Client approach: Create socket → Connection establishment → send the message to the server.

Error handling is also implemented to make sure that the errors are handled properly so that the program does not crash.

2.2 TCP SYN Flood Attack

2.2.1 Overview and Purpose

The main objective of implementing the TCP SYN flood attack is to exploit the TCP three-way handshake by sending the SYN packets with spoofed IP addresses. We initiate too many connections but never complete the connection. This forces the server to allocate resources to the spoofed IP address, which indirectly means we are potentially overwhelming the server.

In the TCP handshake, the server allocates memory for each incoming SYN request. The server will wait for the ACK from the client to complete the handshake. However, the client never completes the request. By sending too many SYN requests and not responding to them using SYN-ACK, we exhaust all the resources on the server.

2.2.2 Attack Implementation using Scapy

Packet Creation: To generate multiple TCP SYN packets that have random IP addresses and port numbers, we use Scapy. We make each packet appear as if it originated from a different client.

Loopback Testing: To ensure that the attack is safe without affecting the external networks, we target the 127.0.0.1 localhost server, which we created in the previous part of our project. This loopback will ensure that the risk of unintentional damage is minimized.

Randomization Process: We have created each SYN packet, which has a random source IP address and port number to simulate multiple different clients. To implement this, we make use of Scapy, which helps in creating and customizing the packet layers, such as IP and TCP layers.

2.2.3 Platform Details

To implement the TCP SYN attack, we used Python programming language using the Scapy library on the local machine. The Python version used is 3.13.0. To observe/monitor the network, capture the packet details, and verify the SYN flood, we have used Wireshark. Wireshark is a network monitoring tool that will also help to verify the SYN packet sent and validate the SYN packets. It will also allow us to observe the effects of TCP SYN flood attacks on the network.

2.2.4 Testing

By running the code and observing the network on Wireshark, we can say that the SYN flood code was sending the SYN packets repeatedly, which was filling up the server's connection queue.

3 Results

3.1 Socket Programming

3.1.1 Expected Output

The server successfully receives the message from the client and displays it.

3.1.2 Actual Output

Running the server:

Hello World!

Server is listening on 127.0.0.1:65525

Waiting for a connection...



```
python3 main_server.py
Hello World!
Server is listening on 127.0.0.1:65525
Waiting for a connection...
```

Figure 3.1.2.1

Running the client:

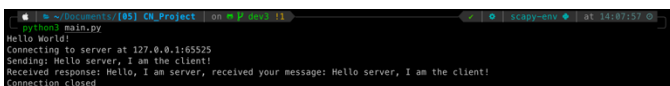
Hello World!

Connecting to the server at 127.0.0.1:65525

Sending: Hello server, I am the client!

Received response: Hello, I am the server, I received your message: Hello server, I am the client!

Connection closed



```
python3 main.py
Hello World!
Connecting to server at 127.0.0.1:65525
Sending: Hello server, I am the client!
Received response: Hello, I am the server, I received your message: Hello server, I am the client!
Connection closed
```

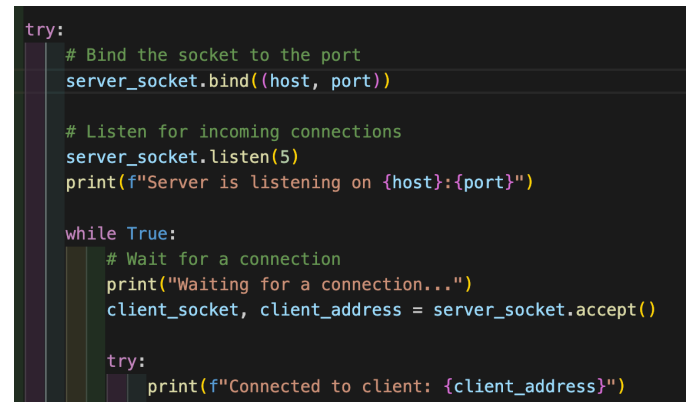
Figure 3.1.2.2

3.1.3 Output Explanation

Server:

To execute the socket programming, we have to ensure that the server is up and running, So we run the server first, and the server listens to any incoming requests, as shown in Figure 3.1.2.1. The server is listening

on IP address 127.0.0.1 and port number 65525.



```
try:
    # Bind the socket to the port
    server_socket.bind((host, port))

    # Listen for incoming connections
    server_socket.listen(5)
    print(f"Server is listening on {host}:{port}")

    while True:
        # Wait for a connection
        print("Waiting for a connection...")
        client_socket, client_address = server_socket.accept()

        try:
            print(f"Connected to client: {client_address}")
```

Figure 3.1.3.1

Client:

Line 1 in Figure 3.1.2.2 prints "Hello World!" it is just to test the working of the Python program.

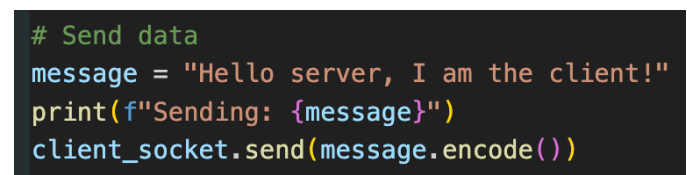
Line 2 in Figure 3.1.2.2, "Connecting to the server at 127.0.0.1:65525", is printed after creating a socket and before actually sending a connection request. The next step will be sending the connection request to the server on IP address 127.0.0.1 and port number 65525. Refer to Figure 3.1.3.2.



```
# Connect to the server
print(f"Connecting to the server at {server_host}:{server_port}")
client_socket.connect((server_host, server_port))
```

Figure 3.1.3.2

Line 3 in Figure 3.1.2.2, "Sending: Hello server, I am the client!" is printed after establishing the connection with the server and before sending the message. The message that we will be sent to the server will be "Hello, server; I am the client!". Refer to Figure 3.1.3.3.



```
# Send data
message = "Hello server, I am the client!"
print(f"Sending: {message}")
client_socket.send(message.encode())
```

Figure 3.1.3.3

Line 4 in Figure 3.1.2.2, "Received response: Hello, I am the server, received your message: Hello server, I am the client!" is displayed by the server. This message shows that the server has received the message sent by the client. Refer to the Figure 3.1.3.4

```
# Receive the data in small chunks and retransmit it
while True:
    data = client_socket.recv(1024).decode()
    if data:
        print(f"Received: {data}")
        # Send acknowledgment
        response = "Hello, I am the server, received your message: " + data
        client_socket.send(response.encode())
    else:
        print("No more data from client")
        break
```

Figure 3.1.3.4

Line 5 in Figure 3.1.2.2, "Connection closed," represents that the communication is completed, and the client closes the connection.

3.2 TCP SYN Flood Attack

3.2.1 Expected Output

The server is inundated with SYN packets, which can be verified through packet capture.

3.2.2 Actual Output

We can observe in Wireshark that there are multiple SYN packets having random IP addresses and port numbers directly to the server, which demonstrates the SYN flood effect.

```
python3 main.py
Starting TCP SYN flood attack simulation
Target: 127.0.0.1:65525
Number of packets to send: 100000
Progress: 22700/100000 packets sent
```

Figure 3.2.2.1

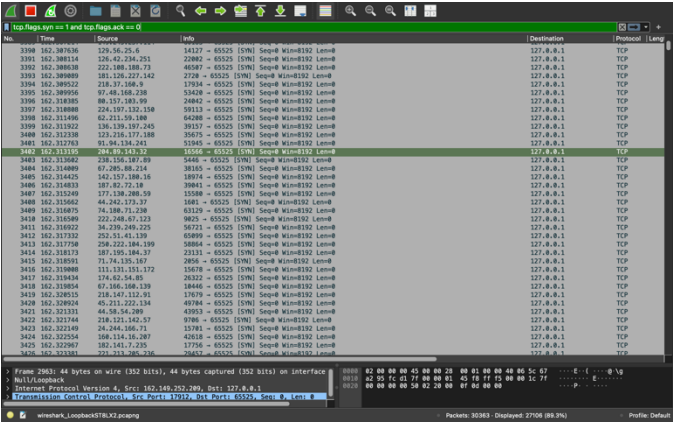


Figure 3.2.2.2

3.2.3 Output Explanation

The high frequency of SYN packets is sent with no corresponding ACK packets, which makes the server's resources exhausted. The source IP addresses are uniformly distributed and random. This will lead to the potential starvation of the connection queue. Refer to Figure 3.2.2.2.

```
for i in range(num_requests):
    # Generate random source IP and port for each packet
    source_ip = f"{randint(1,254)}.{randint(1,254)}.{randint(1,254)}.{randint(1,254)}"
    source_port = randint(1024, 65535)
```

Figure 3.2.3.1 shows the code snippet, which generates a number of random IP addresses and port numbers.

3.2.3.1 Packet Characteristics

- **Protocol:** TCP
- **Flags:** SYN flag set (no ACK)
- **Destination:** 127.0.0.1:65525
- **Source:** Randomly generated for each packet
- **Packet Size:** 40 bytes (typical TCP SYN packet size)
- **Packet Rate:** ~1000 packets per second

3.3 Results summary table

Test	Result	Observations
Socket Communication	Success	Message sent and received as expected.

SYN Flood Attack	Success	Server overwhelmed with SYN requests; confirmed via Wireshark.
------------------	---------	--

4 Conclusion

This project helped to understand network programming and network security. Through the first part of the project, which was socket programming, we learned how the client-server interacts over a network exchanging messages. We gained insights into client-server connection and data transmission.

The next part of the project, which was the TCP SYN flood attack, helped me learn and understand how malicious entities can exploit the TCP handshake connection. This shows the importance of having SYN cookies and firewalls to mitigate such attacks.

We also learned how to monitor the network with the help of tools like Wireshark.

5 References

<https://www.python.org/>

<https://realpython.com/installing-python/>

<https://scapy.net/>

<http://phaethon.github.io/kamene/api/installation.html>

<https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/>