TCP SYN Flood Denial of Service Attack: Implementation and Analysis

Mushfiqur Rahman
Student ID: 2005107

Arnab Dey Kabbya
Student ID: 2005112

July 2025

Contents

1 Introduction

A TCP SYN flood attack is a type of Denial of Service (DOS) attack that exploits vulnerabilities in the TCP three-way handshake process. By overwhelming a target server with half-open connections, the attack consumes system resources, rendering the service unavailable to legitimate users. This project implements a TCP SYN flood attack in Python, showcasing both non-spoofed and IP-spoofed strategies, and analyzes their effectiveness through empirical results.

2 Overview

The TCP SYN flood attack targets the connection establishment phase of the TCP protocol. By sending a large volume of SYN packets without completing the handshake, the attacker forces the server to allocate resources for connections that never finalize, leading to resource exhaustion.

2.1 TCP Three-Way Handshake

The TCP connection establishment involves three steps:

1. **SYN**: The client sends a SYN packet to the server to initiate a connection.
2. **SYN-ACK**: The server responds with a SYN-ACK packet, acknowledging the request.
3. **ACK**: The client sends an ACK packet to complete the handshake.

2.2 SYN Flood Attack Process

In a SYN flood attack, the attacker:

1. Sends numerous SYN packets to the target server.
2. Fails to send the final ACK packet, leaving connections half-open.
3. Overwhelms the server's connection table, exhausting memory and CPU resources.
4. Prevents legitimate users from establishing connections.

3 Implementation Details

3.1 Core Components

The implementation consists of several key components:

- **Packet Building Utilities**: Functions to compute checksums and construct IP and TCP headers.
- **IP Packet Class**: Configurable IPv4 headers with fields like source and destination IPs.
- **TCP Packet Class**: Constructs TCP headers with SYN flags and random sequence numbers.

3.2 Attack Strategies

The project supports two attack modes:

- **Non-Spoofed Attack**: Uses the attacker's real IP address, making it easier to trace but simpler to implement.
- **IP Spoofed Attack**: Uses randomized source IP addresses, increasing attack effectiveness and bypassing basic defenses.

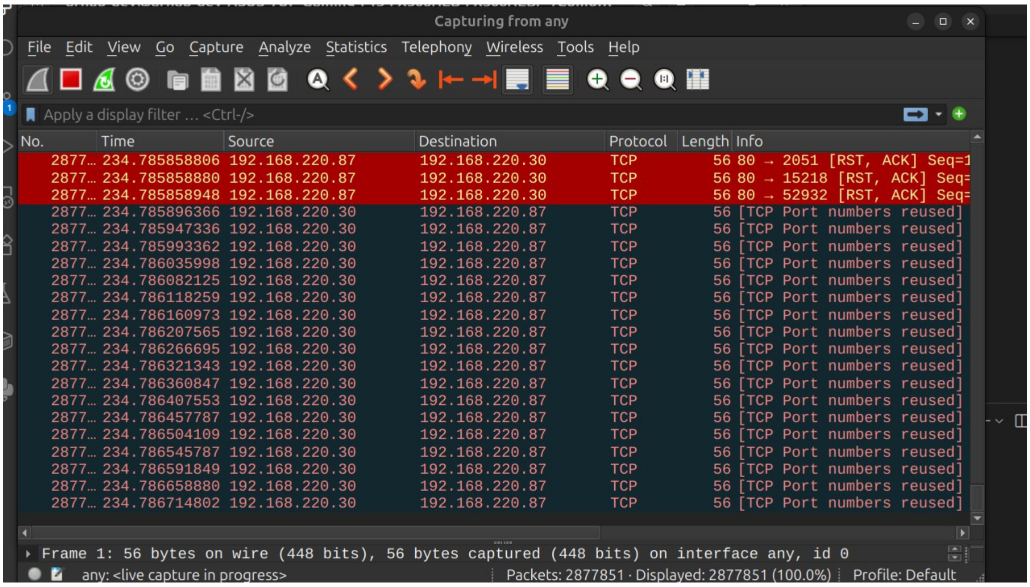
3.3 Key Features

- **Raw Socket Implementation**: Allows direct packet crafting for precise control.
- **Response Monitoring**: Captures SYN-ACK responses to verify target connectivity.
- **Configurable Parameters**: Adjustable target IP, port, attack rate, and thread count.
- **IP Spoofing Option**: Enhances attack effectiveness with randomized source addresses.
- **Rate Control**: Precise control over packets per second (PPS).

4 Attack Results Analysis

The attack's effectiveness was evaluated with and without IP spoofing, as shown in the following figures.

4.1 Without IP Spoofing


Capturing from any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

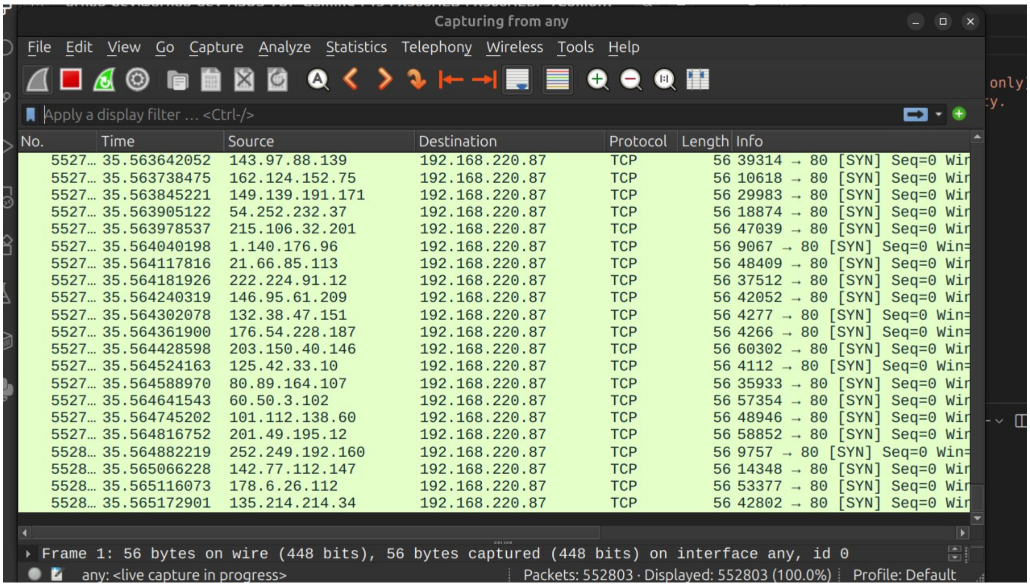
```
No.     Time           Source           Destination      Protocol  Length Info
 2877… 234.785858806 192.168.220.87  192.168.220.30   TCP        56 80 → 2051 [RST, ACK] Seq=1
 2877… 234.785858880 192.168.220.87  192.168.220.30   TCP        56 80 → 15218 [RST, ACK] Seq=
 2877… 234.785858948 192.168.220.87  192.168.220.30   TCP        56 80 → 52932 [RST, ACK] Seq=
 2877… 234.785896366 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.785947336 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.785993362 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786035998 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786082125 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786118259 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786160973 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786207565 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786266695 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786321343 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786360847 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786407553 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786457787 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786504109 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786545787 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786591849 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786658880 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]
 2877… 234.786714802 192.168.220.30  192.168.220.87   TCP        56 [TCP Port numbers reused]

› Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface any, id 0
any: <live capture in progress>   Packets: 2877851 · Displayed: 2877851 (100.0%)   Profile: Default
```

Figure 1: Network traffic during a non-spoofed TCP SYN flood attack. The attacker uses their real IP address, resulting in easily traceable traffic.

- **Attack Method**: Uses the attacker's actual IP address without any IP randomization.
- **Detection Risk**: High, since the traffic originates from a single source.

- **Effect on Target**: Causes moderate resource consumption; blocking is easier.

4.2 With IP Spoofing


Capturing from any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.     Time          Source           Destination     Protocol Length Info
5527… 35.563642052  143.97.88.139    192.168.220.87   TCP      56 39314 → 80 [SYN] Seq=0 Wir
5527… 35.563738475  162.124.152.75   192.168.220.87   TCP      56 10618 → 80 [SYN] Seq=0 Wir
5527… 35.563845221  149.139.191.171  192.168.220.87   TCP      56 29983 → 80 [SYN] Seq=0 Wir
5527… 35.563905122  54.252.232.37    192.168.220.87   TCP      56 18874 → 80 [SYN] Seq=0 Wir
5527… 35.563978537  215.106.32.201   192.168.220.87   TCP      56 47039 → 80 [SYN] Seq=0 Wir
5527… 35.564040198  1.140.176.96     192.168.220.87   TCP      56 9067 → 80 [SYN] Seq=0 Win=
5527… 35.564117816  21.66.85.113     192.168.220.87   TCP      56 48409 → 80 [SYN] Seq=0 Wir
5527… 35.564181926  222.224.91.12    192.168.220.87   TCP      56 37512 → 80 [SYN] Seq=0 Wir
5527… 35.564240319  146.95.61.209    192.168.220.87   TCP      56 42052 → 80 [SYN] Seq=0 Wir
5527… 35.564302078  132.38.47.151    192.168.220.87   TCP      56 4277 → 80 [SYN] Seq=0 Win=
5527… 35.564361900  176.54.228.187   192.168.220.87   TCP      56 4266 → 80 [SYN] Seq=0 Win=
5527… 35.564428598  203.150.40.146   192.168.220.87   TCP      56 60302 → 80 [SYN] Seq=0 Wir
5527… 35.564524163  125.42.33.10     192.168.220.87   TCP      56 4112 → 80 [SYN] Seq=0 Win=
5527… 35.564588970  80.89.164.107    192.168.220.87   TCP      56 35933 → 80 [SYN] Seq=0 Wir
5527… 35.564641543  60.50.3.102      192.168.220.87   TCP      56 57354 → 80 [SYN] Seq=0 Wir
5527… 35.564745202  101.112.138.60   192.168.220.87   TCP      56 48946 → 80 [SYN] Seq=0 Wir
5527… 35.564816752  201.49.195.12    192.168.220.87   TCP      56 58852 → 80 [SYN] Seq=0 Wir
5528… 35.564882219  252.249.192.160  192.168.220.87   TCP      56 9757 → 80 [SYN] Seq=0 Win=
5528… 35.565066228  142.77.112.147   192.168.220.87   TCP      56 14348 → 80 [SYN] Seq=0 Wir
5528… 35.565116073  178.6.26.112     192.168.220.87   TCP      56 53377 → 80 [SYN] Seq=0 Wir
5528… 35.565172901  135.214.214.34   192.168.220.87   TCP      56 42802 → 80 [SYN] Seq=0 Wir

Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface any, id 0
any: <live capture in progress>    Packets: 552803 · Displayed: 552803 (100.0%)    Profile: Default
```

Figure 2: Network traffic during an IP-spoofed TCP SYN flood attack. The attacker sends SYN packets using randomized source IP addresses.

- **Attack Method**: Randomized spoofed IP addresses for each SYN packet.
- **Detection Risk**: Low, because packets appear to come from multiple sources.
- **Effect on Target**: Significantly higher resource exhaustion; filtering becomes complex.

4.3 Victim Impact

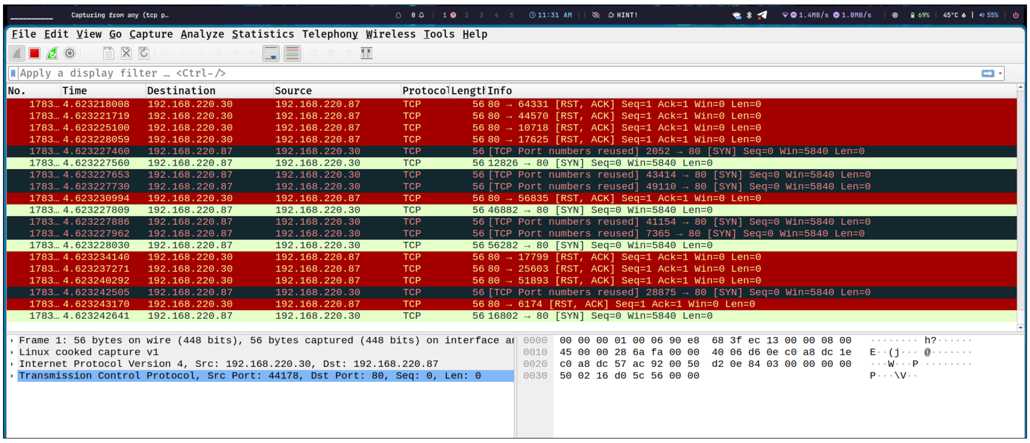
The resource utilization on the victim server differs significantly based on whether IP spoofing is used. Each scenario is explained below with its corresponding visualization.

Without IP Spoofing

- The server experiences moderate resource strain due to repeated SYN packets from a single IP.
- Blocking the attacker's IP using firewall rules or rate-limiting is straightforward.
- Network traffic patterns are consistent and easily identifiable.

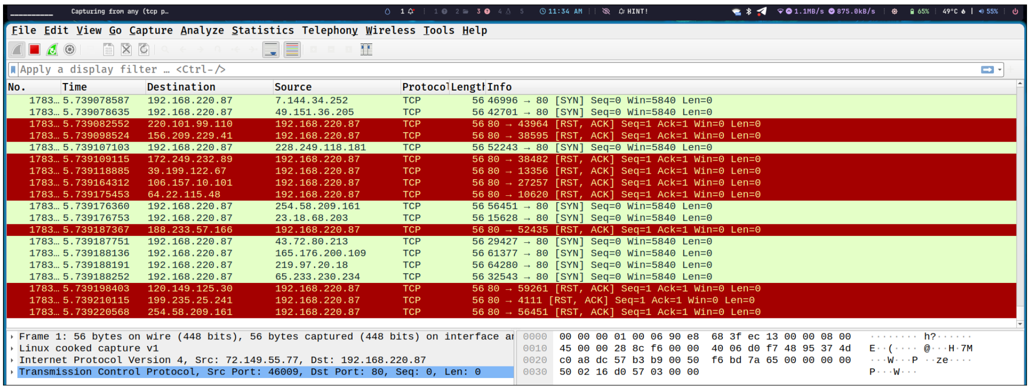
With IP Spoofing

- The server faces severe resource depletion because each SYN packet appears to originate from a different IP.
- Filtering becomes extremely difficult, making the attack far more effective.
- Connection queues fill rapidly, leading to service denial for legitimate users.


Capturing from any (tcp p…
11:31 AM
HINT!
1.4MB/s
1.0MB/s
69%
45°C
55%
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.      Time         Destination      Source           Protocol Length Info
 1783… 4.623218008  192.168.220.30   192.168.220.87   TCP      56 80 → 64331 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 4.623221719  192.168.220.30   192.168.220.87   TCP      56 80 → 44570 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 4.623225100  192.168.220.30   192.168.220.87   TCP      56 80 → 10718 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 4.623228059  192.168.220.30   192.168.220.87   TCP      56 80 → 17625 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 4.623227460  192.168.220.87   192.168.220.30   TCP      56 [TCP Port numbers reused] 2052 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 4.623227560  192.168.220.87   192.168.220.30   TCP      56 12826 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 4.623227653  192.168.220.87   192.168.220.30   TCP      56 [TCP Port numbers reused] 43414 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 4.623227730  192.168.220.87   192.168.220.30   TCP      56 [TCP Port numbers reused] 49110 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 4.623230994  192.168.220.30   192.168.220.87   TCP      56 80 → 56835 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 4.623227809  192.168.220.87   192.168.220.30   TCP      56 46882 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 4.623227886  192.168.220.87   192.168.220.30   TCP      56 [TCP Port numbers reused] 41154 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 4.623227962  192.168.220.87   192.168.220.30   TCP      56 [TCP Port numbers reused] 7365 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 4.623228030  192.168.220.87   192.168.220.30   TCP      56 56282 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 4.623234140  192.168.220.30   192.168.220.87   TCP      56 80 → 17799 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 4.623237271  192.168.220.30   192.168.220.87   TCP      56 80 → 25603 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 4.623240292  192.168.220.30   192.168.220.87   TCP      56 80 → 51893 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 4.623242505  192.168.220.87   192.168.220.30   TCP      56 [TCP Port numbers reused] 28875 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 4.623243170  192.168.220.30   192.168.220.87   TCP      56 80 → 6174 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 4.623242641  192.168.220.87   192.168.220.30   TCP      56 16802 → 80 [SYN] Seq=0 Win=5840 Len=0

› Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface ar
› Linux cooked capture v1
› Internet Protocol Version 4, Src: 192.168.220.30, Dst: 192.168.220.87
› Transmission Control Protocol, Src Port: 44178, Dst Port: 80, Seq: 0, Len: 0

0000  00 00 00 01 00 06 90 e8  68 3f ec 13 00 00 08 00   ········ h?······
0010  45 00 00 28 6a fa 00 00  40 06 d6 0e c0 a8 dc 1e   E··(j··· @·······
0020  c0 a8 dc 57 ac 92 00 50  d2 0e 84 03 00 00 00 00   ···W···P ········
0030  50 02 16 d0 5c 56 00 00                            P···\V··
```

Figure 3: Server resource usage during a non-spoofed TCP SYN flood attack. Traffic originates from a single IP, making detection and blocking easier.


Capturing from any (tcp p…
11:34 AM
HINT!
1.1MB/s
875.0kB/s
65%
49°C
55%
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.      Time         Destination      Source           Protocol Length Info
 1783… 5.739078587  192.168.220.87   7.144.34.252     TCP      56 46996 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 5.739078635  192.168.220.87   49.151.36.205    TCP      56 42701 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 5.739082552  220.101.99.110   192.168.220.87   TCP      56 80 → 43964 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 5.739098524  156.209.229.41   192.168.220.87   TCP      56 80 → 38595 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 5.739107103  192.168.220.87   228.249.118.181  TCP      56 52243 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 5.739109115  172.249.232.89   192.168.220.87   TCP      56 80 → 38482 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 5.739118885  39.199.122.67    192.168.220.87   TCP      56 80 → 13356 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 5.739164312  106.157.10.101   192.168.220.87   TCP      56 80 → 27257 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 5.739175453  64.22.115.48     192.168.220.87   TCP      56 80 → 10620 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 5.739176360  192.168.220.87   254.58.209.161   TCP      56 56451 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 5.739176753  192.168.220.87   23.18.68.203     TCP      56 15628 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 5.739187367  188.233.57.166   192.168.220.87   TCP      56 80 → 52435 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 5.739187751  192.168.220.87   43.72.80.213     TCP      56 29427 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 5.739188136  192.168.220.87   165.176.200.109  TCP      56 61377 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 5.739188191  192.168.220.87   219.97.20.18     TCP      56 64280 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 5.739188252  192.168.220.87   65.233.230.234   TCP      56 32543 → 80 [SYN] Seq=0 Win=5840 Len=0
 1783… 5.739198403  120.149.125.30   192.168.220.87   TCP      56 80 → 59261 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 5.739210115  199.235.25.241   192.168.220.87   TCP      56 80 → 4111 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 1783… 5.739220568  254.58.209.161   192.168.220.87   TCP      56 80 → 56451 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

› Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface ar
› Linux cooked capture v1
› Internet Protocol Version 4, Src: 72.149.55.77, Dst: 192.168.220.87
› Transmission Control Protocol, Src Port: 46009, Dst Port: 80, Seq: 0, Len: 0

0000  00 00 00 01 00 06 90 e8  68 3f ec 13 00 00 08 00   ········ h?······
0010  45 00 00 28 8c f6 00 00  40 06 d0 f7 48 95 37 4d   E··(···· @···H·7M
0020  c0 a8 dc 57 b3 b9 00 50  f6 bd 7a 65 00 00 00 00   ···W···P ··ze····
0030  50 02 16 d0 57 03 00 00                            P···W···
```

Figure 4: Server resource usage during an IP-spoofed TCP SYN flood attack. The system struggles with diversified spoofed IP traffic, making mitigation challenging.

5 Technical Implementation

5.1 Packet Structure

The TCP SYN flood attack constructs raw packets by manually crafting IP and TCP headers. The key fields used in the attack are summarized below:

IP Header Fields

Field	Description
Version	4 (IPv4)
Header Length	5 (20 bytes)
Type of Service	0
Total Length	40 bytes (IP + TCP)
Identification	Randomly generated
Flags	Don't Fragment
TTL	64
Protocol	TCP (6)
Source IP	Attacker IP (or spoofed)
Destination IP	Victim server IP

Table 1: IP Header Fields used in the SYN flood packet

TCP Header Fields

Field	Description
Source Port	Randomized
Destination Port	Target service port
Sequence Number	Randomized
Acknowledgment Number	0
Header Length	5 (20 bytes)
Flags	SYN (0x02)
Window Size	5840
Checksum	Calculated dynamically

Table 2: TCP Header Fields used in the SYN flood packet

The packet construction ensures compliance with the TCP/IP protocol while allowing flexibility for attack customization. The fields highlighted above are dynamically modified during the flood to avoid detection and improve attack efficiency.

5.2 Attack Flow

The attack follows these steps:

1. **Initialization**: Sets up raw sockets with IP header inclusion.
2. **Target Validation**: Verifies the target IP address format.
3. **Connectivity Test**: Sends an initial SYN packet to confirm reachability.
4. **Thread Deployment**: Launches multiple worker threads for parallel packet transmission.
5. **Flood Execution**: Continuously sends SYN packets at the configured rate.
6. **Monitoring**: Tracks SYN-ACK responses to adjust attack parameters.

6 Defense Mechanisms

To mitigate TCP SYN flood attacks, the following defenses can be implemented:

6.1 Network-Level Defenses

- **SYN Cookies**: Use stateless connection handling to avoid resource allocation for half-open connections.
- **Connection Rate Limiting**: Restrict the number of new connections per IP address.
- **Firewall Rules**: Block traffic exhibiting suspicious patterns.
- **Load Balancing**: Distribute incoming connections across multiple servers.

6.2 System-Level Defenses

- **TCP Backlog Tuning**: Increase the connection queue size to handle more half-open connections.
- **Timeout Reduction**: Accelerate the cleanup of half-open connections.

- **Resource Monitoring**: Continuously track system resource usage.
- **Intrusion Detection**: Implement automated systems to recognize attack patterns.

7 Conclusion

The TCP SYN flood attack remains a potent threat to network availability. This implementation highlights the simplicity of executing such attacks and the enhanced effectiveness of IP spoofing. By understanding these attack vectors, network administrators can develop robust countermeasures to ensure service availability and security.

8 References

- Eddy, W. (2007). TCP SYN Flooding Attacks and Common Mitigations. RFC 4987, IETF.
- Stevens, W. R. (1994). *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley.
- Postel, J. (1981). Transmission Control Protocol. RFC 793, IETF.