1. Capture a UDP packet, verify the checksum using **16-bit One's Complement Sum** algorithm.

```

*import socket*

*# Function to calculate the 16-bit One's Complement Sum*

*def ones\_complement\_sum(data):*

*total = 0*

*for i in range(0, len(data), 2):*

*if i + 1 < len(data):*

*word = (data[i] << 8) + data[i + 1]*

*total += word*

*elif i < len(data):*

*total += (data[i] << 8)*

*while (total >> 16) > 0:*

*total = (total & 0xFFFF) + (total >> 16)*

*return total*

*# Capture a UDP packet (you may need to install the 'scapy' library)*

*# Replace 'eth0' with your network interface*

*# Note: This example uses scapy for capturing, you may need to install it.*

*from scapy.all import sniff*

*def capture\_udp\_packet():*

*udp\_packet = None*

*def packet\_handler(packet):*

*nonlocal udp\_packet*

*if packet.haslayer("UDP"):*

*udp\_packet = packet*

*sniff(iface="eth0", prn=packet\_handler, count=1)*

*return udp\_packet*

*# Verify the checksum of the captured UDP packet*

*def verify\_checksum(udp\_packet):*

*udp\_data = bytes(udp\_packet['UDP'].payload)*

*received\_checksum = udp\_packet['UDP'].chksum*

*calculated\_checksum = ones\_complement\_sum(udp\_data)*

*if calculated\_checksum == received\_checksum:*

*print("Checksum is valid.")*

*else:*

*print("Checksum is invalid.")*

*# Main function*

*if \_\_name\_\_ == "\_\_main\_\_":*

*udp\_packet = capture\_udp\_packet()*

*if udp\_packet:*

*verify\_checksum(udp\_packet)*

*else:*

*print("No UDP packet captured.")*

```

1. **What's TCP 3-Way Handshake? Draw a diagram to illustrate the process using real packets captured in a TCP session. Fill in the values of some key fields of the packets.**

The TCP 3-way handshake is a process used to establish a connection between two devices over a TCP/IP network. It involves three steps and ensures that both devices are ready to exchange data. Below is a description of the process along with a simplified diagram. Please note that real packet captures would be extensive, so I'll provide a simplified illustration.

**Step 1: SYN (Synchronize)**

The client (A) initiates the connection by sending a TCP segment with the SYN (Synchronize) flag set.

It selects an initial sequence number (ISN), which is typically a random value.

The source port and destination port numbers are specified.

Other TCP header fields are set, including the checksum.

Client (A) Server (B)

| |

| ---------------------- [SYN] ------------------------> |

| Source Port = X Destination Port = Y |

| ISN (Initial Sequence Number) = A |

| |

**Step 2: SYN-ACK (Synchronize-Acknowledgment)**

Upon receiving the SYN segment, the server (B) acknowledges the request by sending a TCP segment with both the SYN and ACK flags set.

Server (B) selects its own ISN and acknowledges the client's sequence number.

Other TCP header fields are set, including the checksum.

Client (A) Server (B)

| |

| <--------------------- [SYN, ACK] ------------------- |

| Source Port = Y Destination Port = X |

| ISN = B ACK = A+1 |

| |

**Step 3: ACK (Acknowledgment)**

Finally, the client (A) acknowledges the server's acknowledgment by sending a TCP segment with only the ACK flag set.

The ACK field is set to the server's ISN + 1.

The client and server are now connected and can start exchanging data.

Client (A) Server (B)

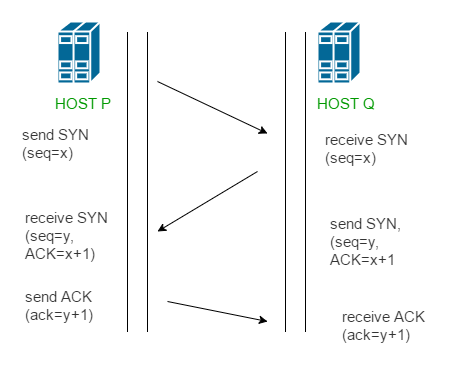
| |

| ------------------------ [ACK] ----------------------> |

| Source Port = X Destination Port = Y |

| ACK = B+1 |

| |



1. **What's TCP 4-Way teardown? Draw a diagram to illustrate the process using real packets captured in a TCP session. Fill in the values of some key fields of the packets.**

The TCP 4-way teardown is the process used to gracefully terminate a TCP connection between two devices. It ensures that both sides are finished sending data and that the connection is closed properly.

**Step 1: Client Sends FIN (Finish)**

The client (A) initiates the termination process by sending a TCP segment with the FIN (Finish) flag set.

The client is done sending data, so it sets the FIN flag and acknowledges any remaining data from the server.

The client's sequence number is incremented.

Other TCP header fields are set, including the checksum.

Client (A) Server (B)

| |

| --------------- [FIN, ACK] ------------------------> |

| Source Port = X Destination Port = Y |

| SEQ = A ACK = B+1 |

| |

**Step 2: Server Acknowledges the FIN**

Upon receiving the FIN from the client, the server (B) acknowledges the client's FIN.

The server sets the ACK flag and acknowledges the client's FIN.

The server may still have data to send, so it continues sending data segments.

Server's sequence number is incremented, and other TCP header fields are set.

Client (A) Server (B)

| |

| <--------------- [ACK] ----------------------------- |

| Source Port = Y Destination Port = X |

| SEQ = B+1 ACK = A+1 |

| |

**Step 3: Server Sends FIN**

Once the server is done sending data, it sends its own FIN to the client.

The server sets the FIN flag to signal that it's finished sending.

The server acknowledges any remaining data from the client.

The server's sequence number is incremented, and other TCP header fields are set.

Client (A) Server (B)

| |

| <---------------- [FIN, ACK] ----------------------- |

| Source Port = Y Destination Port = X |

| SEQ = B+1 ACK = A+1 |

| |

**Step 4: Client Acknowledges Server's FIN**

The client acknowledges the server's FIN with an ACK.

The client's ACK field is set to the server's sequence number + 1.

The client may continue sending data if needed.

The connection is now properly closed.

Client (A) Server (B)

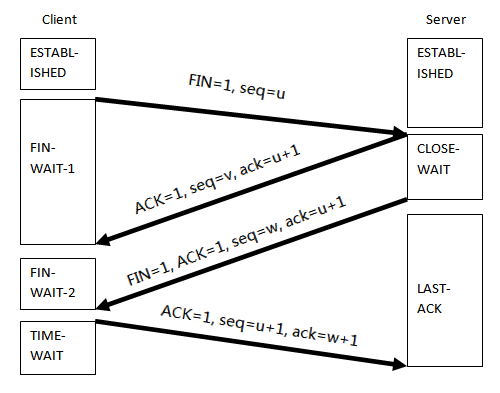
| |

| --------------- [ACK] -------------------------------> |

| Source Port = X Destination Port = Y |

| SEQ = A+1 ACK = B+2 |

| |



Bonus:

Find two interview questions about TCP, and provide the answer. please provide the reference.

**Interview Question 1:**

Question: What is the difference between TCP and UDP, and in what scenarios would you choose one over the other for data transmission?

Answer: TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are two transport layer protocols used for data transmission in computer networks. The main differences are:

Reliability: TCP is a reliable protocol that ensures the ordered, error-checked, and complete delivery of data. UDP, on the other hand, is not reliable and does not guarantee these features.

Connection-oriented vs. Connectionless: TCP is connection-oriented, meaning it establishes a connection before data transfer and ensures a clean teardown after data transfer. UDP is connectionless and does not establish a connection; it simply sends data.

Overhead: TCP has more overhead due to its reliability features, such as sequencing, acknowledgment, and error correction. UDP has less overhead.

Use Cases: TCP is suitable for applications where data integrity and reliability are critical, such as web browsing, email, and file transfer. UDP is suitable for real-time applications like video streaming, online gaming, and VoIP, where low latency is more important than error-checking and retransmission.

Reference: RFC 793 - Transmission Control Protocol and RFC 768 - User Datagram Protocol.

**Interview Question 2:**

Question: What is the purpose of the TCP 3-way handshake, and how does it work?

Answer: The TCP 3-way handshake is a critical process for establishing a connection between two devices in a TCP/IP network. Its purposes are:

SYN (Synchronize): In the first step, the client sends a TCP segment with the SYN (Synchronize) flag set to initiate the connection. It selects an initial sequence number (ISN) for the connection.

SYN-ACK (Synchronize-Acknowledgment): The server responds with a TCP segment containing both the SYN and ACK flags set, acknowledging the client's request. The server also selects its own ISN and acknowledges the client's sequence number.

ACK (Acknowledgment): Finally, the client acknowledges the server's acknowledgment by sending a TCP segment with only the ACK flag set. The ACK field is set to the server's ISN + 1. At this point, the connection is established.

The 3-way handshake ensures that both sides are ready to exchange data, sets the initial sequence numbers, and confirms that the connection is valid and reliable.

Reference: RFC 793 - Transmission Control Protocol.