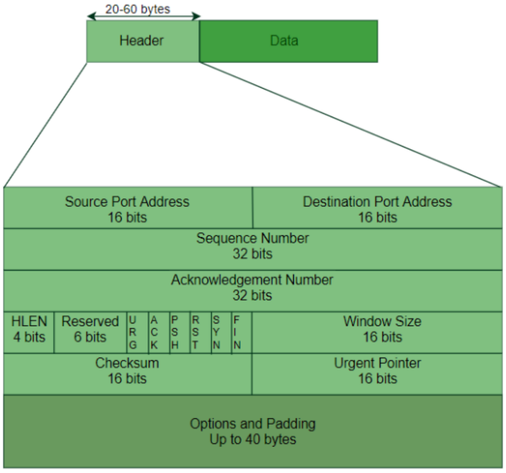
NAME: ANANYA PILLAI

Experiment: Maintaining the integrity of TCP segments using TCP Checksum Calculation/Verification

Aim:

- **To understand the role of data integrity is risk assessment.**
- **To write a python program to calculate and verity the TCP checksum at sender and receiver sides respectively to ensure the data integrity.**
- **To test the code with real TCP checksum values captured in Wireshark tool.**
- **To prepare a documentation with appropriate information (code, explanation, screenshots, etc.).**
- **Your name, VIT register number should be carried in real TCP payloads. The same should be captured in Wireshark screenshots.**



TCP Checksum has 3 main fields TCP Pseudo Header,TCP Header and TCP

- The **TCP Pseudo header** contains source IP address, destination IP address, reserved field, protocol number(6 for TCP) and TCP segment length.
- The **TCP Header** contains the source port, destination port, sequence number, acknowledgment number, header length, reserved bits, flags, window size and checksum.
- The **TCP Data** contains the Data Payload value in hexadecimal.

TCP (Transmission Control Protocol) uses a checksum mechanism to ensure the integrity of the transmitted data. The TCP checksum is a 16-bit value calculated over the TCP header,TCP Pseudo Header and data. It is used by the receiving end to verify that the data received is the same as the data sent by the sender.

Here's how the TCP checksum is calculated:

1. The TCP header and data are divided into 16-bit words (2 bytes).

2. All the 16-bit words are added together, including a pseudo-header that contains the source and destination IP addresses, protocol number (6 for TCP), and the TCP length.
3. If there is an odd number of bytes, a zero byte is appended to the data before the checksum calculation.
4. The checksum is then obtained by taking the one's complement of the sum.
5. The calculated checksum is placed in the TCP header.

When the receiver receives the TCP segment, it performs the same checksum calculation over the received data, including the TCP header. If the calculated checksum matches the value in the received TCP header, it indicates that the data has been received without any errors. If the checksum doesn't match, it suggests that the data may have been corrupted during transmission, and the receiver discards the segment.

The TCP checksum provides a basic level of error detection and helps identify transmission errors .

SERVER CODE: (tcp_server.py)

```
import socket

# Create a socket object

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Define the host and port

host = 'localhost'

port = 12345

# Bind the socket to the host and port

server_socket.bind((host, port))


# Listen for incoming connections

server_socket.listen(1)

print("Server is listening on {}:{}".format(host, port))

while True:

    # Accept a client connection

    client_socket, address = server_socket.accept()

    print("Connection established with {}:{}".format(address[0], address[1]))


    # Receive data from the client

    name = client_socket.recv(1024).decode()
    reg_number = client_socket.recv(1024).decode()

    # Print the received data
    print("DATA:", name +reg_number)
    # Close the connection with the client
    client_socket.close()
    print("Connection closed with {}:{}".format(address[0], address[1]))
```

CLIENT CODE: (tcp_client.py)

```
import socket
# Create a socket object
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Define the host and port
host = 'localhost'
port = 12345
# Connect to the server
client_socket.connect((host, port))
# Get user input
name = input("Enter your name: ")
reg_number = input("Enter your registration number: ")
# Create a string with name and registration number separated by a comma
data = name + reg_number
# Send the data to the server
client_socket.send(data.encode())
# Close the connection with the server
client_socket.close()
```

TCP CHECKSUM CODE: (tcp_checksum.py)

```
def convert_decimal_to_binary(decimal, num_bits):
    # Convert decimal to binary
    binary = bin(decimal)[2:].zfill(num_bits)
    return binary
```

```
def calculate_checksum(source_ip, dest_ip, reserved1, protocol, tcp_length,source_port, dest_port,
sequence, ack, hlen, reserved2,flags, window, checksum_in, urgent_pointer, tcp_data):
    # Convert IP addresses to binary strings
    source_ip_binary = convert_decimal_to_binary(source_ip, 32)
    dest_ip_binary = convert_decimal_to_binary(dest_ip, 32)

    # Convert other inputs to binary strings
    protocol = convert_decimal_to_binary(protocol, 8)
    source_port = convert_decimal_to_binary(source_port, 16)
    dest_port = convert_decimal_to_binary(dest_port, 16)
    tcp_length = convert_decimal_to_binary(tcp_length, 16)
    sequence = convert_decimal_to_binary(sequence, 32)
    ack = convert_decimal_to_binary(ack, 32)
    hlen = convert_decimal_to_binary(hlen, 4)
    window = convert_decimal_to_binary(window, 16)
    reserved1 = convert_decimal_to_binary(reserved1, 8)
    reserved2 = convert_decimal_to_binary(reserved2, 6)
    flags = convert_decimal_to_binary(flags, 6)
    checksum_in = convert_decimal_to_binary(checksum_in, 16)
    urgent_pointer = convert_decimal_to_binary(urgent_pointer, 16)

    # Convert TCP data from hexadecimal to binary
    tcp_data_binary = bin(int(tcp_data, 16))[2:].zfill(len(tcp_data) * 4)

    # Pad TCP data with zeros to make its length a multiple of 16
    if len(tcp_data_binary) % 16 != 0:
        tcp_data_binary += '0' * (16 - len(tcp_data_binary) % 16)

    # Concatenate all binary strings
    message = source_ip_binary + dest_ip_binary + reserved1 + protocol + tcp_length + source_port + dest_port + sequence + ack + hlen + reserved2 + flags + window + checksum_in + urgent_pointer + tcp_data_binary

    # Perform one's complement addition
    checksum = 0
    while len(message) >= 16:
        value = int(message[:16], 2)
        checksum += value
        message = message[16:]
        if len(message) < 16:
            break

    # Add the remaining 16-bit value if present
    if len(message) > 0:
        value = int(message, 2)
        checksum += value

    # Fold 1's complement carry
    while checksum >> 16:
        checksum = (checksum & 0xFFFF) + (checksum >> 16)

    # Take one's complement
    checksum = checksum ^ 0xFFFF
    return hex(checksum)[2:].zfill(4).upper()

# Example usage
source_ip_input = input("Enter source IP address (decimal dotted format): ")
dest_ip_input = input("Enter destination IP address (decimal dotted format): ")
reserved1_input = int(input("Enter the reserved bits (in decimal): "))
protocol_input = int(input("Enter protocol number (in decimal): "))
tcp_length_input = int(input("Enter TCP segment length (in decimal): "))
```

```
source_port_input = int(input("Enter source port number (in decimal): "))
dest_port_input = int(input("Enter destination port number (in decimal): "))
sequence_input = int(input("Enter the sequence number (in decimal): "))
ack_input = int(input("Enter the acknowledgment number (in decimal): "))
hlen_input = int(input("Enter the header length (in decimal): "))
reserved2_input = int(input("Enter the reserved bits (in decimal): "))
flags_input = int(input("Enter the flag bits (in decimal): "))
window_input = int(input("Enter the window size (in decimal): "))
checksum_in_input = int(input("Enter the existing checksum (in decimal): "))
urgent_pointer_input = int(input("Enter the urgent pointer (in decimal): "))
tcp_data_input = input("Enter TCP data (in hexadecimal): ")

# Convert IP addresses from decimal dotted format to decimal
source_ip = int(''.join(format(int(x), '08b') for x in source_ip_input.split('.')), 2)
dest_ip = int(''.join(format(int(x), '08b') for x in dest_ip_input.split('.')), 2)

checksum = calculate_checksum(source_ip, dest_ip, reserved1_input,
protocol_input,tcp_length_input, source_port_input, dest_port_input,sequence_input, ack_input,
hlen_input, reserved2_input,flags_input, window_input, checksum_in_input,urgent_pointer_input,
tcp_data_input)

print("Calculated checksum:", checksum)
```

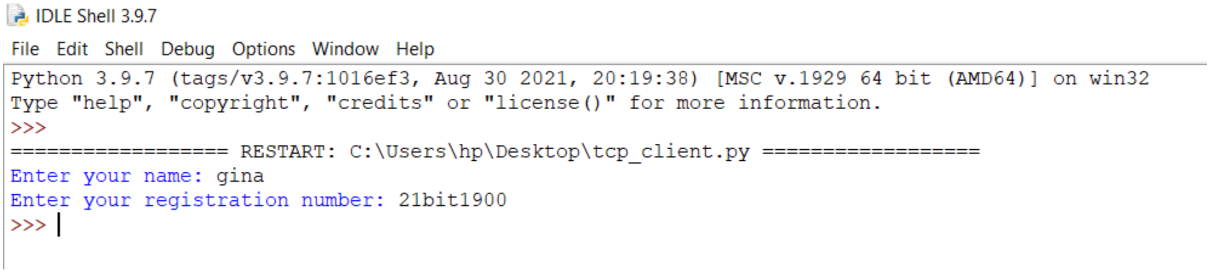
SCREENSHOTS:

1) Frame 98

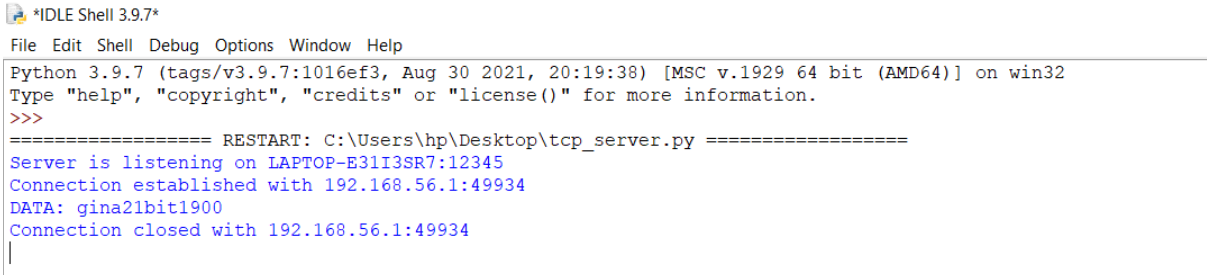
Enter your name: gina

Enter your registration number: 21bit1900

Combining the name and registration number and storing it in DATA variable



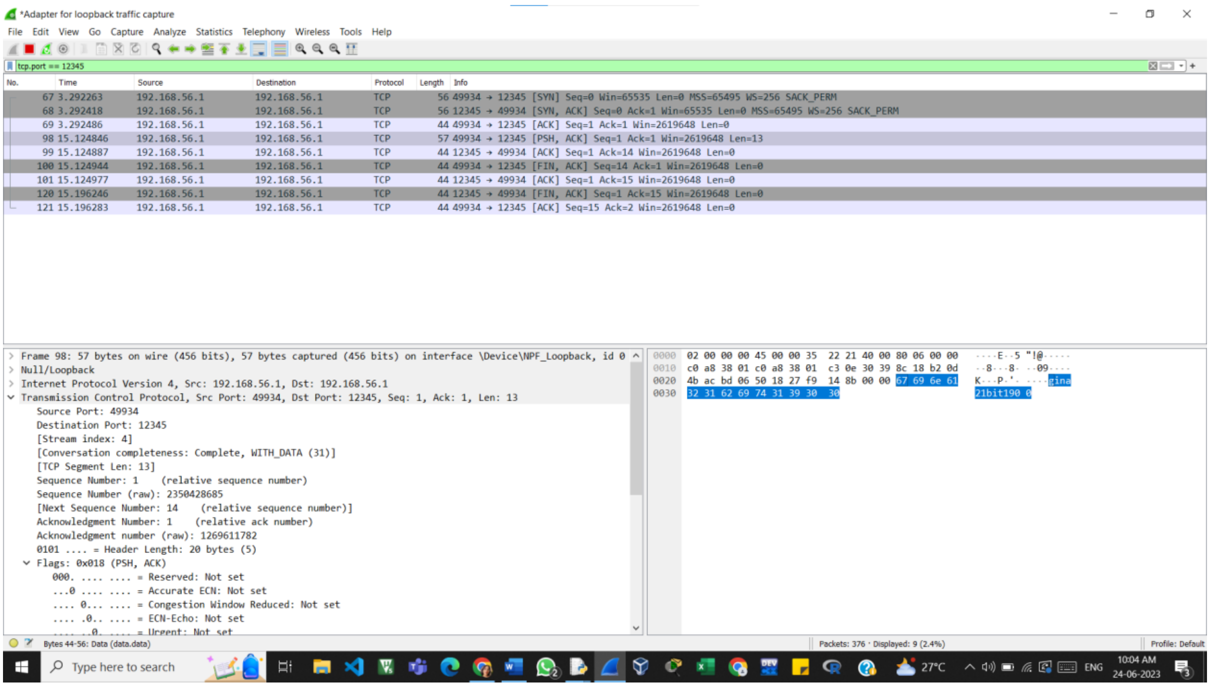
```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:\Users\hp\Desktop\tcp_client.py ==================
Enter your name: gina
Enter your registration number: 21bit1900
>>> |
```



```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:\Users\hp\Desktop\tcp_server.py ==================
Server is listening on LAPTOP-E31I3SR7:12345
Connection established with 192.168.56.1:49934
DATA: gina21bit1900
Connection closed with 192.168.56.1:49934
```

In Wireshark we use the **adapter for loopback traffic for capture** and then run the server and client, a connection is established then we get the necessary frames and select the frame which contains the input data **gina21bit1900** . As shown in the wireshark capture below **frame:98**

Message in wireshark is gina21bit1900



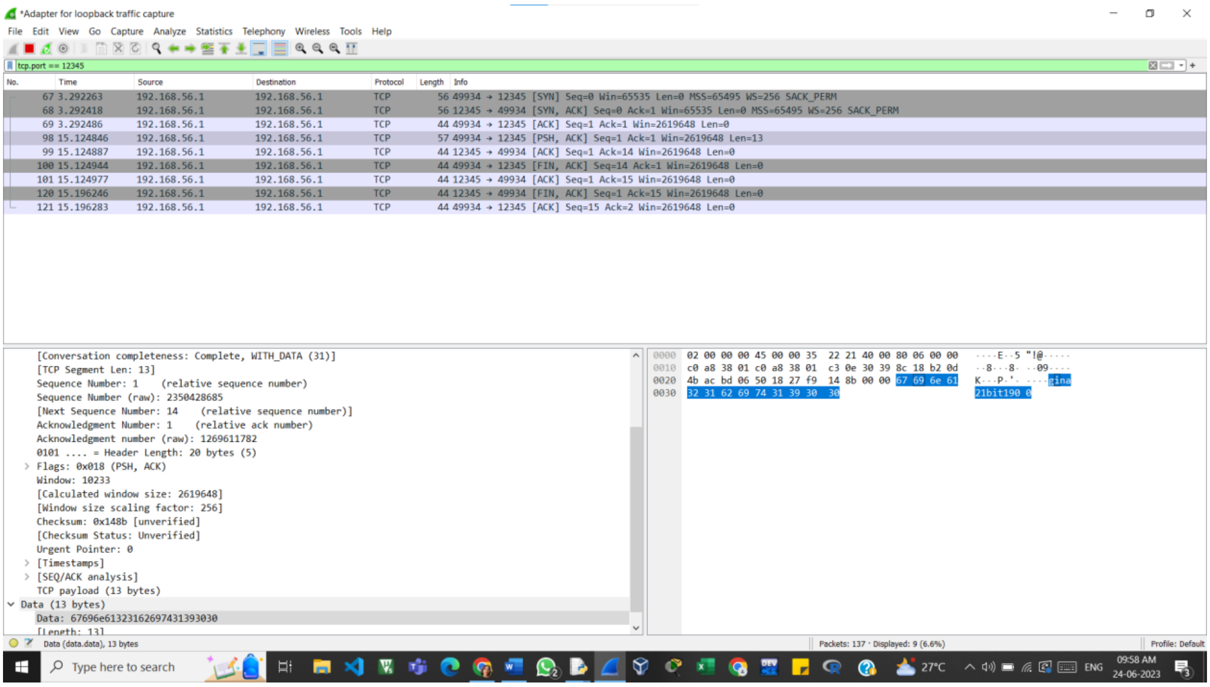
*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.port == 12345

No.   Time        Source          Destination     Protocol  Length  Info
   67 3.292263    192.168.56.1    192.168.56.1    TCP       56 49934 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   68 3.292418    192.168.56.1    192.168.56.1    TCP       56 12345 → 49934 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   69 3.292486    192.168.56.1    192.168.56.1    TCP       44 49934 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
   98 15.124846   192.168.56.1    192.168.56.1    TCP       57 49934 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=13
   99 15.124887   192.168.56.1    192.168.56.1    TCP       44 12345 → 49934 [ACK] Seq=1 Ack=14 Win=2619648 Len=0
  100 15.124944   192.168.56.1    192.168.56.1    TCP       44 49934 → 12345 [FIN, ACK] Seq=14 Ack=1 Win=2619648 Len=0
  101 15.124977   192.168.56.1    192.168.56.1    TCP       44 12345 → 49934 [ACK] Seq=1 Ack=15 Win=2619648 Len=0
  120 15.196246   192.168.56.1    192.168.56.1    TCP       44 12345 → 49934 [FIN, ACK] Seq=1 Ack=15 Win=2619648 Len=0
  121 15.196283   192.168.56.1    192.168.56.1    TCP       44 49934 → 12345 [ACK] Seq=15 Ack=2 Win=2619648 Len=0
```

```
> Frame 98: 57 bytes on wire (456 bits), 57 bytes captured (456 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 192.168.56.1, Dst: 192.168.56.1
v Transmission Control Protocol, Src Port: 49934, Dst Port: 12345, Seq: 1, Ack: 1, Len: 13
    Source Port: 49934
    Destination Port: 12345
    [Stream index: 4]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 13]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 2350428685
    [Next Sequence Number: 14    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 1269611782
    0101 .... = Header Length: 20 bytes (5)
  v Flags: 0x018 (PSH, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Accurate ECN: Not set
      .... 0... .... = Congestion Window Reduced: Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
```

```
0000  02 00 00 00 45 00 00 35  22 21 40 00 80 06 00 00   ····E··5 "!@·····
0010  c0 a8 38 01 c0 a8 38 01  c3 0e 30 39 8c 18 b2 0d   ··8···8· ··09····
0020  4b ac bd 06 50 18 27 f9  14 8b 00 00 67 69 6e 61   K···P·'· ····gina
0030  32 31 62 69 74 31 39 30  30                        21bit190 0
```

Bytes 44-56: Data (data.data)
Packets: 376 · Displayed: 9 (2.4%)
Profile: Default
Type here to search
27°C
ENG
10:04 AM
24-06-2023


*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.port == 12345

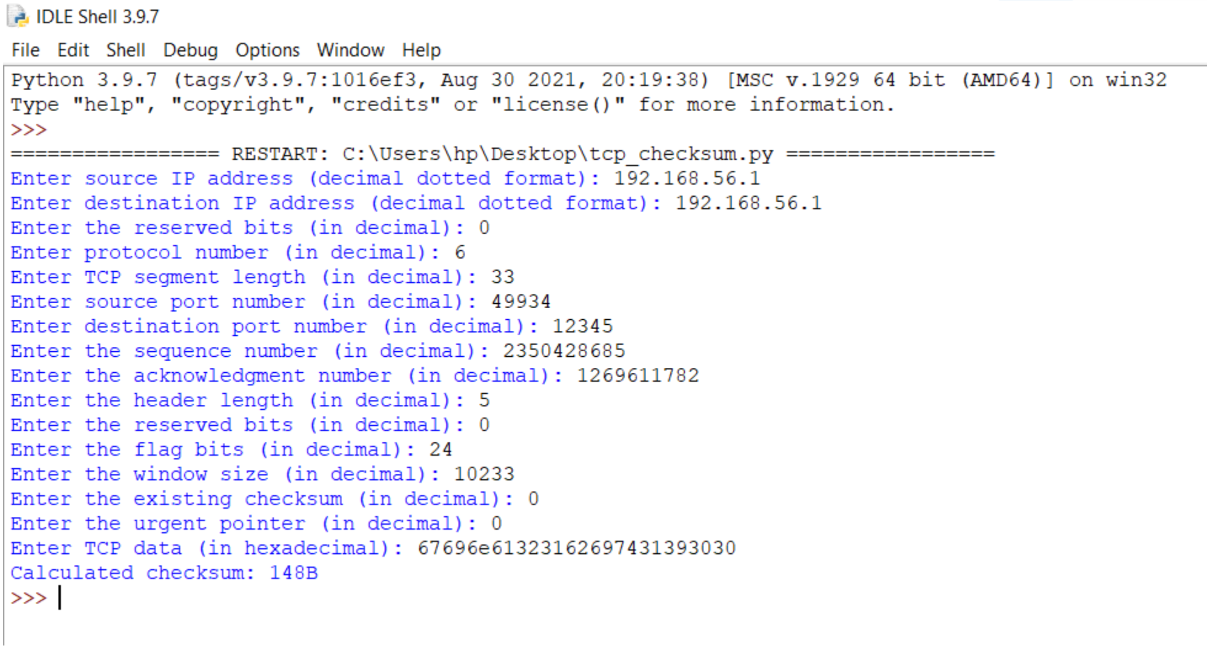
```
No.   Time          Source          Destination     Protocol  Length  Info
   67 3.292263      192.168.56.1    192.168.56.1    TCP        56 49934 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   68 3.292418      192.168.56.1    192.168.56.1    TCP        56 12345 → 49934 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   69 3.292486      192.168.56.1    192.168.56.1    TCP        44 49934 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
   98 15.124846     192.168.56.1    192.168.56.1    TCP        57 49934 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=13
   99 15.124887     192.168.56.1    192.168.56.1    TCP        44 12345 → 49934 [ACK] Seq=1 Ack=14 Win=2619648 Len=0
  100 15.124944     192.168.56.1    192.168.56.1    TCP        44 49934 → 12345 [FIN, ACK] Seq=14 Ack=1 Win=2619648 Len=0
  101 15.124977     192.168.56.1    192.168.56.1    TCP        44 12345 → 49934 [ACK] Seq=1 Ack=15 Win=2619648 Len=0
  120 15.196246     192.168.56.1    192.168.56.1    TCP        44 12345 → 49934 [FIN, ACK] Seq=1 Ack=15 Win=2619648 Len=0
  121 15.196283     192.168.56.1    192.168.56.1    TCP        44 49934 → 12345 [ACK] Seq=15 Ack=2 Win=2619648 Len=0
```

```
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 13]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 2350428685
    [Next Sequence Number: 14    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 1269611782
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 10233
    [Calculated window size: 2619648]
    [Window size scaling factor: 256]
    Checksum: 0x148b [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
    TCP payload (13 bytes)
v Data (13 bytes)
    Data: 67696e61323162697431393030
    [Length: 13]
```

```
0000  02 00 00 00 45 00 00 35  22 21 40 00 80 06 00 00   ····E··5 "!@·····
0010  c0 a8 38 01 c0 a8 38 01  c3 0e 30 39 8c 18 b2 0d   ··8···8· ··09····
0020  4b ac bd 06 50 18 27 f9  14 8b 00 00 67 69 6e 61   K···P·'· ····gina
0030  32 31 62 69 74 31 39 30  30                        21bit190 0
```

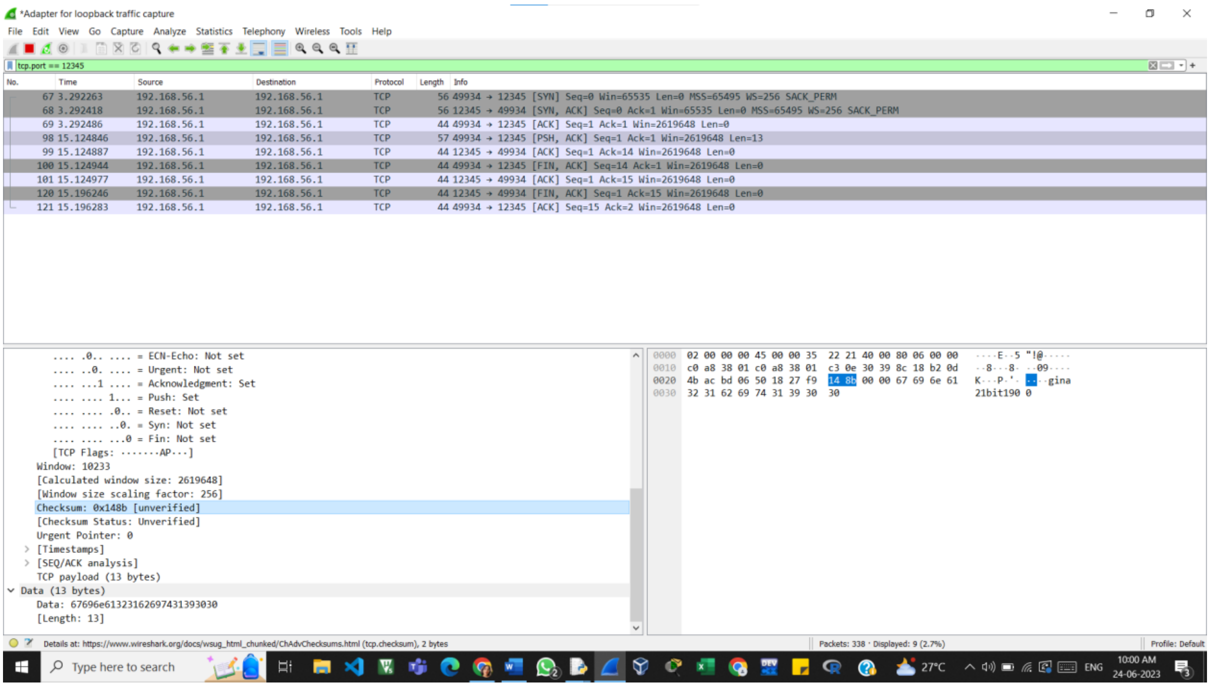
Data (data.data), 13 bytes
Packets: 137 · Displayed: 9 (6.6%)
Profile: Default
Type here to search
27°C
ENG
09:58 AM
24-06-2023

The tcp checksum in senders side is calculated and the value is 148B



```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: C:\Users\hp\Desktop\tcp_checksum.py =================
Enter source IP address (decimal dotted format): 192.168.56.1
Enter destination IP address (decimal dotted format): 192.168.56.1
Enter the reserved bits (in decimal): 0
Enter protocol number (in decimal): 6
Enter TCP segment length (in decimal): 33
Enter source port number (in decimal): 49934
Enter destination port number (in decimal): 12345
Enter the sequence number (in decimal): 2350428685
Enter the acknowledgment number (in decimal): 1269611782
Enter the header length (in decimal): 5
Enter the reserved bits (in decimal): 0
Enter the flag bits (in decimal): 24
Enter the window size (in decimal): 10233
Enter the existing checksum (in decimal): 0
Enter the urgent pointer (in decimal): 0
Enter TCP data (in hexadecimal): 67696e61323162697431393030
Calculated checksum: 148B
>>> |
```

Wireshark checksum is also shown which is : 0x148b



*Adapter for loopback traffic capture

tcp.port == 12345

```
No.   Time        Source          Destination     Protocol  Length  Info
   67 3.292263    192.168.56.1    192.168.56.1    TCP       56 49934 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   68 3.292418    192.168.56.1    192.168.56.1    TCP       56 12345 → 49934 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   69 3.292486    192.168.56.1    192.168.56.1    TCP       44 49934 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
   98 15.124846   192.168.56.1    192.168.56.1    TCP       57 49934 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=13
   99 15.124887   192.168.56.1    192.168.56.1    TCP       44 12345 → 49934 [ACK] Seq=1 Ack=14 Win=2619648 Len=0
  100 15.124944   192.168.56.1    192.168.56.1    TCP       44 49934 → 12345 [FIN, ACK] Seq=14 Ack=1 Win=2619648 Len=0
  101 15.124977   192.168.56.1    192.168.56.1    TCP       44 12345 → 49934 [ACK] Seq=1 Ack=15 Win=2619648 Len=0
  120 15.196246   192.168.56.1    192.168.56.1    TCP       44 12345 → 49934 [FIN, ACK] Seq=1 Ack=15 Win=2619648 Len=0
  121 15.196283   192.168.56.1    192.168.56.1    TCP       44 49934 → 12345 [ACK] Seq=15 Ack=2 Win=2619648 Len=0
```

```
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·······AP···]
    Window: 10233
    [Calculated window size: 2619648]
    [Window size scaling factor: 256]
    Checksum: 0x148b [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
    TCP payload (13 bytes)
v Data (13 bytes)
    Data: 67696e61323162697431393030
    [Length: 13]
```

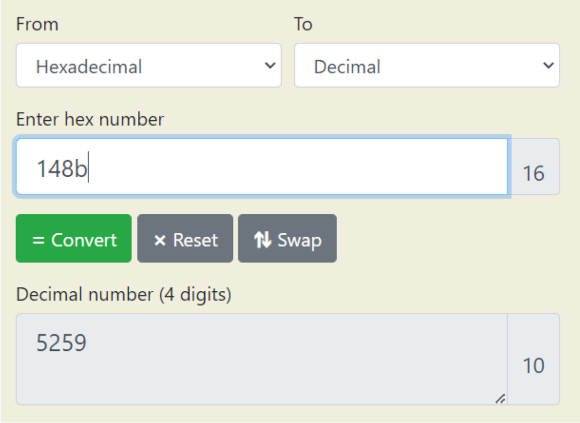
```
0000  02 00 00 00 45 00 00 35  22 21 40 00 80 06 00 00   ····E··5 "!@·····
0010  c0 a8 38 01 c0 a8 38 01  c3 0e 30 39 8c 18 b2 0d   ··8···8· ··09····
0020  4b ac bd 06 50 18 27 f9  14 8b 00 00 67 69 6e 61   K···P·'· ····gina
0030  32 31 62 69 74 31 39 30  30                        21bit190 0
```

Details at: https://www.wireshark.org/docs/wsug_html_chunked/ChAdvChecksums.html (tcp.checksum), 2 bytes

Packets: 338 · Displayed: 9 (2.7%)

Profile: Default

Now the tcp checksum gives recievers side checksum which is calculated as :

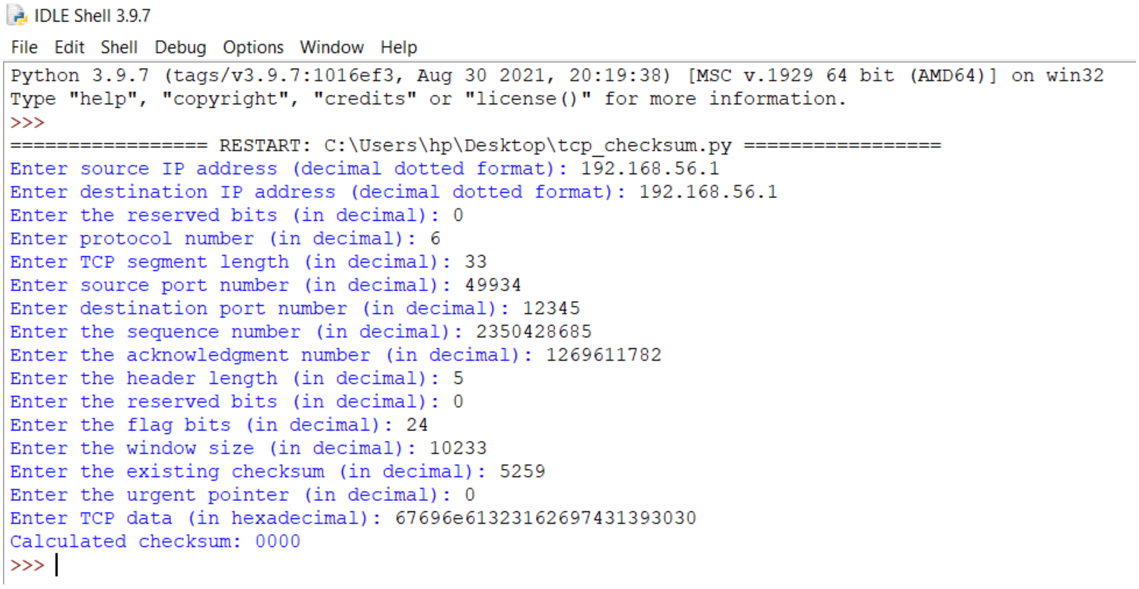


From: Hexadecimal

To: Decimal

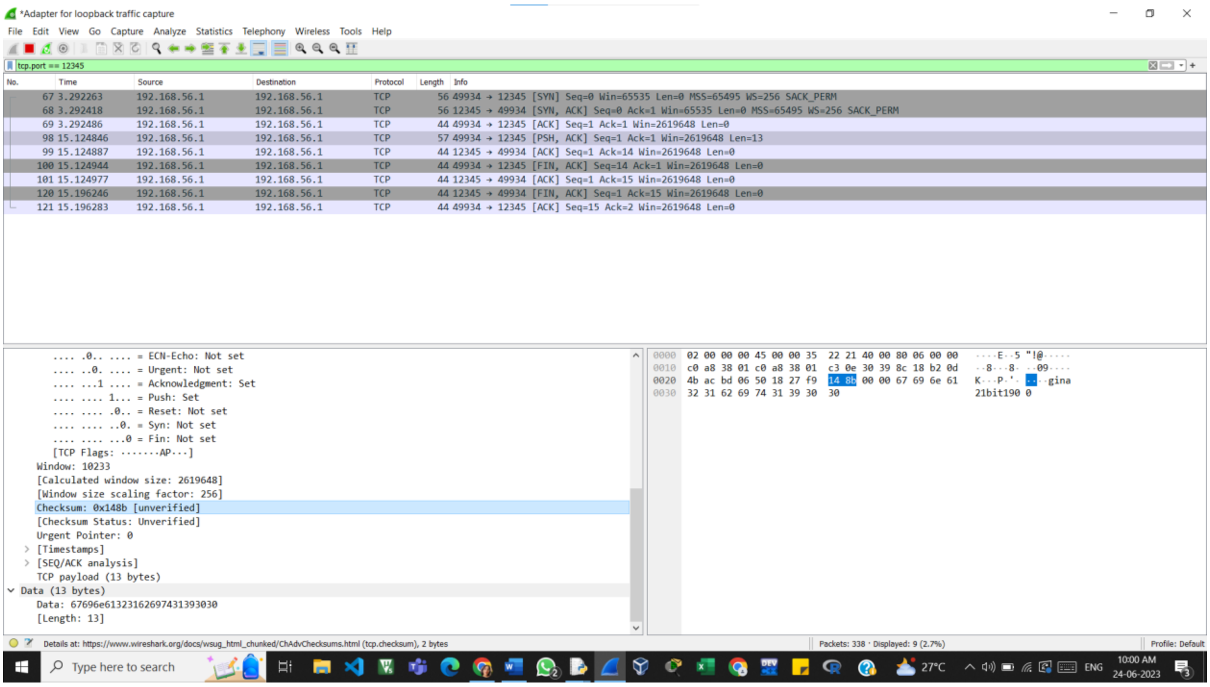
Enter hex number: 148b (16)

Decimal number (4 digits): 5259 (10)



```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: C:\Users\hp\Desktop\tcp_checksum.py =================
Enter source IP address (decimal dotted format): 192.168.56.1
Enter destination IP address (decimal dotted format): 192.168.56.1
Enter the reserved bits (in decimal): 0
Enter protocol number (in decimal): 6
Enter TCP segment length (in decimal): 33
Enter source port number (in decimal): 49934
Enter destination port number (in decimal): 12345
Enter the sequence number (in decimal): 2350428685
Enter the acknowledgment number (in decimal): 1269611782
Enter the header length (in decimal): 5
Enter the reserved bits (in decimal): 0
Enter the flag bits (in decimal): 24
Enter the window size (in decimal): 10233
Enter the existing checksum (in decimal): 5259
Enter the urgent pointer (in decimal): 0
Enter TCP data (in hexadecimal): 67696e6132316269743139303
Calculated checksum: 0000
>>> |
```

Wireshark checksum is also shown which is :


*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.port == 12345
```

```
No.   Time        Source          Destination     Protocol Length Info
   67 3.292263    192.168.56.1    192.168.56.1    TCP      56 49934 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   68 3.292418    192.168.56.1    192.168.56.1    TCP      56 12345 → 49934 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   69 3.292486    192.168.56.1    192.168.56.1    TCP      44 49934 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
   98 15.124846   192.168.56.1    192.168.56.1    TCP      57 49934 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=13
   99 15.124887   192.168.56.1    192.168.56.1    TCP      44 12345 → 49934 [ACK] Seq=1 Ack=14 Win=2619648 Len=0
  100 15.124944   192.168.56.1    192.168.56.1    TCP      44 49934 → 12345 [FIN, ACK] Seq=14 Ack=1 Win=2619648 Len=0
  101 15.124977   192.168.56.1    192.168.56.1    TCP      44 12345 → 49934 [ACK] Seq=1 Ack=15 Win=2619648 Len=0
  120 15.196246   192.168.56.1    192.168.56.1    TCP      44 12345 → 49934 [FIN, ACK] Seq=1 Ack=15 Win=2619648 Len=0
  121 15.196283   192.168.56.1    192.168.56.1    TCP      44 49934 → 12345 [ACK] Seq=15 Ack=2 Win=2619648 Len=0
```

```
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·······AP···]
      Window: 10233
      [Calculated window size: 2619648]
      [Window size scaling factor: 256]
      Checksum: 0x148b [unverified]
      [Checksum Status: Unverified]
      Urgent Pointer: 0
    > [Timestamps]
    > [SEQ/ACK analysis]
      TCP payload (13 bytes)
  ∨ Data (13 bytes)
      Data: 67696e61323162697431393030
      [Length: 13]
```

```
0000  02 00 00 00 45 00 00 35  22 21 40 00 80 06 00 00   ····E··5 "!@·····
0010  c0 a8 38 01 c0 a8 38 01  c3 0e 30 39 8c 18 b2 0d   ··8···8· ··09····
0020  4b ac bd 06 50 18 27 f9  14 8b 00 00 67 69 6e 61   K···P·'· ····gina
0030  32 31 62 69 74 31 39 30  30                        21bit190 0
```

Details at: https://www.wireshark.org/docs/wsug_html_chunked/ChAdvChecksums.html (tcp.checksum), 2 bytes

Packets: 338 · Displayed: 9 (2.7%)

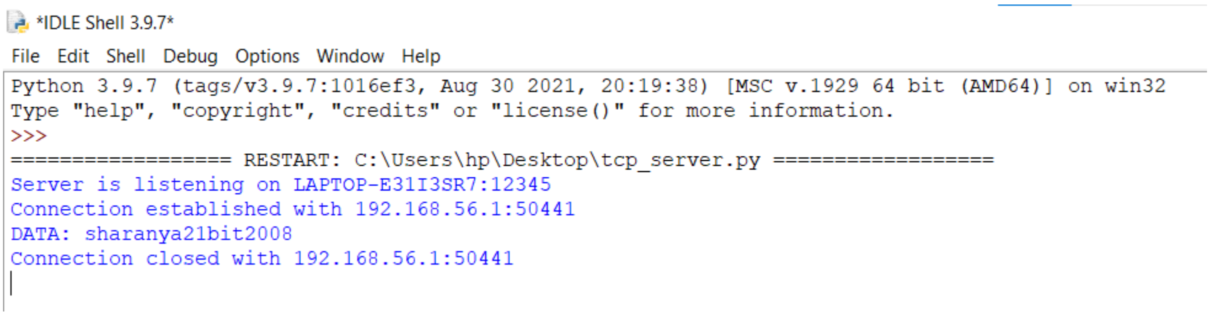
Profile: Default

2) Frame 98

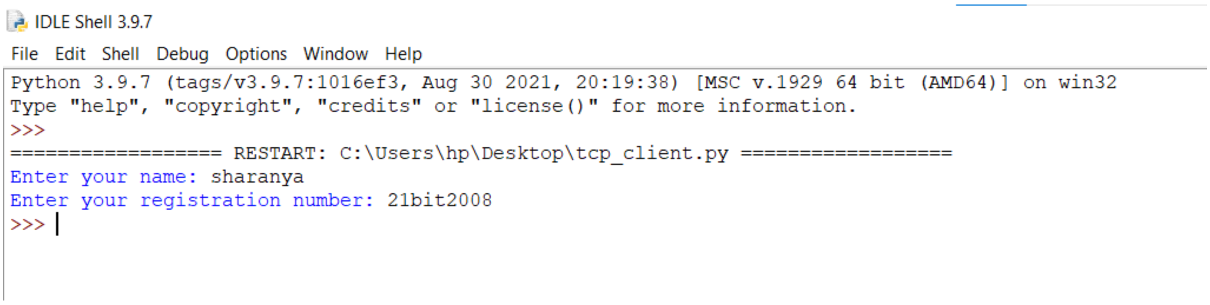
Enter your name: sharanya

Enter your registration number: 21bit2008

Combining the name and registration number and storing it in DATA variable



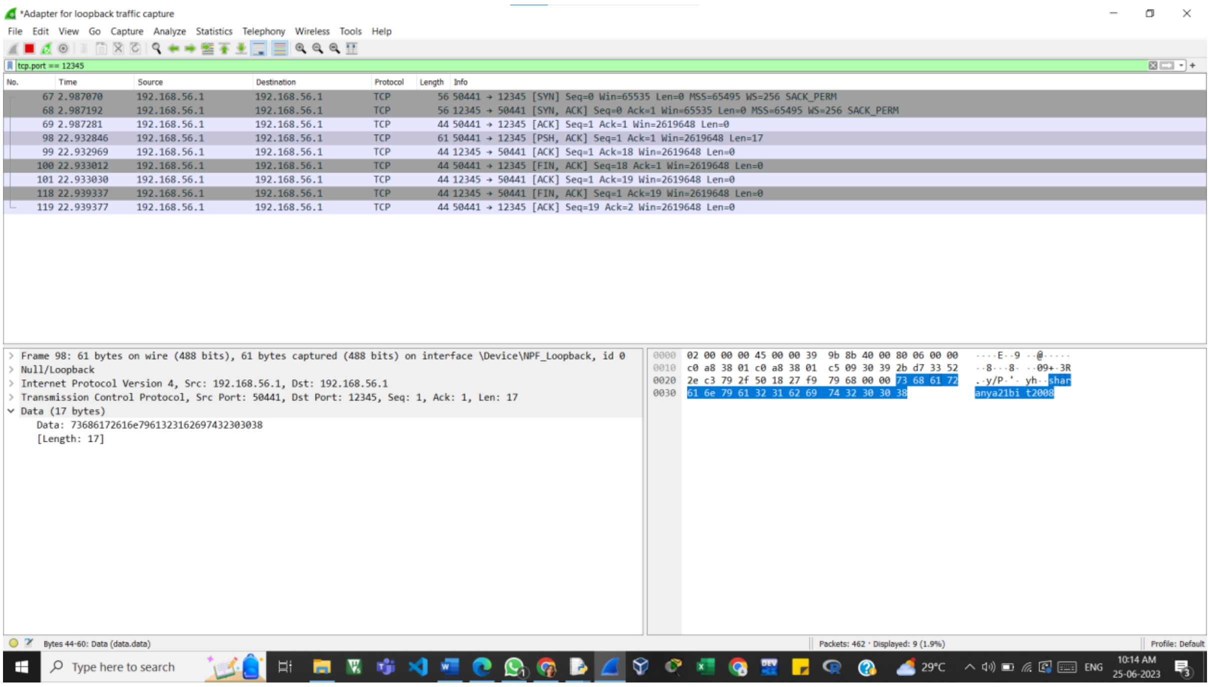
```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:\Users\hp\Desktop\tcp_server.py ==================
Server is listening on LAPTOP-E31I3SR7:12345
Connection established with 192.168.56.1:50441
DATA: sharanya21bit2008
Connection closed with 192.168.56.1:50441
```



```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:\Users\hp\Desktop\tcp_client.py ==================
Enter your name: sharanya
Enter your registration number: 21bit2008
>>>
```

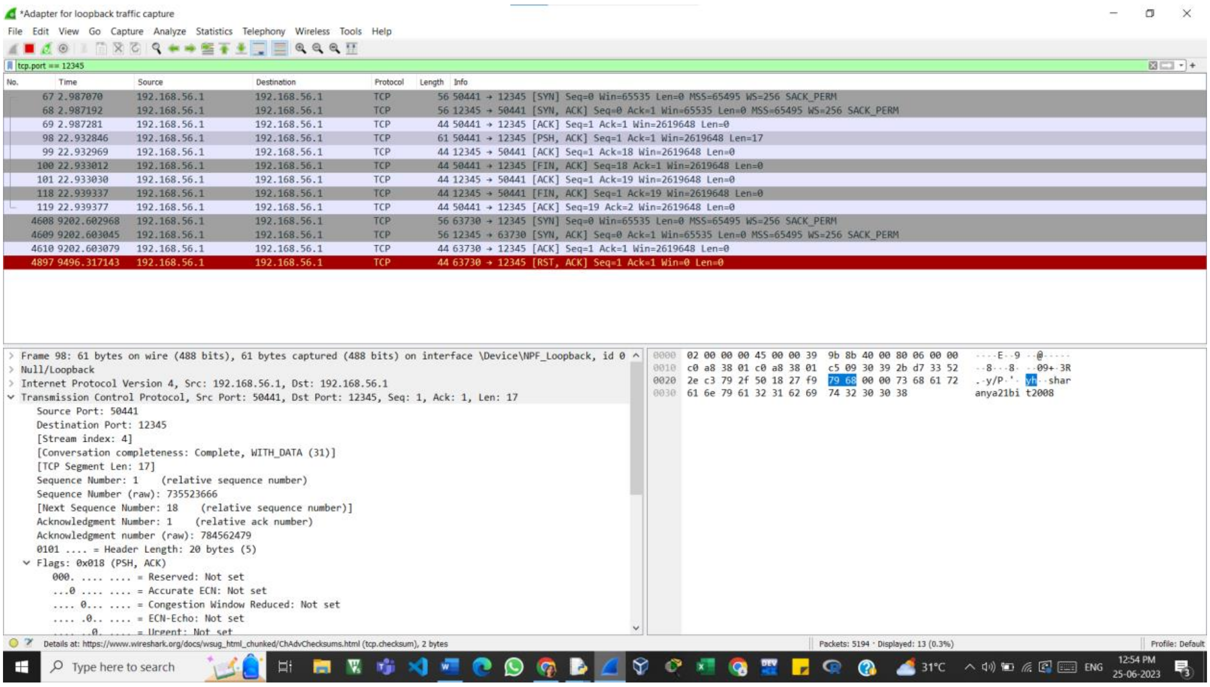
In Wireshark we use the **adapter for loopback traffic for capture** and then run the server and client, a connection is established then we get the necessary frames and select the frame which contains the input data **sharanya21bit2008** . As shown in the wireshark capture below **frame:98**

Message in wireshark is sharanya21bit2008



*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.port == 12345

No.  Time         Source          Destination     Protocol Length Info
 67 2.987070      192.168.56.1    192.168.56.1    TCP      56 50441 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 68 2.987192      192.168.56.1    192.168.56.1    TCP      56 12345 → 50441 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 69 2.987281      192.168.56.1    192.168.56.1    TCP      44 50441 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
 98 22.932846     192.168.56.1    192.168.56.1    TCP      61 50441 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=17
 99 22.932969     192.168.56.1    192.168.56.1    TCP      44 12345 → 50441 [ACK] Seq=1 Ack=18 Win=2619648 Len=0
100 22.933012     192.168.56.1    192.168.56.1    TCP      44 50441 → 12345 [FIN, ACK] Seq=18 Ack=1 Win=2619648 Len=0
101 22.933030     192.168.56.1    192.168.56.1    TCP      44 12345 → 50441 [ACK] Seq=1 Ack=19 Win=2619648 Len=0
118 22.939337     192.168.56.1    192.168.56.1    TCP      44 12345 → 50441 [FIN, ACK] Seq=1 Ack=19 Win=2619648 Len=0
119 22.939377     192.168.56.1    192.168.56.1    TCP      44 50441 → 12345 [ACK] Seq=19 Ack=2 Win=2619648 Len=0

> Frame 98: 61 bytes on wire (488 bits), 61 bytes captured (488 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 192.168.56.1, Dst: 192.168.56.1
> Transmission Control Protocol, Src Port: 50441, Dst Port: 12345, Seq: 1, Ack: 1, Len: 17
v Data (17 bytes)
    Data: 73686172616e7961323162697432303038
    [Length: 17]

0000  02 00 00 00 45 00 00 39  9b 8b 40 00 80 06 00 00   ····E··9 ··@·····
0010  c0 a8 38 01 c0 a8 38 01  c5 09 30 39 2b d7 33 52   ··8···8· ··09+·3R
0020  2e c3 79 2f 50 18 27 f9  79 68 00 00 73 68 61 72   .·y/P·'· yh··shar
0030  61 6e 79 61 32 31 62 69  74 32 30 30 38            anya21bi t2008

Bytes 44-60: Data (data.data)    Packets: 462 · Displayed: 9 (1.9%)    Profile: Default
```


*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Type here to search
31°C
ENG
12:54 PM
25-06-2023

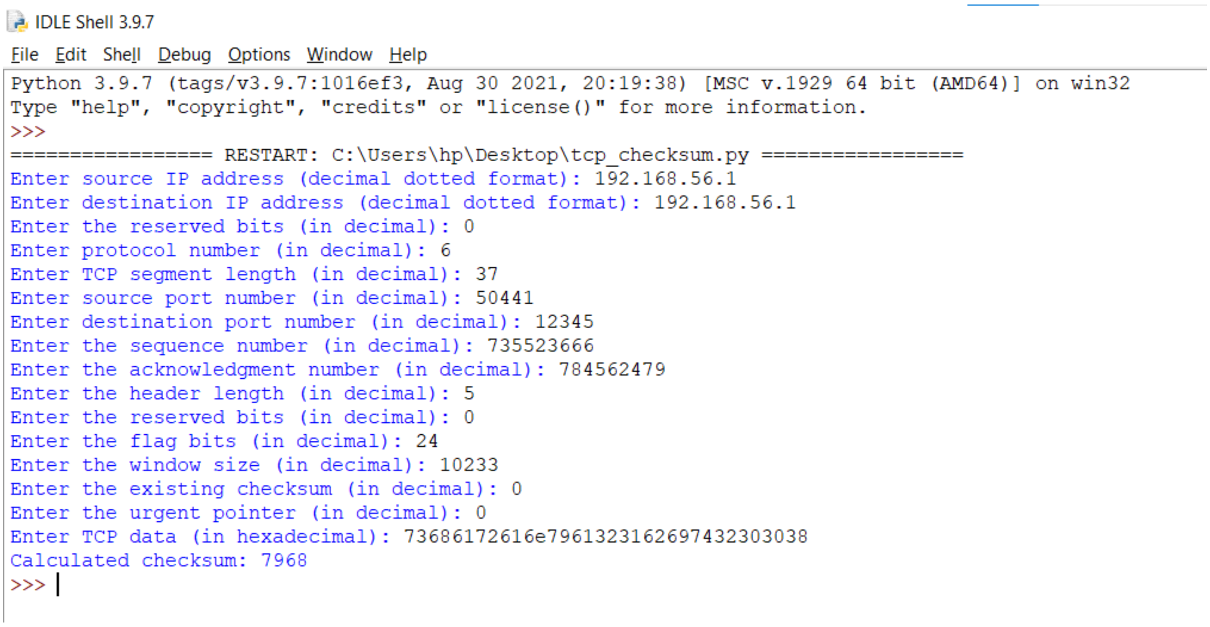
```
tcp.port == 12345

No.   Time            Source          Destination     Protocol  Length  Info
   67 2.987070        192.168.56.1    192.168.56.1    TCP       56 50441 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   68 2.987192        192.168.56.1    192.168.56.1    TCP       56 12345 → 50441 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   69 2.987281        192.168.56.1    192.168.56.1    TCP       44 50441 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
   98 22.932846       192.168.56.1    192.168.56.1    TCP       61 50441 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=17
   99 22.932969       192.168.56.1    192.168.56.1    TCP       44 12345 → 50441 [ACK] Seq=1 Ack=18 Win=2619648 Len=0
  100 22.933012       192.168.56.1    192.168.56.1    TCP       44 50441 → 12345 [FIN, ACK] Seq=18 Ack=1 Win=2619648 Len=0
  101 22.933030       192.168.56.1    192.168.56.1    TCP       44 12345 → 50441 [ACK] Seq=1 Ack=19 Win=2619648 Len=0
  118 22.939337       192.168.56.1    192.168.56.1    TCP       44 12345 → 50441 [FIN, ACK] Seq=1 Ack=19 Win=2619648 Len=0
  119 22.939377       192.168.56.1    192.168.56.1    TCP       44 50441 → 12345 [ACK] Seq=19 Ack=2 Win=2619648 Len=0
 4608 9202.602968     192.168.56.1    192.168.56.1    TCP       56 63730 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 4609 9202.603045     192.168.56.1    192.168.56.1    TCP       56 12345 → 63730 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 4610 9202.603079     192.168.56.1    192.168.56.1    TCP       44 63730 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
 4897 9496.317143     192.168.56.1    192.168.56.1    TCP       44 63730 → 12345 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

```
> Frame 98: 61 bytes on wire (488 bits), 61 bytes captured (488 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 192.168.56.1, Dst: 192.168.56.1
v Transmission Control Protocol, Src Port: 50441, Dst Port: 12345, Seq: 1, Ack: 1, Len: 17
    Source Port: 50441
    Destination Port: 12345
    [Stream index: 4]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 17]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 735523666
    [Next Sequence Number: 18    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 784562479
    0101 .... = Header Length: 20 bytes (5)
  v Flags: 0x018 (PSH, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Accurate ECN: Not set
      .... 0... .... = Congestion Window Reduced: Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
```

```
0000  02 00 00 00 45 00 00 39  9b 8b 40 00 80 06 00 00   ····E··9 ··@·····
0010  c0 a8 38 01 c0 a8 38 01  c5 09 30 39 2b d7 33 52   ··8···8· ··09+·3R
0020  2e c3 79 2f 50 18 27 f9  79 68 00 00 73 68 61 72   .·y/P·'· yh··shar
0030  61 6e 79 61 32 31 62 69  74 32 30 30 38            anya21bi t2008
```

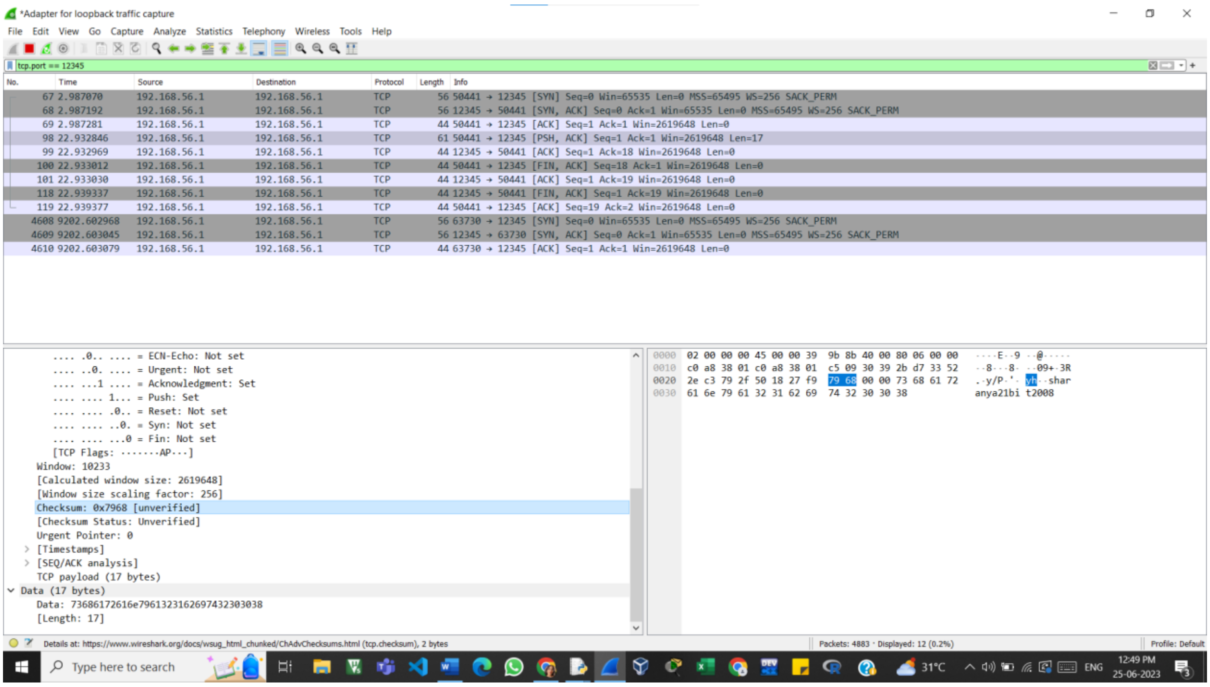
```
Details at: https://www.wireshark.org/docs/wsug_html_chunked/ChAdvChecksums.html (tcp.checksum), 2 bytes
Packets: 5194 · Displayed: 13 (0.3%)
Profile: Default
```

The tcp checksum in senders side is calculated and the value is 7968



```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: C:\Users\hp\Desktop\tcp_checksum.py =================
Enter source IP address (decimal dotted format): 192.168.56.1
Enter destination IP address (decimal dotted format): 192.168.56.1
Enter the reserved bits (in decimal): 0
Enter protocol number (in decimal): 6
Enter TCP segment length (in decimal): 37
Enter source port number (in decimal): 50441
Enter destination port number (in decimal): 12345
Enter the sequence number (in decimal): 735523666
Enter the acknowledgment number (in decimal): 784562479
Enter the header length (in decimal): 5
Enter the reserved bits (in decimal): 0
Enter the flag bits (in decimal): 24
Enter the window size (in decimal): 10233
Enter the existing checksum (in decimal): 0
Enter the urgent pointer (in decimal): 0
Enter TCP data (in hexadecimal): 73686172616e796132316269743230303038
Calculated checksum: 7968
>>> |
```

Wireshark checksum is also shown which is : 0x7968



*Adapter for loopback traffic capture

tcp.port == 12345

```
No.  Time          Source        Destination   Protocol Length Info
  67 2.987070      192.168.56.1  192.168.56.1  TCP      56 50441 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  68 2.987192      192.168.56.1  192.168.56.1  TCP      56 12345 → 50441 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  69 2.987281      192.168.56.1  192.168.56.1  TCP      44 50441 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
  98 22.932846     192.168.56.1  192.168.56.1  TCP      61 50441 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=17
  99 22.932969     192.168.56.1  192.168.56.1  TCP      44 12345 → 50441 [ACK] Seq=1 Ack=18 Win=2619648 Len=0
 100 22.933012     192.168.56.1  192.168.56.1  TCP      44 50441 → 12345 [FIN, ACK] Seq=18 Ack=1 Win=2619648 Len=0
 101 22.933030     192.168.56.1  192.168.56.1  TCP      44 12345 → 50441 [ACK] Seq=1 Ack=19 Win=2619648 Len=0
 118 22.939337     192.168.56.1  192.168.56.1  TCP      44 12345 → 50441 [FIN, ACK] Seq=1 Ack=19 Win=2619648 Len=0
 119 22.939377     192.168.56.1  192.168.56.1  TCP      44 50441 → 12345 [ACK] Seq=19 Ack=2 Win=2619648 Len=0
4608 9202.602968   192.168.56.1  192.168.56.1  TCP      56 63730 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
4609 9202.603045   192.168.56.1  192.168.56.1  TCP      56 12345 → 63730 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
4610 9202.603079   192.168.56.1  192.168.56.1  TCP      44 63730 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
```

```
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·······AP···]
     Window: 10233
     [Calculated window size: 2619648]
     [Window size scaling factor: 256]
     Checksum: 0x7968 [unverified]
     [Checksum Status: Unverified]
     Urgent Pointer: 0
   > [Timestamps]
   > [SEQ/ACK analysis]
     TCP payload (17 bytes)
 v Data (17 bytes)
     Data: 73686172616e7961323162697432303038
     [Length: 17]
```

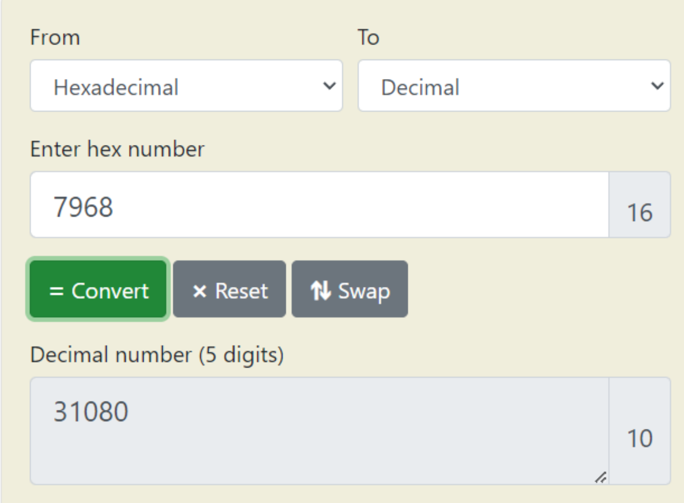
```
0000  02 00 00 00 45 00 00 39  9b 8b 40 00 80 06 00 00   ····E··9 ··@·····
0010  c0 a8 38 01 c0 a8 38 01  c5 09 30 39 2b d7 33 52   ··8···8· ··09+·3R
0020  2e c3 79 2f 50 18 27 f9  79 68 00 00 73 68 61 72   .·y/P·'· yh··shar
0030  61 6e 79 61 32 31 62 69  74 32 30 30 38            anya21bi t2008
```

Details at: https://www.wireshark.org/docs/wsug_html_chunked/ChAdvChecksums.html (tcp.checksum), 2 bytes

Packets: 4883 · Displayed: 12 (0.2%)

Profile: Default

Now the tcp checksum gives recievers side checksum which is calculated as :

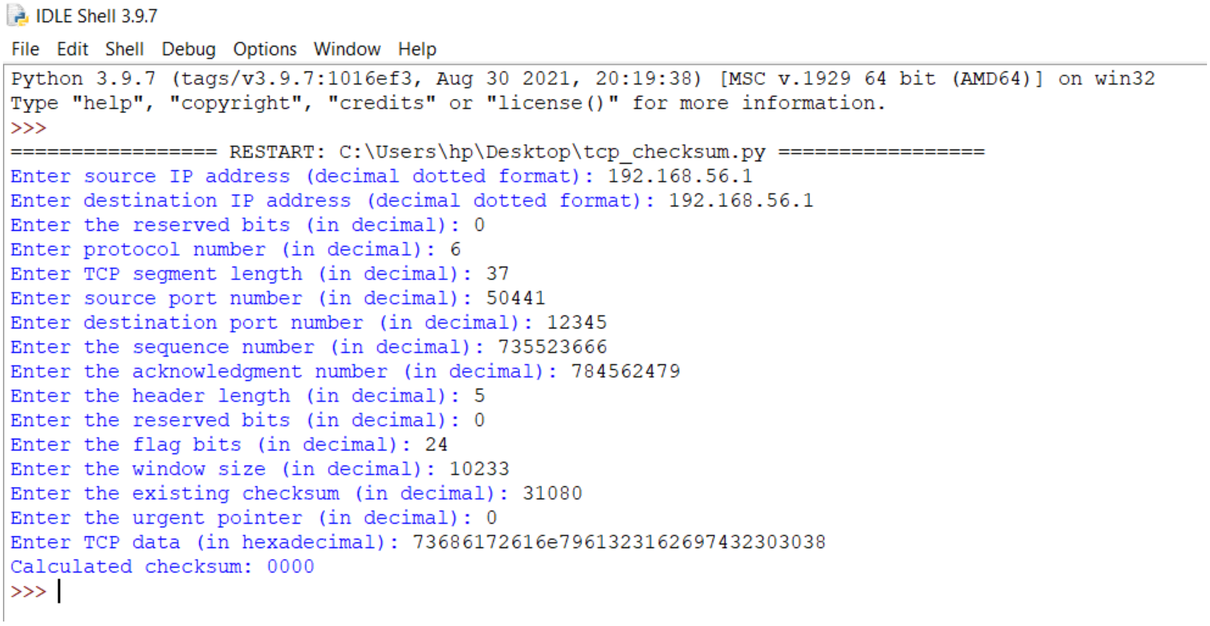


From: Hexadecimal

To: Decimal

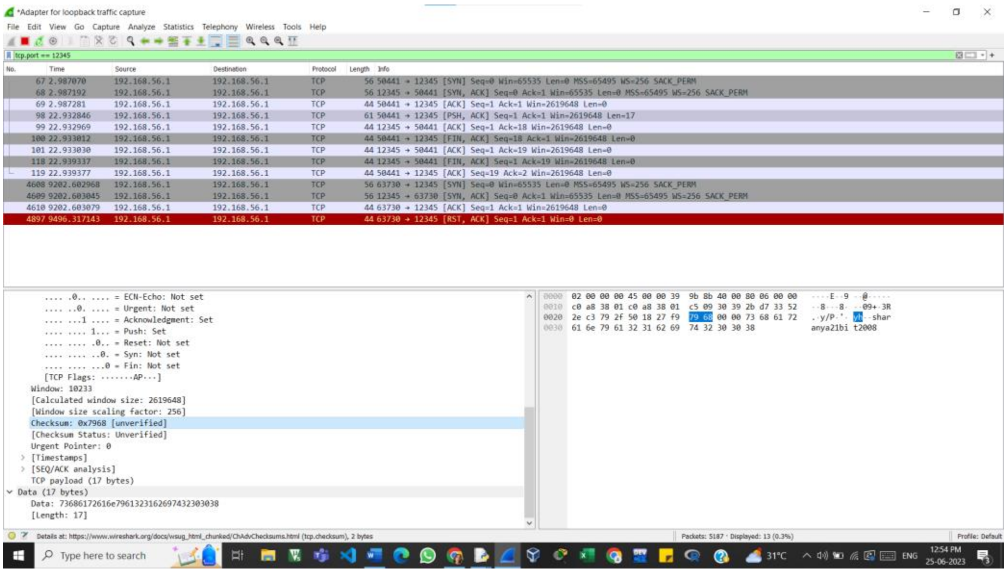
Enter hex number: 7968 (16)

Decimal number (5 digits): 31080 (10)



```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: C:\Users\hp\Desktop\tcp_checksum.py =================
Enter source IP address (decimal dotted format): 192.168.56.1
Enter destination IP address (decimal dotted format): 192.168.56.1
Enter the reserved bits (in decimal): 0
Enter protocol number (in decimal): 6
Enter TCP segment length (in decimal): 37
Enter source port number (in decimal): 50441
Enter destination port number (in decimal): 12345
Enter the sequence number (in decimal): 735523666
Enter the acknowledgment number (in decimal): 784562479
Enter the header length (in decimal): 5
Enter the reserved bits (in decimal): 0
Enter the flag bits (in decimal): 24
Enter the window size (in decimal): 10233
Enter the existing checksum (in decimal): 31080
Enter the urgent pointer (in decimal): 0
Enter TCP data (in hexadecimal): 73686172616e796132316269743230303038
Calculated checksum: 0000
>>> |
```

Wireshark checksum is also shown which is :


*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

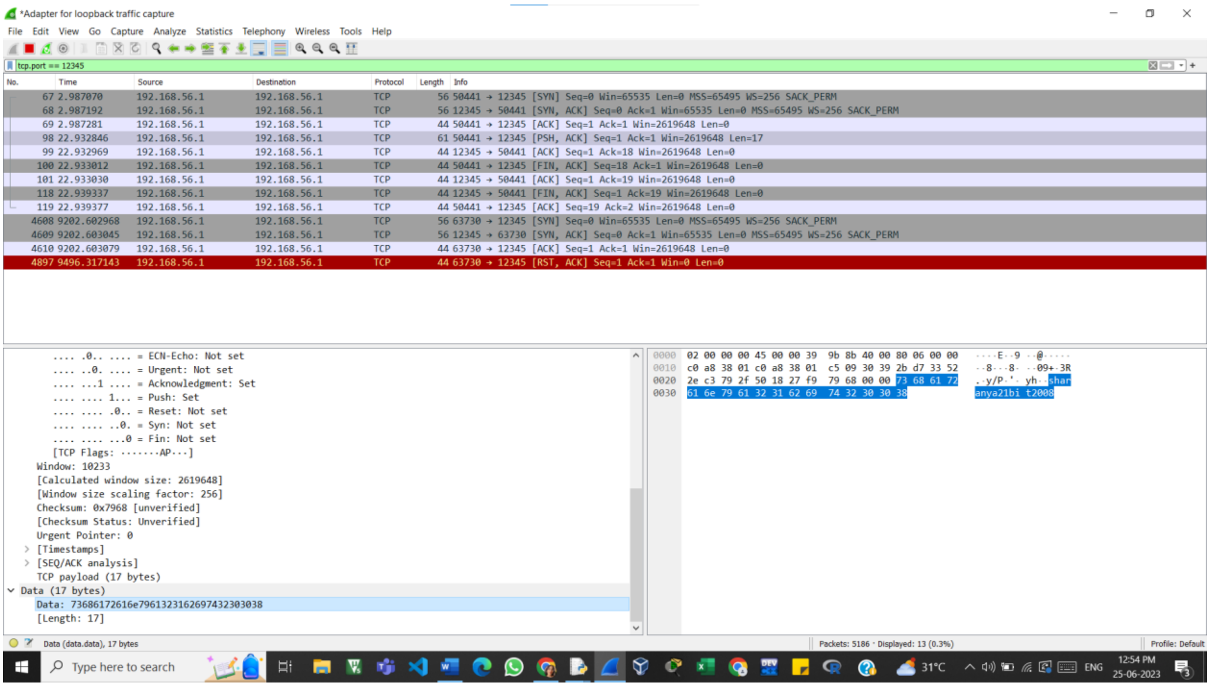
```
tcp.port == 12345
```

```
No.   Time           Source         Destination    Protocol Length Info
  67 2.987070       192.168.56.1   192.168.56.1   TCP      56 50441 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  68 2.987192       192.168.56.1   192.168.56.1   TCP      56 12345 → 50441 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  69 2.987281       192.168.56.1   192.168.56.1   TCP      44 50441 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
  98 22.932846      192.168.56.1   192.168.56.1   TCP      61 50441 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=17
  99 22.932969      192.168.56.1   192.168.56.1   TCP      44 12345 → 50441 [ACK] Seq=1 Ack=18 Win=2619648 Len=0
 100 22.933012      192.168.56.1   192.168.56.1   TCP      44 50441 → 12345 [FIN, ACK] Seq=18 Ack=1 Win=2619648 Len=0
 101 22.933030      192.168.56.1   192.168.56.1   TCP      44 12345 → 50441 [ACK] Seq=1 Ack=19 Win=2619648 Len=0
 118 22.939337      192.168.56.1   192.168.56.1   TCP      44 12345 → 50441 [FIN, ACK] Seq=1 Ack=19 Win=2619648 Len=0
 119 22.939377      192.168.56.1   192.168.56.1   TCP      44 50441 → 12345 [ACK] Seq=19 Ack=2 Win=2619648 Len=0
4608 9202.602968    192.168.56.1   192.168.56.1   TCP      56 63730 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
4609 9202.603045    192.168.56.1   192.168.56.1   TCP      56 12345 → 63730 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
4610 9202.603079    192.168.56.1   192.168.56.1   TCP      44 63730 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4897 9496.317143    192.168.56.1   192.168.56.1   TCP      44 63730 → 12345 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

```
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·······AP···]
    Window: 10233
    [Calculated window size: 2619648]
    [Window size scaling factor: 256]
    Checksum: 0x7968 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
    TCP payload (17 bytes)
v Data (17 bytes)
    Data: 73686172616e796132316269743230303038
    [Length: 17]
```

```
0000  02 00 00 00 45 00 00 39  9b 8b 40 00 80 06 00 00   ····E··9 ··@·····
0010  c0 a8 38 01 c0 a8 38 01  c5 09 30 39 2b d7 33 52   ··8···8· ··09+·3R
0020  2e c3 79 2f 50 18 27 f9  79 68 00 00 73 68 61 72   .·y/P·'· yh··shar
0030  61 6e 79 61 32 31 62 69  74 32 30 30 38            anya21bi t2008
```

```
Details at: https://www.wireshark.org/docs/wsug_html_chunked/ChAdvChecksums.html (tcp.checksum), 2 bytes    Packets: 5187 · Displayed: 13 (0.3%)    Profile: Default
```


*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.port == 12345
Data (data.data), 17 bytes
Packets: 5186 · Displayed: 13 (0.3%)
Profile: Default
Type here to search
31°C
ENG
12:54 PM
25-06-2023

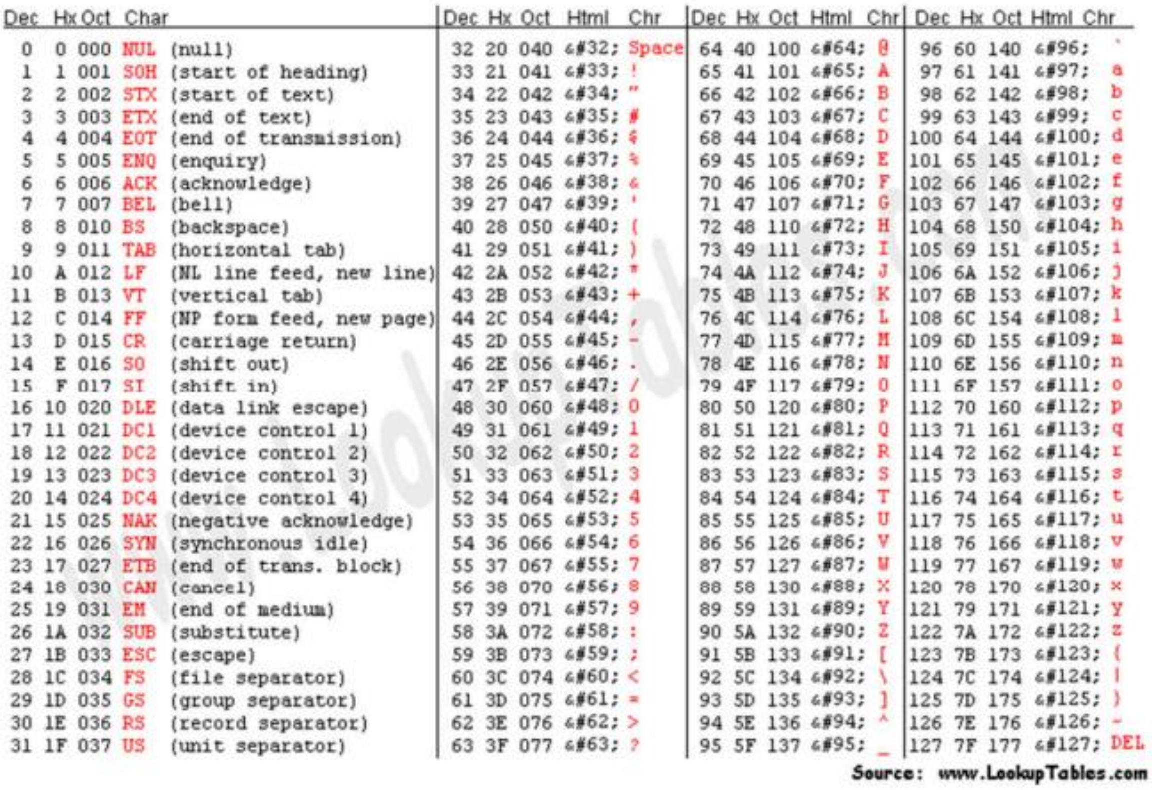
```
No.   Time          Source         Destination    Protocol  Length Info
   67 2.987070      192.168.56.1   192.168.56.1   TCP       56 50441 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   68 2.987192      192.168.56.1   192.168.56.1   TCP       56 12345 → 50441 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   69 2.987281      192.168.56.1   192.168.56.1   TCP       44 50441 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
   98 22.932846     192.168.56.1   192.168.56.1   TCP       61 50441 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=17
   99 22.932969     192.168.56.1   192.168.56.1   TCP       44 12345 → 50441 [ACK] Seq=1 Ack=18 Win=2619648 Len=0
  100 22.933012     192.168.56.1   192.168.56.1   TCP       44 50441 → 12345 [FIN, ACK] Seq=18 Ack=1 Win=2619648 Len=0
  101 22.933030     192.168.56.1   192.168.56.1   TCP       44 12345 → 50441 [ACK] Seq=1 Ack=19 Win=2619648 Len=0
  118 22.939337     192.168.56.1   192.168.56.1   TCP       44 12345 → 50441 [FIN, ACK] Seq=1 Ack=19 Win=2619648 Len=0
  119 22.939377     192.168.56.1   192.168.56.1   TCP       44 50441 → 12345 [ACK] Seq=19 Ack=2 Win=2619648 Len=0
 4608 9202.602968   192.168.56.1   192.168.56.1   TCP       56 63730 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 4609 9202.603045   192.168.56.1   192.168.56.1   TCP       56 12345 → 63730 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 4610 9202.603079   192.168.56.1   192.168.56.1   TCP       44 63730 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
 4897 9496.317143   192.168.56.1   192.168.56.1   TCP       44 63730 → 12345 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

```
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·······AP···]
    Window: 10233
    [Calculated window size: 2619648]
    [Window size scaling factor: 256]
    Checksum: 0x7968 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
    TCP payload (17 bytes)
v Data (17 bytes)
    Data: 73686172616e7961323162697432303038
    [Length: 17]
```

```
0000  02 00 00 00 45 00 00 39  9b 8b 40 00 80 06 00 00   ····E··9 ··@·····
0010  c0 a8 38 01 c0 a8 38 01  c5 09 30 39 2b d7 33 52   ··8···8· ··09+·3R
0020  2e c3 79 2f 50 18 27 f9  79 68 00 00 73 68 61 72   .·y/P·'· yh··shar
0030  61 6e 79 61 32 31 62 69  74 32 30 30 38            anya21bi t2008
```

Manual verification in ASCII/HEX

ASCII Table:



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

TCP Data Value	ASCII/HEX VERIFICATION
67696e61323162697431393030	gina21bit1900
73686172616e7961323162697432303038	sharanya21bit2008

Reference: Data integrity & Risk assessment:

- Data integrity is an important aspect of risk assessment. In the context of information security, risk assessment involves identifying and evaluating potential risks or threats to the confidentiality, integrity, and availability of data and systems.
- Data integrity refers to the accuracy, completeness, and consistency of data throughout its lifecycle. It ensures that data remains unaltered and maintains its intended state and meaning. When assessing risks, it is crucial to consider potential threats and vulnerabilities that could compromise the integrity of data.
- The TCP checksum helps ensure the integrity of data during transmission by detecting errors or changes in the TCP segment. It provides a means to verify that the received segment has not been corrupted or tampered with in transit.
- When a TCP segment is to be transmitted, the sender calculates the checksum based on the segment's contents.
- Upon receiving the segment, the receiver performs the same checksum calculation on the received data. If the calculated checksum at the receiver matches the checksum included in the TCP segment, it indicates that the segment has arrived without any errors or alterations. In this case, the receiver can trust the integrity of the data.
- However, if the calculated checksum at the receiver does not match the checksum in the segment, it implies that the segment has been corrupted during transmission. The receiver can then discard the segment or request retransmission, ensuring that only error-free segments are processed.
- By including the checksum and performing the verification process, the TCP checksum provides a basic level of data integrity checking. It helps detect common errors, such as bit flips, data corruption, or transmission errors, that could occur during segment transmission

THANK YOU !