Lab 5

Computer Networks Lab (CS302)

Alan Tony (191CS207)

Aashish Prateek (191CS225)

Ans.1.

Code:

SSL_server.py

```
# Example SSL server program that listens at port 15001
import ssl
import socket
import datetime
import time

ipAddress   = "127.0.0.1"
port        = 15001

# Create a server socket
serverSocket = socket.socket()
serverSocket.bind((ipAddress, port))

# Listen for incoming connections
serverSocket.listen()
print("Server listening:")

while(True):
    # Keep accepting connections from clients
    (clientConnection, clientAddress) = serverSocket.accept()

    # Make the socket connection to the clients secure through SSLSocket
    secureClientSocket = ssl.wrap_socket(clientConnection,
                                         server_side=True,
                                         ca_certs="./cert/CA/CA.pem",
                                         certfile="./cert/CA/localhost/localhos
t.crt",
                                         keyfile="./cert/CA/localhost/localhost
.decrypted.key",
                                         cert_reqs=ssl.CERT_REQUIRED,
                                         ssl_version=ssl.PROTOCOL_TLSv1_2)

    # Get certificate from the client
    client_cert = secureClientSocket.getpeercert()

    clt_subject    = dict(item[0] for item in client_cert['subject'])
    clt_commonName = clt_subject['commonName']

    # Check the client certificate bears the expected name as per server's
policy
    if not client_cert:
        raise Exception("Unable to get the certificate from the client")

    if clt_commonName != 'localhost':
        raise Exception("Incorrect common name in client certificate")

    # Check time validity of the client certificate
    t1  = ssl.cert_time_to_seconds(client_cert['notBefore'])
```

SSL_client.py

```
import socket
import ssl
import os
import time

# IP address and the port number of the server
sslServerIP    = "127.0.0.1"
sslServerPort  = 15001

# Create an SSL context
context                     = ssl.SSLContext()
context.verify_mode         = ssl.CERT_REQUIRED

# Load CA certificate with which the client will validate the server
certificate
context.load_verify_locations("./cert/ca-bundle.crt")

# Load client certificate
context.load_cert_chain(certfile="./cert/CA/localhost/localhost.crt",
keyfile="./cert/CA/localhost/localhost.decrypted.key")

# Create a client socket
clientSocket        = socket.socket()

# Make the client socket suitable for secure communication
secureClientSocket  = context.wrap_socket(clientSocket)
secureClientSocket.connect((sslServerIP, sslServerPort))

# Obtain the certificate from the server
server_cert = secureClientSocket.getpeercert()

# Validate whether the Certificate is indeed issued to the server
subject         = dict(item[0] for item in server_cert['subject'])
commonName      = subject['commonName']

if not server_cert:
    raise Exception("Unable to retrieve server certificate")

if commonName != 'localhost':
    print(commonName)
    raise Exception("Incorrect common name in server certificate")

notAfterTimestamp   = ssl.cert_time_to_seconds(server_cert['notAfter'])
notBeforeTimestamp  = ssl.cert_time_to_seconds(server_cert['notBefore'])
currentTimeStamp    = time.time()

if currentTimeStamp > notAfterTimestamp:
    raise Exception("Expired server certificate")
```

Output:

Server side:

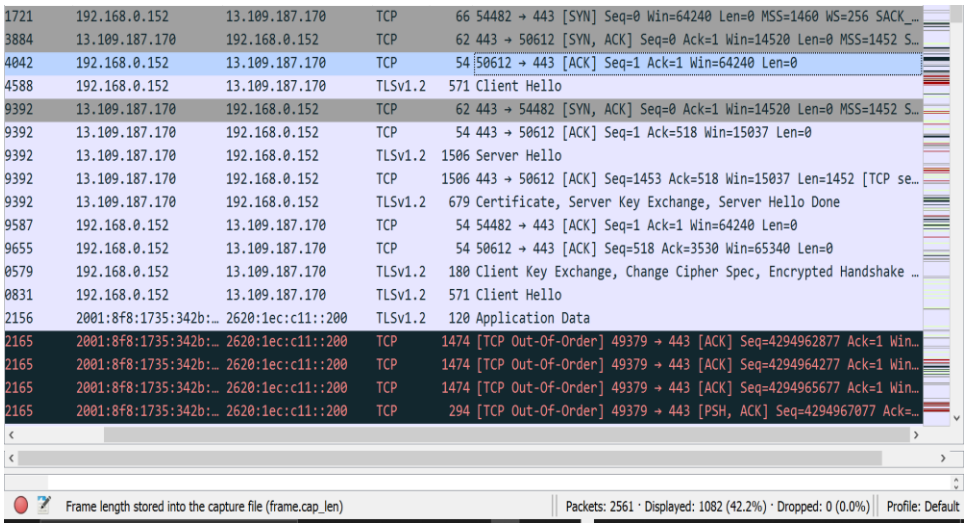
```
Anaconda Prompt (anaconda3) - python  SSL_server.py

(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\CN-Lab-CS302\Lab5\Q1>python SSL_server.py
Server listening:
Securely sent 2021-10-16 14:26:01.718358 to ('127.0.0.1', 57097)
```

Client side:

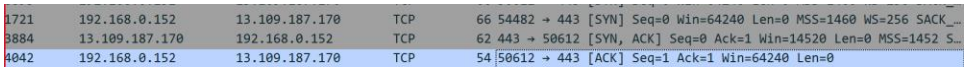
```
Anaconda Prompt (anaconda3)

(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\CN-Lab-CS302\Lab5\Q1>python SSL_client.py
Secure communication received from server:2021-10-16 14:26:01.718358

(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\CN-Lab-CS302\Lab5\Q1>
```

Ans.2.

(a.) Following is a snapshot of the Wireshark TCP trace:

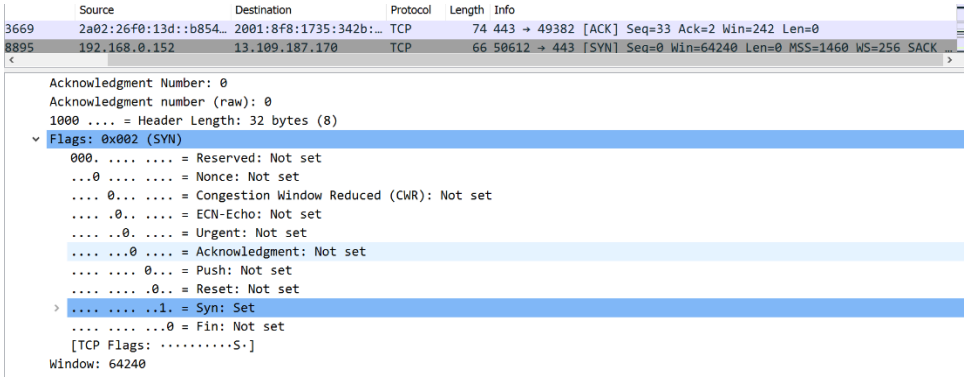


Following is the three-way handshake as seen in Wireshark:

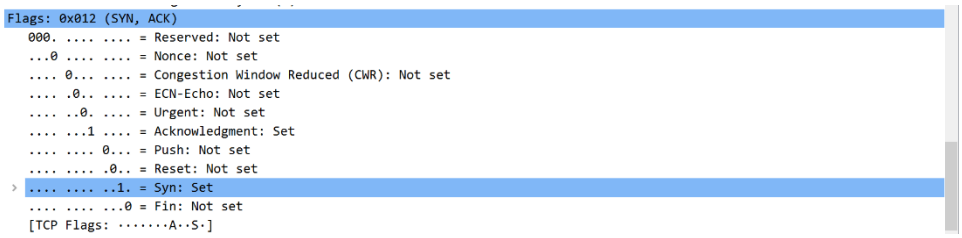


As we can clearly observe the trace for TCP packets above, we take a close look at the first three rows of the trace and we observe that my laptop with IP address 192.168.0.152 is sending a [SYN] request to IP address of server which is 13.109.187.170 whose sequence number is = 0,

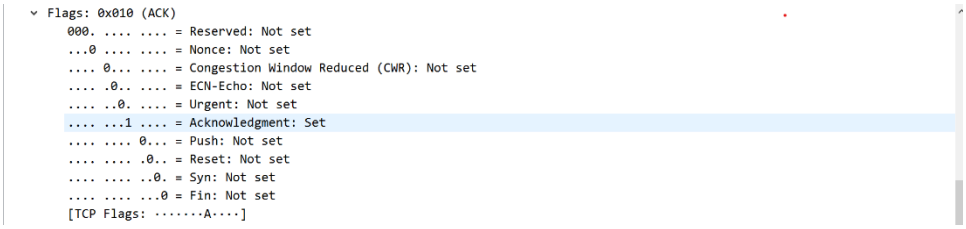
This is the first step in the three-way handshake where my laptop sends a TCP packet with sequence number 0. And we also observe that the syn bit in the flags is set to 'true':



Now in the second part(way) of the handshake we observe that second row of the handshake where the serve responds with a SYN-ACK message with Seq=0 and Ack=1. The sequence and acks numbers are related between the client and server. So, for the TCP handshake example, if the client sends a seq=0, the server responds with ack=1. Ack is basically acknowledgement number. We can even observe this is the flags section where the Ack bit is set to 1 along with the syn bit. This happens as server acknowledges that it received from the client.

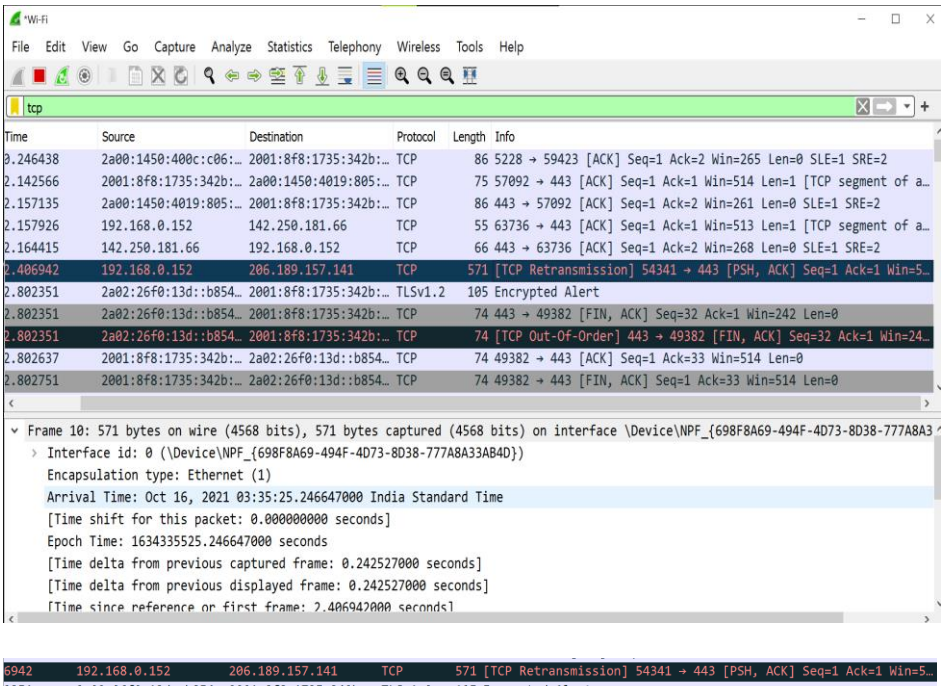


Finally, we have the third part(way) of the handshake where we can see the client that initiated the TCP session sends an acknowledgement to complete the 3-way handshake. Here the Syn bit is not set. And the type of packet sent is an ACK (acknowledgement from the client side to server). Here Seq=1 and Ack=1 because the TCP-Syn from the server sent a seq=0 and ack=1 in the TCP Syn-Ack. And correspondingly we can observe the flags:

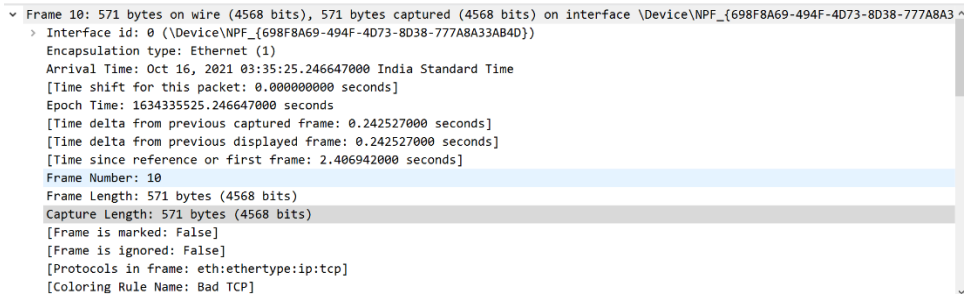


Thus, the three-way handshake is completed between server and client.

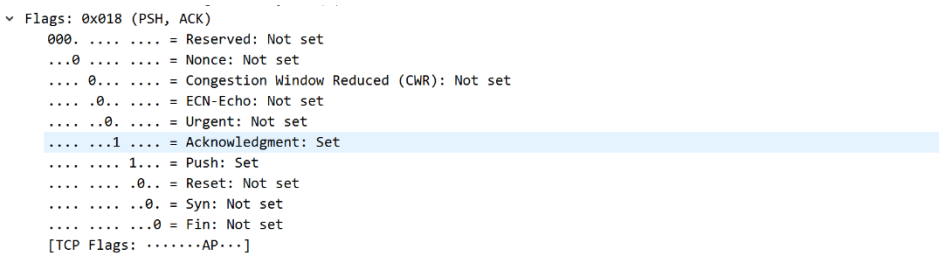
(b.) Yes, there are some retransmission packets as we can observe in the snapshot of the trace below:



The above is the retransmission packet zoomed in. Following are the details of the packet:



Retransmission of TCP packets happens due to network congestion. We observe that the push bit is set here.



*****