

# 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/localhost.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:

No.	Time	Source	Destination	Protocol	Length	Info
1721	0.000000	192.168.0.152	13.109.187.170	TCP	66	54482 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_...
3884	0.000000	13.109.187.170	192.168.0.152	TCP	62	443 → 50612 [SYN, ACK] Seq=0 Ack=1 Win=14520 Len=0 MSS=1452 S...
4042	0.000000	192.168.0.152	13.109.187.170	TCP	54	50612 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4588	0.000000	192.168.0.152	13.109.187.170	TLSv1.2	571	Client Hello
9392	0.000000	13.109.187.170	192.168.0.152	TCP	62	443 → 54482 [SYN, ACK] Seq=0 Ack=1 Win=14520 Len=0 MSS=1452 S...
9392	0.000000	13.109.187.170	192.168.0.152	TLSv1.2	1506	Server Hello
9392	0.000000	13.109.187.170	192.168.0.152	TCP	54	443 → 50612 [ACK] Seq=1 Ack=518 Win=15037 Len=0
9392	0.000000	13.109.187.170	192.168.0.152	TLSv1.2	679	Certificate, Server Key Exchange, Server Hello Done
9587	0.000000	192.168.0.152	13.109.187.170	TCP	54	54482 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
9655	0.000000	192.168.0.152	13.109.187.170	TCP	54	50612 → 443 [ACK] Seq=518 Ack=3530 Win=65340 Len=0
9579	0.000000	192.168.0.152	13.109.187.170	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake ...
9831	0.000000	192.168.0.152	13.109.187.170	TLSv1.2	571	Client Hello
2156	0.000000	2001:8f8:1735:342b::...	2620:1ec:c11::200	TLSv1.2	120	Application Data
2165	0.000000	2001:8f8:1735:342b::...	2620:1ec:c11::200	TCP	1474	[TCP Out-Of-Order] 49379 → 443 [ACK] Seq=4294962877 Ack=1 Win...
2165	0.000000	2001:8f8:1735:342b::...	2620:1ec:c11::200	TCP	1474	[TCP Out-Of-Order] 49379 → 443 [ACK] Seq=4294964277 Ack=1 Win...
2165	0.000000	2001:8f8:1735:342b::...	2620:1ec:c11::200	TCP	1474	[TCP Out-Of-Order] 49379 → 443 [ACK] Seq=4294965677 Ack=1 Win...
2165	0.000000	2001:8f8:1735:342b::...	2620:1ec:c11::200	TCP	294	[TCP Out-Of-Order] 49379 → 443 [PSH, ACK] Seq=4294967077 Ack=...

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

No.	Time	Source	Destination	Protocol	Length	Info
1721	0.000000	192.168.0.152	13.109.187.170	TCP	66	54482 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_...
3884	0.000000	13.109.187.170	192.168.0.152	TCP	62	443 → 50612 [SYN, ACK] Seq=0 Ack=1 Win=14520 Len=0 MSS=1452 S...
4042	0.000000	192.168.0.152	13.109.187.170	TCP	54	50612 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0

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':

	Source	Destination	Protocol	Length	Info
3669	2a02:26f0:13d::b854...	2001:8f8:1735:342b::...	TCP	74	443 → 49382 [ACK] Seq=33 Ack=2 Win=242 Len=0
8895	192.168.0.152	13.109.187.170	TCP	66	50612 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK

  

```

Acknowledgment Number: 0
Acknowledgment number (raw): 0
1000 .... = Header Length: 32 bytes (8)
▼ Flags: 0x002 (SYN)
 000. .... = Reserved: Not set
...0 .... = Nonce: Not set
... 0... = Congestion Window Reduced (CWR): Not set
... .0.. = ECN-Echo: Not set
... ..0. = Urgent: Not set
... ...0 = Acknowledgment: Not set
... .... 0... = Push: Not set
... ..... 0.. = Reset: Not set
> ... ..1. = Syn: Set
... ..0. = Fin: Not set
[TCP Flags: .....S.]
Window: 64240

```

Now in the second part(way) of the handshake we observe that second row of the handshake where the server 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.

```

Flags: 0x012 (SYN, ACK)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
... 0... = Congestion Window Reduced (CWR): Not set
... .0.. = ECN-Echo: Not set
... ..0. = Urgent: Not set
... ...1 = Acknowledgment: Set
... .... 0... = Push: Not set
... ..... 0.. = Reset: Not set
> ... ..1. = Syn: Set
... ..0. = Fin: Not set
[TCP Flags: .....A..S.]

```

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:

```

▼ Flags: 0x010 (ACK)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
... 0... = Congestion Window Reduced (CWR): Not set
... .0.. = ECN-Echo: Not set
... ..0. = Urgent: Not set
... ...1 = Acknowledgment: Set
... .... 0... = Push: Not set
... ..... 0.. = Reset: Not set
... ..0. = Syn: Not set
... ..0. = Fin: Not set
[TCP Flags: .....A....]

```

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:

Time	Source	Destination	Protocol	Length	Info
0.246438	2a00:1450:400c:c06::...	2001:8f8:1735:342b::...	TCP	86	5228 → 59423 [ACK] Seq=1 Ack=2 Win=265 Len=0 SLE=1 SRE=2
2.142566	2001:8f8:1735:342b::...	2a00:1450:4019:805::...	TCP	75	57092 → 443 [ACK] Seq=1 Ack=1 Win=514 Len=1 [TCP segment of a...]
2.157135	2a00:1450:4019:805::...	2001:8f8:1735:342b::...	TCP	86	443 → 57092 [ACK] Seq=1 Ack=2 Win=261 Len=0 SLE=1 SRE=2
2.157926	192.168.0.152	142.250.181.66	TCP	55	63736 → 443 [ACK] Seq=1 Ack=1 Win=513 Len=1 [TCP segment of a...]
2.164415	142.250.181.66	192.168.0.152	TCP	66	443 → 63736 [ACK] Seq=1 Ack=2 Win=268 Len=0 SLE=1 SRE=2
2.406942	192.168.0.152	206.189.157.141	TCP	571	[TCP Retransmission] 54341 → 443 [PSH, ACK] Seq=1 Ack=1 Win=5...
2.802351	2a02:26f0:13d::b854...	2001:8f8:1735:342b::...	TLSv1.2	105	Encrypted Alert
2.802351	2a02:26f0:13d::b854...	2001:8f8:1735:342b::...	TCP	74	443 → 49382 [FIN, ACK] Seq=32 Ack=1 Win=242 Len=0
2.802351	2a02:26f0:13d::b854...	2001:8f8:1735:342b::...	TCP	74	[TCP Out-Of-Order] 443 → 49382 [FIN, ACK] Seq=32 Ack=1 Win=24...
2.802637	2001:8f8:1735:342b::...	2a02:26f0:13d::b854...	TCP	74	49382 → 443 [ACK] Seq=1 Ack=33 Win=514 Len=0
2.802751	2001:8f8:1735:342b::...	2a02:26f0:13d::b854...	TCP	74	49382 → 443 [FIN, ACK] Seq=1 Ack=33 Win=514 Len=0

  

Frame 10: 571 bytes on wire (4568 bits), 571 bytes captured (4568 bits) on interface \Device\NPF\_{698F8A69-494F-4D73-8D38-777A8A3...}

Interface id: 0 (\Device\NPF\_{698F8A69-494F-4D73-8D38-777A8A33AB4D})

Encapsulation type: Ethernet (1)

Arrival Time: Oct 16, 2021 03:35:25.246647000 India Standard Time

[Time shift for this packet: 0.000000000 seconds]

Epoch Time: 1634335525.246647000 seconds

[Time delta from previous captured frame: 0.242527000 seconds]

[Time delta from previous displayed frame: 0.242527000 seconds]

[Time since reference or first frame: 2.406942000 seconds]

  

Time	Source	Destination	Protocol	Length	Info
2.406942	192.168.0.152	206.189.157.141	TCP	571	[TCP Retransmission] 54341 → 443 [PSH, ACK] Seq=1 Ack=1 Win=5...

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

```

Frame 10: 571 bytes on wire (4568 bits), 571 bytes captured (4568 bits) on interface \Device\NPF_{698F8A69-494F-4D73-8D38-777A8A3...}
  Interface id: 0 (\Device\NPF_{698F8A69-494F-4D73-8D38-777A8A33AB4D})
  Encapsulation type: Ethernet (1)
  Arrival Time: Oct 16, 2021 03:35:25.246647000 India Standard Time
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1634335525.246647000 seconds
  [Time delta from previous captured frame: 0.242527000 seconds]
  [Time delta from previous displayed frame: 0.242527000 seconds]
  [Time since reference or first frame: 2.406942000 seconds]
  Frame Number: 10
  Frame Length: 571 bytes (4568 bits)
  Capture Length: 571 bytes (4568 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ethertype:ip:tcp]
  [Coloring Rule Name: Bad TCP]

```

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

```

Flags: 0x018 (PSH, ACK)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .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...]

```

\*\*\*\*\*