

CS6903: Network Security

Assignment 6: Secure chat using openssl and MITM attacks Report and Readme

PLAGIARISM STATEMENT

I certify that this assignment/report is my own work, based on my personal study and/or research and that I have acknowledged all materials and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment/report has not previously been submitted for assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, I understand my responsibility to report honor violations by other students if I become aware of them.

Name: Manisha Mahapatra, Reisha Ali

Date: 02/04/2023

Signature: CS22MTECH14009, CS23RESCH01001

Description:

In this programming assignment, we have implemented a secure peer-to-peer chat application using openssl in C/C++ and demonstrated how Alice and Bob could chat with each other using it. We have also implemented evil Trudy who is trying to intercept the chat messages between Alice and Bob by launching various MITM attacks.

Setup:

Accessing assigned VM : `ssh ubuntu@10.200.13.113`

Task – 1: Generate keys and certificates

OpenSSL commands used:

For RootCA

- Create rootCA key
openssl ecparam -name secp521r1 -genkey -out rootCA.pem
- View rootCA.pem key
cat rootCA.pem
- Create ExtCA file
cat >> ExtCA.ext
basicConstraints = CA:true
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, keyCertSign, cRLSign
- Create root CSR and generate v3 x509 self-signed certificate
openssl req -new -key rootCA.pem -out rootCAcsr.pem
openssl x509 -req -days 365 -in rootCAcsr.pem -signkey rootCA.pem -out rootCA.crt -extfile ExtCA.ext

For IntermediateCA

- Create intermediateCA key
openssl genrsa -out interCA.pem 2048
- View interCA.pem key
cat interCA.pem
- Generate public key
openssl rsa -in interCA.pem -pubout -out interCA_p.pem
- View interCA_p.pem
cat interCA_p.pem
- Create intermediate CSR
openssl req -new -key interCA.pem -out interCAcsr.pem
- Create v3 x509 certificate signed by rootCA
openssl x509 -req -in interCAcsr.pem -CA rootCA.crt -extfile ExtCA.ext -CAkey rootCA.pem -CAcreateserial -out interCA.crt -days 365 -sha256

For Alice

- Create Alice key
openssl genrsa -out alice.pem 1024
- View alice.pem key
cat alice.pem
- Generate Alice public key
openssl rsa -in alice.pem -pubout -out alice_p.pem
- View alice_p.pem
cat alice_p.pem
- Create Alice CSR
openssl req -new -key alice.pem -out alicecsr.pem
- Generate digest and sign to generate hash
openssl dgst -sha256 -sign alice.pem -out alice_sign.pem alicecsr.pem
- At RootCA:
- Verify that CSR request sent by Alice
openssl dgst -sha256 -verify alice_p.pem -signature alice_sign.pem alicecsr.pem
- Create v3 x509 certificate signed by interCA
openssl x509 -req -in alicecsr.pem -CA interCA.crt -extfile ExtCA.ext -CAkey interCA.pem -CAcreateserial -out alice.crt -days 365 -sha256
- Generate digest and sign to generate hash
openssl dgst -sha256 -sign interCA.pem -out interCA_sign.pem alice.crt

At Alice end:

- Verify that certificate sent by Intermediate CA
openssl dgst -sha256 -verify interCA_p.pem -signature interCA_sign.pem alice.crt
- Verify Alice certificate
openssl verify -verbose -CAfile rootCA.crt alice.crt
In .pem format
openssl verify -verbose -CAfile rootCA.crt alicecert.pem

For Bob

- Create Bob key
openssl ecparam -name prime256v1 -genkey -out bob.pem
 - View bob.pem key
cat bob.pem
 - Generate Bob public key
openssl rsa -in bob.pem -pubout -out bob_p.pem
 - View bob_p.pem
cat bob_p.pem
 - Create Bob CSR
openssl req -new -key bob.pem -out bobcsr.pem
 - Generate digest and sign to generate hash
openssl dgst -sha256 -sign bob.pem -out bob_sign.pem bobcsr.pem
- At RootCA:
- Verify that CSR request sent by Bob
openssl dgst -sha256 -verify bob_p.pem -signature bob_sign.pem bobcsr.pem
 - Create v3 x509 certificate signed by interCA
openssl x509 -req -in bobcsr.pem -CA interCA.crt -extfile ExtCA.ext -CAkey interCA.pem -CAcreateserial -out bob.crt -days 365 -sha256
 - Generate digest and sign to generate hash
openssl dgst -sha256 -sign interCA.pem -out interCA_sign.pem bob.crt
- At Alice end:
- Verify that certificate sent by Intermediate CA
openssl dgst -sha256 -verify interCA_p.pem -signature interCA_sign.pem bob.crt
 - Verify Bob certificate
openssl verify -verbose -CAfile rootCA.crt bob.crt
In .pem format
openssl verify -verbose -CAfile rootCA.crt bobcert.pem

Process followed for checking integrity and authenticity:

- ◆ Alice and Bob created their private key and CSR.
- ◆ Sending .csr files to interCA along with signed hash (digital signature) to prove that those cert req are indeed sent by them.
- ◆ Then interCA verifies the digital signature and issues the certificate to Alice/Bob.
- ◆ Along with the .crt file interCA will now send a signed digest of .crt to Alice/Bob to prove that the certificate was indeed issued by interCA.
- ◆ Since interCA's public key is available to Alice/Bob they can verify in the same manner.

Task – 2: Secure Chat App

Setup and message flow for secure_chat:

In secure_chat we have secure_chat_app.cpp which takes command line arguments.

For server and client we created two functions in secure_chat_app

Commands:

For compiling the code-

```
secure_chat/home/ubuntu/secure_chat_alice/secure_chat# g++ secure_chat_app.cpp -lssl  
-lcrypto -o secure_chat_app
```

For server:

```
./secure_chat_app -s <hostname> <portNo>
```

For client:

```
./secure_chat_app -c <hostname> <portNo>
```

For collecting traffic data and saving in .pcap file we use tcpdump-
tcpdump -i eth0 -w TASK2.pcap

At Server():

- Socket creation for the server & binding it to the server port then set it in listen mode (waiting for the client).
- After getting connected with client address, a chat session is initiated with 'chat_request' and 'start_tls'.
- Server waits for client until it sends the chat_request message and once received it sends 'chat_reply' message
- The server waits for the client's 'start_tls' message and once received, the server acknowledges it by sending 'start_tls_ack' message to the client.
- 'start_comm_flag' and 'start_tls_flag' flag variables are used to ensure that the TCP hello and TLS handshake occurs for only one time.
- Later the server loads its keys and certificate into the SSL Context and verifies the certificate received from the client into the context.
- Wraps the client socket with SSL Context and performs TLS handshake with the client thereby a secure TLS connection is established.
- Then for the further communication, the new TLS socket is used for auto encrypted and decrypted data.

At Client():

- Socket creation for the client.
- From serverhostname, the server ip address is resolved using gethostbyname() and connect() is used for connecting to server.
- Once connected, client sends the 'chat_request' to server and waits for the server's reply
- After receiving 'chat_reply' from server, client sends 'start_tls' to the server.
- Once receiving start_tls_ack then loads its keys and certificate into SSL Context for making the secure TLS connection.
- Wraps the client socket with SSL Context and performs TLS handshake with the server.
- Now Secure TLS connection is established, client starts messaging the server.
- If the reply for start_tls is not an acknowledgement , then the client continues sending the messages in the TCP socket without TLS handshake
- When a client receives or sends 'term' then the TLS & TCP connections are closed and the program terminates.

Screenshots:

With Alice-

```
root@alice1:~/Alice# ./secure_chat_app -c bob1 8001
IP address of Client:
172.31.0.3
Client Socket Created
Enter message for Server:
chat_request
Received From Server: chat_reply
Enter message for Server:
start_tls
Received From Server: start_tls_ack
Client ctx created
C = IN, ST = OD, L = RKL, O = RSP, OU = NS, CN = Alice1.com
C = EU, ST = London, L = RoseBay, O = EY, OU = NS, CN = Bob1.com
Server Certificate Valid
Enter message for Server on TLS:
Hello Alice
Server received on tls: Hello Bob
Enter message for Server on TLS:
GoodBye.
Server received on tls: Ok Bye.
Enter message for Server on TLS:
term
Connection Terminated
```

With Bob-

```
root@bob1:~/Bob# ./secure_chat_app -s 8001
.....Waiting for client connection.....
Client request accepted from IP: 172.31.0.2
Received From Client: chat_request
Enter message for Client:
chat_reply
Received From Client: start_tls
Enter message for Client:
start_tls_ack
Server ctx created
C = EU, ST = London, L = RoseBay, O = EY, OU = NS, CN = Bob1.com 4here in this
C = IN, ST = OD, L = RKL, O = RSP, OU = NS, CN = Alice1.com
Client Certificate Valid
Received from Client on TLS: Hello Alice
Enter message for Client on TLS:
Hello Bob
Received from Client on TLS: GoodBye.
Enter message for Client on TLS:
Ok Bye.
Connection terminated from Client
```

Unencrypted initial chat_request message-

```
2 172.31.0.3 8004 33150 → 8004 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=13 TSval=369242149
<
  Ethernet II, Src: Xensourc_6c:6d:b0 (00:16:3e:6c:6d:b0), Dst: Xensourc_ec:4b:d4 (00:16:
  Internet Protocol Version 4, Src: 172.31.0.2, Dst: 172.31.0.3
  Transmission Control Protocol, Src Port: 33150, Dst Port: 8004, Seq: 1, Ack: 1, Len: 13
    Source Port: 33150
    Destination Port: 8004
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 13]
    Sequence Number: 1 (relative sequence number)
    Sequence Number (raw): 3193877778
    [Next Sequence Number: 14 (relative sequence number)]
    Acknowledgment Number: 1 (relative ack number)
    Acknowledgment number (raw): 3851004156
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x018 (PSH, ACK)
    Window: 502
    [Calculated window size: 64256]
    [Window size scaling factor: 128]
    Checksum: 0x5877 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    [Timestamps]
    [SEQ/ACK analysis]
    TCP payload (13 bytes)
  Data (13 bytes)
    Data: 636861745f7265717565737400
    [Length: 13]
<
0000 00 16 3e ec 4b d4 00 16 3e 6c 6d b0 08 00 45 00 -->·K···>lm···E·
0010 00 41 35 ec 40 00 40 06 ac 87 ac 1f 00 02 ac 1f ·A5·@·@·
0020 00 03 81 7e 1f 44 be 5e b5 12 e5 89 a8 fc 80 18 ··~·D·^·
0030 01 f6 58 77 00 00 01 01 08 0a dc 15 e1 75 5b f9 ··Xw·····u[·
0040 ff 0f 63 68 61 74 5f 72 65 71 75 65 73 74 00 ··chat_request·
```

```

172.31.0.3 172.31.0.2 33150 8004 → 33150 [PSH, ACK] Seq=1 Ack=14 Win=65152 Len=11 TSval=1543127
Frame 6: 77 bytes on wire (616 bits), 77 bytes captured (616 bits)
Ethernet II, Src: Xensourc_ec:4b:d4 (00:16:3e:ec:4b:d4), Dst: Xensourc_6c:6d:b0 (00:16:3e:ec:6d:b0)
Internet Protocol Version 4, Src: 172.31.0.3, Dst: 172.31.0.2
Transmission Control Protocol, Src Port: 8004, Dst Port: 33150, Seq: 1, Ack: 14, Len: 11
Data (11 bytes)
Data: 636861745f7265706c7900
[Length: 11]
0000 00 16 3e 6c 6d b0 00 16 3e ec 4b d4 08 00 45 00  ..>lm... >.K...E.
0010 00 3f de 4b 40 00 40 06 04 2a ac 1f 00 03 ac 1f  .?.K@.@..*.....
0020 00 02 1f 44 81 7e e5 89 a8 fc be 5e b5 1f 80 18  .D~...^.....
0030 01 fd 58 75 00 00 01 01 08 0a 5b fa 42 65 dc 15  .Xu....[.Be...
0040 e1 75 63 68 61 74 5f 72 65 70 6c 79 00  .uchat_r eply.

```

```

172.31.0.2 172.31.0.3 8004 Client Hello
> Frame 12: 254 bytes on wire (2032 bits), 254 bytes captured (2032 bits)
> Ethernet II, Src: Xensourc_6c:6d:b0 (00:16:3e:6c:6d:b0), Dst: Xensourc_ec:4b:d4 (00:16:3e:ec:4b:d4)
> Internet Protocol Version 4, Src: 172.31.0.2, Dst: 172.31.0.3
> Transmission Control Protocol, Src Port: 33150, Dst Port: 8004, Seq: 24, Ack: 26, Len: 188
> Transport Layer Security
  > TLSv1.2 Record Layer: Handshake Protocol: Client Hello
0000 00 16 3e ec 4b d4 00 16 3e 6c 6d b0 08 00 45 00 --> K --> 1m -- E-
0010 00 f0 35 f0 40 d0 00 06 ad b4 ac 1f 00 02 ac 1f -- 5 - @ - @ -
0020 00 03 81 7e 1f 44 be 5e b5 29 e5 89 a9 15 80 18 -- -- D ^ ( )
0030 01 f6 59 26 00 00 01 01 08 0a dc 16 50 7d 5b fa -- Y & -- P ] [ -
0040 8d ef 16 03 01 00 b7 01 00 00 b3 03 03 a7 a4 21 -- -- -- -- !
0050 47 c8 91 d3 04 f3 3c 54 c4 5b 28 c4 52 19 2a c1 G -- -- < T -- [ ( - R * -
0060 d7 9e cc eb c5 77 67 ed 72 93 33 87 90 00 00 38 -- -- wg - r - 3 - 8
0070 c0 2c c0 30 00 9f cc a9 cc a8 cc aa c0 2b c0 2f -- , 0 -- -- + /
0080 00 9e c0 24 c0 28 00 6b c0 23 c0 27 00 67 c0 0a -- - $ - ( - k - # - ' - g -
0090 c0 14 00 39 c0 09 c0 13 00 33 00 9d 00 9c 00 3d -- - 9 -- -- 3 -- =
00a0 00 3c 00 35 00 2f 00 ff 01 00 00 52 00 0b 00 04 -- < 5 / -- -- R --
00b0 03 00 01 02 00 0a 00 0c 00 0a 00 1d 00 17 00 1e -- -- -- --
00c0 00 19 00 18 00 23 00 00 00 16 00 00 00 17 00 00 -- -- # --
00d0 00 0d 00 2a 00 28 04 03 05 03 06 03 08 07 08 08 -- -- * - ( --
00e0 08 09 08 0a 08 0b 08 04 08 05 08 06 04 01 05 01 -- -- -- --
00f0 06 01 03 03 03 01 03 02 04 02 05 02 06 02 -- -- -- 1 --

```

```

172.31.0.2 172.31.0.3 8004 Application Data
<
>
  Frame 20: 122 bytes on wire (976 bits), 122 bytes captured (976 bits)
  Ethernet II, Src: Xensourc_6c:6d:b0 (00:16:3e:6c:6d:b0), Dst: Xensourc_ec:4b:d4 (00:16:3e:ec:4b:d4)
  Internet Protocol Version 4, Src: 172.31.0.2, Dst: 172.31.0.3
  Transmission Control Protocol, Src Port: 33150, Dst Port: 8004, Seq: 3009, Ack: 3634, Len: 56
  Transport Layer Security
    TLSv1.2 Record Layer: Application Data Protocol: Application Data
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 51
      Encrypted Application Data: d1bd7e4340832ef03447ea53c4e9bde9b91bd30af9686a5e00aa2c6b41b59ea53b.
0000 00 16 3e ec 4b d4 00 16 3e 6c 6d b0 08 00 45 00  K... >lm...E
0010 00 6c 35 f5 40 00 40 06 ac 53 ac 1f 00 02 ac 1f  -15 @-@-
0020 00 03 81 7e 1f 44 be 5e c0 d2 e5 89 b7 2d 80 18  ---D^.....
0030 01 f5 58 a2 00 00 01 01 08 0a dc 16 9b 16 5b fa  -X.....[
0040 8d fc 17 03 03 00 33 d1 bd 7e 43 40 83 2e f0 34  ....3...C@...4
0050 47 ea 53 c4 e9 bd e9 b9 1b d3 0a f9 68 6a 5e 00  G.S.....hj^
0060 aa 2c 6b 41 b5 9e a5 3b 72 38 3c 38 67 a0 d8 29  ,kA...; r8<8g-)
0070 f9 b5 3e b8 bf 97 6c 10 76 fc  ->...l v

```

```

172.31.0.3 172.31.0.2 33150 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
<
>
> Frame 18: 1348 bytes on wire (10784 bits), 1348 bytes captured (10784 bits)
> Ethernet II, Src: Xensourc_ec:4b:d4 (00:16:3e:ec:4b:d4), Dst: Xensourc_6c:6d:b0 (00:16:3e:6c:6d:b0)
> Internet Protocol Version 4, Src: 172.31.0.3, Dst: 172.31.0.2
> Transmission Control Protocol, Src Port: 8004, Dst Port: 33150, Seq: 2352, Ack: 3009, Len: 1282
> Transport Layer Security
  < TLSv1.2 Record Layer: Handshake Protocol: New Session Ticket
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 1226
    < Handshake Protocol: New Session Ticket
      Handshake Type: New Session Ticket (4)
      Length: 1222
      > TLS Session Ticket
    < TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
    < TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 40
      Handshake Protocol: Encrypted Handshake Message

```

Supported ciphersuites as specified in the program:

```
//Initialize list of cipher suites
```

```
const char* str = "ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384";
```

For other ciphersuites, we not be able to perform verification and thus the connection will fail.

Flow Graph-

172.31.0.2	172.31.0.3	Comment
33150	33150 → 8004 [SYN] Seq=0 Win=64240 ...	TCP: 33150 → 8004 [SYN] Seq=0 Win=64240 Len=0 M..
33150	8004 → 33150 [SYN, ACK] Seq=0 Ack=1 ...	TCP: 8004 → 33150 [SYN, ACK] Seq=0 Ack=1 Win=651..
33150	33150 → 8004 [ACK] Seq=1 Ack=1 Win=...	TCP: 33150 → 8004 [ACK] Seq=1 Ack=1 Win=64256 L...
33150	33150 → 8004 [PSH, ACK] Seq=1 Ack=1 ...	TCP: 33150 → 8004 [PSH, ACK] Seq=1 Ack=1 Win=642..
33150	8004 → 33150 [ACK] Seq=1 Ack=14 Win=...	TCP: 8004 → 33150 [ACK] Seq=1 Ack=14 Win=65152 ...
33150	8004 → 33150 [PSH, ACK] Seq=1 Ack=1...	TCP: 8004 → 33150 [PSH, ACK] Seq=1 Ack=14 Win=65..
33150	33150 → 8004 [ACK] Seq=14 Ack=12 Wi...	TCP: 33150 → 8004 [ACK] Seq=14 Ack=12 Win=64256..
33150	33150 → 8004 [PSH, ACK] Seq=14 Ack=...	TCP: 33150 → 8004 [PSH, ACK] Seq=14 Ack=12 Win=6..
33150	8004 → 33150 [ACK] Seq=12 Ack=24 Wi...	TCP: 8004 → 33150 [ACK] Seq=12 Ack=24 Win=65152..
33150	8004 → 33150 [PSH, ACK] Seq=12 Ack=...	TCP: 8004 → 33150 [PSH, ACK] Seq=12 Ack=24 Win=6..
33150	33150 → 8004 [ACK] Seq=24 Ack=26 Wi...	TCP: 33150 → 8004 [ACK] Seq=24 Ack=26 Win=64256..
33150	Client Hello	TLSv1.2: Client Hello
33150	8004 → 33150 [ACK] Seq=26 Ack=212 ...	TCP: 8004 → 33150 [ACK] Seq=26 Ack=212 Win=6502..
33150	Server Hello, Certificate, Server Key Ex...	TLSv1.2: Server Hello, Certificate, Server Key Exchan...
33150	33150 → 8004 [ACK] Seq=212 Ack=235...	TCP: 33150 → 8004 [ACK] Seq=212 Ack=2352 Win=63..
33150	Certificate, Client Key Exchange, Certifi...	TLSv1.2: Certificate, Client Key Exchange, Certificate ..
33150	8004 → 33150 [ACK] Seq=2352 Ack=30...	TCP: 8004 → 33150 [ACK] Seq=2352 Ack=3009 Win=6..
33150	New Session Ticket, Change Cipher Spe...	TLSv1.2: New Session Ticket, Change Cipher Spec, En...
33150	33150 → 8004 [ACK] Seq=3009 Ack=36...	TCP: 33150 → 8004 [ACK] Seq=3009 Ack=3634 Win=6..
33150	Application Data	TLSv1.2: Application Data
33150	8004 → 33150 [ACK] Seq=3634 Ack=30...	TCP: 8004 → 33150 [ACK] Seq=3634 Ack=3065 Win=6..
33150	Application Data	TLSv1.2: Application Data
33150	33150 → 8004 [ACK] Seq=3065 Ack=36...	TCP: 33150 → 8004 [ACK] Seq=3065 Ack=3687 Win=6..
33150	Application Data	TLSv1.2: Application Data
33150	8004 → 33150 [ACK] Seq=3687 Ack=30...	TCP: 8004 → 33150 [ACK] Seq=3687 Ack=3099 Win=6..
33150	33150 → 8004 [FIN, ACK] Seq=3099 Ack...	TCP: 33150 → 8004 [FIN, ACK] Seq=3099 Ack=3687 W..
33150	8004 → 33150 [FIN, ACK] Seq=3687 Ack...	TCP: 8004 → 33150 [FIN, ACK] Seq=3687 Ack=3099 W..
33150	33150 → 8004 [ACK] Seq=3100 Ack=36...	TCP: 33150 → 8004 [ACK] Seq=3100 Ack=3688 Win=6..
33150	8004 → 33150 [ACK] Seq=3688 Ack=31...	TCP: 8004 → 33150 [ACK] Seq=3688 Ack=3100 Win=6..

Task – 3: START_SSL downgrade attack #1 for eavesdropping

Setup and message flow for secure_chat_interceptor:

In secure_chat_interceptor we have secure_chat_interceptor.cpp which takes command line arguments.

For server and client we created two functions in secure_chat_app as above.

In addition we have functions for performing the specific attacks as mentioned.

Commands:

To poison the etc/hosts file of Alice, Bob containers-

```
bash poison-dns-alice1-bob1.sh
```

For compiling the code (Client, Server)-

```
secure_chat/home/ubuntu/secure_chat_alice/secure_chat# g++ secure_chat_app.cpp -lssl -lcrypto -o secure_chat_app
```

For compiling the code (Trudy)-

```
secure_chat/home/ubuntu/secure_chat_alice/secure_chat# g++  
secure_chat_interceptor.cpp -lssl -lcrypto -o secure_chat_interceptor
```

For server:

```
./secure_chat_app -s <hostname> <portNo>
```

For client:

```
./secure_chat_app -c <hostname> <portNo>
```

For Trudy:

```
./secure_chat_interceptor -d alice1 bob1 <portNo>
```

For collecting traffic data and saving in .pcap file we use tcpdump-
tcpdump -i eth0 -w TASK3.pcap

TLS protocol downgrade attack for eavesdropping:

- The function fake_connection(clienthostname, serverhostname) allows to create fake connections for the downgrade attack.
- In the fake_connection() function a fake server socket and a fake client socket are created which is then connected with Alice and Bob (Client and server).
- Once the 'chat_hello' is received from Client, Trudy forwards it to Server. Again when it receives 'chat_reply' from Server, it forwards to Client.
- When Trudy receives the 'start_tls' from Client, it blocks the message from reaching Server and sends fake 'start_tls not supported' message to Client.
- Then Client will not initiate TLS session and proceed with unsecure chat over socket. And the connection between Trudy and server proceeds as usual with TCP.
- Hence downgrade attack by Trudy will be Successful.
- After receiving 'term' message from either Client or Server, the connection is terminated after closing the tcp and tls sockets properly.

Screenshots:

With Alice-


```

root@alice1:~/Alice# ./secure_chat_app -c bob1 8001
IP address of Client:
172.31.0.4
Client Socket Created
Enter message for Server:
chat_request
Received From Server: chat_reply
Enter message for Server:
start_tls
Received From Server: start_tls_not_supported
Enter message for Server on socket:
Hello Alice
Server received on socket: Hello Bob
Enter message for Server on socket:
GoodBye.
Server received on socket: Ok Bye.
Enter message for Server on socket:
term
Connection Terminated

```

With Bob-

```

root@bob1:~/Bob# ./secure_chat_app -s 8001
.....Waiting for client connection.....
Client request accepted from IP: 172.31.0.4
Received From Client: chat_request
Enter message for Client:
chat_reply
Received From Client: Hello Alice
Enter message for Client:
Hello Bob
Received from Client on socket: GoodBye.
Enter message for Client on socket:
Ok Bye.
Connection terminated from Client

```

With Trudy-

```

root@trudy1:~/Trudy# ./secure_chat_interceptor -d alice1 bob1 8001
.....Waiting for client connection.....
Client request accepted from ip: 172.31.0.2
Client socket created
Received message from Client on socket: chat_request
Received message from Server on socket: chat_reply
Received message from Client on socket: start_tls
Enter Fake TLS not supported message for Client: start_tls_not_supported
Received message from Client on socket: Hello Alice
Received message from Server on socket: Hello Bob
Received message from Client on socket: GoodBye.
Received message from Server on socket: Ok Bye.
Received message from Client on socket: term
Connection terminated from Client.

```

Trudy successfully performed downgrade attack by sending 'start_tls_not_supported' to client without the server having knowledge of 'start_tls' message-

Source	Destination	Info	Protocol
172.31.0.4	172.31.0.2	8001 → 51032 [PSH, ACK] Seq=12 Ack=24 Win=65152 Len=24 TSval=2266084290 TSe...	TCP
Frame 17: 90 bytes on wire (720 bits), 90 bytes captured (720 bits)			
Ethernet II, Src: Xensourc_cb:b1:38 (00:16:3e:cb:b1:38), Dst: Xensourc_6c:6d:b0 (00:16:3e:6c:6d:b0)			
Internet Protocol Version 4, Src: 172.31.0.4, Dst: 172.31.0.2			
Transmission Control Protocol, Src Port: 8001, Dst Port: 51032, Seq: 12, Ack: 24, Len: 24			
Data (24 bytes)			
Data: 73746172745f746c735f6e6f745f737570706f7274656400			
[Length: 24]			
0000	00 16 3e 6c 6d b0	00 16 3e cb b1 38 08 00 45 00	..>lm.. >..8..E.
0010	00 4c b6 73 40 00 40 06	2b f4 ac 1f 00 04 ac 1f	..L.s@.@. +.....
0020	00 02 1f 41 c7 58 b6 0e	cf ca 87 ed cc 19 80 18	...A.X.....
0030	01 fd 58 83 00 00 01 01	08 0a 87 11 b3 c2 70 05	..X.....p.
0040	a7 16 73 74 61 72 74 5f	74 6c 73 5f 6e 6f 74 5f	..start_tls_not_
0050	73 75 70 70 6f 72 74 65	64 00	supporte d.

Thus message continues without encryption over socket-

Source	Destination	Info	Protocol
172.31.0.2	172.31.0.4	51032 → 8001 [PSH, ACK] Seq=24 Ack=36 Win=64256 Len=12 TSval=1879445803 TSe...	TCP
Frame 19: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)			
Ethernet II, Src: Xensourc_6c:6d:b0 (00:16:3e:6c:6d:b0), Dst: Xensourc_cb:b1:38 (00:16:3e:cb:b1:38)			
Internet Protocol Version 4, Src: 172.31.0.2, Dst: 172.31.0.4			
Transmission Control Protocol, Src Port: 51032, Dst Port: 8001, Seq: 24, Ack: 36, Len: 12			
Data (12 bytes)			
Data: 48656c6c6f20416c69636500			
[Length: 12]			
0000	00 16 3e cb b1 38	00 16 3e 6c 6d b0 08 00 45 00	..>..8.. >lm...E.
0010	00 40 9d f4 40 00 40 06	44 7f ac 1f 00 02 ac 1f	..@..@..@. D.....
0020	00 04 c7 58 1f 41 87 ed	cc 19 b6 0e cf e2 80 18	...X.A.....
0030	01 f6 58 77 00 00 01 01	08 0a 70 06 11 2b 87 11	..Xw.....p..+..
0040	b3 c2 48 65 6c 6c 6f 20	41 6c 69 63 65 00	..Hello Alice.

Source	Destination	Info	Protocol
172.31.0.4	172.31.0.3	50972 → 8001 [PSH, ACK] Seq=14 Ack=12 Win=64256 Len=12 TSval=1255176249 TSe...	TCP
Frame 21: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)			
Ethernet II, Src: Xensourc_cb:b1:38 (00:16:3e:cb:b1:38), Dst: Xensourc_ec:4b:d4 (00:16:3e:ec:4b:d4)			
Internet Protocol Version 4, Src: 172.31.0.4, Dst: 172.31.0.3			
Transmission Control Protocol, Src Port: 50972, Dst Port: 8001, Seq: 14, Ack: 12, Len: 12			
Data (12 bytes)			
Data: 48656c6c6f20416c69636500			
[Length: 12]			
0000	00 16 3e ec 4b d4 00 16	3e cb b1 38 08 00 45 00	..>.K... >..8..E.
0010	00 40 54 8c 40 00 40 06	8d e6 ac 1f 00 04 ac 1f	..@T@.@.
0020	00 03 c7 1c 1f 41 ed f4	ee 00 d2 c8 ea f7 80 18A.....
0030	01 f6 58 78 00 00 01 01	08 0a 4a d0 78 39 2b 4d	..Xx.....J.x9+M
0040	3d 47 48 65 6c 6c 6f 20	41 6c 69 63 65 00	=Hello Alice.

Flow Graph-

172.31.0.2

172.31.0.4

172.31.0.3
Comment

51032	51032 → 8001 [SYN] Seq=0 Win=64240 ...	8001	TCP: 51032 → 8001 [SYN] Seq=0 Win=64240 Len=0 M.
51032	8001 → 51032 [SYN, ACK] Seq=0 Ack=1 ...	8001	TCP: 8001 → 51032 [SYN, ACK] Seq=0 Ack=1 Win=651
51032	51032 → 8001 [ACK] Seq=1 Ack=1 Win=...	8001	TCP: 51032 → 8001 [ACK] Seq=1 Ack=1 Win=64256 L..
50972	50972 → 8001 [SYN] Seq=0 Win=64240 ...	8001	TCP: 50972 → 8001 [SYN] Seq=0 Win=64240 Len=0 M.
50972	8001 → 50972 [SYN, ACK] Seq=0 Ack=1 ...	8001	TCP: 8001 → 50972 [SYN, ACK] Seq=0 Ack=1 Win=651
50972	50972 → 8001 [ACK] Seq=1 Ack=1 Win=...	8001	TCP: 50972 → 8001 [ACK] Seq=1 Ack=1 Win=64256 L..
51032	51032 → 8001 [PSH, ACK] Seq=1 Ack=1 ...	8001	TCP: 51032 → 8001 [PSH, ACK] Seq=1 Ack=1 Win=642
51032	8001 → 51032 [ACK] Seq=1 Ack=14 Win...	8001	TCP: 8001 → 51032 [ACK] Seq=1 Ack=14 Win=65152 ..
50972	50972 → 8001 [PSH, ACK] Seq=1 Ack=1 ...	8001	TCP: 50972 → 8001 [PSH, ACK] Seq=1 Ack=1 Win=642
50972	8001 → 50972 [ACK] Seq=1 Ack=14 Win...	8001	TCP: 8001 → 50972 [ACK] Seq=1 Ack=14 Win=65152 ..
50972	8001 → 50972 [PSH, ACK] Seq=1 Ack=1...	8001	TCP: 8001 → 50972 [PSH, ACK] Seq=1 Ack=14 Win=65
50972	50972 → 8001 [ACK] Seq=14 Ack=12 Wi...	8001	TCP: 50972 → 8001 [ACK] Seq=14 Ack=12 Win=64256.
51032	8001 → 51032 [PSH, ACK] Seq=1 Ack=1...	8001	TCP: 8001 → 51032 [PSH, ACK] Seq=1 Ack=14 Win=65
51032	51032 → 8001 [ACK] Seq=14 Ack=12 Wi...	8001	TCP: 51032 → 8001 [ACK] Seq=14 Ack=12 Win=64256.
51032	51032 → 8001 [PSH, ACK] Seq=14 Ack=...	8001	TCP: 51032 → 8001 [PSH, ACK] Seq=14 Ack=12 Win=6
51032	8001 → 51032 [ACK] Seq=12 Ack=24 Wi...	8001	TCP: 8001 → 51032 [ACK] Seq=12 Ack=24 Win=65152.
51032	8001 → 51032 [PSH, ACK] Seq=12 Ack=...	8001	TCP: 8001 → 51032 [PSH, ACK] Seq=12 Ack=24 Win=6
51032	51032 → 8001 [ACK] Seq=24 Ack=36 Wi...	8001	TCP: 51032 → 8001 [ACK] Seq=24 Ack=36 Win=64256.
51032	51032 → 8001 [PSH, ACK] Seq=24 Ack=...	8001	TCP: 51032 → 8001 [PSH, ACK] Seq=24 Ack=36 Win=6
51032	8001 → 51032 [ACK] Seq=36 Ack=36 Wi...	8001	TCP: 8001 → 51032 [ACK] Seq=36 Ack=36 Win=65152.
50972	50972 → 8001 [PSH, ACK] Seq=14 Ack=...	8001	TCP: 50972 → 8001 [PSH, ACK] Seq=14 Ack=12 Win=6
50972	8001 → 50972 [ACK] Seq=12 Ack=26 Wi...	8001	TCP: 8001 → 50972 [ACK] Seq=12 Ack=26 Win=65152.
50972	8001 → 50972 [PSH, ACK] Seq=12 Ack=...	8001	TCP: 8001 → 50972 [PSH, ACK] Seq=12 Ack=26 Win=6
50972	50972 → 8001 [ACK] Seq=26 Ack=22 Wi...	8001	TCP: 50972 → 8001 [ACK] Seq=26 Ack=22 Win=64256.
51032	8001 → 51032 [PSH, ACK] Seq=36 Ack=...	8001	TCP: 8001 → 51032 [PSH, ACK] Seq=36 Ack=36 Win=6
51032	51032 → 8001 [ACK] Seq=36 Ack=46 Wi...	8001	TCP: 51032 → 8001 [ACK] Seq=36 Ack=46 Win=64256.
51032	51032 → 8001 [PSH, ACK] Seq=36 Ack=...	8001	TCP: 51032 → 8001 [PSH, ACK] Seq=36 Ack=46 Win=6
51032	8001 → 51032 [ACK] Seq=46 Ack=45 Wi...	8001	TCP: 8001 → 51032 [ACK] Seq=46 Ack=45 Win=65152.
50972	50972 → 8001 [PSH, ACK] Seq=26 Ack=...	8001	TCP: 50972 → 8001 [PSH, ACK] Seq=26 Ack=22 Win=6
50972	8001 → 50972 [ACK] Seq=22 Ack=35 Wi...	8001	TCP: 8001 → 50972 [ACK] Seq=22 Ack=35 Win=65152.
50972	8001 → 50972 [PSH, ACK] Seq=22 Ack=...	8001	TCP: 8001 → 50972 [PSH, ACK] Seq=22 Ack=35 Win=6

Task – 4: START_SSL downgrade attack #2 for tampering

Setup and message flow for secure_chat_interceptor:

In secure_chat_interceptor we have secure_chat_interceptor.cpp which takes command line arguments.

For server and client we created two functions in secure_chat_app as above.

In addition we have functions for performing the specific attacks as mentioned.

Commands:

For compiling the code (Client, Server)-

```
secure_chat/home/ubuntu/secure_chat_alice/secure_chat# g++ secure_chat_app.cpp -lssl -lcrypto -o secure_chat_app
```

For compiling the code (Trudy)-

```
secure_chat/home/ubuntu/secure_chat_alice/secure_chat# g++ secure_chat_interceptor.cpp -lssl -lcrypto -o secure_chat_interceptor
```

For server:

```
./secure_chat_app -s <hostname> <portNo>
```

For client:

```
./secure_chat_app -c <hostname> <portNo>
```

For Trudy:

```
./secure_chat_interceptor -m alice1 bob1 <portNo>
```

For collecting traffic data and saving in .pcap file we use tcpdump-
tcpdump -i eth0 -w TASK4.pcap

TLS protocol downgrade attack for tampering:

- The function fake_connection(clienthostname, serverhostname) allows to create fake connections for the downgrade attack.
- In the fake_connection() function a fake server socket and a fake client socket are created which is then connected with Alice and Bob (Client and server).
- Once the 'chat_hello' is received from Client, Trudy forwards it to Server. Again when it receives 'chat_reply' from Server, it forwards to Client.
- Once Trudy receives the 'start_tls' from Client for the first time(which is checked using a 'start_tls_flag' to avoid repeated ssl wrapping), it sends it to the Server and receives the reply, and sends it back to Client.
- If Server sends 'start_tls_ack' then the Trudy creates two TLS pipes (one for Server and Trudy and other for Client and Trudy) and does the handshake.
- Further chat continues using the secured socket of TLS which generate encrypted application data.
- We have used 'term' as the termination message for all the programs.
- To depict the MITM functionality for tampering messages we have included the feature which allows Trudy to tamper the sent messages. This seriously affects meaningful communication and thus cause loss of information leading to miscommunication, misrepresentation and various losses.
- After receiving 'term' message from either Client or Server, the program is terminated after closing the TCP and TLS sockets properly.

Screenshots:

With Alice-

```

root@alice1:~/Alice# ./secure_chat_app -c bob1 8001

IP address of Client:
172.31.0.4

Client Socket Created

Enter message for Server:
chat_request

Received From Server: chat_reply

Enter message for Server:
start_tls

Received From Server: start_tls_ack

Client ctx created
C = IN, ST = OD, L = RKL, O = RSP, OU = NS, CN = Alice1.com
C = AK, ST = OT, L = BayBeach, O = GenP, OU = Dev, CN = Bob1.com
Server Certificate Valid

Enter message for Server on TLS:
Hello Alice. This is Bob. Transfer Rs200 to G.

Server received on tls: Hello Bob. Please confirm for Transfer Rs200 for G.

Enter message for Server on TLS:
Confirm.

Server received on tls: Done.

Enter message for Server on TLS:
term

Connection Terminated

```

With Bob-

```

root@bob1:~/Bob# ./secure_chat_app -s 8001
.....Waiting for client connection.....

Client request accepted from IP: 172.31.0.4

Received From Client: chat_request

Enter message for Client:
chat_reply

Received From Client: start_tls

Enter message for Client:
start_tls_ack

Server ctx created
C = EU, ST = London, L = RoseBay, O = EY, OU = NS, CN = Bob1.com 4here in this
C = IN, ST = TE, L = HYD, O = IIT, OU = ACN, CN = Alice1.com
Client Certificate Valid

Received from Client on TLS: Hello Alice. This is Bob. Transfer Rs20000 to Trudy.

Enter message for Client on TLS:
Hello Bob. Please confirm for Transfer Rs20000 to Trudy.

Received from Client on TLS: Confirm.

Enter message for Client on TLS:
Done.

Connection terminated from Client
root@bob1:~/Bob# █

```

With Trudy-

```

root@trudy1:~/Trudy# ./secure_chat_interceptor -m alice1 bob1 8001
.....Waiting for client connection.....

Client request accepted from ip: 172.31.0.2
Client socket created

From Alice on socket: chat_request
From Client on socket: chat_reply
From Alice on socket: start_tls
From Client on socket: start_tls_ack
Fake Server ctx created
Client Certificate Valid.
Fake Client ctx created
Server Certificate Valid.

From Client on TLS: Hello Alice. This is Bob. Transfer Rs200 to G.
Enter 1 to tamper:
1
Enter tampered data to send to Server:
Hello Alice. This is Bob. Transfer Rs20000 to Trudy.
From Server on TLS: Hello Bob. Please confirm for Transfer Rs20000 to Trudy.
Enter 1 to tamper:
1
Enter tampered data to send to Client:
Hello Bob. Please confirm for Transfer Rs200 for G.

```

```

From Client on TLS: Confirm.
Enter 1 to tamper:
0
From Server on TLS: Done.
Enter 1 to tamper:
0
From Client on TLS: term
Enter 1 to tamper:
0
Connection terminated.

```

```

Aborted (core dumped)
root@trudy1:~/Trudy# █

```

As a result of message tampering by Trudy, messages were changed. This leads to loss of proper meaning and causes confusion, misscommunication between Client and Server.
Creation of two TLSv1.2 pipes (2 handshakes)-

Source	Destination	Info	Protocol
172.31.0.2	172.31.0.4	Client Hello	TLSv1.2
172.31.0.4	172.31.0.2	8001 → 55804 [ACK] Seq=20 Ack=212 Win=65024 Len=0 TSval=2331534883 TSecr=1944885903	TCP
172.31.0.4	172.31.0.2	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done	TLSv1.2
172.31.0.2	172.31.0.4	55804 → 8001 [ACK] Seq=212 Ack=2518 Win=63744 Len=0 TSval=1944885908 TSecr=2331534890	TCP
172.31.0.2	172.31.0.4	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Hand...	TLSv1.2
172.31.0.4	172.31.0.2	8001 → 55804 [ACK] Seq=2518 Ack=3009 Win=63616 Len=0 TSval=2331534896 TSecr=1944885914	TCP
172.31.0.4	172.31.0.2	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message	TLSv1.2
172.31.0.4	172.31.0.3	Client Hello	TLSv1.2

<p>Frame 23: 254 bytes on wire (2032 bits), 254 bytes captured (2032 bits)</p> <p>Ethernet II, Src: Xensourc_6c:6d:b0 (00:16:3e:6c:6d:b0), Dst: Xensourc_cb:b1:38 (00:16:3e:cb:b1:38)</p> <p>Internet Protocol Version 4, Src: 172.31.0.2, Dst: 172.31.0.4</p> <p>Transmission Control Protocol, Src Port: 55804, Dst Port: 8001, Seq: 24, Ack: 26, Len: 188</p> <p>Transport Layer Security</p> <ul style="list-style-type: none"> TLSv1.2 Record Layer: Handshake Protocol: Client Hello <ul style="list-style-type: none"> Content Type: Handshake (22) Version: TLS 1.0 (0x0301) Length: 183 Handshake Protocol: Client Hello <ul style="list-style-type: none"> Handshake Type: Client Hello (1) Length: 179 Version: TLS 1.2 (0x0303) Random: bc91c30b8b963c8ddfb765ec44db51391e52bb4ef6633ced03c23dcb10d936f8 Session ID Length: 0 Cipher Suites Length: 56 Cipher Suites (28 suites) Compression Methods Length: 1 Compression Methods (1 method) Extensions Length: 82 Extension: ec_point_formats (len=4) Extension: supported_groups (len=12) Extension: session_ticket (len=0) Extension: encrypt_then_mac (len=0) Extension: extended_master_secret (len=0) Extension: signature_algorithms (len=42) [JA3 Fullstring: 771,49196-49200-159-52393-52392-52394-49195-49199-158-49188-49192-107-49187-49191-103-49162-491 [JA3: fbe7e189e37a07ee33706f86bc746344]

Encrypted messages between Trudy and Client/Server (Bob/Alice)-

Source	Destination	Info
172.31.0.2	172.31.0.4	Application Data

<p>Frame 39: 142 bytes on wire (1136 bits), 142 bytes captured (1136 bits)</p> <p>Ethernet II, Src: Xensourc_6c:6d:b0 (00:16:3e:6c:6d:b0), Dst: Xensourc_cb:b1:38 (00:16:3e:cb:b1:38)</p> <p>Internet Protocol Version 4, Src: 172.31.0.2, Dst: 172.31.0.4</p> <p>Transmission Control Protocol, Src Port: 55804, Dst Port: 8001, Seq: 3009, Ack: 3800, Len: 76</p> <p>Transport Layer Security</p> <ul style="list-style-type: none"> TLSv1.2 Record Layer: Application Data Protocol: Application Data <ul style="list-style-type: none"> Content Type: Application Data (23) Version: TLS 1.2 (0x0303) Length: 71 <p>Encrypted Application Data: c3c2e17404aa04d6359bc029b646941c97624171e202eb899f5c4c3b13dbf7f3ba</p>

0000	00 16 3e cb b1 38 00 16 3e 6c 6d b0 08 00 45 00	>...8...>lm...E.
0010	00 80 a4 2c 40 00 40 06 3e 07 ac 1f 00 02 ac 1f	...@.@...>.....
0020	00 04 d9 fc 1f 41 26 63 76 35 d1 fb 43 08 80 18	...A&c v5...C...
0030	01 f5 58 b7 00 00 01 01 08 0a 73 ed 0c b8 8a f8	...X...s...
0040	66 31 17 03 03 00 47 c3 c2 e1 74 04 aa 04 d6 35	f1...G...t...5
0050	9b c0 29 b6 46 94 1c 97 62 41 71 e2 02 eb 89 9f	...)F...bAq....
0060	5c 4c 3b 13 db f7 f3 ba be c2 65 ef 86 3a 3f 60	\L;.....e...?;
0070	66 38 55 37 41 47 fa 05 c3 52 07 72 25 07 05 b3	f8U7AG...R.r%...
0080	f2 5c 34 3f 8f 22 a4 32 71 7a da 56 7a a9	.\4?..."2 qz.VZ.

Source	Destination	Info
172.31.0.4	172.31.0.3	Application Data

<p>Frame 41: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits)</p> <p>Ethernet II, Src: Xensourc_cb:b1:38 (00:16:3e:cb:b1:38), Dst: Xensourc_ec:4b:d4 (00:16:3e:ec:4b:d4)</p> <p>Internet Protocol Version 4, Src: 172.31.0.4, Dst: 172.31.0.3</p> <p>Transmission Control Protocol, Src Port: 42068, Dst Port: 8001, Seq: 3010, Ack: 3634, Len: 82</p> <p>Transport Layer Security</p> <ul style="list-style-type: none"> TLSv1.2 Record Layer: Application Data Protocol: Application Data <ul style="list-style-type: none"> Content Type: Application Data (23) Version: TLS 1.2 (0x0303) Length: 77 <p>Encrypted Application Data: d9c2980692a42fe9d16f3c75e084eca4653af03c09359842452b3dcebe6c912d2c</p>

0000	00 16 3e ec 4b d4 00 16 3e cb b1 38 08 00 45 00	>...K...>...8...E.
0010	00 86 5c 4f 40 00 40 06 85 dd ac 1f 00 04 ac 1f	...\0@.@...>.....
0020	00 03 a4 54 1f 41 f5 cc 24 6d 63 b3 94 6d 80 18	...T.A...\$mc...m...
0030	01 f5 58 be 00 00 01 01 08 0a 4e b7 c2 92 2f 34	...X...N.../4
0040	55 02 17 03 03 00 4d d9 c2 98 06 92 a4 2f e9 d1	U...M.../...
0050	0f 3c 75 e0 84 ec a4 65 3a f0 3c 09 35 98 42 45	0<u...e...<5.BE
0060	2b 3d ce be 6c 91 2d 2c a8 58 26 2e 8d 1f 9c 85	+...l...X&.....
0070	54 77 d3 a8 82 79 25 47 2f e4 a0 ff e8 5a c4 57	Tw...y%G /...Z.W
0080	e5 e0 70 6a 43 91 62 34 b6 e4 18 69 a0 cb 52 31	..pjC.b4...i...R1
0090	66 22 5e 6c	f"^\1

Flow Graph-

172.31.0.2

172.31.0.3
Comment

172.31.0.4

49800	49800 → 8001 [SYN] Seq=0 Win=64240 ...	8001	TCP: 49800 → 8001 [SYN] Seq=0 Win=64240 Len=0 M
49800	8001 → 49800 [SYN, ACK] Seq=0 Ack=1 ...	8001	TCP: 8001 → 49800 [SYN, ACK] Seq=0 Ack=1 Win=65'
49800	49800 → 8001 [ACK] Seq=1 Ack=1 Win=...	8001	TCP: 49800 → 8001 [ACK] Seq=1 Ack=1 Win=64256 L.
		47004	47004 → 8001 [SYN] Seq=0 Win=64240 ...
		47004	8001 → 47004 [SYN, ACK] Seq=0 Ack=1 ...
		47004	47004 → 8001 [ACK] Seq=1 Ack=1 Win=...
49800	49800 → 8001 [PSH, ACK] Seq=1 Ack=1 ...	8001	TCP: 49800 → 8001 [PSH, ACK] Seq=1 Ack=1 Win=642
49800	8001 → 49800 [ACK] Seq=1 Ack=14 Win=...	8001	TCP: 8001 → 49800 [ACK] Seq=1 Ack=14 Win=65152
		47004	47004 → 8001 [PSH, ACK] Seq=1 Ack=1 ...
		47004	8001 → 47004 [ACK] Seq=1 Ack=14 Win=...
		47004	8001 → 47004 [PSH, ACK] Seq=1 Ack=1 ...
		47004	47004 → 8001 [ACK] Seq=14 Ack=12 Win=...
49800	8001 → 49800 [PSH, ACK] Seq=1 Ack=1 ...	8001	TCP: 8001 → 49800 [PSH, ACK] Seq=1 Ack=14 Win=65
49800	49800 → 8001 [ACK] Seq=14 Ack=12 Wi...	8001	TCP: 49800 → 8001 [ACK] Seq=14 Ack=12 Win=64256
49800	49800 → 8001 [PSH, ACK] Seq=14 Ack=...	8001	TCP: 49800 → 8001 [PSH, ACK] Seq=14 Ack=12 Win=6
49800	8001 → 49800 [ACK] Seq=12 Ack=24 Wi...	8001	TCP: 8001 → 49800 [ACK] Seq=12 Ack=24 Win=65152
		47004	47004 → 8001 [PSH, ACK] Seq=14 Ack=...
		47004	8001 → 47004 [ACK] Seq=12 Ack=24 Wi...
		47004	8001 → 47004 [PSH, ACK] Seq=12 Ack=...
		47004	47004 → 8001 [ACK] Seq=24 Ack=26 Wi...
49800	8001 → 49800 [PSH, ACK] Seq=12 Ack=...	8001	TCP: 8001 → 49800 [PSH, ACK] Seq=12 Ack=24 Win=6
49800	49800 → 8001 [ACK] Seq=24 Ack=26 Wi...	8001	TCP: 49800 → 8001 [ACK] Seq=24 Ack=26 Win=64256
49800	Client Hello	8001	TLSv1.2: Client Hello
49800	8001 → 49800 [ACK] Seq=26 Ack=212 ...	8001	TCP: 8001 → 49800 [ACK] Seq=26 Ack=212 Win=6502
49800	Server Hello, Certificate, Server Key Ex...	8001	TLSv1.2: Server Hello, Certificate, Server Key Exchan.
49800	49800 → 8001 [ACK] Seq=212 Ack=251 ...	8001	TCP: 49800 → 8001 [ACK] Seq=212 Ack=2518 Win=63
49800	Certificate, Client Key Exchange, Certifi...	8001	TLSv1.2: Certificate, Client Key Exchange, Certificate
49800	8001 → 49800 [ACK] Seq=2518 Ack=30...	8001	TCP: 8001 → 49800 [ACK] Seq=2518 Ack=3009 Win=6
49800	New Session Ticket, Change Cipher Spe...	8001	TLSv1.2: New Session Ticket, Change Cipher Spec, En
		47004	Client Hello
			TLSv1.2: Client Hello

Files submitted

- CS22MTECH14009|CS23RESCH01001.tgz

- Alice

- alice.crt
- alice.pem
- alicecert.pem
- alicecsr.pem

- Bob

- bob.crt
- bob.pem
- bobcert.pem
- bobcsr.pem

- Trudy

- fake_alice
 - fake_alice.crt
 - fake_alicecsr.pem
- fake_bob
 - fake_bob.crt
 - fake_bobcsr.pem
- fake_alice.pem
- fake_alicecert.pem
- fake_bob.pem
- fake_bobcert.pem

- Root

- Intermediate
 - interCA.crt
 - interCA.pem
 - interCAcsr.pem
- Root
 - root.crt
 - rootCA.pem
 - rootCAcsr.pem
- ExtCA.ext
- rootCA.crt

- pcap files

- TASK2.pcap
- TASK3.pcap
- TASK4.pcap

- secure_chat_app.cpp

- secure_chat_interceptor.cpp

- Report.pdf

- Makefile

Steps to run (README):

- Download and extract CS22MTECH14009|CS23RESCH01001.tgz
- Open Terminal in present folder with MakeFile.
- Below command to transfer the files/folders to the vm

For folders

```
scp -r <foldername> ubuntu@10.200.13.113:
```

For files

```
scp <filename> ubuntu@10.200.13.113:
```

```
manisha@manisha-HP-Laptop-15-bs1xx:~/Desktop$ scp -r CS22MTECH14009|CS23RESCH01001 ubuntu@10.200.13.113:
```

- Connect to assigned vm with below command

```
ssh ubuntu@10.200.13.113
```
- Once logged in we move to the directory CS22MTECH14009|CS23RESCH01001
- Here we will push the files including certificate, keys and .cpp codes to the respective containers with Makefile lxc commands
- In each container the rootCA.crt needs to be transferred to the trust store.

```
cp rootCA.crt /usr/local/share/ca-certificates/
```

Task2:

- Start the server by running secure_chat_app in server mode

```
./secure_chat_app -s 8001
```

```
root@bob1:~/Bob# ./secure_chat_app -s 8001  
.....Waiting for client connection.....
```

- Start the client by running secure_chat_a in client mode

```
./secure_chat_app -c bob1 8001
```

```
root@alice1:~/Alice# ./secure_chat_app -c bob1 8001
```

Task3:

- First we poison the DNS with below command

```
bash poison-dns-alice1-bob1.sh
```

```
ubuntu@ns00-gold:~$ bash poison-dns-alice1-bob1.sh
```

- Start the server by running secure_chat_app in server mode

```
./secure_chat_app -s 8001
```

```
root@bob1:~/Bob# ./secure_chat_app -s 8001  
.....Waiting for client connection.....
```

- Start the interceptor by running secure_chat_interceptor in downgrade attack mode

```
./secure_chat_interceptor -d alice1 bob1 8001
```

```
root@trudy1:~/Trudy# ./secure_chat_interceptor -m alice1 bob1 8001  
.....Waiting for client connection.....
```

- Start the client by running secure_chat_a in client mode

```
./secure_chat_app -c bob1 8001
```

```
root@alice1:~/Alice# ./secure_chat_app -c bob1 8001
```

- e) After execution is completed, we unpoison the DNS

```
bash unpoison-dns-alice1-bob1.sh
```

```
ubuntu@ns00-gold:~$ bash unpoison-dns-alice1-bob1.sh
```

Task4:

- a) First we poison the DNS with below command

```
bash poison-dns-alice1-bob1.sh
```

```
ubuntu@ns00-gold:~$ bash poison-dns-alice1-bob1.sh
```

- b) Start the server by running secure_chat_app in server mode

```
./secure_chat_app -s 8001
```

```
root@bob1:~/Bob# ./secure_chat_app -s 8001  
.....Waiting for client connection.....
```

- c) Start the interceptor by running secure_chat_interceptor in downgrade attack mode

```
./secure_chat_interceptor -m alice1 bob1 8001
```

```
root@trudy1:~/Trudy# ./secure_chat_interceptor -m alice1 bob1 8001  
.....Waiting for client connection.....
```

- d) Start the client by running secure_chat_a in client mode

```
./secure_chat_app -c bob1 8001
```

```
root@alice1:~/Alice# ./secure_chat_app -c bob1 8001
```

- e) After execution is completed, we unpoison the DNS

```
bash unpoison-dns-alice1-bob1.sh
```

```
ubuntu@ns00-gold:~$ bash unpoison-dns-alice1-bob1.sh
```

Important code snippets: Task2-

```
//-----  
//Function for certificate setup  
//-----  
void configureCertificates(SSL_CTX* ctx, char* CertFile, char* KeyFile)  
{  
    if ( SSL_CTX_use_certificate_file(ctx, CertFile, SSL_FILETYPE_PEM) <= 0 )  
    {  
        cout<<"\nCertificate file not valid"<<endl;  
        ERR_print_errors_fp(stderr);  
        abort();  
    }  
    if ( SSL_CTX_use_PrivateKey_file(ctx, KeyFile, SSL_FILETYPE_PEM) <= 0 )  
    {  
        cout<<"\nKey file not valid"<<endl;  
        ERR_print_errors_fp(stderr);  
        abort();  
    }  
    if ( !SSL_CTX_check_private_key(ctx) )  
    {  
        cout<<"\nKey not match with certificate file"<<endl;  
        abort();  
    }  
}
```

The function takes 3 argument- an SSL_CTX pointer called ctx and two character pointers called CertFile and KeyFile.

- SSL_CTX* **ctx**: a pointer to an SSL_CTX object, which is a data structure that holds the configuration information for SSL/TLS connections.
- char* **CertFile**: a pointer to a string that represents the path to the PEM-encoded certificate file.
- char* **KeyFile**: a pointer to a string that represents the path to the PEM-encoded private key file.

The first conditional statement verifies if the certificate file at the location specified by **CertFile** is valid. If it is not valid, an error message is displayed and the program is aborted.

The second conditional statement verifies if the private key file at the location specified by **KeyFile** is valid. If it is not valid, an error message is displayed and the program is aborted.

The third conditional statement checks if the private key matches the certificate file. If it doesn't match, an error message is displayed and the program is aborted.

```
//-----  
//Function to verify certificate  
//-----  
int verify_the_certificate(SSL *ssl)  
{  
    int result;  
    X509 *cert = SSL_get_peer_certificate(ssl);  
    if (cert == nullptr)  
    {  
        ERR_print_errors_fp(stderr);  
        cout<<"\nCertificate Not Given by Peer"<<endl;  
        abort();  
    }  
    int err = SSL_get_verify_result(ssl);  
    if (err != X509_V_OK)  
    {  
        ERR_print_errors_fp(stderr);  
        const char *err_string = X509_verify_cert_error_string(err);  
        printf("\nCertificate Not Valid : %s\n", err_string);  
        abort();  
    }  
    result = err;  
    return result;  
},,
```

- Inside the function, `ssl` pointer retrieves the peer's certificate using the **SSL_get_peer_certificate** function, which returns an X509 certificate pointer. If the peer's certificate is not provided, the function displays an error message using **ERR_print_errors_fp** function, then aborts the program.
- Next, the function retrieves the verification result of the certificate using **SSL_get_verify_result** function. If the certificate is not valid, an error message is displayed using **ERR_print_errors_fp** function and the **X509_verify_cert_error_string** function is called to obtain a human-readable error string. Then, the program is aborted.
- Finally, the function sets the verification result to a variable called **result** and returns it.

```
//-----
//client function
//-----
int client(const char *hostname, int port)
{
    int client_sd;
    X509* cert;
    X509* peer_cert;
    SSL_CTX *ctx;
    int start_tls_flag = 0;
    int start_comm_flag = 0;
    SSL *ssl;
    char send_buffer[MAX];
    char receive_buffer[MAX];
    struct hostent *host;
    struct sockaddr_in addr;
```

- The client function takes two arguments: a hostname (represented as a C-style string) and a port number. It returns an integer value indicating the success or failure of the communication process.
- The function first creates a TCP socket using the **socket()** system call and initializes a **sockaddr_in** structure with the server's IP address and port number. It then connects to the server using the **connect()** system call. If the connection fails, it prints an error message and aborts the program.
- After establishing the connection, the function prompts the user to enter a message to send to the server. It reads the user's input, copies it into a buffer, and sends it to the server using the **send()** system call. It then waits for a response from the server using the **recv()** system call and prints the received message to the console.
- If the received message is "**start_tls_ack**", the function initiates a TLS handshake with the server. It uses the OpenSSL library to create a TLS/SSL context object and configure it with the client's X.509 certificate and private key. It then sets up a new SSL connection using the **SSL_new()** function and attaches it to the existing TCP socket using the **SSL_set_fd()** function. Finally, it performs the TLS handshake using the **SSL_connect()** function.
- If the TLS handshake succeeds, the function enters a loop that allows the user to send messages to the server over the encrypted channel. It reads the user's input, sends it to the server using the **SSL_write()** function, and waits for a response using the **SSL_read()** function. If the received message is "term", the function terminates the loop and closes the connection.
- If the TLS handshake fails, the function enters a similar loop that sends messages to the server over the original unencrypted channel.

```

int server(int port)
{
    X509 *cert;
    X509 *peer_cert;
    int start_tls_flag = 0;
    int start_comm_flag = 0;
    char send_buffer[MAX];
    char receive_buffer[MAX];
    SSL *ssl;
    int server_sd;
    SSL_CTX *ctx;
    int connection;
    struct sockaddr_in addr, client_addr;
    server_sd = socket(AF_INET, SOCK_STREAM, 0);
    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = INADDR_ANY;

```

- The function **server()** takes an integer port as input and returns an integer.
- The function initializes various variables and structures, including **cert** and **peer_cert** of type X509, **send_buffer** and **receive_buffer** of type char, **ssl** of type SSL, **server_sd** of type int, **ctx** of type SSL_CTX, and **addr** and **client_addr** of type struct sockaddr_in.
- The function creates a socket using the **socket()** function, sets up the **addr** structure, binds the socket to the specified port using **bind()**, and starts listening for incoming connections using **listen()**.
- When a client connects to the server, the function accepts the connection using **accept()**, receives a message from the client using **recv()**, and checks the message for a specific string. If the string is "**term**", the function closes the connection and aborts. If the string is "**chat_request**", the function sets a flag to indicate that a chat session has started.
- The function prompts the user to enter a message to send to the client using **getline()**, copies the message into **send_buffer**, and sends the message to the client using **send()**.
- If the client sends a message "**start_tls**", the function sets up a TLS connection with the client by initializing the SSL library using **SSL_library_init()**, creating a new SSL context using **InitCTX()**, configuring the context with server certificates using **configureCertificates()**, setting the context to verify the client's certificate using **SSL_CTX_set_verify()**, creating a new SSL object using **SSL_new()**, setting the SSL object to use the established connection using **SSL_set_fd()**, and performing the SSL handshake using **SSL_accept()**.
- If the SSL handshake is successful, the function verifies the client's certificate using **verify_the_certificate()**, sets a flag to indicate that TLS has started, and enters a loop to receive messages from the client and print them to the console.
- If the client does not send a "**start_tls**" message, the function simply prompts the user to enter another message to send to the client and repeats the process.

Task3-

```

int mitm_attack_1(int connection, int client_sd)
{
    char sendMessageAlice[MAX];
    char receiveMessageAlice[MAX];
    char receiveMessageBob[MAX];
    char sendMessageBob[MAX];
    int tls_flag = 0;
    while(true)
    {
        int alice_msg = recv(connection, receiveMessageAlice, MAX, 0);
        if(alice_msg < 0)
        {
            cout << "Message not recieved from Client." << endl;
            return 0;
        }
    }
}

```

- It takes two parameters, "**connection**" and "**client_sd**"(connection to the client that the attacker is impersonating), which are integer values representing sockets.
- The function begins by declaring four character arrays: "**sendMessageAlice**", "**receiveMessageAlice**", "**receiveMessageBob**", and "**sendMessageBob**". It also sets a variable "**tls_flag**" to 0.
- The function uses a while loop to continuously intercept and manipulate messages until the connection is terminated. The loop starts by receiving a message from the client using **recv()** function and storing it in **receiveMessageAlice**. If there is an error in receiving the message, the function returns 0.
- If the received message from the client starts with the string "**start_tls**", the attacker sends a fake message to the client indicating that TLS is not supported. The attacker sets the **tls_flag** to 1 to indicate that the client should not try to use TLS.
- The attacker then forwards the message from the client to the server by copying the content of **receiveMessageAlice** to **sendMessageBob**. If the received message from the client starts with "**term**", the attacker forwards the message to the server and terminates the connection by breaking out of the loop.
- The attacker then receives a message from the server using **recv()** function and stores it in **receiveMessageBob**. If there is an error in receiving the message, the loop is terminated.
- The attacker then forwards the message from the server to the client by copying the content of **receiveMessageBob** to **sendMessageAlice**. If the received message from the server starts with "**term**", the attacker forwards the message to the client and terminates the connection by breaking out of the loop.
- Once the loop is terminated, the connections to the client and server are closed using **close()** function.

Task4-

```

int mitm_attack_2(int connection, int client_sd)
{
    char sendMessageAlice[MAX]; //client
    char receiveMessageAlice[MAX];
    char receiveMessageBob[MAX]; //server
    char sendMessageBob[MAX];
    X509* cert;
    X509* peer_cert;
    SSL_CTX *ctx_client;
    SSL_CTX *ctx_server;
    SSL *ssl_client;
    SSL *ssl_server;
    int client_ver = 0;
    int server_ver = 0;
    int start_tls_flag = 0;
    int start_comm_flag = 1;
    int count = 2;
    string s;
}

```

- The **mitm_attack_2** function takes two integer arguments **connection** and **client_sd** which are socket descriptors for the connection.
- The function sets up two SSL contexts, one for the fake server and the other for the fake client, and two SSL objects for the SSL connections with Client and Server. It initializes these SSL objects with fake certificates and establishes a TLS connection with both parties.
- The function then enters into a loop to intercept and modify messages between Client and Server.
- If it receives the message "**start_tls_ack**" from Server and if the start_tls_flag is not set, it initializes the fake TLS connections with Client and Server.
- The function then enters into another loop to receive messages over the fake TLS connection. Once a message is received (from Client/Server), then it outputs the message to the console(Trudy) and prompts to enter "1" to tamper with the message. If the attacker enters "1", the function modifies the message (as per input by attacker) and sends the modified message to the Server/Client(other party).
- The function continues this loop until the connection is terminated by either party or until there is an error in the connection.