Assignment 3: Secure chat using openssl and MITM attacks

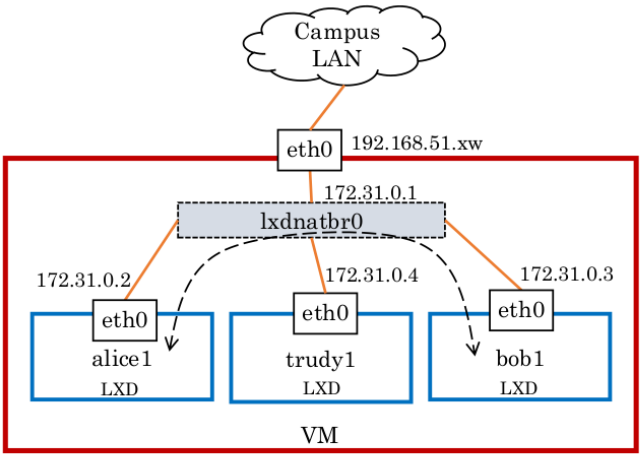
It's a group project with Max of 3 students per group playing the roles of Alice/Bob/Trudy!

Team - Ajinkya Bokade (CS17BTECH11001)
Niraj Kamble (CS17BTECH11024)
M Sai Subodh (CS17BTECH11023)

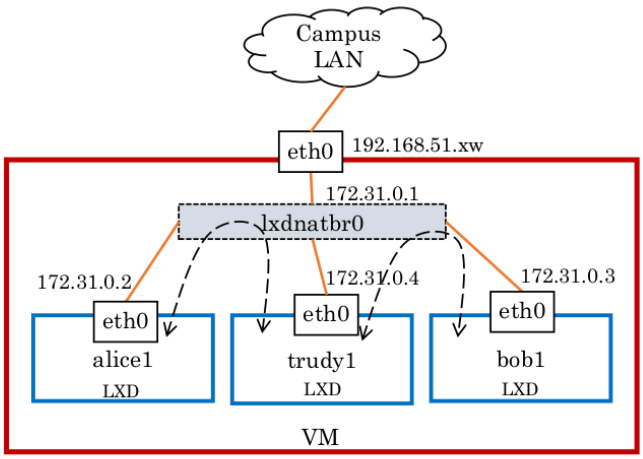
In this programming assignment, your group will implement a secure peer-to-peer chat application using openssl in C/C++ or Python and demonstrate how Alice and Bob could chat with each other using it. Plus you will also implement evil Trudy who is going to intercept the chat messages between Alice and Bob by launching various MITM attacks.

Setup:

Each group will be given the IP address (192.168.51.xw) of QEMU-KVM VM in NeWS lab's cloud. The VM runs three LXD containers (one LXD container each for Alice, Trudy and Bob in the VM provided) which are configured in a star topology i.e., With the switch at the center, Alice, Bob, Trudy are connected to switch ports. Make a note of hostnames assigned to Alice, Trudy and Bob from your VM. DNS is set up properly so ping using names from Alice to Bob and vice-versa works. Under normal circumstances, Trudy does not come in the traffic forwarding path between Alice and Bob. But when DNS is poisoned by Trudy, all traffic between Alice, and Bob is intercepted by Trudy.



When DNS is secure



When DNS is poisoned by trudy1

Task 1: (10M)

Use the OpenSSL commands to create a root CA certificate (self-signed using 2048-bit RSA), a certificate of Alice (issued i.e., signed by the root CA) and a certificate of Bob (issued by the root CA). Ensure that you provide realistic meta-data while creating certificates like values for CN, OU, L, Country, etc of your choice. Save these certificates as root.crt, alice.crt and bob.crt, save their CSRs and key-pairs in .pem files and verify that they are valid using openssl. You can complete this task either on the VM provided (recommended) or on your local machine.

Creating root CA certificate in VM (remote system)

openssl req -x509 -newkey rsa:2048 -keyout root.pem -out root.crt -days 365
Passphrase - root crt

Creating Alice csr in Alice LXD

Generate private key-
openssl genpkey -out alice-private.pem -algorithm RSA -pkeyopt rsa_keygen_bits:2048
View private key in text form-
openssl pkey -in alice-private.pem -text -noout
Generate public key-
openssl pkey -in alice-private.pem -pubout -out alice-public.pem
View public key in text form-
openssl rsa -noout -text -in alice-public.pem -pubin
Generate CSR-
openssl req -new -key alice-private.pem -out alice.csr
Challenge password - alice
View CSR in text form-
openssl req -text -in alice.csr -noout

Copy csr from alice1 LXD container to vm-
lxc file pull alice1/root/alice.csr .

Sign alice.csr by root CA -
openssl x509 -req -in alice.csr -CA root.crt -CAkey root.pem -CAcreateserial -out alice.crt -days 200 -sha256
View certificate in text format-
openssl x509 -in alice.crt -text -noout

Copy crt from VM to alice1 container-
lxc file push alice.crt alice1/root/

Checking validity of key-pairs, certificates, CSR's-
diff <(openssl x509 -noout -modulus -in alice.crt | openssl md5) <(openssl rsa -noout -modulus -in alice-private.pem | openssl md5) && diff <(openssl rsa -noout -modulus -in alice-private.pem | openssl md5) <(openssl req -noout -modulus -in alice.csr | openssl md5)

We are checking the validity of key-pairs and certificates by comparing their md5 hashes and similarly comparing for key-pairs and CSR's. Thus if there is no difference in output, it means all three are valid and compatible with each other.

```
ns@ns14:~$ lxc exec alice1 bash
root@alice1:~# openssl verify -trusted root.crt alice.crt
alice.crt: OK
root@alice1:~#
```

Hence we can see that the alice.crt is verified against the public key from the root certificate present in the trust store. We now know that the certificate is indeed issued by the root CA and is not tampered with.

Creating Bob csr

Generate private key-
openssl genpkey -out bob-private.pem -algorithm RSA -pkeyopt rsa_keygen_bits:2048
View private key in text form-
openssl pkey -in bob-private.pem -text -noout
Generate public key-
openssl pkey -in bob-private.pem -pubout -out bob-public.pem
View public key in text form-
openssl rsa -noout -text -in bob-public.pem -pubin
Generate CSR-
openssl req -new -key bob-private.pem -out bob.csr
Challenge password - bob123
View CSR in text form-
openssl req -text -in bob.csr -noout

Copy csr from alice1 LXD container to vm-
lxc file pull bob1/root/bob.csr .

Sign alice.csr by root CA-
openssl x509 -req -in bob.csr -CA root.crt -CAkey root.pem -CAcreateserial -out bob.crt -days 200 -sha256
View certificate in text format-
openssl x509 -in bob.crt -text -noout

Copy crt from VM to bob1 container-
lxc file push bob.crt bob1/root/

Checking validity of key-pairs, certificates, CSR's-
diff <(openssl x509 -noout -modulus -in bob.crt | openssl md5) <(openssl rsa -noout -modulus -in bob-private.pem | openssl md5) && diff <(openssl rsa -noout -modulus -in bob-private.pem | openssl md5) <(openssl req -noout -modulus -in bob.csr | openssl md5)

```
ns@ns14:~$ lxc exec bob1 bash
root@bob1:~# openssl verify -trusted root.crt bob.crt
bob.crt: OK
```

Hence we can see that the bob.crt is verified against the public key from the root certificate present in the trust store. We now know that the certificate is indeed issued by the root CA and is not tampered with.

How to communicate among LXDs and between VM and LXD?
You can use the Linux based commands like SCP for transferring the files like CSRs and certs. You have to be sure about the integrity and clearly, show that the files transferred are indeed sent by the intended sender and received by the intended receiver.
For example, if you send the CSR to the Signing Authority, then the signing authority should be able to verify that it is sent by the intended sender and similarly when receiving the certificate back it should be verified that it is indeed signed and sent by the actual authority. You can use signing and verification concepts for this using openssl.

Verification by receiver that the file is sent by actual authority

1) Root CA creates a digest of alice.crt and signs it and sends it to alice along with alice.crt.
   openssl dgst -sha256 -sign root.pem -out alice.crt.sha256 alice.crt
2) Alice verifies that the alice.crt is sent by root CA by comparing the sent digest decrypted by root public key and hash of alice.crt she received.
   openssl dgst -sha256 -verify root-public.pem -signature alice.crt.sha256 alice.crt

Similarly Root CA can verify that the alice.csr was sent by Alice by using alice-public.pem

Task 2: (20M)

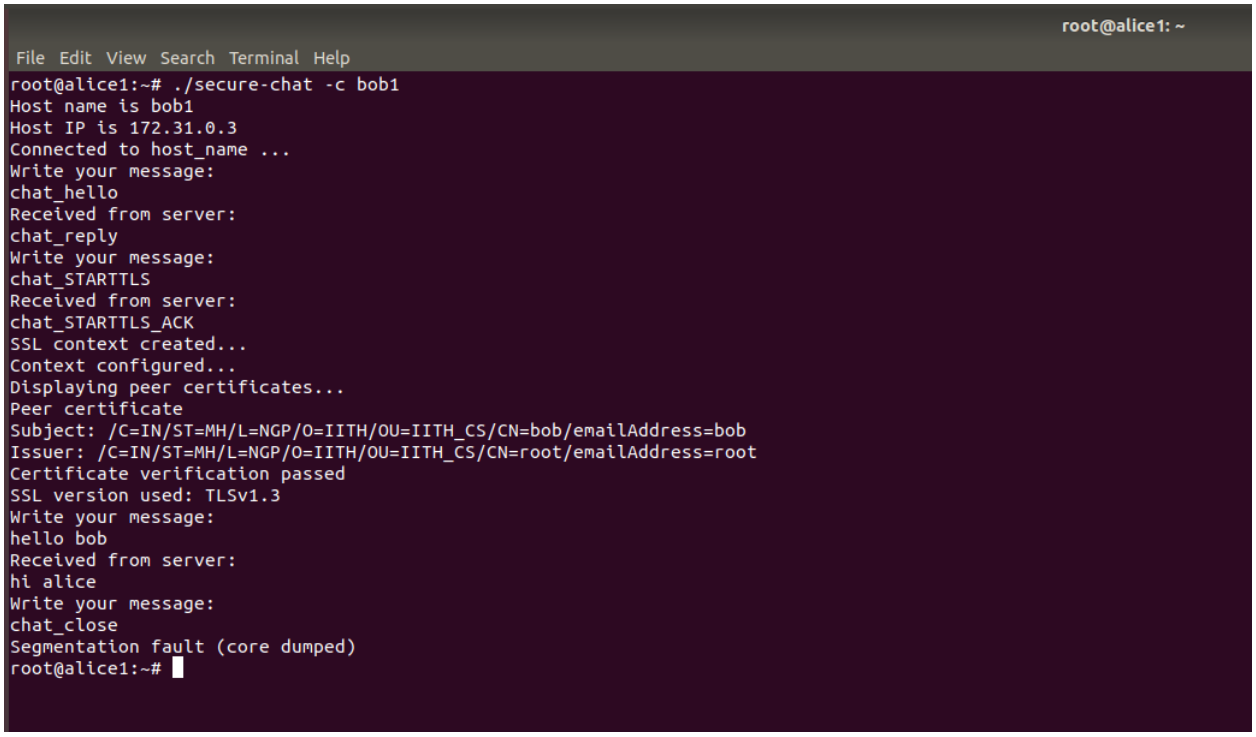
Write a peer-to-peer application (`secure_chat_app`) for chatting which uses TLS and TCP as the underlying protocols for secure and reliable communication. *Note that the* `secure_chat_app` *works like HTTPS except that here it's a peer-to-peer paradigm where Alice plays the role of the client and Bob plays the role of the server (half-duplex communication)*. The same program should have different functions for server and client code which can be chosen using command line options "-s" and "-c <serverhostname>" respectively. Feel free to define your own chat headers (if necessary) and add them to the chat payload before giving it to TLS/TCP. Make sure that the application uses hostnames for communication between Alice and Bob but not hard-coded IP addresses (refer `gethostbyname(3)`). The application should perform the following operations:

a) Establish a TCP connection between Alice and Bob. Bob starts the app using "`secure_chat_app -s`", and Alice starts the app using "`secure_chat_app -c bob1`"
b) Alice sends a *chat_hello* message to Bob and Bob replies with a *chat_reply* message. It works like a handshake at the application layer. Note that these messages are sent in

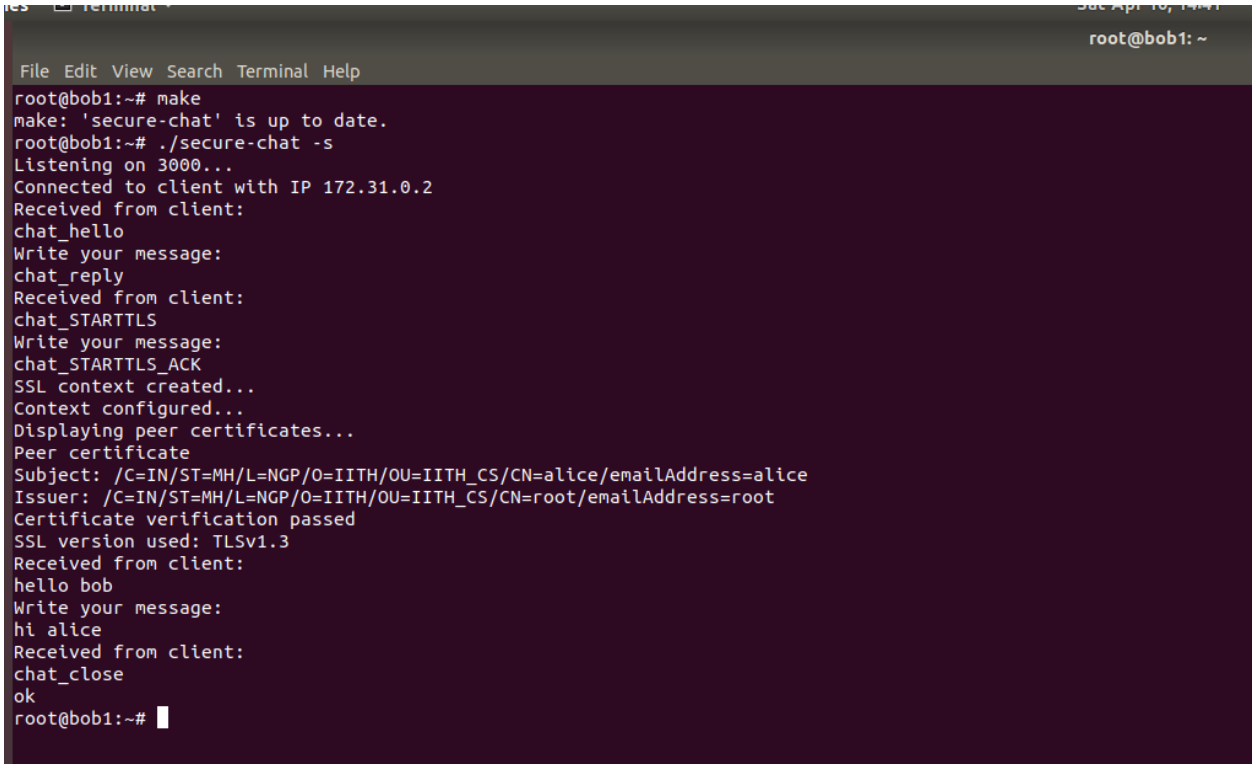
plain text. Show that it is indeed the case by capturing pcap traces at Alice-LXD/Bob-LXD.

c) Alice initiates a secure chat session by sending out *chat_STARTTLS* message and getting *chat_STARTTLS_ACK* from Bob. Your program should load the respective private key and certificate for both Alice and Bob. Furthermore, each of them should have pre-loaded the certificate of the root CA in their trust stores.
    i) For example, if Alice sends a *chat_STARTTLS* message to Bob, upon parsing the message, Bob initiates replies with *chat_STARTTLS_ACK.* Upon parsing this ACK from Bob, Alice initiates **TLS 1.3** handshake by first sending a *client_hello* message as we discussed in TLS lesson.
    ii) Alice gets the certificate of Bob and verifies that. She also provides her certificate to Bob for verification. So, Alice and Bob use their certificates to perform mutual authentication.
d) Upon establishing a secure TLS 1.3 pipe, it is used by Alice and Bob to exchange their encrypted chat messages. Show that it is indeed the case by capturing pcap traces at Alice-LXD/Bob-LXD.
e) Either of them sends a *chat_close* message which triggers closure of TLS connection and finally TCP connection.

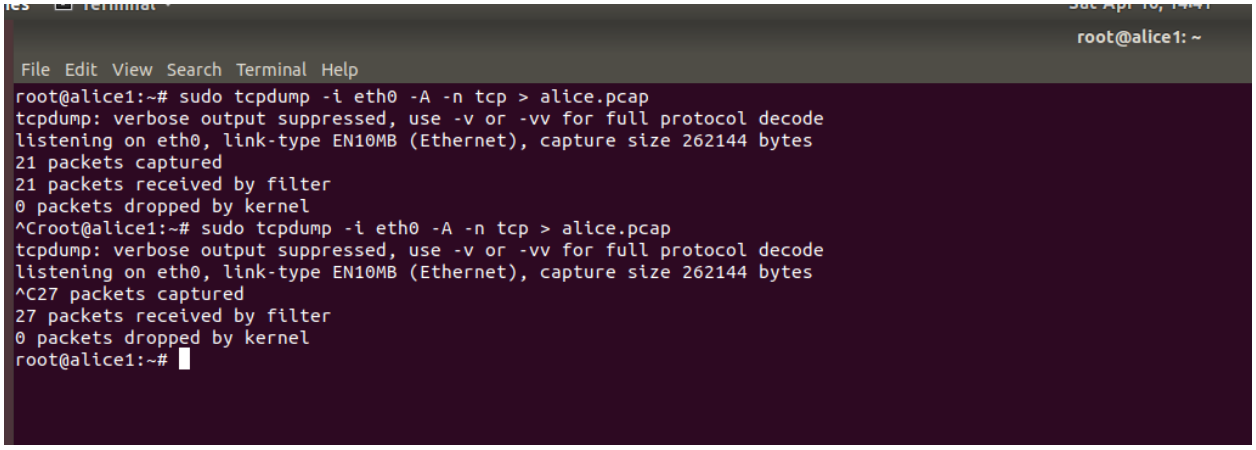
Screenshots



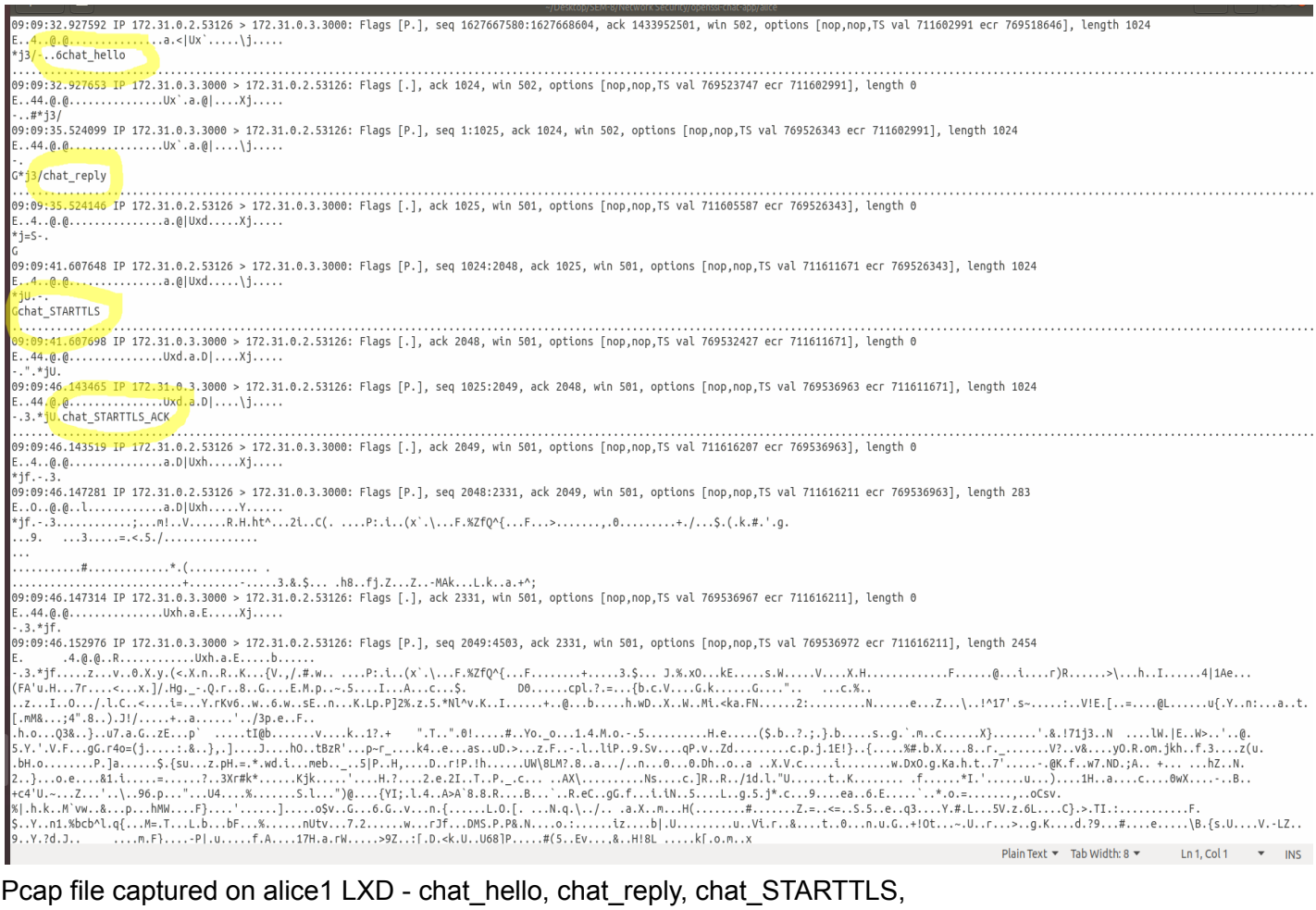
alice1 LXD - TLS1.3 connection is established between Alice and Bob as observed above. Either party verifies the certificate sent by the other party to ensure mutual authentication.



bob1 LXD



To capture tcp dump on alice1 LXD



Pcap file captured on alice1 LXD - chat_hello, chat_reply, chat_STARTTLS, chat_STARTTLS_ACK messages are exchanged in plain text manner as seen above. After establishing TLS1.3 pipe, messages are exchanged in encrypted format as seen above.

Flow of the task :-

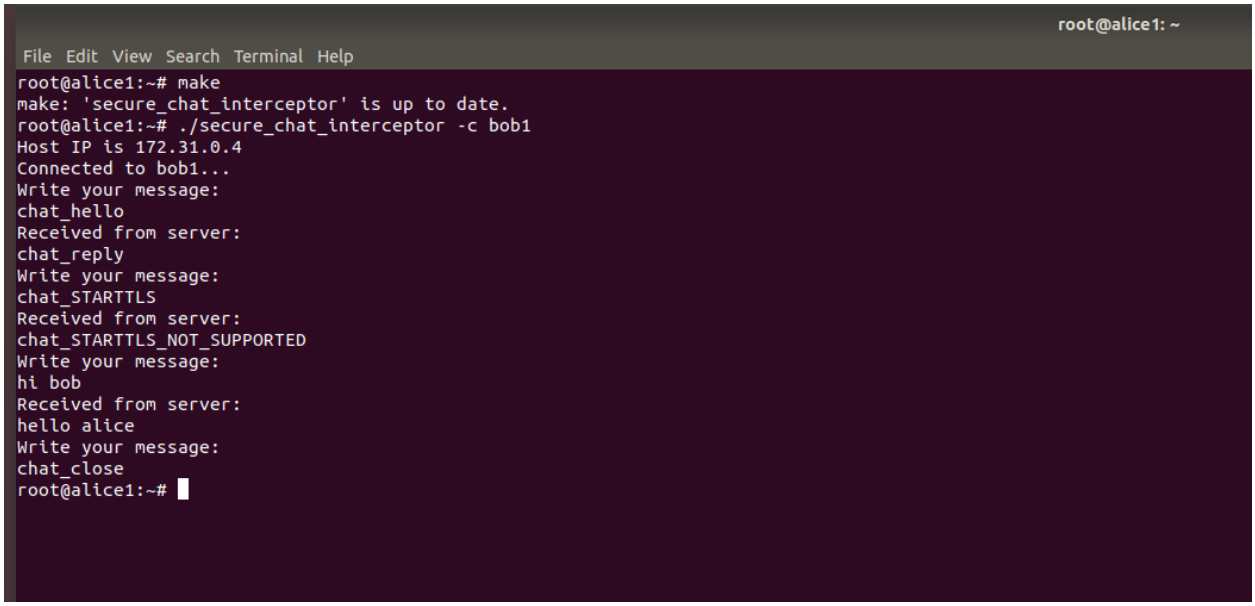
1) Alice establishes a connection with Bob by connecting to his hostname and port.
2) Alice sends a chat_hello message to Bob and Bob replies with a chat_reply message. This works like a handshake at the application layer. These messages are sent in plain text (insecure manner) which are indicated in the screenshot above in pcaps file.
3) Alice sends a message chat_STARTTLS to indicate Bob to start a secure connection using TLS 1.3. Bob acknowledges that by sending chat_STARTTLS_ACK. Now Alice and Bob both are connected in a secure manner using TLS1.3. Both exchange certificates with each other and are verified on either side to ensure mutual authentication.
4) Now onwards all the messages are exchanged in a secure manner between them which can be observed from the screenshots above of the pcaps file.
5) To terminate the chat either party can send a chat_close message which triggers the closure of TLS connection and TCP connection.

Task 3: STARTTLS downgrade attack (20M)

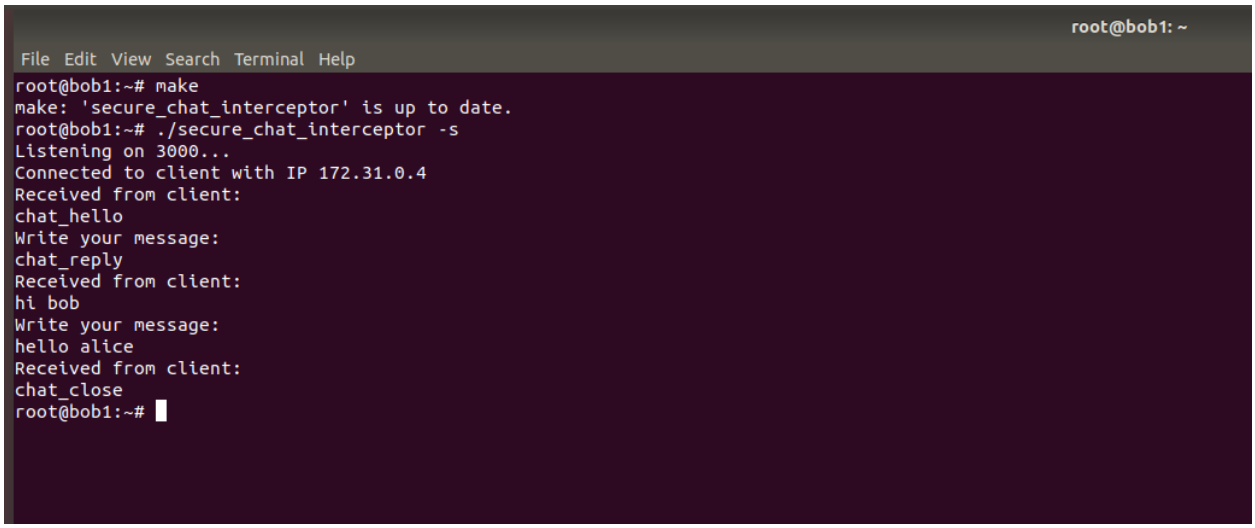
Downgrade attack by Trudy by blocking the chat_STARTTLS message from Alice (Bob) to Bob (Alice).

a) If Alice receives *chat_STARTTLS_NOT_SUPPORTED* message after sending *chat_STARTTLS* to Bob, it is assumed that Bob does not want to have secure chat communication.
b) In this attack, Trudy blocks *chat_STARTTLS* from reaching Bob and sends a fake reply message *chat_STARTTLS_NOT_SUPPORTED* to Alice and thereby forcing Alice and Bob to have unsecure chat communication for successfully intercepting their communication. Show that it is indeed the case by capturing pcap traces at Trudy/Alice/Bob LXDs.
c) You need to write a program (`secure_chat_interceptor`) to launch this downgrade attack (`-d` command line option) from Trudy-LXD. For this task, you can assume that Trudy poisoned /etc/hosts file of Alice (Bob) and replaced the IP address of Bob (Alice) with that of her. It's a kind of DNS spoofing for launching MITM attacks. In this attack, Trudy only plays with *chat_STARTTLS* message(s) and forwards the rest of the traffic as it is.
   To poison /etc/hosts file of Alice, Bob containers, use the following command from inside the VM
   `bash ~/poison-dns-alice1-bob1.sh`
   To revert back the /etc/hosts file of Alice, Bob containers, use the following command from inside the VM.
   `bash ~/unpoison-dns-alice1-bob1.sh`
   To start downgrade attack, start the secure chat interceptor program using the following command from inside the Trudy LXD.
   `./secure_chat_interceptor -d alice1 bob1`

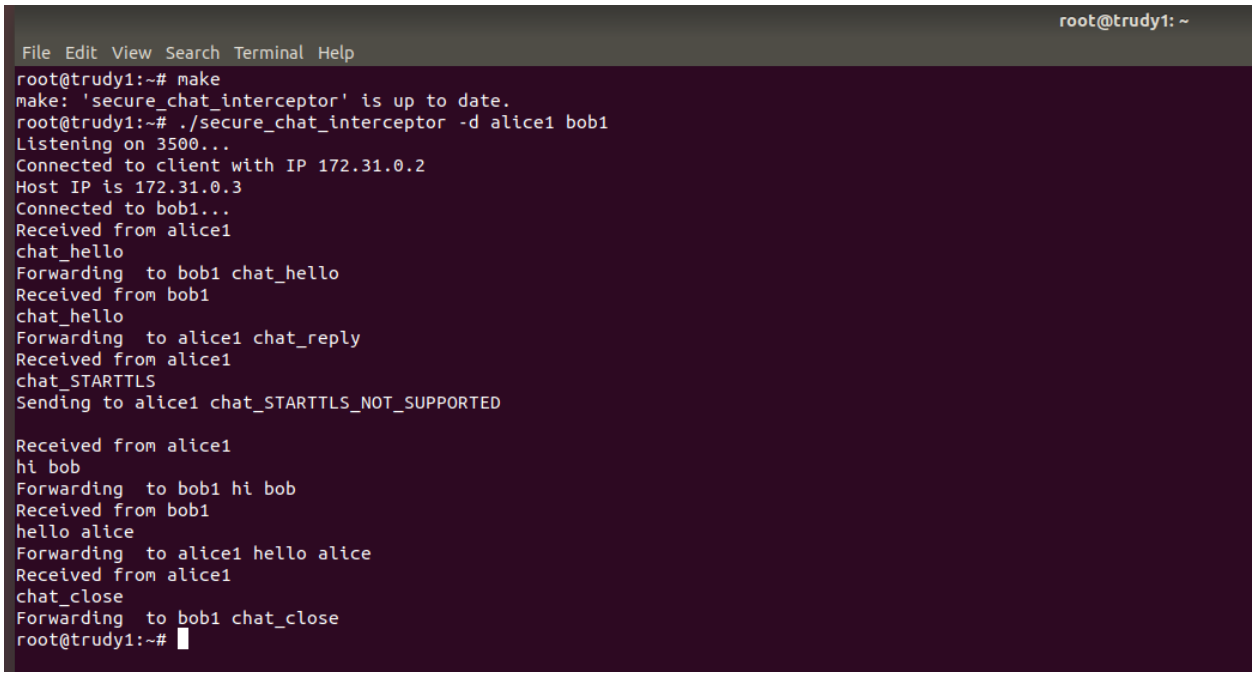
Screenshots



alice1 LXD



bob1 LXD



trudy1 LXD - Trudy plays the role of middle man to initiate downgrade attack. It simply forwards messages from Alice to Bob and from Bob to Alice except when Alice tries to initiate secure connection by sending chat_STARTTLS message then it sends chat_STARTTLS_NOT_SUPPORTED to prevent establishing secure connection.

Pcap file captured on alice LXD - As observed above all the messages exchanged between Alice and Bob are in plain text format.

Flow of the task :-

Before executing the task, /etc/hosts file of Alice, Bob LXD containers are poisoned by Trudy. Trust replaces the IP address of Bob and Alice in Alice and Bob containers respectively by his own IP.

Alice tries to establish a connection with Bob by connecting to his hostname and port. But since /etc/hosts file is poisoned, Alice results in establishing connection with Trudy. Similarly, Trudy establishes connection with Bob impersonating as Alice.

Alice sends a chat_hello message as a part of handshake at application layer, Trudy simply forwards it to Bob.

Bob sends a chat_reply message, Trudy forwards it to Alice. These messages are sent in plain text as indicated in screenshots above of pcap files captured.

Now, Alice sends a chat_STARTTLS message in order to establish a secure TLS connection. Trudy sends chat_STARTTLS_NOT_SUPPORTED message to Alice inorder to launch downgrade attack. Alice by seeing the message assumes that Bob doesn't support TLS or doesn't want to establish a secure connection. Thus Trudy becomes successful in downgrade attack.

Now onwards, Alice and Bob communciates in insecure manner and messages are sent

in plain text. This can be observed in screenshots above.
To terminate the chat either party can send a chat_close message which triggers the closure of TLS connection and TCP connection.

Task 4: (30M)

Active MITM attack by Trudy to tamper the chat communication between Alice and Bob. For this task also, you can assume that Trudy poisoned /etc/hosts file of Alice (Bob) and replaced the IP address of Bob (Alice) with that of her.

a) Also assume that Trudy hacks into the server of root CA and issues fake/shadow certificates for Alice and Bob. Save these fake certificates as fakealice.crt and fakebob.crt, save their CSRs and key-pairs in .pem files and verify that they are indeed valid using openssl.
b) Rather than launching the STARTTLS downgrade attack, in this case Trudy sends the fake certificate of Bob when Alice sends a client_hello message and vice versa. This certificate is indeed signed by the root CA, so it's verification succeeds at Alice. So, two TLS 1.3 pipes are setup: one between Alice and Trudy; the other between Trudy and Bob. Trudy decrypts messages from Alice to Bob (and from Bob to Alice) and re-encrypts them as-it-is or by altering the message content as she desires! Show that it is indeed the case by capturing pcap traces at Trudy/Alice/Bob LXDs.
c) Enhance the secure_chat_interceptor program to launch this active MITM attack from Trudy-LXD.
   To start MITM attack, start the secure chat interceptor program using the following command from inside the Trudy LXD.
   `./secure_chat_interceptor -m alice1 bob1`

In this Task, Trudy creates a private-public key pair and sends two CSRs corresponding to fakealice.crt and fakebob.crt to root CA using this new public key created. Trudy hacks the root CA and issues the certificates corresponding to these two CSRs.
The certificate exchange in the two connections Alice-Trudy and Trudy-Bob is as follows:

- Alice-Trudy: Alice sends alice.crt to Trudy (due to DNS poisoning). Trudy sends fakebob.crt to Alice.
- Trudy-Bob: Trudy sends fakealice.crt to Bob. Bob sends bob.crt to Trudy (due to DNS poisoning).

Flow of the task :-
Before executing the task, /etc/hosts file of Alice, Bob LXD containers are poisoned by Trudy. Trust replaces the IP address of Bob and Alice in Alice and Bob containers respectively by his own IP.

1. Alice tries to establish a connection with Bob by connecting to his hostname and port.
2. But since the /etc/hosts file is poisoned, Alice results in establishing connection with Trudy.

3. Similarly, Trudy establishes connection with Bob impersonating as Alice.
4. Alice sends a chat_hello message as a part of handshake at application layer, Trudy simply forwards it to Bob.
5. Bob sends a chat_reply message, Trudy forwards it to Alice. These messages are sent in plain text as indicated in screenshots of pcap files captured.
6. Alice sends a chat_STARTTLS message. Trudy forwards to Bob. Bob replies with chat_STARTTLS_ACK and Trudy forwards this to Alice.
7. Alice sends alice.crt which is intercepted by Trudy. Trudy sends fakebob.crt to Alice and fakealice.crt to Bob. Bob replies with the bob.crt message which is intercepted by Trudy.
8. Alice, Trudy and Bob verify the received certificates. Alice and Bob create and configure context for their respective connections (with Trudy). Trudy configures context for connections with Alice and Bob.
9. Now onwards, the communication between Alice and Bob is intercepted by Trudy and the messages are encrypted separately in the two connections i.e. Trudy decrypts messages received from Alice(Bob), encrypts this message and sends it to Bob(Alice) and vice versa (messages from Bob to Alice). This can be observed in screenshots.
10. To terminate the chat either party can send a chat_close message which triggers the closure of TLS connection and TCP connection.

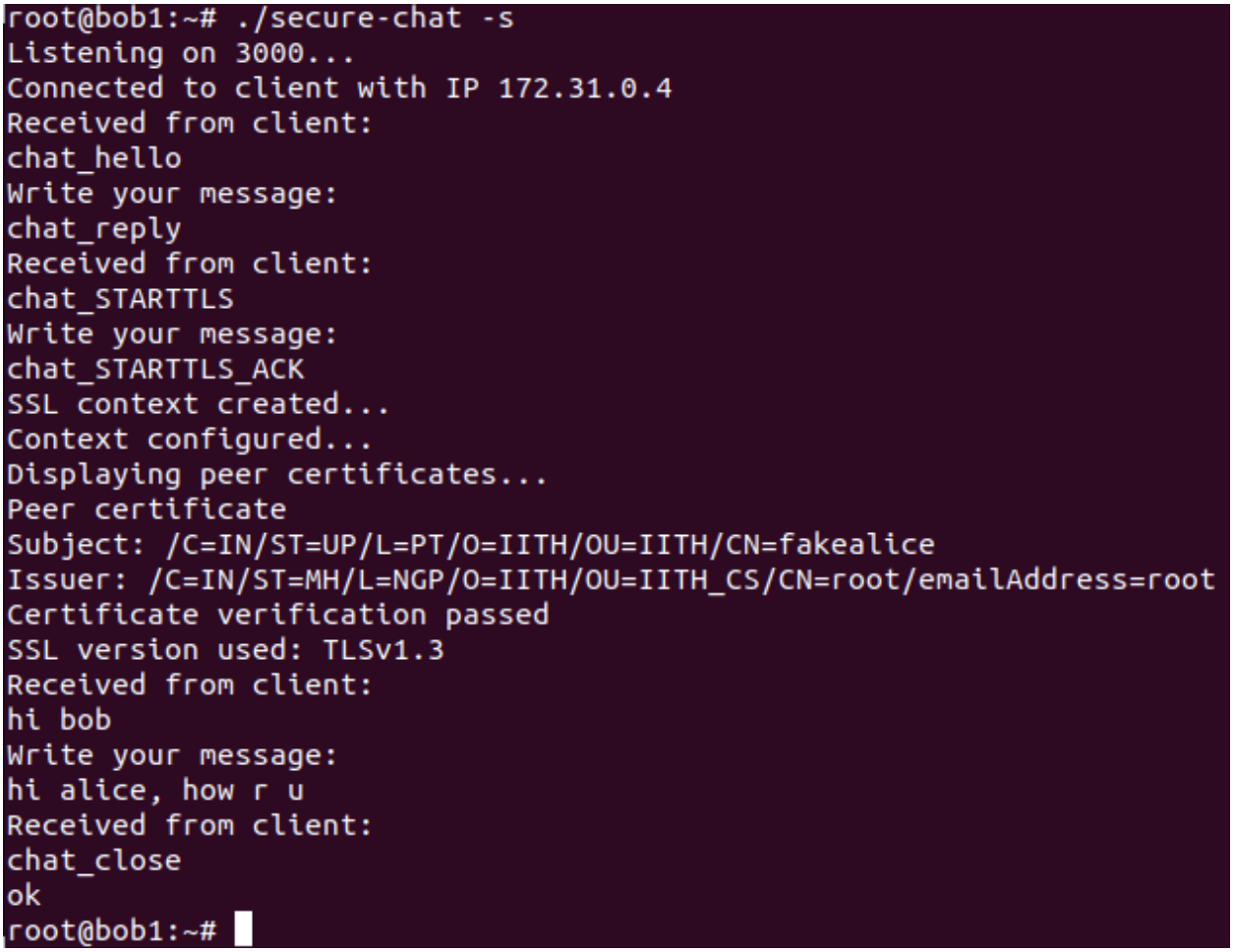
Screenshots

```
root@trudy1:~# ./secure_chat_interceptor_en -m alice1 bob1
Listening on 3500...
Connected to client with IP 172.31.0.2
Host IP is 172.31.0.3
Connected to bob1...
Received from alice1
chat_hello
Forwarding  to bob1 chat_hello
Received from bob1
chat_hello
Forwarding  to alice1 chat_reply
Received from alice1
chat_STARTTLS
Forwarding  to bob1 chat_STARTTLS
Received from bob1
chat_STARTTLS
Forwarding  to alice1 chat_STARTTLS_ACK
Alice SSL context created...
Enter PEM pass phrase:
Alice Context configured...
Bob SSL context created...
Enter PEM pass phrase:
Bob Context configured...
Displaying alice peer certificates...
Peer certificate
Subject: /C=IN/ST=MH/L=NGP/O=IITH/OU=IITH_CS/CN=alice/emailAddress=alice
Issuer: /C=IN/ST=MH/L=NGP/O=IITH/OU=IITH_CS/CN=root/emailAddress=root
Displaying bob peer certificates...
Peer certificate
Subject: /C=IN/ST=MH/L=NGP/O=IITH/OU=IITH_CS/CN=bob/emailAddress=bob
Issuer: /C=IN/ST=MH/L=NGP/O=IITH/OU=IITH_CS/CN=root/emailAddress=root
Alice Certificate verification passed
Bob Certificate verification passed
Alice SSL version used: TLSv1.3
Bob SSL version used: TLSv1.3
Received from alice1
hi bob
Forwarding  to bob1 hi bob
Received from bob1
hi alice, how r u
Forwarding  to alice1 hi alice, how r u
Received from alice1
chat_close
Forwarding  to bob1 chat_close
```

Trudy

```
root@alice1:~# ./secure_chat_app -c bob1
Host name is bob1
Host IP is 172.31.0.4
Connected to host_name ...
Write your message:
chat_hello
Received from server:
chat_reply
Write your message:
chat_STARTTLS
Received from server:
chat_STARTTLS_ACK
SSL context created...
Context configured...
Displaying peer certificates...
Peer certificate
Subject: /C=IN/ST=UP/L=PT/O=IITH/OU=IITH/CN=fakebob
Issuer: /C=IN/ST=MH/L=NGP/O=IITH/OU=IITH_CS/CN=root/emailAddress=
root
Certificate verification passed
SSL version used: TLSv1.3
Write your message:
hi bob
Received from server:
hi alice, how r u
Write your message:
```

alice


root@bob1:~# ./secure-chat -s
Listening on 3000...
Connected to client with IP 172.31.0.4
Received from client:
chat_hello
Write your message:
chat_reply
Received from client:
chat_STARTTLS
Write your message:
chat_STARTTLS_ACK
SSL context created...
Context configured...
Displaying peer certificates...
Peer certificate
Subject: /C=IN/ST=UP/L=PT/O=IITH/OU=IITH/CN=fakealice
Issuer: /C=IN/ST=MH/L=NGP/O=IITH/OU=IITH_CS/CN=root/emailAddress=root
Certificate verification passed
SSL version used: TLSv1.3
Received from client:
hi bob
Write your message:
hi alice, how r u
Received from client:
chat_close
ok
root@bob1:~#

bob

```
E..4G.@.@..............q.OZ-..&....\k.....
e..{..H.chat_hello
................................................................................
16:33:09.409391 IP 172.31.0.4.3500 > 172.31.0.2.52502: Flags [.], ack 1025, win 502,
options [nop,nop,TS val 3905605004 ecr 1705572219], length 0
E..4aR@.@..-............-..&q.SZ....Xk.....
....e..{
16:33:39.642229 IP 172.31.0.4.3500 > 172.31.0.2.52502: Flags [P.], seq 1:1025, ack
1025, win 502, options [nop,nop,TS val 3905635236 ecr 1705572219], length 1024
E..4aS@.@.},............-..&q.SZ....\k.....
..C.e..{chat_reply
................................................................................
16:33:39.642489 IP 172.31.0.2.52502 > 172.31.0.4.3500: Flags [.], ack 1025, win 501,
options [nop,nop,TS val 1705602452 ecr 3905635236], length 0
E..4G.@.@..............q.SZ-..&....Xk.....
e.m...C.
16:33:59.166241 IP 172.31.0.2.52502 > 172.31.0.4.3500: Flags [P.], seq 1025:2049, ack
1025, win 501, options [nop,nop,TS val 1705621975 ecr 3905635236], length 1024
E..4G.@.@..............q.SZ-..&....\k.....
e.....C.chat_STARTTLS
................................................................................
16:33:59.166330 IP 172.31.0.4.3500 > 172.31.0.2.52502: Flags [.], ack 2049, win 501,
options [nop,nop,TS val 3905654761 ecr 1705621975], length 0
E..4aT@.@..+............-..&q.WZ....Xk.....
....e...
16:34:16.351238 IP 172.31.0.4.3500 > 172.31.0.2.52502: Flags [P.], seq 1025:2049, ack
2049, win 501, options [nop,nop,TS val 3905671945 ecr 1705621975], length 1024
E..4aU@.@.}*............-..&q.WZ....\k.....
...     e...chat_STARTTLS_ACK
................................................................................
16:34:16.351298 IP 172.31.0.2.52502 > 172.31.0.4.3500: Flags [.], ack 2049, win 501,
options [nop,nop,TS val 1705639161 ecr 3905671945], length 0
E..4H.@.@..............q.WZ-..&....Xk.....
e......
16:34:16.356350 IP 172.31.0.2.52502 > 172.31.0.4.3500: Flags [P.], seq 2049:2332, ack
2049, win 501, options [nop,nop,TS val 1705639166 ecr 3905671945], length 283
E..OH.@.@..c............q.WZ-..&....Y......
e..... ..........r..@.m=).......i).M.;...
$...`... ...qNk..o..M..E.&......I44.%...p.>.......,.0.........+./...$.(.k.#.'.g.
...9.   ...3.....=.<.5./...............
...
...........#............*.(........... .
```

Alice1.pcap

References:

1. [OpenSSL Cookbook: Chapter 1. OpenSSL Command Line (feistyduck.com)]
2. [/docs/man1.1.1/man3/index.html (openssl.org)]
3. [OpenSSL client and server from scratch, part 1 – Arthur O'Dwyer – Stuff mostly about C++ (quu xplusone.github.io)]

4. [ssl — TLS/SSL wrapper for socket objects — Python 3.9.2 documentation]
5. [Secure programming with the OpenSSL API – IBM Developer]
6. [Simple TLS Server - OpenSSLWiki]
7. [The /etc/hosts file (tldp.org)]

Appendix: Some help regarding the setup.

- To login to the VM use **`ssh ns@192.168.51.xw`** command.
- To list containers use **`lxc ls`** command from inside the VM.
- To login to a container, use **`lxc exec <containername> bash`** command from inside the VM.
- To logout from the respective container use **`exit`** command.
- To capture packet traces inside a container (eg. trudy1), use **`lxc exec trudy1 -- sudo tcpdump -i eth1 -nn not tcp port 22`**
- Root file-system of a container (eg alice1) is located as a subtree under its storage pool **`/var/snap/lxd/common/lxd/storage-pools/default/containers/alice1/rootfs`**
- To copy files from local system to remote system:
  `scp <filename> root@<destination-ip>:~/`
  To copy files from remote system to local system
  `scp root@<source-ip>:~/<filename> .`

```
Install the OpenSSL library, for the ubuntu use the below command.

sudo apt-get install libssl-dev

g++ -Wall -o secure-chat secure_chat_app.cpp -L/usr/lib -lssl -lcrypto

scp secure_chat_app.cpp  ns@192.168.51.124:/home/ns

lxc file push secure_chat_app.cpp alice1/root/

sudo tcpdump -i eth0 -A -n tcp > alice.pcap
scp ns@192.168.51.124:/home/ns/alice.pcap .
g++ -Wall -o secure-chat-interceptor secure_chat_interceptor.cpp -L/usr/lib
-lssl -lcrypto

scp secure_chat_interceptor.cpp  ns@192.168.51.124:/home/ns
```

```
lxc file push secure_chat_interceptor.cpp alice1/root/
```

```
netstat -a | grep <portno>
```

PLAGIARISM STATEMENT

I certify that this assignment/report is our own work, based on our combined personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, tutorials, sample programs, and any other kind of document, electronic or personal communication. We 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 we have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. We pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, we understand our responsibility to report honour violations by other students if any of us become aware of it.

Names: Ajinkya Bokade
Date: 01/04/2021
Signature: AB

I certify that this assignment/report is our own work, based on our combined personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, tutorials, sample programs, and any other kind of document, electronic or personal communication. We 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 we have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. We pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, we understand our responsibility to report honour violations by other students if any of us become aware of it.

Names: Niraj Kamble
Date: 01/04/2021
Signature: NK

I certify that this assignment/report is our own work, based on our combined personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, tutorials, sample programs, and any other kind of document, electronic or personal communication. We 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 we have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. We pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, we understand our responsibility to report honour violations by other students if any of us become aware of it.

Names: M Sai Subodh
Date: 01/04/2021
Signature: MSS