Report on

Secure Chat Using OpenSSL and
MITM attacks

Course

(CS6903) Network Security

Date

07/04/2022

Submitted by

Group No. 3

Nilesh Shivanand Kale CS21MTECH11022
Ayan Kumar Pahari CS21MTECH14003
Harinder Kaur CS21RESCH11008

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Index

I. Introduction

OpenSSL is a software library for applications that secure communications over computer networks against eavesdropping or need to identify the party at the other end. It is widely used by Internet servers, including the majority of HTTPS websites. It offers a wide range of cryptographic functionalities including, encryption/decryption, authentication, computing hash digests, verifications etc. In this project we will try to simulate the real time working of secure communication using TLS/SSL protocols over the standard TCP sockets through the socket programming, also it simulates the series of attacks which include the downgrade attack and the MITM (Man in the middle attack). The future sections talk about the actual program design, and the process flow for the scenarios.

II. Experiment Simulation

The program scenarios are now executed on the linux vm (LXC) containers, Each container represents a particular entity from among Alice, Bob and Trudy. All the respective files and certificates are maintained in their respective root directory structures.

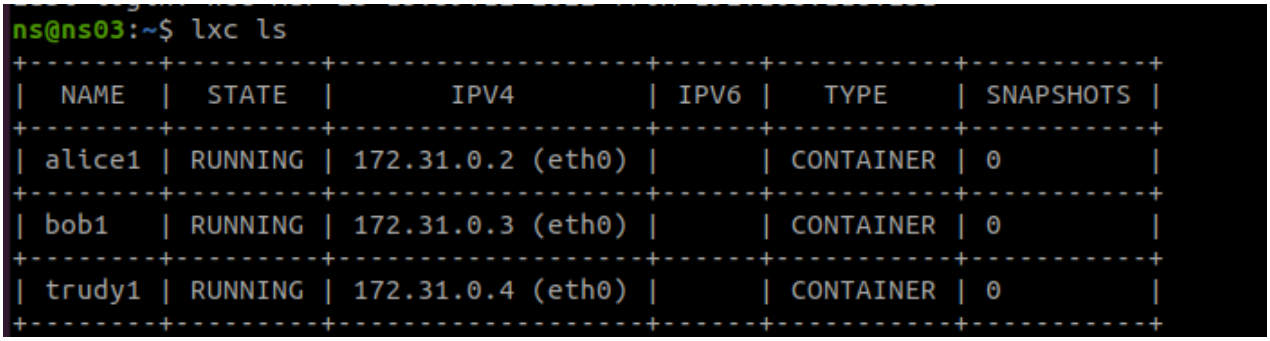


Figure 1. Screenshot of lxc containers running at VM machine

II.I Task 1

Role / Entity	Played by
Alice	Nilesh
Bob	Ayan
Trudy (Root CA) For Task 1	Harinder

TASK 1 : Creating Public-private key pairs and generation of Certificates (Refer Appendix A for key details and certificates)

1.1 Root created self signed certificate as follows:
Local machine commands (192.168.118.231-IITH LAN) :-

1. Command to create Elliptic Curve key with ES512 signatures:-
   openssl ecparam -name brainpoolP512t1 -genkey -noout -out private.ec.pem
2. Now input this Elliptic Curve key to CSR request with following command:-
   openssl req -key private.ec.pem -new -x509 -days 365 -out root.crt

Elliptic Curve cryptography is based on the algebraic structure rather than finite fields. Advantage of using EC keys is that they provide more security than finite keys with smaller length of keys.

1.2 Explanation of commands (Openssl Foundation, n.d.) :-

- **Ecparam:** This command is used to generate or manipulate EC parameter files.
  - **Name arg**: This argument is used to specify the list of currently implemented Elliptic Curve functions. For generating Elliptic Curve key, brainpoolIP512t1 standard elliptic curve database is used (ttps://neuromancer.sk/std/brainpool/brainpoolP512t1).
  - **Genkey**: This option will generate EC private keys based on the specified function/parameters.
  - **Noout**: This option will inhibit the command to output the encoded part of the command. So, it is always safe to turn on no command.
  - **Out:** This field is used to specify the output file for writing keys.
- **Openssl EC**: This command is used to process the Elliptic curve keys. Major use of this command is to convert EC keys between various forms or to print a certain part of the EC key. We used it for getting the public part of EC key generated so that it can be fed into the CSR request.
  - **In**: filed used to specify the EC curve as input.
  - **Pubout**: public key part to be printed out.
  - Root_pub_key.pem now contains the public EC key of the root user.

```
ns@ns03:~/root$ openssl ecparam -name brainpoolP512t1 -genkey -noout -out private.ec.pem
ns@ns03:~/root$ openssl req -key private.ec.pem -new -x509 -days 365 -out root.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:TS
Locality Name (eg, city) []:Hyderabad
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Real CA
Organizational Unit Name (eg, section) []:Real CA
Common Name (e.g. server FQDN or YOUR name) []:Real CA
Email Address []:realca@gmail.com
ns@ns03:~/root$
```

Figure 1. Creation of Elliptic Curve Key pair by root

```
ns@ns03:~/root$ openssl ec -in private.ec.pem -pubout > root_pub_key.pem
read EC key
writing EC key
ns@ns03:~/root$
```

Figure 2. Extracting Public key from Elliptic Curve pair

1.3 ALICE AND BOB KEY GENERATION AND CSR REQUEST

Commands executed on lxc container alice1 (IP: 172.31.0.2).

- Command for RSA 2048 bit key generation and CSR request is as follows:
  Openssl req -newkey rsa:2048 -nodes -keyout alice.key -days 365 -out alice.csr

Similarly, Bob created CSR request by running common don lxc VM (IP: 172.31.0.3)

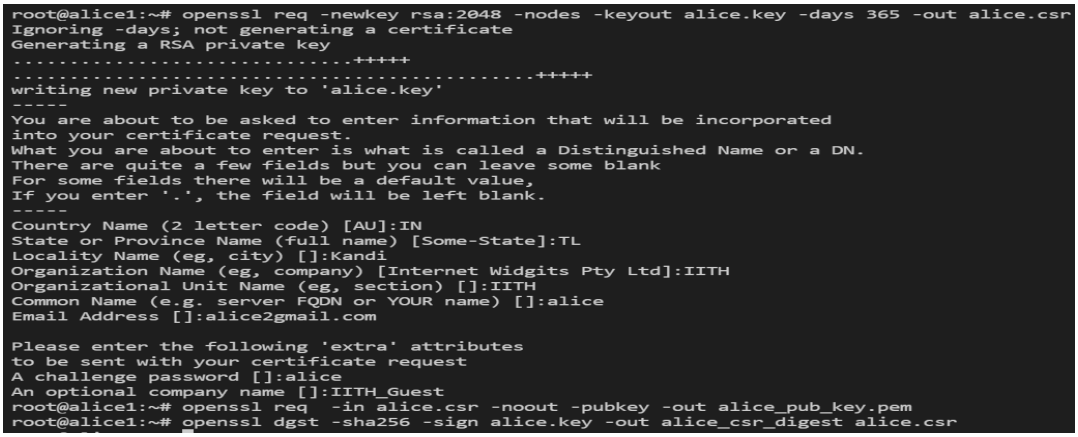


Figure 3. Alice CSR generation

```
root@bob1:~# openssl req -newkey rsa:2048 -nodes -keyout bob.key -days 365 -out bob.csr
Ignoring -days; not generating a certificate
Generating a RSA private key
.....................................................+++++
.....................................................................................
.....+++++
writing new private key to 'bob.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:T1
Locality Name (eg, city) []:Sangareddy
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IITH
Organizational Unit Name (eg, section) []:IITH
Common Name (e.g. server FQDN or YOUR name) []:bob
Email Address []:bob@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:alice
An optional company name []:IITH_Guest
root@bob1:~# openssl  -req -in bob.csr -noout -pubkey -out bob_pub_key.pem
Invalid command '-req'; type "help" for a list.
root@bob1:~# openssl  req -in bob.csr -noout -pubkey -out bob_pub_key.pem
root@bob1:~# openssl dgst -sha256 -sign bob.key -out bob_csr_digest bob.csr
```

Figure 4. Bob CSR generation

Alice and Bob send their CSR, respective digest and public key to root for signing

```
ns@ns03:~/root$ lxc file pull alice1/root/alice.csr ./alice.csr
ns@ns03:~/root$ lxc file pull alice1/root/alice_pub_key.pem ./alice_pub_key.pem
ns@ns03:~/root$ lxc file pull alice1/root/alice_csr_digest ./alice_csr_digest
ns@ns03:~/root$ lxc file pull bob1/root/bob.csr ./bob.csr
ns@ns03:~/root$ lxc file pull bob1/root/bob_pub_key.pem ./bob_pub_key.pem
ns@ns03:~/root$ lxc file pull bob1/root/bob_csr_digest ./bob_csr_digest
```

Figure 5. Pulling CSR and digest from lxc containers

1.5 Root CA: Integrity Check and Certificate Generation

CA verifies weather it is signed by right sender or not by verifying Bob and Alice CSR file digest with the following command:-

Openssl dgst -sha256 -verify alice_pub_key.pem -signature alice_csr_digest alice.csr

Openssl dgst -sha256 -verify bob_pub_key.pem -signature bob_csr_digest bob.csr

```
ns@ns03:~/root$ openssl dgst -sha256 -verify alice_pub_key.pem -signature alice_csr_digest alice.csr
Verified OK
ns@ns03:~/root$ openssl dgst -sha256 -verify bob_pub_key.pem -signature bob_csr_digest bob.csr
Verified OK
```

Figure 6. Verification of digest

CA signs Alice's and Bob's CSR with the following command:-

Openssl x509 -req -CA root.crt -days 365 -in alice.csr -CAkey private.ec.pem -CAcreateserial -o ut alice.crt

```
ns@ns03:~/root$ openssl x509 -req -CA root.crt -days 365 -in alice.csr -CAkey private.ec.pem -CAcreateserial -o
ut alice.crt
Signature ok
subject=C = IN, ST = TL, L = Kandi, O = IITH, OU = IITH, CN = alice, emailAddress = alice2gmail.com
Getting CA Private Key
ns@ns03:~/root$ openssl x509 -req -CA root.crt -days 365 -in bob.csr -CAkey private.ec.pem -CAcreateserial -out
 bob.crt
Signature ok
subject=C = IN, ST = T1, L = Sangareddy, O = IITH, OU = IITH, CN = bob, emailAddress = bob@gmail.com
Getting CA Private Key
```

Figure 7. Generation of certificates for Bob and Alice

CA calculate digest of alice.crt and bob.crt and signs it using its own private key with the following command:-

Openssl dgst -sha256 -sign private.ec.pem -out alice_crt_digets alice.crt

```
ns@ns03:~/root$ openssl dgst -sha256 -sign private.ec.pem -out alice_crt_digest alice.crt
ns@ns03:~/root$ openssl dgst -sha256 -sign private.ec.pem -out bob_crt_digest bob.crt
```

Figure 8. Creating digest of CRT

CA sends the created certificate, its message digest and own's public key to Alice and Bob.

```
ns@ns03:~/root$ lxc file  push ./root_pub_key.pem alice1/root/root-pub_key.pem
ns@ns03:~/root$ lxc file  push ./alice.crt alice1/root/alice.crt
ns@ns03:~/root$ lxc file  push ./alice_crt_digest alice1/root/alice_crt_digest
ns@ns03:~/root$ lxc file  push ./root_pub_key.pem bob1/root/root-pub_key.pem
ns@ns03:~/root$ lxc file  push ./bob.crt bob1/root/bob.crt
ns@ns03:~/root$ lxc file  push ./bob_crt_digest bob1/root/bob_crt_digest
```

Figure 9. Sending signed digest, certificate and public key

Alice and Bob verify that the certificate is signed by the right CA or not.

```
root@alice1:~# openssl dgst -sha256 -verify root_pub_key.pem -signature alice_crt_digest alice.crt
Verified OK
```

```
root@bob1:~# openssl dgst -sha256 -verify root_pub_key.pem -signature bob_crt_digest bob.crt
Verified OK
```

Figure 10. Digest and signature verification at client Side

Note: Root CA certificate is of ECC 512 bits type and Alice and Bob's certificate is of RSA 2048 bits type.

All the files are transferred to the root file system of the virtual machine using **scp** and pushed in an individual container using **push** command.

II.II TASK 2

The following illustration depicts the message flow for the TLS communication message flow between the client and the server over the established TCP socket connection.

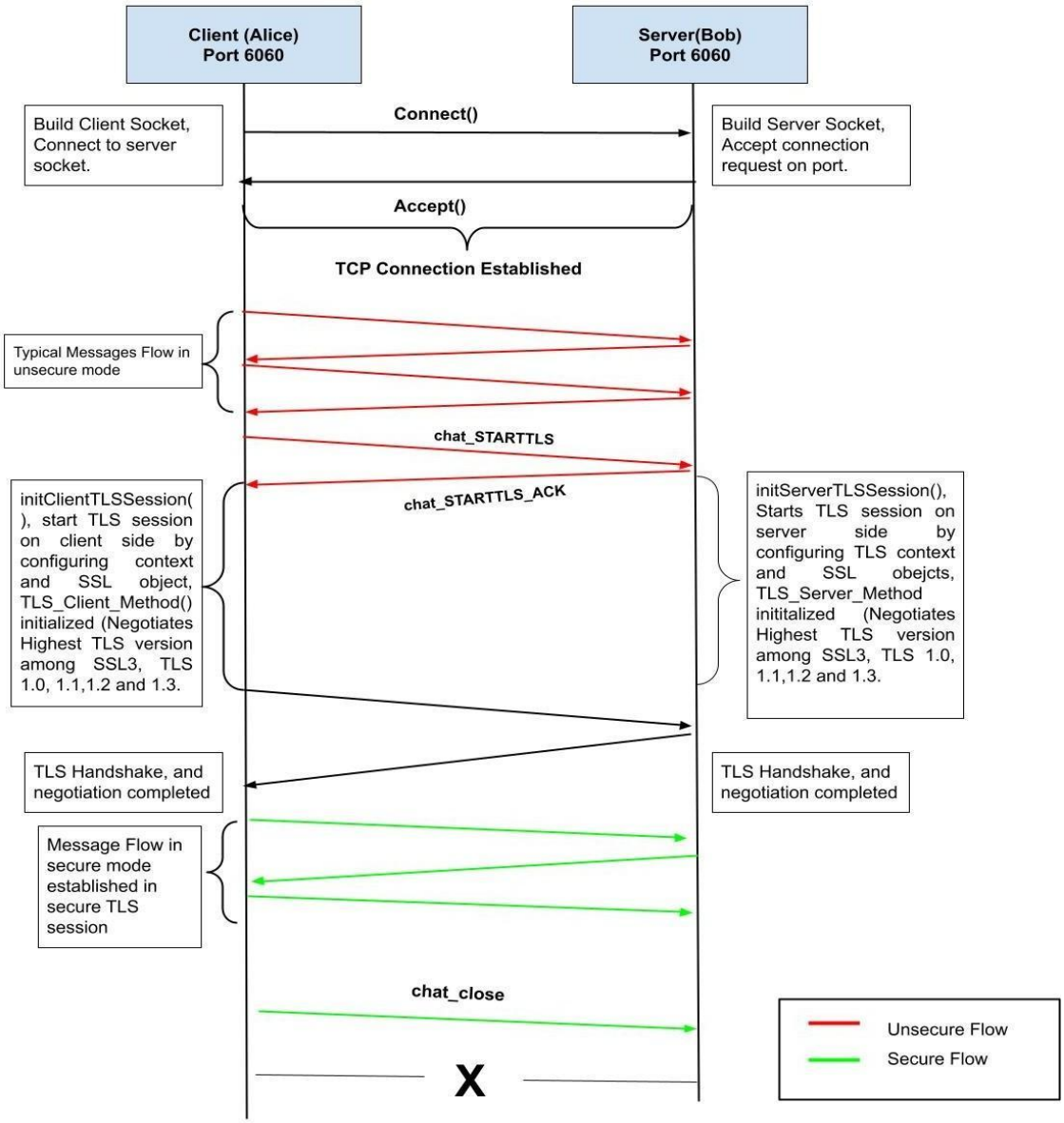


Figure 11. TLS communication flow between Alice and Bob

Commands used

Server (Bob) execution:

```
Python3 secure_chat_app.py -s
```

Client (Alice) execution:

```
Python3 secure_chat_app.py -c bob1
```

Started dumping for packets capture in both Alice and Bob's bash

```
ns@ns03:~$ lxc shell alice1
root@alice1:~# tcpdump -w alice2.pcpng
tcpdump: listening on eth0, link-type EN10MB (Ethernet),
capture size 262144 bytes
^C47 packets captured
47 packets received by filter
0 packets dropped by kernel
```

```
ns@ns03:~$ lxc shell  bob1
root@bob1:~# tcpdump -w bob2.pcpng
tcpdump: listening on eth0, link-type EN10MB (Ethernet),
capture size 262144 bytes
^C47 packets captured
47 packets received by filter
0 packets dropped by kernel
```

Figure 12. TCPDUMP instance running at Alice and Bob side

Started a chat application. Initially communicated without TLS and then started chatting with TLS as follows.

```
ns@ns03:~$ lxc shell alice1
root@alice1:~# python3 secure_chat_app.py -c bob1
I am client
Connected to server

****************************************
You are chatting in insecure  manner
****************************************

Client : Hello, I am client
Server: Yes Alice.  Go ahead
Client : I need to share my pin in secure fashion.
Server: Yes, start TLS first.
Client : chat_STARTTLS
Server: chat_STARTTLS_ACK

****************************************
TLS is enabled at server side. Now chat will be encrypted
****************************************

Client : my pin is 472973
Server: I  got your pin.
Client : chat_close
root@alice1:~#
```

```
ns@ns03:~$ lxc shell bob1
root@bob1:~# python3 secure_chat_app.py -s
I am server
Waiting for incoming connections.
Received connection from  172.31.0.2 ( 60230 )

****************************************
You are chatting in insecure  manner
****************************************

Client: Hello, I am client
Server : Yes Alice.  Go ahead
Client: I need to share my pin in secure fashion.
Server : Yes, start TLS first.
Client: chat_STARTTLS
Sending Acknowledgement

****************************************
TLS is enabled at server side. Now chat will be encrypted
****************************************

Client: my pin is 472973
Server : I  got your pin.
Client: chat_close
Closing connection
root@bob1:~#
```

Figure 13. TLS connection running at Alice and Bob side

After following the TCP stream for the captured packet, we can see messages are flowing in plain text before starting TLS. Once TLS starts, messages are in encrypted form.

Figure 14. Encrypted communication between Alice and Bob (over TLS connection)

II.III Task 3

The following illustration describes the process/ message flow for the Task 3 where the attacker performs the downgrade attack through a well-defined and channelized attacking strategy between the client and the server.

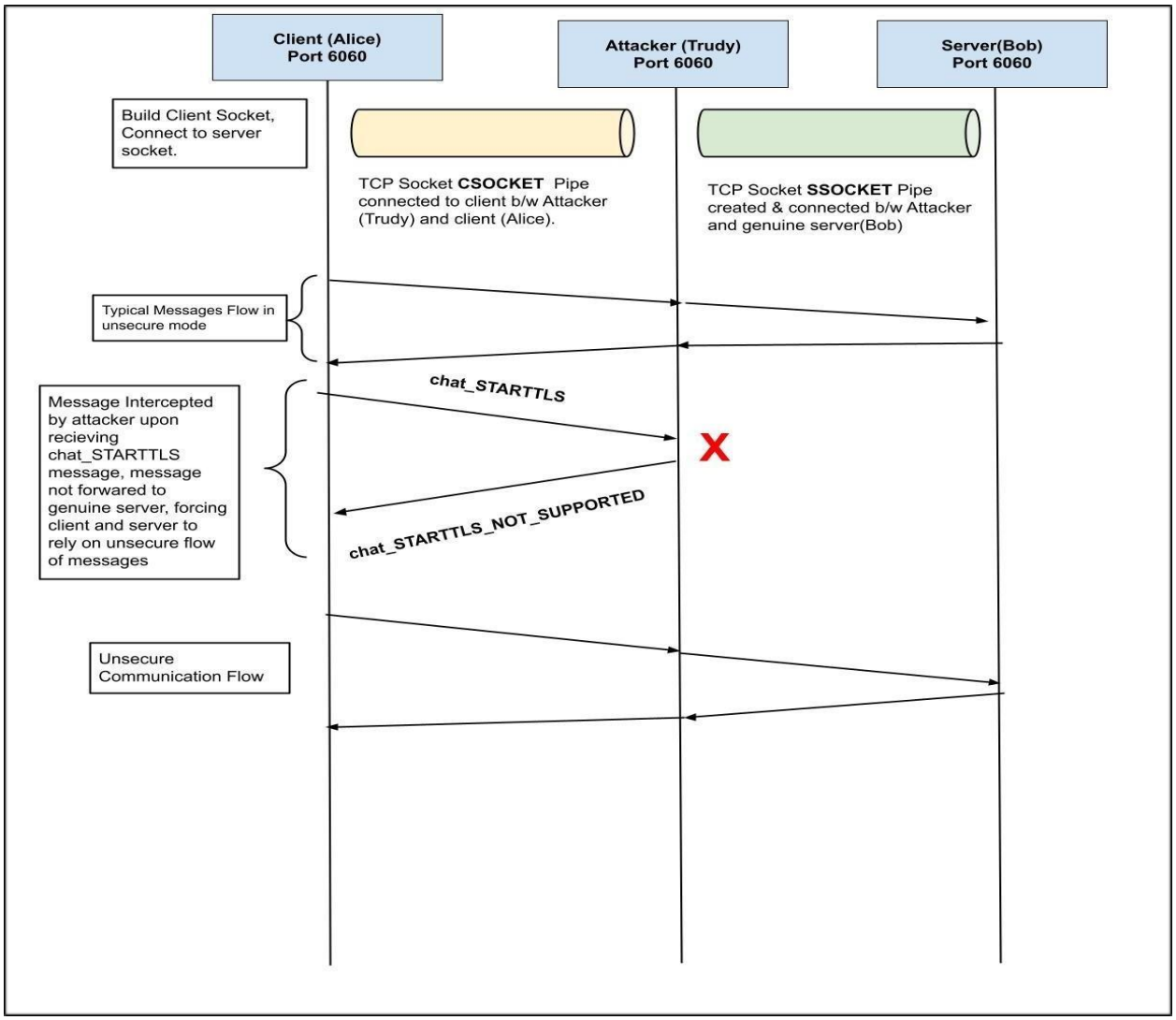


Figure 15. Communication Flow for downgrade attack

Algorithm for TLS_DOWNGRADE_ATTACK

Downgrade_Attack_TLSConnection (VictimIP, ServerIP)

1. VictimClientServer: Create Socket and bind it with victim IP and default port.
2. Create Socket and bind it with copying the real server (Bob)' IP and port.
3. pktReceiveFLag:=True
4. Repeat while pktReceiveFlag==TRUE
   FLag_VictimeServerReply:=1
   receiveMessage:=MessageFromRealClient
   if (receiveMessage == "chat_STARTTLS_CONNECTION")
   FakeMessage:= "STARTTLS_not_supported"

```
        FLag_VictimeServerReply:=0
        send(FakeMessage, VictimSocket)
else
    if (receiveMessage == ‘’chat_close”)
        Forward Message (VictimSocket)
else
     Encode (receiveMessage)
     Forward(receiveMessage,victimSocket)
if (Flag_VictimServerReply== TRUE)
   Get reply from victim server
   Decode (Server Message)
   Send encoded message to Victim
```

Steps to run the setup for Task3:

1. Start bash windows for Alice, Bob and Trudy.
2. Run interceptor file in Trudy using command
   Python3 secure_chat_interceptor.py -d alice1 bob1
3. Start server in Bob using command
   Python3 secure_chat_app.py -s
4. Start client in Alice bash using command
   Python3 secure_chat_app.py -c bob1

Poisoning The DNS to make Trudy settle in the middle of the communication, by changing the IP addresses of Alice and Bob to herself. Trudy redirects the communication through her.

Before DNS poisoning, the IP addresses of Alice and Bob are correct.
After DNS poisoning, the IP address for Alice and Bob is poisoned to Trudy’s one.

```
root@alice1:~# nslookup bob1
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   bob1
Address: 172.31.0.4

root@alice1:~#
```

```
root@bob1:~# nslookup alice1
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   alice1
Address: 172.31.0.4

root@bob1:~#
```

Snapshot for our experiment is:

```
root@alice1:~# python3 secure_chat_app.py -c b
ob1
I am client
Connected to server

***************************************
You are chatting in insecure  manner
***************************************

Client : Hi, I want to talk to Bob.
Server: Yes, I am Bob. Go  ahead.
Client : Is our conversation confidential?
Server: yes.
Client : chat_STARTTLS
Server: chat_STARTTLS_NOT_SUPPORTED
Client : My ATM pin is 6883.
Server: I got it. Thank you.
Client : chat_close
root@alice1:~#
```

```
root@bob1:~# python3 secure_chat_app.py -s
I am server
Waiting for incoming connections.
Received connection from  172.31.0.4 ( 49636 )

***************************************
You are chatting in insecure  manner
***************************************

Client: Hi, I want to talk to Bob.
Server : Yes, I am Bob. Go  ahead.
Client: Is our conversation confidential?
Server : yes.
Client: My ATM pin is 6883.
Server : I got it. Thank you.
Client: chat_close
Closing connection
root@bob1:~#
```

```
root@trudy1:~# python3 secure_chat_interceptor
.py -d alice1 bob1
Downgrade attack
Trudy's server is ready.
Received connection from victm client 172.31.0
.2 ( 33124 )
Connected to victim server
client: Hi, I want to talk to Bob.
Server: Yes, I am Bob. Go  ahead.
client: Is our conversation confidential?
Server: yes.
client: chat_STARTTLS
client: My ATM pin is 6883.
Server: I got it. Thank you.
client: chat_close
Victim client closing connection
root@trudy1:~#
```

TCP stream snapshot for captured packet by Trudy

Conclusion: Here Trudy blocks the message between and can make downgrade attacks happen using DNS poisoning.

II.IV TASK 4

The following illustration shows the MITM attack message flow, where the attacker establishes a two-way socket pipeline between server and the client, thereby capturing the messages that pass through it, and modifying according to the attackers need.

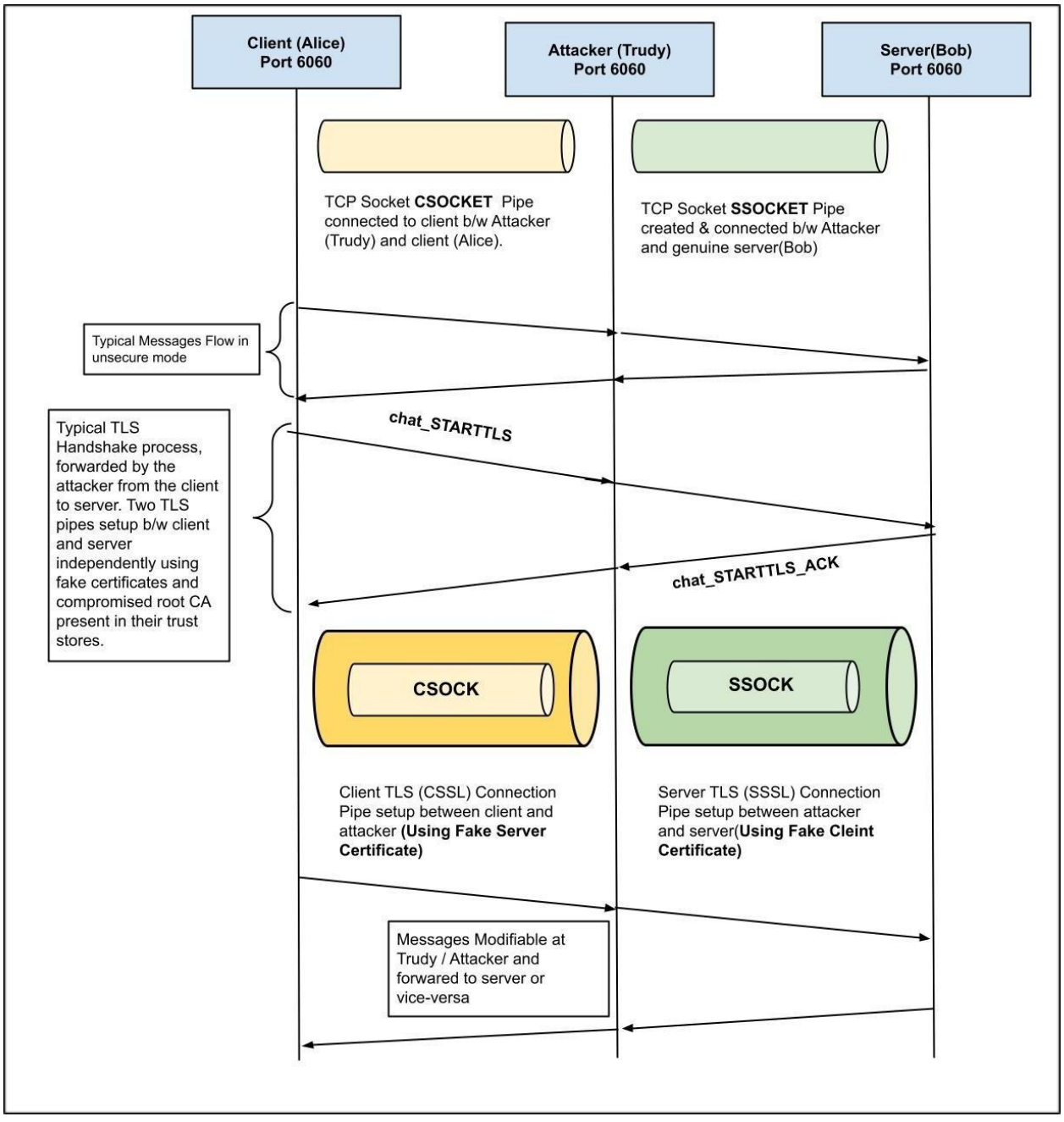


Figure: MITM Attack Message Flow

Steps to run the setup for Task4:

1. Start bash windows for Alice, Bob and Trudy.
2. Run interceptor file in Trudy using command
   Python3 secure_chat_interceptor.py -m alice1 bob1
3. Start server in Bob using command
   Python3 secure_chat_app.py -s
4. Start client in Alice bash using command
   Python3 secure_chat_app.py -c bob1

DNS poisoning is done before running the experiment setup.

Trudy has created fake certificates for Alice and Bob with the names **fake_alice.crt** and **fake_bob.crt** by following steps as mentioned below.

1. Created CSR for both Alice and Bob

```
root@trudy1:~# openssl req -newkey rsa:2048 -nodes -keyout fake_alice.key -days 365 -out fake_alice.csr
Ignoring -days; not generating a certificate
Generating a RSA private key
.....................................................+++++
.............................................+++++
writing new private key to 'fake_alice.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:TL
Locality Name (eg, city) []:Kandi
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IITH
Organizational Unit Name (eg, section) []:IITH
Common Name (e.g. server FQDN or YOUR name) []:alice
Email Address []:alice@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:alice
An optional company name []:IITH_Guest
```

```
root@trudy1:~# openssl req -newkey rsa:2048 -nodes -keyout fake_bob.key -days 365 -out fake_bob.csr
Ignoring -days; not generating a certificate
Generating a RSA private key
.........+++++
.....................................................................................................
.....................+++++
writing new private key to 'fake_bob.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:T1
Locality Name (eg, city) []:Sangareddy
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IITH
Organizational Unit Name (eg, section) []:IITH
Common Name (e.g. server FQDN or YOUR name) []:bob
Email Address []:bob@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:alice
An optional company name []:IITH_Guest
```

1. Get CSR signed by root CA assuming security for root certificate authority is compromised.

```
ns@ns03:~/root$ openssl x509 -req -CA root.crt -days 365 -in fake_alice.csr -CAkey private.ec.pem -CAcreateserial -out fake_alice.crt
Signature ok
subject=C = IN, ST = TL, L = Kandi, O = IITH, OU = IITH, CN = alice, emailAddress = alice@gmail.com
Getting CA Private Key
ns@ns03:~/root$ openssl x509 -req -CA root.crt -days 365 -in fake_bob.csr -CAkey private.ec.pem -CAcreateserial -out fake_bob.crt
Signature ok
subject=C = IN, ST = T1, L = Sangareddy, O = IITH, OU = IITH, CN = bob, emailAddress = bob@gmail.com
Getting CA Private Key
```

Our setup sample output like:

```
root@alice1:~# python3 secure_chat_app.py -c b
ob1
I am client
Connected to server

****************************************
You are chatting in insecure  manner
****************************************

Client : Alice here. Want secure chat.
Server: Yes, go with TLS security.
Client : chat_STARTTLS
Server: chat_STARTTLS_ACK

****************************************
TLS is enabled at server side. Now chat will b
e encrypted
****************************************

Client : Are we secure from  MITM?
Server: Yes.No one can listen to  our conversa
tion now.
Client : chat_close
root@alice1:~#
```

```
root@bob1:~# python3 secure_chat_app.py -s
I am server
Waiting for incoming connections.
Received connection from  172.31.0.4 ( 50348 )

****************************************
You are chatting in insecure manner
****************************************

Client: Alice here. Want secure chat.
Server : Yes, go with TLS security.
Client: chat_STARTTLS
Sending Acknowledgement

****************************************
TLS is enabled at server side. Now chat will be
 encrypted
****************************************

Client: Are we secure from  MITM?
Server : Yes.No one can listen to  our conversa
tion now.
Client: chat_close
Closing connection
root@bob1:~#
```

```
root@trudy1:~# python3 secure_chat_interceptor
.py -m alice1 bob1
MITM attack
Trudy's server is ready.
Received connection from victm client 172.31.0
.2 ( 33836 )
Connected to victim server
client: Alice here. Want secure chat.
Server: Yes, go with TLS security.
client: chat_STARTTLS
Server: chat_STARTTLS_ACK
client: Are we secure from  MITM?
Server: Yes.No one can listen to  our conversa
tion now.
client: chat_close
Victim client closing connection
root@trudy1:~#
```

At the same time, dumping for Wireshark traces was enabled to capture packets. Following snapshots show the statistics for captured packets. Number of packets captured by Trudy (Man-in-Middle) is the total of packets captured by Alice and Bob together.

For Alice:

```
ns@ns03:~$ lxc exec alice1 -- tcpdump -w alice
_task4.pcap
tcpdump: listening on eth0, link-type EN10MB (
Ethernet), capture size 262144 bytes
^C34 packets captured
34 packets received by filter
0 packets dropped by kernel
```

For Bob:

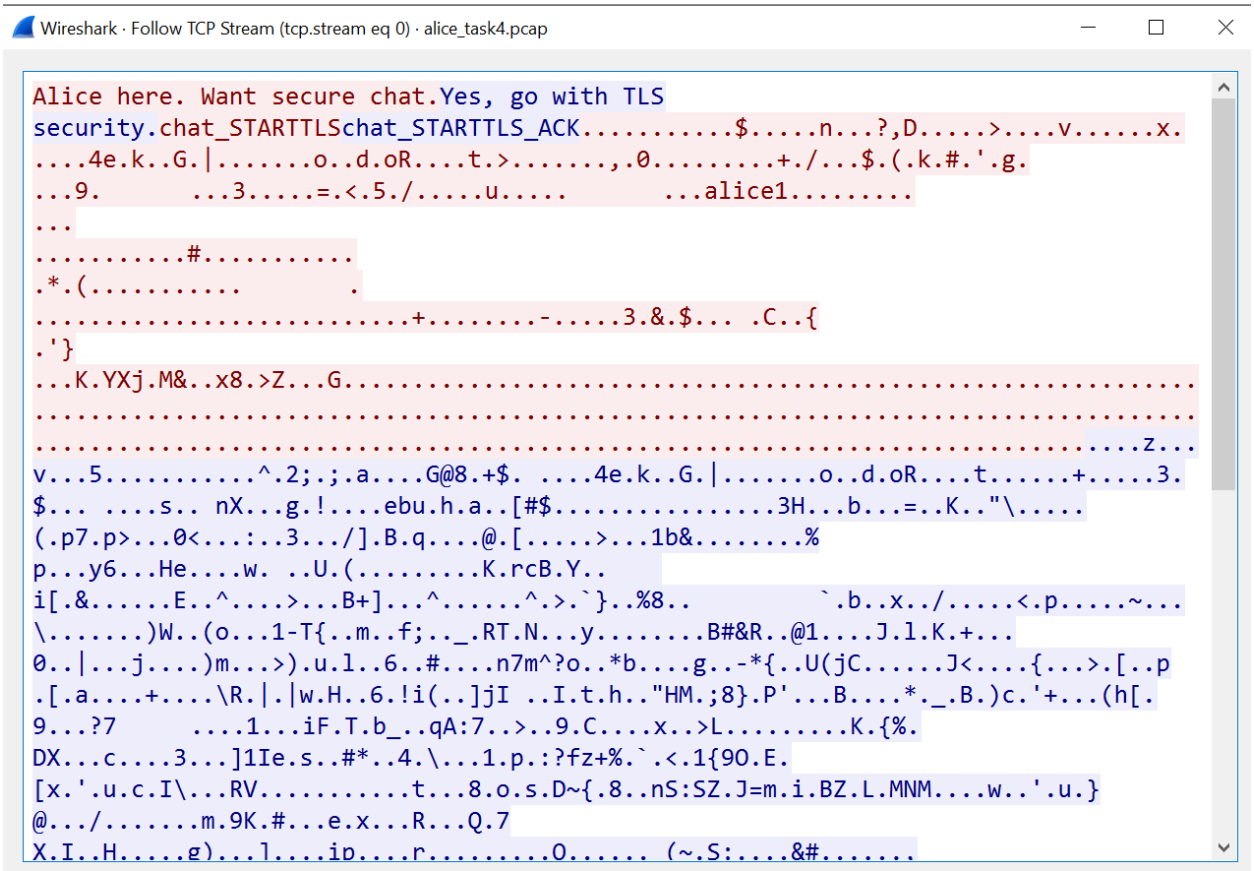
```
ns@ns03:~$ lxc exec bob1 -- tcpdump -w bob_task
4.pcap
tcpdump: listening on eth0, link-type EN10MB (E
thernet), capture size 262144 bytes
^C34 packets captured
34 packets received by filter
0 packets dropped by kernel
```

For Trudy:

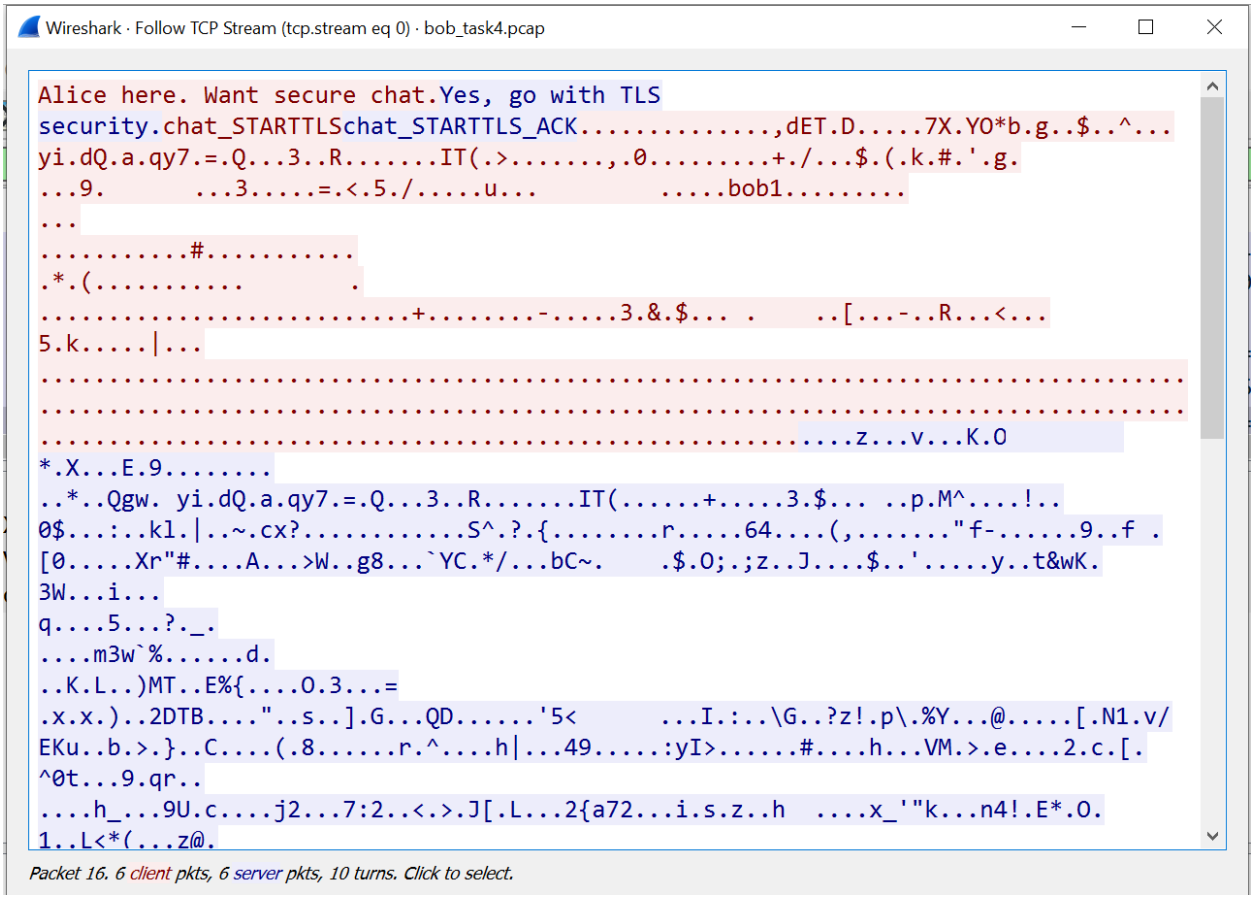
```
ns@ns03:~$ lxc exec trudy1 -- tcpdump -w trudy
_task4.pcap
tcpdump: listening on eth0, link-type EN10MB (
Ethernet), capture size 262144 bytes
^C68 packets captured
68 packets received by filter
0 packets dropped by kernel
```

TCP stream snapshots are as follows. Which tells that, even though Alice and Bob think that they are communicating securely, they are not actually!

Trace of Alice


Wireshark · Follow TCP Stream (tcp.stream eq 0) · alice_task4.pcap
Alice here. Want secure chat.Yes, go with TLS
security.chat_STARTTLSchat_STARTTLS_ACK..........$.....n...?,D.....>....v.....x.
....4e.k..G.|.......o..d.oR....t.>.......,.0.........+./...$.(.k.#.'.g.
...9.   ...3.....=.<.5./.....u.....   ...alice1.........
...
...........#...........
.*.(...........   .
.........................+........-.....3.&.$... .C..{
.'}
...K.YXj.M&..x8.>Z...G.....
....z...
v...5...........^.2;.;.a....G@8.+$. ....4e.k..G.|.......o..d.oR....t......+.....3.
$... ....s.. nX...g.!....ebu.h.a..[#$..............3H...b...=..K.."\.....
(.p7.p>...0<...:..3.../].B.q....@.[.....>...1b&........%
p...y6...He....w. ..U.(.........K.rcB.Y..
i[.&......E..^....>...B+]...^......^.>.`}..%8..          `.b..x../.....<.p.....~...
\.......)W..(o...1-T{..m..f;.._.RT.N...y........B#&R..@1....J.l.K.+...
0..|...j....)m...>).u.l..6..#....n7m^?o..*b....g..-*{..U(jC......J<....{...>.[..p
.[.a....+....\R.|.|w.H..6.!i(..]jI ..I.t.h.."HM.;8}.P'...B....*._.B.)c.'+...(h[.
9...?7   ....1...iF.T.b_..qA:7..>..9.C....x..>L.........K.{%.
DX...c....3...]1Ie.s..#*..4.\...1.p.:?fz+%.`.<.1{9O.E.
[x.'.u.c.I\...RV...........t...8.o.s.D~{.8..nS:SZ.J=m.i.BZ.L.MNM....w..'.u.}
@.../.......m.9K.#...e.x...R...Q.7
X.I..H.....g)...l....ip....r.........O..... (~.S:....&#.....

Trace of Bob:



Conclusion: Due to compromising of security for certificate authorities, attackers can issue fake certificates for real users and with the help of DNS poisoning technique, one can listen to their conversations as proofed in setup.

III. Conclusion

Through this project we were able to understand the various security aspects in a communication scenario in the real world. We identified the security frameworks used and their use cases in the real world by addressing the actual communication scenario through the TLS/SSL sessions. We also got to know the real time attacks and vulnerabilities that exist in such communication scenarios and identify their impact and workflows. First task of the assignment is carried on to know the importance and real time usage of CA certificates which can be used to establish secure communication. TLS connection security is implemented with socket programming in python. On top of that coding, downgrade attack and MITM attack is simulated in a controlled environment. Experimental setup for our assignment is limited and does not include real world patches that already patched many vulnerabilities of protocol. Major learning from this experiment is that it is not just the design of the protocol that needs to be free from vulnerabilities, sometimes implementation of protocols also pose a serious threat to secure connections. Therefore programmers should look for more secure coding by checking buffer overflow errors, message interception. For example, in downgrade attack Trudy easily convinced Alice to come down to non TLS connection.

Bibliography

i. References

[1]. **https://www.openssl.org/docs/man1.1.1/man3/**

[2]. **https:/wiki./openssl.org/index.php/Simple_TLS_Server**

[3]. **http://manpages.ubuntu.com/manpages/xenial/man8/update-ca-certificates.8.html**

ii. Deliverables Enclosed

Deliverables	Description
<cs21mtech11022|cs21mtech14003| cs21resch11008>.tgz	The main compressed tarball of the entire project directory.
Alice, Bob, Trudy Subfolders	Subfolders containing each of Alice, Bob and Trudy's files.
Certificate Request Files	alice.csr , bob.csr
Public Keys	alice_public.pem, bob_public.pem, root_public.pem
Private_Keys	alice.pem, bob.pem and root.pem
Signed Certificates	root.crt (Self signed), alice.crt, bob.crt
Hash Digests (For Integrity Check)	alice_csr_digest, alice_crt_digest, bob_csr_digest and bob_crt_digest
Program Codes	secure_chat_app.py and secure_chat_interceptor.py in their source directories

Plagiarism Statement

We 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 plagiarized 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 honor violations by other students if any of us become aware of it.

Names: Harinder, Nilesh, Ayan
Date: 07/04/2022
Signature: