Assignment 6: Secure chat using openssl and MITM attacks

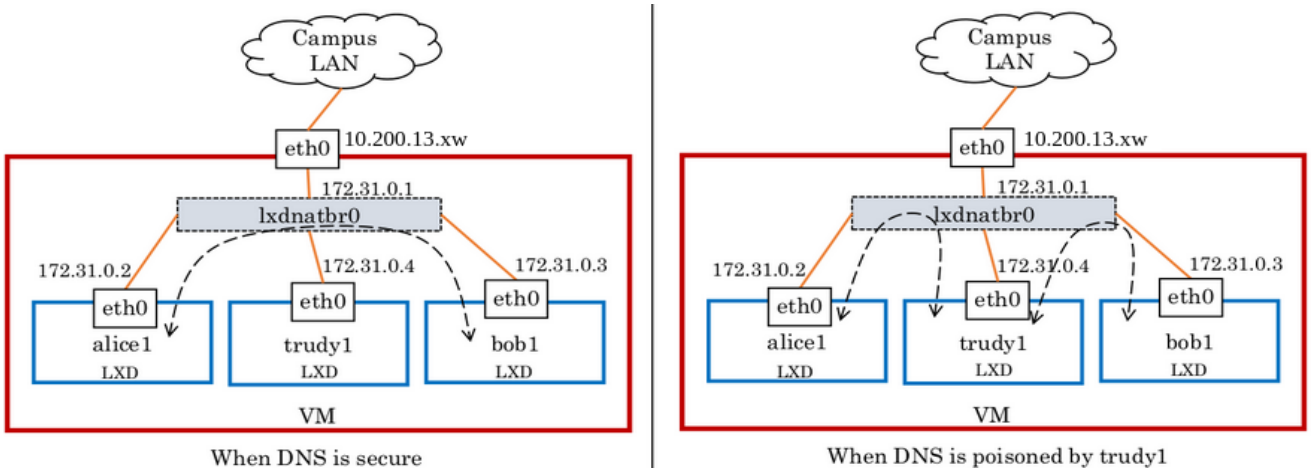
Rochisnu Dutta (ID22RESCH11016)
Nikhil Kori (CS22MTECH11014)

It's a group assignment with Max of 2 students per group playing the roles of Alice/Bob/Trudy! Select a different partner from the ones you paired up with for the other assignments and term project.

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

Setup:

Each group will be given a dedicated QEMU-KVM VM with the IP address (10.200.13.xw) in NeWS lab's cloud for completing this assignment. Refer Appendix B for some help on how to work with containers. The VM runs three LXD containers (one LXD container each for Alice, Trudy and Bob in the VM provided) which are configured in a star topology i.e., with the switch at the center, Alice, Bob, Trudy are connected to the switch ports. **Make a note of hostnames assigned to Alice, Trudy and Bob from your VM.** DNS is already set up properly so ping using hostnames from Alice to Bob and vice-versa works. Under normal circumstances, Trudy does not come in the traffic forwarding path between Alice and Bob. But when DNS is poisoned by Trudy, all traffic between Alice, and Bob is intercepted by Trudy as the MITM attacker. You can also launch a similar MITM attack using ARP cache poisoning.



Please upload ssh public key to your github profile and share your github user-id with TAs to get access to your group's VM through IITH's Wireguard VPN (if you are not on the campus). The three LXDs are reachable from inside the VM, and cannot be reached from the campus LAN.

Task 1: Generate keys and certificates (10M)

Use the OpenSSL commands to create a root CA certificate (Subject name: NTS Root R1, V3 X.509 certificate, self-signed using 512-bit ECC Private Key of the root), an intermediate CA certificate (Subject Name: NTS CA 1R3, V3 X.509 certificate with 2048-bit RSA public key, signed by NTS Root R1), a certificate of Alice (Subject Name: Alice1.com with 1024-bit RSA public key, issued i.e., signed by the intermediate CA, NTS 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, NTS 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 local machine.

How to communicate among LXDs and between VM and LXD?

You can use the Linux based commands like SCP for transferring the files like CSRs and certs.

You have to be sure about the integrity and clearly show that the files transferred are indeed sent by the intended sender and received by the intended receiver. For example, if you send the CSR to the Signing Authority (the Intermediate CA), then the signing authority should be able to verify that it is sent by the intended sender and similarly when receiving the certificate back it should be verified that it is indeed signed and sent by the actual authority. You can use signing and verification concepts applied in the Openssl tutorial for this.

Steps To create root certificate:

1) openssl ecparam -genkey -name secp512r1 -out root.key
2) openssl req -new -x509 -key root.key -out root.crt -days 365 -subj "/CN=NTS Root R1" -addext "keyUsage=critical,keyCertSign,cRLSign" -addext "basicConstraints=critical,CA:true"

```
ubuntu@ns00-gold:~$ openssl x509 -in root.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            05:24:66:35:bf:bb:53:68:cb:14:9d:e3:d3:f8:95:f2:57:53:06:2d
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = IN, ST = Telangana, L = Kandi, O = NTS Root R1, OU = Root CA, CN = NTS Root R1
        Validity
            Not Before: Mar 21 20:14:32 2023 GMT
            Not After : Mar 18 20:14:32 2033 GMT
        Subject: C = IN, ST = Telangana, L = Kandi, O = NTS Root R1, OU = Root CA, CN = NTS Root R1
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (512 bit)
                pub:
                    04:11:24:1a:9f:5f:fe:2e:44:02:45:9b:c6:e7:c3:
                    51:68:b6:86:2f:d0:7b:e1:2c:c3:07:c9:39:7e:a0:
                    8c:e4:3c:05:b5:3f:85:d8:f9:7e:03:57:d9:0f:5d:
                    1c:f8:bb:2e:72:10:65:86:5f:a5:8a:82:e4:10:82:
                    05:47:b2:b5:fe:4e:c6:dc:ff:ba:f7:da:1a:b8:6a:
                    72:08:e0:93:da:28:49:26:5f:e3:d3:a5:64:24:a2:
                    29:ef:5d:b1:68:ee:8d:0c:50:1c:47:78:15:8a:8d:
                    a2:60:b5:5c:4f:b8:8d:9f:e6:0c:4d:37:ac:1d:51:
                    73:4e:62:d2:c6:f3:ce:36:3b
                ASN1 OID: brainpoolP512r1
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                C3:3C:05:2C:0C:59:87:25:CF:3F:7A:DF:A2:09:FA:6E:7D:97:98:06
            X509v3 Authority Key Identifier:
                C3:3C:05:2C:0C:59:87:25:CF:3F:7A:DF:A2:09:FA:6E:7D:97:98:06
            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:81:84:02:40:58:d2:67:f2:81:5e:bb:64:fc:61:2e:16:70:
        86:8a:49:be:ef:02:5c:46:6b:56:9b:f5:d5:48:a3:b6:5a:17:
        93:5b:e1:15:28:99:0b:21:8a:5a:42:c2:1f:9e:0b:5b:fa:be:
        4a:ef:8e:8c:04:d0:14:8e:a0:87:31:5c:d0:f8:64:02:40:11:
        d7:3e:5e:2c:c6:46:c7:2e:b8:d0:23:19:35:37:ac:e9:94:11:
        3e:ae:ed:ef:8c:52:cc:12:5e:8c:86:c1:c8:45:6c:09:fe:8d:
        e9:9c:c2:b7:3a:2d:e7:8a:17:df:f4:68:3b:47:0b:de:af:06:
        ae:54:bc:43:e1:6e:bb:e4:dc
ubuntu@ns00-gold:~$
```

Steps to create intermediate certificate:

1) openssl genrsa -out int.key 2048
2) openssl req -new -key int.key -out int.csr -subj "/CN=NTS CA 1R3"
3) openssl x509 -req -in int.csr -CA root.crt -CAkey root.key -out int.crt -days 365 -CAcreateserial -extfile <(printf "subjectKeyIdentifier=hash\nauthorityKeyIdentifier=keyid,issuer\nkeyUsage=critical,digitalSignature,keyCertSign,cRLSign\nbasicConstraints=critical,CA:true,pathlen:0")

```
ubuntu@ns00-gold:~$ openssl x509 -in int.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            24:6e:24:3b:fc:aa:11:e3:e9:6a:a6:c6:41:63:ce:24:0b:b5:bb:90
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = IN, ST = Telangana, L = Kandi, O = NTS Root R1, OU = Root CA, CN = NTS Root R1
        Validity
            Not Before: Mar 21 20:42:47 2023 GMT
            Not After : Mar 20 20:42:47 2024 GMT
        Subject: C = IN, ST = Telangana, L = Kandi, O = NTS Intermediate, OU = NTS CA 1R3, CN = NTS CA 1R3
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ca:70:9c:22:9d:a1:06:54:bc:93:40:cf:ea:84:
                    ff:a9:8b:98:ec:1b:7f:0d:e3:09:01:d0:bb:1f:91:
                    c3:d3:31:e5:86:3f:fd:ae:33:2b:eb:26:c8:81:79:
                    02:ec:c9:e6:19:d7:cf:18:67:f0:73:44:62:0d:7a:
                    4c:99:c4:63:79:ce:f2:6b:d5:b5:27:df:20:82:4a:
                    f2:67:c8:66:f1:60:c9:c9:64:8d:58:4e:09:f1:be:
                    c6:0a:c8:20:d1:76:75:1e:43:cb:ad:4e:63:e5:21:
                    7e:f2:11:63:36:f5:c5:fe:8d:e7:af:5b:ae:fa:60:
                    ed:c2:84:70:dd:3d:04:bb:dd:3f:0d:30:52:dc:27:
                    8f:ec:48:e7:48:83:9a:ba:54:47:27:d4:31:1e:f9:
                    85:3f:8f:76:98:1d:b4:84:6c:34:c7:f9:54:2d:7f:
                    f5:4b:89:07:a3:00:24:eb:57:fb:2b:9f:32:74:d3:
                    61:26:0a:4b:04:10:45:33:7a:ff:1e:e1:f9:67:18:
                    52:a5:d2:af:73:83:8a:fb:a5:96:e0:28:2e:37:e2:
                    6c:e5:86:a7:27:53:f3:28:cb:2c:5d:18:70:2b:4c:
                    87:1e:9e:11:8b:82:6a:8a:16:3c:d6:f9:14:3a:a3:
                    e8:91:b8:06:c7:cb:8e:31:70:61:10:ec:7e:8a:e9:
                    9a:4f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                F3:47:CF:94:8D:B5:5E:57:B0:BF:E3:19:72:05:11:75:BE:6C:B8:4A
            X509v3 Authority Key Identifier:
                C3:3C:05:2C:0C:59:87:25:CF:3F:7A:DF:A2:09:FA:6E:7D:97:98:06
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:81:84:02:40:48:a3:7f:b5:10:37:26:4b:15:8c:77:b9:5d:
        cc:12:41:00:fd:51:33:22:6e:b1:97:8d:19:cc:25:d5:28:b0:
        2e:b5:47:93:d5:84:b7:9d:4a:6a:2e:46:77:b7:19:20:b0:0a:
        6c:ef:46:a6:66:5a:b3:b6:25:d9:e2:81:8e:16:79:02:40:0f:
```

Steps to create Alice's certificate:

1) openssl genrsa -out alice.key 2048 (Note: Alice key size is assumed to be 2048 bits)
2) openssl req -new -key alice.key -out alice.csr -subj "/C=IN/ST=Telangana/L=IIT/O=Alice1.com/OU=IIT/CN=Alice1.com"
3) openssl x509 -req -in alice.csr -CA int.crt -CAkey int.key -out alice.crt -days 365 -CAcreateserial -extfile <(printf "subjectAltName=DNS:Alice1.com")

```
ubuntu@ns00-gold:~$ openssl x509 -in alice.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            26:0d:64:f6:15:65:82:0f:26:73:39:5a:6a:d8:7c:24:f0:57:93:56
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IN, ST = Telangana, L = Kandi, O = NTS Intermediate, OU = NTS CA 1R3, CN = NTS CA 1R3
        Validity
            Not Before: Mar 28 12:10:34 2023 GMT
            Not After : Mar 27 12:10:34 2024 GMT
        Subject: C = IN, ST = Telangana, L = IIT, O = Alice1.com, OU = IIT, CN = Alice1.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:dc:16:36:b5:95:b1:7e:b1:69:49:ae:a7:8c:79:
                    a1:30:09:e3:de:35:9f:d7:e4:c9:1c:e6:c3:93:a7:
                    d8:9d:6c:39:16:aa:53:29:4c:19:dc:ec:42:cc:25:
                    83:91:28:ed:b1:28:ef:d2:ed:c2:ae:a4:ec:a9:27:
                    ad:bf:4a:1e:37:0e:2a:e1:47:49:9f:9c:1d:63:a5:
                    b4:3e:3a:a9:6c:f4:a2:c5:8b:53:f7:aa:9d:f5:6b:
                    26:27:7d:7b:3e:2a:08:3a:a7:4c:ae:e0:ac:24:d9:
                    17:6f:41:30:99:3f:bf:69:2b:b5:1d:a1:46:92:32:
                    a5:a2:3a:37:be:64:2f:ab:97:ac:1f:17:06:4f:ac:
                    c1:c4:3c:7d:bc:d7:d7:94:2f:90:9c:fc:04:93:71:
                    7f:48:b6:aa:92:bd:20:5a:d6:0c:1b:67:00:35:13:
                    8b:95:4d:9c:d6:10:f7:3b:7f:4c:55:ed:ec:7f:5f:
                    da:46:16:78:0b:0b:40:8f:5b:30:c5:c9:cb:ac:86:
                    29:26:dc:19:e0:2a:d5:53:7d:62:2b:7a:78:21:f3:
                    59:0f:a9:b7:03:d0:88:32:00:f8:ef:5d:cf:e9:1b:
                    a8:55:1e:7a:92:bd:f4:82:6b:d2:fe:62:90:a0:2e:
                    37:46:63:9f:89:59:d0:d0:a4:c5:a5:0f:fe:d4:1a:
                    24:41
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:Alice1.com
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        16:e8:4d:28:44:98:c0:e9:f1:20:97:ca:61:7d:38:5e:3d:c0:
        fa:ec:e7:39:23:b9:4a:b3:06:25:d6:da:b4:78:25:81:11:47:
        98:f9:a8:5f:b7:5e:0f:f0:1e:41:42:4e:c7:1b:7c:27:75:96:
        49:7f:b7:38:dd:f1:d9:3a:3a:4c:d6:eb:77:d9:ae:cc:98:e4:
        c5:8b:ba:55:ca:d2:94:58:6a:a1:78:2c:5f:97:2b:95:a2:51:
        b3:4c:fa:e3:8b:eb:cb:73:c1:a3:0c:ee:a4:d3:51:67:7b:23:
        17:3d:79:e6:6f:8d:6d:57:9b:89:4d:e6:0c:bd:7d:a5:b3:bd:
        f8:9d:e2:27:2e:f1:7a:c5:c0:e1:27:06:5a:72:d2:e4:87:dd:
        d8:a3:1b:5b:6e:7e:64:de:92:a7:15:e0:25:d6:26:cd:0a:b7:
        3c:46:42:93:81:a7:f0:a0:b1:bf:74:a4:98:11:b9:3e:2c:7f:
```

Steps to create Bob's certificate:

1) openssl ecparam -genkey -name prime256v1 -out bob.key
2) openssl req -new -key bob.key -out bob.csr -subj "/C=IN/ST=Telangana/L=Kandi/O=bob1.com/OU=IIT Hyderabad/CN=bob1.com"
3) openssl x509 -req -in bob.csr -CA int.crt -CAkey int.key -out bob.crt -days 365 -CAcreateserial -extfile <(printf"subjectAltName=DNS:Bob1.com")

```
ubuntu@ns00-gold:~$ openssl x509 -in bob.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            26:0d:64:f6:15:65:82:0f:26:73:39:5a:6a:d8:7c:24:f0:57:93:54
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IN, ST = Telangana, L = Kandi, O = NTS Intermediate, OU = NTS CA 1R3, CN = NTS CA 1R3
        Validity
            Not Before: Mar 21 20:47:42 2023 GMT
            Not After : Mar 20 20:47:42 2024 GMT
        Subject: C = IN, ST = Telangana, L = Kandi, O = Bob1.com, OU = IIT Hyderabad, CN = Bob1.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:6e:a5:5f:49:8e:62:36:51:bf:d1:96:88:79:90:
                    04:f1:81:45:8c:e6:7f:5e:8e:01:e8:29:f4:9f:5b:
                    c8:55:4a:14:a7:6a:f4:30:70:9e:5c:79:77:1a:b4:
                    fa:4b:f8:ba:3a:c5:22:ac:b7:c1:97:e2:bf:a3:74:
                    20:f5:20:59:df
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:Bob1.com
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        20:0c:8f:68:60:77:04:5a:4c:32:af:1d:86:80:f3:a2:10:b1:
        ad:5e:78:9a:e5:47:e3:19:b5:06:6b:65:1a:5e:bd:9f:6e:e0:
        f4:86:44:ab:fd:61:23:28:83:4b:12:e4:cb:63:32:28:72:98:
        bd:da:1f:44:6e:77:37:58:6e:04:dd:52:42:e9:d5:b9:87:a5:
        02:d1:15:fb:17:a4:83:13:7a:6a:8e:01:79:ec:67:4f:ca:c4:
        64:a4:1c:a3:f3:46:56:c7:71:7e:03:fa:1c:51:9c:08:53:47:
        b0:7f:4f:35:b8:77:29:74:6e:83:c0:af:6c:d0:cb:31:90:e3:
        da:81:02:b5:d5:57:a9:e7:43:cb:42:79:87:8e:58:7b:9e:e5:
        3b:f2:96:04:ae:0a:5b:90:3c:89:57:4d:1d:59:7d:df:3e:36:
        4a:b6:4e:45:2a:79:86:49:56:0c:b8:36:df:78:4c:4f:8a:d3:
        15:3d:ac:24:34:47:60:2b:03:c3:89:43:ab:00:97:22:6c:1b:
        4f:4d:ee:09:d0:a6:89:93:b4:35:2c:34:02:76:47:7d:11:d1:
        e7:82:6a:d5:96:0e:b4:9f:d3:5c:5e:41:51:19:18:79:ad:6d:
        66:5e:32:85:4e:a6:53:9f:c4:4a:0e:1e:87:04:e1:b7:8e:69:
        dd:42:a9:c7
ubuntu@ns00-gold:~$
```

To save csr files and key pairs in PEM files:

1) openssl ec -in root.key -outform PEM -out root.pem
2) openssl rsa -in int.key -outform PEM -out int.pem
3) openssl ec -in bob.key -outform PEM -out bob-private.pem
4) openssl rsa -in alice.key -outform PEM -out alice-private.pem
5) openssl req -in bob.csr -outform PEM -out bob_csr.pem
6) openssl req -in alice.csr -outform PEM -out alice_csr.pem

Verification of certificate and PEM files:

1) openssl verify -CAfile CAfile.pem int.crt (verifying intermediate certificate)
2) openssl verify -CAfile CAfile.pem alice.crt (verifying alice certificate)

3) openssl verify -CAfile CAfile.pem bob.crt (verifying bob certificate)

```
ubuntu@ns00-gold:~$ openssl verify -CAfile CAfile.pem alice.crt
alice.crt: OK
ubuntu@ns00-gold:~$ openssl verify -CAfile CAfile.pem bob.crt
bob.crt: OK
ubuntu@ns00-gold:~$
```

Task 2: Secure Chat App (20M)

Write a peer-to-peer application (`secure_chat_app`) for chatting which uses TLS 1.2 and TCP as the underlying protocols for secure and reliable communication. *Note that the* `secure_chat_app` *works like HTTPS except that here it's a peer-to-peer paradigm where Alice plays the role of the client and Bob plays the role of the server 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 TLS/TCP. 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:

a) Establish a TCP connection between Alice and Bob. Bob starts the app using "`secure_chat_app -s`", and Alice starts the app using "`secure_chat_app -c bob1`"

b) Alice sends a *chat_hello* 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.

c) 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.

   i) 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 **TLS 1.2** handshake by first sending a *client_hello* message as we discussed in the TLS lesson.

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

   iii) TLS 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.

d) Upon establishing a secure TLS 1.2 pipe, it is used by Alice and Bob to exchange their encrypted chat messages. Show that it is indeed the case by capturing Pcap traces at Alice-LXD/Bob-LXD.

e) 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.

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

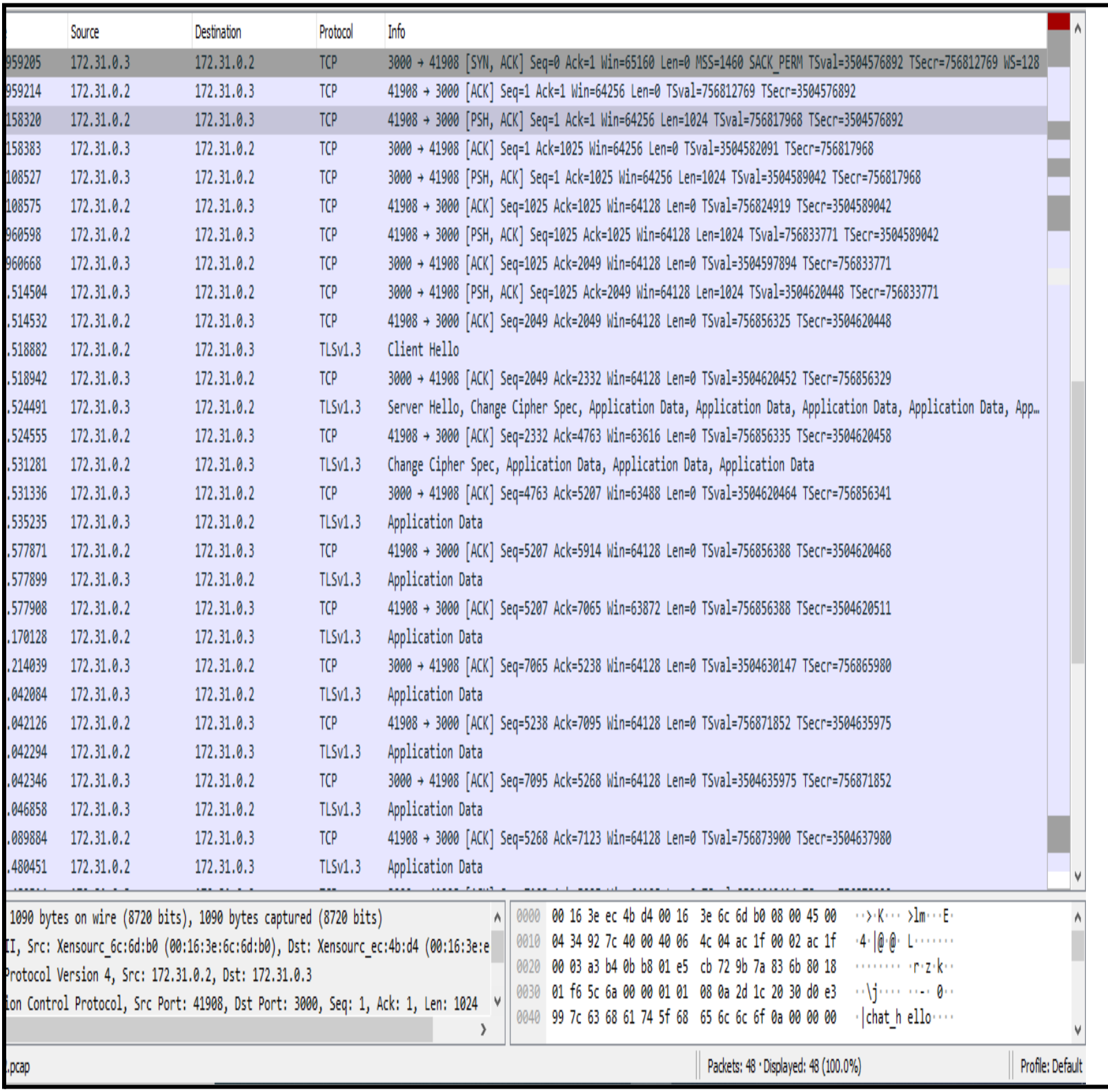
Alice successfully connects to Bob and forms a TCP connection.

```
root@alice1:~# ./a.out -c bob1
Host name is bob1
Host IP is 172.31.0.3
Connected to host_name ...
Write your message:
chat_hello
Received from server:
chat_reply
Write your message:
chat_STARTTLS
Received from server:
chat_STARTTLS_ACK
SSL context created...
Context configured...
Displaying peer certificates...
Peer certificate
Subject: /C=IN/ST=Telangana/L=Kandi/O=Bob1.com/OU=IIT Hyderabad/CN=Bob1.com
Issuer: /C=IN/ST=Telangana/L=Kandi/O=NTS Intermediate/OU=NTS CA 1R3/CN=NTS CA 1R3
Certificate verification passed
SSL version used: TLSv1.3
Write your message:
hi bob
Received from server:
hello alice
Write your message:
nice to meet you
Received from server:
same here
Write your message:
chat_close
Segmentation fault (core dumped)
root@alice1:~#
```

Later on, Alice requested Bob to start a TLS connection (chat_STARTTLS). In response to that, Bob acknowledged that request (chat_STARTTLS_ACK) and built up a TLS pipe between Alice and him. To terminate the connection, “chat_close” is sent by Alice (client).

```
root@bob1:~# ./a.out -s
Listening on 3000...
Connected to client with IP 172.31.0.2
Received from client:
chat_hello
Write your message:
chat_reply
Received from client:
chat_STARTTLS
Write your message:
chat_STARTTLS_ACK
SSL context created...
Context configured...
Displaying peer certificates...
Peer certificate
Subject: /C=IN/ST=Telangana/L=IIT/O=Alice1.com/OU=IIT/CN=Alice1.com
Issuer: /C=IN/ST=Telangana/L=Kandi/O=NTS Intermediate/OU=NTS CA 1R3/CN=NTS CA 1R3
Certificate verification passed
SSL version used: TLSv1.3
Received from client:
hi bob
Write your message:
hello alice
Received from client:
nice to meet you
Write your message:
same here
Received from client:
chat_close
ok
root@bob1:~#
```

Given below is the screenshot of the data packets showing TCP and TLS connection between Alice and Bob:



	Source	Destination	Protocol	Info
959205	172.31.0.3	172.31.0.2	TCP	3000 → 41908 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=3504576892 TSecr=756812769 WS=128
959214	172.31.0.2	172.31.0.3	TCP	41908 → 3000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=756812769 TSecr=3504576892
158320	172.31.0.2	172.31.0.3	TCP	41908 → 3000 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=1024 TSval=756817968 TSecr=3504576892
158383	172.31.0.3	172.31.0.2	TCP	3000 → 41908 [ACK] Seq=1 Ack=1025 Win=64256 Len=0 TSval=3504582091 TSecr=756817968
108527	172.31.0.3	172.31.0.2	TCP	3000 → 41908 [PSH, ACK] Seq=1 Ack=1025 Win=64256 Len=1024 TSval=3504589042 TSecr=756817968
108575	172.31.0.2	172.31.0.3	TCP	41908 → 3000 [ACK] Seq=1025 Ack=1025 Win=64128 Len=0 TSval=756824919 TSecr=3504589042
960598	172.31.0.2	172.31.0.3	TCP	41908 → 3000 [PSH, ACK] Seq=1025 Ack=1025 Win=64128 Len=1024 TSval=756833771 TSecr=3504589042
960668	172.31.0.3	172.31.0.2	TCP	3000 → 41908 [ACK] Seq=1025 Ack=2049 Win=64128 Len=0 TSval=3504597894 TSecr=756833771
.514504	172.31.0.3	172.31.0.2	TCP	3000 → 41908 [PSH, ACK] Seq=1025 Ack=2049 Win=64128 Len=1024 TSval=3504620448 TSecr=756833771
.514532	172.31.0.2	172.31.0.3	TCP	41908 → 3000 [ACK] Seq=2049 Ack=2049 Win=64128 Len=0 TSval=756856325 TSecr=3504620448
.518882	172.31.0.2	172.31.0.3	TLSv1.3	Client Hello
.518942	172.31.0.3	172.31.0.2	TCP	3000 → 41908 [ACK] Seq=2049 Ack=2332 Win=64128 Len=0 TSval=3504620452 TSecr=756856329
.524491	172.31.0.3	172.31.0.2	TLSv1.3	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data, App…
.524555	172.31.0.2	172.31.0.3	TCP	41908 → 3000 [ACK] Seq=2332 Ack=4763 Win=63616 Len=0 TSval=756856335 TSecr=3504620458
.531281	172.31.0.2	172.31.0.3	TLSv1.3	Change Cipher Spec, Application Data, Application Data, Application Data
.531336	172.31.0.3	172.31.0.2	TCP	3000 → 41908 [ACK] Seq=4763 Ack=5207 Win=63488 Len=0 TSval=3504620464 TSecr=756856341
.535235	172.31.0.3	172.31.0.2	TLSv1.3	Application Data
.577871	172.31.0.2	172.31.0.3	TCP	41908 → 3000 [ACK] Seq=5207 Ack=5914 Win=64128 Len=0 TSval=756856388 TSecr=3504620468
.577899	172.31.0.3	172.31.0.2	TLSv1.3	Application Data
.577908	172.31.0.2	172.31.0.3	TCP	41908 → 3000 [ACK] Seq=5207 Ack=7065 Win=63872 Len=0 TSval=756856388 TSecr=3504620511
.170128	172.31.0.2	172.31.0.3	TLSv1.3	Application Data
.214039	172.31.0.3	172.31.0.2	TCP	3000 → 41908 [ACK] Seq=7065 Ack=5238 Win=64128 Len=0 TSval=3504630147 TSecr=756865980
.042084	172.31.0.3	172.31.0.2	TLSv1.3	Application Data
.042126	172.31.0.2	172.31.0.3	TCP	41908 → 3000 [ACK] Seq=5238 Ack=7095 Win=64128 Len=0 TSval=756871852 TSecr=3504635975
.042294	172.31.0.2	172.31.0.3	TLSv1.3	Application Data
.042346	172.31.0.3	172.31.0.2	TCP	3000 → 41908 [ACK] Seq=7095 Ack=5268 Win=64128 Len=0 TSval=3504635975 TSecr=756871852
.046858	172.31.0.3	172.31.0.2	TLSv1.3	Application Data
.089884	172.31.0.2	172.31.0.3	TCP	41908 → 3000 [ACK] Seq=5268 Ack=7123 Win=64128 Len=0 TSval=756873900 TSecr=3504637980
.480451	172.31.0.2	172.31.0.3	TLSv1.3	Application Data

```
1090 bytes on wire (8720 bits), 1090 bytes captured (8720 bits)
II, Src: Xensourc_6c:6d:b0 (00:16:3e:6c:6d:b0), Dst: Xensourc_ec:4b:d4 (00:16:3e:e
Protocol Version 4, Src: 172.31.0.2, Dst: 172.31.0.3
ion Control Protocol, Src Port: 41908, Dst Port: 3000, Seq: 1, Ack: 1, Len: 1024
```

```
0000  00 16 3e ec 4b d4 00 16  3e 6c 6d b0 08 00 45 00   ··>·K··· >lm···E·
0010  04 34 92 7c 40 00 40 06  4c 04 ac 1f 00 02 ac 1f   ·4·|@·@· L·······
0020  00 03 a3 b4 0b b8 01 e5  cb 72 9b 7a 83 6b 80 18   ········ ·r·z·k··
0030  01 f6 5c 6a 00 00 01 01  08 0a 2d 1c 20 30 d0 e3   ··\j···· ··-· 0··
0040  99 7c 63 68 61 74 5f 68  65 6c 6c 6f 0a 00 00 00   ·|chat_h ello····
```

.pcap | Packets: 48 · Displayed: 48 (100.0%) | Profile: Default

Task 3: START_SSL downgrade attack #1 for eavesdropping (20M)

Downgrade attack by Trudy by blocking 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_STARTTLS* to Bob, it assumes that Bob does not have capability to set up secure chat communication.

b) In this attack, Trudy blocks *chat_START_SSL* from reaching Bob and sends a fake 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.

c) You need to write a program (`secure_chat_interceptor`) to launch this downgrade attack (`-d` command line option) from Trudy-LXD. For this task, you can assume that Trudy poisoned 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.

To poison /etc/hosts file of Alice, Bob containers, use the following command from inside the VM

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

To revert back the /etc/hosts file of Alice, Bob containers, use the following command from inside the VM.

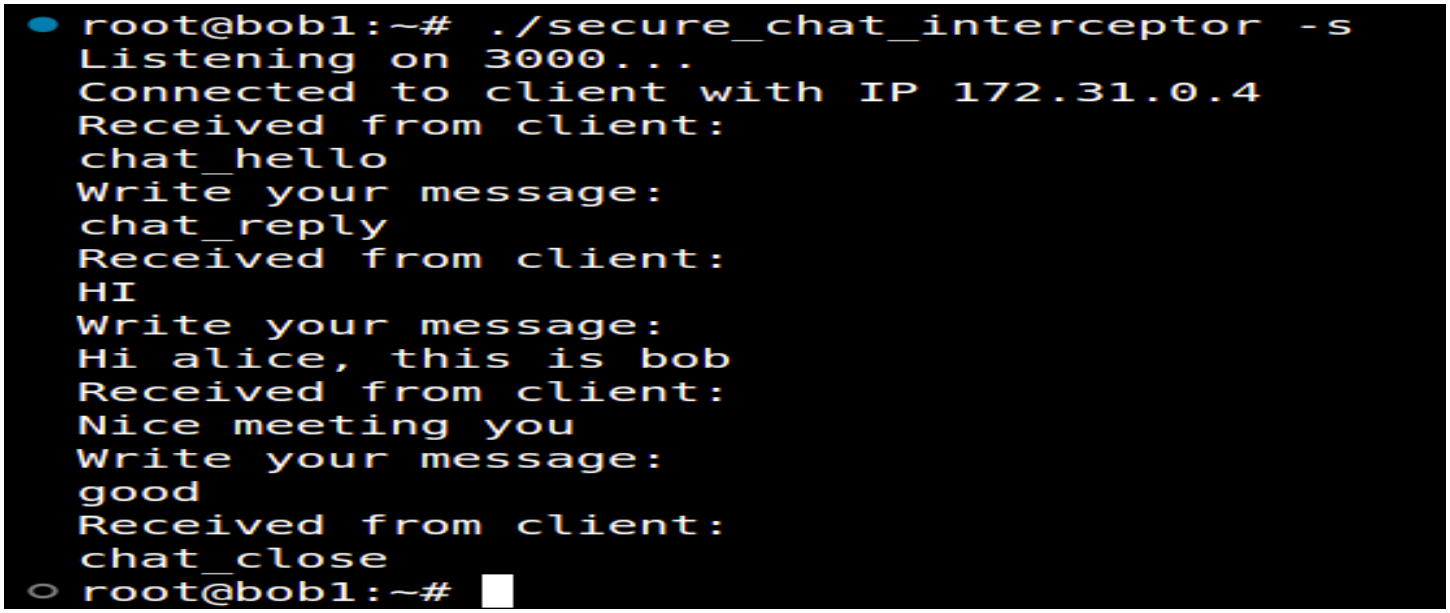
`bash ~/unpoison-dns-alice1-bob1.sh`

To start a downgrade attack, start the secure chat interceptor program using the following command from inside the Trudy LXD.

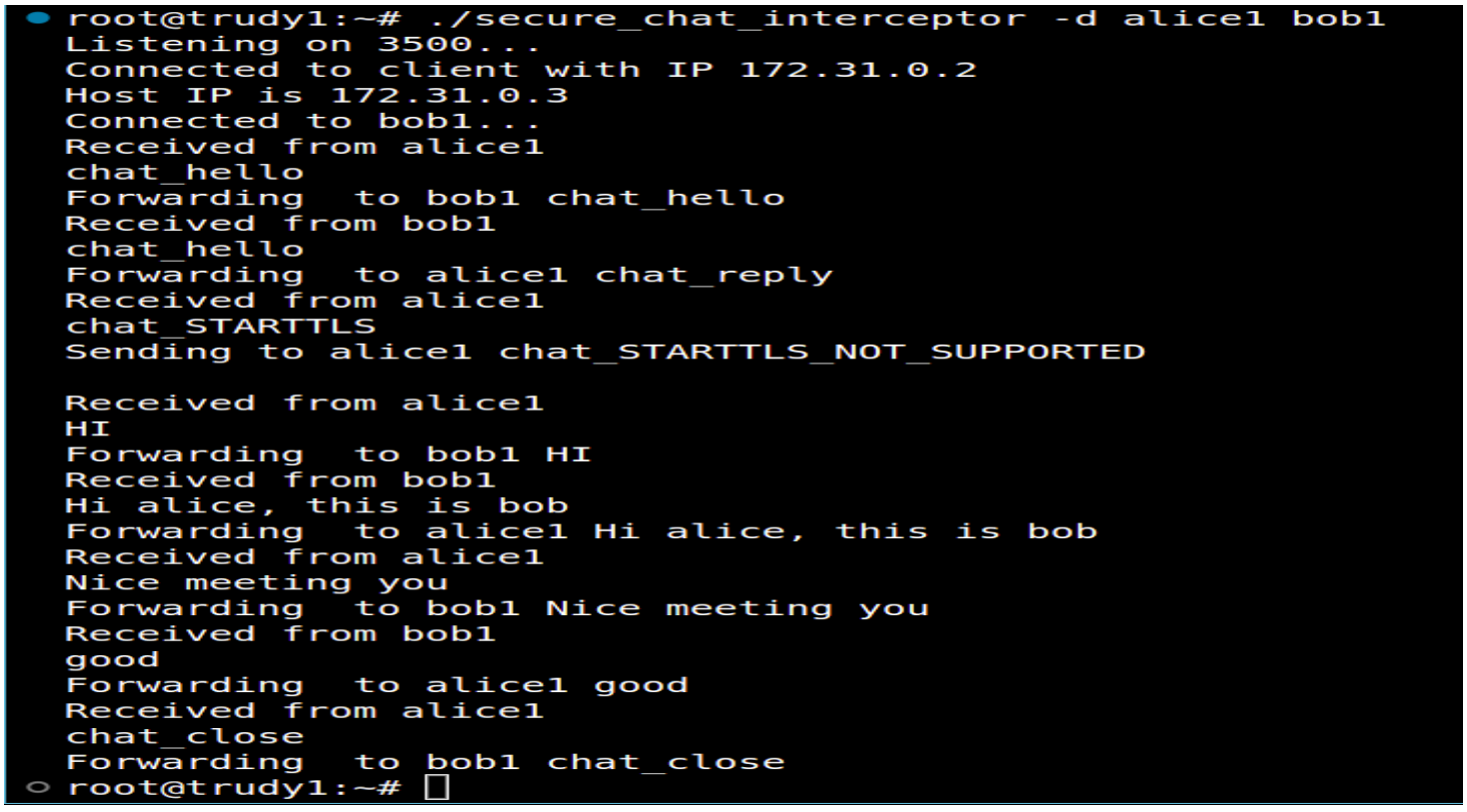
`./secure_chat_interceptor -d alice1 bob1`

Alice1:

Bob1:



Trudy1:

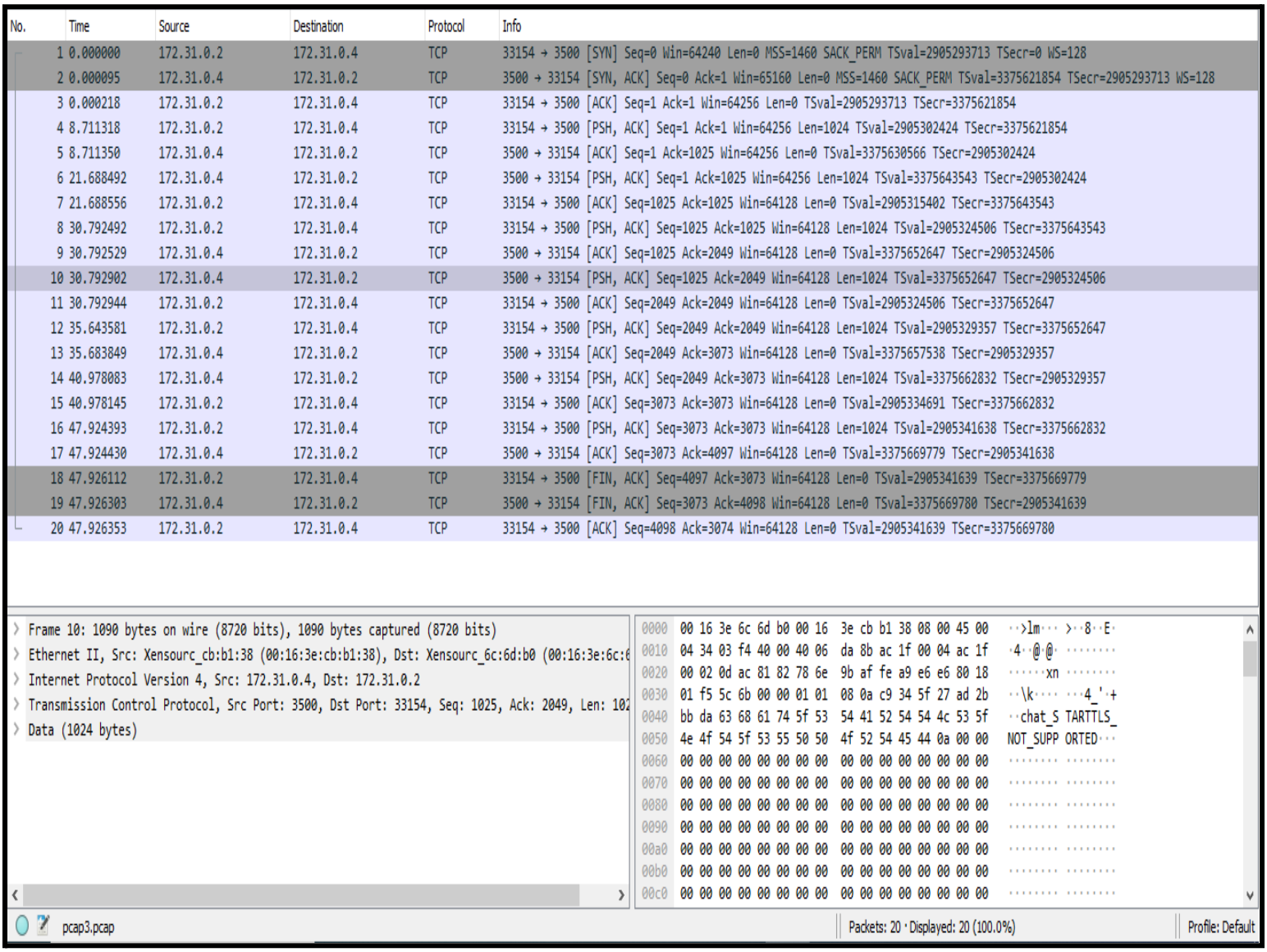


In this task, Trudy launches a downgrade attack and successfully prevents Alice and Bob from establishing a secure TLS connection. The moment Alice asked to start a TLS connection

(chat_STARTTLS), Trudy disrupted it by sending Alice “chat_STARTTLS _NOT_SUPPORTED” showing that Bob is unable to establish a TLS connection. So, they started to exchange messages in plain text.

In this case, as we poisoned host files. So, Alice instead of building a connection with Bob, it establishes a connection with Trudy (which eventually acts as Bob). Exactly the same thing is being replicated for Bob where Trudy acts as Alice.

Data packets exchange between Alice and Trudy is shown below:



No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.31.0.2	172.31.0.4	TCP	33154 → 3500 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2905293713 TSecr=0 WS=128
2	0.000095	172.31.0.4	172.31.0.2	TCP	3500 → 33154 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=3375621854 TSecr=2905293713 WS=128
3	0.000218	172.31.0.2	172.31.0.4	TCP	33154 → 3500 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2905293713 TSecr=3375621854
4	8.711318	172.31.0.2	172.31.0.4	TCP	33154 → 3500 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=1024 TSval=2905302424 TSecr=3375621854
5	8.711350	172.31.0.4	172.31.0.2	TCP	3500 → 33154 [ACK] Seq=1 Ack=1025 Win=64256 Len=0 TSval=3375630566 TSecr=2905302424
6	21.688492	172.31.0.4	172.31.0.2	TCP	3500 → 33154 [PSH, ACK] Seq=1 Ack=1025 Win=64256 Len=1024 TSval=3375643543 TSecr=2905302424
7	21.688556	172.31.0.2	172.31.0.4	TCP	33154 → 3500 [ACK] Seq=1025 Ack=1025 Win=64128 Len=0 TSval=2905315402 TSecr=3375643543
8	30.792492	172.31.0.2	172.31.0.4	TCP	33154 → 3500 [PSH, ACK] Seq=1025 Ack=1025 Win=64128 Len=1024 TSval=2905324506 TSecr=3375643543
9	30.792529	172.31.0.4	172.31.0.2	TCP	3500 → 33154 [ACK] Seq=1025 Ack=2049 Win=64128 Len=0 TSval=3375652647 TSecr=2905324506
10	30.792902	172.31.0.4	172.31.0.2	TCP	3500 → 33154 [PSH, ACK] Seq=1025 Ack=2049 Win=64128 Len=1024 TSval=3375652647 TSecr=2905324506
11	30.792944	172.31.0.2	172.31.0.4	TCP	33154 → 3500 [ACK] Seq=2049 Ack=2049 Win=64128 Len=0 TSval=2905324506 TSecr=3375652647
12	35.643581	172.31.0.2	172.31.0.4	TCP	33154 → 3500 [PSH, ACK] Seq=2049 Ack=2049 Win=64128 Len=1024 TSval=2905329357 TSecr=3375652647
13	35.683849	172.31.0.4	172.31.0.2	TCP	3500 → 33154 [ACK] Seq=2049 Ack=3073 Win=64128 Len=0 TSval=3375657538 TSecr=2905329357
14	40.978083	172.31.0.4	172.31.0.2	TCP	3500 → 33154 [PSH, ACK] Seq=2049 Ack=3073 Win=64128 Len=1024 TSval=3375662832 TSecr=2905329357
15	40.978145	172.31.0.2	172.31.0.4	TCP	33154 → 3500 [ACK] Seq=3073 Ack=3073 Win=64128 Len=0 TSval=2905334691 TSecr=3375662832
16	47.924393	172.31.0.2	172.31.0.4	TCP	33154 → 3500 [PSH, ACK] Seq=3073 Ack=3073 Win=64128 Len=1024 TSval=2905341638 TSecr=3375662832
17	47.924430	172.31.0.4	172.31.0.2	TCP	3500 → 33154 [ACK] Seq=3073 Ack=4097 Win=64128 Len=0 TSval=3375669779 TSecr=2905341638
18	47.926112	172.31.0.2	172.31.0.4	TCP	33154 → 3500 [FIN, ACK] Seq=4097 Ack=3073 Win=64128 Len=0 TSval=2905341639 TSecr=3375669779
19	47.926303	172.31.0.4	172.31.0.2	TCP	3500 → 33154 [FIN, ACK] Seq=3073 Ack=4098 Win=64128 Len=0 TSval=3375669780 TSecr=2905341639
20	47.926353	172.31.0.2	172.31.0.4	TCP	33154 → 3500 [ACK] Seq=4098 Ack=3074 Win=64128 Len=0 TSval=2905341639 TSecr=3375669780

```
> Frame 10: 1090 bytes on wire (8720 bits), 1090 bytes captured (8720 bits)
> Ethernet II, Src: Xensourc_cb:b1:38 (00:16:3e:cb:b1:38), Dst: Xensourc_6c:6d:b0 (00:16:3e:6c:6
> Internet Protocol Version 4, Src: 172.31.0.4, Dst: 172.31.0.2
> Transmission Control Protocol, Src Port: 3500, Dst Port: 33154, Seq: 1025, Ack: 2049, Len: 102
> Data (1024 bytes)
```

```
0000  00 16 3e 6c 6d b0 00 16  3e cb b1 38 08 00 45 00   ··>lm··· >··8··E·
0010  04 34 03 f4 40 00 40 06  da 8b ac 1f 00 04 ac 1f   ·4··@·@· ········
0020  00 02 0d ac 81 82 78 6e  9b af fe a9 e6 e6 80 18   ······xn ········
0030  01 f5 5c 6b 00 00 01 01  08 0a c9 34 5f 27 ad 2b   ··\k···· ···4_'·+
0040  bb da 63 68 61 74 5f 53  54 41 52 54 54 4c 53 5f   ··chat_S TARTTLS_
0050  4e 4f 54 5f 53 55 50 50  4f 52 54 45 44 0a 00 00   NOT_SUPP ORTED···
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```

pcap3.pcap | Packets: 20 · Displayed: 20 (100.0%) | Profile: Default

Task 4: START_SSL downgrade attack #2 for tampering (30M)

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

b) Rather than launching the START_SSL downgrade attack #1, 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 TLS 1.2 pipes are set up: one between Alice and Trudy; the other between Trudy and Bob. Trudy is now like a malicious proxy and decrypts messages from Alice to Bob (and from Bob to Alice) and re-encrypts 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.

c) Enhance 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`

Alice:

```
root@alice1:~# ./task4 -c bob1
Host name is bob1
Host IP is 172.31.0.4
Connected to host_name ...
Write your message:
chat_hello
Received from server:
chat_reply
Write your message:
chat_STARTTLS
Received from server:
chat_STARTTLS_ACK
SSL context created...
Context configured...
Displaying peer certificates...
Peer certificate
Subject: /C=IN/ST=Telangana/L=Kandi/O=bob1.com/OU=IIT Hyderabad/CN=bob1.com
Issuer: /C=IN/ST=Telangana/L=Kandi/O=NTS Intermediate/OU=NTS CA 1R3/CN=NTS CA 1R3
Certificate verification passed
SSL version used: TLSv1.3
Write your message:
Hi bob, this is alice
Received from server:
Hi alice, how are you.
Write your message:
this is the password: Alice@123
Received from server:
thnq alice
Write your message:
chat_close
Segmentation fault (core dumped)
root@alice1:~#
```

As mentioned above, a TLS pipe is created between Alice and Trudy

Bob:

```
root@bob1:~# ./secure_chat -s
Listening on 3000...
Connected to client with IP 172.31.0.4
Received from client:
chat_hello
Write your message:
chat_reply
Received from client:
chat_STARTTLS
Write your message:
chat_STARTTLS_ACK
SSL context created...
Context configured...
Displaying peer certificates...
Peer certificate
Subject: /C=IN/ST=Telangana/L=IIT/O=Alice1.com/OU=IIT/CN=Alice1.com
Issuer: /C=IN/ST=Telangana/L=Kandi/O=NTS Intermediate/OU=NTS CA 1R3/CN=NTS CA 1R3
Certificate verification passed
SSL version used: TLSv1.3
Received from client:
Hi bob, this is alice
Write your message:
Hi alice, how are you.
Received from client:
this is the password: Alice@123
Write your message:
thnq alice
Received from client:
chat_close
ok
root@bob1:~#
```

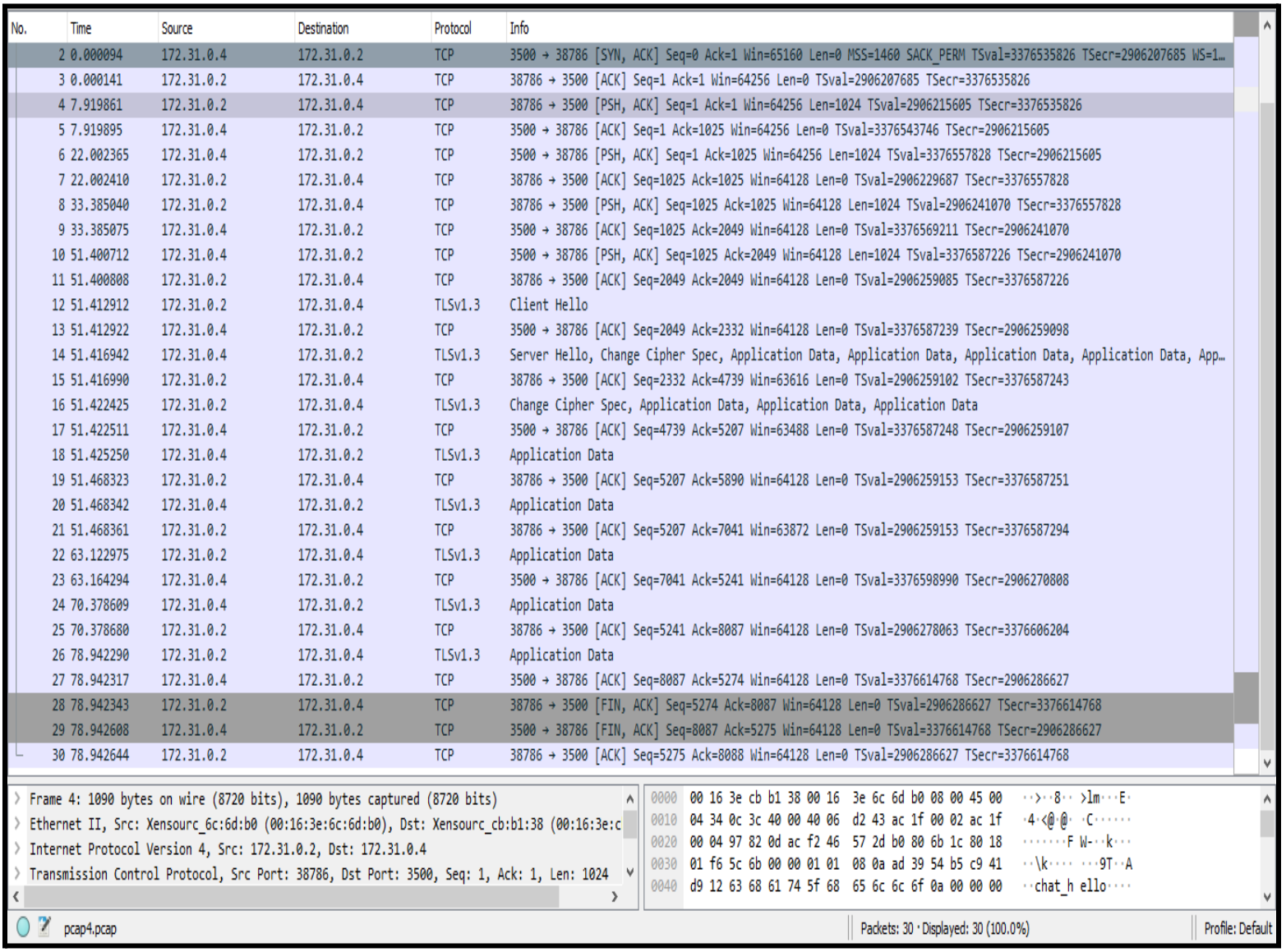
A second TLS pipe is established between Bob and Trudy.

Trudy:

```
root@trudy1:~# ./try -m alice1 bob1
Listening on 3500...
Connected to client with IP 172.31.0.2
Host IP is 172.31.0.3
Connected to bob1...
Received from alice1
chat_hello
Forwarding  to bob1 chat_hello
Received from bob1
chat_hello
Forwarding  to alice1 chat_reply
Received from alice1
chat_STARTTLS
Forwarding  to bob1 chat_STARTTLS
Received from bob1
chat_STARTTLS
Forwarding  to alice1 chat_STARTTLS_ACK
Alice SSL context created...
Alice Context configured...
Bob SSL context created...
Bob Context configured...
Displaying alice peer certificates...
Peer certificate
Subject: /C=IN/ST=Telangana/L=IIT/O=Alice1.com/OU=IIT/CN=Alice1.com
Issuer: /C=IN/ST=Telangana/L=Kandi/O=NTS Intermediate/OU=NTS CA 1R3/CN=NTS CA 1R3
Displaying bob peer certificates...
Peer certificate
Subject: /C=IN/ST=Telangana/L=Kandi/O=Bob1.com/OU=IIT Hyderabad/CN=Bob1.com
Issuer: /C=IN/ST=Telangana/L=Kandi/O=NTS Intermediate/OU=NTS CA 1R3/CN=NTS CA 1R3
Alice Certificate verification passed
Bob Certificate verification passed
Alice SSL version used: TLSv1.3
Bob SSL version used: TLSv1.3
```

```
Alice Certificate verification passed
Bob Certificate verification passed
Alice SSL version used: TLSv1.3
Bob SSL version used: TLSv1.3
Received from alice1
Hi bob, this is alice
Forwarding  to bob1 Hi bob, this is alice
Received from bob1
Hi alice, how are you.
Forwarding  to alice1 Hi alice, how are you.
Received from alice1
this is the password: Alice@123
Forwarding  to bob1 this is the password: Alice@123
Received from bob1
thnq alice
Forwarding  to alice1 thnq alice
Received from alice1
chat_close
Forwarding  to bob1 chat_close
root@trudy1:~#
```

Data packets captured between Alice and Trudy is shown below:



No.	Time	Source	Destination	Protocol	Info
2	0.000094	172.31.0.4	172.31.0.2	TCP	3500 → 38786 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=3376535826 TSecr=2906207685 WS=1…
3	0.000141	172.31.0.2	172.31.0.4	TCP	38786 → 3500 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2906207685 TSecr=3376535826
4	7.919861	172.31.0.2	172.31.0.4	TCP	38786 → 3500 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=1024 TSval=2906215605 TSecr=3376535826
5	7.919895	172.31.0.4	172.31.0.2	TCP	3500 → 38786 [ACK] Seq=1 Ack=1025 Win=64256 Len=0 TSval=3376543746 TSecr=2906215605
6	22.002365	172.31.0.4	172.31.0.2	TCP	3500 → 38786 [PSH, ACK] Seq=1 Ack=1025 Win=64256 Len=1024 TSval=3376557828 TSecr=2906215605
7	22.002410	172.31.0.2	172.31.0.4	TCP	38786 → 3500 [ACK] Seq=1025 Ack=1025 Win=64128 Len=0 TSval=2906229687 TSecr=3376557828
8	33.385040	172.31.0.2	172.31.0.4	TCP	38786 → 3500 [PSH, ACK] Seq=1025 Ack=1025 Win=64128 Len=1024 TSval=2906241070 TSecr=3376557828
9	33.385075	172.31.0.4	172.31.0.2	TCP	3500 → 38786 [ACK] Seq=1025 Ack=2049 Win=64128 Len=0 TSval=3376569211 TSecr=2906241070
10	51.400712	172.31.0.4	172.31.0.2	TCP	3500 → 38786 [PSH, ACK] Seq=1025 Ack=2049 Win=64128 Len=1024 TSval=3376587226 TSecr=2906241070
11	51.400808	172.31.0.2	172.31.0.4	TCP	38786 → 3500 [ACK] Seq=2049 Ack=2049 Win=64128 Len=0 TSval=2906259085 TSecr=3376587226
12	51.412912	172.31.0.2	172.31.0.4	TLSv1.3	Client Hello
13	51.412922	172.31.0.4	172.31.0.2	TCP	3500 → 38786 [ACK] Seq=2049 Ack=2332 Win=64128 Len=0 TSval=3376587239 TSecr=2906259098
14	51.416942	172.31.0.4	172.31.0.2	TLSv1.3	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data, App…
15	51.416990	172.31.0.2	172.31.0.4	TCP	38786 → 3500 [ACK] Seq=2332 Ack=4739 Win=63616 Len=0 TSval=2906259102 TSecr=3376587243
16	51.422425	172.31.0.2	172.31.0.4	TLSv1.3	Change Cipher Spec, Application Data, Application Data, Application Data
17	51.422511	172.31.0.4	172.31.0.2	TCP	3500 → 38786 [ACK] Seq=4739 Ack=5207 Win=63488 Len=0 TSval=3376587248 TSecr=2906259107
18	51.425250	172.31.0.4	172.31.0.2	TLSv1.3	Application Data
19	51.468323	172.31.0.2	172.31.0.4	TCP	38786 → 3500 [ACK] Seq=5207 Ack=5890 Win=64128 Len=0 TSval=2906259153 TSecr=3376587251
20	51.468342	172.31.0.4	172.31.0.2	TLSv1.3	Application Data
21	51.468361	172.31.0.2	172.31.0.4	TCP	38786 → 3500 [ACK] Seq=5207 Ack=7041 Win=63872 Len=0 TSval=2906259153 TSecr=3376587294
22	63.122975	172.31.0.2	172.31.0.4	TLSv1.3	Application Data
23	63.164294	172.31.0.4	172.31.0.2	TCP	3500 → 38786 [ACK] Seq=7041 Ack=5241 Win=64128 Len=0 TSval=3376598990 TSecr=2906270808
24	70.378609	172.31.0.4	172.31.0.2	TLSv1.3	Application Data
25	70.378680	172.31.0.2	172.31.0.4	TCP	38786 → 3500 [ACK] Seq=5241 Ack=8087 Win=64128 Len=0 TSval=2906278063 TSecr=3376606204
26	78.942290	172.31.0.2	172.31.0.4	TLSv1.3	Application Data
27	78.942317	172.31.0.4	172.31.0.2	TCP	3500 → 38786 [ACK] Seq=8087 Ack=5274 Win=64128 Len=0 TSval=3376614768 TSecr=2906286627
28	78.942343	172.31.0.2	172.31.0.4	TCP	38786 → 3500 [FIN, ACK] Seq=5274 Ack=8087 Win=64128 Len=0 TSval=2906286627 TSecr=3376614768
29	78.942608	172.31.0.4	172.31.0.2	TCP	3500 → 38786 [FIN, ACK] Seq=8087 Ack=5275 Win=64128 Len=0 TSval=3376614768 TSecr=2906286627
30	78.942644	172.31.0.2	172.31.0.4	TCP	38786 → 3500 [ACK] Seq=5275 Ack=8088 Win=64128 Len=0 TSval=2906286627 TSecr=3376614768

```
> Frame 4: 1090 bytes on wire (8720 bits), 1090 bytes captured (8720 bits)
> Ethernet II, Src: Xensourc_6c:6d:b0 (00:16:3e:6c:6d:b0), Dst: Xensourc_cb:b1:38 (00:16:3e:c
> Internet Protocol Version 4, Src: 172.31.0.2, Dst: 172.31.0.4
> Transmission Control Protocol, Src Port: 38786, Dst Port: 3500, Seq: 1, Ack: 1, Len: 1024
```

```
0000  00 16 3e cb b1 38 00 16  3e 6c 6d b0 08 00 45 00   ··>··8·· >lm···E·
0010  04 34 0c 3c 40 00 40 06  d2 43 ac 1f 00 02 ac 1f   ·4·<@·@· ·C······
0020  00 04 97 82 0d ac f2 46  57 2d b0 80 6b 1c 80 18   ·······F W-··k···
0030  01 f6 5c 6b 00 00 01 01  08 0a ad 39 54 b5 c9 41   ··\k···· ···9T··A
0040  d9 12 63 68 61 74 5f 68  65 6c 6c 6f 0a 00 00 00   ··chat_h ello····
```

pcap4.pcap | Packets: 30 · Displayed: 30 (100.0%) | Profile: Default

Details of your chat protocol like its headers and typical message flow. For example, HTTP uses GET/POST/OK methods for message flow between client and server

A client programme sends messages to a server, which then relays them to one or more other clients, as is the case with most chat protocols. Some of the typical headers and message flows used in chat protocols are listed below:

1. Alice established a connection with Bob through his hostname and port number.

2. chat_hello : Alice sent this message to Bob.

3. chat_reply: Bob sent this message in response to Alice.

4. chat_STARTTLS: Till now the messages are exchanged in plain text. So, Alice sent this message to Bob and requested him to establish a secure TLS connection.

5. chat_STARTTLS_ACK: In response to the above message, Bob sent this reply to Alice and created a TLS pipe.

6. After this set up, all the messages are going to be exchanged in an encrypted format.

Once the host file is poisoned i.e Trudy successfully replace Bob's IP address with that of her

7. chat_STARTTLS_NOT_SUPPORTED: Whenever Alice requests for a TLS setup, Trudy will block that request and send this message to indicate that a secure connection cannot be established.

After that, Trudy will successfully fool Alice and Bob by impersonating their fake certificates. In this way, she launches a Man-in-the-middle attack.

8. chat_close: This command is used to terminate the TLS and TCP connections between Bob and Alice.

Details on how various MITM attacks are realized by Trudy

A cyberattack known as a "man-in-the-middle" (MITM) attack occurs when an attacker intercepts communication between two parties and modifies it for their own malevolent ends. As the attacker, Trudy is capable of carrying out a variety of MITM assaults using various techniques, including:

1. ARP Spoofing: To link Trudy's MAC address to the router's IP address, Trudy can send forged Address Resolution Protocol (ARP) packets to the victim's device. By doing so, Trudy will be able to eavesdrop on traffic between the victim and the router.

2. DNS spoofing: By changing the Domain Name System (DNS) records, Trudy might cause traffic from the victim to be diverted to a malicious website that imitates the target website in every way. The victim's login credentials or other private data can then be captured by Trudy.

3. SSL/TLS Stripping: Trudy can intercept SSL/TLS traffic and change it to reduce the connection's encryption level (downgrade attack). By doing this, Trudy will be able to record sensitive data in plaintext, including login credentials and credit card details.

4. Email spoofing: Trudy can send the victim an email while posing as a trustworthy sender and include a link to a malicious website or file. When the victim clicks the link or downloads the attachment, Trudy can take over the victim's computer and stealthy spyware.

5. WiFi Spoofing: By creating a fake WiFi network that mimics a real one, Trudy can deceive the victim's device into connecting to it. The victim's internet traffic can then be intercepted by Trudy, enabling Trudy to gather private data.

In our example, we mainly show TLS stripping where trudy is actually intercepting the communication between Alice and Bob.

Contribution:-

Rochisnu:

1. Created all the certificates and key pairs using openSSL commands.

2. For this assignment, I become Bob, the server.

3. I wrote the server program, which basically provides service to the client and waits for client connection.

4. It listens for incoming connection requests on a specific port number and IP address, and when a client connects, it creates a new socket to handle the communication with that client.

5. The server program typically performs some type of processing or computation based on the client's requests and returns the results to the client.

6. Besides this, I act as Trudy to launch a downgrade attack (eavesdropping).

7. I wrote the report.

Nikhil:-

1. I become Alice, the client.
2. I wrote the client side program which initiates a connection to the server and sends requests to it. The client program typically connects to a specific IP address and port number of the server, and then sends one or more messages to the server requesting a specific service. In this case the service is a TLS connection.

3. Once the server responds to the client's request, the client terminates the connection.
4. Later on, I poison the host files by using the openSSL command : `bash ~/poison-dns-alice1-bob1.sh`
5. I wrote the program for Trudy in which she launches a man-in-the-middle attack.
6. I capture the packet traces inside the container and perform all the verification tests.

ANTI-PLAGIARISM STATEMENT

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: Rochisnu Dutta
Date: 2/04/2023
Signature: Rochisnu Dutta (ID22RESCH11016)

Names: Nikhil Kori
Date: 2/04/2023
Signature: Nikhil Kori (CS22MTECH11014)

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)

The assignment will be evaluated by the TAs on a scheduled date and time. Make sure your demo works without any glitches.

Late Submission Policy: 10% penalty for each late day beyond the buffer/slip days.

Appendix: Some help regarding the setup

- To login to the VM allocated, type **`ssh ns@10.200.13.xw`** command.
- To list out the containers inside the VM, use **`lxc ls`** command from inside the VM.
- To login to a container, use **`lxc exec <containername> bash`** command from inside the VM.
- To logout from the respective container use **`exit`** command.
- To capture packet traces inside a container (eg. trudy1), use **`lxc exec trudy1 -- sudo tcpdump -i eth1 -nn not tcp port 22`**
- Root file-system of a container (eg alice1) is located as a subtree under its storage pool **`/var/snap/lxd/common/lxd/storage-pools/default/containers/alice1/rootfs`**
- To copy files from local system to remote system:
  `scp <filename> root@<destination-ip>:~/`
  To copy files from remote system to local system
  `scp root@<source-ip>:~/<filename> .`

- **To create a new container:**
  lxc init -p default-profile ubuntu:20.04 <CONTAINER_NAME>
- **To start container:**
  lxc start <CONTAINER_NAME>

  Note: You can configure it to autostart using this command:
  lxc config set <CONTAINER_NAME> boot.autostart true