REPORT

Anshul Sangrame (CS21BTECH11004)
Cheekatla Hema Sri(CS21BTECH11013)
Rajiv Shailesh Chitale(CS21BTECH11051)

Details of Application

- **Keys and Certificate generation:**
  - **Root**
    - Creation of root CA certificate(Subject name: iTS Root R1, V3 X.509 certificate, self-signed using 512-bit ECC Private Key of the root)

    - Execute the following command for generating the private key for root.
      ```
      openssl ecparam -name prime256v1 -genkey -noout -out root-key.pem
      ```

    - Execute the following command for generating a self-signed CA certificate of the root with given details,
      ```
      openssl req -new -x509 -key root-key.pem -out root-cert.crt -days
      3650 \
          -subj "/C=IN/ST=Telangana/O=iTS Root/OU=iTS/CN=iTS Root R1" \
          -extensions v3_ca \
          -config <(cat <<-EOF
      [v3_ca]
      subjectKeyIdentifier=hash
      basicConstraints=critical,CA:true
      keyUsage=critical,digitalSignature,cRLSign,keyCertSign
      EOF
      )
      ```
  - **Intermediate**
    - Generation of an intermediate CA certificate (Subject Name: iTS CA 1R3, V3 X.509 certificate with 4096-bit RSA public key, signed by iTS Root R1),

    - Generate 4096-bit RSA key via following command:
      ```
      openssl genrsa -out inter-key.pem 4096
      ```

    - Generate certificate signing request for intermediate CA via following command:
      ```
      openssl req -new -key inter-key.pem -out inter-csr.csr \
      -subj "/C=IN/ST=Telangana/O=iTS CA/OU=iTS/CN=iTS CA 1R3" \
      -addext "subjectKeyIdentifier=hash" \
      -addext "extendedKeyUsage=serverAuth,clientAuth" \
      -addext "basicConstraints=critical,CA:true,pathlen:0" \
      -addext "keyUsage=critical,digitalSignature,cRLSign,keyCertSign"
      ```

- Sign the CSR with root CA certificate with the following command:

```
openssl x509 -req -in inter-csr.csr -CA root-cert.crt -CAkey root-key.pem
-out inter-cert.crt -days 365 \
-extfile <(echo -e "extendedKeyUsage = serverAuth,
clientAuth\nbasicConstraints = critical, CA:true, pathlen:0\nkeyUsage =
critical, digitalSignature, cRLSign, keyCertSign")
```

- **Alice:**
  - Generation of certificate of Alice (Subject Name: Alice1.com with 1024-bit RSA public key, issued i.e., signed by the intermediate CA, iTS CA 1R3)

  - Generate 1024-bit RSA key for alice via following command

```
openssl genrsa -out alice-key.pem 2048
```

  - Generate certificate signing request for certificate of alice via following command

```
openssl req -new -key alice-key.pem -out alice-csr.csr \
-subj "/C=IN/ST=Telangana/O=Alice/OU=Alice/CN=Alice1.com" \
-addext "nsCertType=client"\
-addext "subjectKeyIdentifier=hash" \
-addext "extendedKeyUsage=clientAuth" \
-addext "basicConstraints=CA:FALSE" \
-addext "keyUsage=critical,nonRepudiation,digitalSignature,keyEncipherment"
```

  - Sign the CSR of alice with intermediate CA certificate generated via following command:

```
openssl x509 -req -in alice-csr.csr -CA inter-cert.crt -CAkey inter-key.pem
-out alice-cert.crt -days 365     -extfile <(echo -e
"extendedKeyUsage=clientAuth\nbasicConstraints=CA:FALSE\nkeyUsage=critical,
nonRepudiation,digitalSignature,keyEncipherment\nsubjectKeyIdentifier=hash\
nnsCertType=client")
```

    [**Note:** Since the current openssl libraries used do not allow certificates signed by 1024-bit RSA key, we have used alice certificate generated by 2048-bit RSA key]

- **Bob**
  - Generation of certificate of Bob (Subject Name: Bob1.com with 256-bit ECC public key, issued i.e., signed by the intermediate CA, iTS CA 1R3)

  - Generate ECC key for bob via following command:

```
openssl ecparam -name prime256v1 -genkey -noout -out bob-key.pem
```

  - Generate certificate request for certificate of bob via following command

```
openssl req -new -key bob-key.pem -out bob-csr.csr \
    -subj "/C=IN/ST=Telangana/O=Bob/OU=Bob/CN=Bob1.com" \
    -addext "nsCertType=client"\
    -addext "subjectKeyIdentifier=hash" \
    -addext "extendedKeyUsage=clientAuth" \
```

```
    -addext "basicConstraints=CA:FALSE" \
    -addext
"keyUsage=critical,nonRepudiation,digitalSignature,keyEncipherment"
```

- Sign the CSR of bob with intermediate CA certificate generated via following command:

```
openssl x509 -req -in bob-csr.csr -CA inter-cert.crt -CAkey inter-key.pem
-out bob-cert.crt -days 365     -extfile <(echo -e
"extendedKeyUsage=clientAuth\nbasicConstraints=CA:FALSE\nkeyUsage=critical,
nonRepudiation,digitalSignature,keyEncipherment\nsubjectKeyIdentifier=hash\
nnsCertType=client")
```

- **Secure chat application:**
  - **Application data protocol:**

    Format:

    #----------#
    | type |
    | time |
    #----------#
    | Body(variable size) |
    #----------#

    The flow of messages:

    There are two types of messages: control and data. The message will have a fixed-size header (with two fields type and time) and body.
    The client starts by sending CHAT_HELLO, and the server responds with CHAT_OK_REPLY. After that client sends CHAT_START_SSL to the server to start SSL with which the server either responds with CHAT_START_SSL_ACK(to start SSL) or CHAT_START_SSL_NOT_SUPPORTED (if the server does not support SSL). If the client does not support SSL, he can send CHAT_NO_SSL instead of CHAT_START_SSL to which the server responds CHAT_NO_SSL_ACK. After that client and server continue with a DTLS connection if they agree upon START_SSL or continue with UDP without DTLS if they agree upon NO_SSL.

  - **Client-side UDP connection:**

```
void client_connection::establish_conn()
{
    struct sockaddr_in to_addr;
    to_addr.sin_family = AF_INET;
    to_addr.sin_port = htons(port);
    hostent *lh;
    if ((lh = gethostbyname(to_name.c_str())) == NULL)
    {
        herror("gethostbyname");
        throw runtime_error("");
    }
    to_addr.sin_addr.s_addr = *(in_addr_t *)lh->h_addr_list[0];
    if (connect(sockfd, (struct sockaddr *)&to_addr, sizeof(to_addr)) < 0)
    {
        throw runtime_error(string("socket connection failed: ") + strerror(errno));
    }
    message chat_hello = {
        .hdr = {
            .type = CONTROL,
            .time = time(NULL)
        },
        .body = "CHAT_HELLO"
    };
    send_msg(chat_hello);
    message reply = read_msg();
    if (reply.hdr.type != CONTROL || reply.body != "CHAT_OK_REPLY")
    {
        throw runtime_error(string("socket connection failed: invalid CHAT_OK_REPLY"));
    }
}
```

The above code establishes a UDP connection for the client. It gets the address of the server by hostname then sends a CHAT_HELLO control message and receives a CHAT_OK_REPLY message

- **Server-side UDP connection:**

```
void server_connection::establish_conn()
{
    struct sockaddr_in to_addr;
    char buff[BUFF_SIZE];
    socklen_t len = sizeof(to_addr);
    int n;

    if ((n = recvfrom(sockfd, buff, BUFF_SIZE, 0, (struct sockaddr *)&to_addr, &len)) < 0)
    {
        throw runtime_error(string("socket connection failed: ") + strerror(errno));
    }

    struct hostent *host = gethostbyaddr(&to_addr.sin_addr, sizeof(to_addr.sin_addr), AF_INET);
    if (host == NULL) {
        herror("gethostbyaddr");
        throw runtime_error("");
    }
    to_name = host->h_name;

    message msg = construct_message(string(buff,n));

    if (msg.hdr.type != CONTROL || msg.body != "CHAT_HELLO")
    {
        throw runtime_error(string("socket connection failed: invalid CHAT_HELLO"));
    }

    if (connect(sockfd, (struct sockaddr *)&to_addr, sizeof(to_addr)) < 0)
    {
        throw runtime_error(string("socket connection failed: ") + strerror(errno));
    }

    message reply = {
        .hdr = {
            .type = CONTROL,
            .time = time(NULL)
        },
        .body = "CHAT_OK_REPLY"
    };
    send_msg(reply);
}
```

The above code establishes a UDP connection for the server. It receives CHAT_HELLO and gets the hostname of the client by its received IP address then sends a CHAT_OK_REPLY control message back.

- **DTLS connection**

```
void connection::prepare_ctx()
{
    ctx = SSL_CTX_new(DTLS_method());
    SSL_CTX_set_min_proto_version(ctx, DTLS1_2_VERSION);

    const char *cipher_list = "ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA

    if (SSL_CTX_set_cipher_list(ctx, cipher_list) == 0)
    {
        ERR_print_errors_fp(stderr);
        throw runtime_error("Unable to set cipher suites");
    }

    if (SSL_CTX_use_certificate_file(ctx, CERT_LOC, SSL_FILETYPE_PEM) <= 0)
    {
        ERR_print_errors_fp(stderr);
        throw runtime_error("Can't load certificate");
    }

    if (SSL_CTX_use_PrivateKey_file(ctx, PRIVATE_KEY_LOC, SSL_FILETYPE_PEM) <= 0)
    {
        ERR_print_errors_fp(stderr);
        throw runtime_error("Can't load private key");
    }

    if (!SSL_CTX_load_verify_locations(ctx, CERT_CHAIN_LOC, NULL))
    {
        ERR_print_errors_fp(stderr);
        throw runtime_error("Error in loading certificate");
    }

    SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, verify_callback);
    SSL_CTX_set_verify_depth(ctx, 4);

    // SSL_CTX_set_session_ticket_cb(ctx, NULL, , NULL);
}
```

The given code would set the context for the TLS connection. The function written in the below code snippet is ‘prepare_ctx()’, which describes the context of the TLS connection that is being established. First the protocol is set to DTLS 1.2. Then some set of cipher suites that the user supports are set using ‘SSL_CTX_set_cipher_list’ function. Then the certificates that are used for handshake and the trusted certificates are loaded in this function. Finally, we set a certificate verification callback. This function totally describes the establishment of DTLS connection. The context preparation of server and client are similar. They both load and set their corresponding certificates and cipher suites and callback functions.

Finally to establish the DTLS connection between server and client, client calls ‘SSL_connect’ function and server calls ‘SSL_accept’ function. On successful execution, the DTLS handshake would be successful and the further messages could be sent via TLS encryption.

Details of MITM attacks

- **SSL Downgrade attack for eavesdropping**
  In this section, trudy performs downgrade attack by intercepting start_SSL messages, such that both alice and bob would not establish TLS connection, resulting in alice and bob having unsecure chat communication. Then Trudy can successfully intercept the connection between Alice and bob.

  For this we execute the **poison-dns-alice1-bob1.sh** This would poison the arp cache, such that whenever alice/bob try to connect to other, the would be connected to trudy.

```
void server_connection::startSSL()
{
    message msg = read_msg();
    if (msg.hdr.type == CONTROL && msg.body == "CHAT_NO_SSL")
    {
        message reply = {
            .hdr = {
                .type = CONTROL,
                .time = time(NULL)},
            .body = "CHAT_NO_SSL_ACK"};
        send_msg(reply);
        return;
    }
    if (msg.hdr.type != CONTROL || msg.body != "CHAT_START_SSL")
    {
        throw runtime_error(string("SSL socket connection failed: invalid start_ssl"));
    }
    message reply = {
        .hdr = {
            .type = CONTROL,
            .time = time(NULL)},
        .body = "CHAT_START_SSL_NOT_SUPPORTED"};
    send_msg(reply);
}
```

  Here is how Trudy intercepts the client by pretending to be a server. As server, trudy whenever receives CHAT_NO_SSL, it acknowledges it and whenever receives CHAT_START_SSL it responds with CHAT_START_SSL_NOT_SUPPORTED. Hence the client would not start an SSL handshake and continue the communication with the UDP connection itself.

```
void client_connection::startSSL()
{
    message start_ssl = {
        .hdr = {
            .type = CONTROL,
            .time = time(NULL)
        },
        .body = "CHAT_NO_SSL"
    };
    send_msg(start_ssl);
    message reply = read_msg();
    if (reply.hdr.type != CONTROL)
    {
        throw runtime_error(string("SSL socket connection failed: invalid response to start_ssl"));
    }
    if (reply.body == "CHAT_NO_SSL_ACK") return;
    throw runtime_error(string("SSL socket connection failed: invalid response to no_ssl"));
}
```

While pretending as a client to connect to the server, Trudy simply sends CHAT_NO_SSL to the server, so that server would acknowledge it with CHAT_NO_SSL_ACK. And then sends CHAT_START_SSL_NOT_SUPPORTED to the client whenever the client sends CHAT_START_SSL. Hence, trudy intercepts the communication to make client and server agree to unsecure communication.

- **Active MITM attack:**
  In this attack, Trudy establishes two DTLS sessions one with the client and one with the server. Here Trudy signs fake client and server certificates using an intermediate CA certificate since we assume that Trudy has hacked an intermediate CA server.

Credit statement:

- Ch Hema Sri:
  - Generated keys, CSRs, and certificates for task 1 and task4
  - Worked on DTLS connection part (‘prepare_ctx(): configuring SSL_CTX object’)
  - Fixed certificate verification error due to different EC parameters
  - Documentation for task1 (readme)
  - Demonstration video for task 2

    - Task 1, passive attack, DTLS connection report
- Anshul Sangrame:
    - Worked on establishing UDP connection.
    - Worked on sending and receiving data in bytes and converting it to message format
    - Worked on DTLS connection(Converting UDP socket to DTLS socket and performing read and write from that socket)
    - Integrated TUI(Text User Interface) handler with DTLS connection by designing class connection which is used in TUI handler
    - Demonstration video for task 3
    - Send ARP gratuitous message in Task 5
- Rajiv:
    - Made handler for passive and active MITM
    - Made TUI(Text User Interface) using ncurses
    - Worked on Application layer protocol
    - Worked on gracefully ending the DTLS connection
    - Demonstration video for task 4
    - IP spoofing in Task 5

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: Anshul Sangrame, Cheekatla Hema Sri, Rajiv Shailesh Chitale
Date: 10/04/2024
Signature: Anshul, CHS, R.S.C