ECE568 Lab #2: OpenSSL Programming

Introduction

To complete this lab, you will have to learn how to use the OpenSSL library, which is installed on ECF. You are provided with code for a simple client and a simple server application, with an accompanying Makefile. You are also provided with a set of key files that will be used for authentication purposes.

Lab 2 may be done individually, or in groups of two. Your assignment needs to work on the ECF lab machines (this is where it will be graded). It will be due at 11:59pm on Nov 2, 2020.

Overview of files provided

server.c

The server takes as input a port (defaults to 8765). It then binds to this port and waits for connections. Upon connection of a client, a child is forked to handle the session. The server receives a message, whose length is at most 255 characters, prints this message to `stdout`, and then sends an answer to the client.

client.c

The client takes as input a host and a port (defaults to `localhost` and 8765). It connects to that host, sends a message, then reads from the socket. Upon receipt of a message from the server, of length at most 255 characters, it displays the message on `stdout` and exits.

Key files

Three key files are provided: `568ca.pem`, `alice.pem` and `bob.pem` (x509 certificates). `568ca.pem` contains the public key of the certificate authority (CA) ece568, which can be used to verify the authenticity of both the client and the server's public keys. The password (key phrase) for the certificates (both Alice's and Bob's) is "`password`". Bob is running the server, and Alice is running the client.

You will need to generate additional key pairs and certificates for testing. This can be done by using the openssl command line tool. The example below will give you an idea of how to use the command line tool. For more help, please refer to the resources section.

Creating a root certificate

```
openssl req -new -x509 -extensions v3_ca -keyout <ca_private_key.pem>
-out <certificate.pem> -days 365 -config <openssl_config_file>
```

where:

- `ca_private_key.pem`: Certificate Authority's private key.
- `certificate.pem`: Certificate Authority's certificate.
- `openssl_config_file`: See resources for a configuration file example.

Creating a signing request and public/private key pair

```
openssl req -new -nodes -out <name-req.pem> -keyout
<name-private.pem> -days 365 -config <openssl_config_file>
```

where:

- name-req.pem: signing request
- name-private.pem: private key

Signing a public key

```
openssl ca -out <name-cert.pem> -days 365 -config
<openssl_config_file> -infiles <name-req.pem>
```

Viewing a certificate

```
openssl x509 -in certificate.pem -noout -text
```

Your Task

You have to secure the communication channel between the client and the server through the use of SSL by modifying the client and the server. Your modified client and server will send messages over the encrypted channel that you set up with SSL. The behavior of your client and server must conform to the following specifications. Note that anything appearing in the `< >` brackets should be verbatim from the certificate or network message. In other words something like `<server Common Name>` should contain only "`Bob's Server`" and nothing else. To keep things simple, your client or server may exit on error, after you have printed the error string.

Client Specification

Every message the client prints should be preceded with the string `ECE568-CLIENT`. It may print other messages for debugging, but they will be ignored for marking purposes. Your client must do the following:

1. Only communicate with servers using SSLv3 or TLSv1. If the server does not support one of those protocols, then print:

```
ECE568-CLIENT: SSL connect error
```

2. Only communicate with a protocol that uses the SHA1 hash function. If server does not use SHA1, then print:

```
ECE568-CLIENT: SSL connect error
<processPID>:error:14077410:SSL
routines:SSL23_GET_SERVER_HELLO:sslv3 alert handshake
failure:s23_clnt.c:744:
```

Note the 2nd part of the message can be obtained from the SSL error routines `ERR_print_errors` or `ERROR_print_errors_fp`.

3. Only communicate with Bob's Server by checking that the Common Name (CN) of the server matches "`Bob's Server`", and that the e-mail address of the server certificate subject matches "ece568bob@ecf.utoronto.ca". The client must verify that the server's certificate has a valid signature from the CA.

- If both CN and Email are correct, then print the server information, request sent, and response returned by the server:

```
ECE568-CLIENT: <server CN> <server email> <certificate issuer>
ECE568-CLIENT: <client request> <server response>
```

- If one of them is not correct, indicate the mismatched field, and do not output any server information:

```
ECE568-CLIENT: Server Common Name doesn't match
```

or

```
ECE568-CLIENT: Server Email doesn't match
```

- If the certificate is not valid, then print:

```
ECE568-CLIENT: Certificate does not verify
```

4. The client should shutdown the SSL connection correctly.

5. The client should report if the server does not shutdown correctly. If this happens, it should print:

```
ECE568-CLIENT: Premature close
```

Server Specification

Every message the server prints should be preceded with the string `ECE568-SERVER`. Your server must do the following:

1. The server must support SSLv2, SSLv3 and TLSv1.

2. The server must support all cipher suites available for SSLv2, SSLv3 and TLSv1.

3. The server should only communicate with clients with a valid certificate signed by the CA:

- If the client certificate is not signed by the proper CA the server should print:
  ```
  ECE568-SERVER: SSL accept error
  <processPID>:error:140890B2:SSL routines:SSL3_GET_CLIENT_CERTIFICATE: no certificate returned:s3_srvr.c:3329:
  ```
- If the client does not present a certificate then the server should print:
  ```
  ECE568-SERVER: SSL accept error
  <processPID>:error:140890C7:SSL routines:SSL3_GET_CLIENT_CERTIFICATE: peer did not return a certificate:s3_srvr.c:3312:
  ```

Note the 2nd part of the message can be obtained from the SSL error routines `ERR_print_errors` or `ERROR_print_errors_fp`.

4. If the client has a valid certificate, the server should print the CN and Email of the client, as well as the client request and server response:

```
ECE568-SERVER: <client CN> <client email>
ECE568-SERVER: <client request> <server response>
```

5. The server should shutdown the connection properly.

6. The server should report of the client does not shutdown correctly. If this happens, it should print:

```
ECE568-SERVER: Incomplete shutdown
```

In addition, please explain what you did, in at most **300 words**. These explanations must be in the file `explanations_lab2.txt` along with the identities of your group members just as in previous labs.

Resources

The following resources will be very useful. Please take the time to read them.

1. The OpenSSL man pages on ECF (i.e. `man ssl`). The following man pages will be useful (you may find others helpful as well):

- `ssl`
- `x509`
- `ciphers`
- `BIO_new_socket`
- `SSL_connect`
- `SSL_read`
- `SSL_write`
- `SSL_shutdown`
- `SSL_CTX_set_verify`
- `SSL_get_verify_result`
- `SSL_CTX_set_cipher_list`
- `openssl/objects.h`
- `SSL_CTX_new`
- `SSL_CTX_set_options`

2. The openssl utility on ECF can be used to verify certificates, as well as generate your public/private key pairs and certificates for testing.

3. Certificate generation and signing tutorial

https://www.digitalocean.com/community/tutorials/openssl-essentials-working-with-ssl-certificates-private-keys-and-csrs

https://www.keycdn.com/blog/openssl-tutorial

http://www.flatmtn.com/article/setting-openssl-create-certificates.html

http://www.flatmtn.com/article/creating-pkcs12-certificates.html

4. Read an introduction to OpenSSL programming:

http://www.linuxjournal.com/article/4822
http://www.linuxjournal.com/article/5487

Submission and Marking

Please explain in at most **300 words**, what you did to get the lab working. These

explanations must be in the file `explanations_lab2.txt`, which must also contain the names and student numbers of your group members prefixed by # and separated by commas. Do not prefix other lines by # as this would confuse the automated marking scripts.

```
#first1 last1, studentnum1, e-mail address1
#first2 last2, studentnum2, e-mail address2
```

It is very important that the information you submit is correct as your mark will be assigned by student number, and the e-mail you give here will be used to get the results of the lab back to you. This file should be placed in a single directory along with the code for your client and server, the Makefile needed to compile code, and any other files needed to build your client and server. Marking will be partly automated. **You should use the `FMT xxx` strings provided in the source code to output messages to the marker and you should also NOT alter what the strings `secret` and `answer` are initialized to.**

Before submitting, you should test that your submission is formatted properly with the commands `/share/copy/ece568f/bin/check568_lab2`. Go to the lab directory and type this command. Do not submit until the script indicates that the submission appears to be properly formatted. In addition, check that the script correctly identifies your group members.

To submit your work, run the command `/share/copy/ece568f/bin/submit568_lab2`. The command tests that your submission is properly formatted and identifies the group members based on your txt file. Check that the information reported by the script is correct. If your submission is properly formatted, the command will then ask you to confirm that you want to submit. Note that you can submit multiple times. Your last submission will be marked. The submission command will cease to work after the lab is due.