Computer and Network Security
Project

Jeroen Famaey and Esteban Municio- University of Antwerp
(esteban.municio@uantwerpen.be - M.G.325)

2016 – 2017

Name student:

1 Introduction

The project is an assignment which aggregates most of the topics viewed in the course and has the goal to give a broad overview of the common security techniques applied in current systems. This includes symmetric cipher, asymmetric cipher, hash algorithms and trust mechanisms.

The aim of this assignment is to implement in Python your own version of SSL and a data sharing application on top of it. SSL (Secure Socket Layer) is a very known protocol that provide efficient data encryption and authentication between applications at socket level. That means that it can provide secure communication in scenarios where the data is sent across an insecure network. There are a number of configurations available for SSL and also different SSL versions. In order to make it easier, we ask you to implement a specific "home-made" SSL on top of TCP in which you can apply the code you generated in Lab sessions 1, 2, 3, and 4 as much as you can. Over your SSL implementation you have to build a small basic Python application that send secure data, in text and image format. At the Application layer, you are free to implement whatever you want, so you can choose the user interface (file system, graphical, web based or command line) and the protocol over SSL (you have to design how the data is processed over the SSL connection). However the minimum set of requirements for this application is stated next in the Use Cases chapter. Both your SSL implementation and your application also has to be written in Python. However for your application, you can make use of as many external components as you want from other languages (i.e. scripts, databases, web interfaces, etc.) always that no external installations are required (in case of using web interfaces, applications that use Apache are allowed).

Your SSL implementation will be based on a Client-Server scheme, and you will be able to test on your own laptop using the *localhost* address. However you can test your SSL implementation with other students by using the actual IP address. Aplications on top of SSL may also tested as far as you ensure compatibility between the different implementations.

2 Specification

Your SSL implementation will be based in the following scheme:

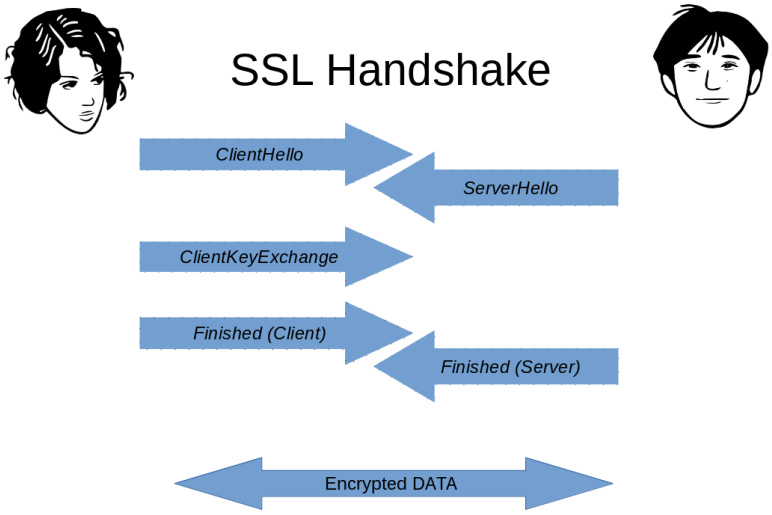

Figure 1: SSL Handshake example

You can encode the messages as you want as far as they contain the following information:

ClientHello This message is the first message sent by the client in order to initiate an SSL connection:

1. Message ID (1 byte). Is the identifier of the message. The value for ClientHello is 0x01. Other than this value will involve the server send an ErrorMessage with the Unexpected Message error.

2. SSL Version (1 byte). In order to show that your SSL is not standard you will use the value 0x64. The version has to be checked in the server, if the value does not match, an ErrorMessage with the Unsupported Field error will be sent by the server.

3. Protocol Version (1 byte). In order to show that the protocol over SSL is not standard you will use the value 0x65. The version has to be checked in the server. If the value does not match, an ErrorMessage with the Unsupported Field error will be sent by the server.

4. Client-random (32 bytes). A random string of 32 bytes that is generated in the client with a random number generator.

5. Ciphersuit (2 bytes). It will have the value of 0x002F, which correspond to symmetric encryption AES-128-CBC, asymmetric encryption RSA, and hash function SHA1 (SSL_RSA_WITH_AES_128_CBC_SHA). The value has to be checked in the server, if the value does not match, an ErrorMessage with Unsupported Field will be sent by the server.

6. EndOfMessage (2 bytes). The end of the message will be communicated using the value 0xF0F0. If other bytes than 0xF0F0 are detected here the connection will be reset with an Unsupported structure error.

ServerHello This message is the server response to the client.

1. Message ID (1 byte). Is the identifier of the message. The value for ServerHello is 0x02. Other than this value will involve the client send an ErrorMessage with the Unexpected Message error.

2. SSL Version (1 byte). Value 0x64. The version has to be checked in the client. If the value does not match, an ErrorMessage with the Unsupported Field error will be sent by the client.

3. Protocol Version (1 byte). Value 0x65. The version has to be checked. If the value does not match, an ErrorMessage with the Unsupported Field error will be sent by the client.

4. Server-random (32 bytes). A random string of 32 bytes that is generated in the server with a random number generator.

5. Session ID (2 bytes). The identifier of the session can have a value from 0 to 65535. If the maximum number is reached, the count start in 0 again. The value is stored in a variable in the server.

6. Ciphersuit (2 bytes). It will have the value of 0x002F, which according to the RFC correspond to symmetric encryption AES-128-CBC, asymmetric encryption RSA, and hash function SHA1 (SSL_RSA_WITH_AES_128_CBC_SHA). The value has to be checked in the client. If the value does not match, an ErrorMessage with the Unsupported Field error will be sent by the client.

7. Client Certificate request (1 byte). It is a flag that means that a client certificate is required. The default value is 0x01

8. Server Certificate (1203 bytes). The server's certificate has to be sent to the client in order to allow the client to check the authenticity of the server. The server certificate (client-04.pem) is available in Blackboard for this assignment. The client has to check that the certificate is valid, by looking at the expiration date and the Certification authority (it has to be issued by *orion*). Furthermore, this certificate will allow the client to know the public key of the server. In order to validate the certificate and

extract the public key from it, use your code from Lab Session 4. If the certificate is not valid, an ErrorMessage with Incorrect Server Certificate will be sent by the client.

9. EndOfMessage (2 bytes). The end of the message will be communicated using the value 0xF0F0. If other bytes than 0xF0F0 are detected here the connection will be reset with an Unsupported structure error.

ClientKeyExchange Once the server has finished sending all the information, the client has to provide the following information after verifying the Server Certificate:

1. Message ID (1 byte). Is the identifier of the message. The value for ClientKeyExchange is 0x03. Other than this value will involve the server send an ErrorMessage with the Unexpected Message error.

2. Session ID (2 bytes). The identifier of the session given by the server. The value has to be checked in the server, no lower values than the value given by the server are allowed. If the value does not match, an ErrorMessage with Bar Session ID will be sent by the server.

3. Client Certificate(1203 length). This certificate (client-05.pem) is available in Blackboard for this assignment. The server will check the validity of the certificate looking at the expiration date and the certification authority (it also has to be issued by *orion*). In order to validate the certificate, use your code from Lab Session 4. If the certificate is not valid, an ErrorMessage with Incorrect Client Certificate will be sent by the server.

4. Pre-master key length (2 bytes). Is the length of the following encrypted pre-master key. The length of the ClientKeyExchange will be checked considering this value.

5. Pre-master key (Variable size). It is a random 48 byte string encrypted with the server's public key. Use your code from Lab Session 2 to encrypt this string with the server's public key. This pre-master key will be used, together with the Client-random and Server-random to generate the master secret used for the AES encryption.

6. Login password (20 bytes). It is the SHA1 hash of the client's login password. For perform this operation, use the your code from Lab Session 3 for performing the SHA1 function over the login password. The server will check that this user is allowed to use the service given by the server, by comparing its public key, user ID and login password. Suppose that the user has already register previously in the server and these values are stored in the server. The login password for the client is "Konklave-123" and the user ID is the CN value of the client's certificate (client-05.pem). If the login is unsuccessful, an ErrorMessage with Incorrect Login will be sent by the client.

7. EndOfMessage (2 bytes). The end of the message will be communicated using the value 0xF0F0. If other bytes than 0xF0F0 are detected here the connection will be reset. If other bytes than 0xF0F0 are detected here the connection will be reset with an Unsupported structure error.

Once the Server has received this information, both ends can generate their own the master key. First the server has to decrypt the pre_master_secret with its private key. This private key (server_prvkey.key), available in Blackboard for this assignment, is stored secretly in the server and only the server has access to it. Using your code from Lab Session 4 you will be able to extract the main information from the server's private key. As before, use your code from Lab Session 2 to decrypt the pre_master_secret. Once the server has the pre_master_secret, the master secret is generated as follows:

$$\begin{aligned} master\_secret \quad = \quad & SHA(pre\_master\_secret + SHA('A' + pre\_master\_secret + Client\_random + Server\_random)) + \\ & SHA(pre\_master\_secret + SHA('BB' + pre\_master\_secret + Client\_random + Server\_random)) + \\ & SHA(pre\_master\_secret + SHA('CCC' + pre\_master\_secret + Client\_random + Server\_random)) \end{aligned}$$

For perform this operation, use the your code from Lab Session 3 for performing the SHA1 function over the different strings. For getting the master secret, the client also perform this operation at their side, since it has all required information already. Once the server and the client have performed this operation, they will have the same master secret without having to send it through the channel. This master secret will be valid until the end of the session and has a length of 20 bytes. Since you are using AES-128-CBC, remove the last 4 bytes in order to have a 16-byte password.

FinishedClient By giving the previous information the client is ready to start a secure connection, and switch to secure mode. This is communicated to the to server by sending a FinishedClient message:

1. Message ID (1 byte). Is the identifier of the message. The value for FinishedClient is 0x04. Other than this value will involve the server send an ErrorMessage with the Unexpected Message error.

2. Session ID (2 bytes). The identifier of the session given by the server. The value has to be checked in the server, no lower values than the value given by the server are allowed. If the value does not match, an ErrorMessage with Bar Session ID will be sent by the server.

3. State (1 byte). The value will be 0x00 when the everything has gone without errors

4. EndOfMessage (2 bytes). The end of the message will be communicated using the value 0xF0F0. If other bytes than 0xF0F0 are detected here the connection will be reset with an Unsupported structure error.

FinishedServer When receiving the ClientKeyExchange message, the server has also all information needed for start a secure connection. After the client certificate is verified, the server switches to secure mode and and starts to encrypt symmetrically with AES all following messages in the session. For this task, use your code from Lab Session 1. Have in mind that you will have to include padding since not all messages will not be 16-byte multiple. Once the server sends the FinishedServer message encrypted will mean the server is ready for listen to encrypted connections. Only when this FinishedServer message is received in the other end, the client can switch to secure mode and start to send encrypted data over the secured socket. If the server can not decode the message, an ErrorMessage with Bad Encryption will be sent to the server and the connection will be reset.

1. Message ID (1 byte). Is the identifier of the message. The value for FinishedServer is 0x05. Other than this value will involve the client send an ErrorMessage with the Unexpected Message error.

2. Session ID (2 bytes). The identifier of the session. Has to be checked in the client. If the value does not match, an ErrorMessage with Bar Session ID will be sent by the client

3. State (1 byte). The value will be 0x00 when the everything has gone without errors

4. EndOfMessage (2 bytes). The end of the message will be communicated using the value 0xF0F0. If other bytes than 0xF0F0 are detected here the connection will be reset with an Unsupported structure error.

The length of the FinishedServer message is 5 bytes. However since it is encoded with AES, the actual length will be of 16 byte after adding the padding. So the server has to check if the length is indeed 16 bytes (or 24 bytes if they are encoded in base64)

3 Use Cases

In this project you will have to implement an application that uses SSL to send secure data from the users to the server. This data will be logged in the server and users will be allowed to read this information. There will be five use cases, or in other words, 5 buttons/commands/actions that the user is able to perform:

1. Establish SSL connection. Before any interaction with the server, a SSL connection has to be negotiated and established following the specifications mentioned above. Once the SSL connection is ready a number of operations can be performed.

2. Send a message to the server. The message will be stored in the server together with the user ID and the date as a entry. Every time a text is uploaded successfully an ACK message has to be sent. The client will not consider the data correctly sent until it receives an ACK successfully from the server.

3. Send an image to the server. As the message, the image will be stored in the server together with the user ID and the date as a entry. Every time an image is uploaded successfully an ACK message has to be sent. The client will not consider the data correctly sent until it receives an ACK successfully from the server.

4. Read content. Users will be able to retrieve the data (both text and images) from the server. All users will be allowed to read content from other users, so the user basically gets all entries stored in the server. The content will be displayed in temporal order, specifying the type of data (text of image) and displaying them one after another. In case of non-graphical user interfaces, the images will be listed as their name and extension.

5. Close SSL Connection. Once the user no longer wants to upload or read content, the user can close the SSL connection. The SSL connection can be closed simply by calling the function close() or shutdown() in the python socket.

On top of SSL, you are free to design the protocol messages for sending and retrieving data to and from the server.

4 Error management

Your application has to respond correctly to errors in both the set up and secure connection phase. For this reason you have to implement also the functions that will manage the possible errors. Regarding in which phase errors occur, this functions will behave differently:

Set up phase. During the set up phase client and server have to agree what kind of SSL connection they want to establish. In SSL always the client suggests a configuration, but it is the server who really decides which one has to be used. In this project, you will only use one configuration, so if any of the fields do not match to those specified above, the client or the server will send an ErrorMessage resetting the connection and stopping the SSL set up phase. Also the length of the messages has to be checked in every SSL Set-up message.

Secure Connection phase Once the SSL connection has been established, errors can occur, i.e. a packet has not been properly encrypted and the information is not legible. In this case the end that receives the wrong packet, has to ignore the content and send a ErrorMessage with the Bad Encryption error. In the Secure Connection phase, if errors persist after 10 tries, the receiver will reset the SSL connection. Bad Encryption errors can happen in both directions, sending data to the server and acknowledged data to the client. The length of the encrypted messages will be managed at the Application layer after decrypt the message.

The ErrorMessage's are all defined with the following structure:

1. Message ID (1 byte). Is the identifier of the message. The value for Error messages is 0x06

2. Session ID (2 bytes). The identifier of the session.

3. State (1 byte). The value will depend of the cause of the error. The considered errors are the following:

Unexpected Message Type	0x01
Unknown Message Type	0x02
Length error	0x03
Bad Session ID	0x04
Unsupported field	0x05
Unsupported structure	0x06
Incorrect Server Certificate	0x07
Incorrect Client Certificate	0x08
Incorrect Login	0x09
Bad Encryption	0x0A
Unknown Error	0x0B

4. EndOfMessage (2 bytes). The end of the message will be noticed by the value 0xF0F0.

5 Assignment

1. Implement an application that allows secure exchange of text and images between a client and a server by emulating the SSL protocol. Following the specification given above, this application will use asymmetric encryption and hash functions for the key exchange, certificates to ensure authenticity and symmetric encryption for sending the user data. Provide also at least a simple usable user interface. Reuse the code of the previous labs as much as you can. These include your AES, RSA and SHA implementations. Do not use the Python Openssl library. You have to code you own SSL handshake. However you are free to use the Python Crypto library (i.e., for AES encryption or for reading certificates)
2. There are many features that you have to design on your own. For the most important ones, give a simple explanation about why you have chosen this alternative and no other.
3. As you could have realized, this implementation is vulnerable to a Replay attack. That means that an external user can impersonate the genuine client and access the server without knowing the information stored in the server or the client. Implement this attack from a user that can only obtain the information sent through the channel. Neglect the MAC and IP layers.
4. Once you have successfully implemented the Replay attack, design and code in your SSL implementation a fix that can prevent this kind of attacks.
5. Is there any other attack for which your implementation could be still vulnerable. If yes, propose a fix in the design in order to avoid them. You don't need to implement these fixes. If not, explain why your implementation is not vulnerable and how it prevents possible attacks.

6 Submitting your solution

The deadline for submitting your solution is Thursday December 15th, 2016 23:59. To submit your solution do the following:

- Create a text file answering the questions stated above and explaining how the application works and what is the task of each file. Also include in this text file how the application has to be executed and installed (if required)
- Clean the code files and prepare other required files.
- Create a compressed file with everything. Name the compressed file **cns-project-name.zip** (where *"name"* is your UA user name) and upload it to Blackboard.