ELEC-C7420 - Basic principles in networking

Assignment 7: SSL/TLS

Introduction

The main target of this experience is for you to get familiar with the implementation of the encryption in the network in the transport layer, to this end we will observe the SSL/TLS (Secure Sockets Layer / Transport Layer Security) in action. In this sense we will use SSL/TLS to encrypt and make secure the TCP connections in one widely used protocol as is HTTP by implementing HTTPS: HTTP over SSL.

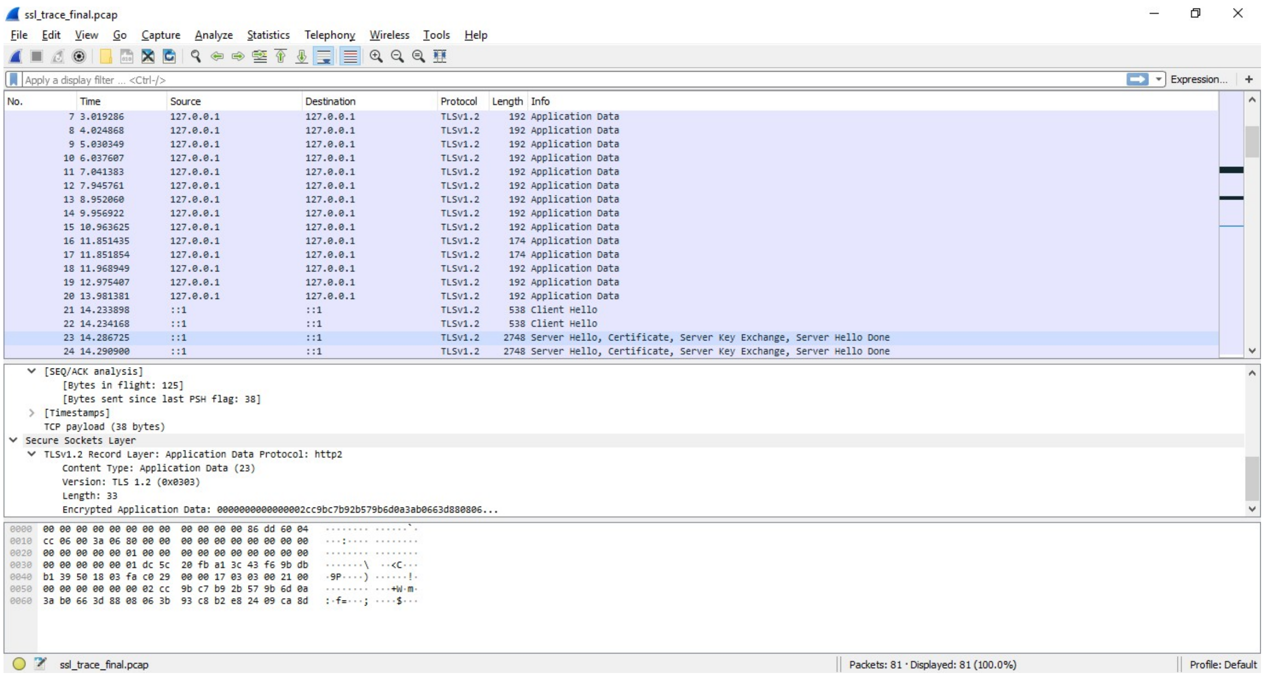
The main motivation to implement HTTPS over the webpages we access daily is to provide a secure means to access the information by employing authentication techniques of the accessed website, protecting the integrity and the privacy of the data we share with the web browsers, among other benefits. The implementation of HTTPS or SSL/TLS to secure the TCP connections during the HTTP connection, helps us to prevent:

- Man-in-the-middle attacks.
- Eavesdropping.
- Tampering the communication

The bi-directional encryption of communications between the client and the servers enables a secure way for us to ensure that we are on a secure communication channel with the website we are accessing without any means of interference from attackers or for being attacked by an impostor of the original website. This is why, historically the HTTPS implementation was primarily intended to be used for payment transactions over the World Wide Web, securing e-mail messages or to exchange sensitive information between corporations. However, in the recent years the HTTPS protocol has been extended to practically all the websites in the network that were not originally intended to use encryption like entertainment webpages, protecting the page authenticity on all type of websites, securing user accounts and keeping communications safe and private.

Step 1: open a trace

1) Download the trace from MyCourses and open it in your wireshark, in there you will be able to see the SSL messages as shown in the following figure:



```
No.  Time       Source     Destination  Protocol  Length  Info
   7 3.019286   127.0.0.1  127.0.0.1    TLSv1.2    192 Application Data
   8 4.024868   127.0.0.1  127.0.0.1    TLSv1.2    192 Application Data
   9 5.030349   127.0.0.1  127.0.0.1    TLSv1.2    192 Application Data
  10 6.037607   127.0.0.1  127.0.0.1    TLSv1.2    192 Application Data
  11 7.041383   127.0.0.1  127.0.0.1    TLSv1.2    192 Application Data
  12 7.945761   127.0.0.1  127.0.0.1    TLSv1.2    192 Application Data
  13 8.952060   127.0.0.1  127.0.0.1    TLSv1.2    192 Application Data
  14 9.956922   127.0.0.1  127.0.0.1    TLSv1.2    192 Application Data
  15 10.963625  127.0.0.1  127.0.0.1    TLSv1.2    192 Application Data
  16 11.851435  127.0.0.1  127.0.0.1    TLSv1.2    174 Application Data
  17 11.851854  127.0.0.1  127.0.0.1    TLSv1.2    174 Application Data
  18 11.968949  127.0.0.1  127.0.0.1    TLSv1.2    192 Application Data
  19 12.975407  127.0.0.1  127.0.0.1    TLSv1.2    192 Application Data
  20 13.981381  127.0.0.1  127.0.0.1    TLSv1.2    192 Application Data
  21 14.233898  ::1        ::1          TLSv1.2    538 Client Hello
  22 14.234168  ::1        ::1          TLSv1.2    538 Client Hello
  23 14.286725  ::1        ::1          TLSv1.2   2748 Server Hello, Certificate, Server Key Exchange, Server Hello Done
  24 14.290900  ::1        ::1          TLSv1.2   2748 Server Hello, Certificate, Server Key Exchange, Server Hello Done
```

```
    [SEQ/ACK analysis]
        [Bytes in flight: 125]
        [Bytes sent since last PSH flag: 38]
    [Timestamps]
    TCP payload (38 bytes)
Secure Sockets Layer
    TLSv1.2 Record Layer: Application Data Protocol: http2
        Content Type: Application Data (23)
        Version: TLS 1.2 (0x0303)
        Length: 33
        Encrypted Application Data: 0000000000000002cc9bc7b92b579b6d0a3ab0663d880806...
```

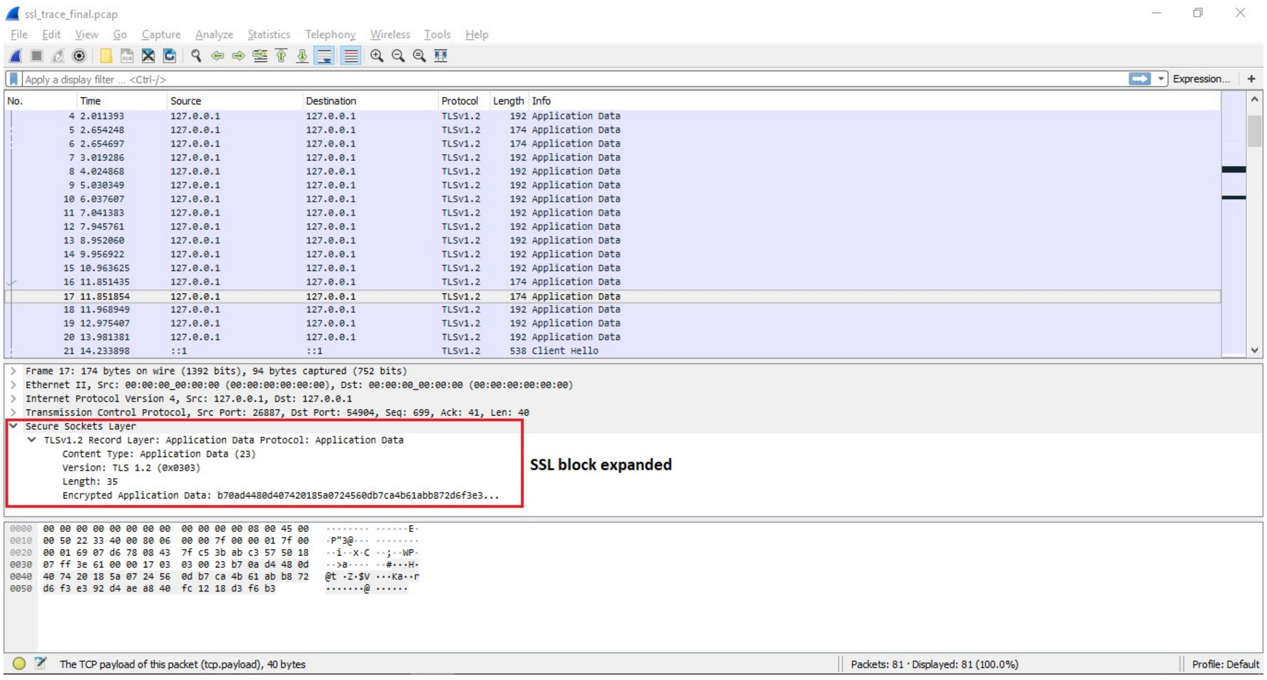
```
0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 04   ........ ......`.
0010  cc 06 00 3a 06 80 00 00  00 00 00 00 00 00 00 00   ...:.... ........
0020  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 01 dc 5c  20 fb a1 3c 43 f6 9b db   .......\  ..<C...
0040  b1 39 50 18 03 fa c0 29  00 00 17 03 03 00 21 00   .9P....) ......!.
0050  00 00 00 00 00 00 02 cc  9b c7 b9 2b 57 9b 6d 0a   ........ ...+W.m.
0060  3a b0 66 3d 88 08 06 3b  93 c8 b2 e8 24 09 ca 8d   :.f=...; ....$...
```

Figure. Trace of HTTPS traffic

Step 2: inspect the trace

2) In this case the trace is already filtered to show only the SSL packets, however, if you capture your own trace you may apply the "`ssl`" filter protocol in wireshark, this filter will help you to simplify the display by showing only SSL and TLS messages. It will exclude other TCP segments that are part of the trace, such as Acks and connection open/close.

3) Select a TLS message somewhere in the middle of your trace for which the Info reads "Application Data" & expand its Secure Sockets Layer block (by using the "+" expander or icon). For instance, we select the packet #17 and we expand this packet as shown in the figure below:


ssl_trace_final.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
SSL block expanded

```
No.  Time       Source     Destination  Protocol  Length Info
   4 2.011393   127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
   5 2.654248   127.0.0.1  127.0.0.1    TLSv1.2   174 Application Data
   6 2.654697   127.0.0.1  127.0.0.1    TLSv1.2   174 Application Data
   7 3.019286   127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
   8 4.024868   127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
   9 5.030349   127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
  10 6.037607   127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
  11 7.041383   127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
  12 7.945761   127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
  13 8.952060   127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
  14 9.956922   127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
  15 10.963625  127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
  16 11.851435  127.0.0.1  127.0.0.1    TLSv1.2   174 Application Data
  17 11.851854  127.0.0.1  127.0.0.1    TLSv1.2   174 Application Data
  18 11.968949  127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
  19 12.975407  127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
  20 13.981381  127.0.0.1  127.0.0.1    TLSv1.2   192 Application Data
  21 14.233898  ::1        ::1          TLSv1.2   538 Client Hello

> Frame 17: 174 bytes on wire (1392 bits), 94 bytes captured (752 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 26887, Dst Port: 54904, Seq: 699, Ack: 41, Len: 40
v Secure Sockets Layer
  v TLSv1.2 Record Layer: Application Data Protocol: Application Data
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 35
      Encrypted Application Data: b70ad4480d407420185a0724560db7ca4b61abb872d6f3e3...

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 50 22 33 40 00 80 06  00 00 7f 00 00 01 7f 00   ·P"3@··· ········
0020  00 01 69 07 d6 78 08 43  7f c5 3b ab c3 57 50 18   ··i··x·C ··;··WP·
0030  07 ff 3e 61 00 00 17 03  03 00 23 b7 0a d4 48 0d   ··>a···· ··#···H·
0040  40 74 20 18 5a 07 24 56  0d b7 ca 4b 61 ab b8 72   @t ·Z·$V ···Ka··r
0050  d6 f3 e3 92 d4 ae a8 40  fc 12 18 d3 f6 b3         ·······@ ······

The TCP payload of this packet (tcp.payload), 40 bytes    Packets: 81 · Displayed: 81 (100.0%)    Profile: Default
```

Figure. Example of SSL/TLS packet

If we observe the “Application Data” packet we can see that it is a generic TLS message carrying the encrypted information from the application it is encrypting for, in our case the contents of the webpage in HTTP.

As we know form the lectures, the lower layer protocol stack are TCP and IP because SSL/TLS runs on top of TCP/IP. The SSL layer contains a “TLS Record Layer”. This is the foundational sublayer for TLS. All messages contain records. Expand this block to see its content details:

- Content Type field: this indicates what is in the content of the corresponding record.
- Version identifier: It will be a constant value for the SSL connection, in our case we are working in the TLSv1.2.
- Length field: giving the length of the record.
- Application Data records are sent after SSL has secured the connection, so the contents will show up as encrypted data.

Step 3: SSL handshake

An important part of SSL is the initial handshake that establishes a secure connection. The handshake proceeds in several phases. There are slight differences for different versions of TLS and depending on the encryption scheme that is in use. The usual outline for a brand-new connection is:

a. Client (the browser) and Server (the web server) both send their Hellos.

b. Server sends its certificate to Client to authenticate (and optionally asks for Client Certificate).

c. Client sends keying information and signals a switch to encrypted data.

d. Server signals a switch to encrypted data.

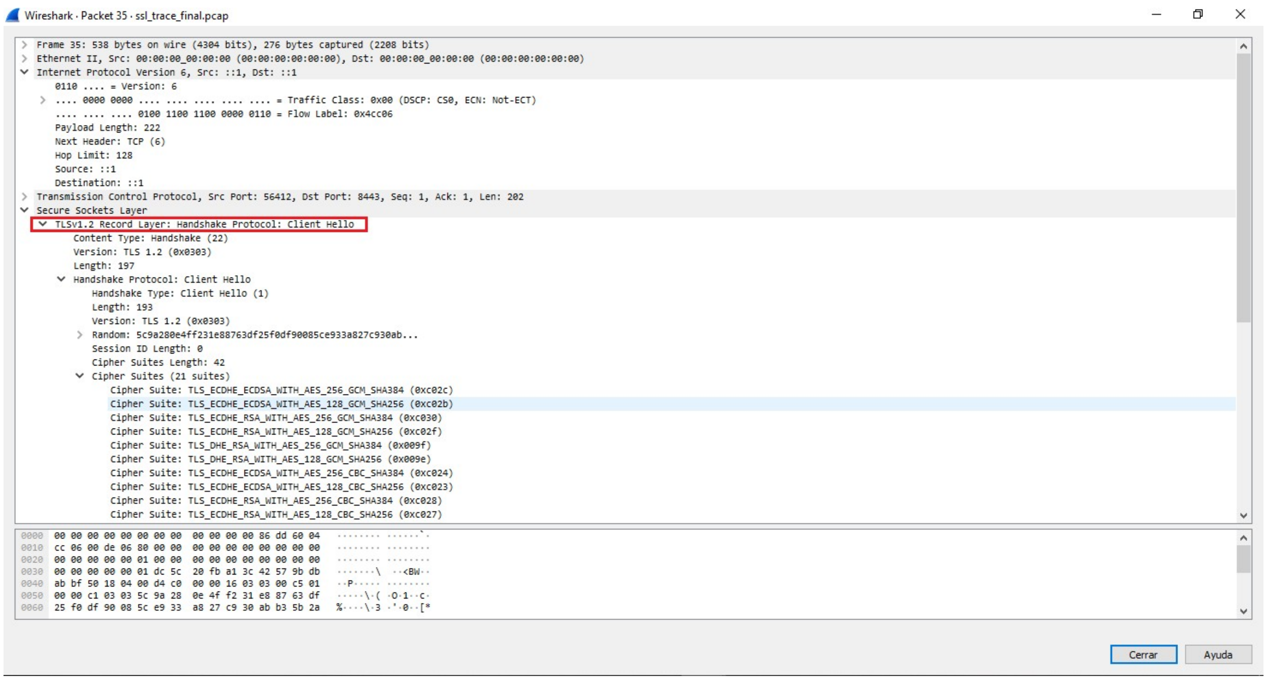
e. Both Client and Server send encrypted data.

f. An Alert is used to tell the other party that the connection is closing.

Note that there is also a mechanism to resume sessions for repeat connections between the same client and server to skip most of steps b and c.

2) The client sends a “Hello” packet to the server with a group of information that the server will use to establish the session. We can see several important fields here worth mentioning. First, the time (GMT seconds since midnight Jan 1, 1970) and random bytes (size 28) are included. This will be used later in the protocol to generate our symmetric encryption key. The client can send an optional session ID to quickly resume a previous TLS connection and skip portions of the TLS handshake. Arguably the most important part of the Client Hello message is the list of cipher suites, which dictate the key exchange algorithm, bulk encryption algorithm (with key length), MAC, and a pseudo-random function. The list should be ordered by client preference. The collection of these choices is a “cipher suite”, and the server is responsible for choosing a secure one it supports or return an error if it doesn’t support any. The final field specified in the specification is for compression methods. However, secure clients will advertise that they do not support compression (by passing “null” as the only algorithm) to avoid the CRIME

attack. Finally, the Client Hello can have a number of different extensions. A common one is server name, which specifies the host-name the connection is meant for, so webservers hosting multiple sites can present the correct certificate[1].



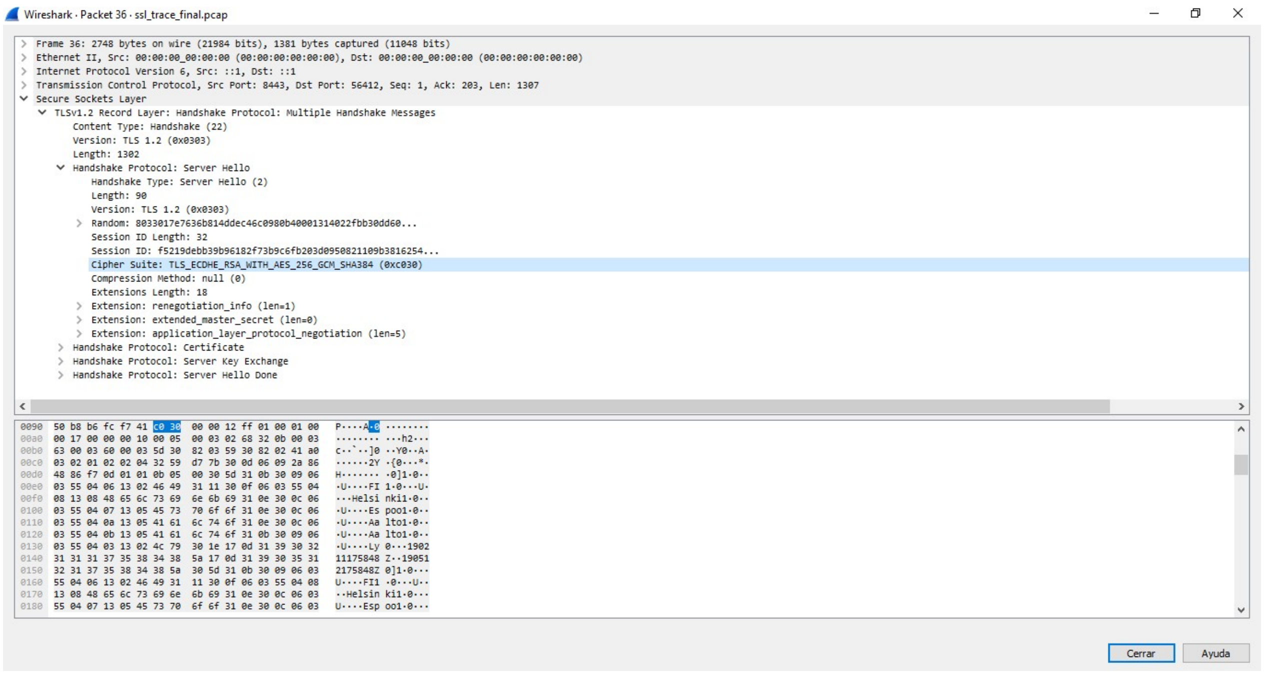
```
> Frame 35: 538 bytes on wire (4304 bits), 276 bytes captured (2208 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
v Internet Protocol Version 6, Src: ::1, Dst: ::1
    0110 .... = Version: 6
  > .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0100 1100 1100 0000 0110 = Flow Label: 0x4cc06
    Payload Length: 222
    Next Header: TCP (6)
    Hop Limit: 128
    Source: ::1
    Destination: ::1
> Transmission Control Protocol, Src Port: 56412, Dst Port: 8443, Seq: 1, Ack: 1, Len: 202
v Secure Sockets Layer
  v TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 197
    v Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 193
        Version: TLS 1.2 (0x0303)
      > Random: 5c9a280e4ff231e88763df25f0df90085ce933a827c930ab...
        Session ID Length: 0
        Cipher Suites Length: 42
      v Cipher Suites (21 suites)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
          Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x009f)
          Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x009e)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 04   ········ ······`·
0010  cc 06 00 de 06 80 00 00  00 00 00 00 00 00 00 00   ········ ········
0020  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ········ ········
0030  00 00 00 00 00 01 dc 5c  20 fb a1 3c 42 57 9b db   ·······\  ··<BW··
0040  ab bf 50 18 04 00 d4 c0  00 00 16 03 03 00 c5 01   ··P····· ········
0050  00 00 c1 03 03 5c 9a 28  0e 4f f2 31 e8 87 63 df   ·····\·( ·O·1··c·
0060  25 f0 df 90 08 5c e9 33  a8 27 c9 30 ab b3 5b 2a   %····\·3 ·'·0··[*
```

Figure. Client Hello packet

3) The server will send this message in response to a ClientHello message when it was able to find an acceptable set of algorithms. If it cannot find such a match, it will respond with a handshake failure alert. An important field in the ServerHello packet is the “session_id” field, this is the identity of the session corresponding to this connection. If the ClientHello.session_id was non-empty, the server will look in its session cache for a match. If a match is found and the server is willing to establish the new connection using the specified session state, the server will respond with the same value as was supplied by the client. This indicates a resumed session and dictates that the parties must proceed directly to the Finished messages. Otherwise, this field will contain a different value identifying the new session. The server may return an empty session_id to indicate that the session will not be cached and therefore cannot be resumed. If a session is

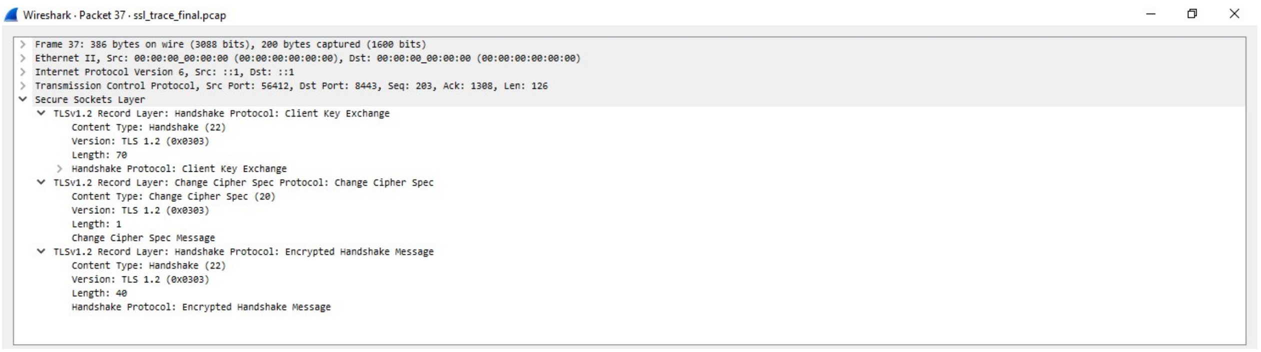
resumed, it must be resumed using the same cipher suite it was originally negotiated with. Note that there is no requirement that the server resume any session even if it had formerly provided a session_id. Clients MUST be prepared to do a full negotiation (including negotiating new cipher suites) during any handshake[2].



```
> Frame 36: 2748 bytes on wire (21984 bits), 1381 bytes captured (11048 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 8443, Dst Port: 56412, Seq: 1, Ack: 203, Len: 1307
v Secure Sockets Layer
  v TLSv1.2 Record Layer: Handshake Protocol: Multiple Handshake Messages
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 1302
    v Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 90
        Version: TLS 1.2 (0x0303)
      > Random: 8033017e7636b814ddec46c0980b40001314022fbb30dd60...
        Session ID Length: 32
        Session ID: f5219debb39b96182f73b9c6fb203d0950821109b3816254...
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
        Compression Method: null (0)
        Extensions Length: 18
      > Extension: renegotiation_info (len=1)
      > Extension: extended_master_secret (len=0)
      > Extension: application_layer_protocol_negotiation (len=5)
    > Handshake Protocol: Certificate
    > Handshake Protocol: Server Key Exchange
    > Handshake Protocol: Server Hello Done
```

```
0090  50 b8 b6 fc f7 41 c0 30  00 00 12 ff 01 00 01 00   P····A·0 ········
00a0  00 17 00 00 00 10 00 05  00 03 02 68 32 0b 00 03   ········ ···h2···
00b0  63 00 03 60 00 03 5d 30  82 03 59 30 82 02 41 a0   c··`··]0 ··Y0··A·
00c0  03 02 01 02 02 04 32 59  d7 7b 30 0d 06 09 2a 86   ······2Y ·{0···*·
00d0  48 86 f7 0d 01 01 0b 05  00 30 5d 31 0b 30 09 06   H······· ·0]1·0··
00e0  03 55 04 06 13 02 46 49  31 11 30 0f 06 03 55 04   ·U····FI 1·0···U·
00f0  08 13 08 48 65 6c 73 69  6e 6b 69 31 0e 30 0c 06   ···Helsi nki1·0··
0100  03 55 04 07 13 05 45 73  70 6f 6f 31 0e 30 0c 06   ·U····Es poo1·0··
0110  03 55 04 0a 13 05 41 61  6c 74 6f 31 0e 30 0c 06   ·U····Aa lto1·0··
0120  03 55 04 0b 13 05 41 61  6c 74 6f 31 0b 30 09 06   ·U····Aa lto1·0··
0130  03 55 04 03 13 02 4c 79  30 1e 17 0d 31 39 30 32   ·U····Ly 0···1902
0140  31 31 31 37 35 38 34 38  5a 17 0d 31 39 30 35 31   11175848 Z··19051
0150  32 31 37 35 38 34 38 5a  30 5d 31 0b 30 09 06 03   2175848Z 0]1·0···
0160  55 04 06 13 02 46 49 31  11 30 0f 06 03 55 04 08   U····FI1 ·0···U··
0170  13 08 48 65 6c 73 69 6e  6b 69 31 0e 30 0c 06 03   ··Helsin ki1·0···
0180  55 04 07 13 05 45 73 70  6f 6f 31 0e 30 0c 06 03   U····Esp oo1·0···
```

Figure. ServerHello message



```
> Frame 37: 386 bytes on wire (3088 bits), 200 bytes captured (1600 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 56412, Dst Port: 8443, Seq: 203, Ack: 1308, Len: 126
v Secure Sockets Layer
  v TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 70
    > Handshake Protocol: Client Key Exchange
  v TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
  v TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 40
      Handshake Protocol: Encrypted Handshake Message
```

Figure. Client Key Exchange message

SSL/TLS trivia:

The SSL protocol ensures:

a) Confidentiality and data integrity

b) Authenticity

c) Man in the middle attacks

d) Confidentiality, integrity and authenticity

The difference between SSL and TLS is:

a) The physical level in which each protocol is used

b) The volume of data they can protect

c) TLS is better than SSL in protecting against several attacks

d) They are exactly the same

The SSL Handshake Protocol enables:

a) The definition of how to encapsulate the data transmitted between client and server

b) The configuration of keys for a one time pad encryption

c) The negotiation of the security parameters required to establish a secure communication between client and server

d) Web clients to use vulnerable encryption algorithms

The SSL/TLS Record Protocol enables:

a) The exchange of session keys for encrypting a communication

b) The definition of how to encapsulate the data transmitted between client and server

c) The storage of session keys used to ensure the integrity of the sent data

d) The SSL handshake protocol to use 1024-bit RSA keys

SSL/TLS protects against MITM attacks on online banking access:

a) Provided that a Web client has the guarantee that the digital certificate received from the Web server is valid

b) Regardless of the nature of the certificate that identifies the server

c) SSL doesn't protect against MITM attacks

d) Whenever a user enters the URL of the bank in a web browser to confirm that it's the correct URL

Exercise

Choose one of the following options to encrypt in the transport layer:

1) Remote connection using SSH and Telnet.
2) Transfer files via FTP and SFTP.
3) Web server to create client/server requests.
4) Encrypt with SSL the python sockets applications implemented in the first part of the course.
5) Create a sketch in Arduino to connect a WiFi network and make an HTTPS request.
6) Connect to an existing HTTP/HTTPS web server, in addition write an 1-2 pages report about TLSv1.3.

For any of the implementations you choose, you should show the packets without encryption and after encryption to see the difference and the protocol handshake. You can refer to the following material for the implementations:

- https://www.arduino.cc/en/Tutorial/WiFiNINAWiFiSSLClient
- https://github.com/xliu59/SSL-TLS_SOCKET
- https://tomcat.apache.org/tomcat-9.0-doc/ssl-howto.html
- https://websiteforstudents.com/create-ssl-tls-self-signed-certificates-on-ubuntu-16-04-18-04-18-10/

The submission is in the form of a report explaining your results, wireshark captures, code and relevant information in a zip file by 27-03-2020

Assessment Criteria

1. Implementation – 4 points
2. Explanation including all required files – 3 points
3. Report – 3 points

References:

[1] https://kevincurran.org/com320/labs/wireshark/lab-ssl.pdf (Accessed on 26.3.2019)

[2] The Transport Layer Security (TLS) Protocol Version 1.2 RFC specification. T. Dierks, E. Rescorla. Aug. 2008. https://tools.ietf.org/html/rfc5246#page-42

[3] http://www.cs.kent.edu/~mallouzi/ccn%20Spring%202014/lab-ssl.pdf (Accessed on 26.3.2019)

[4] James F. Kurose and Keith W. Ross. Computer Networking – A Top-Down Approach. Pearson.

[5] http://www.criptored.upm.es/intypedia/docs/en/video9/ExercisesIntypedia009.pdf. Author: PhD Alfonso Muñoz Muñoz (Accessed on 26.3.2019)