CNS Assignment
SSL Wireshark Assignment

Harsh Agarwal
CS15BTECH11019

1)

a) Frame 1
   Source : client (192.168.117.63)
   No of SSL records : 1
   SSL Type : Handshake(22) Client Hello

b) Frame 2
   Source : server(216.58.196.36)
   No of SSL records : 1
   SSL Type : Handshake(22) Server Hello

c) Frame 3
   Source : server(216.58.196.36)
   No of SSL records : 3
   SSL Type : Handshake(22) Certificate, Handshake(22) Server Key Exchange, Handshake(22) Server Hello Done

d) Frame 4
   Source : client (192.168.117.63)
   No of SSL records : 1
   SSL Type : Handshake(22) Client Hello

e) Frame 5
   Source : server(216.58.196.36)
   No of SSL records : 1
   SSL Type : Handshake(22) Server Hello

f) Frame 6
   Source : server(216.58.196.36)
   No of SSL records : 3
   SSL Type : Handshake(22) Certificate, Handshake(22) Server Key Exchange, Handshake(22) Server Hello Done

g) Frame 7
   Source : client(192.168.117.63)
   No of SSL records : 3
   SSL Type : Handshake(22) Client Key Exchange, Change Cipher Spec(20), Handshake(22) Encrypted Handshake Message

h) Frame 8
   Source : server(216.58.196.36)
   No of SSL records : 3

SSL Type : Handshake(22) New Session Ticket, Change Cipher Spec(20), Handshake(22) Encrypted Handshake Message

Timing Diagram

Client Server

→Client Hello→

←Server Hello←

←Certificate←

←Server Key Exchange←

←Server Hello Done←

→Client Hello→

←Server Hello←

←Certificate←

←Server Key Exchange←

←Server Hello Done←

→Client Key Exchange→

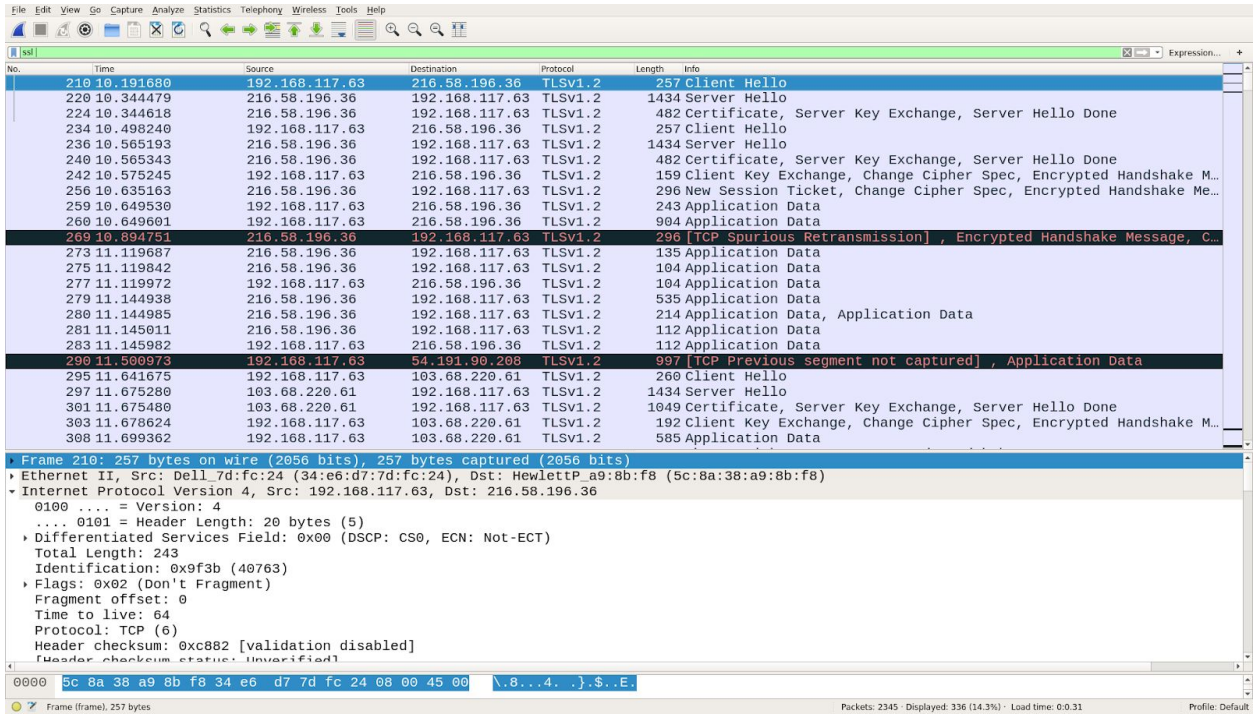
→Change Cipher Spec→

→Encrypted Handshake Message→

←New Session Ticket←

←Change Cipher Spec←

←Encrypted Handshake Message←

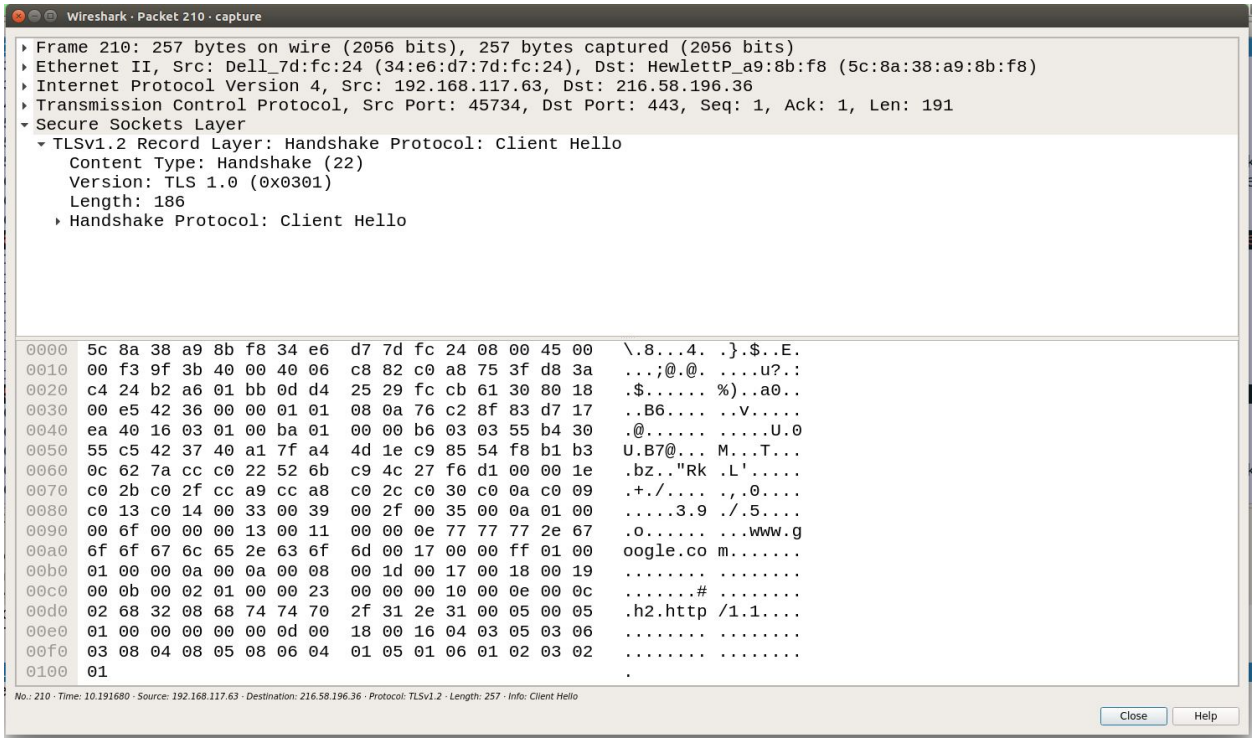


2)

Content Type :: 1 bytes

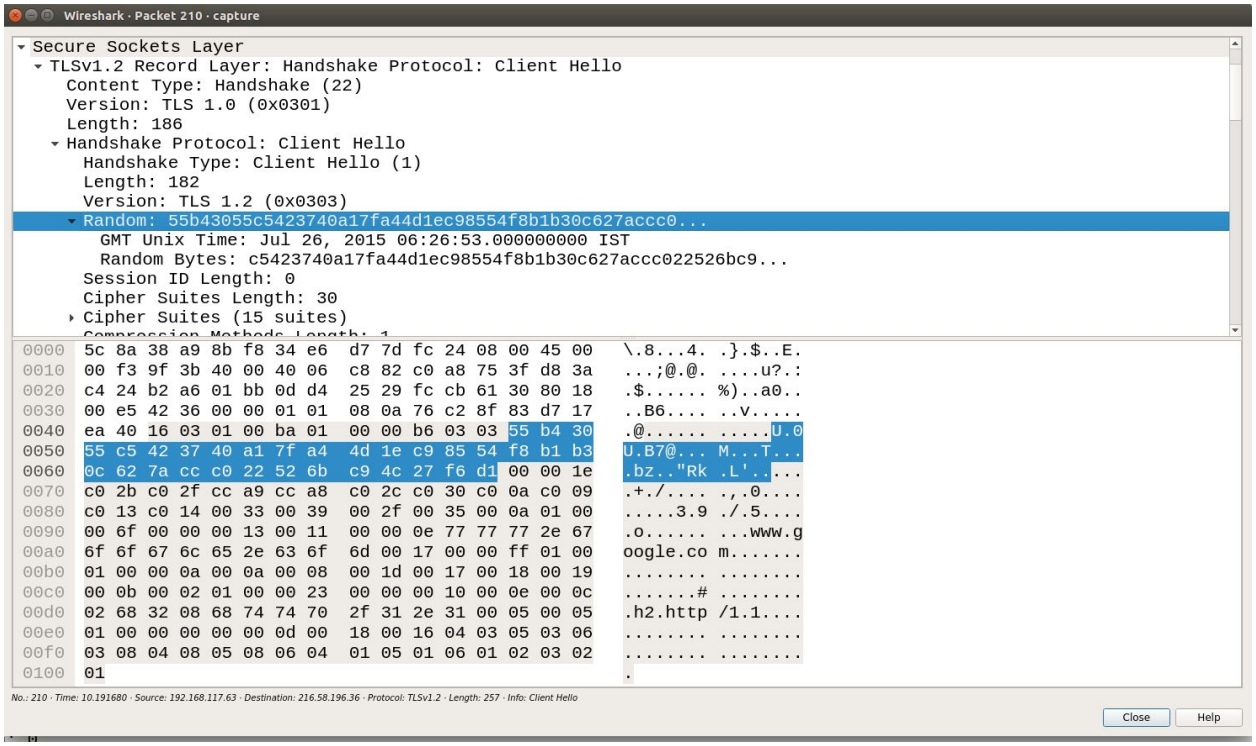
Version :: 2 bytes
Length :: 2 bytes

3) Client Hello's Content Type is Handshake(22)



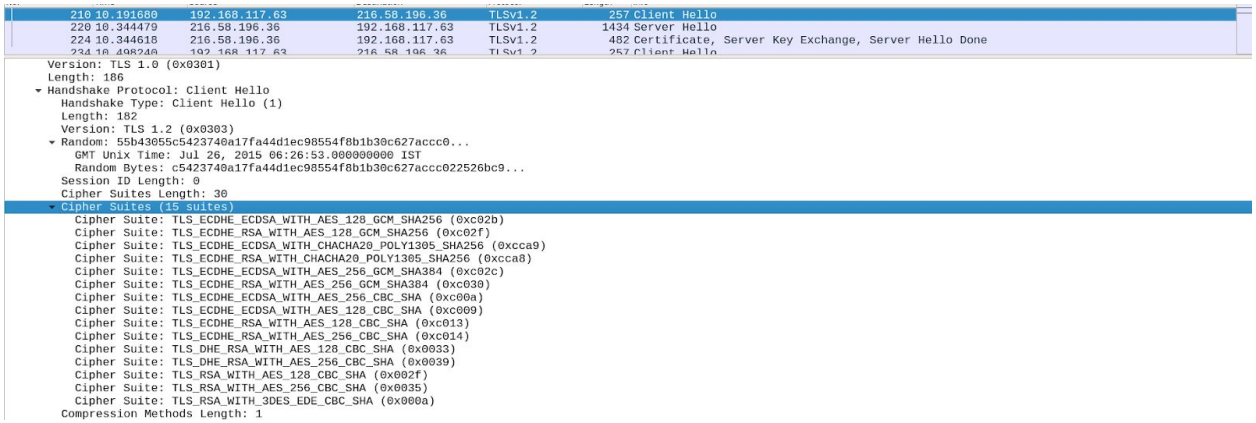
4)
In packet noonce is shown by Random field which contains GMT Unix Time and Random Bytes
Complete Random Field :: 55 b4 30 55 c5 42 37 40 a1 7f a4 4d 1e c9 85 54
0010 f8 b1 b3 0c 62 7a cc c0 22 52 6b c9 4c 27 f6 d1
GMT Unix Time :: 55 b4 30 55
Random Bytes :: c5 42 37 40 a1 7f a4 4d 1e c9 85 54

0010 f8 b1 b3 0c 62 7a cc c0 22 52 6b c9 4c 27 f6 d1



5)

Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
Public-Key algorithm :: RSA, ECDSA(Elliptic Curve Digital Signature Algorithm)
Symmetric-Key Algorithm :: AES_128_GCM, CHACHA20_POLY1305
Hash Algorithm :: SHA256, SHA384, SHA



6) Chosen cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

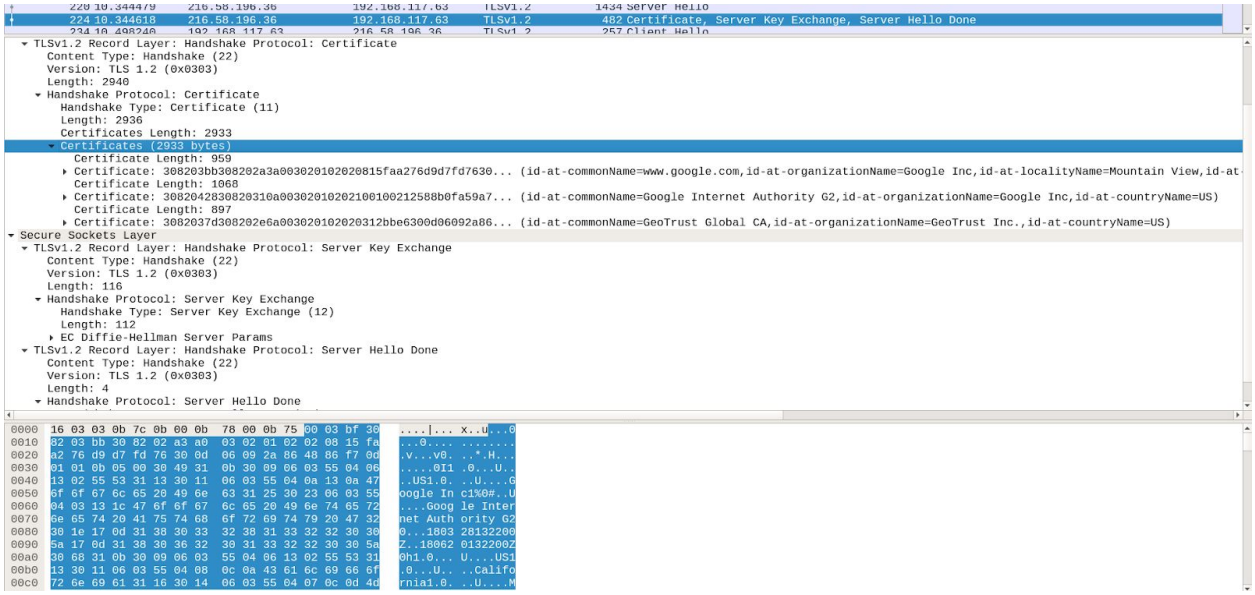
TLS - the protocol used
ECDHE - the key exchange mechanism - elliptic curve diffie-hellman key exchange
ECDSA - the algorithm of the authentication key - Elliptic Curve Digital Signature Algorithm
AES - the symmetric encryption algorithm - Advanced Encryption Standard
128 - the key size of the above
GCM - the mode of the above - Galois/Counter Mode (GCM)
SHA256 - the MAC used by the algorithm - Secure Hash Algorithm-256

7) Yes noonce is there - 32 bytes
Purpose :: A *nonce* is a unique value chosen by an entity in a protocol, and it is used to protect that entity against replay attacks. It is often a random number issued in an authentication protocol to ensure that old communications cannot be reused in replay attacks.

8) No there is no session ID
Purpose :: A session ID is a unique number that a Web site's server assigns a specific user for the duration of that user's visit (session). The session ID can be stored as a cookie, form field, or URL. As session IDs are often used to identify a user that has logged into a website.

9) Certificate is in a different record. Yes certificate fits into a single frame.

Infact 3 SSL records are in the frame along with certificate



10)
X.509 certificate ::
-----BEGIN CERTIFICATE-----
MIIDfTCCAuagAwIBAgIDErvmMA0GCSqGSIb3DQEBBQUAME4xCzAJBgNVBAYTAlVT
MRAwDgYDVQQKEwdFcXVpZmF4MS0wKwYDVQQLEyRFcXVpZmF4IFNlY3VyZSBDZXJ0
aWZpY2F0ZSBBdXRob3JpdHkwHhcNMDIwNTIxMDQwMDAwWhcNMTgwODIxMDQwMDAw
WjBCMQswCQYDVQQGEwJVUzEWMBQGA1UEChMNR2VvVHJ1c3QgSW5jLjEbMBkGA1UE
AxMSR2VvVHJ1c3QgR2xvYmFsIENBMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEA2swYYzD99BcjGlZ+W988bDjkcbd4kdS8odhM+KhDtgPpTSEHCIjaWC9m
OSm9BXiLnTjoBbdqfnGk5sRgprDvgOSJKA+eJdbtg/OtppHHmMlCGDUUna2YRpIu
T8rxh0PBFpVXLVDviS2Aelet8u5fa9IAjbkU+BQVNdnARqN7csiRv8lVK83Qlz6c
JmTM386DGXHKTubU1XupGc1V3sjs0l44U+VcT4wt/lAjNvxm5suOpDkZALeVAjmR
Cw7+OC7RHQWa9k0+bw8HHa8sHo9gOeL6NlMTOdReJivbPagUvTLrGAMoUgRx5asz
PeE4uwc2hGKceeoWMPRfwCvocWvk+QIDAQABo4HwMIHtMB8GA1UdIwQYMBaAFEjm
aPkr0rKV10fYIyAQTzOYkJ/UMB0GA1UdDgQWBBTAephojYn7qwVkDBF9qn1luMrM
TjAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwIBBjA6BgNVHR8EMzAxMC+g
LaArhilodHRwOi8vY3JsLmdlb3RydXN0LmNvbS9jcmxzL3NlY3VyZWNhLmNybDBO
BgNVHSAERzBFMEMGBFUdIAAwOzA5BggrBgEFBQcCARYtaHR0cHM6Ly93d3cuZ2Vv
dHJ1c3QuY29tL3Jlc291cmNlcy9yZXBvc2l0b3J5MA0GCSqGSIb3DQEBBQUAA4GB
AHbhEm5OSxYShjAGsoEIz/AIx8dxfmbuwu3UOx//8PDITtZDOLC5MH0Y0FWDomrL
NhGc6Ehmo21/uBPUR/6LWlxz/K7ZGzIZOKuXNBSqltLroxwUCEm2u+WR74M26x1W
b8ravHNjkOR/ez4iyz0H7V84dJzjA1BOoa+Y7mHyhD8S
-----END CERTIFICATE-----

a) Public key length :: 2048 bits

b) Public key encryption algorithm :: rsaEncryption

c) Signature Algorithm :: sha1WithRSAEncryption

d) SignatureValue ::
AHbhEm5OSxYShjAGsoEIz/AIx8dxfmbuwu3UOx//8PDITtZDOLC5MH0Y0FWDomrLNhGc6Eh
mo21/uBPUR/6LWlxz/K7ZGzIZOKuXNBSqltLroxwUCEm2u+WR74M26x1Wb8ravHNjkOR/ez4iy
z0H7V84dJzjA1BOoa+Y7mHyhD8S

e) Certificate issuer ::

Issuer: C=US, O=Equifax, OU=Equifax Secure Certificate Authority

f) Common name and Alternative name (if any) ::
DNS Name: www.onlinesbi.com

g) Key usage and purpose of the certificate ::
Critical
Certificate Sign, CRL Sign

10)

It is used to provide greater consistency between TLS cipher suites. This and 2 noonces above are used to generate the master key.
This uses Diffie Hellman Key exchange.
Length : 32 bit
Key shared by client to server :: Pubkey:
27:ac:1b:61:0c:e1:3e:15:47:4f:67:a2:e7:8c:62:11:2c:37:1c:90:83:57:5a:9d:0d:07:82:0f:7b:6f:43:3e
Actually, this is part of the DH exchange. This will go to server and finally both client and server will have same key . This key is the pre-master secret. It is not explicitly mentioned in the packet but calculated on both hosts.

11) The change cipher spec protocol is used to change the encryption being used by the client and server. It is used as part of the handshake process to switch to symmetric key encryption.
Length :: 6 bytes

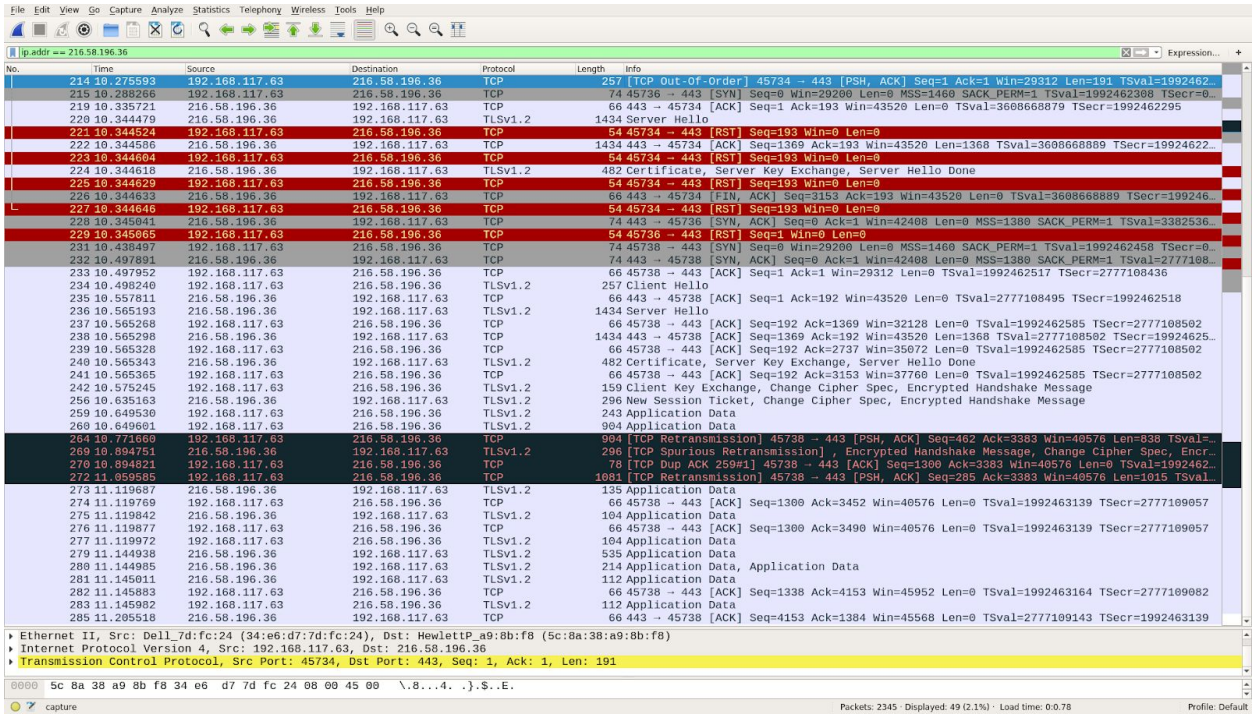
12) A finished message is being encrypted using the symmetric keys negotiated earlier. This includes a a sender code. If this message can be decrypted properly, then we are good to go and can start encrypting traffic

13) Yes.
Server has different sender code which is hashed to become the message payload . This is different from client finished message as client will send his own code.

14)
Message is encrypted using agreed symmetric keys & algorithms.
Yes a MAC is there with each record for every application data message.
No, Wireshark doesn't distinguish.

15)
The thing I found on this wireshark Lab is different SSL Record types, the different encryption and decryption, algorithms, new session ticket , how application data is encrypted , how SSL closes the connection.

16)



Total 49 packets are exchanged in the entire session (displayed at bottom)
Duration of HTTPS session :: 0.929925 sec

17)
Using noonces and pre_master , master_secret is found using PRF (pseudo-random function).
Pseudo Code :
master_secret = PRF(pre_master_secret, "master secret", ClientHello.random + ServerHello.random)
PRF(secret, label, seed) = P_<hash>(secret, label + seed)

P_hash(secret, seed) = HMAC_hash(secret, A(1) + seed) +
HMAC_hash(secret, A(2) + seed) +
HMAC_hash(secret, A(3) + seed) +
A(0) = seed
A(i) = HMAC_hash(secret, A(i-1))

Algorithm :: label + client-nonce+ server-noonce is calculated and treated as seed. Then secret and seed is hashed sequentially until 48 bytes string is obtained. This is the master key. From the master key the following are derived

- client_write_MAC_key
- server_write_MAC_key
- client_write_key
- Server_write_key

The master_secret is found using PRF- the same way as premaster_key. This is repeated until sufficient length is obtained to get the 4 diff keys
key_block = PRF(SecurityParameters.master_secret,
"key expansion",
SecurityParameters.server_random +
SecurityParameters.client_random);

The browser does these things for us.