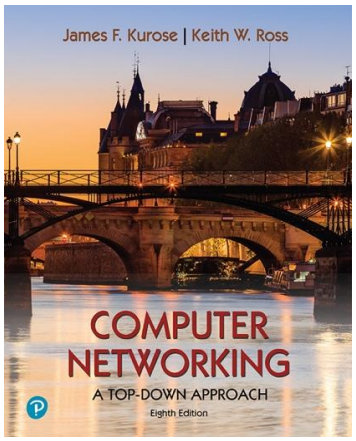
Wireshark Lab: TLS v8.1

Supplement to *Computer Networking: A Top-Down Approach, 8th ed.,* J.F. Kurose and K.W. Ross



"Tell me and I forget. Show me and I remember. Involve me and I understand." Chinese proverb

---

In this lab, we'll investigate Transport Layer Security (known as TLS) and aspects of the authentication, data integrity, and confidentiality services provided by TLS. TLS is the successor to the now-deprecated Secure Sockets Layer (SSL). So, it'd be preferable to do this v8.1 TLS Wireshark lab, rather than the v8.0 SSL Wireshark Lab.

We'll investigate TLS by analyzing a Wireshark packet trace captured during the retrieval of a web page via HTTPS - a secure version of HTTP, which implements TLS on top of HTTP. We'll look at TLS's client-server handshaking protocol in some detail, since that's where most of the interesting action happens. You may want to review Section 8.6 in the text[1]; useful online resources for learning more about TLS are here[2], here[3], and here[4]; and, of course, in RFC 5246.

1. Capturing packets in an TLS session

The first step in this lab is to capture the packets in an TLS session. To do this, you should startup Wireshark and begin packet capture, retrieve the homepage from https://www.cics.umass.edu using the browser of your choice, and then stop Wireshark packet capture. The 's' after 'http' will cause the **Hypertext Transfer Protocol Secure (HTTPS)** - an extension of HTTP – to be used to securely retrieve the homepage from www.cics.umass.edu. Here, "securely" means that the www.cics.umass.edu server will be authenticated by your web browser, that the transmission of your client HTTP GET

---

[1] References to figures and sections are for the 8th edition of our text, *Computer Networks, A Top-down Approach, 8th ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2020.* Our authors' website for this book is http://gaia.cs.umass.edu/kurose_ross You'll find lots of interesting open material there.

[2] "What is Transport Layer Security (TLS)?", Cloudflare, https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/

[3] A. Munshi, "TLS v1.2 handshake overview", https://medium.com/@ethicalevil/tls-handshake-protocol-overview-a39e8eee2cf5

[4] "Taking a Closer Look at the SSL/TLS Handshake," https://www.thesslstore.com/blog/explaining-ssl-handshake/#the-tls-12-handshake-step-by-step

request and the server's reply will be encrypted, and the integrity of all message content will be cryptographically verified. Of course, the authentication, integrity and encryption of a computer science department's webpage may not be as critical as that for Internet commerce and banking sites, but the same TLS protocol and TLS messages are used in all cases.

If you're unable to run Wireshark on a live network connection, you can download a packet trace that was captured while following the steps above on one of the author's computers [5]. In addition, you may well find it valuable to download this trace even if you've captured your own trace and use it, as well as your own trace, when you explore the questions below.

2. A first look at the captured trace

Let's first set Wireshark's display so that only the packets to and from www.cics.umass.edu, whose IP address is 128.119.240.84, are displayed. To do this, enter

```
ip.addr == 128.119.240.84
```

in Wireshark's display filter window. Your screen should look similar to what is shown in Figure 1.

[5] You can download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces-8.1.zip and extract the trace file *tls-wireshark-trace1*. This trace file can be used to answer this Wireshark lab without actually capturing packets on your own. This trace was made using Wireshark running on one of the author's computers, while performing the steps indicated in this Wireshark lab. Once you've downloaded a trace file, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the trace file name.

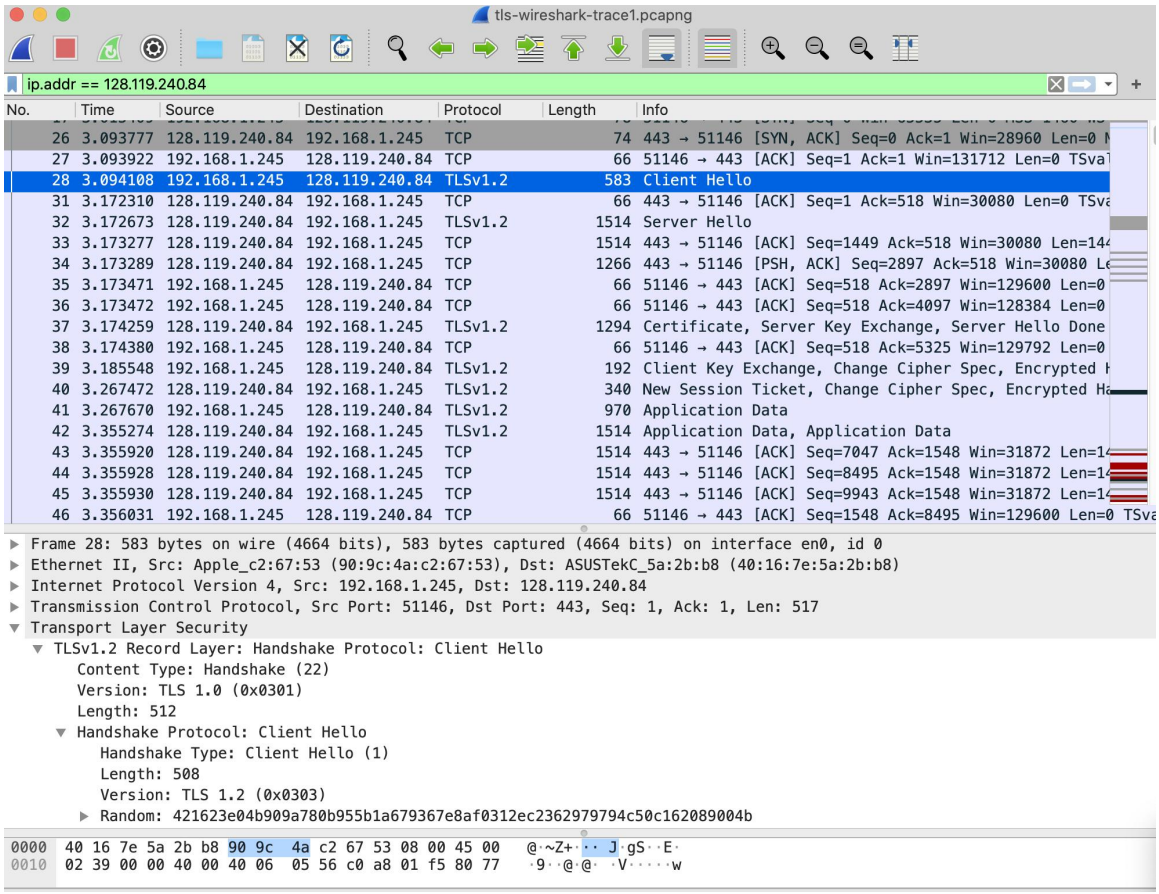


Figure 1: Wireshark display showing TCP and TLS message to/from 128.119.240.84

It's important to keep in mind that an Ethernet frame (containing an IP datagram containing an TCP segment) may contain one or more TLS records. (This is very different from HTTP, for which each frame contains either one complete HTTP message or a portion of a HTTP message.) Also, a TLS record may not completely fit into an Ethernet frame, in which case multiple frames will be needed to carry the record.

In answering the questions below[6], you can use either your own live trace, or use the Wireshark captured packet file *TLS-wireshark-trace1* in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces-8.1.zip

We've said earlier that HTTPS implements TLS running "over" TCP. That means that a TCP connection must first be established between your browser and the web server for www.cics.umass.edu before TLS and HTTP messages can be exchanged, just as we saw with the vanilla (non-TLS) HTTP protocol.

---

[6] For the author's class, when answering the following questions with hand-in assignments, students sometimes need to print out specific packets (see the introductory Wireshark lab for an explanation of how to do this) and indicate where in the packet they've found the information that answers a question. They do this by marking paper copies with a pen or annotating electronic copies with text in a colored font. There are also learning management system (LMS) modules for teachers that allow students to answer these questions online and have answers auto-graded for these Wireshark labs at http://gaia.cs.umass.edu/kurose_ross/lms.htm

1. What is the packet number in your trace that contains the initial TCP SYN message? (By "packet number," we meant the number in the "No." column at the left of the Wireshark display, not the sequence number in the TCP segment itself).
2. Is the TCP connection set up before or after the first TLS message is sent from client to server?

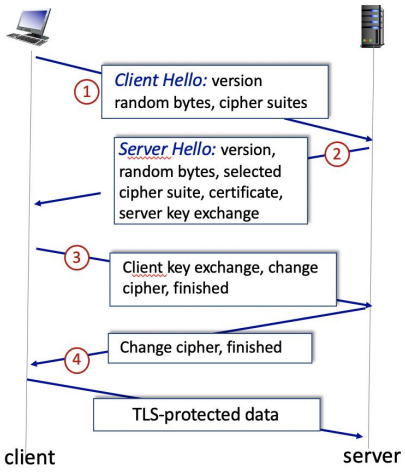


Figure 2: TLS handshaking

3. The TLS Handshake: *Client Hello* message

We learned that, as shown in Figure 2, the client-server TLS handshake begins with the client sending a TLS *Client Hello* message.

3. What is the packet number in your trace that contains the TLS *Client Hello* message?
4. What version of TLS is your client running, as declared in the *Client Hello* message?
5. How many cipher suites are supported by your client, as declared in the *Client Hello* message? A cipher suite is a set of related cryptographic algorithms that determine how session keys will be derived, and how data will be encrypted and be digitally signed via a HMAC algorithm.
6. Your client generates and sends a string of "random bytes" to the server in the *Client Hello* message. What are the first two hexadecimal digits in the random bytes field of the *Client Hello* message? Enter the two hexadecimal digits (without spaces between the hex digits and without any leading '0x' , using lowercase letters where needed). *Hint:* be careful to fully dig into the `Random` field to find the `Random Bytes` subfield (do not consider the `GMT UNIX Time` subfield of `Random`).
7. What is the purpose(s) of the "random bytes" field in the *Client Hello* message? Note: you'll have do some searching and reading to get the answer to this question; see section 8.6 and in RFC 5246 (section 8.1 in RFC 5246 in particular).

3. The TLS Handshake: *Server Hello* message

Next, let's take a look at the second step of the TLS handshake, the TLS *Server Hello* message, which is sent in response to the earlier TLS *Client Hello* message.

8. What is the packet number in your trace that contains the TLS *Server Hello* message?
9. Which cipher suite has been chosen by the server from among those offered in the earlier *Client Hello* message?
10. Does the *Server Hello* message contain random bytes, similar to how the *Client Hello* message contained random bytes? And if so, what is/are their purpose(s)?

In response to the initial *Client Hello* message, the server also sends back two important additional pieces of information (that may be contained in a different packet than the initial part of the *Server Hello* message. The first important piece of information returned is a certificate issued by a trusted certification authority (CA; see section 8.3 in the text) that binds the public key (and other information) of the web server to that web server's identity. Your client may use the server's public key for a number of different purposes, including deriving the symmetric keys to encrypt data being sent over this HTTPS/TLS/TCP session and for generating HMAC digital signatures. You can, of course, look at the server's certificate in Wireshark. You can also view the server's certificate in your web browser after the www.cs.umass.edu server has responded to your request (and the certificate formatted a lot prettier too) – here's how.

The second piece of additional information returned from the server are parameters needed for the symmetric encryption of the data (HTTP messages in this case) being exchanged.

Let's dig a bit deeper into the public key certificate.

11. What is the packet number in your trace for the TLS message part that contains the public key certificate for the www.cics.umass.edu server (actually the www.cs.umass.edu server)?
12. A server may return more than one certificate. If more than one certificate is returned, are all of these certificates for www.cs.umass.edu? If not all are for www.cs.umass.edu, then who *are* these other certificates for? You can determine who the certificate is for by checking the `id-at-commonName` field in the retuned certificate.
13. What is the name of the certification authority that issued the certificate for `id-at-commonName=www.cs.umass.edu`?
14. What digital signature algorithm is used by the CA to sign this certificate? Hint: this information can be found in `signature` subfield of the `SignedCertificate` field of the certificate for www.cs.umass.edu.
15. Let's take a look at what a real public key looks like! What are the first four hexadecimal digits of the modulus of the public key being used by www.cics.umass.edu? Enter the four hexadecimal digits (without spaces between the hex digits and without any leading '0x' , using lowercase letters where needed,

and including any leading 0s after '0x'). Hint: this information can be found in `subjectPublicKeyInfo` subfield of the `SignedCertificate` field of the certificate for www.cs.umass.edu.

16. Look in your trace to find messages between the client and a CA to get the CA's public key information, so that the client can verify that the CA-signed certificate sent by the server is indeed valid and has not been forged or altered. Do you see such message in your trace? If so, what is the number in the trace of the first packet sent from your client to the CA? If not, explain why the client did not contact the CA.

The *Server Hello* message is always terminated by an explicit *Server Hello Done* record.

17. What is the packet number in your trace for the TLS message part that contains the *Server Hello Done* TLS record?

4. The TLS Handshake: wrapping up the handshake

After the exchange of *Hello* messages, the client responds to the TLS *Server Hello* message with its own public key information, a declaration (via a *Change Cipher Spec* record) that all further communication will be encrypted via the negotiated algorithm and key, and an *Encrypted Handshake Message* record that contains encrypted information (e.g., a cryptographic hash of all messages exchanged during this handshake) to prevent man-in-the-middle replay attacks.

18. What is the packet number in your trace for the TLS message that contains the public key information, *Change Cipher Spec,* and *Encrypted Handshake* message, being sent from client to server?
19. Does the client provide its own CA-signed public key certificate back to the server? If so, what is the packet number in your trace containing your client's certificate?

Lastly, the server responds to the client with its own a declaration (via a *Change Cipher Spec* record) that all further communication will be encrypted via the negotiated algorithm and key, and an *Encrypted Handshake Message* record which also contains encrypted information (e.g., a cryptographic hash of all messages exchanged during this handshake) to prevent man-in-the-middle replay attacks.

5. Application data

Once the TLS handshaking has completed, the encrypted application data can begin to flow over the HTTP-over-TLS-over-TCP connection. Of course, since this data is encrypted we can't actually examine the contents of these encrypted messages (and isn't that just the point!). But we can still ask a few last questions about the encrypted traffic.

20. What symmetric key cryptography algorithm is being used by the client and server to encrypt application data (in this case, HTTP messages)?
21. In which of the TLS messages is this symmetric key cryptography algorithm finally decided and declared?

22. What is the packet number in your trace for the first encrypted message carrying application data from client to server?
23. What do you think the content of this encrypted application-data is, given that this trace was generated by fetching the homepage of www.cics.umass.edu?

Lastly, let's take a look at how the client closes the TLS connection. You'll have to dig around a bit to answer this question (see the earlier, footnoted references).

24. What packet number contains the client-to-server TLS message that shuts down the TLS connection? Because TLS messages are encrypted in our Wireshark traces, we can't actually look *inside* a TLS message and so we'll have to make an educated guess here.