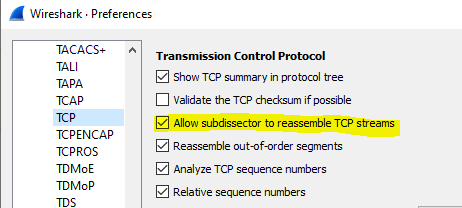
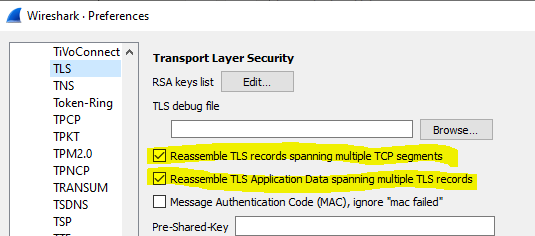
# Cryptography Homework 10 TLS

https://www.youtube.com/watch?v=86cQJ0MMses

## Important Wireshark Note

As of November 2019, the current version of Wireshark is 3.0.6. There were major changes between version 2.6 and the current version. In order to view TLS certificates, the following preferences must be set.

Edit > Preferences > Protocols > TCP. There must be a check in Allow subdissector to reassemble TCP streams.  


Edit > Preferences > Protocols > TLS. Reassemble TLS records spanning multiple TCP segments, and Reassemble TLS Application Data spanning multiple TLS records.  


As of November 2019, the version of Wireshark installed on Kali and Ubuntu is 2.6.10, which normally works without changing preferences.

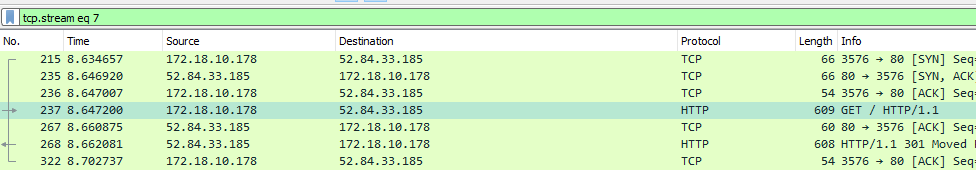
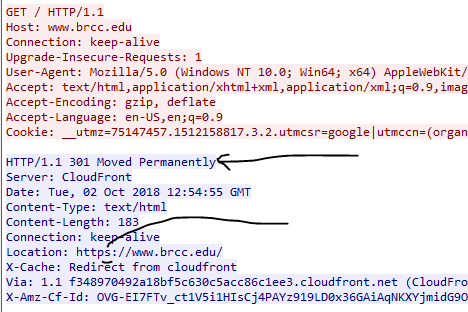
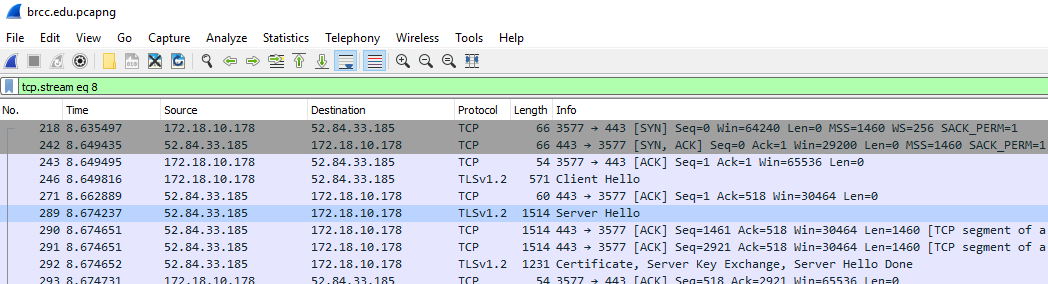
Note from @PacketJay: ‘Turning on [TCP] "reassemble out of order packets" will hog memory when your trace doesn't have all segments available. That's the main reason it isn't turned on by default because you might run into out-of-memory situations. My recommendation is to create a specific profile for TLS examinations which has all the settings you need for it.’

## Following a TLS Connection

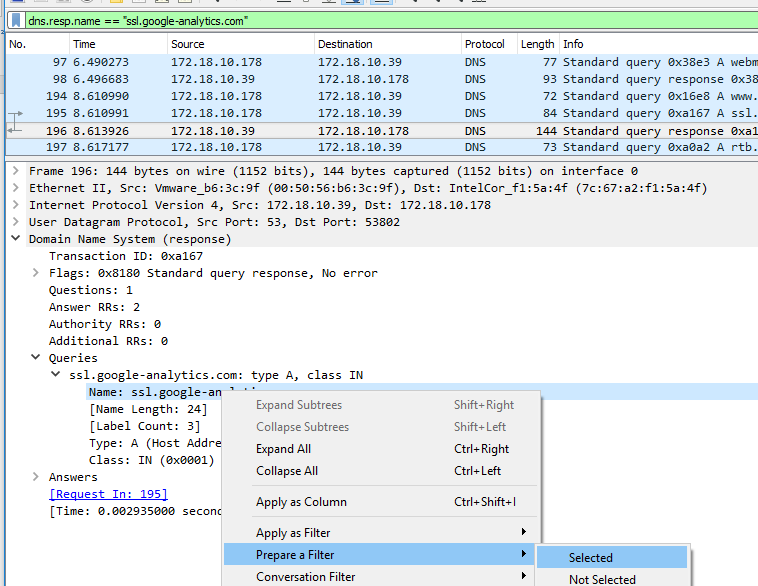
The file, brcc.edu.png, is a Wireshark packet capture file (pcap) that includes a TLS 1.2 session with <https://www.brcc.edu/>. We will use that as an example to revisit the slides showing an HTTPS connection. Open the file in Wireshark.

### Finding the connection using DNS and IP addresses

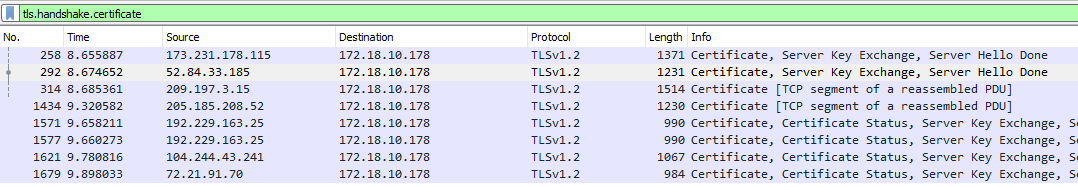
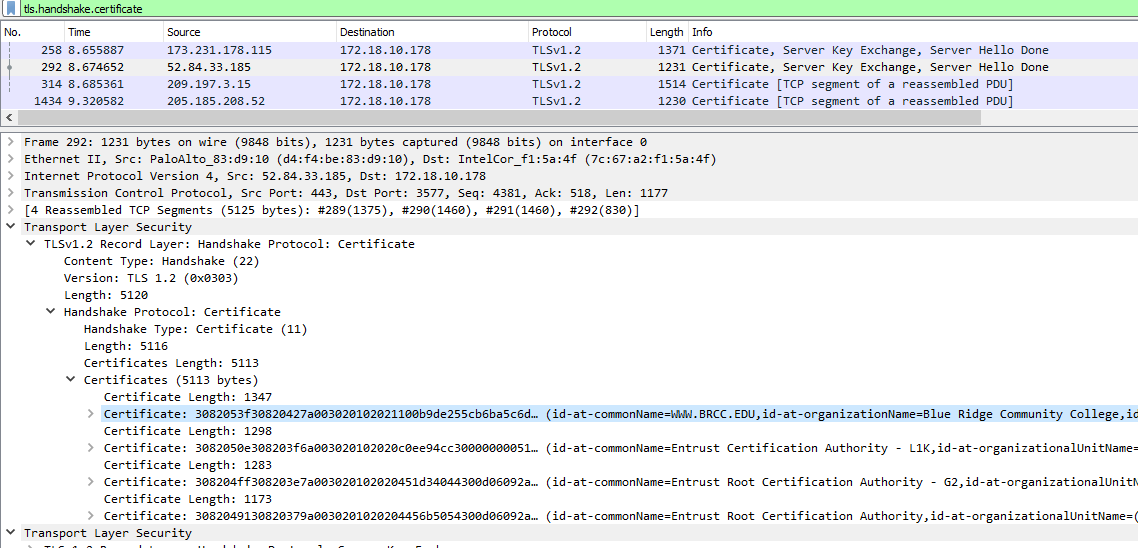
One of the hardest parts of using Wireshark on real world traffic is separating the traffic we want to see from the noise. The pcap file just includes the BRCC connection and normal Windows background traffic--we’re not even trying to separate one HTTPS session from bunches of others. One way to find what we want is to follow the way network traffic normally evolves. First locate the DNS query and response for [www.brcc.edu](http://www.brcc.edu). Then use the DNS response to find the IP address of the server. Then create a filter for the server IP addresses to find the session.

1. Use a display filter of dns. Does this limit things enough so that you can easily find the query for [www.brcc.edu](http://www.brcc.edu)? If not, try a better filter, dns contains “brcc” (note: because of the way DNS compresses data, it will not work if you use “brcc.edu”. )  
   What is the IP address of the server for [www.brcc.edu](http://www.brcc.edu)?
2. You should get four possible addresses for [www.brcc.edu](http://www.brcc.edu). You could enter the addresses into display filters one at a time and see which one the browser contacted, but you can save yourself some effort. Note that all four are on the same /24 network. You can catch all four, and hopefully not much background noise, by using this display filter:  
   ip.addr == 123.123.123.0/24  
   Note that 123.123.123.x is not the real address--you should use your answer to step 1.
3. When your IP filter is working, the first thing you should see is an HTTP session on port 80. All it contains is an HTTP 1.1 301 “Moved Permanently” error, which redirects the browser to HTTPS on port 443 (See the heading for Location in the Moved Permanently packet.)  
     
     
   This is one place where HTTP Strict Transport Security (HSTS) helps us. If an attacker executes a MITM attack against us, they won’t be able to stop the connection from moving to HTTPS. (The browser opened three connections; the one with source port 3576 is the one we are interested in.)
4. Immediately after the redirection, the browser starts an TLS handshake with the server. What is the client’s source port number for the HTTPS/TLS 1.2 connection?
5. Right-click a packet in the TLS connection and select Follow > TCP Stream. This will clear all other traffic from our Window.  
   
6. Examine the Client Hello packet. (Note that this is a TLS 1.2 connection with the longer handshake.) How many choices of cipher suites does the browser offer?
7. Still in the Client Hello packet, find the extension called Server Name Indicator (Extensions: server\_name). This portion of the session is unencrypted and allows the server to determine which web site the browser wants. (Remember that several sites can be hosted on the same IP address.) If you drill down, you should see “www.brcc.edu.”
8. Find the Server Hello packet. What cipher suite does the server select?
   1. TLS version \_\_\_\_\_\_\_\_
   2. Key Exchange \_\_\_\_\_\_\_\_
   3. Certificate \_\_\_\_\_\_\_\_
   4. Symmetric Encryption \_\_\_\_\_\_\_\_
   5. Hash \_\_\_\_\_\_\_\_
9. Note that the TLS protocol allows the hosts to put multiple TLS records into one packet. What other TLS records are included in the packet that holds the client’s “Change Cipher Spec” record?

### A helpful hint for finding Wireshark display filters.

The lists of Wireshark display filters are long and complicated. Rather than looking at the lists, if you can find the general type of traffic you want, you can use it to find the display filter name. We’ll use the DNS search we did before as an example for finding specific display filters. Find any DNS query, right-click on the query name, and select Prepare a Filter > Selected. We aren’t looking for ssl.google-analyticis.com, but this does show us that a correct display filter is dns.qry.name or dns.resp.name (finding the correct display filter can be time consuming, and this way works well for me: find a sample of the traffic I want and then use Prepare a Filter to see what the display filter should be.)  
  
  
Change the ==”ssl.google-analytics.com part of the filter to contains “brcc.edu”.  

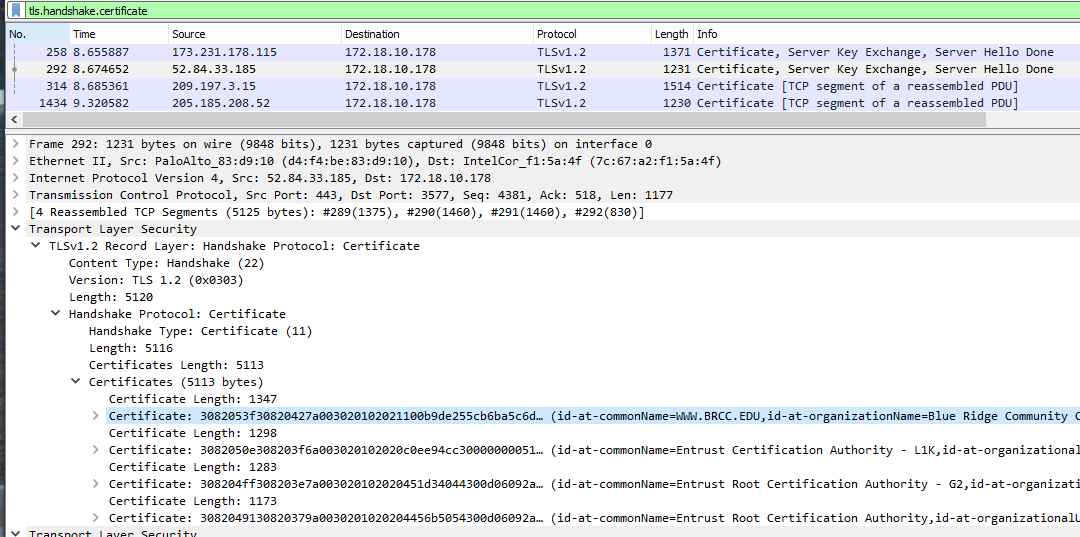

### Finding the connection using certificates instead of DNS and IP addresses

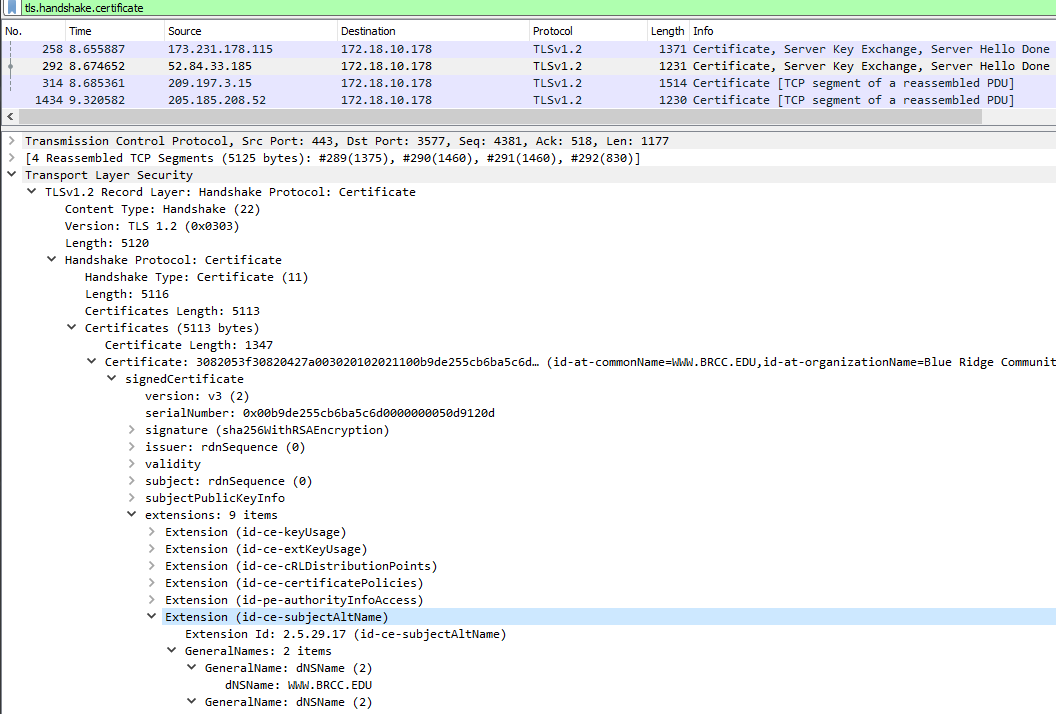
(While TLS 1.2 is still in use, this is very helpful. We’ll have to find something else for TLS 1.3.) Searching DNS traffic to find IP addresses, and then filtering IP addresses can be time consuming. It may be faster to search for certificates. Use the same pcap file as before, brcc.edu.png. This time use this display filter: tls.handshake.certificate (Note: older versions of Wireshark call this ssl.handshake.certificate .)  
  
Now you can open the SSL portion in the packet details pane so that the certificate name is displayed. Then you can cycle through the packets until you find the certificate name you want and select Follow > TCP stream on that packet.  


## Monitoring Certificate Names with Wireshark

The file, certificates.pcapng, contains the results of some random browsing. Although the HTTPS sessions are encrypted, we can get a general idea of what is taking place by looking at the digital certificates from the browsing. The certificates are transferred during the TLS/SSL handshake before encryption is enabled, so they are available in the network traffic.

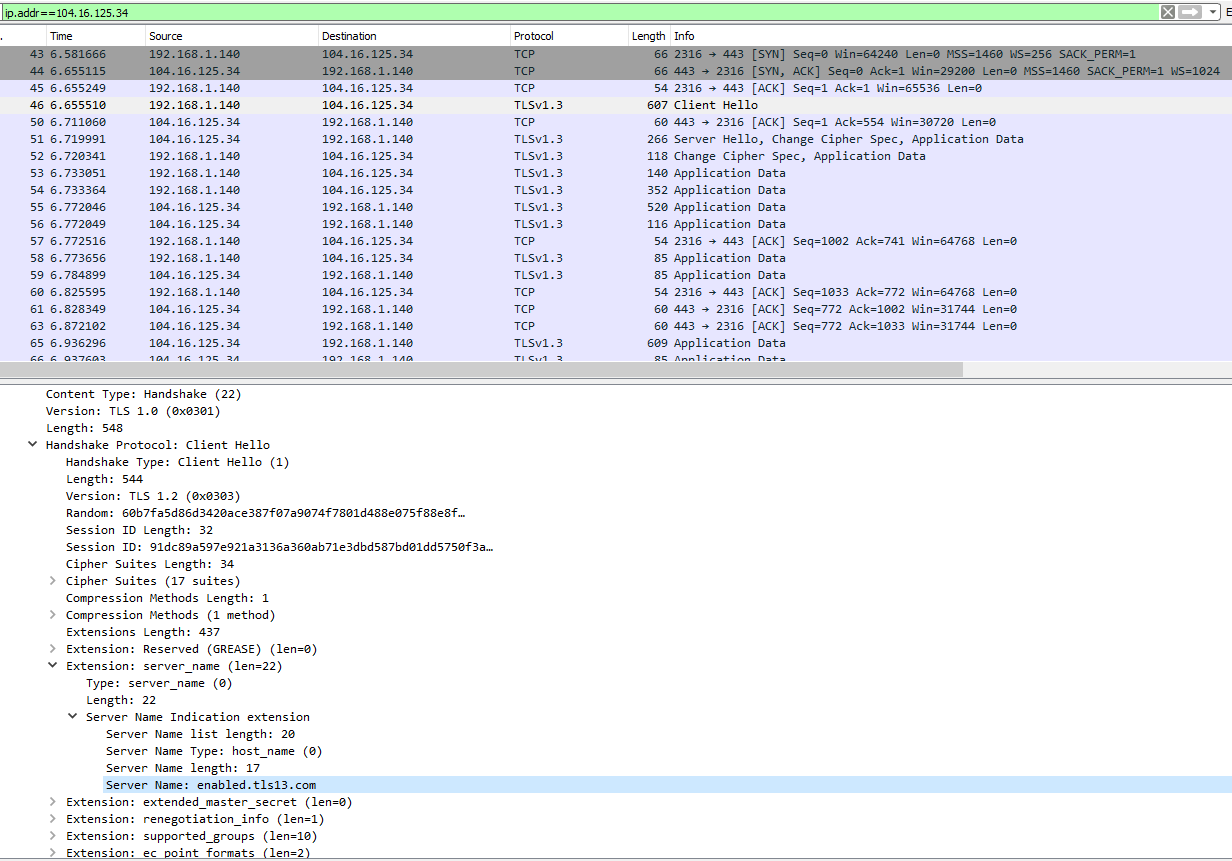
Open the file in Wireshark, and then use the display filter, tls.handshake.certificate. After opening the sections in the packet details that allow you to see the certificate names, you should be able to scroll through the packets and see the certificates. Note that your browser will accept both Common Names (CN) and Subject Alternative Names (SAN) as matches for what is displayed in the browser navigation bar.

Here’s where the CN can be found. (This is from the brcc.edu.pcng file to prevent giving you spoilers. You should use certificate.pcng.)  


Here’s where the SANs can be found (same note as above.)  


1. What certificates did you find?
2. Does the Amazon certificate for s3.amazon.com have any SANs?

# TLS 1.3

Start a packet capture, and then point your browser to <https://enabled.tls13.com/>. This is a simple site to test your browser to see if it supports TLS 1.3. You should find that the cipher suite list is still present. Since the server no longer gives us an unencrypted copy of its certificate, about all we have left is the Server Name Indication (SNI) extension, which is present in both TLS 1.2 and 1.3. Note that this does show that we are going to enabled.tls13.com. It allows the server to host multiple DNS names on the same IP address, just as the HOST field does for unencrypted HTTP.  


Note that the server replies with its Hello message and then immediately begins transmitting encrypted data. The handshake is much shorter in TLS 1.3. Also note that the certificate is not present in the unencrypted data.

The site <https://www.ssllabs.com/ssltest/viewMyClient.html> will test your browser to see what protocols it supports and let you know if your browser supports vulnerable protocols. Most likely your browser supports some outdated protocols so you can browse to sites that have not yet updated to secure protocols.

1. Does your browser support any insecure or weak protocols? What are they?