Lab: Practice Using Wireshark

Lab Objectives:

Obtain (or improve) familiarity with the traffic/protocol analyzer—Wireshark. Obtain (or improve) confidence regarding the extensive information that can be gleaned from "captured" network traffic. Understand how skilled usage of such a tool, along with knowledgeable interpretation of the protocols, facilitates both troubleshooting and forensic investigation.

Discussion:

The ability to "inspect" network packets and make sense of what action they convey is an important knowledge element for anyone who is in the business of cyber security. As we learn in a typical computer networking course, each protocol layer in the stack (e.g., [**Ethernet** [**IP** [**UDP** [**DNS**]]]]) provides "clues" regarding both the individual packet and the (possibly) larger "conversation" of which it is a part.

In this lab, you will familiarize yourself with Wireshark (http.//www.wireshark.org/download.html), and apply some of the knowledge you gained in this (or other) course to answer questions about some packets that have been provided. You are welcome to work with others if you find it helpful.

There are *approximately* (I change this lab from time to time) 53 questions to be answered. For optimal learning, try and determine the answers yourself, using the hints if you need them. Note that over half of the questions have the answers provided, but the text color for these answers has been turned to white, so that you cannot immediately see them. When you are ready to check an answer (or have given up trying to determine it), you can highlight the area where the answer **would** be, then change the highlighted font color from white (hence invisible) to black (hence visible).

==== **INSTALLATION/SETUP** (Below assumes installation/usage on Windows) ==========

There are two packet trace files included with this exercise: **`pax1forWSLab.cap`** and **`pax2forWSLab.cap`**. You will find them in the same course folder where you found this file.

I recommend you download the newest version of Wireshark (it's FREE) from the Internet (it's easy to find). There are versions for both Windows and MacOS!

Note, if you opt to download a more recent version, during the download process you may be prompted to also download (and/or install) several optional items. You do **not** *need* any of these additional items. However, if you foresee ever wanting to merge separate captured packet files together, you should select the "merge" option that is offered.

Launch the Wireshark application. Normally, you would likely be capturing actual "live" traffic from a network that you are interested in. This would be done by selecting **Capture/Start** from the menu bar, or by selecting the **Start a new live capture** button on the toolbar (3rd from left). Doing so would provide you with a new pop-up window that affords you the opportunity to make some decisions regarding which packets are captured. For the purpose of this tutorial; however, **the packets have already been captured for you**; therefore you need only open the file containing them, as discussed next.

Select **File** then **Open** from the menu bar, or select the **Open a capture file** button on the toolbar. Navigate to the directory holding the **`pax1forWSLab`** file that was provided as part of this exercise. Open this file. You should now see the complete collection of previously captured packets in the main window of Wireshark.

Most of the tool interaction is fairly intuitive; particularly given what you already know about the structure of *frames*, *packets*, *segments*, and *messages*; however a few basic ideas and useful features are provided below to help the novices among you. The tool also has a **Help** drop down menu available on the menu bar. You may also Google for a complete Wireshark user's guide if you want additional information, or search for and watch hours of youtube videos. The knowledge is out there and easy to access.

==== BASIC USAGE/INTERACTION WITH WIRESHARK ==========

Wireshark is comprised of three main windows, or "**panes**" as they are usually referred to.

The **top pane** displays the list of packets in an abbreviated form. By clicking on a packet in this pane, you control what is displayed in the two lower panes. There are many ways to modify what information is presented in this top pane. If you understand the basics of windows drop down menus, then you should have no problem changing/modifying your top pane to present the packet information as you prefer. The top pane is routinely referenced for the "big picture".

The **middle pane** is the "tree view" pane. It displays the packet selected in the top pane in more detail – layer by layer. You can "drill-down" on some of the information provided in this pane in order to get at ever more detailed information. This is where most of your packet analysis *work* gets done.

The **bottom pane** provides the "raw" view of the packet selected in the top pane. On the left side of this pane you will see the information in hex. On the right side you will see the exact same information presented in ASCII format. On occasion you will be able to read certain payload (TCP/IP layer 5) information directly from this section. Aside from that, you won't likely spend much time in the bottom pane. Look here when the analyzer cannot "interpret" for you, and you need the raw information so you can look it up in the appropriate RFC yourself.

You should now take a look at each of the items on the menu bar, and each of the buttons on the toolbar. Most of these are self-explanatory. I will only elaborate below on those that are of particular interest.

Menu bar. **Edit → Mark Packet** (toggle) -- here's a nice feature to "highlight" (mark) a particular packet that may be of interest to you. As you can see, **Ctrl-M** is the keyboard shortcut to do this.

Menu bar. **View → Time Display Format** – choose the format that is most helpful in your situation. I recommend **Time of Day** and **Automatic** for precision.

Menu bar. **View → Name Resolution** – Check this out! Wireshark offers four resolutions for you. *name*, *MAC*, *network*, and *transport*. You may remember that the left ½ of a MAC address (the left 24 bits) are unique to each vendor. If you enable MAC resolution, Wireshark will replace the left half of the MAC address with the appropriate vendor. This might be of interest in some situations. Transport resolution simply replaces the protocol number from the IP header (e.g., protocol #6) with the name of the protocol (e.g., TCP).

Menu bar. **View → Colorize Packet List** and **Colorizing Rules** – Here's a neat feature for helping you sort out different kinds of traffic features. If you simply select **Colorize Packet List** you will see that

many colors are already assigned for different protocols/fields. This will not help much, unless/until you take a little time to familiarize yourself with the color assignments. Select **Colorizing Rules**, further down in this drop-down menu list, to see the current color assignments.
You will see that you may alter these rules to suit your needs. Very handy!

Menu bar. **Statistics → Conversations** – Great… easy way to see at a glance "who" (IP address) is talking to "whom". If you recall the class discussion of "socket-pair", you understand now how the protocol analyzer does this.

Menu bar. **Statistics → Endpoints** – Quick summary of all hosts originating or receiving traffic.

Menu bar. **Statistics → Flow Graph…** -- Take a look at this. A nice graphical way to view the traffic.

Toolbar. **Reload this capture file** (9th from the left) → You can use this to "reset" the captured file back to the way it was after you do any sort of filtering.

Toolbar. **Find a packet…** (11th from the left) → Self explanatory and heavily utilized.

Toolbar. **Go to the packet with number…** (14th from the left) → Self explanatory, and handy for making easy reference to packets for the benefit of another person; e.g., I might suggest that you "check out the payload of packet #1344", and you could easily do this.

Toolbar. **Edit/apply display filter…** (4th from the right) → Self explanatory and heavily utilized.

Column headers in the top (packets) pane → Click on any of these headers to re-order the packets based upon that header criterion. Click the same column header again to reverse the order. The most popular/typical (and default) ordering of packets is incremental by time. It may—for whatever reason—be efficacious for you to sort on some other packet characteristic.

Okay, we're almost finished with the basic usage/interaction introduction. Last item is very important; the syntax for creating display filter rules. Note that I am focusing here only on **display** filtering, not on **capture** filtering. Though similar, the syntax is a little different. I'll leave it to you to consult the Wireshark Help files or a User's Guide if/when you need to write any capture filters. In all of "my" labs, we are never capturing live traffic, but rather simply viewing traffic that has already been captured.

Using display filters, you can choose which, of the *all* of the packets in the capture file, are displayed. This is useful to reduce unwanted "noise" from the top pane of packets. Display filters will **not** affect the data captured (i.e., you don't lose anything). Every time you change the filter criteria, all packets will be reread from the capture file and processed against your new criteria.

Wireshark offers a very powerful display filter language for this. It can be used for a wide range of purposes: from simply, "show only packets from a specific IP address", to very complex filters like, "find all packets where some particular *application* specific flag combination set".

Some common display filter rule examples.

Ethernet example. display all traffic to or from the Ethernet address 08.00.08.15.ca.fe

```
eth.addr==08.00.08.15.ca.fe
```

IP example. display all traffic to or from the IP address 192.168.0.10

```
ip.addr==192.168.0.10
```

TCP example. display all traffic to or from the TCP port 80 (http) of all machines

```
tcp.port==80
```

Combination example. display all traffic to or from 192.168.0.10 except http

```
ip.addr==192.168.0.10 and tcp.port!=80
```

As you type your rule expression, you will notice that the background will change between **red** and **green**. Red indicates that your rule is not syntactically correct… green means it is.

NOTE: When **negation** logic (e.g., "… and **not** tcp port = 80") is desired as part of your filter rule, you may notice the failure to obtain a **green** (good) indicator for your rule syntax. If you notice this, try changing the form of your expression as follows…

Change it from → `and tcp.port != 80` to → `and !(tcp.port == 80)`

Wireshark will help you build filter rules. Click on the **Edit/apply display filter…** button (4th from the right), and then click on the **Expression…** found near the bottom-right of the Display Filter window that pops up. You can figure it out from there. We will use this rule-building tool later in this exercise.

==== BEGIN TRAFFIC ANALYSIS AND QUESTION/ANSWER ==========

Open the file `pax1forWSLab.cap` if you have not done so already.

Q1. Is there any DHCP traffic in this trace? (yes / no)

A1.

Hint: Select the **Protocol** column header in the top pane in order to sort all the packets alphabetically by protocol. Scroll down until you see the DHCP traffic all together. If you did not have a handy analyzer like Wireshark at your disposal, you could manually search for source IPs 0.0.0.0 in the list of packets. As you may remember, 0.0.0.0 is the "John Doe" of IP addresses. It's what a computer with no IP address uses when trying to *obtain* an IP address.

Q2. What is the IP of the DHCP server?

A2.

Hint: Click on the **Protocol** column header in the top pane, then scroll down to DHCP. Look to see which IP is "offering" IP addresses. Be careful not to assume ALL non-0.0.0.0 IPs involved in DHCP are DHCP servers. If you recall our discussion of port numbers 67 and 68 back in Section 1, that is also helpful in answering this. port 67 = bootp**s** (**s**erver) and port 68 = bootp**c** (**c**lient)

Click on packet #706. This is the 1st DHCP request packet in this capture file.

Q3. Which company manufactured the NIC that this DHCP client is using?

A3.

Hint: Ensure you have a check mark in front of **Enable for MAC Layer** under the **View → Enable Name Resolution** on the menu bar. Look at the Ethernet layer information in the middle pane while you have packet #706 selected in the top pane.

Q4. Is this packet (packet #706) a fragment?

A4.

Hint: You must look at BOTH the **MF** (**M**ore **F**ragments) flag AND the **fragment offset** value in the IP header. To see this information, ensure you click the plus sign '+' that is in front of the "Internet Protocol" line in the middle pane.

Q5. Could this packet (packet #706) have *gotten* fragmented (i.e., maybe later in its routing)?

A5.

Hint: Check to see what the **DF** (**D**on't **F**ragmentf) flag is set to.

Q6. What port number do DHCP servers (aka bootstrap protocol [bootps] servers) *listen* on?

A6.

Hint: Look at layer 4 information (UDP in the case of this packet) in the middle pane, as this is where port info is found. Also, read my hint under **Q2**.

Q7. What is the Host Name of the DHCP client (sender of this packet)?

A7.

Hint: At what protocol layer do you think you'd find this information? Ans. layer 5… i.e.., the *payload* layer; after all, it is the layer that is "speaking" the language of DHCP. Drill down into the layer 5 information in the middle pane. In other words, make sure that you "expand" the layer 5 information in the middle pane by clicking on the '+' sign in front of "Bootstrap Protocol" (previous name for DHCP).

Q8. Is this client requesting **any** IP address, or does it already have one in mind?

A8.

Hint: (Same hint as for **Q7**)

Q9. <Question Removed>

Note that some information in the payload is in plain text format, and thus you can read it directly from the "raw" hex format in the bottom pane.

Q10. What two pieces of information can you read directly from the ASCII side of the payload section in the bottom pane of this (#706) packet?

A10.

Hint: None needed. Note that we now know about the client's OS as well: Microsoft.

Make sure that you still have your packets sorted by **Protocol** column (we did this initially right before starting on **Q2** – see above)

What packet number immediately follows the one we're currently looking at (#706) in the top pane? Ans. It is #708. Let's find packet #707 and see what we're missing. Could be un-related traffic of course, as the packet numbers are simply assigned based upon time of capture by Wireshark… but in this case it is related.

Use the **Go to packet with number…** button (14th from the left) on the toolbar to go to packet #707.

Q11. What layer 3 protocol is carried in this frame?

A11.

Hint: You cannot always get this answer by looking at the Protocol column in the top pane for this, as this column will sometimes tell you the layer 3 protocol (e.g., ARP), sometimes tell you the layer 4 protocol (e.g., TCP), and sometimes tell you the layer 5 protocol (e.g., DHCP which you just saw). Better to look into the layer 2 header information in the middle pane; specifically, look at the type field. Remember. the layer 2 **type** field tells you what it is carrying (usually a layer 3 protocol), the layer 3 **protocol** field tells you what it is carrying (usually—but not always—a layer 4 protocol), and the layer 4 **port** number tells you what it is carrying (usually layer 5 application-layer info… or "payload *proper*" as I sometimes refer to it).

Q12. Can you discern the purpose of this packet (#707) as it relates to the DHCP request in packet #706? If so, explain.

A12.

Hint: The DHCP client asked for a particular IP address (see **Q8**) and the DHCP server is ensuring that no other host already has this IP. (The hint is pretty much the answer in this case)

Assuming you still have your packets ordered by **Protocol**, scroll back down to the DHCP traffic. Select packet #708. This is a response from the DHCP server.

Q13. Are DHCP responses sent only to the requester (unicast), or to all on the local network (limited broadcast)?

A13.

Hint: What is the destination IP address? Hopefully you recognize the "limited broadcast" IP. You may also look at the destination MAC address, and—hopefully—recognize the "all Fs" layer 2 broadcast address. This makes sense – better to broadcast the new IP assignment so that other hosts on that network will now know of that new host's existence and IP, and can "complain" if they already have that assigned IP.

Q14. Does DHCP use TCP or UDP?

A14.

Hint: None required… I hope.

Q15. Since DHCP uses UDP, and since responses go to the "limited broadcast" address of 255.255.255.255 (FF.FF.FF.FF.FF.FF at layer 2), how does the requesting client know how to correlate DHCP responses to **its** particular DHCP query (vice replies broadcast to other DHCP requests)?

A15

Hint: Look for some identifying "ID" information in the DHCP payload. You will also see this in the top pane. This is information that both server and client can use to put corresponding DHCP query-response traffic together. Similar to how the IP identification number is used to re-associate IP fragments.

As you can see in packet #708, the client does receive the IP address it requested. If you take a quick peek at packet #705 (do not select it) in the top pane, you will see that this client appears to have just "released" this IP, and is now trying to "renew" it. This can be accomplished manually (i.e., human-initiated) by issuing the **`ipconfig /release`** command followed by the **`ipconfig /renew`** command on most Windows machines.

Q16. How long is the lease period for IPs doled out by this DHCP server? (Ensure packet #708 is still selected.)

A16.

Hint: Look in the payload—using the middle pane—of packet #708 and look for "lease time".

Let's shift gears and see if there are any fragmented packets in this capture file.

Q17. How would you know if a packet was a fragment?

A17.

Hint: You'd have to look at 2 things to ensure you caught all three instances of: the 1st fragment, the last fragment, and any "middle" fragment. This was discussed during lecture, but here is a little chart I've created to summarize this information for you.

	IP Byte [6]			IP Byte [7]											
	R	DF	MF	Fragment Offset											
1st frag	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Middle frag(s)	0	0	1	non zero											
Last frag	0	0	0	non zero											

Q18. What is the correct syntax for a display filter rule that will reveal only fragmented packets in the top pane?

A18.

Hint: Use the **Edit/apply display filter…** button (4th from the right on the toolbar), and click on the **Expression** button near the bottom right of the display filter window. You will build your 2-part rule (see **Q17**) by ***OR***ing the two necessary rules together. Note that some (all?) versions of Wireshark expect the double pipe symbol '||' for the logical ***OR*** operation. Depending on your version of Wirewhark, and your optional settings, you may also be able to use the word "or" (instead of the double-pipe). Remember… all of the logic that comprises "cyber" is man-made… and subject to variations in implementation and naming. You must be flexible when working in this discipline!

You should find eight fragmented packets. 6 will be labeled as protocol IP and 2 will be labeled protocol UDP.

Q19. How many original packets did these eight fragments come from?

A19.

Hint: The fragments are all between "dot100" and "dot102", so you need to look at the IP Identification numbers to figure out which belong together. Counting **first** fragments (look for "off=0" in the top pane, or "Fragment offset: 0" in middle pane) or **last** fragments (look for a non-zero "Fragment offset" value along with a "0x00" value for the "Flags" in the middle pane) could also be used to answer this question. You will notice that the last fragments of these **two** fragmented packets (yes, that's the answer to **Q19**) are listed as UDP, vice IPv4, as are the non last fragments. Just an oddity of Wireshark to label this way. Obviously, Wireshark *could* easily and correctly indicate that all eight of these fragments are UDP by simply looking at the Protocol field in each fragments IP header ($11_{16} = 17_{10}$ = UDP).

Q20. Is packet #724 a 1st fragment, a last fragment, or a middle fragment?

A20.

Hint: Look at the MF flag and the fragment offset value. Also see that packet #728 is the same

Clear your display filter expression by clicking on the **Clear** button that is adjacent to the expression window.

Reorder all packets by ascending time.

Go to packets #335 and #336.

Q21. Are ARP **replies** broadcast (sent to the "all-Fs" physical address), or only sent to the requester (unicast)?

A21.

Hint: Look at packets #335 and #336. Compare this to DHCP replies we looked at earlier.

Let's work with colorization a bit. Select the **Colorize Packet List** button (near the middle of the toolbar). Now select the **Edit Coloring Rules…** button (3rd from the right) on the toolbar.

Q22. What is the default color scheme given to ICMP error messages?

A22. _____ (color) lettering/numbering on a _____ (color) background

Hint: Look at the **Edit Coloring Rules...** window and find "ICMP errors" in the name column.

Now scroll through the entire list of captured packets (no display filter in place) and see if you find any ICMP error messages. You should find quite a few, all clustered between packet #784 and #894 inclusive.

Take a look at packet #783 (the packet immediately preceding the first ICMP error packet). This is what a Unix version of a "ping" (ICMP echo request) looks like. That is, Unix uses a UDP packet sent to a non-listening region of the port space (typically in the 33000-35000 range) in order to elicit an ICMP error message. If the ICMP error message received is "host unreachable" (e.g., packet #785), the Unix sender knows that no such IP exists. If the ICMP error message received is "port unreachable" (e.g., packet #849), the Unix sender knows that the IP **is** there — thus… ping behavior achieved.

Q23. Looking closely at the IP layer of packet #783, what would you guess is the purpose of this packet?

A23.

Hint: Look at the TTL value.

From the above (**Q23**) analysis, you can understand why an ICMP error due to time to live exceeded in transit (TTL went to zero) is returned to 201.100.1.99 (via packet #784). This is as expected. Did you notice when looking at the DHCP traffic earlier, that we are working with class C (mask = 255.255.255.0) addresses? If so, then you can clearly see that the traceroute sender (201.100.**1**.99) is **not** in the same network as the endpoint of the traceroute (201.100.**6**.98); and thus a TTL = 1 packet will certainly "die" on a router as it tries to make the next "hop" toward the "dot6" network. If you take a quick look at some of the later UDP traffic from 201.100.1.99 to 201.100.6.98, you will see the TTL values steadily incrementing. 1—2—3—4—etc. You may also notice that two packets are sent for each TTL value in this trace, though this aspect of traceroute functionality can be altered depending on which tool one might be using to perform the traceroute.

Q24. Click on packet #784. From this packet we learn that Cisco routers appear to set their initial TTL value to _____ ?

A24.

Hint: Time to live exceeded in transit messages are sent by routers… and the source MAC indicates "Cisco" (specifically, "Cisco_f9:00:21"). Look into packet #784's IP header to find the TTL value. We know that this router is directly connected to the network this traffic was sniffed from, because 1) it is sending the time-exceeded message to the TTL=1 UDP packet we sniffed on this same network, and 2) because the TTL value of this packet where it was captured, is the maximum possible value for an 8-bit value ← another hint for **A24**.

Q25. How do we know for sure that packet #784 (ICMP time to live exceeded in transit) is in direct response to packet #783?

A25.

Hint: Look into the ICMP payload of packet #784. According to the RFC, ICMP error messages should carry the **entire IP header plus the next 8 bytes** of whatever packet elicited the error. Note that this **plus the next 8 bytes** would include/capture both the source and destination port numbers of a UDP or TCP header.

Q26. Which packet (#?) is the first packet coming **back from** the designated endpoint (201.100.6.98) of this traceroute?

A26.

Hint: You could simply scroll down until you see the first unreachable packet from 201.100.6.98 to 201.100.1.99, or you could try the following display filter `ip.src == 201.100.6.98` You'll notice that this filter not only shows packets (nine of them) **from** the desired IP, but also shows any packets (#870, #872, and #874) where this IP address appears in the ICMP payload. For example, click on packet #870 and expand the ICMP protocol section in the middle pane. There you can clearly see "Src: 201.100.6.98" is included in this ICMP error message's payload. Clear that display filter if you tried it, and try this one instead `icmp.type == 3 and icmp.code == 3` You may recall from lecture, and/or by referencing your TCP/IP Reference Guide, that ICMP message *type* 3 is an "unreachable" and that *code* 3 used in conjunction with *type* 3 indicates that the reason for the unreachability was that the sender attempted to connect to a port where no service was listening. You'll notice that this filter gets rid of some of the unwanted traffic (the udp traffic), but still includes some traffic other than what we wanted to see. Last, you may want to try "and'ing" the two above rules together to further restrict the traffic (only six packets would be seen). You may also experiment with other display filter ideas if you're interested. Whichever way you do it… you should see that packet **#840** is the answer.

Deselect the **Colorize Packet List** option by clicking on that button in the toolbar again.

In the display **Filter** window enter "cdp" in the filter string field and then click the **Apply** button. You should now be looking at 15 CDP (Cisco Discovery Protocol) packets in the top pane. CDP is essentially an information sharing protocol that Cisco devices can use to learn about one another. This is yet another example of the "auto switch" being thrown and having technology do more of our work for us. And… yet another example of where an attacker may passively obtain information about your network. Depending on the risk assessment of your network, you may opt to disable CDP on all of your devices and/or not permit it to pass any router or firewall. Disabling CDP is a typical task/recommendation found in router hardening guides.

Q27. Take a quick look at the CDP payload portion of these 15 packets in the middle pane. What two models of Cisco routers are sending CDP information on this network?

A27.

Hint: Look in payload information, or look at the ASCII text in the bottom pane.

Q28. What IOS (Internetwork Operating System) Version is the Cisco 7000 router running?

A28.

Hint: Look in payload information in the middle pane, or look at the ASCII text in the bottom pane. Find the number (10.X) that follows the word "Version".

In the display **Filter** window enter "rip" in the filter string field then click the **Apply** button. You should now be looking at 19 RIPv1 and 1 RIPv2 packets.

Compare the route information held by router 201.100.1.1 (in packet #763) to the route information held by router 201.100.1.2 (in packet #732).

Q29. Which of the two routers has the shortest "hop" connection to the 201.100.5.0 network?

A29.

Hint: Look into the RIP payload of each of these two packets.

Where is the network **mask** information in these RIPv1 responses? Ans. Nowhere. RIPv1 predated the usage of CIDR (Classless Interdomain Routing, aka "classless"), so masks are assumed based on the classful *first octet rule* that you may recall from our Section 1 material. Take a look at the payload of the single RIPv2 packet (packet #729). Though this is a request packet for which there is no corresponding response, you can clearly see that RIPv2 does include a **netmask** as a protocol field.

In the display **Filter** window enter "dns" in the filter string then click the **Apply** button. You should now be looking at 24 DNS packets. Select packet #1.

Q30. Is this DNS client requesting iterative or recursive lookup service?

A30.

Hint: Look in the DNS payload and expand the flags field. You should find what you would expect of a *client* request (recall the *iterative* vs *recursive* query issue and how excessive use of recursive behavior has the effect of moving the work toward the "top" of the DNS hierarchy… which makes this an issue for the 'A' of the "CIA Triad".

Q31. What was the 1st IP address provided back to the client by this DNS server?

A31.

Hint: Look in the payload of the DNS response packet (packet #2) in the "Answers" section. You may also notice that this information was provided from the cache of some name server, and that the current time to live value (as it sits in that server's cache, aging out) is 4 minutes, 17 seconds.

<Q32 removed>

Let's look at some Web traffic.

In the display **Filter** window enter "http" in the filter string then click the **Apply** button. Select the first packet in the top pane (packet #6). Now select **Analyze → Follow TCP Stream** from the menu bar. You will be presented with a new window that shows the entire packet breakdown for this "conversation". Close this new window. Notice that a display filter was automatically created for you that specifies the "socket-pair" used to identify all packets in this "conversation" (aka TCP stream).

Now, in the top pane, with this display filter in place, we also see all other related TCP traffic that is involved in this specific client-server exchange of http traffic.

Notice what Wireshark has done with the Initial Sequence Numbers (ISNs) from both the client and server; reset them to zero in order to display a "relative" seq# to you the user. This is merely a convenience feature. The actual sequence numbers in the actual packets do not change.

Q33. What was the actual ISN chosen by the client in packet #6?

A33.

Hint: Look into the layer 4 information in the middle pane and select (highlight) the sequence number field, then look at the bottom pane to see the actual ("raw") value in hexadecimal that is highlighted.

Q34. What Web browser is the client using?

A34.

Hint: Look in the http payload of packet #6

Look at packets #6 through #20 in this Web "conversation". For you "old-schoolers", you may recall that http creates a separate session (i.e., a new socket-pair) for **each** page/item that is downloaded from a Web server. This was true for http version 1.0, but we are looking at http version 1.1. Version 1.1 enhances the performance of http by keeping the session alive for a "while". Thus if the user (or his Web browser) is issuing multiple requests to the same server, the speed is improved as no new TCP handshakes are needed.

Q35. What is the keep-alive value requested by the client?

A35.

Hint: Look at payload of packet #6. Though not stated, the units are seconds, so if you do the math, you should come up with 5 minutes. The whole implementation of http keep-alives is NOT very standardized (similar to the TOS—Type Of Service—field we discussed with IP). If you want to read a short description of this try http.//www.io.com/~maus/HttpKeepAlive.html or Google "http keep-alives"

You will notice that the client issues **two** *get* requests. One (packet #6) is generated by the user entering a fqdn (or URL) into his/her Web browser's address window. The other *get* request (packet #8) is issued by the user's Web browser. Look at the last part of the payload of packet #7, and you will see where the Web **server** tells the Web **browser** about a path and a file name to obtain an image file ("/images/logo.gif"). An fqdn (e.g., www.google.com) plus a path to a file (e.g., /images/logo.gif) is what yields a URL (e.g., www.google.com/images/logo.gif). The Web browser sees this URL and initiates a second *get* request in order to obtain it.

Q36. Is packet #9 an IP fragment?

A36.

Hint: Check the MF flag and fragment offset value

The "reassembled PDU" comment may have thrown you on question #36. What the comment means is that the logo.gif file was too big to carry in one packet, so it was split up (**at layer 4**) to be carried in separate (thus no need to fragment) packets. ← From this… you hopefully have a more clear distinction between what is meant by *segmentation*, as opposed to *fragmentation*.

Q37. How many total packets did it take to deliver the entire logo.gif file?

A37.

Hint: Count them in the top pane, or look at the layer 4 information provided in the middle pane for packet #19.

If you look briefly at packets #11, #14, #17, and #20, you can see that the client is busy acknowledging the data as it arrives. You will notice, by looking at the sequence numbers, total length numbers, and the ack numbers, that all packets were accounted for in each case.

Q38. Did the Web Server "set" a cookie on this client, or did the client voluntarily provide a previously stored cookie to the Web Server?

A38.

Hint: Look into payload of packets #6 and #7 for signs of a "cookie"… then determine which (client or server) is sending that cookie by noting the port numbers used (well-known vs ephemeral).

Don't mistake that lack of "readability" of the cookie for a lack of information. The server will know how to interpret the syntax of the cookie. It is quite possible that the cookie you see is not the actual information at all, but simply a "pointer" to what might be a much larger database entry that is available to the server; e.g., the user's account information or "shopping cart".
Clear the display filter.

Filter again for "http" traffic.

Select packet #92, then select **Analyze → Follow TCP Stream** from the menu bar. Close the new window that pops up.

Q39. How many total packets are involved in this session?

A39.

Hint: Click on **Statistics → Summary** from the menu bar, then look near the bottom right of the report page that pops up. Make sure you look at the **Displayed** column rather than the **Captured** column. You should see also that the conversation was only ~26 seconds in duration.

Q40. What resource was being requested?

A40.

Hint: Look in payload of packet #92 (the first "GET" request following the 3-way handshake)

Q41. What kind of Web server is being used in this session?

A41.

Hint: Look into payload of packet #94

Note that in packet #94, the Web server is challenging the user for a password before providing access.

Q42. The user provided an incorrect password followed later by the correct password. What was the incorrect, and the correct password?

A42.

Hint: Look in the payloads of the two "GET" packets (#96 and #99) for authorization information, and then look to see how the server (www.clarkson.edu) responded to each (packets #97 and #100) . Note these were obscured by some well known algorithm (not a hash) on the wire… but Wireshark was "smart" enough to un-do this common obscuration and display the actual username and password. You can see what I'm talking about by comparing the "Authorization" information found in the middle pane to the corresponding information found in the ASCII section of the bottom pane. Not so secure… but better than nothing. Remember. "risk management" vice "risk avoidance". We won't protect "national secrets" this way, but we may use it for very low-risk applications.

Q43. What document does this user request from this web site?

A43.

Hint: Look at packet #111. Note that it takes quite a few packets to completely deliver this document, and since the document is in plaint text, you can actually read the document in the ASCII portion of the bottom pane.

Let's take a look at some SSH traffic. **Clear** the display filter, the put "ssh" in the display filter and **Apply** it. Then select the topmost packet (packet #948), then select **Analyze → Follow TCP Stream** from the menu bar to display the entire SSH session (30 total packets). Close the "Follow TCP Stream" window that pops up.

You will notice that this is SSHv1. It has some vulnerabilities that have largely been addressed with later versions of SSH. You are about to learn of v1's major vulnerability.

Q44. Which is true about this SSH session?

a. Only the client authenticates to the server
b. Only the server authenticates to the client
c. Both server and client authenticate one another
d. Neither server nor client can be sure of the other's identity

Hint: Look at all of the 30 packets. Based on what you've learned in CS3600 and this course, do you see anything that provides "proof" of identity? Specifically, look at packet #951 and notice "Server: Public Key" in the info column of the top pane. Click on packet #951 and completely expand its SSH payload in the middle pane, and notice that it is carrying a "naked" public key. If you received a "naked" **public key** in the mail along with a note that said "this is Bob's public key", use it to encrypt things to Bob. Would you be confident using that key to encrypt sensitive data to Bob? ← Please say "no"!

Q45. Who (client or server) chooses the session key?

A45.

Hint: Identify client and server based on IPs and who initiated the session, or… just read the helpful info provided in the Info column of the middle pane.

Q46. Could an attacker "on the wire" between the two "legitimate" endpoints (client and server) set himself up for a man-in-the-middle (MITM) situation given this handling/management of keys?

A46. ~~**yes**~~

Hint: Think through the possible attack scenario/sequence. Attacker intercepts the public key sent by the server and inserts his own public key. The client, not knowing he's actually using the attacker's public key, encrypts his (client's) chosen session key with the attacker's public key. The attacker intercepts this and decrypts the session key using his private key. The attacker re-encrypts this session key with the server's public key and forwards it back to the server. The attacker now has a copy of the session key and neither of the legitimate endpoints is aware of this.

Q47. TCP uses a 3-way "hello" handshake. How many packets are involved in a TCP "goodbye" hug?

A47.

Hint: Look at the last packets in this session. The answer to is; either 3 or 4 as there are two ways to *gracefully* terminate a TCP session (abrupt ways include TCP resets, app-layer aborts, and simple time-out of the session). A 3-way occurs when the two end-points are both finished sending data, and a 4-way occurs when the client is finished sending data but the server is not. Look to see which version has occurred in this instance.

Now **Clear** your display filter and create a new one for "ssl" and **Apply** it, then select **Analyze → Follow TCP Stream** from the menu bar to display the entire SSL/TLS session (21 total packets). Close the window that pops up.

Also select **View → Expand All** from the menu bar. By expanding all of the layer information in the middle pane, you can see just how complex TLS (Transport Layer Security) is. Note TLS is the newer name for SSL.

In the SSL payload of packet #927 you will see the client offering 26 different combinations of secure protocols (called a "cipher spec" or a "cipher suite") from which the server can choose.

Q48. What **Cipher Suite** did the server choose to use?

A48.

Hint: Look into payload of packet #929

Q49. <Question removed>

In packet #929 you can see the X.509 certificate structure in the payload.

Q50. Who is the CA for this certificate (i.e., who is the "signer" and issuer)?

A50.

Hint: Look for "commonName=" in the "**issuer**" section of the payload. You may also notice this particular signing CA office appears to be in South Africa (countryName=ZA).

Q51. Who/what does this certificate bind a pubic key **to**? (that **is** the purpose of a certificate after all!)

A51.

Hint: Look for "commonName=" in the "**subject**" section of the payload.

Close this capture file, and open the file **`pax2forWSLab.cap`**

In this capture we will see what a **TCP session hijack** looks like. This is where your solid understanding of TCP should come in handy.

With the new capture file now open, you can select **Statistics → Summary** to see that this capture includes 98,125 packets. Select packet #1, then select **Analyze → Follow TCP Stream** from the menu bar to display this entire telnet session. Close the window that pops up. If you again select **Statistics → Summary** you will see that this entire capture file is of just one single telnet session.

Hidden among this session are two IP-spoofed packets; more specifically, a "spoof" that results in an attacker taking on the IP address of another host on that same network (vice simply changing his/her IP address to anything other than what it was "assigned").

Q52. Can you think of how we might identify these packets?

A52.

Hint: If an IP spoofer (impersonator) changes his IP to the IP of X, but does not also change his MAC to that of X, then it may be possible to detect this. Local traffic with IP X would be "seen" coming **from** two different MACs. It would not be "seen" going **to** two different MACs as the client or router sending **to** the IP would only be sending to the (single) MAC it had learned from a previous (legitimate) ARP. Note that the sniffer would have to be in the same network as the impersonat**or** and impersonat**ee** to detect this; because otherwise (sniffer on different network than impersonat**or**/**ee**) the MACs on the sniffed frames would belong to the network's serving router(s) (think about "hops"!). Only if both the impersonat**or** and impersonat**ee** are on the same network as the sniffer will you detect multiple MACs per one IP; which would be an indicator of IP spoofing going on.

In case you could not figure out **Q52**, here is the answer. Let's first check to see if anyone has spoofed the source IP address used in packet #1. Add the below to the display filter that is already in place (i.e., the one automatically created when you selected **Follow TCP Stream** above) and click **Apply**

. . . and ip.src_host == 192.168.1.103 and eth.src != 00.06.5b.d5.1e.e7

You should see that there are two spoofed packets; #**461** and #**656**. We can note that the impersonator is spoofing the client… not the server. If you like you may alter the filter rule extension you just made so that it will check to see if anyone has spoofed the .101 (server) IP as well. To save you some time, I can tell you that this is not the case. **Clear** the current display filter.

Let's select **Go → Go to Packet** and go to packet #**461**. When packet #**461** is selected, type **Ctrl+M** to mark this packet for easy eyeball reference. Do the same for packet #**656**, then scroll back to the top of the packet capture.

You can now scroll down--pages at a time--and briefly see that a lot of "normal" telnet traffic is occurring, UNTIL… we arrive at our attacker's (i.e., the impersonator) packet #**461** (which should be marked). Now observe the "hijack".

Q53. What **ack** number does the server send the client in packet #460?

A53.

Q54. What **sequence** number does the **attacker** provide in packet #**461**?

A54.

Do you see that the attacker is providing the expected sequence number?

Q55. How many bytes of payload did the **attacker** send in packet #**461**?

A55.

Do you see the havoc this wreaks with the remainder of the session? In packet #462 the server acks the attacker's last transmission of data with the expected "ack 243". Then starting in packet #463 and continuing for over 95,000 packets the client and server argue about this mismatch in expected sequence numbers. Then in packet #**656** the attacker leaves his calling card in the .profile file on the telnet server. Notice that the attacker is "*synchronized*" with the server (seq# & ack#); while the client cannot recover from the desynchronization forced by the attacker. You may also note, by inspecting the telnet payload of packet #**461**, that the attacker issued several backspaces (\b\b\b\b\b\b\b\b) followed by two newlines (\n\n); an attempt to erase the last command the client had issued, and provide a newline with which to enter his own command to the telnet server. An easy DoS attack, would be for the attacker to simply issue a RST (reset) packet, or a FIN/ACK packet to the server. But de-synchronizing, as was done in this example, results in more "blood loss" for this DoS attack.

That's it!