Homework+Lab 3: Due at start of class on 08-Apr.

Please complete all six homework problems, and the twenty-one lab problems.

Homework Problems

1. Why, according to Freeman[1], is it dangerous to equate bit-rate and bandwidth? Please provide a descriptive answer of at least several sentences that demonstrates your understanding.

2. Stallings Problem 3.15 from the "Problems" section, not the "Review Questions".

3. Stallings Problem 3.16.

4. Stallings Problem 4.2.

5. Stallings Problem 4.11.

6. Stallings Problem 4.14.

[1] "Bits, Symbols, Bauds, and Bandwidth" by Roger L. Freeman which is available from within the WPI network at `http://goo.gl/fSHo2`

Lab 3: Exploring HTTP

[adapted from J. Kurose and K.W. Ross]
With a basic understanding of Wireshark and DNS, we're now ready to use Wireshark to investigate protocols in operation. In this lab, we'll explore several parts of the HTTP protocol: the basic GET/response interaction, HTTP message formats, retrieving large HTML files, retrieving HTML files with embedded objects, and HTTP authentication and security. Before beginning these labs, please read Section 24.3 of the text.

Part I: The Basic HTTP GET/response interaction

Let's begin our exploration of HTTP by downloading a very simple HTML file – one that is very short, and contains no embedded objects. Do the following:

- Start up your web browser.
- Start up the Wireshark packet sniffer, as described in the Introductory lab (but don't yet begin packet capture). Enter "http" (just the letters, not the quotation marks) in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window. We're only interested in the HTTP protocol here, and don't want to see the clutter of all captured packets.
- Wait a bit more than one minute (we'll see why shortly), and then begin Wireshark packet capture.
- Enter the following to your browser
  `http://spinlab.wpi.edu/wireshark/lab3part1.html`
  Your browser should display the very simple, one-line HTML file.
- Stop Wireshark packet capture.

Your Wireshark window should look similar to the window shown in Figure 1.

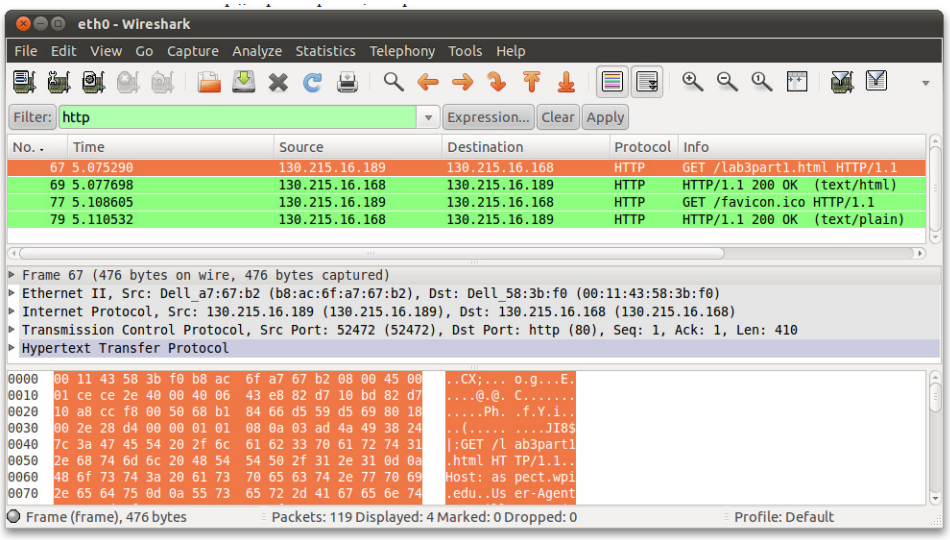

Figure 1: Wireshark display after `lab3part1.html` has been retrieved by your browser

The example in Figure 1 shows in the packet-listing window that two HTTP messages were captured: the GET message (from your browser to the `spinlab.wpi.edu` web server) and the response message from the server to your browser. The packet-contents window shows details of the selected message (in this case the HTTP GET message, which is highlighted in the packet-listing window). Recall that since the HTTP message was carried inside a TCP segment, which was carried inside an IP datagram, which was carried within an Ethernet frame, Wireshark displays the Frame, Ethernet, IP, and TCP packet information as well. We want to minimize the amount of non-HTTP data displayed (we're interested in HTTP here, and will be investigating these other protocols is later labs), so make sure the boxes at the far left of the Frame, Ethernet, IP and TCP information have a plus sign (which means there is hidden, undisplayed information), and the HTTP line has a minus sign (which means that all information about the HTTP message is displayed).

Note: You should ignore any HTTP GET and response for favicon.ico. If you see a reference to this file, it is your browser automatically asking the server if it (the server) has a small icon file that should be displayed next to the displayed URL in your browser. We'll ignore references to this file in this lab.

By looking at the information in the HTTP GET and response messages, answer the following questions. When answering the following questions, you should print out the GET and response messages (see the introductory Wireshark lab for an explanation of how to do this) and indicate where in the message you've found the information that answers the following questions.

1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?

2. What languages (if any) does your browser indicate that it can accept to the server?

3. What is the IP address of your computer? Of the `spinlab.wpi.edu` server?

4. What is the status code returned from the server to your browser?

5. When was the HTML file that you are retrieving last modified at the server?

6. How many bytes of content are being returned to your browser?

7. By inspecting the raw data in the packet content window, do you see any headers within the data that are not displayed in the packet-listing window? If so, name one.

Part II: The HTTP CONDITIONAL GET/response interaction

Recall from Chapter 24 of the text, that most web browsers perform object caching and thus perform a conditional GET when retrieving an HTTP object. Before performing the steps below, make sure your browser's cache is empty. (To do this under Firefox, select Tools→Clear Private Data, or for Internet Explorer, select Tools→Internet Options→Delete File; these actions will remove cached files from your browser's cache.) Now do the following:

- Start up your web browser, and make sure your browser's cache is cleared, as discussed above.
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser
  `http://spinlab.wpi.edu/wireshark/lab3part2.html`
  Your browser should display another very simple HTML file.

- Quickly enter the same URL into your browser again (or simply select the refresh button on your browser). **This is an important step.**
- Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.

Answer the following questions:

8. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE" line in the HTTP GET?
9. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?
10. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE:" line in the HTTP GET? If so, what information follows the "IF-MODIFIED-SINCE:" header?
11. What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.

Part III: Retrieving Long Documents

In our examples thus far, the documents retrieved have been simple and short HTML files. Let's next see what happens when we download a long HTML file. Do the following:

- Start up your web browser, and make sure your browser's cache is cleared, as discussed above.
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser
  `http://spinlab.wpi.edu/wireshark/lab3part3.html`
  Your browser should display the rather lengthy US Bill of Rights.
- Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed.

In the packet-listing window, you should see your HTTP GET message, followed by a multiple-packet response to your HTTP GET request. This multiple-packet response deserves a bit of explanation. Recall from Chapter 24 that the HTTP response message consists of a status line, followed by header lines, followed by the entity body. In the case of our HTTP GET, the entity body in the response is the entire requested HTML file. In our case here, the HTML file is rather long, and at about 4500 bytes is too large to fit in one TCP packet. The single HTTP response message is thus broken into several pieces by TCP, with each piece being contained within a separate TCP segment. Each TCP segment is recorded as a separate packet by Wireshark, and the fact that the single HTTP response was fragmented across multiple TCP packets is indicated by the "Continuation" phrase displayed by Wireshark. We stress here that there is no "Continuation" message in HTTP.

Answer the following questions:

12. How many HTTP GET request messages were sent by your browser (again, ignoring requests for favicon.ico)?

13. How many data-containing TCP segments were needed to carry the single HTTP response?

14. What is the status code and phrase associated with the response to the HTTP GET request?

15. Are there any HTTP status lines in the transmitted data associated with a TCP- induced "Continuation"?

Part IV: HTML Documents with Embedded Objects

Now that we've seen how Wireshark displays the captured packet traffic for large HTML files, we can look at what happens when your browser downloads a file with embedded objects, i.e., a file that includes other objects (in the example below, image files) that are stored on another server(s). Do the following:

- Start up your web browser, and make sure your browser's cache is cleared, as discussed above.
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser
  `http://spinlab.wpi.edu/wireshark/lab3part4.html`
  Your browser should display a short HTML file with two images. These two images are referenced in the base HTML file. That is, the images themselves are not contained in the HTML; instead the URLs for the images are contained in the downloaded HTML file. Your browser will have to retrieve these logos from the indicated web sites. Our school's logo is retrieved from the www.wpi.edu web site. The cartoon image is stored at the `imgs.xkcd.com` server.
- Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed.

Answer the following questions:

17. How many HTTP GET request messages were sent by your browser? To which Internet addresses were these GET requests sent?

18. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.

Part V: HTTP Authentication

Finally, let's try visiting a web site that is password-protected and examine the sequence of HTTP message exchanged for such a site. I have placed a file in a password protected area of the spinlab website. The username and password were sent previously by class email. Let's access this "secure" password-protected site. Do the following:

- Make sure your browser's cache is cleared, as discussed above, and close down your browser. Then, start up your browser
- Start up the Wireshark packet sniffer Enter the following URL into your browser
  `http://spinlab.wpi.edu/wireshark/prot/lab3part5.html`
  Type the requested user name and password into the pop up box.
- Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.

Now let's examine the Wireshark output. You might want to first read up on HTTP authentication by reviewing the easy-to-read material on "HTTP Access Authentication Framework" at `http://frontier.userland.com/stories/storyReader$2159`

Answer the following questions:

19. What is the server's response (status code and phrase) in response to the initial HTTP GET message from your browser?
20. When your browser's sends the HTTP GET message for the second time, what new field is included in the HTTP GET message?

The username and password that you entered are encoded in the string of characters immediately following the "Authorization: Basic " header in the client's HTTP GET message (and terminated by a "\r\n" which correspond to hexadecimal codes 0d 0a). While it may appear that your username and password are encrypted, they are simply encoded in a format known as Base64 format.

21. Copy the string of text that appears after "Authorization: Basic ", and enter it into the Base64 decoder here – `http://www.motobit.com/util/base64-decoder-encoder.asp`. Make sure to click on "decode". To what ASCII string does the Base64 string decode to? (Hint: you should see the username and password for the course website embedded in the string).

Since anyone can download a tool like Wireshark and sniff packets (not just their own) passing by their network adaptor, and anyone can translate from Base64 to ASCII (as you just did), it should be clear to you that simple passwords on WWW sites are not secure unless additional measures are taken. Fortunately, most banks and financial institutions use a more secure form of http called https. We will discuss this briefly toward the end of the semester.