CSCI 4220 Lab 8

Lab 8: HTTP in Wireshark

This lab is adapted from materials provided as a supplement to Wireshark Lab: "*Computer Networking: A Top-Down Approach, 7th ed., J.F. Kurose and K.W. Ross*". The objective is to use Wireshark to observe several HTTP behaviors. Keep in mind that many sites will use HTTPS/TLS and that Wireshark will not be able to decrypt the contents of those sessions, so this approach is often of limited use without a lot of extra work.

You will collect several traces throughout this exercise. Make sure to save each one as a new file so that you can refer back to them and have them for submission. There are 17 questions in total but they are all worksheet-style short questions like in the FTP lab.

To start with, you will look at a very straightforward request. Open Wireshark and start a trace on your internet-connected interface. To help filter out unwanted packets, you should use a filter of **http** . Keep in mind that filters of **property != value** will probably not do what you want, you will want to use **!(property == value)** instead. Filters support boolean logic so you can use || and **&&** to make compound filters. Visit this URL, then stop the trace and answer 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 gaia.cs.umass.edu server?
4. What is the status code returned from the server to your browser? (ignore the favicon.ico request)
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.

Next, you will examine how conditional HTTP requests can differ. Start a new trace, visit this URL, and then without stopping the trace, visit the same URL again (refreshing the page is fine). Stop the trace. To keep the filter relevant, you may want to use:
http && !(http.request.uri == "/favicon.ico") && !(http.response.code == 404) .

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.

Next, you will request a file that is much larger than the previous two files (approximately 4500 bytes) and see how HTTP handles delivering such a long request. Start a new trace and visit this URL. Once the page has loaded, stop the trace. Keep in mind that “TCP segment of a reassembled PDU” or “Continuation” is a note made by Wireshark - there is no such annotation in the actual HTTP data! You should not use a filter that specifies **http** but you may want to filter based on **ip.addr** .

12. How many HTTP GET request messages did your browser send? Which packet number in the trace contains the GET message for the Bill or Rights?
13. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?
14. What is the status code and phrase in the response?
15. How many data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights?

Finally, you will look at a page that has embedded elements. This is how most webpages are designed. Start a new trace and visit this URL . Once the page has loaded, you can stop the trace.

16. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?
17. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.

Submission

Submit a single PDF, `Lab8_answers.pdf` that contains your responses to all the questions (`1-7` from part 1, `8-11` from part 2, `12-15` from part 3, and `16-17` from part 4). Also submit your captures from the four parts as `Lab8_Part1.pcapng`, `Lab8_Part2.pcapng`, `Lab8_Part3.pcapng`, and `Lab8_Part4.pcapng`