Assignment 4: Packet Capturing Using Wireshark

This assignment should be completed as a group of TWO. Prepare the report, and submit it to Ms. Shilpa Budhkar (b.shilpa@iitg.ernet.in) at RS Lab, Department of CSE.

Submission Deadline: March 31, 2014.

Part A: Initials

1. Download and install wireshark on your laptop/desktop from here:
http://www.wireshark.org/

2. Read the following page to make sure you have capture privileges:
http://wiki.wireshark.org/CaptureSetup/CapturePrivileges

Run wireshark

3. Start capturing traffic: “Capture/Interfaces”

4. While continuing to capture traffic start your browser and direct it to www.iitg.ernet.in

5. Stop capturing.

6. Filter the packets that belong to the http session between your host and the IITG web server.

7. Take a screenshot of this result. How many packets were transmitted from the IITG web server to your client in this.

Part B: HTTP

In this experiment, we’ll explore several aspects of the HTTP protocol: the basic GET/response interaction, HTTP message formats, retrieving large HTML files, and retrieving HTML files with embedded objects.

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 first assignment (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://202.141.80.32/~sukumar/cs349/HTTP-wireshark-file1.html
- Your browser should display the very simple, one-line HTML file. Stop Wireshark packet capture.

You can see in the packet-listing window that two HTTP messages are captured: the GET message (from your browser to the 202.141.80.32 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). Recall that since the HTTP message is carried inside a TCP segment, which is again carried inside an IP datagram, that is 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, 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).

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 (to do so, select Print from the Wireshark File command menu, and select the "Selected Packet Only" and "Print as displayed" radial buttons, and then click OK) 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?

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.

II. The HTTP CONDITIONAL GET/response Interaction:

Recall, 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://202.141.80.32/~sukumar/cs349/HTTP-wireshark-file2.html Your browser should display a very simple four-line HTML file.
- Quickly enter the same URL into your browser again (or simply select the refresh button on your browser)
- 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:

1. 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?

2. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?

3. 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?

4. 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

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://202.141.80.32/~sukumar/cs349/HTTP-wireshark-file3.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 that the HTTP response message consists of a status line, followed by header lines, followed by a blank line, 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 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:

1. How many HTTP GET request messages were sent by your browser?

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

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

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

Part C: UDP

Since UDP is a streamlined, no-thrills protocol - simple and sweet - this part of this lab is quick and simple. Start capturing packets in Wireshark and then do something that will cause your host to send and receive several UDP segments (hint: you might want to recall that DNS messages are carried inside UDP segments, but you can do anything you want to ensure that there are UDP segments in your captured trace). Whenever possible, when answering a question you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. Annotate the printout to explain your answer. To print a packet, use File -> Print, choose Selected packet only, and select the minimum amount of packet detail that you need to answer the question.

a. Select one packet. From this packet, determine how many fields there are in the UDP header. (Do not look in the textbook! Answer these questions directly from what you observe in the packet trace.) Name these fields as they are named in the Wireshark display of segment fields.

b. What are the source and destination port numbers, in both decimal and hexadecimal format. (Hint: the hexadecimal format is given in the data in the bottommost panel in the wireshark display, and so it's easier just to read it out

from there rather than converting the decimal number to hex).

c. What is the value in the Length field in both decimal and hexadecimal format. What is the meaning of this value (i.e., this value is the length of what?)

d. What is the protocol number for UDP? Give your answer in both hexadecimal and decimal notation. (To answer this question, you'll need to look into the IP header.)

e. Examine a pair of UDP packets in which the first packet is sent by your host and the second packet is a reply to the first packet. Describe the relationship between the port numbers in the two packets.

Part D: TCP

We'll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You'll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text of Alice in Wonderland), and then transfer the file to a Web server using the HTTP POST method. We're using the POST method rather than the GET method as we'd like to transfer a large amount of data from your computer to another computer. Of course, we'll be running Wireshark during this time to obtain the trace of the TCP segments sent and received from your computer. Do the following:

- Bypass the IP address 202.141.80.32 to your web browser proxy settings.
- Start up your web browser.
- Go to http://202.141.80.32/~sukumar/cs349/ALC/alice.txt and retrieve an ASCII copy of Alice in Wonderland. Store this file somewhere on your computer.
- Next go to http://202.141.80.32/~sukumar/cs349/TCP-wireshark-file1.html
- Use the Browse button in this form to enter the name of the file (full path name) on your computer containing Alice in Wonderland (or do so manually). Don't yet press the "Upload alice.txt file" button.
- Now start up Wireshark and begin packet capture (Capture -> Start) and then press OK on the Wireshark Packet Capture Options screen (we'll not need to select any options here).
- Returning to your browser, press the "Upload alice.txt file" button to upload the file to the server. Once the file has been uploaded, a short congratulations message will be displayed in your browser window.
- Stop Wireshark packet capture.

Before analyzing the behavior of the TCP connection in detail, let's take a high level view of the trace.

- First, filter the packets displayed in the Wireshark window by entering "tcp" (lowercase, no quotes, and don't forget to press return after entering!) into the display filter specification window towards the top of the Wireshark window. What you should see is series of TCP segments between your computer and 202.141.80.32. You should see the initial three-way handshake containing a SYN message. You should see an HTTP POST message inside a TCP segment and a series of tcp segments being sent from your computer to 202.141.80.32, as well as ACKs being returned from 202.141.80.32. (**Remember to bypass the IP 202.141.80.32 to your web browser proxy settings. Guess what will happen if you do not do so!**).

Answer the following questions for the TCP segments:

1. Print out a captured packet and indicate where you see the information that answers the following:
   a) What is the IP address and TCP port number used by your client computer (source) to transfer the file to 202.141.80.32?
   b) What is the IP address and TCP port number used by the server?

2. Print out a captured packet and indicate where you see the information that answers the following:
   a) What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and 202.141.80.32?
   b) What is it in the segment that identifies the segment as a SYN segment?

3. Print out a captured packet and indicate where you see the information that answers the following:
   a) What is the sequence number of the SYNACK segment sent by 202.141.80.32 to the client computer in reply to the SYN?
   b) What is the value of the ACKnowledgement field in the SYNACK segment?
   c) How did 202.141.80.32 server determine that value?
   d) What is it in the segment that identifies the segment as a SYNACK segment?

4. Print out a captured packet and indicate where you see the information that answers the following: What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you'll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a "POST" within its DATA field. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection.
   a) What are the sequence numbers of the first six segments in the TCP connection (including the segment containing the HTTP POST)? At what time was each segment sent?
   b) When was the ACK for each segment received?
   c) Do you see evidence of the use of cumulative ACKs in your trace? Explain.
   d) Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments?
   e) What is the length of each of the first six TCP segments?
   f) What is the minimum amount of available buffer space advertised at the receiver for the entire trace? Does the lack of receiver buffer space ever throttle the sender?
   g) Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question? How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment? Explain.