# Project – Wireshark Lab – HTTP

Name : Spencer Vital

# Key Concepts / Outcomes

|  |
| --- |
| Having gotten our feet wet with the Wireshark packet sniffer in the introductory lab, we’re now ready to use Wireshark to investigate protocols in operation. In this lab, we’ll explore several aspects 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. |

# Tasks to complete

|  |
| --- |
| 1. 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:   1. Start up your web browser. 2. 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). 3. Wait a bit more than one minute (we’ll see why shortly), and then begin Wireshark packet capture. 4. Enter the following to your browser <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html> Your browser should display the very simple, one-line HTML file. 5. Stop Wireshark packet capture.   Your Wireshark window should look similar to the window shown in Figure 1. If you are unable to run Wireshark on a live network connection, you can download a packet trace that was created when the steps above were followed.[[1]](#footnote-1)    **Figure 1:** Wireshark Display after http://gaia.cs.umass.edu/wireshark-labs/ HTTP-wireshark-file1.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 gaia.cs.umass.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 OK 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 or a right-pointing triangle (which means there is hidden, undisplayed information), and the HTTP line has a minus sign or a down-pointing triangle (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 pesky file in this lab.).  By looking at the information in the HTTP GET and response messages, answer the following questions using a RED FONT.   1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?   The browser is running HTTP version 1.1. The server is also running HTTP version 1.1.   1. What languages (if any) does your browser indicate that it can accept to the server?   The browser accepts en-US language to the server.   1. What is the IP address of your computer? Of the gaia.cs.umass.edu server?   The IP address of my computer is 192.168.1.102. The IP address of the gaia.cs.umass.edu server is 128.119.245.12.   1. What is the status code returned from the server to your browser?   The status code returned from the server to my browser is 200.   1. When was the HTML file that you are retrieving last modified at the server?   The HTML file that I am retrieving was last modified at the server at Feb 20, 2023, 06:59:01.   1. How many bytes of content are being returned to your browser?   127 bytes.   1. 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.   No, there are no headers within the data that are not displayed in the packet-listing window.   1. Find the initial HTTP handshake sequence. Take a screen grab of the two frames and insert it here:     In your answer to question 5 above, you might have been surprised to find that the document you just retrieved was last modified within a minute before you downloaded the document. That’s because (for this particular file), the gaia.cs.umass.edu server is setting the file’s last-modified time to be the current time, and is doing so once per minute. Thus, if you wait a minute between accesses, the file will appear to have been recently modified, and hence your browser will download a “new” copy of the document.  2. The HTTP CONDITIONAL GET/response interaction  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 Recent History* and check the Cache box, 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://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html> Your browser should display a very simple five-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. * (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-2 packet trace to answer the questions below; see footnote 1. This trace file was gathered while performing the steps above on one of the author’s computers.)   Answer the following questions using a RED FONT:   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? Take a screen grab and insert it here:      1. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell? Take a screen grab and insert it here:   The server responded with a 200 OK status code and returned the HTML file in 471 bytes of data.     1. 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? Take a screen grab and insert it here:   The time when the file was last modified.     1. 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? Take a screen grab showing the content and insert it here:   Since the file has not been modified since the time specified in the header, the server responded with a 304 Not Modified status code and no content was returned.  Graphical user interface, text, application  Description automatically generated  3. 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://gaia.cs.umass.edu/wireshark-labs/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. * (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-3 packet trace to answer the questions below; see footnote 1. This trace file was gathered while performing the steps above on one of the author’s computers.)   In the packet-listing window, you should see your HTTP GET message, followed by a multiple-packet TCP response to your HTTP GET request. This multiple-packet response deserves a bit of explanation.  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. In recent versions of Wireshark, Wireshark indicates each TCP segment as a separate packet, and the fact that the single HTTP response was fragmented across multiple TCP packets is indicated by the “TCP segment of a reassembled PDU” in the Info column of the Wireshark display. Earlier versions of Wireshark used the “Continuation” phrase to indicate that the entire content of an HTTP message was broken across multiple TCP segments.. We stress here that there is no “Continuation” message in HTTP!  Answer the following questions using a RED FONT:   1. 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? Take a screen grab of the frame and insert it here: 2. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request? Take a screen grab of both the Get and Response frames and insert it here: 3. What is the status code and phrase in the response? Take a screen grab and insert it here: 4. How many data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights? Take a screen grab showing the segments and insert it here:   4. 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://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.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. As discussed in the textbook, your browser will have to retrieve these logos from the indicated web sites. Our publisher’s logo is retrieved from the gaia.cs.umass.edu web site. The image of the cover for our 5th edition (one of our favorite covers) is stored at the caite.cs.umass.edu server. (These are two different web servers inside cs.umass.edu). * 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 using a RED FONT:   1. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent? Take a screen grab of all the GET requests and insert here: (hint, you may need to sort, or filter, to show them all together…)   6 HTTP GET Request.  gaia.cs.umass.edu/favicon.ico  <http://dl.delivery.mp.microsoft.com/filestreamingservice/files/20fa4f26-6a42-4eee-8146-e2276afa6897/pieceshash>     1. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain your answer and then take a screen grab showing what you think happened:   Based on the order of the HTTP response messages, it appears that the images were downloaded in parallel.    5 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. The URL  http://gaia.cs.umass.edu/wireshark-labs/protected\_pages/HTTP-wireshark-file5.html is password protected. The username is “wireshark-students” (without the quotes), and the password is “network” (again, without the quotes). So 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://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.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.   Answer the following questions using a RED FONT:   1. What is the server’s response (status code and phrase) in response to the initial HTTP GET message from your browser? Take a screen grab and insert here:   The server's response to the initial HTTP GET message from the browser was a "401 Unauthorized" response.     1. When your browser’s sends the HTTP GET message for the second time, what new field is included in the HTTP GET message? Take screen grabs of both GET requests and insert here:   The second HTTP GET message sent by the browser includes an "Authorization" field in the request header, containing the encoded username and password for authentication.     1. Explain the difference between an HTTP frame and an HTTPS frame.   An HTTP frame is an unencrypted frame that is sent over the network in plaintext, while an HTTPS frame is an encrypted frame that is sent over the network using SSL/TLS encryption.   1. Find the packet that contains the user name and password that you entered above. What do you see? Take a screen grab of the user name and password that was captured by Wireshark and insert it here:   The user name and password are included in the Authorization field of the HTTP GET request message.    Note: The username (wireshark-students) and password (network) that you entered are encoded in the string of characters (d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms=) following the “Authorization: Basic” header in the client’s HTTP GET message. While it may appear that your username and password are encrypted, they are simply encoded in a format known as Base64 format. The username and password are *not* encrypted! To see this, go to <http://www.motobit.com/util/base64-decoder-encoder.asp> and enter the base64-encoded string d2lyZXNoYXJrLXN0dWRlbnRz and decode. *Voila!* You have translated from Base64 encoding to ASCII encoding, and thus should see your username! To view the password, enter the remainder of the string Om5ldHdvcms= and press decode. 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 (you just did it!), it should be clear to you that simple passwords on WWW sites are not secure unless additional measures are taken. |
|  |

# What to submit to LMS

|  |
| --- |
| Download and edit this file directly from the LMS.  Answer each question USING RED TEXT inline and then upload this file back to the LMS. |

1. [↑](#footnote-ref-1)