Wireshark Lab 9: HTTP

Introduction

In this lab, we'll explore how web browsers communicate with web servers using the **HTTP protocol** and analyze real-world HTTP traffic using **Wireshark**.

We will learn about:

- How HTTP GET requests work
- How browsers handle long files and embedded content
- How **authentication** happens during HTTP sessions
- The difference between **authentication** and **authorization**

Authentication vs Authorization (Quick Recap)

Concept	Description
Authentication	Proving your identity (e.g., username & password)
Authorization	What you are allowed to access (permissions)

In simple terms:

- Authentication: "Who are you?"
- Authorization: "What can you do?"

In this lab, we'll see how **basic authentication** works in HTTP and why it is not secure unless used with HTTPS.

Authentication Protocols and Standards

These protocols are used to **implement authentication** in software and networks:

1. HTTP Basic Authentication

- Used in web applications (often insecure without HTTPS).

- Sends the **username and password encoded in Base64** in the HTTP header.
- Easily viewable in Wireshark unless HTTPS is used.

Header example:

Authorization: Basic dXNlcjpwYXNz
Decoded: user:pass

Risk: If not used with HTTPS, credentials can be intercepted in plaintext.

2. HTTPS (Secure HTTP)

- HTTP over TLS (encryption layer).
- Encrypts the entire communication, including credentials.
- Prevents password sniffing in Wireshark.

Safe for transmitting authentication data.

3. SSH (Secure Shell)

- Used for secure remote login to servers.
- Supports **password-based or key-based** authentication.
- All traffic is encrypted, including the login credentials.
- In Wireshark, you **cannot see** passwords due to encryption.

4. Kerberos

- A network authentication protocol using **tickets**.
- Ensures **mutual authentication**: both client and server verify each other.
- Often used in enterprise networks (e.g., Windows domains).
- Prevents passwords from being sent over the network.

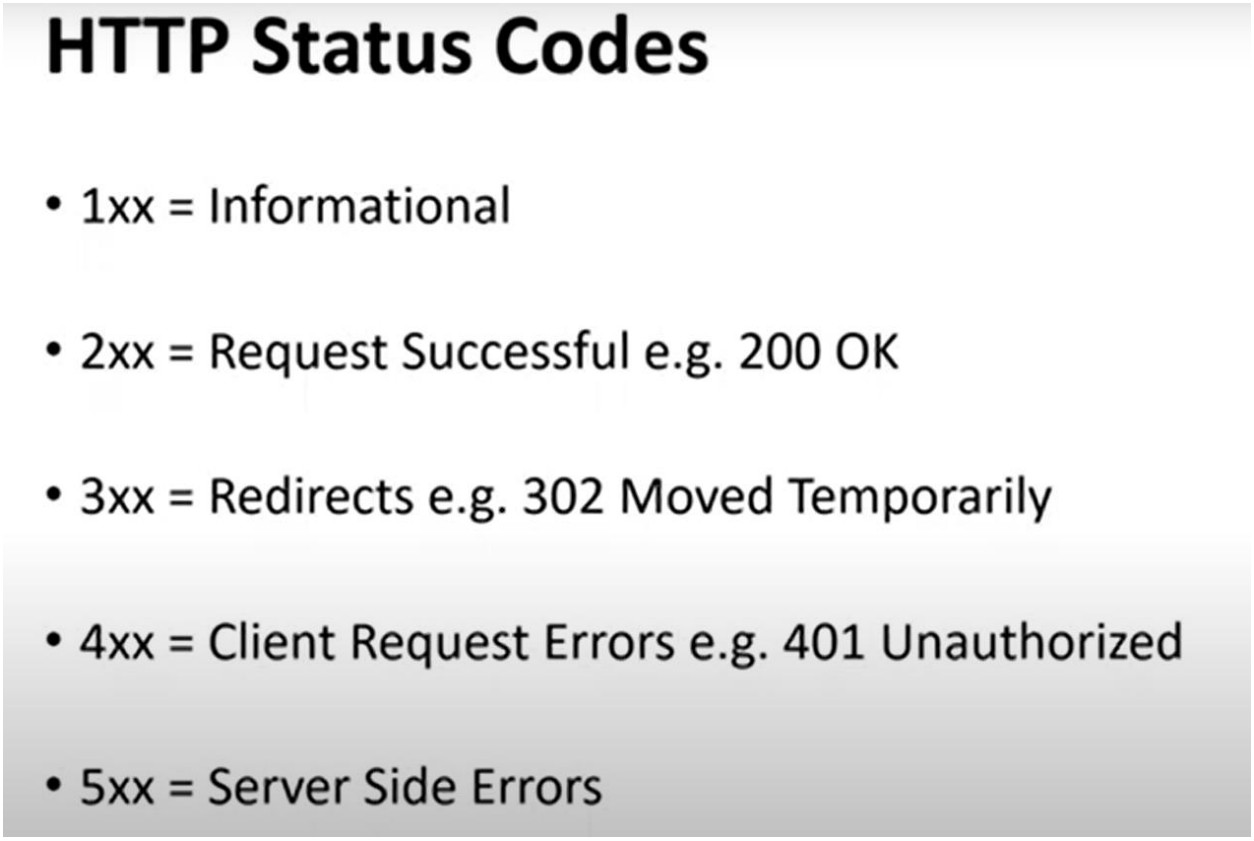
- Packets can be seen in Wireshark under:

nginx

CopyEdit

kerberos

- You'll see **ticket-granting requests**, not passwords.



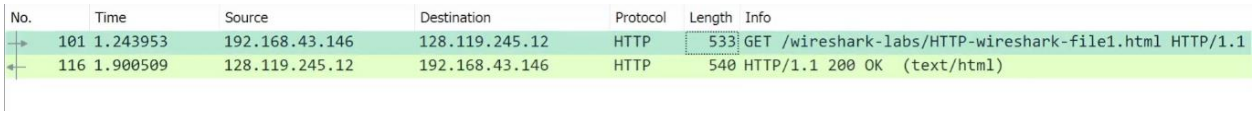
The Basic HTTP GET/response interaction

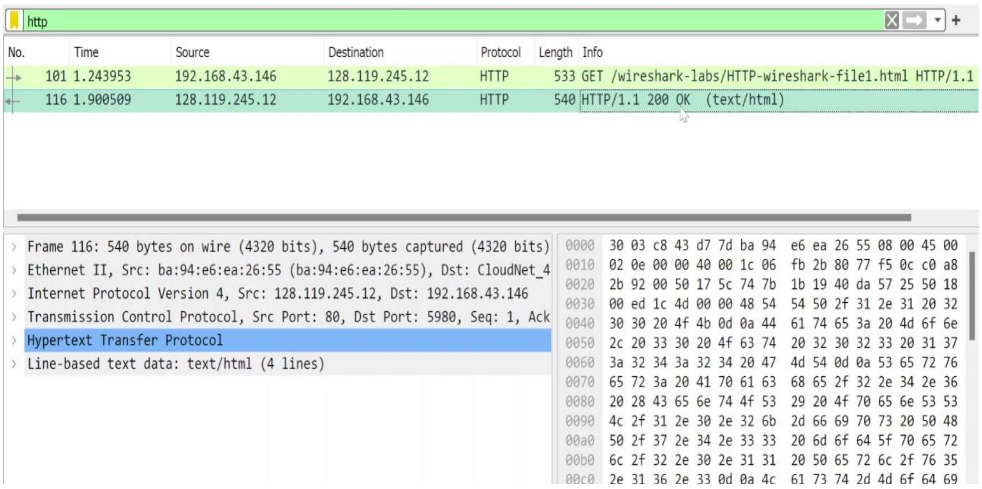
Let's begin our exploration of HTTP by downloading a very simple HTML file - one that is very short, and contains no embedded objects. Do the following:

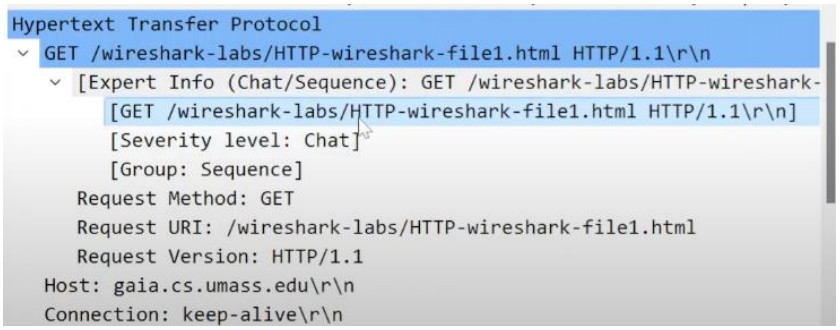
1. Start up your web browser.
2. Start up the Wireshark packet sniffer, as described in the Introductory lab (but don't yet begin packet capture). Enter "http" (just the letters, not the quotation marks) in the display-filter-specification window, so that only captured HTTP messages will be displayed 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]

[1] Download the zip file http://gaia.cs.umass.edu/wireshark-

labs/wireshark-traces.zip and extract the file http-ethereal-trace-1. The traces in this zip file were collected by Wireshark running on one of the author'

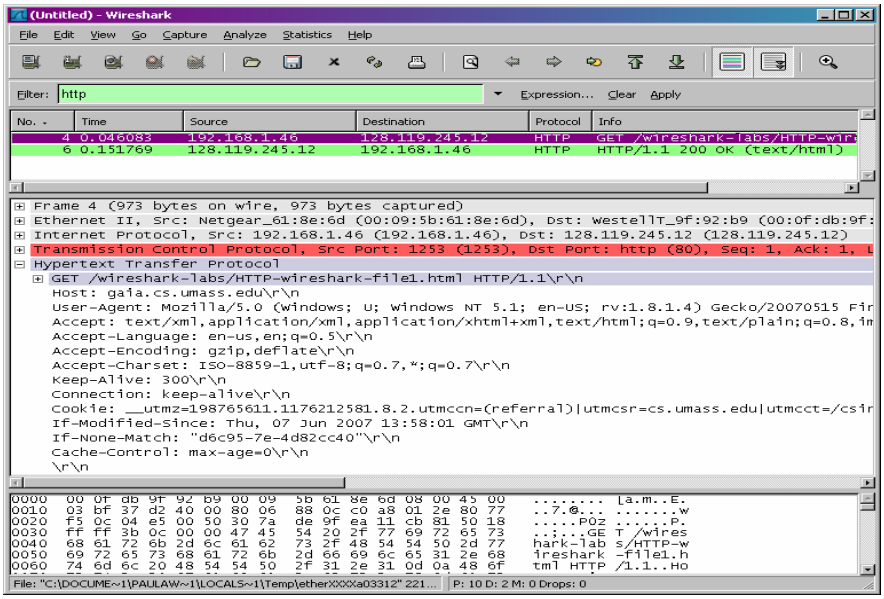


Figure 1: Wireshark Display after http://gaia.cs.umass.edu/wireshark-labs/ HTTP- wireshark-file1.html has been retrieved by your browser

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 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

---

computers, while performing the steps indicated in the Wireshark lab. Once you have downloaded the trace, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the http-ethereal-trace-1 trace file. The resulting display should look just like Figure 1.

displayed (we're interested in HTTP here, and will be investigating these

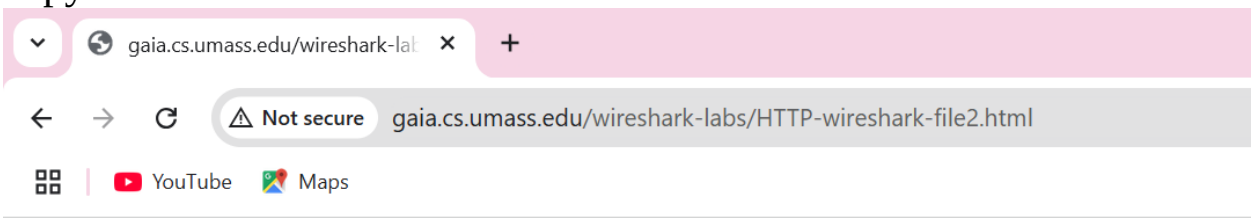
other protocols is later labs), so make sure the boxes at the far left of the Frame, Ethernet, IP and TCP information have a plus sign (which means there is hidden, undisplayed information), and the HTTP line has a minus sign (which means that all information about the HTTP message is displayed).
(*Note:* 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 pesky 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 gaia.cs.umass.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.

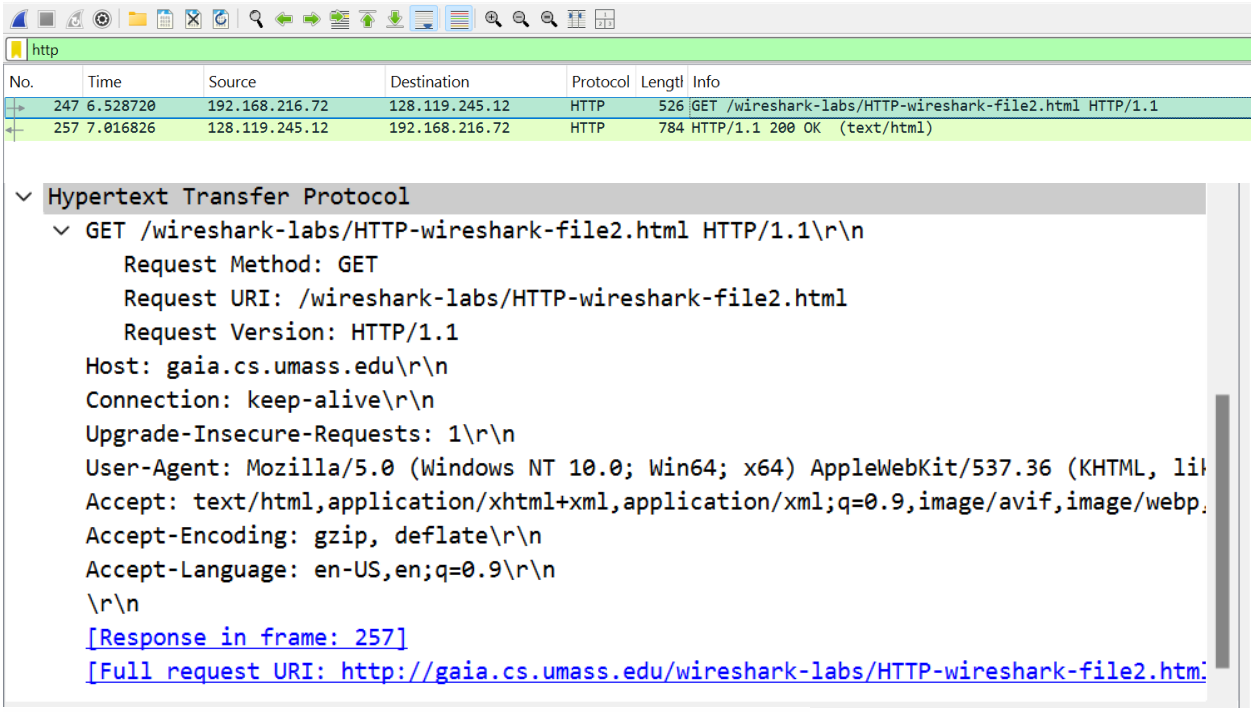
In your answer to question 5 above, you might have been surprised to find that the document you just retrieved was last modified within a minute before you downloaded the document. That's because (for this particular file), the gaia.cs.umass.edu server is setting the file's last-

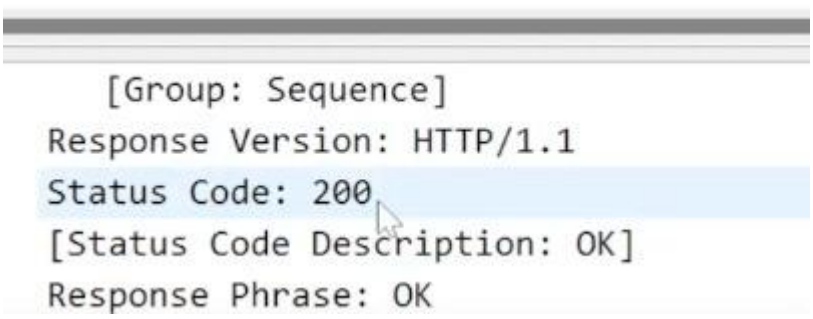
modified time to be the current time, and is doing so once per minute. Thus, if you wait a minute between accesses, the file will appear to have been recently modified, and hence your browser will download a “new” copy of the document.



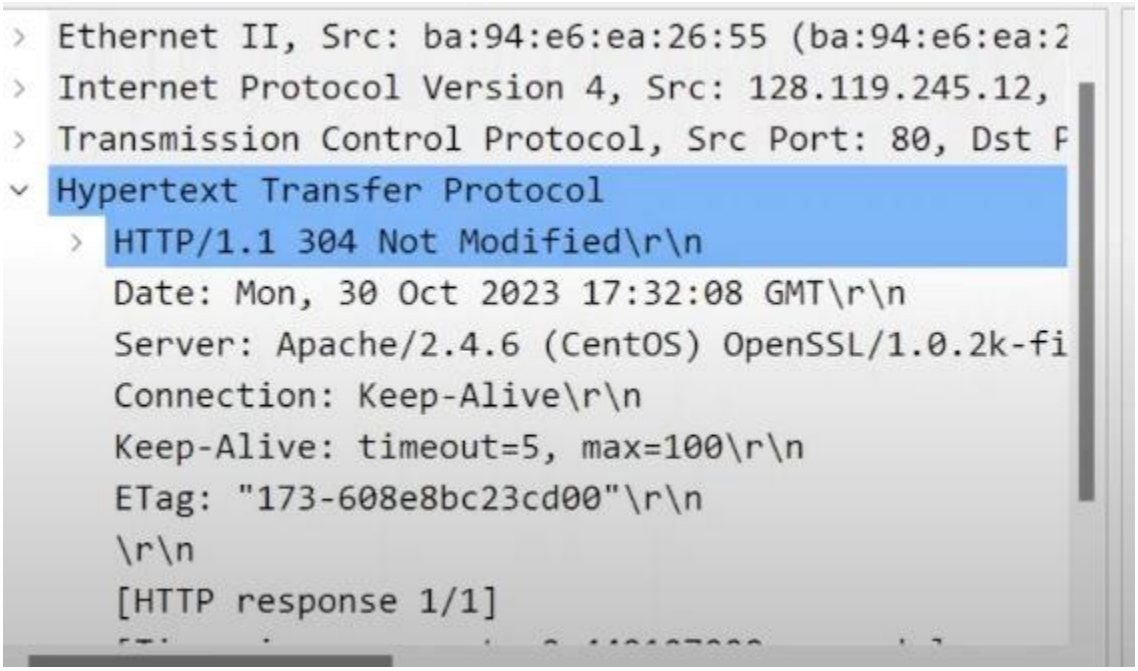
Congratulations again! Now you've downloaded the file lab2-2.html.
This file's last modification date will not change.

Thus if you download this multiple times on your browser, a complete copy
will only be sent once by the server due to the inclusion of the IN-MODIFIED-SINCE
field in your browser's HTTP GET request to the server.





Again make request and check status changed



The HTTP CONDITIONAL GET/response interaction

Recall from Section 2.2.6 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 Netscape 7.0, select *Edit->Preferences->Advanced->Cache* and clear the memory and disk cache.

For 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:

1. Start up your web browser, and make sure your browser's cache is cleared, as discussed above.
2. Start up the Wireshark packet sniffer
3. 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.
4. Quickly enter the same URL into your browser again (or simply select the refresh button on your browser)
5. 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.

6. (*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:

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.

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:

1. Start up your web browser, and make sure your browser's cache is cleared, as discussed above.
2. Start up the Wireshark packet sniffer
3. 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.
4. Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed.
5. (*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 response to your HTTP GET request. This multiple-packet response deserves a bit of explanation. Recall from Section 2.2 (see Figure 2.9 in the text) 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 (see Figure 1.20 in the text). 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"?

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:

1. Start up your web browser, and make sure your browser's cache is cleared, as discussed above.
2. Start up the Wireshark packet sniffer
3. 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 www.aw-bc.com web site. The image of our book's cover is stored at the manic.cs.umass.edu server.

1. Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed.
2. (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-4 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:

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

browser? To which Internet addresses were these GET requests sent?

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

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.
  - (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-5 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.)

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:

a. What is the server's response (status code and phrase) in response to the initial HTTP GET message from your browser?
b. 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 (wirehsark-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.securitystats.com/tools/base64.php and enter the base64-encoded string d2lyZXNoYXJrLXN0dWRlbnRz and press 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.