# TCN 5080 - Project 2

## Description:

This project will give you hands-on experience with the Wireshark network protocol analyzer, by investigating the behavior of SSL. Wireshack is a popular open-source tool for network troubleshooting, analysis, and protocol development. You will need to capture and analyze a trace of SSL packets, and answer a series of questions, to demonstrate your understanding of the SSL protocol. Note that Wireshark has already been installed in the graduate lab. You can also install Wireshark on your personal computer.

You are provided with two PDF documents:

- "Wireshark Lab: SSL" is the manual for the SSL project. You need to follow the instructions and answer the questions.
- "WiresharkLab: Getting Started" is an introduction to the tool. Follow the exercises for practice.

Specifically, you need to:

- Submit your captured trace file from the SSL lab exercise (Section 1 describes how this can be done).
- Use your captured trace file (not the one provided by the manual) to answer Q3-5.
- Use the trace file provided by the manual (see the second paragraph of Section 1) to answer the remaining questions.
- Please provide the Wireshark screenshots to support your answers. If you do not provide the supporting information, you will receive only partial credit.

This is an independent project. You must work on your own.

## Submission Guide:

Submit your trace file (not the trace file provided) and the word/pdf document through Moodle.

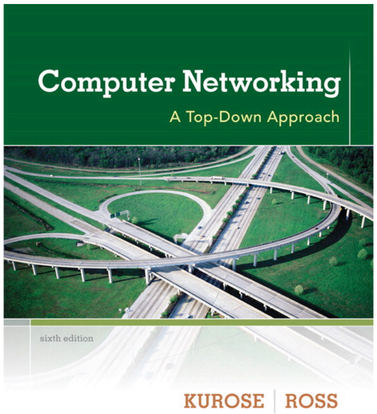
## Grading Criteria:

| | |
|---|---|
| Captured trace | 15% |
| Q1, Q5, Q10 | 10% each |
| Q2-4, Q6-9, Q11-14 | 5% each |

# Wireshark Lab: Getting Started v6.0

Supplement to *Computer Networking: A Top-Down Approach, 6th ed.,* J.F. Kurose and K.W. Ross

*"Tell me and I forget. Show me and I remember. Involve me and I understand."* Chinese proverb



One's understanding of network protocols can often be greatly deepened by "seeing protocols in action" and by "playing around with protocols" – observing the sequence of messages exchanged between two protocol entities, delving down into the details of protocol operation, and causing protocols to perform certain actions and then observing these actions and their consequences. This can be done in simulated scenarios or in a "real" network environment such as the Internet. In the Wireshark labs you'll be doing in this course, you'll be running various network applications in different scenarios using your own computer (or you can borrow a friends; let me know if you don't have access to a computer where you can install/run Wireshark). You'll observe the network protocols in your computer "in action," interacting and exchanging messages with protocol entities executing elsewhere in the Internet. Thus, you and your computer will be an integral part of these "live" labs. You'll observe, and you'll learn, by doing.

In this first Wireshark lab, you'll get acquainted with Wireshark, and make some simple packet captures and observations.

The basic tool for observing the messages exchanged between executing protocol entities is called a **packet sniffer**. As the name suggests, a packet sniffer captures ("sniffs") messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a *copy* of packets that are sent/received from/by application and protocols executing on your machine.

Figure 1 shows the structure of a packet sniffer. At the right of Figure 1 are the protocols (in this case, Internet protocols) and applications (such as a web browser or ftp client) that normally run on your computer. The packet sniffer, shown within the dashed rectangle in Figure 1 is an addition to the usual software in your computer, and consists of two parts. The **packet capture library** receives a copy of every link-layer frame that is sent from or received by your computer. Recall from the discussion from section 1.5 in the text (Figure 1.24[1]) that messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable. In Figure 1, the assumed physical media is an Ethernet, and so all upper-layer protocols are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives you all messages sent/received from/by all protocols and applications executing in your computer.

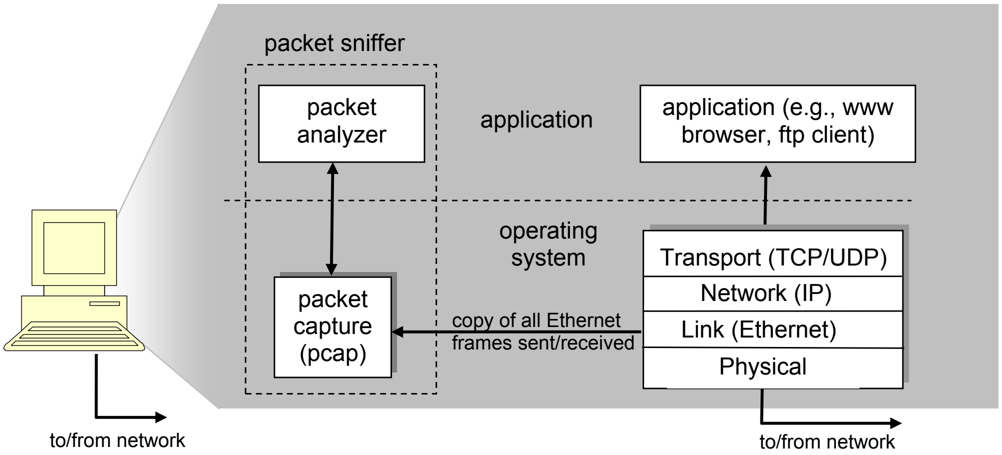


**Figure 1:** Packet sniffer structure

The second component of a packet sniffer is the **packet analyzer**, which displays the contents of all fields within a protocol message. In order to do so, the packet analyzer must "understand" the structure of all messages exchanged by protocols. For example, suppose we are interested in displaying the various fields in messages exchanged by the HTTP protocol in Figure 1. The packet analyzer understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so that it can extract the TCP segment within the IP datagram. Finally, it understands the TCP segment structure, so it can extract the HTTP message contained in the TCP segment. Finally, it understands the HTTP protocol and so, for example, knows that the first bytes of an HTTP message will contain the string "GET," "POST," or "HEAD," as shown in Figure 2.8 in the text.

We will be using the Wireshark packet sniffer [http://www.wireshark.org/] for these labs, allowing us to display the contents of messages being sent/received from/by protocols at different levels of the protocol stack. (Technically speaking, Wireshark is a packet analyzer that uses a packet capture library in your computer). Wireshark is a free network protocol analyzer that runs on Windows, Linux/Unix, and Mac computers. It's an ideal packet analyzer for our labs – it is stable, has a large user base and well-documented support that includes a user-guide (http://www.wireshark.org/docs/wsug_html_chunked/),

[1] References to figures and sections are for the 6th edition of our text, *Computer Networks, A Top-down Approach, 6th ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2012.*

man pages (http://www.wireshark.org/docs/man-pages/), and a detailed FAQ (http://www.wireshark.org/faq.html), rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface. It operates in computers using Ethernet, serial (PPP and SLIP), 802.11 wireless LANs, and many other link-layer technologies (if the OS on which it's running allows Wireshark to do so).

## Getting Wireshark

In order to run Wireshark, you will need to have access to a computer that supports both Wireshark and the *libpcap* or *WinPCap* packet capture library. The *libpcap* software will be installed for you, if it is not installed within your operating system, when you install Wireshark.  See http://www.wireshark.org/download.html for a list of supported operating systems and download sites

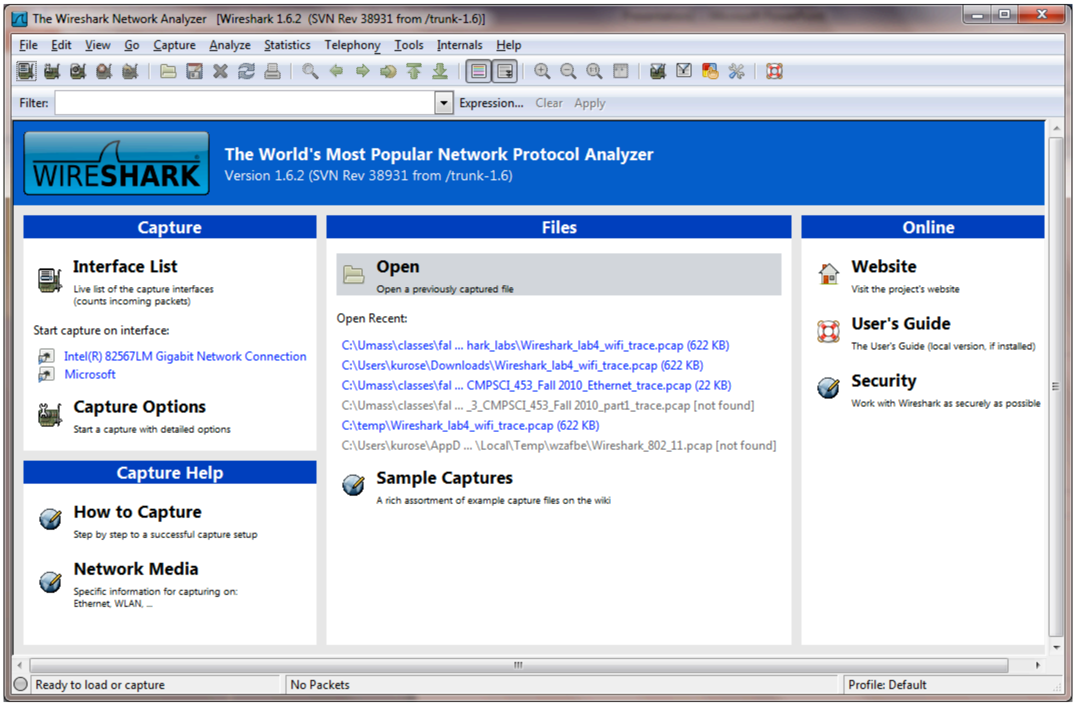
Download and install the Wireshark software:

- Go to http://www.wireshark.org/download.html and download and install the Wireshark binary for your computer.

The Wireshark FAQ has a number of helpful hints and interesting tidbits of information, particularly if you have trouble installing or running Wireshark.

## Running Wireshark

When you run the Wireshark program, you'll get a startup screen, as shown below:



The World's Most Popular Network Protocol Analyzer
Version 1.6.2 (SVN Rev 38931 from /trunk-1.6)

**Interface List**
Live list of the capture interfaces (counts incoming packets)

Start capture on interface:
Intel(R) 82567LM Gigabit Network Connection
Microsoft

**Capture Options**
Start a capture with detailed options

**How to Capture**
Step by step to a successful capture setup

**Network Media**
Specific information for capturing on: Ethernet, WLAN, ...

**Open**
Open a previously captured file

Open Recent:
C:\Umass\classes\fal ... hark_labs\Wireshark_lab4_wifi_trace.pcap (622 KB)
C:\Users\kurose\Downloads\Wireshark_lab4_wifi_trace.pcap (622 KB)
C:\Umass\classes\fal ... CMPSCI_453_Fall 2010_Ethernet_trace.pcap (22 KB)
C:\Umass\classes\fal ... _3_CMPSCI_453_Fall 2010_part1_trace.pcap [not found]
C:\temp\Wireshark_lab4_wifi_trace.pcap (622 KB)
C:\Users\kurose\AppD ... \Local\Temp\wzafbe\Wireshark_802_11.pcap [not found]

**Sample Captures**
A rich assortment of example capture files on the wiki

**Website**
Visit the project's website

**User's Guide**
The User's Guide (local version, if installed)

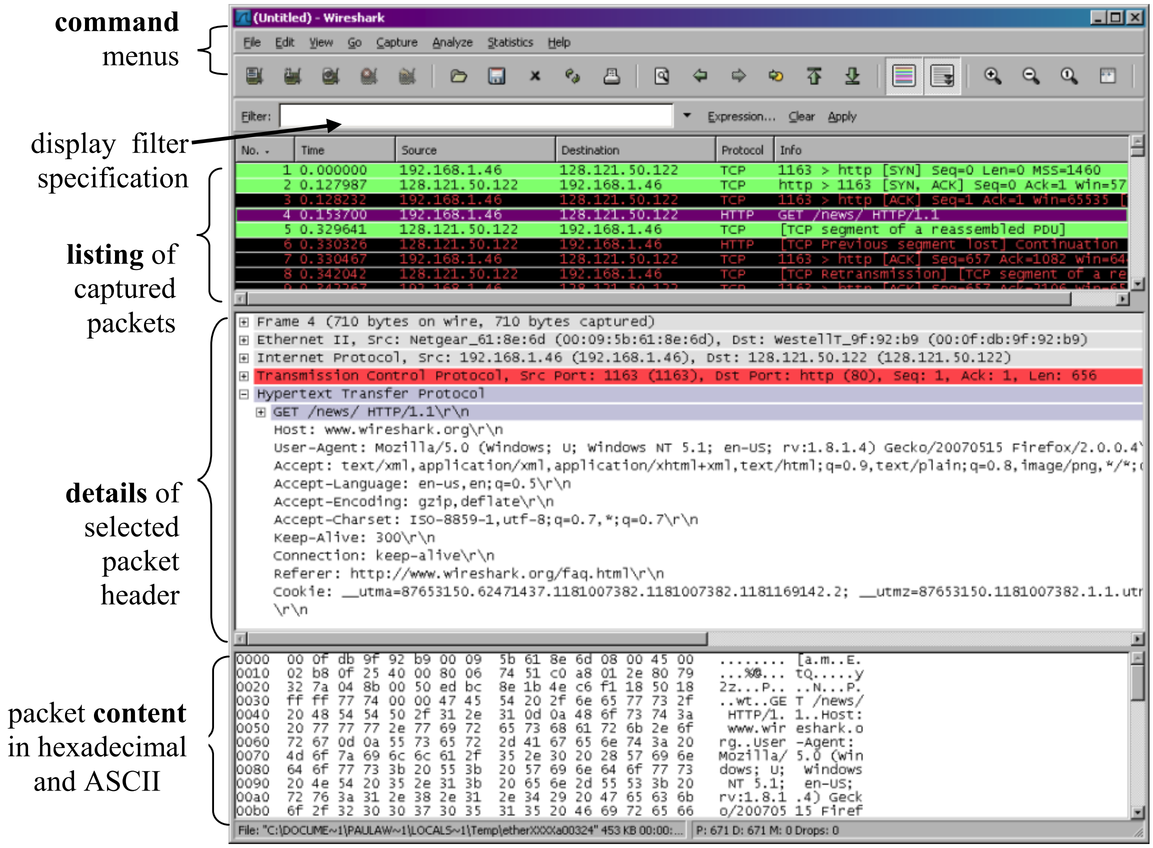
**Security**
Work with Wireshark as securely as possible

Ready to load or capture | No Packets | Profile: Default

**Figure 2:** Initial Wireshark Screen

Take a look at the upper left hand side of the screen – you'll see an "Interface list". This is the list of network interfaces on your computer. Once you choose an interface, Wireshark will capture all packets on that interface. In the example above, there is an Ethernet interface (Gigabit network Connection) and a wireless interface ("Microsoft").

If you click on one of these interfaces to start packet capture (i.e., for Wireshark to begin capturing all packets being sent to/from that interface), a screen like the one below will be displayed, showing information about the packets being captured. Once you start packet capture, you can stop it by using the Capture pull down menu and selecting Stop.



**command** menus

display filter specification

**listing** of captured packets

**details** of selected packet header

packet **content** in hexadecimal and ASCII

```
No. -   Time        Source            Destination       Protocol  Info
      1 0.000000    192.168.1.46      128.121.50.122    TCP       1163 > http [SYN] Seq=0 Len=0 MSS=1460
      2 0.127987    128.121.50.122    192.168.1.46      TCP       http > 1163 [SYN, ACK] Seq=0 Ack=1 Win=57
      3 0.128232    192.168.1.46      128.121.50.122    TCP       1163 > http [ACK] Seq=1 Ack=1 Win=65535 [
      4 0.153700    192.168.1.46      128.121.50.122    HTTP      GET /news/ HTTP/1.1
      5 0.329641    128.121.50.122    192.168.1.46      TCP       [TCP segment of a reassembled PDU]
      6 0.330326    128.121.50.122    192.168.1.46      HTTP      [TCP Previous segment lost] Continuation
      7 0.330467    192.168.1.46      128.121.50.122    TCP       1163 > http [ACK] Seq=657 Ack=1082 Win=64
      8 0.342042    128.121.50.122    192.168.1.46      TCP       [TCP Retransmission] [TCP segment of a re
```

```
⊞ Frame 4 (710 bytes on wire, 710 bytes captured)
⊞ Ethernet II, Src: Netgear_61:8e:6d (00:09:5b:61:8e:6d), Dst: WestellT_9f:92:b9 (00:0f:db:9f:92:b9)
⊞ Internet Protocol, Src: 192.168.1.46 (192.168.1.46), Dst: 128.121.50.122 (128.121.50.122)
⊞ Transmission Control Protocol, Src Port: 1163 (1163), Dst Port: http (80), Seq: 1, Ack: 1, Len: 656
⊟ Hypertext Transfer Protocol
  ⊞ GET /news/ HTTP/1.1\r\n
    Host: www.wireshark.org\r\n
    User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
    Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;
    Accept-Language: en-us,en;q=0.5\r\n
    Accept-Encoding: gzip,deflate\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    Keep-Alive: 300\r\n
    Connection: keep-alive\r\n
    Referer: http://www.wireshark.org/faq.html\r\n
    Cookie: __utma=87653150.62471437.1181007382.1181007382.1181169142.2; __utmz=87653150.1181007382.1.1.utr
    \r\n
```

```
0000  00 0f db 9f 92 b9 00 09  5b 61 8e 6d 08 00 45 00   ........ [a.m..E.
0010  02 b8 0f 25 40 00 80 06  74 51 c0 a8 01 2e 80 79   ...%@... tQ.....y
0020  32 7a 04 8b 00 50 ed bc  8e 1b 4e c6 f1 18 50 18   2z...P.. ..N...P.
0030  ff ff 77 74 00 00 47 45  54 20 2f 6e 65 77 73 2f   ..wt..GE T /news/
0040  20 48 54 54 50 2f 31 2e  31 0d 0a 48 6f 73 74 3a    HTTP/1. 1..Host:
0050  20 77 77 77 2e 77 69 72  65 73 68 61 72 6b 2e 6f    www.wir eshark.o
0060  72 67 0d 0a 55 73 65 72  2d 41 67 65 6e 74 3a 20   rg..User -Agent:
0070  4d 6f 7a 69 6c 6c 61 2f  35 2e 30 20 28 57 69 6e   Mozilla/ 5.0 (Win
0080  64 6f 77 73 3b 20 55 3b  20 57 69 6e 64 6f 77 73   dows; U;  Windows
0090  20 4e 54 20 35 2e 31 3b  20 65 6e 2d 55 53 3b 20    NT 5.1;  en-US;
00a0  72 76 3a 31 2e 38 2e 31  2e 34 29 20 47 65 63 6b   rv:1.8.1 .4) Geck
00b0  6f 2f 32 30 30 37 30 35  31 35 20 46 69 72 65 66   o/200705 15 Firef
```

File: "C:\DOCUME~1\PAULAW~1\LOCALS~1\Temp\etherXXXXa00324" 453 KB 00:00:... P: 671 D: 671 M: 0 Drops: 0

**Figure 3:** Wireshark Graphical User Interface, during packet capture and analysis

The Wireshark interface has five major components:

- The **command menus** are standard pulldown menus located at the top of the window. Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data, and exit the Wireshark application. The Capture menu allows you to begin packet capture.

- The **packet-listing window** displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is *not* a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest-level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.
- The **packet-header details window** provides details about the packet selected (highlighted) in the packet-listing window. (To select a packet in the packet-listing window, place the cursor over the packet's one-line summary in the packet-listing window and click with the left mouse button.). These details include information about the Ethernet frame (assuming the packet was sent/received over an Ethernet interface) and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the plus minus boxes to the left of the Ethernet frame or IP datagram line in the packet details window. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be expanded or minimized. Finally, details about the highest-level protocol that sent or received this packet are also provided.
- The **packet-contents window** displays the entire contents of the captured frame, in both ASCII and hexadecimal format.
- Towards the top of the Wireshark graphical user interface, is the **packet display filter field,** into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (and hence the packet-header and packet-contents windows). In the example below, we'll use the packet-display filter field to have Wireshark hide (not display) packets except those that correspond to HTTP messages.

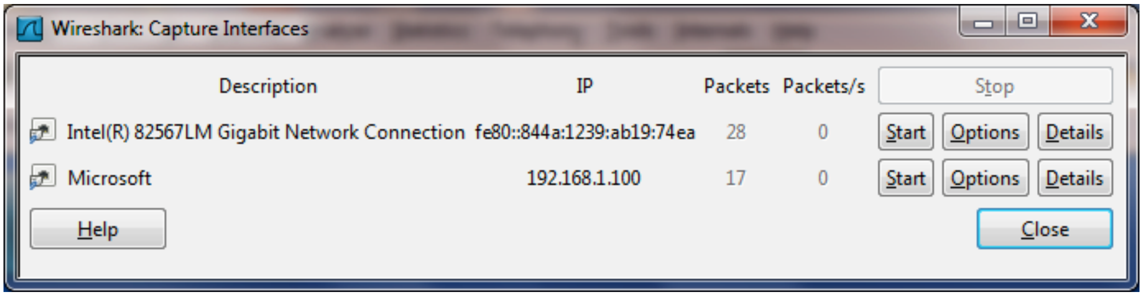
## Taking Wireshark for a Test Run

The best way to learn about any new piece of software is to try it out! We'll assume that your computer is connected to the Internet via a wired Ethernet interface. Indeed, I recommend that you do this first lab on a computer that has a wired Ethernet connection, rather than just a wireless connection. Do the following

1. Start up your favorite web browser, which will display your selected homepage.

2. Start up the Wireshark software. You will initially see a window similar to that shown in Figure 2. Wireshark has not yet begun capturing packets.

3. To begin packet capture, select the Capture pull down menu and select *Interfaces.* This will cause the "Wireshark: Capture Interfaces" window to be displayed, as shown in Figure 4.



| Description | IP | Packets | Packets/s |
|---|---|---|---|
| Intel(R) 82567LM Gigabit Network Connection | fe80::844a:1239:ab19:74ea | 28 | 0 |
| Microsoft | 192.168.1.100 | 17 | 0 |

**Figure 4:** Wireshark Capture Interface Window

4. You'll see a list of the interfaces on your computer as well as a count of the packets that have been observed on that interface so far. Click on *Start* for the interface on which you want to begin packet capture (in the case, the Gigabit network Connection). Packet capture will now begin - Wireshark is now capturing all packets being sent/received from/by your computer!

5. Once you begin packet capture, a window similar to that shown in Figure 3 will appear. This window shows the packets being captured. By selecting *Capture* pulldown menu and selecting *Stop*, you can stop packet capture. But don't stop packet capture yet. Let's capture some interesting packets first. To do so, we'll need to generate some network traffic. Let's do so using a web browser, which will use the HTTP protocol that we will study in detail in class to download content from a website.

6. While Wireshark is running, enter the URL: http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html and have that page displayed in your browser. In order to display this page, your browser will contact the HTTP server at gaia.cs.umass.edu and exchange HTTP messages with the server in order to download this page, as discussed in section 2.2 of the text. The Ethernet frames containing these HTTP messages (as well as all other frames passing through your Ethernet adapter) will be captured by Wireshark.

7. After your browser has displayed the INTRO-wireshark-file1.html page (it is a simple one line of congratulations), stop Wireshark packet capture by selecting stop in the Wireshark capture window. The main Wireshark window should now look similar to Figure 3. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! The HTTP message exchanges with the gaia.cs.umass.edu web server should appear somewhere in the listing of packets captured. But there will be many other types of packets displayed as well (see, e.g., the many different protocol types shown in the *Protocol* column in Figure 3). Even though the only action you took was to download a web page, there were evidently many other protocols running on your computer that are unseen by the user. We'll learn much more about these protocols as we progress through the text! For now, you should just be aware that there is often much more going on than "meet's the eye"!

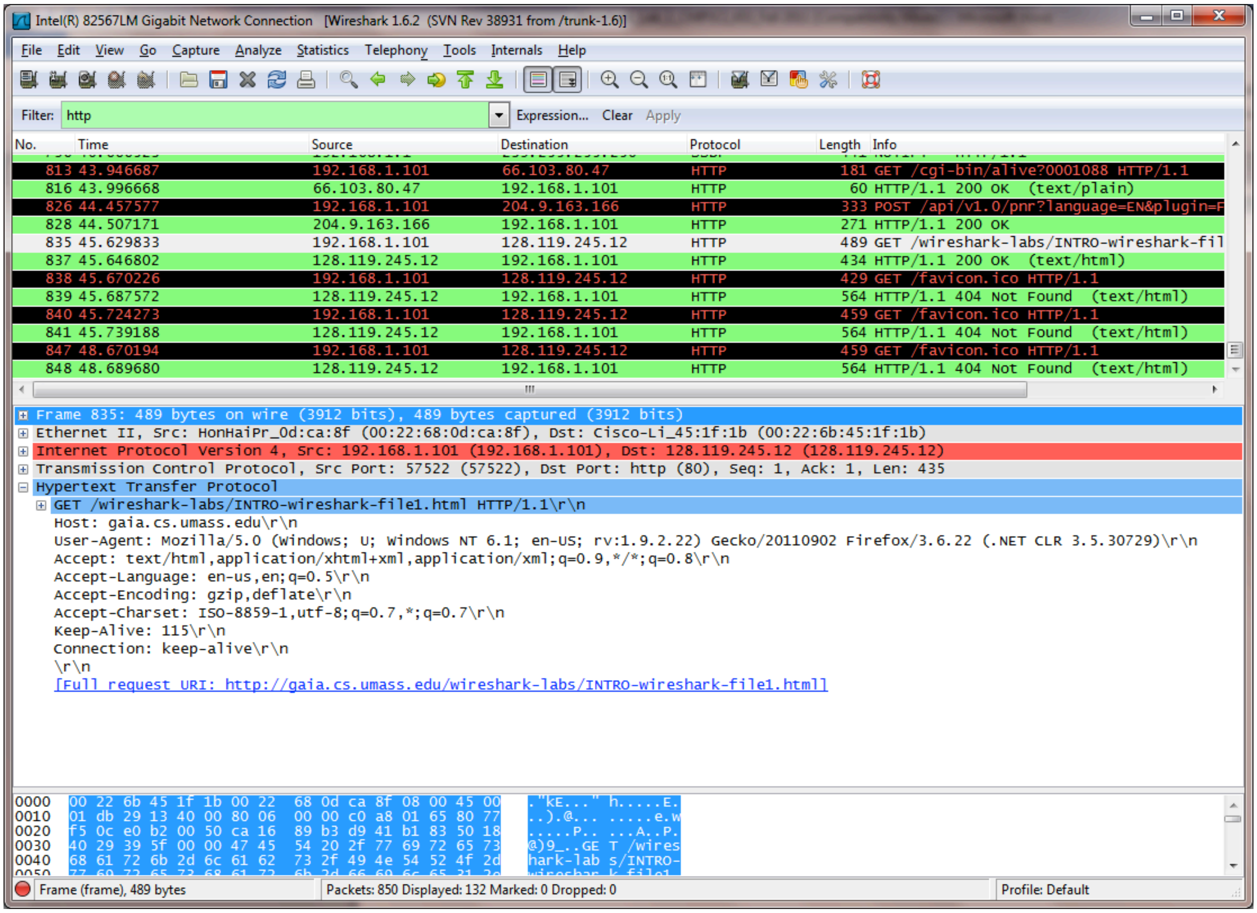
8. Type in “http” (without the quotes, and in lower case – all protocol names are in lower case in Wireshark) into the display filter specification window at the top of the main Wireshark window. Then select *Apply* (to the right of where you entered “http”). This will cause only HTTP message to be displayed in the packet-listing window.

9. Find the HTTP GET message that was sent from your computer to the gaia.cs.umass.edu HTTP server. (Look for an HTTP GET message in the “listing of captured packets” portion of the Wireshark window (see Figure 3) that shows “GET” followed by the gaia.cs.umass.edu URL that you entered. When you select the HTTP GET message, the Ethernet frame, IP datagram, TCP segment, and HTTP message header information will be displayed in the packet-header window[2]. By clicking on ‘+’ and ‘-‘ right-pointing and down-pointing arrowheads to the left side of the packet details window, *minimize* the amount of Frame, Ethernet, Internet Protocol, and Transmission Control Protocol information displayed. *Maximize* the amount information displayed about the HTTP protocol. Your Wireshark display should now look roughly as shown in Figure 5. (Note, in particular, the minimized amount of protocol information for all protocols except HTTP, and the maximized amount of protocol information for HTTP in the packet-header window).

10. Exit Wireshark

Congratulations! You’ve now completed the first lab.

[2] Recall that the HTTP GET message that is sent to the gaia.cs.umass.edu web server is contained within a TCP segment, which is contained (encapsulated) in an IP datagram, which is encapsulated in an Ethernet frame. If this process of encapsulation isn’t quite clear yet, review section 1.5 in the text



Intel(R) 82567LM Gigabit Network Connection [Wireshark 1.6.2 (SVN Rev 38931 from /trunk-1.6)]
File
Edit
View
Go
Capture
Analyze
Statistics
Telephony
Tools
Internals
Help
Expression...
Clear
Apply

Filter: http

```
No.  Time          Source           Destination      Protocol  Length Info
 813 43.946687     192.168.1.101    66.103.80.47     HTTP       181 GET /cgi-bin/alive?0001088 HTTP/1.1
 816 43.996668     66.103.80.47     192.168.1.101    HTTP        60 HTTP/1.1 200 OK  (text/plain)
 826 44.457577     192.168.1.101    204.9.163.166    HTTP       333 POST /api/v1.0/pnr?language=EN&plugin=F
 828 44.507171     204.9.163.166    192.168.1.101    HTTP       271 HTTP/1.1 200 OK
 835 45.629833     192.168.1.101    128.119.245.12   HTTP       489 GET /wireshark-labs/INTRO-wireshark-fil
 837 45.646802     128.119.245.12   192.168.1.101    HTTP       434 HTTP/1.1 200 OK  (text/html)
 838 45.670226     192.168.1.101    128.119.245.12   HTTP       429 GET /favicon.ico HTTP/1.1
 839 45.687572     128.119.245.12   192.168.1.101    HTTP       564 HTTP/1.1 404 Not Found  (text/html)
 840 45.724273     192.168.1.101    128.119.245.12   HTTP       459 GET /favicon.ico HTTP/1.1
 841 45.739188     128.119.245.12   192.168.1.101    HTTP       564 HTTP/1.1 404 Not Found  (text/html)
 847 48.670194     192.168.1.101    128.119.245.12   HTTP       459 GET /favicon.ico HTTP/1.1
 848 48.689680     128.119.245.12   192.168.1.101    HTTP       564 HTTP/1.1 404 Not Found  (text/html)
```

```
⊞ Frame 835: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits)
⊞ Ethernet II, Src: HonHaiPr_0d:ca:8f (00:22:68:0d:ca:8f), Dst: Cisco-Li_45:1f:1b (00:22:6b:45:1f:1b)
⊞ Internet Protocol Version 4, Src: 192.168.1.101 (192.168.1.101), Dst: 128.119.245.12 (128.119.245.12)
⊞ Transmission Control Protocol, Src Port: 57522 (57522), Dst Port: http (80), Seq: 1, Ack: 1, Len: 435
⊟ Hypertext Transfer Protocol
  ⊞ GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1\r\n
    Host: gaia.cs.umass.edu\r\n
    User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.22) Gecko/20110902 Firefox/3.6.22 (.NET CLR 3.5.30729)\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-us,en;q=0.5\r\n
    Accept-Encoding: gzip,deflate\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    Keep-Alive: 115\r\n
    Connection: keep-alive\r\n
    \r\n
    [Full request URI: http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html]
```

```
0000  00 22 6b 45 1f 1b 00 22  68 0d ca 8f 08 00 45 00   ."kE..." h.....E.
0010  01 db 29 13 40 00 80 06  00 00 c0 a8 01 65 80 77   ..).@... .....e.w
0020  f5 0c e0 b2 00 50 ca 16  89 b3 d9 41 b1 83 50 18   .....P.. ...A..P.
0030  40 29 39 5f 00 00 47 45  54 20 2f 77 69 72 65 73   @)9_..GE T /wires
0040  68 61 72 6b 2d 6c 61 62  73 2f 49 4e 54 52 4f 2d   hark-lab s/INTRO-
```

Frame (frame), 489 bytes | Packets: 850 Displayed: 132 Marked: 0 Dropped: 0 | Profile: Default

**Figure 5:** Wireshark window after step 9

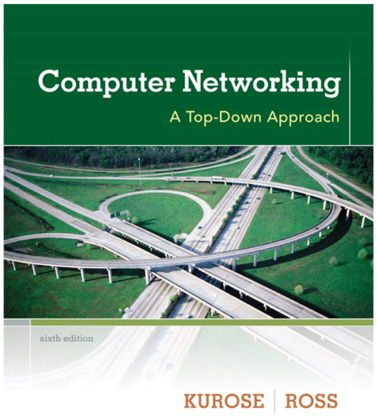
## What to hand in

The goal of this first lab was primarily to introduce you to Wireshark. The following questions will demonstrate that you've been able to get Wireshark up and running, and have explored some of its capabilities. Answer the following questions, based on your Wireshark experimentation:

1. List 3 different protocols that appear in the protocol column in the unfiltered packet-listing window in step 7 above.
2. How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received? (By default, the value of the Time column in the packet-listing window is the amount of time, in seconds, since Wireshark tracing began. To display the Time field in time-of-day format, select the Wireshark *View* pull down menu, then select Time *Display Format*, then select *Time-of-day*.)
3. What is the Internet address of the gaia.cs.umass.edu (also known as www-net.cs.umass.edu)? What is the Internet address of your computer?
4. Print the two HTTP messages (GET and OK) referred to in question 2 above. 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.

# Wireshark Lab: SSL v6.0

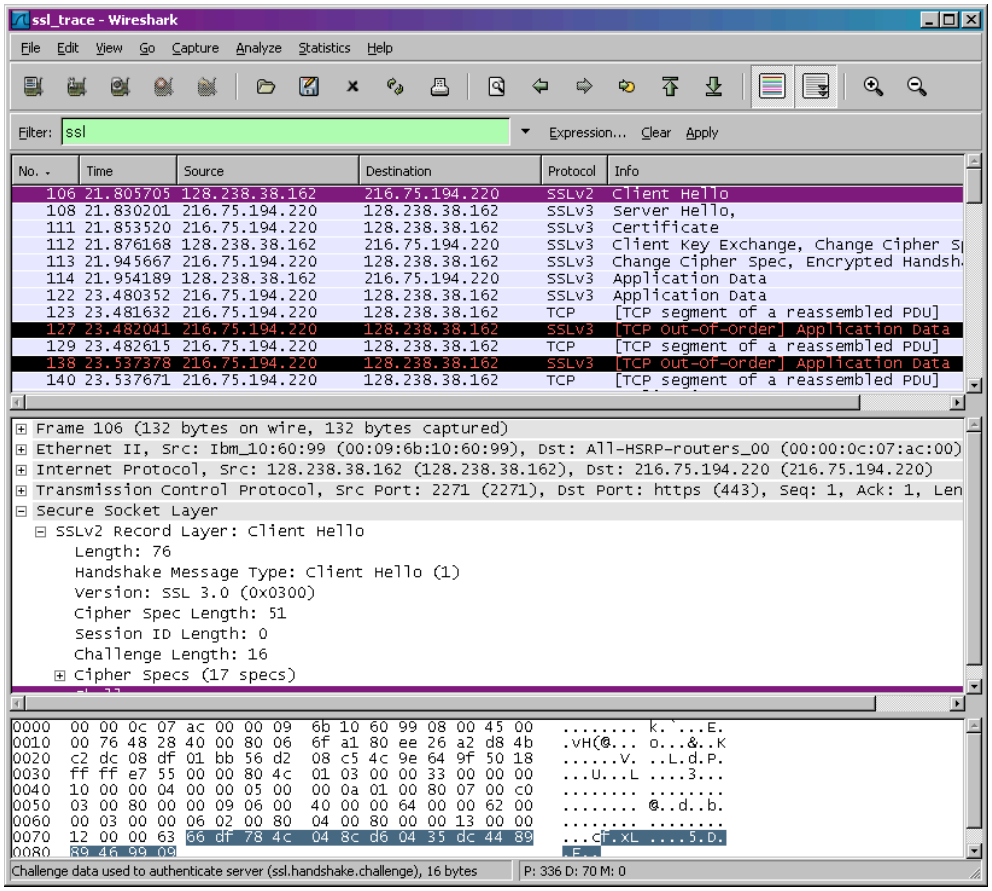
Supplement to *Computer Networking: A Top-Down Approach, 6th ed.,* J.F. Kurose and K.W. Ross



*"Tell me and I forget. Show me and I remember. Involve me and I understand."* Chinese proverb

In this lab, we'll investigate the Secure Sockets Layer (SSL) protocol, focusing on the SSL records sent over a TCP connection. We'll do so by analyzing a trace of the SSL records sent between your host and an e-commerce server. We'll investigate the various SSL record types as well as the fields in the SSL messages. You may want to review Section 8.6 in the text[1].

[1] References to figures and sections are for the 6th edition of our text, *Computer Networks, A Top-down Approach, 6th ed.,* J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2012.


ssl_trace - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: ssl
Expression... Clear Apply

```
No. -  Time       Source          Destination     Protocol Info
  106 21.805705 128.238.38.162  216.75.194.220  SSLv2    Client Hello
  108 21.830201 216.75.194.220  128.238.38.162  SSLv3    Server Hello,
  111 21.853520 216.75.194.220  128.238.38.162  SSLv3    Certificate
  112 21.876168 128.238.38.162  216.75.194.220  SSLv3    Client Key Exchange, Change Cipher S
  113 21.945667 216.75.194.220  128.238.38.162  SSLv3    Change Cipher Spec, Encrypted Handsh
  114 21.954189 128.238.38.162  216.75.194.220  SSLv3    Application Data
  122 23.480352 216.75.194.220  128.238.38.162  SSLv3    Application Data
  123 23.481632 216.75.194.220  128.238.38.162  TCP      [TCP segment of a reassembled PDU]
  127 23.482041 216.75.194.220  128.238.38.162  SSLv3    [TCP Out-Of-Order] Application Data
  129 23.482615 216.75.194.220  128.238.38.162  TCP      [TCP segment of a reassembled PDU]
  138 23.537378 216.75.194.220  128.238.38.162  SSLv3    [TCP Out-Of-Order] Application Data
  140 23.537671 216.75.194.220  128.238.38.162  TCP      [TCP segment of a reassembled PDU]
```

```
⊞ Frame 106 (132 bytes on wire, 132 bytes captured)
⊞ Ethernet II, Src: Ibm_10:60:99 (00:09:6b:10:60:99), Dst: All-HSRP-routers_00 (00:00:0c:07:ac:00)
⊞ Internet Protocol, Src: 128.238.38.162 (128.238.38.162), Dst: 216.75.194.220 (216.75.194.220)
⊞ Transmission Control Protocol, Src Port: 2271 (2271), Dst Port: https (443), Seq: 1, Ack: 1, Len
⊟ Secure Socket Layer
  ⊟ SSLv2 Record Layer: Client Hello
      Length: 76
      Handshake Message Type: Client Hello (1)
      Version: SSL 3.0 (0x0300)
      Cipher Spec Length: 51
      Session ID Length: 0
      Challenge Length: 16
    ⊞ Cipher Specs (17 specs)
```

```
0000   00 00 0c 07 ac 00 00 09  6b 10 60 99 08 00 45 00   ........ k.`...E.
0010   00 76 48 28 40 00 80 06  6f a1 80 ee 26 a2 d8 4b   .vH(@... o...&..K
0020   c2 dc 08 df 01 bb 56 d2  08 c5 4c 9e 64 9f 50 18   ......V. ..L.d.P.
0030   ff ff e7 55 00 00 80 4c  01 03 00 00 33 00 00 00   ...U...L ....3...
0040   10 00 00 04 00 00 05 00  00 0a 01 00 80 07 00 c0   ........ ........
0050   03 00 80 00 00 09 06 00  40 00 00 64 00 00 62 00   ........ @..d..b.
0060   00 03 00 00 06 02 00 80  04 00 80 00 00 13 00 00   ........ ........
0070   12 00 00 63 66 df 78 4c  04 8c d6 04 35 dc 44 89   ...cf.xL ....5.D.
0080   89 46 99 09                                        .F..
```

Challenge data used to authenticate server (ssl.handshake.challenge), 16 bytes | P: 336 D: 70 M: 0

## 1. Capturing packets in an SSL session

The first step is to capture the packets in an SSL session. To do this, you should go to your favorite e-commerce site and begin the process of purchasing an item (but terminating before making the actual purpose!). After capturing the packets with Wireshark, you should set the filter so that it displays only the Ethernet frames that contain SSL records sent from and received by your host. (An SSL record is the same thing as an SSL message.) You should obtain something like screenshot on the previous page.

If you have difficulty creating a trace, you should download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip and extract the *ssl-ethereal-trace-1* packet trace.

## 2. A look at the captured trace

Your Wireshark GUI should be displaying only the Ethernet frames that have SSL records. It is important to keep in mind that an Ethernet frame may contain one or more SSL records. (This is very different from HTTP, for which each frame contains either one complete HTTP message or a portion of a HTTP message.) Also, an SSL record may not

completely fit into an Ethernet frame, in which case multiple frames will be needed to carry the record.
Whenever possible, when answering a question below, you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. Annotate the printout[2] to explain your answer. To print a packet, use *File->Print*, choose *Selected packet only*, choose *Packet summary line,* and select the minimum amount of packet detail that you need to answer the question

1. For each of the first 8 Ethernet frames, specify the source of the frame (client or server), determine the number of SSL records that are included in the frame, and list the SSL record types that are included in the frame. Draw a timing diagram between client and server, with one arrow for each SSL record.
2. Each of the SSL records begins with the same three fields (with possibly different values). One of these fields is "content type" and has length of one byte. List all three fields and their lengths.

ClientHello Record:

3. Expand the ClientHello record. (If your trace contains multiple ClientHello records, expand the frame that contains the first one.) What is the value of the content type?
4. Does the ClientHello record contain a nonce (also known as a "challenge")? If so, what is the value of the challenge in hexadecimal notation?
5. Does the ClientHello record advertise the cyber suites it supports? If so, in the first listed suite, what are the public-key algorithm, the symmetric-key algorithm, and the hash algorithm?

ServerHello Record:

6. Locate the ServerHello SSL record. Does this record specify a chosen cipher suite? What are the algorithms in the chosen cipher suite?
7. Does this record include a nonce? If so, how long is it? What is the purpose of the client and server nonces in SSL?
8. Does this record include a session ID? What is the purpose of the session ID?
9. Does this record contain a certificate, or is the certificate included in a separate record. Does the certificate fit into a single Ethernet frame?

Client Key Exchange Record:

[2] What do we mean by "annotate"? If you hand in a paper copy, please highlight where in the printout you've found the answer and add some text (preferably with a colored pen) noting what you found in what you 've highlight. If you hand in an electronic copy, it would be great if you could also highlight and annotate.

10. Locate the client key exchange record. Does this record contain a pre-master secret? What is this secret used for? Is the secret encrypted? If so, how? How long is the encrypted secret?

Change Cipher Spec Record (sent by client) and Encrypted Handshake Record:

11. What is the purpose of the Change Cipher Spec record? How many bytes is the record in your trace?
12. In the encrypted handshake record, what is being encrypted? How?
13. Does the server also send a change cipher record and an encrypted handshake record to the client? How are those records different from those sent by the client?

Application Data

14. How is the application data being encrypted? Do the records containing application data include a MAC? Does Wireshark distinguish between the encrypted application data and the MAC?
15. Comment on and explain anything else that you found interesting in the trace.