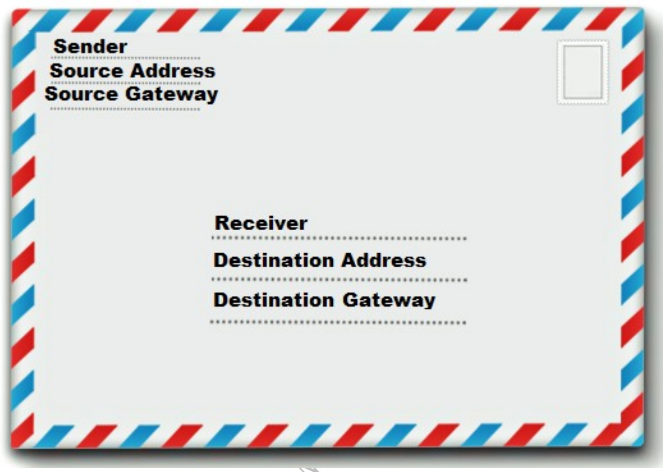
Lab 1. Wireshark

A **packet** is a quantum of information transmitted electronically on the wire, or through the air as a radio frequency transmission. Each packet has some data *within* it, and some data *about* it. The closest object in the everyday physical world, which is analogous to a packet, is a posted letter. A posted letter is a quantum of communication. Information *about* the communication is written on the outside of the envelope, while the content of the letter lies written on the paper *within* the envelope. In the world of computer networks, what is on the outside of the envelope is called the **header**; what is on the inside of the envelope is called the **payload**.



At a most basic level of understanding, each network **packet** has a sender or "source address" and a recipient or "destination address". The packet traverses a path as it is being delivered from the sender to its intended recipient. As it is moves along the route to its eventual destination, the intermediate devices that it visits follow a set of specific procedures to ensure that the transportation is efficient and successful. These specific procedures constitute a "network protocol". In the physical world the analogy of different protocols include the internal mechanisms that are at play in UPS, versus FedEx versus the U.S. postal system.

A **protocol** is a specification or "standard" which legislates the correct procedures and messages by which a set of devices coordinates to achieve a particular end goal. Some of the more common network protocols and their corresponding documentation are listed in the RFC's column of the chart below:

Protocol	Name	RFC
Internet Protocol	IP	http://www.rfc-editor.org/rfc/rfc791.txt
Internet Control Message Protocol	ICMP	http://www.rfc-editor.org/rfc/rfc792.txt
Transmission Control Protocol	TCP	http://www.rfc-editor.org/rfc/rfc793.txt
User Datagram Protocol	UDP	http://www.rfc-editor.org/rfc/rfc768.txt
File Transfer Protocol	FTP	http://www.rfc-editor.org/rfc/rfc765.txt
Simple Mail Transfer Protocol	SMTP	http://www.rfc-editor.org/rfc/rfc788.txt
HyperText Transfer Protocol	HTTP	http://www.rfc-editor.org/rfc/rfc2068.txt
Domain Name Service	DNS	http://www.rfc-editor.org/rfc/rfc1035.txt

Each of the above protocols has a different purpose, and prescribed behaviors by which that purpose is achieved; this is spelled out in each of the RFC documents. Once you understand the purpose, rules, and messages that the protocol uses, as you examine network traffic using **packet capture** software, you can reason about the context and causes of the observed communication sequences. From these, you can draw inferences about what must be occurring at the level of actual human users and their software.

In this lab, a software utility called **Wireshark** will be used to capture and observe packets. Note that although we use the jargon term "capture", what we are doing does not restrict packets in any way. The term "capturing" is intended in the sense of "photographing", and not in the sense of "putting in a cage".

The context of this exercise will be to capture the network traffic that occurs when you engage in the common act of visiting a website. By examining the digital network artifacts of this commonplace action, you will begin to realize how almost every action within the digital ecosystem leaves behind observable traces, and (in later labs), how you can be best positioned to examine these traces in order to and draw valuable inferences about what is going on in your network at any point in time.

Goals. By the end of this lab, you should have learned:

1. How to capture network traffic using Wireshark, a packet-capture utility.
2. An overview of some common network protocols that enter into the narrative whenever you browse to a website.
3. A basic understanding of how HTTP security operates.

Setup. To carry out this lab, make sure you have installed the following software on your machine:

1. WinPcap on Windows[1] or libpcap on Linux.
2. Wireshark[2] on Windows or Linux.
3. traceroute on Linux.

Competency 1. Knowing what happens when you go to a website

What happens when you go to cybersec21.com in a web browser? From your perspective at this point, perhaps you think "I type in cybersec21.com into your web browser, press enter, and see the web page". Let's slow it down and look at all the steps in between:

a. You type in cybersec21.com into your web browser and press enter.

1 http://www.mirrorservice.org/sites/ftp.wiretapped.net/pub/security/packet-capture/winpcap/install/default.htm

2 http://www.wireshark.org/download.html

b. Your computer talks to the DNS server to translate the name "cybersec21.com" to the IP address of the web server of Cybersec21. The DNS server replies with the IP address of Cybersec21's web server.
c. Your computer talks to the Cybersec21 web server using TCP/IP to set up a connection.
d. Your computer sends an HTTP GET message across the TCP/IP connection. A response comes back from the web server, carrying an HTTP status code, as well as Cybersec21's main webpage.
e. Your computer closes the TCP/IP connection to Cybersec21's web server.
f. Your web browser software renders the HTML content of the web page probably formatting it using the CSS and usually modifying it by executing Javascript allowing you to see the web page.

In the steps (a) to (f), different protocols come into play. In step (b), your computer was talking to a DNS server according to the DNS procedures and messages specified in RFC 1035. It was following the DNS protocol to achieve the objective of getting the IP address of cybersec21.com.

In steps (c) and (e), it was talking to a machine at Cybersec21, using the TCP/IP procedures and messages specified in RFC 793 and RFC 791. It was following TCP/IP protocols to achieve the objective of setting up a reliable communication channel with cybersec21.com. In step (d) it was talking to a web server at Cybersec21, using the HTTP procedures and messages specified in RFC 2068. It was talking to cybersec21.com over the communication channel it established in step (c), following HTTP protocols to achieve the objective of getting a specific web page. In step (e) your machine was following TCP/IP protocols to take down up a reliable communication channel with cybersec21.com.

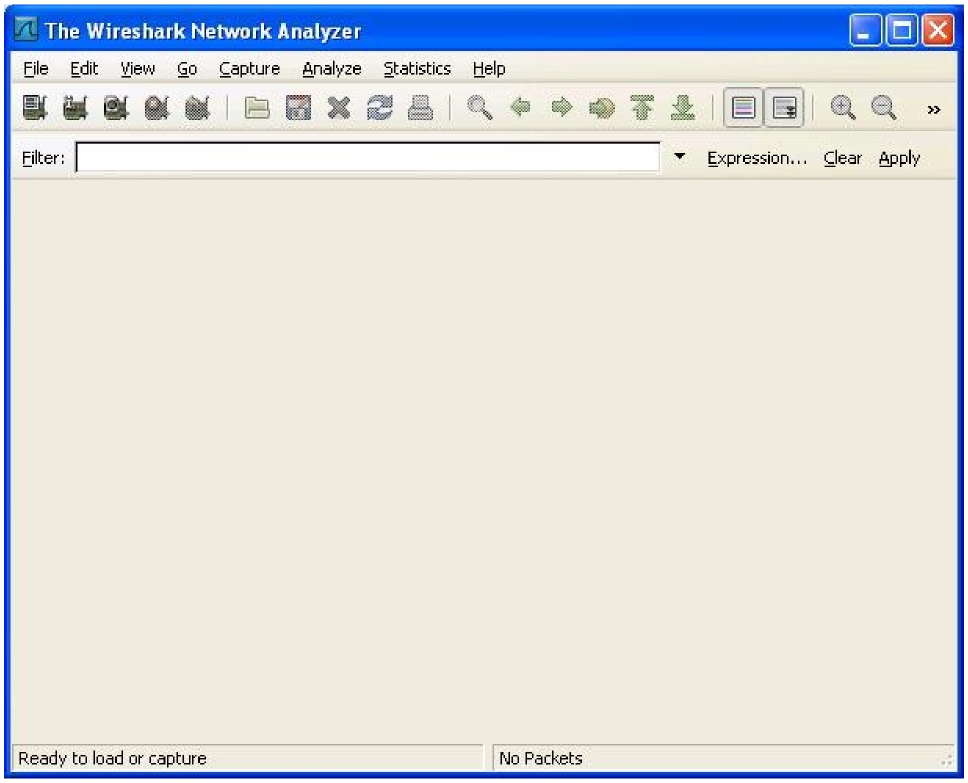
Each of these protocols is quite complex, and each RFC document can be hundreds of pages long. The reason is that protocol specifications have to be absolutely specify every possible situation. For a protocol to be effective there can be no ambiguity, and these details are what make the RFCs so voluminous. The detail is necessary so that people who are building new machines (hardware and software) need to be able to have them coordinate reliably within the existing ecosystem of machines, so that the computers can coordinate with each other to collectively achieve desired outcomes. The good news is that although the RFCs are voluminous and the protocols described in them are very complex, we only need to understand them well enough to be able to make sense of the messages we see going back and forth. We just want to be able to deduce what the machines (and the humans who are using them) are doing, based on the messages we see going back and forth. Because we are not in the business of building new machines, we don't need to understand every little detail described in the RFCs.

Let's test the above description of steps (a) to (f) by actually capturing the packets that travel on the network when we visit the Cybersec21 website.

Competency 2. Doing a simple packet capture

At this point, all of your software should be already installed as described in the **Setup** section of this Lab. If you have any browser open at this point you should ensure that you clear its cache and then fully quit it. Now re-launch your browser program.

Launch Wireshark from the command line, or by double-clicking on its icon. You should see a window that looks similar to the below:

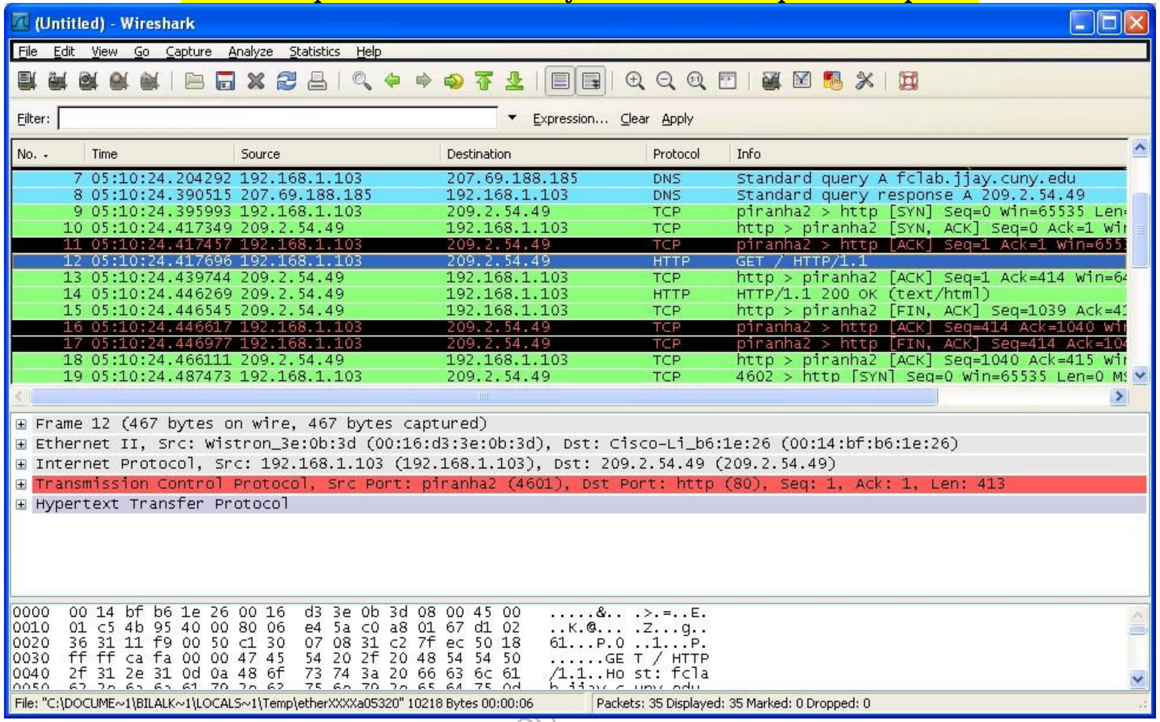

The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
Ready to load or capture
No Packets

B) A computer can have multiple network interfaces, some wired and some wireless. Wireshark needs to be told which of these it is to capture packets from. It is recommended that you machine is using Ethernet and you select your wired network connection. This is due to the prohibition of running packet captures on a wireless network under U.S. law. To specify which interface to capture packets from, select *Capture* > *Options* from the dropdown menu and then select the interface from which you wish to capture traffic. Before you can begin capturing traffic on this interface you need to ensure that your operating system will allow Wireshark to open ports on the interface you've selected.

C) Once you've specified an interface and told your OS to permit Wireshark to open ports, you can begin capturing traffic by selecting *Capture* > *Start*. This tells Wireshark to begin capturing network traffic.

D) Now go back to your web browser and surf to *http://cybersec21.com*. Once the webpage has appeared, return to Wireshark and select *Capture* > *Stop* from the drop down menu. At this point, in Wireshark, you should see something similar to what is depicted on the following page.

TODO: Replace with actual cybersec21.com packet capture


(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. -  Time             Source          Destination     Protocol Info
     7 05:10:24.204292 192.168.1.103   207.69.188.185  DNS      Standard query A fclab.jjay.cuny.edu
     8 05:10:24.390515 207.69.188.185  192.168.1.103   DNS      Standard query response A 209.2.54.49
     9 05:10:24.395993 192.168.1.103   209.2.54.49     TCP      piranha2 > http [SYN] Seq=0 Win=65535 Len=
    10 05:10:24.417349 209.2.54.49     192.168.1.103   TCP      http > piranha2 [SYN, ACK] Seq=0 Ack=1 Win
    11 05:10:24.417457 192.168.1.103   209.2.54.49     TCP      piranha2 > http [ACK] Seq=1 Ack=1 Win=655
    12 05:10:24.417696 192.168.1.103   209.2.54.49     HTTP     GET / HTTP/1.1
    13 05:10:24.439744 209.2.54.49     192.168.1.103   TCP      http > piranha2 [ACK] Seq=1 Ack=414 Win=64
    14 05:10:24.446269 209.2.54.49     192.168.1.103   HTTP     HTTP/1.1 200 OK (text/html)
    15 05:10:24.446545 209.2.54.49     192.168.1.103   TCP      http > piranha2 [FIN, ACK] Seq=1039 Ack=41
    16 05:10:24.446617 192.168.1.103   209.2.54.49     TCP      piranha2 > http [ACK] Seq=414 Ack=1040 Win
    17 05:10:24.446977 192.168.1.103   209.2.54.49     TCP      piranha2 > http [FIN, ACK] Seq=414 Ack=104
    18 05:10:24.466111 209.2.54.49     192.168.1.103   TCP      http > piranha2 [ACK] Seq=1040 Ack=415 Win
    19 05:10:24.487473 192.168.1.103   209.2.54.49     TCP      4602 > http [SYN] Seq=0 Win=65535 Len=0 MS
```

```
⊞ Frame 12 (467 bytes on wire, 467 bytes captured)
⊞ Ethernet II, Src: Wistron_3e:0b:3d (00:16:d3:3e:0b:3d), Dst: Cisco-Li_b6:1e:26 (00:14:bf:b6:1e:26)
⊞ Internet Protocol, Src: 192.168.1.103 (192.168.1.103), Dst: 209.2.54.49 (209.2.54.49)
⊞ Transmission Control Protocol, Src Port: piranha2 (4601), Dst Port: http (80), Seq: 1, Ack: 1, Len: 413
⊞ Hypertext Transfer Protocol
```

```
0000  00 14 bf b6 1e 26 00 16  d3 3e 0b 3d 08 00 45 00   .....&.. .>.=..E.
0010  01 c5 4b 95 40 00 80 06  e4 5a c0 a8 01 67 d1 02   ..K.@... .Z...g..
0020  36 31 11 f9 00 50 c1 30  07 08 31 c2 7f ec 50 18   61...P.0 ..1...P.
0030  ff ff ca fa 00 00 47 45  54 20 2f 20 48 54 54 50   ......GE T / HTTP
0040  2f 31 2e 31 0d 0a 48 6f  73 74 3a 20 66 63 6c 61   /1.1..Ho st: fcla
```

```
File: "C:\DOCUME~1\BILALK~1\LOCALS~1\Temp\etherXXXXa05320" 10218 Bytes 00:00:06    Packets: 35 Displayed: 35 Marked: 0 Dropped: 0
```

In Wireshark's window you see four distinct window panels: (i) A display packet filter specification, (ii) A packet list, (iii) A packet detail (which is hierarchically organized, based on nested packet structure), and (iv) A packet hex dump.

The first panel, *(i) display packet filter specification,* allows you to filter the packets that have just been captured. Thus if you wanted to only see the HTTP traffic then you would type in HTTP into the filter and apply it. This would result in the second panel, *(ii) packet list*, to display only those HTTP packets. If there is no filter then it will show all the packets that have been collected during the capture session. The third panel, *(iii) packet detail*, allows you to drill down and explore that particular packet and all of the information in that packet, potentially including its contents. Whilst the packet list provides a good overview of the packet the packet detail window shows all the information about that packet. Finally we have (iv) *the packet hex dump*, which provides a raw listing of the bits that comprise the packet (as hexadecimal numbers and ASCII characters).

Often, when looking at network traffic, you are only interested in a small subset of packets that have been captured. In order to print only these packets of interest you would select the packets, right click and then "mark" them. Once you have done this you can go to File > Print > Marked packets only > Print details as displayed.

Q1. What protocols are active when you browse cybersec21.com?

You can see from the capture that a very large number of packets are involved when you do something as simple as visit the Cybersec21 website. Within Wireshark's display, packets are numbered sequentially in the order that they were captured, and the number column shows you the index of the packet. To find out what protocol the packet relates to, you would look under the protocol column. From the example in the capture you just did, you can check and see that that all the protocols we claimed would be involved (DNS, TCP/IP and HTTP) were in fact caught sending messages back and forth in the time immediately following our visit to the Cybersec21 website.

Q2. How long did cybersec21.com take to respond?

The "timestamp" column indicates the exact time at which the packet was observed. For example, it is possible to find out how long it took for the website that you just visited to reply. To do this you should locate both the HTTP GET message and the HTTP OK message. Once you have this, look at the timestamps of both and subtract one from the other. To get the total time elapsed between your machine sending the GET and your receipt of the returned status message.

Q3. What is your IP address, and what is cybersec21.com's IP address?

The "source" and "destination" columns show you the IP address of the machine that the packet claims to have been sent by, and the address to which it was destined. For packets leaving your machine, the source IP address is your machine whilst the destination IP address is the IP of whatever service you were communicating with (the DNS server, Cybersec21's web server, etc.) For packets sent to your machine, the destination IP address is your IP address, while the source IP address is the IP of whatever service you were communicating with (the DNS server, Cybersec21's web server, etc.)

Competency 3. Understanding HTTP

A) We will now examine just the packets that were transferred as part of the HyperText Transfer Protocol (HTTP)[3] when you went to cybersec21.com. To make this easy, use the filter option in Wireshark's topmost panel to show only those packets which have something to do with the HTTP protocol.

B) The HTTP protocol operates in a sequence of HTTP **requests** and **responses**[4]. An HTTP server listening at that address on that port receives the client's HTTP request message. Upon receiving the request, the server sends back a status code and a Response. The body of this response message is typically the requested resource (although an error message may be returned if the requested resource is unavailable or is unchanged since the last time the client requested it). Each HTTP packet, whether it is a request or response, has its own header and payload (recall, in our analogy of an envelope, the header is the writing on the outside of the envelope, while the payload is the data inside the envelope).

Q4. What version of HTTP is cybersec21.com's web server using?

It is important to be able to distinguish what version of a protocol your machine is running (in this case we are looking closely at HTTP). Older versions generally have more documented vulnerablities. This will depend on the browser and the operating system you are running on your own machine. In addition, it is also useful to determine what version of HTTP the server that you communicating with is running. To do this look at the packet list window, click on the HTTP GET packet and look under the info column. At the end of that message you should see HTTP/ followed by a number. The number indicates the version that is being used by your browser.

To find out the **protocol version** of HTTP being used by Cybersec21's web server you merely follow the same process with respect to the HTTP OK packet, locating the HTTP/ number. This is an example of information in the header of an HTTP request packet. Version numbers of protocols are often negotiated early on in the packet exchanges between the two machines. As an analyst it is useful to keep track of version numbers of protocols, since outdated protocols can indicate that one or both machines are running old versions of software that might contain vulnerabilities.

You may also want to find out other **features** of the browser and server. For example, you might want to know what languages this browser can handle. This is another example of information in the header of an HTTP request packet. To find

[3] 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.

[4] HTTP requests and responses travel "over" TCP/IP, so the client initiates a request by establishing a TCP connection to the IP address of the web server at a well-known port (typically port 80, sometime port 8080). The rationale and process for doing this is covered in Competency 4.

out this information, you need to locate a source packet that has come from your machine and in the packet detail window select Hypertext Transfer Protocol and then locate the Accept-Language line.

More generally, a **software version** is the number associated with the current code revision of a piece of software. As the underlining code requires new features (including patches to fix bugs and prevent misuse of the software), the vendor releases new software versions, which are assigned a higher version number. Within Wireshark, packet formats often implicitly reveal the version numbers of the software that is generating them, including the versions of operating system, web servers, and/or web browsers. When you are able to identify the version number of a piece of software you can determine whether that version is compatible with other software that may be operating on the same network, as well as whether it has any known security vulnerabilities which can be exploited by attackers. An attacker or investigator can thus learn a lot about the ecosystem from ancillary features included inside packet traffic.

Q5. What URL did the user visit at cybersec21.com's website?

An **HTTP** request packet specifies the file that the browser wishes to retrieve (relative to the root of the webserver's root directory). This is an example of information in the payload of an HTTP request packet.

When a client sends requests to a server via HTTP, the server's response is often a specific **HTTP status code**. An HTTP status code is a three-digit number representing a specific response/action or designation from a device, as specified by **IANA** (Internet Assigned Numbers Authority) and reviewed and maintained by IETF (Internet Engineering Task Force), as part of the HTTP. These codes and their interpretations are listed in this chart below:

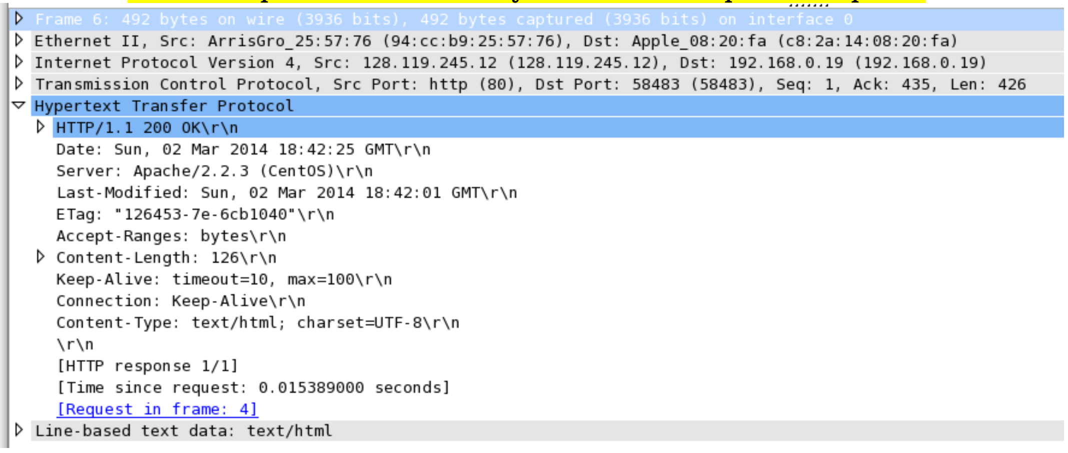
Status Code	Response Message	Interpretation
200	OK	Request accepted
301	Moved Permanently	Webpage moved/redirected
400	Bad Request	Error in data request
401	Unauthorized	Authentication needed/failed
403	Forbidden	Rights/Authentication needed
404	Page Not Found	Webpage not present
500	Internal Server Error	Problem processing request

You can review the HTTP status codes and their corresponding RFCs online[5].

5 http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml

You know that your browser's request was successful if the server returns a Status Code: 200.

TODO: Replace with actual cybersec21.com packet capture

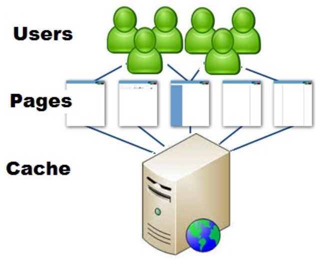


```
▷ Frame 6: 492 bytes on wire (3936 bits), 492 bytes captured (3936 bits) on interface 0
▷ Ethernet II, Src: ArrisGro_25:57:76 (94:cc:b9:25:57:76), Dst: Apple_08:20:fa (c8:2a:14:08:20:fa)
▷ Internet Protocol Version 4, Src: 128.119.245.12 (128.119.245.12), Dst: 192.168.0.19 (192.168.0.19)
▷ Transmission Control Protocol, Src Port: http (80), Dst Port: 58483 (58483), Seq: 1, Ack: 435, Len: 426
▽ Hypertext Transfer Protocol
  ▷ HTTP/1.1 200 OK\r\n
    Date: Sun, 02 Mar 2014 18:42:25 GMT\r\n
    Server: Apache/2.2.3 (CentOS)\r\n
    Last-Modified: Sun, 02 Mar 2014 18:42:01 GMT\r\n
    ETag: "126453-7e-6cb1040"\r\n
    Accept-Ranges: bytes\r\n
  ▷ Content-Length: 126\r\n
    Keep-Alive: timeout=10, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.015389000 seconds]
    [Request in frame: 4]
▷ Line-based text data: text/html
```

Q6. What happens if you visit a website a second time?

With respect to networks and computer transmissions, a “cache” is a saved copy of some data. In the case of HTTP, the cache consists of the website's HTML code and images. By maintaining a cache, the client can avoid re-requesting the same data over and over again; the requester can be given the copy of the data from the cache, avoiding the latency involved in round trip traffic in the wide area Internet. Because the cache is maintained close to the requester, the requester receives the requested resource much faster than if the request was transmitted to the remote server and processed there. Caching technology reduces bandwidth, server load and network congestion.

A **browser cache** refers to copies of website data that is retained by your Web Browser (typically on your local hard drive), in order to serve you requests faster for websites when the user requests them. While browser caching in information technology was designed to improve efficiently, it also provides a history of data and a point of data collection as a web browser collects and caches the web sites a user visits. Organizations may also implement a web caches or “proxies”, across a set of user machines, and thereby minimize the traffic bottlenecks[6]. In this event, files cached by user A may be used to service the requests of other users later.



6Duane Wessels, Web Caching (O'Reilly and Associates, 2001). ISBN 1-56592-536-X

How does a request sent by a web browsing client to a web server result in the use of **browser-cached document**? Informally, when a client requests a commonly requested website, the web server quickly replies and indicate that the website has not changed since the last time it was retrieved. This tells the browser that it can go ahead and use its **cached** copy of the web page. The client sees the page faster, and the server can quickly move on to process other requests; both the client and the server end up benefiting from the cache. Caching is important to understand because within the HTTP response packets, you will sometimes notice that the packet **body** does *not* contain any webpage data from the web server.

Not surprisingly, timestamps play a vital role in determining whether the browser needs to be given the "new version" of a page or told to use the existing page stored locally in its cache. To see this decision-making process in action, find the first HTTP GET packet and look at the details of the packet. You should not see an "IF-MODIFIED-SINCE" line, however if you look at the second HTTP GET packet request, this relates to the second time you visited that same webpage, you will notice this now appears in the packet detail. The second GET request is known as a **conditional get method**[7] which occurs because the browser sees you are requesting a resource for which there is an entry in the cache. The additional field in the conditional get is asking the server to check if the data being requested has changed since the browser last retrieved it.

In summary, whenever you see the conditional statement from the browser to the server, you know the browser has a cache copy. To determine whether the server fulfilled the request by returning content or not, inspect the server's response to the HTTP GET message. Browse the packet detail and look for the line "Content-Length". If the number is greater than zero then you know that the server returned content to the browser. If however you see the following line in the server's response "HTTP/1.1 304 Not Modified" then it did not return the contents of the file because the cached file had not been modified. The purpose of such caching is to speed up web document retrieval by allowing the client browser to safely use its cached copy whenever it is known to be identical to the copy at the server.

Q7. What are the security implications of having a web cache?

A browser cache is a copy of the web documents that were retrieved by a user. As such, it is open to being inspected and analyzed by both investigators and attackers. Organizational web caches (such as filters and proxies) offer a similar repository of data for analysis. This will be the subject of Lab 7 on **Squid**.

7 http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

Competency 4. Understanding TCP

It is often said that the HTTP protocol operates “over” the TCP/IP protocol. The word “over” here indicates that HTTP sends its messages back and forth by chopping them up and putting them inside TCP/IP packets. When these TCP/IP packets arrive at the destination, their contents are extracted and the original HTTP message is reassembled and then given to the right program at the destination for processing.

In our analogy of the envelope, the story would be something like this: Every HTTP packet (e.g. a GET request) has both a header and a body; think of this like posted letter. The letter inside (body) specifies what resource you want to request, and the writing on the envelope (header) specifies what languages you will accept. We take this HTTP packet and tear it up into small pieces. Then we put each of the pieces as a payload of inside a dedicated TCP/IP envelope. Then we post these TCP/IP “letters” (each containing a small piece of the HTTP letter) one after another to the server. The server gets all the TCP/IP letters, in order, and opens them. The TCP/IP letters manage to find the server because of the magic of the IP protocol; the TCP/IP letters arrive in order because of the magic of the TCP protocol. The operating system at the server now opens all these TCP/IP envelopes and extracts the payloads therein. These payloads are concatenated together to produce the original HTTP “letter”. The payload of this HTTP request packet is a GET request; this is handed to the software at the web server. Analogous fragmentation and re-assembly occurs in the reverse direction: When the server replies to the browser, the web page being returned is placed in an HTTP response packet. But this HTTP response packet will need to be chopped up and carried inside a sequence of TCP/IP packets back to the client. Upon receipt, the payloads of the TCP/IP packets are concatenated to reconstitute the HTTP response packet. The payload of this HTTP response packet is the web page data; this is handed to the client web browser software for processing.

What is so special about the TCP/IP protocol?

1) The TCP protocol provides a way to reliably deliver a sequence of packets, in-order, between a program on one machine and a program on another machine.
2) The IP protocol provides a way to attempt to send a single packet from one machine to another machine.

Not surprisingly, the TCP protocol is implemented “over” the IP protocol, which is why it is always written TCP/IP.

While HTTP packets are often called **messages**, TCP packets are called **segments**, and IP packets are called **datagrams**: this is largely a cultural issue.

In order to send messages “over” TCP/IP, one cannot just create TCP/IP packets and send them; one has to first establish a **bidirectional connection** between the two endpoints. The reason both sides have to agree to do this in advance, is that the TCP protocol needs to maintain a lot of state information in order to ensure that data is transmitted reliably and in order between the two endpoints. Establishing a connection allows both endpoints to allocate resources and be prepared to maintain that state information. We are familiar with this kind of requirement from our use of the telephone. If we want to talk to someone we cannot simply pick up the phone and start talking to him or her. We have to declare who we want to talk to (by dialing their telephone "address") and then wait for the phone company to tell us that the connection had been established. The establishment of the telephone connection is necessary because resources must be allocated at the phone company in order for the voice traffic to be handled appropriately.

In establishing a bidirectional TCP connection, there is an asymmetry between the two endpoints. One side has to initiate the establishment connection, and the other has to accept. This is quite similar to the way that a phone connection cannot take place unless one of the two parties takes the initiative to call, and the other accepts the call. In both phone connections and TCP/IP connections, there is a calling party and there is a called party.

Establishing a bidirectional TCP connection requires a three-way handshake between the endpoints: First the calling party sends a TCP/IP packet of type SYN. Then the called party sends a TCP/IP packet of type SYNACK. Finally, the calling party sends a TCP/IP packet of type ACK. In the metaphor of the telephone: when the calling party finishes dialing the number, the SYN packet leaves. When the SYN packet arrives at the called party, the phone starts ringing. If the called party picks up the phone, the SYNACK gets sent. When the SYNACK arrives at the calling party, they know that the connection is established and they send an ACK packet. When the ACK arrives at the called party, they too know that the connection is established.

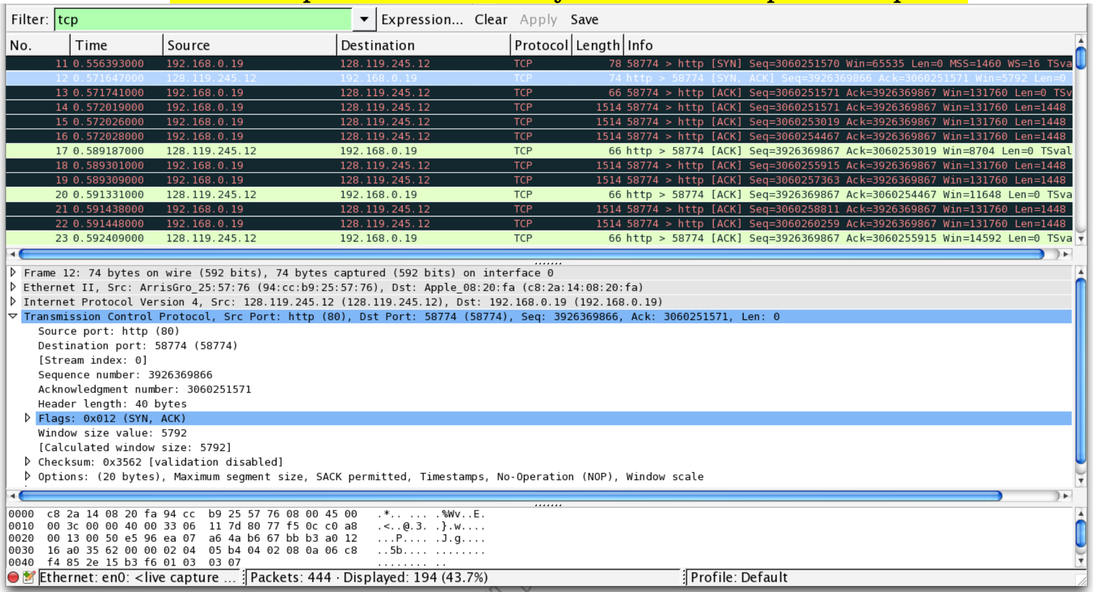
When two programs are done using a TCP connection, one of them must formally hang up on the other. This requires a two-way handshake between the endpoints: First one party sends a TCP/IP packet of type FIN. Then the other party sends a TCP/IP packet of type FINACK.

Q8. How does HTTP travel "over" TCP/IP?

Let's start by taking a high level view of the packet capture. First, filter the packets displayed in the Wireshark window by entering "tcp" into the display filter specification window towards the top of the Wireshark window. What you should see is series of TCP and HTTP messages between your computer and cybersec21.com. You should see the initial three-way handshake containing a SYN, SYNACK and ACK messages. You should then see an HTTP POST message and a series of "HTTP Continuation" messages being sent from your computer to cybersec21.com. In actuality, there is no such thing as an HTTP Continuation message; this is just Wireshark's way of indicating that there are multiple TCP segments being used to carry a single long HTTP message. You should also see TCP ACK segments being returned from cybersec21.com to your computer.

Since this part of the lab is about TCP rather than HTTP, let's change Wireshark's "listing of captured packets" window so that it shows information about the TCP packets containing the HTTP packets, rather than showing information about the HTTP packets themselves. To have Wireshark do this, select *Analyze > Enabled Protocols.* Then uncheck the HTTP box and select *OK*. Also, since we would like to see TCP's sequence numbers (and not the "relative sequence numbers" that Wireshark may display). To do this, go to Edit > Preferences > Protocols > TCP and uncheck "relative sequence numbers".

TODO: Replace with actual cybersec21.com packet capture



```
No.   Time         Source           Destination      Protocol Length Info
   11 0.556393000  192.168.0.19     128.119.245.12   TCP        78 58774 > http [SYN] Seq=3060251570 Win=65535 Len=0 MSS=1460 WS=16 TSva
   12 0.571647000  128.119.245.12   192.168.0.19     TCP        74 http > 58774 [SYN, ACK] Seq=3926369866 Ack=3060251571 Win=5792 Len=0
   13 0.571741000  192.168.0.19     128.119.245.12   TCP        66 58774 > http [ACK] Seq=3060251571 Ack=3926369867 Win=131760 Len=0 TSv
   14 0.572019000  192.168.0.19     128.119.245.12   TCP      1514 58774 > http [ACK] Seq=3060251571 Ack=3926369867 Win=131760 Len=1448
   15 0.572026000  192.168.0.19     128.119.245.12   TCP      1514 58774 > http [ACK] Seq=3060253019 Ack=3926369867 Win=131760 Len=1448
   16 0.572028000  192.168.0.19     128.119.245.12   TCP      1514 58774 > http [ACK] Seq=3060254467 Ack=3926369867 Win=131760 Len=1448
   17 0.589187000  128.119.245.12   192.168.0.19     TCP        66 http > 58774 [ACK] Seq=3926369867 Ack=3060253019 Win=8704 Len=0 TSval
   18 0.589301000  192.168.0.19     128.119.245.12   TCP      1514 58774 > http [ACK] Seq=3060255915 Ack=3926369867 Win=131760 Len=1448
   19 0.589309000  192.168.0.19     128.119.245.12   TCP      1514 58774 > http [ACK] Seq=3060257363 Ack=3926369867 Win=131760 Len=1448
   20 0.591331000  128.119.245.12   192.168.0.19     TCP        66 http > 58774 [ACK] Seq=3926369867 Ack=3060254467 Win=11648 Len=0 TSva
   21 0.591438000  192.168.0.19     128.119.245.12   TCP      1514 58774 > http [ACK] Seq=3060258811 Ack=3926369867 Win=131760 Len=1448
   22 0.591448000  192.168.0.19     128.119.245.12   TCP      1514 58774 > http [ACK] Seq=3060260259 Ack=3926369867 Win=131760 Len=1448
   23 0.592409000  128.119.245.12   192.168.0.19     TCP        66 http > 58774 [ACK] Seq=3926369867 Ack=3060255915 Win=14592 Len=0 TSva
```

```
▷ Frame 12: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▷ Ethernet II, Src: ArrisGro_25:57:76 (94:cc:b9:25:57:76), Dst: Apple_08:20:fa (c8:2a:14:08:20:fa)
▷ Internet Protocol Version 4, Src: 128.119.245.12 (128.119.245.12), Dst: 192.168.0.19 (192.168.0.19)
▽ Transmission Control Protocol, Src Port: http (80), Dst Port: 58774 (58774), Seq: 3926369866, Ack: 3060251571, Len: 0
    Source port: http (80)
    Destination port: 58774 (58774)
    [Stream index: 0]
    Sequence number: 3926369866
    Acknowledgment number: 3060251571
    Header length: 40 bytes
  ▷ Flags: 0x012 (SYN, ACK)
    Window size value: 5792
    [Calculated window size: 5792]
  ▷ Checksum: 0x3562 [validation disabled]
  ▷ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
```

```
0000  c8 2a 14 08 20 fa 94 cc  b9 25 57 76 08 00 45 00   .*.. ... .%Wv..E.
0010  00 3c 00 00 40 00 33 06  11 7d 80 77 f5 0c c0 a8   .<..@.3. .}.w....
0020  00 13 00 50 e5 96 ea 07  a6 4a b6 67 bb b3 a0 12   ...P.... .J.g....
0030  16 a0 35 62 00 00 02 04  05 b4 04 02 08 0a 06 c8   ..5b.... ........
0040  f4 85 2e 15 b3 f6 01 03  03 07                     ........ ..
```

```
Ethernet: en0: <live capture ...   Packets: 444 · Displayed: 194 (43.7%)   Profile: Default
```

Q9. How do programs identify themselves to each other across the network?

It is possible to have two services (e.g. web servers A and B or a database server D) running on the same one physical machine, with each of the two services expecting to have clients connect to them over the network. How is this achieved? A machine has just one IP address for each network interface, so how would a remote client C indicate which of the two services they wanted a connection to? The answer to this lies in the notion of a TCP **port**. A TCP port can be thought of as a 4 byte (low order) extension of the IP address. If services A and B were both running on the same machine that has an IP address X, the A would ask the OS to "listen" for connections on a port, say 80, while B would ask the OS to "listen" on a different port, say 8080. Connection requests from client C would have to specify not only the called party's IP address, but the port number as well.

Following the metaphor of the phone system, a computer (which can run many services) is like a large organization that can have many employees. An IP address is like the phone number of the main switchboard of this large organization, the operating system is like the switchboard operator. Although there are many employees at this organization but each has been assigned their own phone extension (port). When you want to establish a connection, you must specify both the phone number of the company (IP address of the computer), and also the extension of the employee (port number of the service). If you, the client, attempt to establish a connection, the operator (operating system) at the switchboard will inform the right employee (service) listening at that extension (port number) that you dialed.

As you have previously been informed, to find out the IP address of the server you can look at the source column in any packet comes from the server, or the destination column of any packet that goes to the server. However, if you wanted to find out what port were used for this request then you would need to expand the Transmission Control Protocol (TCP) segment and locate the "Source port" or "Destination port" fields, respectively. As you can see from the example above, it you can verify that the web server is communicating on port 80; this port is almost always reserved for use by the web server.

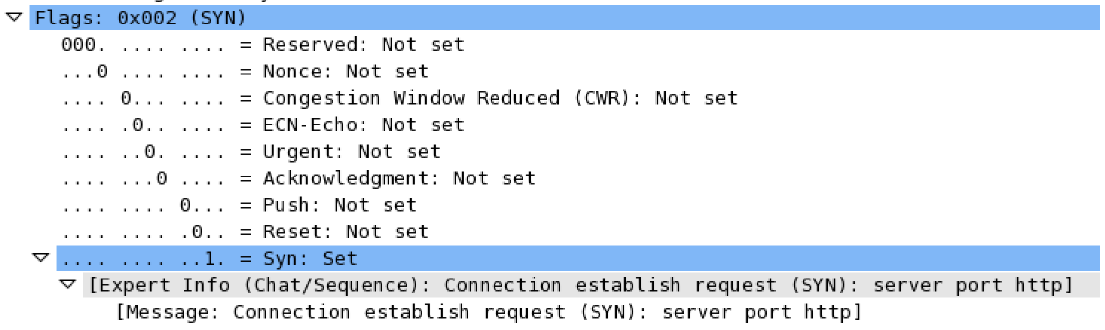
So far we have seen that ports are useful because they allow clients to differentiate between two services running on the same machine. Ports are also useful on the client side, because they allow servers to differentiate between two clients on the same machine. As a concrete example, imagine that you are using a machine with IP address Y and you decide to open two browser windows (let's call them browsers 1 and 2) and connect them both at the same time to the one web server (running on IP X, port 80) at cybersec21.com. You would expect, quite reasonably, that the data in the two browser windows will remain separate and that each browser will get its own copy of data, without any chance of data getting garbled between the two windows. In order to achieve this, the web server at X.80 must have had a way to differentiate browser 1 from browser 2 at the client machine Y. The way that this is done is that at the time when the two browsers each established their (separate) TCP/IP connections to X.80, the operating system assigned each of the two connections a source port; let's say browser 1's connection was assigned source port C and browser 2's connection was assigned source port D. Now the server knows that there are two clients: one at Y.C and one at Y.D. When the server gets requests from Y.C it sends the replies to Y.C; when it gets requests from Y.D it sends the replies to Y.D. No data corruption can occur because the client connections are assigned source ports by the OS at the calling party.

To determine the IP address of the machine where your browser is running, you would look at the destination IP address either under the Destination column or the "Dst" value in the Internet Protocol line. You can also discover what port your machine is using by following the same method as above, only this time looking at the "Destination port". You should note that this port is not the same port as the server but in fact a number above the well-known ports (0-1023). In the example that you see above, the port number of the computer is 58774.

Q10. How does a TCP/IP connection get established?

Recall that TCP/IP connections have to be established using a 3-way handshake: SYN, SYNACK, ACK. As you look through the captured packets you may be asking yourself "*How does one know whether the TCP packet is a SYN packet?*". The answer is determined by looking at the the TCP/IP packet's header, which contains 8 different binary bits of "flags", as shown below.

TODO: Replace with actual cybersec21.com packet capture

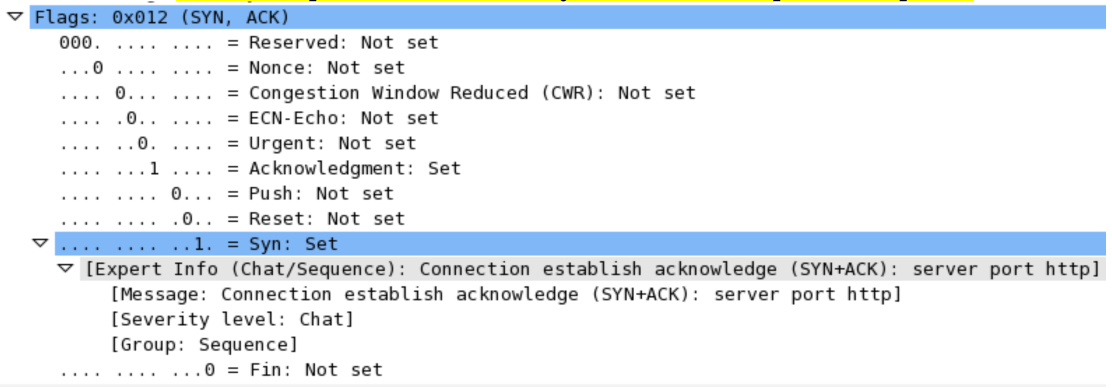


```
▽ Flags: 0x002 (SYN)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...0 .... = Acknowledgment: Not set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
  ▽ .... .... ..1. = Syn: Set
    ▽ [Expert Info (Chat/Sequence): Connection establish request (SYN): server port http]
        [Message: Connection establish request (SYN): server port http]
```

As you can see, in the packet above, the SYN bit is set to 1 (which signifies true). Given that there are no other flags set, it indicates that this is a TCP SYN packet and this is further confirmed by looking at what the "*Expert info*" message shows. In this instance it states "*[Connection establish request (SYN): server port http]*". The SYN packet contains the **calling party's initial sequence number**.

The next stage in the TCP handshake is the SYN ACK reply that is sent when the called party accepts the incoming connection. The SYNACK packet has both its SYN and ACK flags set to 1, as shown below:

TODO: Replace with actual cybersec21.com packet capture



```
▽ Flags: 0x012 (SYN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
  ▽ .... .... ..1. = Syn: Set
    ▽ [Expert Info (Chat/Sequence): Connection establish acknowledge (SYN+ACK): server port http]
        [Message: Connection establish acknowledge (SYN+ACK): server port http]
        [Severity level: Chat]
        [Group: Sequence]
    .... .... ...0 = Fin: Not set
```

In the SYNACK packet you see the **called party's initial sequence number**.

Q11. What are TCP sequence numbers used for?

It is important to notice that there are two sets of sequence numbers: one that are generated at the calling party and increases as more and more data is sent to the called party; another that are generated at the called party and increases as more data is sent to the calling party. TCP connections are bidirectional. If there is a connection between A and B, then TCP supports two simultaneous streams of data one going from A to B and one going from B to A. In a TCP/IP packet P that is being sent from A to B, the sequence number field indicates the offset in the data stream (from A to B) where the payload of P should be inserted. The acknowledgement number in P communicates to B the maximal contiguous block of sequence numbers that have been received by A so far (computed from the TCP/IP packets in the stream flowing from B to A).

Sequence numbers play an extremely important role when it comes to the TCP protocol, because they are used to (1) reassemble the received packets in the right order (since they may get sent through different parts of the Internet), and (2) to ensure that all the packets are received and acknowledged (a serious backlog in acknowledgement triggers TCP to retransmit old data).

If you want to look up what the sequence number of the TCP SYN request used to initiate the TCP connection between the client computer and the web server, then you need to locate the line under the Transmission Control Protocol that reads "Sequence number".

Q12. What about other types of HTTP messages?

Normally, when uploading files to a web server, the browser uses a HTTP POST command to inform the server that it has data to provide and to store the data in the browser's request message. So how do you locate the HTTP POST messages in Wireshark? It is a little more difficult than our past examples. 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.

Competency 5. Understanding IP

TCP uses sequence numbers, acknowledgement numbers and port numbers to provide reliable bidirectional stream between a *program* on one machine and a *program* on another machine. But, TCP is implemented “over” IP, and IP is a much simpler protocol that merely allows you to (*attempt* to) send a single packet from one *machine* to another *machine*. Each machine is identified by an IP address. An IP packet thus has in its header, a source IP address and a destination IP address. Each TCP segment is typically carried within a single IP datagram.

When an IP packet arrives into the port of a router, the router looks at the destination IP address in the packet header and decides which port to **forward** the packet out through. The process repeats in this way until the packet reaches its destination. There are other **routing protocols** by which routers exchange information in order to be able to make such forwarding decisions intelligently. Examples of routing protocols include OSPF and RIP. IP circumvents the possibility of routing loops by putting a **Time-to-Live** (TTL) field in the header of each IP packet. This integer value is decremented at each router. If the field ever reaches 0, the packet is dropped and an **Internet Control Message Protocol** (ICMP Time Exceeded) packet is sent back to the source of the dropped packet, informing them that the packet failed to reach its destination because its TTL was exceeded. Thus even if routing loops do arise, packets fail to circulate in the Internet indefinitely; once their TTL is exhausted, they are dropped from the network.

We can analyze the journey of IP datagrams sent and received by using the **traceroute** program. Traceroute operates by sending packets (**ICMP Echo Requests**, or TCP SYN packets) with successively increasing TTLs, starting with TTL=1. Each router receives the packet, decrements the TTL value; if the TTL value reaches 0 the packet is dropped and the router sends an **ICMP Time Exceeded** message back to the source. In this way, traceroute uses the returned Time Exceeded messages to build the list of routers that packet are expected to traverse as they journey towards the destination address.

Q13. What path do IP packets take to get to cybersec21.com?

Before beginning this part of lab, you’ll probably want to review section 3.4 of RFC 2151[8] to update yourself on the operation of the traceroute program. You’ll also want to have RFC 791[9] on hand as well, for a discussion of the IP protocol.

With the Unix traceroute command, the size of the UDP datagram sent towards the destination can be explicitly set by indicating the number of bytes in the datagram; this value is entered in the traceroute command line immediately after the name or

8http://www.rfc-editor.org/rfc/rfc2151.txt

9http://www.rfc-editor.org/rfc/rfc791.txt address of the destination. For example, to send traceroute datagrams of 56 bytes towards cybersec21.com, we do: traceroute cybersec21.com 56

In your traces, you should be able to see the series of ICMP Echo Request (in the case of Windows machine) or the UDP segment (in the case of Unix) sent by your computer and the ICMP TTL-exceeded messages returned to your computer by the intermediate routers. When looking at an IP packet, you can find out what the upper layer protocol field is by looking at the "*Protocol*" line. If you look at the first ICMP Echo Request message sent you should see Protocol: ICMP (0x01).

Q14. Why are IP packets sometimes fragmented?

Enter three traceroute commands, one with a length of 56 bytes, one with a length of 2000 bytes, and one with a length of 3500 bytes. Each time, before issuing the command, 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). Let the traceroute get to the destination, then stop Wireshark tracing.

If you are going to delve into specifics with the IP layer then it would be useful to know how to determine the size of the IP datagram and how it is broken down. The minimum packet length of an IP datagram would be 20 bytes as that is the size of the IP header with 0 bytes of data. To calculate the payload portion of the IP datagram you merely do the simple calculation: Total length – Header length = Payload length.

IP packets are often fragmented which allows for the data to be broken up into smaller pieces to transfer across the network and then rebuilt at the other side. To resolve whether the IP datagram can be or has been fragmented or not, you need to look at the flags in the IP header. There are three different flags worth mentioning: 1) Reserved bit, 2) Don't Fragment and 3) More Fragments.[10] When the "Don't Fragment" flag is set it means that the datagram should not be fragmented. If the "More Fragments" flag is set then it indicates that the IP datagram itself is fragmented and there are more fragmented packets to come. You can determine whether this is the first fragment or not by looking at the "Fragment offset" value. If it is set to 0 then you know that it is the first in the series of fragments. If you were to look at the next ICMP packet that was sent and locate the fragment offset value this time, it will have changed. This is so the datagram can be rebuilt at the other side by determining how far from the start of the datagram this particular part fits. You know there are no more further fragments due when the "More fragments" field is set to 0.

10http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/ip-packet.html

Competency 6. Understanding HTTP Security

Q15. How does HTTP implement password protected directories?

Start packet capture. Open a browser and go to the following password protected website

http://cybersec21.com/protected/secret.htm

The username is “cybersec21” and the password is “let-me-in-now”. Now go back to Wireshark, stop the packet capture and filter the packets so you only see HTTP traffic.

To determine what the server’s response, i.e. the status code and phrase, to your initial HTTP GET message was you should look at the info column in the packet list window or expand the Hypertext Transfer Protocol in the packet detail window. You should see a Status Code: 401 and the Response Phrase: Authorization Required.

After you entered the user credentials, assuming you did it correctly, you will see that there is a second HTTP GET message from you computer to the server. If you look at those details you should see that there is a new field included in the message this time. The new Authentication field should be shown with an encoded message (d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms=). This is because the username (cybersec21) and password (let-me-in-now) that you entered are encoded in the string of characters (d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms=) that you see 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 Base65 decode website[11] and enter the base64-encoded string d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms= and press decode. Voila! You have translated from Base64 encoding to ASCII encoding, and thus should see both your username and password! Since anyone can download a tool like Wireshark and sniff packets (not just their own) passing by their network adapter, and anyone can translate from Base64 to ASCII (you just did it!), it should be clear to you that simple passwords on WWW sites using .htaccess files *are not secure* unless additional measures are taken.

11https://www.base64decode.org