CS 2600 Packet Sniffer Lab
Using Wireshark v3.0 and Npcap v0.99 or libcap

Lab Exercise 4: Filtering Packets

Approximate time required: 2 - 3 hours

Introduction

In Wireshark Lab 3, you installed Wireshark and captured some ICMP (ping) traffic. You probably noticed that it was challenging to locate and isolate the “interesting” ICMP packets from other packets that were unavoidably captured. In this lab exercise, we will learn several methods for filtering “interesting” packets from the “uninteresting” ones. We’ll also learn how to store captured network traffic for future reference.

This lab can be completed using a single computer with an Internet connection accessed through that computer’s Ethernet or 802.11 WiFi adapter. Or it can be completed in the Networking Research Lab using Ethernet. It will be a bit simpler if you do not run Wireshark in a virtual machine. Because some Wireshark functions are not very intuitive, you’ll need to follow the directions for configuring the filters carefully. As in Lab 3, if you are running a firewall on your own computer that filters (blocks) ICMP Request and/or Reply messages (“pings”), disable that filter for the duration of this lab.

Generating and Capturing Web Traffic

Web browsers use an Application layer protocol called the HyperText Transfer Protocol (HTTP) communicate with web servers. Wireshark can dissect HTTP headers, and can also show you the embedded content, for example HTML, as text. Wireshark cannot render HTML graphically, as a browser does.

Open your web browser. If you have the option, use Firefox, and avoid Chrome if possible. Type in the name of a relatively simple *non-https* website such as `http://www.manpagez.com/`, `http://www.oidview.com/mibs/detail.html` or `http://www.base64online.com/mac address.php`, *but don’t download the page just yet.* (Chrome will download the page, and pre-load other links on that page before you press ENTER. If you must use Chrome, wait to type in the website URL until after you have started the Wireshark capture.)

Start Wireshark, click **Capture | Options...**, select the active interface, and click **Start**.

Back in your browser, hit ENTER to download the web page.

After the page is visible in the browser, go back to Wireshark and stop the capture using 🟥.

In Wireshark, view the “Protocol” column while scrolling through the *Packet List* (upper) pane. You should see packets carrying a variety of upper-layer protocols (mostly TCP and HTTP, with some random TLS, ARP, UDP, DNS, ICMP, etc.). Our intent is to collect a mixture of packet types, which we will then filter to view packets of interest.

> If you don’t see any HTTP packets in the “Protocol” column, check the address bar in your browser to see if the browser has changed the “`http`” in your URL to “`https`”. If so, try a different website.
>
> If the only packet types that you see are TCP and HTTP packets, repeat the previous step using a different website, in order to collect a greater variety of packet types. Or, open a command window, type one of the following commands, then try the previous web page download again:
>
> (For MS Windows 7, 8.1 and 10)
> ```
> ipconfig /flushdns
> ```
>
> (For Linux)
> ```
> /etc/init.d/nscd restart
> ```
>
> (For macOS 10.7 and later)
> ```
> sudo killall -HUP mDNSResponder
> ```

The protocols listed in the Wireshark “Protocol” column are not all of the same category, in the sense that they don’t all function at the same layer of the 4-layer Internet model. TCP and UDP are Transport (layer 3) protocols in the Internet Model, while HTTP and DNS belong in the Application layer (4). ICMP is a layer 3 protocol. Wireshark identifies them based on which layer is most likely to be of interest to the user. For example, an HTTP packet uses a TCP header, but Wireshark calls it an “HTTP” packet, rather than a “TCP” packet. However, a TCP control message that does not contain Application layer data such as HTTP is labeled as “TCP”. Similarly, DNS messages are labeled as “DNS”, even though they are carried within UDP packets.

After you have successfully captured a mixture of different protocols, save the capture information to a file with **File | Save As...** Enter a file name in the *File name* field and leave the *Save as type:* set to *“Wireshark/. -pcapng (*.pcapng; *.pcapng.gz;...”.*

> If you were planning to use this file with another network analysis tool which was more than a couple of years old, you would probably want to save it in the older ‘“Wireshark/tcpdump/... - pcap (*.pcap; *.pcap.gz;...” format. (This might be the case if you were using the *tcpdump* utility on an old Linux embedded system, for example.) Both `*.pcapng` and `*.pcap` are supported by a wide range of open source and commercial network analysis tools .

Colorizing Captured Packets (Optional)

By default, Wireshark differentiates the various types of traffic in the *Packet List* pane by assigning different colors to different types of packets. If you wanted to customize your colorizing scheme, you would turn off the default colors by clicking **View | Colorize Packet List**. Then you would click **View | Coloring Rules. | +** to define your own colors. Play around with this if you like, then close the *Coloring Rules* window.

Display Filters

Colorization is useful when you need visual separation between different types of packets in the *Packet List* pane. But what if you only want to display one or more specific packet types, without deleting other types of packets from the receive buffer or file where they are stored? *Display filters* solve this problem.

First, we'll filter out all non-HTTP (non-Web) traffic, using the capture file that you just saved, and should still be viewing in Wireshark. Click **Analyze | Display Filters...** Click **+** at the lower left. In the "Name" column, double click on `New display filter` and name your new filter `HTTP Only`.

In the "Filter" column to the right, double click on `ipaddr == host.example.com` and replace it with `http`. (Filter strings are case sensitive, so `HTTP` won't work.) The background of the "Filter" field will remain pink until you complete a legal filter string. Then it will turn green.

Click **OK** to save the filter. Whenever it is activated, this filter will exclude all packets from the display *except* those that contain HTTP. (In Wireshark, to "filter HTTP" means "display HTTP only", not "hide HTTP".)

To activate your new display filter, click the small bookmark icon a t the upper left, and select your "HTTP Only" filter. You will observe that your HTTP filter string is displayed in the green *Filter* field near the upper left corner of the main Wireshark window. In the *Packet List* pane's "Protocol" column, you should now see mostly HTTP packets, and possibly some HTTP-related TCP, OCSP and/or SSDP packets.

> If you don't seen any HTTP packets when your HTTP filter is enabled, it may be because you connected to a website that uses HTTPS, the encrypted version of HTTP. In that case, try a new capture using a different website to ensure that HTTP is being used. (HTTPS adds security to the basic HTTP web protocol, and most websites now use HTTPS predominantly or exclusively.)

Cancel your "HTTP Only" filter by clicking the **Clear** button at the upper right. All captured traffic will be displayed once again. Your "HTTP Only" filter will remain in your list of display filters for future use.

Next, we'll create a more selective filter that displays only HTTP messages *sent from your browser.* We want to view HTTP only, as before, but only the packets with your browser's IP address in the IP header's *Source Address* field.

From a Windows command line or macOS or Linux terminal window, run `ipconfig` (Windows) or `ifconfig` (macOS and Linux). Note your computer's IPv4 address, which will be in dotted decimal format, similar to this: `192.168.1.100`. (If the IPv4 addresses for multiple network adapters are displayed, be sure to obtain the address for the Ethernet or WiFi adapter that Wireshark is using. Be especially careful if you are running virtual machines.)

This time we'll define our display filter using a method suited to situations where you don't know the syntax of the filter string in advance. Click **Expression...** at the top right. When the *Display Filter Expressions* dialog box opens, scroll down to **IPv4** (pressing the I key will get you close). In the "Field Name" list, expand **IPv4 - Internet Protocol Version 4** by clicking on the ">", scroll down, and select **ip.src - Source** (this is the sender's IP address field in the Layer 2 IPV4 header). You will see `ip.src` displayed in the green filter field at the bottom.

Select == from the *Relation* field at the upper right, and then fill in the *Value* (IP address) field with the IPv4 address of your computer, using the standard dotted decimal format (for example, `ip.src == 192.168.1.100`). Click **OK**. Your (incomplete) filter string will be displayed in the *Filter* field at the top.

Edit the *Filter* field by appending `&& http` to the IP address. The resulting display filter, something like `ip.src == 192.168.1.100 && http`, says "display only the packets which were sent by the computer with IPv4 address 192.168.1.100 AND which contain HTTP messages". Click the arrow at the right to activate your filter, and check the *Packet Details* pane below to verify that only HTTP-related packets, with your computer's IP address listed in the "Source" column, are being displayed.

As you become familiar with the more common protocol header field names, you'll be able to quickly create display filter strings that will show you only the packets of interest. Use the technique described on page 3 whenever you think you might want to save a complicated filter for future use, and remember that display filter strings require lower case letters.

Click the **Clear** button at the upper right in the Wireshark main window to cancel the filter and display the entire capture file again.

Working down at the Network Interface/Hardware layer (1) in the Internet Model, you can create a display filter that is the functional equivalent of the previous filter by replacing the client's IP address with its Ethernet (MAC) address in the display filter. (This will work the same way if you are using a WiFi connection). For both types of networks, the syntax would look something like:

```
eth.src == 00:04:5a:82:f6:41 && http
```

Let's try that. Find any packet in the upper *Packet List* pane that shows your computer's IPv4 address as the source, and select that packet.

Down in the *Packet Details* pane, find your 6-byte hexadecimal Ethernet or WiFi source address in the "Ethernet II" header of that frame. (In Windows, you can also find it using `ipconfig /all`). Create a Hardware layer filter similar to the one shown above, and test it. You should see the same list of HTTP and related packets displayed in the *Packet Details* pane that you saw when filtering for your IPv4 source address. This time, however, Wireshark is filtering them based on the source address in the Ethernet or WiFi frame header.

If Wireshark recognizes the manufacturer code (OUI) in the first three bytes of your Ethernet address, it will substitute the manufacturer's name, for example "AsustekC_21:45:4c". In that case, the numeric Ethernet source address will be listed in parenthesis to the right, as in "00:13:d4:21: 45:4c".

Clear this new display filter when you are finished testing. For more detailed information creating Wireshark display filters, see `https://www.wireshark.org/docs/dfref/`.

Capture Filters

In the previous section you captured all available packets and created display filters in order to view only the packets of interest within that collection. In some situations, it makes sense to *capture* only the specific types of packets that you are interested in. If a large volume of irrelevant traffic is being transmitted, or if you are sniffing traffic over an extended period of time on a busy network, a *capture filter* will limit your capture file to a reasonable size. A capture filter may also help to ensure that the desired packets aren't inadvertently missed by a slow computer or adapter when a high volume of traffic is present.

The syntax of Wireshark capture filter strings is different from display filters. This is because capture filter strings are "passed through" by Wireshark down to the Npcap or libpcap driver, whereas *display* filtering is controlled from within Wireshark. The ability to create capture filters is somewhat dependent on knowledge of upper level-protocols that we haven't covered yet. Nevertheless, we can try a few simple examples to get a feel for how capture filters work.

We'll begin with a capture filter that collects only web (HTTP) traffic. Because HTTP typically uses TCP port 80 to indicate that the destination application is a web server, we will create a capture filter that only captures traffic sent to or from that port. "HTTP" is simply an alias for "port 80", the Internet standard demultiplexing key for any web server (Apache, IIS, etc.). Npcap and libcap use the assigned port number to reference various Application layer protocols. For example, Internet email uses port 25 and domain name (DNS) lookups use port 53. Port numbers do not reference protocols below the Application layer, such as TCP or IP.

In Wireshark, click **Capture | Capture Filters...** Click **+** and create a new capture filter named "HTTP Only". For the capture filter string to the right, use `tcp port 80`. (`http` won't work as a capture filter string.) *Specify all capture filter strings using lower case letters.* (Note that capture filters use a different syntax from the equivalent display filter, which would be `tcp.port == 80`, or simply `http`.) As with display filters, we are filtering "for" HTTP packets, not filtering them out. Click **OK** to add your new filter to the end of the capture filter list.

Click **Capture | Capture Filters.** again and create a second capture filter called "ICMP", to look for ICMP ("ping") messages. For this capture filter string, use `ip proto \icmp`. This filter will capture only IP packets that contain an ICMP payload, such as a ping echo request. Click **OK** to save this capture filter. Unlike HTTP, ICMP is not an application layer protocol, so ICMP is not referenced by a port number (see figure 1 below.)

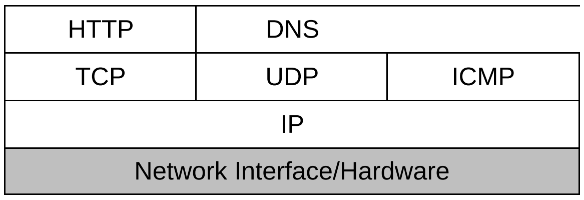

Figure 1. Relationship Between Protocol Layers

Some background for those who are interested: An IPv4 packet carrying an ICMP message as its payload will contain a "1" in the IP header's *Protocol* field. `ip proto \icmp` means "look at the IP header in each packet as it arrives, and capture only the ones that are carrying ICMP as the IP packet payload". The backslash is an escape character, required because "`icmp`" (along with "`tcp`" and "`udp`") are also Wireshark and Npcap keywords. So the IPv4 *Protocol* field works kind of like a TCP or UDP port number, but at the next layer down. Both IP *Protocol* numbers and TCP or UDP port numbers are demultiplexing keys. It's helpful to know that in TCP/IP-related standards, "protocol" usually means "whatever comes next after (to the right of) this header". In other words, "whatever is in the layer above".

Note that a packet which is not captured by Wireshark will still be delivered to the destination application. A capture filter just tells Wireshark which types of packets it should copy, as they are sent and received.

If you currently have an active display filter, click the **Clear** button near the top of the Wireshark main window to cancel it. Close the current capture file, if necessary, with **File | Close**.

In the **Capture** section near the bottom of the main Wireshark window, select the active interface again. Use the green bookmark button immediately above the interfaces to select your ICMP capture filter. Click **Capture | Start** to begin capturing packets.

Open your browser and download the same web page that you used in the first part of the lab exercise. Wait a few seconds, then open a command line window and `ping` the web server that you previously downloaded the web page from. Wait a little bit longer to allow for other miscellaneous traffic to flow (and be ignored). Then stop the packet capture, and examine the *Packet List* pane. In Wireshark, you should see ICMP traffic, but no HTTP, TCP or other packet types. (You may see some other types of ICMP messages besides the Echo (ping) requests and replies.)

> If your ping fails on a Windows computer, it may be defaulting to an IPv6 ping. Try `ping -4 <target>` to force an IPv4 ping.

Consider what has happened. You know that the web page was downloaded because you can see it in your browser. However, Wireshark did not capture the corresponding HTTP packets.

Close this capture file without saving it, and use **Capture | Options** to change the capture filter string at the bottom of the *Capture Interfaces* window to enable your "HTTP Only" filter. (Only one capture filter can be enabled at a time.)

Start another capture, and repeat the same process, downloading the same web page again, and then pinging the same server. After you stop the capture, you should see HTTP traffic and some related TCP packets, but no ICMP traffic from the ping. Leave this capture file open.

For detailed information on the syntax for Wireshark capture filters, see `http://www.manpagez.com/man/7/pcap-filter/`. The Wireshark User's Guide contains some additional capture filter examples in section 4.13.

Questions

1. As noted, Wireshark may use "protocol" to refer to Network (Ethernet), Internet (IP), Transport (TCP) or Application (HTTP) layer protocols from the Internet model, as shown in figure 2 below. If "Protocol" is capitalized, it may refer specifically to the **Protocol** field in the IP header.

Network (Ethernet) Header	Internet (IP) Header	Transport (TCP) Header	Application (HTTP) Header	Optional App Data (HTML)

Figure 2. Relationship of Headers in HTTP Packet

Using the file of capture-filtered HTTP messages obtained in the previous step, select one of the packets that lists HTTP (not TCP) as its protocol in the (upper) *Packet List* pane's "Protocol" column.

In the *Packet Details* (middle) pane, verify that there is a line in the dissection of that packet that says "> Hypertext Transfer Protocol". This confirms the presence of an HTTP header.
Create a new *display* filter called "TCP without HTTP" that only displays TCP traffic which does

not contain an HTTP payload. In the filter string, you can specify the boolean AND as `&&` or `and`. Boolean NOT can be specified with `!` or `not`.

After you have created the filter and enabled it, verify that it works by selecting several of the TCP packets displayed in the *Packet List* pane, and confirming that there are no HTTP headers present in the *Packet Details* pane.

Click **Analyze | Display Filters...** and select your "TCP without HTTP" filter in the "Wireshark: Display Filters" window.

Make a screen shot of the "Wireshark: Display Filters" window showing the filter string that you created. Paste that into your Lab Report and label it "Question 1, part A.".

Click **OK** in the Wireshark **-** Display Filters window and, with this filter still active, scroll the Wireshark *Packet List* pane (if necessary) so that the first packets captured are at the top of the window.

Make a screen shot of the Wireshark main screen, and paste it onto your Lab Report with a caption above it that says "Question 1, part B.".

Clear your display filter and **Close** the current capture file (no need to save the captured packets).

2. *This question is kind of tricky, so read and follow the instructions carefully.*

Before you communicate with any new destination using a URL or Internet domain name, a Domain Name System (DNS) query must be transmitted to a domain name server to obtain the IP address that corresponds to that domain name. This happens when you send an email or use a web browser, for example. DNS queries use UDP port 53 to identify both the domain name server and the DNS client that sent the query.

In a command line window, enter (*but don't execute*) the command to flush the DNS cache again using the appropriate command (`ipconfig /flushdns` in Windows, or see the top of p. 2).

Using the techniques explained in the **Capture Filters** section above, create a *capture* filter that will collect only DNS messages being transported in UDP packets (see figure 1). This will be generally similar to the `tcp port 80` capture filter that you used to capture HTTP. Enable your DNS filter, *but don't start the packet capture yet..*

Close and reopen your browser. Open a new tab and enter `manpagez.com` or one of the other website addresses on p. 1 into your browser's address bar, *but don't press **ENTER** yet.* Close all other tabs.

Now do the following in rapid succession:

- Flush the DNS cache
- Start the Wireshark capture
- Load the web page
- Stop the Wireshark capture

Use the "Protocol" column in the (upper) *Packet List* pane to verify that all of the captured packets are DNS traffic. If not, check your DNS-only capture filter.

Select the first DNS packet in the (upper) *Packet List* pane and use the “Info” column to the right to verify that it contains the domain name that you entered in your web browser. (You may need to move the horizontal scroll bar all the way to the left.) If you see a different domain name, scroll down through the Packet List until you find the domain name that you entered.

> If there are a large number of DNS packets, try creating a display filter that references the target domain name, similar to this:
>
> `dns.qry.name == manpagez.com`
> (Use the website that you loaded, don’t include “`http://`“ or “`www`”)

Select the DNS packet with the target domain name and expand the corresponding DNS query header in the *Packet Details* pane by clicking the “>” next to “Domain Name System (query)”. Then click the “>” next to “Queries” below. Look for the domain name from the URL that you typed into your browser.

> The DNS client on your computer, and your browser (in some cases) caches the IP address returned by each DNS query, so a new DNS query will not normally be generated for a recently accessed server. If you need to repeat this task, close and reopen your browser, and run the appropriate command (such as `ipconfig /flushdns)` before you retry it.

Make a screen shot that shows this expanded DNS query including the domain name in the (middle) Packet Details pane, paste it into your Lab Report, and label it “Question 2.”.

3. Lastly, we’ll filter some Address Resolution Protocol (ARP) traffic. ARP is a protocol that allows a computer with an IP packet waiting to be transmitted to find the MAC (Ethernet or WiFi) address that corresponds to the IP address in that packet’s IPv4 header. This must be done *before* a packet can be framed and transmitted to a new (previously unknown) MAC address. After the MAC address is discovered by ARP, the IP address and its corresponding MAC address are cached, as with DNS.

ARP is a “low-level” protocol, meaning that ARP messages are transmitted as the payload of an Ethernet or WiFi frame, not as the payload of an IP packet, so no IP header is used (see figure 3. below).

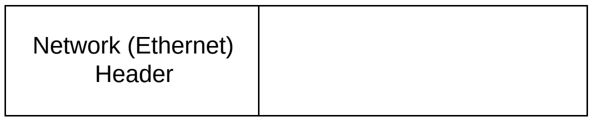

Figure 3. Structure of ARP Message

As a result, Wireshark considers ARP to be a protocol distinct from IP. The syntax for an ARP capture or display filter is simply `arp`.

Build an ARP *capture* filter, save it, and enable it. *Don’t start the capture just yet...*

Find the IPv4 address of another computer *on your own local network* (Ethernet or WiFi) by typing `ipconfig` (Windows) or `ifconfig` (macOS and Linux) *on that computer’s command line*. (You can use a smart phone or tablet as the target if it is connected to your WiFi network). Write down the target address.

Clearing (flushing) the ARP cache will force an ARP request to be transmitted when you ping the target computer. Clear all existing entries in the ARP cache by typing one of the following from the command line:

> (For MS Windows)
> `arp -d *`
>
> > If this command fails because you are running Windows User Account Control, try right clicking on “Command Prompt” (under “Windows System” on the Start Menu) and selecting **More | Run as administrator**.
>
> (For Linux, try...)
> `ip neigh flush all` (If this fails, wait about two minutes for the entries to age out)
>
> (For macOS)
> `sudo arp -a -d`

In rapid succession:

- Clear the ARP cache
- Start a Wireshark capture
- `ping` the IP address of the other computer
- Stop the Wireshark capture

It may take a few tries to get this to work. If you need to repeat the ping, use a different destination each time, or re-clear the ARP cache.

When you have successfully performed an ARP capture, select the first ARP packet in the *Packet List* pane. View the *Info* column to the right and verify that it says “Who has <target IPv4 address>?”, where the target is the computer that you were pinging. If not, scroll down until you find the correct ARP message.

> If you are using a busy network, you might have to search through quite a few ARP messages to find your request. Try a display filter set to
> `arp.dst.proto_ipv4 == <target IPv4 address>`.

After you have located the correct ARP message, expand the Ethernet II header and ARP header in the *Packet Details* pane by clicking the “>” next to “Ethernet II” and “Address Resolution Protocol (request)”.

> The ARP request message triggered by the ping will probably be repeated. If none of the ARP requests are successful, you will not receive an ARP response and your ping request will return a *Destination host unreachable* ping response. If this occurs, you can still use your ARP request to answer this question.

Note that the *Type* field in the Ethernet Type II header is 0806_{16} (ARP), rather than 0800_{16} (IPv4). This explains why TCP/IP environments normally use Ethernet Type II frames. Unless an inner 802.2 Logical Link Control header is also present, there is no equivalent demultiplexing key in an 802.3 frame. In that situation, there is no direct way for the Network Interface/Hardware layer to differentiate between an 802.3 frame containing an IPv4 packet, an 802.3 frame that contains an ARP message, and an 802.3 frame that contains an IPv6 packet. (If this doesn’t make sense, review the frame format diagrams in the Lecture 18 slides.)

Make a screen shot that shows this expanded Ethernet header and ARP request in the Packet Details pane, paste it into your Lab Report, and label it “Question 3.”.

Summary of Deliverables

- Two screen captures for question 1 (Parts A & B).
- DNS screen capture for question 2.
- ARP screen capture for question 3.

Submit your Lab Report to Canvas by the due date. If you are working in the lab, save your files (if you wish) and shut down your workstation. If you are working on your own computer and have modified your firewall settings, *don’t forget to restore them.*