Introduction to Wireshark

Fall 2019

Contents

0 Set Up

For this lab we will use the class provided VM. If you need help setting it up, refer to `Using the Vagrant VM`[1]

1 Introdution

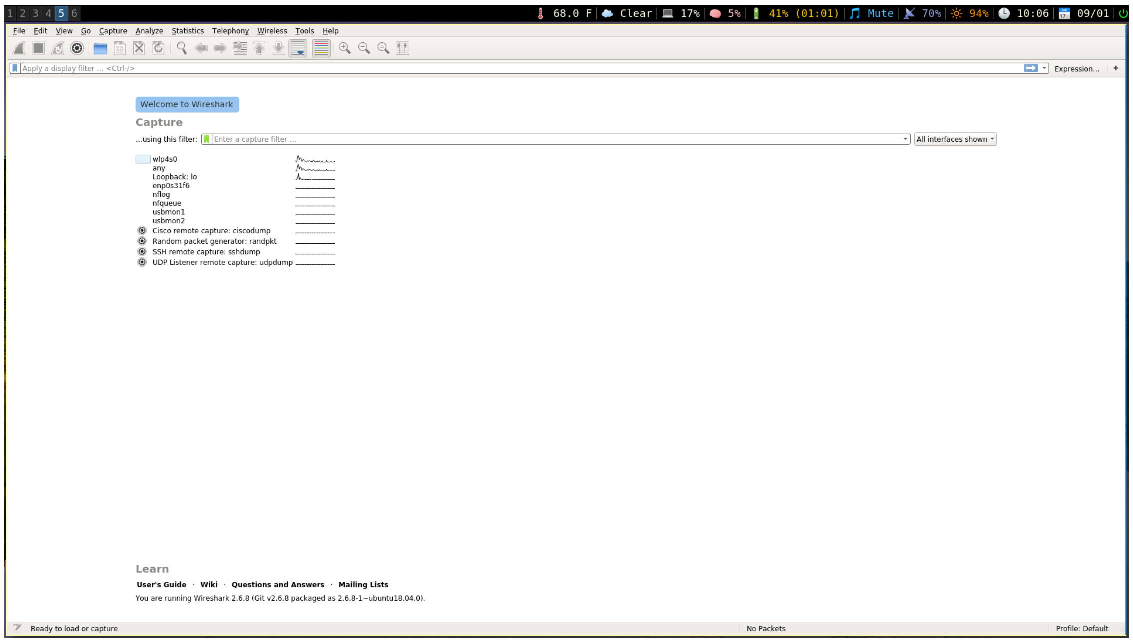
“Wireshark is the world’s foremost and widely-used network protocol analyzer. It lets you see what’s happening on your network at a microscopic level and is the de facto (and often de jure) standard across many commercial and non-profit enterprises, government agencies, and educational

[1]Regardless of what environment you use for development, we will be grading on the class provided VM. Make sure your project is fully functional on the class provided VM or a department machine!

institutions."[2] Today we will be using Wireshark to analyze the packets that come over the wire when you fetch a web page.

2 Configuring Wireshark

To begin, open up Wireshark (simply run `wireshark` from a terminal). You should be presented with a screen like this:


1 2 3 4 5 6
68.0 F | Clear | 17% | 5% | 41% (01:01) | Mute | 70% | 94% | 10:06 | 09/01
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
All interfaces shown
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists
Ready to load or capture
No Packets
Profile: Default

```
wlp4s0
any
Loopback: lo
enp0s31f6
nflog
nfqueue
usbmon1
usbmon2
Cisco remote capture: ciscodump
Random packet generator: randpkt
SSH remote capture: sshdump
UDP Listener remote capture: udpdump
```

You are running Wireshark 2.6.8 (Git v2.6.8 packaged as 2.6.8-1~ubuntu18.04.0).

2.1 Interfaces

Each of these options represents an interface on which packets can be received. In general the three interfaces you will care about are `Loopback` (used for the computer sending things to itself), `Ethernet`, and `Wi-Fi`. Note that not all machines will have the same naming for these devices, and in some cases they may not all be available.

For this lab, we will select the interface used for Wi-Fi. If you are unsure which interface this is, you can run the terminal command `route` and the interface listed under `Use Iface` in the `default` entry should be the interface you select in Wireshark.

3 Analyzing packets

Once we have Wireshark open on the appropriate interface, it automatically begins capturing packets. You may see packets appearing in the window corresponding to various operating system functions. Don't worry about these for right now.

[2]Source: https://www.wireshark.org/

Next, run the command: `wget -r -l1 cs.brown.edu`. You should see a large dump of data in Wireshark. If at any time you want to clear your current log of packets you can stop the current capture by clicking the big red box, then starting another capture by clicking the blue shark fin. Note that you have the option to save the old capture when you click the fin to start a new capture (which may be useful for debugging on later projects).

3.1 Fetch and Start a Capture

When the `wget` command finishes, click the red box to stop the capture (once the capture stops, Wireshark will no longer log additional packets). Look under the `protocol` column. As you scroll down the log of packets, you should see packets from each of the following protocols: DNS, TLS, ARP, TCP, and HTTP. We will be focused on ARP packets for this part of the lab. (Don't worry if you can't find any ARP packets, we'll get to that shortly!)

3.2 Filter

To examine a single protocol, we can use a filter. In the filter bar, enter "arp" and click the right facing arrow. There are many different types of filters that can discriminate between packets based on more than just protocol; if you would like to see and select from a wide range of filters click the "Expression..." button next to the filter bar and browse through the filter options. Later in this lab we will try some more sophisticated filters.

If you do not find any ARP packets when you apply the filter it is likely because the ARP information is cached on your system. To get around this, first clear the log of packets then start a new capture (red stop button then blue shark fin). Then, run `arp -n` to see your ARP table, `sudo ip -s -s neigh flush all` to clear your ARP table, then `arp -n` again to confirm the table has been cleared. Now run the command `wget -r -l1 cs.brown.edu` again to fetch the webpage. As before, click the red box to stop the capture once `wget` finishes.

Now, look at your ARP packets - you should see at least two. Under the info column Wireshark tells you exactly what the ARP packets are doing. We can see a request ARP packet that under `Info` says, "Who has <requested IP>? Tell <requesting IP>." and a response ARP packet that under info says, "<requested IP> is at <destination Ethernet address>". If you click on a single ARP packet you will see in the box below the packet log that Wireshark will actually dissect the packet for you (in hex), and in the third box you can see the raw bytes of the packet. If you hover over certain bytes Wireshark will let you know what part of the packet those bytes belong to. Conversely if you click on a dissected field of the packet Wireshark will tell you which raw bytes it grabbed the data from to attain that field of data.

3.3 Dissect

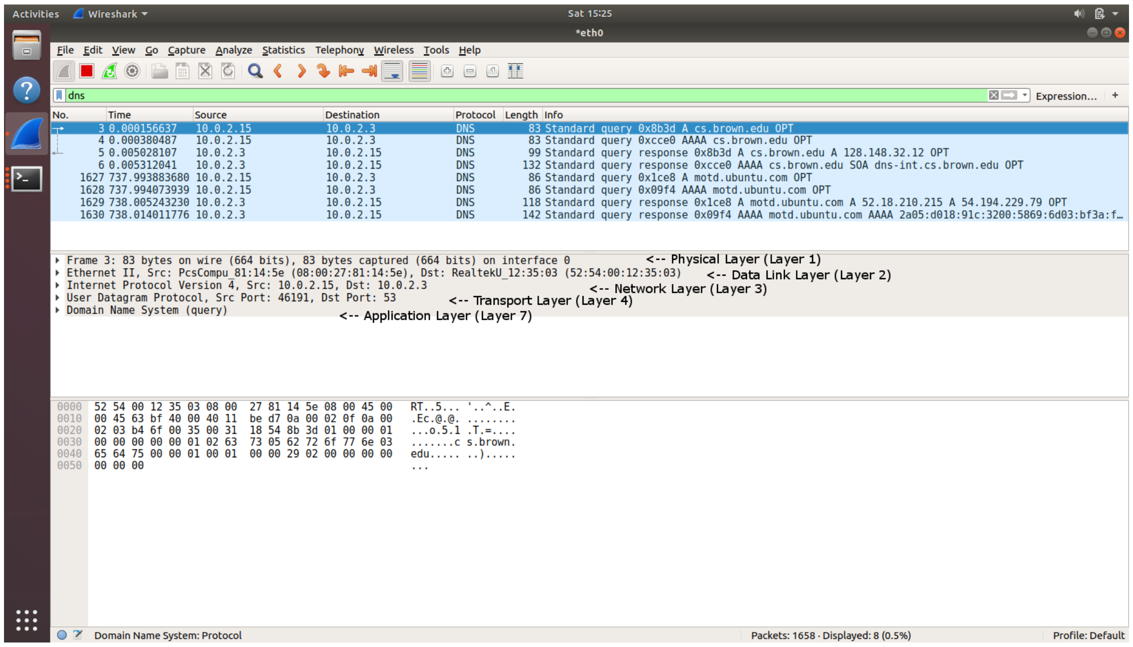
To dive deeper into dissecting, let's examine a DNS query and response. You will learn about DNS later in the course, but for now know that DNS is the primary way that a computer gets the IP address for a web domain (in our case, `cs.brown.edu`).

3.3.1 DNS Query

Our machine will emit a DNS query, and receive a DNS response. Filter on "dns" and click on the first DNS packet (the query). Collapse all the dissected fields, and note that each top level field is a header for a different protocol. The top field is the outermost header, in the sense that for anything parsing the header, the parser would find the top most header, then the next header down the list, then so on until the parser read all the bytes in the packet. Also note our DNS packet contains a header for each layer of the OSI model.

3.3.2 OSI Model within the DNS Query

The first field represents the entire packet of raw bytes that Wireshark received over the wire (from the Physical layer) this is the only field not constructed from a protocol header. The next field is constructed from the Ethernet header that Wireshark parsed (Link layer), within the Ethernet frame is the IP packet (Network layer), within the IP packet is the UDP datagram (Transport layer), and the lowest header is for the DNS query (Application Layer).


Activities
Wireshark
Sat 15:25
*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
dns
Expression...
Domain Name System: Protocol
Packets: 1658 · Displayed: 8 (0.5%)
Profile: Default

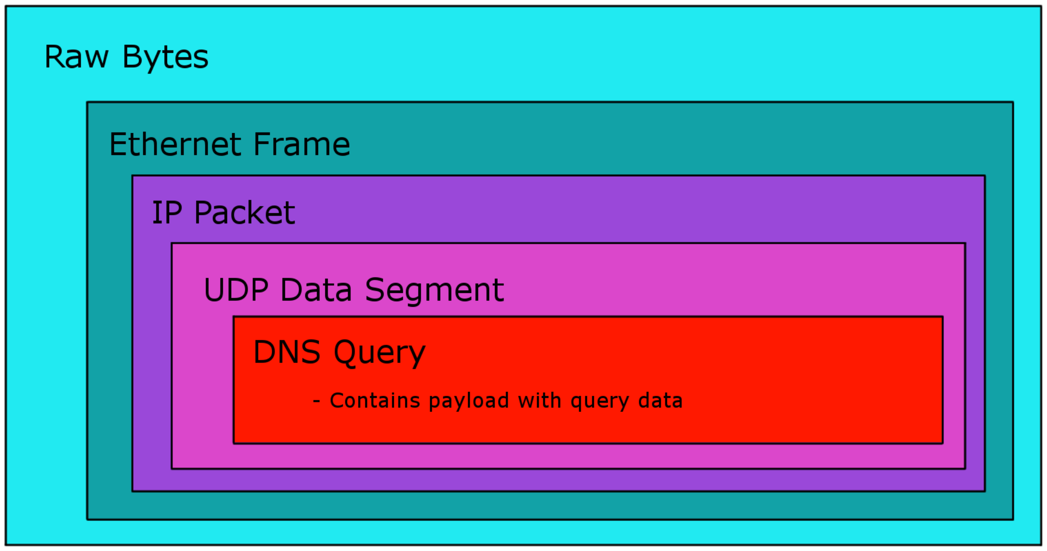
```
No.  Time           Source     Destination  Protocol Length Info
   3 0.000156637    10.0.2.15  10.0.2.3     DNS        83 Standard query 0x8b3d A cs.brown.edu OPT
   4 0.000380487    10.0.2.15  10.0.2.3     DNS        83 Standard query 0xcce0 AAAA cs.brown.edu OPT
   5 0.005028107    10.0.2.3   10.0.2.15    DNS        99 Standard query response 0x8b3d A cs.brown.edu A 128.148.32.12 OPT
   6 0.005312041    10.0.2.3   10.0.2.15    DNS       132 Standard query response 0xcce0 AAAA cs.brown.edu SOA dns-int.cs.brown.edu OPT
1627 737.993883680 10.0.2.15  10.0.2.3     DNS        86 Standard query 0x1ce8 A motd.ubuntu.com OPT
1628 737.994073939 10.0.2.15  10.0.2.3     DNS        86 Standard query 0x09f4 AAAA motd.ubuntu.com OPT
1629 738.005243230 10.0.2.3   10.0.2.15    DNS       118 Standard query response 0x1ce8 A motd.ubuntu.com A 52.18.210.215 A 54.194.229.79 OPT
1630 738.014011776 10.0.2.3   10.0.2.15    DNS       142 Standard query response 0x09f4 AAAA motd.ubuntu.com AAAA 2a05:d018:91c:3200:5869:6d03:bf3a:f…
```

```
▸ Frame 3: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface 0        <-- Physical Layer (Layer 1)
▸ Ethernet II, Src: PcsCompu_81:14:5e (08:00:27:81:14:5e), Dst: RealtekU_12:35:03 (52:54:00:12:35:03)   <-- Data Link Layer (Layer 2)
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.3        <-- Network Layer (Layer 3)
▸ User Datagram Protocol, Src Port: 46191, Dst Port: 53        <-- Transport Layer (Layer 4)
▸ Domain Name System (query)        <-- Application Layer (Layer 7)
```

```
0000  52 54 00 12 35 03 08 00  27 81 14 5e 08 00 45 00   RT..5... '..^..E.
0010  00 45 63 bf 40 00 40 11  be d7 0a 00 02 0f 0a 00   .Ec.@.@. ........
0020  02 03 b4 6f 00 35 00 31  18 54 8b 3d 01 00 00 01   ...o.5.1 .T.=....
0030  00 00 00 00 00 01 02 63  73 05 62 72 6f 77 6e 03   .......c s.brown.
0040  65 64 75 00 00 01 00 01  00 00 29 02 00 00 00 00   edu..... ..).....
0050  00 00 00                                           ...
```

3.3.3 Encapsulation within the DNS Query

If you expand any of these dissected fields, you can see the parsed data of the header plus the payload for the protocol. It is important to note how encapsulation is used here. The Ethernet frame conssits of an Ethernet header plus a payload, where the payload is an IP packet. The IP packet has an IP header and a payload, which is the UDP datagram. The UDP datagram consists of a UDP header and a payload, the DNS query. The DNS query, the last level of encapsulation, contains its header and a payload, which is the query itself. This image may help you visualize it:



3.3.4 Reading Protocol Headers

Now, let's explore these protocols. First, find the host and source Ethernet addresses. Then, for IP find the host and source IP addresses, and for UDP find the host and destination ports (you will learn what these ports mean later in the course). You can find each of these fields by expanding the respective protocol field. For each of these protocols, put your cursor over the addresses/ports and see which raw bytes these fields were extracted from in the packet. Wireshark should highlight these bytes as you hover over the address/port.
Wireshark shows more than the source and destination of a packet within a network layer; you can also see protocol related information. Expand the DNS field of the packet, and completely expand the `Queries` subfield. Notice that in `Queries` Wireshark tells you the domain being queried (`cs.brown.edu`) and the type of response it is looking for (Type being either `AAAA` meaning we are looking for an IPv6 address or `A` meaning we are looking for an IPv4 address).

3.3.5 DNS Response

If we filter for "dns" and select a DNS response packet, we can examine the query response by expanding the `Domain Name System` (DNS) field, then expanding either the `Answers` or `Authoritative Nameserver` subfield. Which subfield you find depends on which response packet you selected. Either way, the subfield should contain an entry for `cs.brown.edu`. Later in the course you will learn exactly what this response means, but for now know that we are able to pick and examine specific data chunks from packets by dissecting them.

3.4 Extract an Image from a HTTP response

As your filter use `http.content_type`, and find a packet with type PNG or JPEG. Click on one of these packets, right click on the PNG or JPEG field of the packet, click on `export packet bytes`

`...`, save the image to the VM desktop, then move the file to `/vagrant`. On your host machine in the same folder as the vagrant file for the VM you will find the image, open the image with a web browser to view what you saved.

3.5 Closing Tips

Use Wireshark! It is extremely helpful in dissecting and understanding the packets that are flying to and from a network interface. For IP you can use it to verify your nodes are sending packets and the packets have the form and fields that you expect. For example, you can use Wireshark to check that within a packet a number is sent with either big endian or little endian, which is much harder to do at the sending, intermediate, or receiving nodes (for example, you may need this functionality to check packet checksums). Wireshark will be very helpful for TCP to check that your nodes are following the TCP protocol. To examine TCP with Wireshark, there will be a separate Wireshark lab during the TCP assignment.
