UC Berkeley CS 168, Fall 2014

Protocols and Tools for Project 3

(version 1.0)

Overview

This document provides supplementary information for both Project 3a and 3b. To accomplish the project, you will need to understand the packet formats of various protocols, so that your firewall can decode packets and apply firewall rules to them. Also, you will need to use various network testing tools to generate network traffic and to verify the behavior of your firewall by tapping network interfaces (`int` and `ext`).

This document is also intended to provide some details on network protocols conceptually covered in the course lectures.

Note that this document only includes a brief introduction to the protocols and network testing tools. For more comprehensive and detailed information, you should refer to the RFC standards and (wo)man pages (yes, you can use `woman` command instead of `man` in the VM). Also note that we do not guarantee the correctness of protocol descriptions provided in this document. If the standards (the spec document includes references) conflict with this document, trust the former.

Endianness

For more details:

- http://en.wikipedia.org/wiki/Endianness
- http://docs.python.org/2/library/struct.html

When a computer stores or transmits multi-byte data, it may use one of two approaches for organizing the bytes. One is to place bytes in decreasing order of significance (i.e., MSB first). This is called "big endian". For example, the number 1234567890 is 0x499602d2 in hex. In big endian systems, the number will be stored as follows:

address	a	a + 1	a + 2	a + 3
data	0x49	0x96	0x02	0xd2

The other one is to place bytes in increasing order of significance ("little endian").

address	a	a + 1	a + 2	a + 3
data	0xd2	0x02	0x96	0x49

Most network protocols are based on big endian. On the other hand, your laptop/desktop (which likely uses the x86 architecture) uses little endian. Sometimes the terms "network order" (big endian) and "host order" (little endian, in x86) are used. Whenever you decode or encode multi-byte data from network packets, you need to convert its endianness before use. You will find the following Python functions useful.

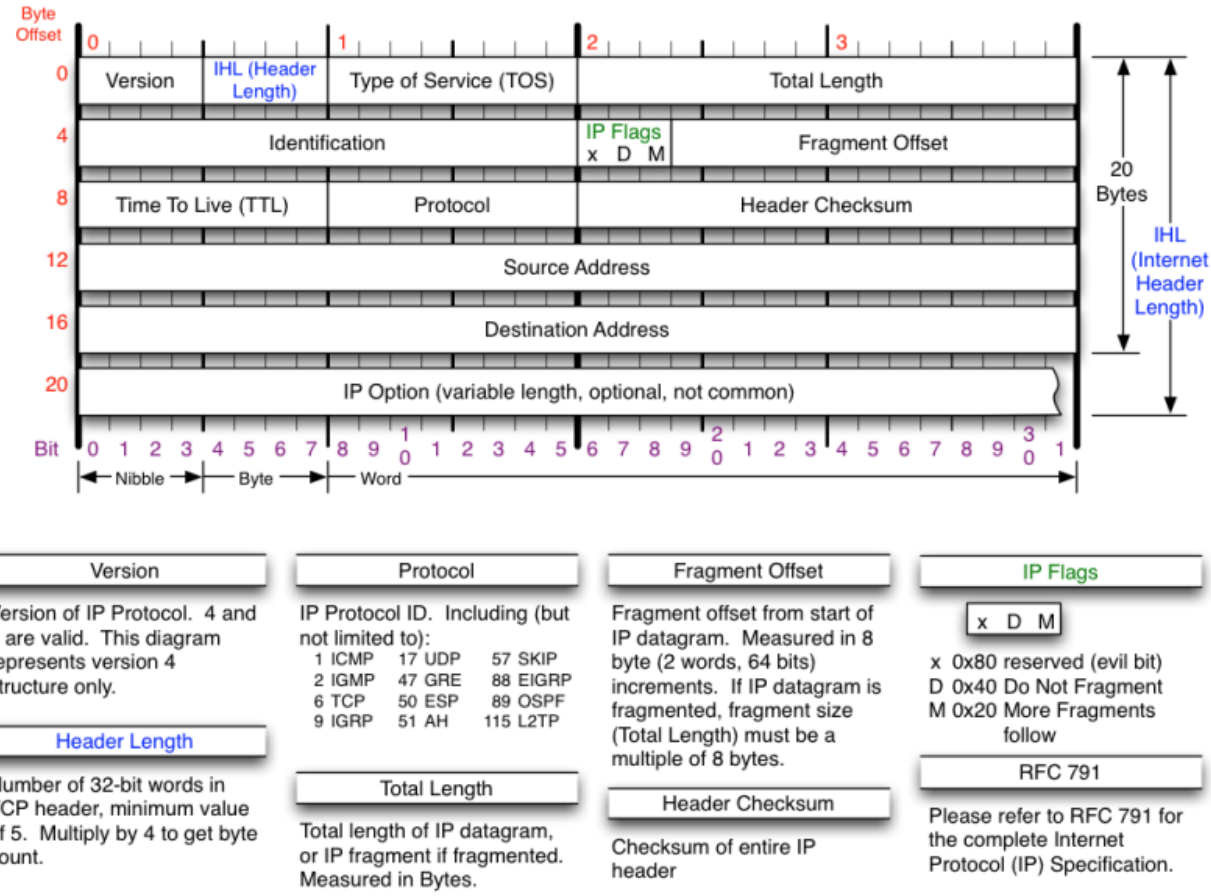
```
socket.htons(0x1234) == 0x3412          # 2B host-order integer → network-order integer
socket.ntohs(0x3412) == 0x1234          # 2B network-order integer → host-order integer
socket.htonl(0x12345678) == 0x78563412      # 4B host-order integer → network-order integer
socket.ntohl(0x78563412) == 0x12345678      # 4B network-order integer → host-order integer

chr(0x12) == '\x12'                         # 1B integer → 1B string
struct.pack('!B', 0x12) == '\x12'                   # 1B integer → 1B string
struct.pack('!H', 0x1234) == '\x12\x34'# 2B integer → 2B big-endian string
struct.pack('!L', 0x12345678) == '\x12\x34\x56\x78'   # 4B integer → 2B big-endian string

ord('\x12') == 0x12                         # 1B string → 1B integer
struct.unpack('!B', '\x12') == (0x12,)          # 1B string → 1B integer
struct.unpack('!H', '\x12\x34') == (0x1234,)    # 2B big-endian string → 2B integer
struct.unpack('!L', '\x12\x34\x56\x78') == (0x12345678,)      # 4B big-endian string → 4B integer

# You can decode multiple fields at once.
```

struct.unpack('!HH', '\x12\x34\x56\x78') == (0x1234, 0x5678)

To learn more about Python's struct module, see the link at the top of this section.

Protocols

IPv4 Header



(http://nmap.org/book/images/hdr/MJB-IP-Header-800x576.png)

In the firewall, you will only see IPv4 packets; the provided base code will hand over IPv4 packets to the Firewall class and blindly pass all non-IPv4 packets.

Header Length

The Header Length field contains the length of the IP header, divided by 4, since the length of the IP header is always a multiple of 4 bytes. The minimum IPv4 header size is 20 bytes, unless it has IP options. In general, most packets do not carry IP options, so the value of this field will usually be 5. If you see a

packet with a smaller header length than 5, you should drop the packet.

Note that a transport-layer header (TCP/UDP/ICMP) may have a variable offset in the `pkt` string in the `handle_packet()` method, depending on the length of the IP header. For example, if the value of the IP header length field is 7 and the packet is TCP, the TCP header will begin at the 28th byte of pkt.

Since this field is only 4 bits wide, you will need to use some bit operations.

Total Length

This field indicates how long the IP packet is, including the IP header itself. Since the `pkt` string has the whole data of the IP packet, the value of the total length field must be equal to `len(pkt)`. If not, you may want to drop the packet (but not required by the project spec).

Identification, Fragment Flags/Offset

These fields are for IP fragmentation. Since you will not see any fragmented packets for this project, you can ignore those fields.

TTL and Header Checksum

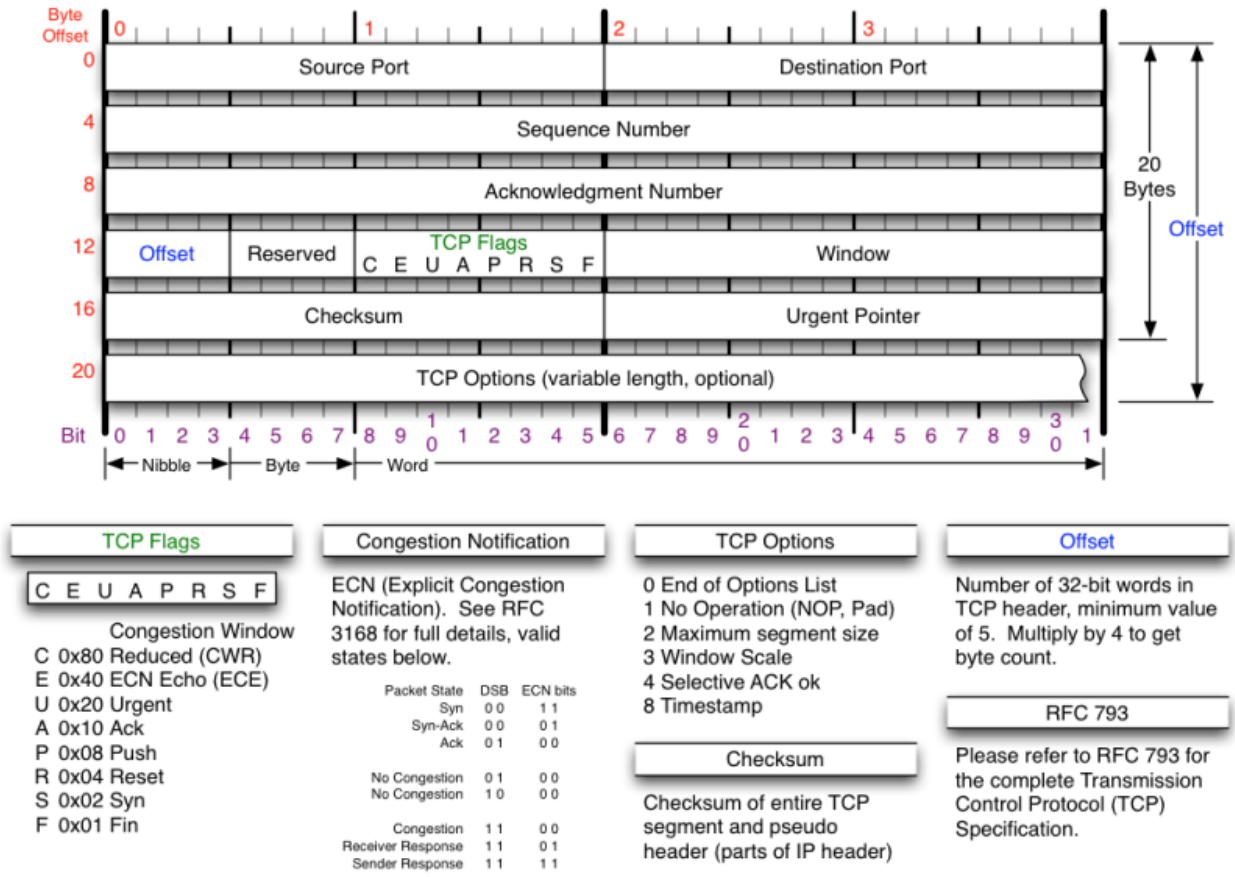
For Project 3a, you are not required check the TTL and checksum of received packets. For Project 3b, you will need to craft IPv4 packets from scratch, which means that the checksum value should be correctly calculated and filled in the field. Note that IP checksum only applies to the header bytes. For IP checksum calculation, refer to the following documents:

- http://en.wikipedia.org/wiki/IPv4_header_checksum
- http://www.thegeekstuff.com/2012/05/ip-header-checksum/

Source / Destination Addresses

IPv4 addresses are represented as a 4-byte binary data. For example, 123.45.67.89 is stored as '\x7b\x2d\x43\x59' in hex. To convert an IPv4 address string into a 4-byte binary data, you can use `socket.inet_aton()`. To convert a 4-byte binary data into an IPv4 address string, use `socket.inet_ntoa()`. Refer to the `bypass.py` file for an example.

TCP Header



(http://nmap.org/book/images/hdr/MJB-TCP-Header-800x564.png)

Source / Destination Ports

For Project 3a, you only need to consider these two fields.

Your firewall should examine “external” ports. For incoming packets (from the outside network to the VM), the source port field contains the external port. For outgoing packets (from the VM to the outside network), the destination port field contains the external port. Do not ignore endianness, since these are 2-byte fields.

Sequence / Acknowledgement Number

As discussed in the lecture, each TCP packet carries its sequence number (in bytes, not in packets). Recall that TCP is a full-duplex protocol. For each direction of a connection, a separate sequence number is used. The Sequence Number field contains the sequence number of the first byte of the TCP segment data.

When the ACK flag is set (it is usually set all the time, except for the very first SYN packet of TCP handshake), the Acknowledgement Number field contains the cumulative ack sequence number. For example, if the ack sequence number is X, it means the receiver successfully received up to X-1'th byte and expects sequence number X for the next data.

Packets with a SYN or FIN flag increases the sequence number by 1. Look at the following example (suppose that the initial sequence numbers are 1000 and 2000).

SYN, seq=1000, no data →
← SYN+ACK, seq=2000, ack=1001, no data
ACK, seq=1001, ack=2001, no data →
ACK, seq=1001, ack=2001, data="hello" →
← ACK, seq=2001, ack=1006, data="world!!"
ACK, seq=1006, ack=2008, no data →
FIN, ACK, seq=1006, ack=2008, no data →
← FIN, ACK, seq=2008, ack=1007, no data
ACK, seq=1007, ack=2009, no data →

For more details about sequence and acknowledgement numbers, read this article:
http://packetlife.net/blog/2010/jun/7/understanding-tcp-sequence-acknowledgment-numbers/

Offset

This is very similar to the Header Length field in the IPv4 header. It specifies the length of the TCP header in bytes, divided by 4. Since the minimum TCP header length is 20 bytes, the value should not be less than 5 (20B). It is the offset of the TCP payload, beginning from the TCP header.

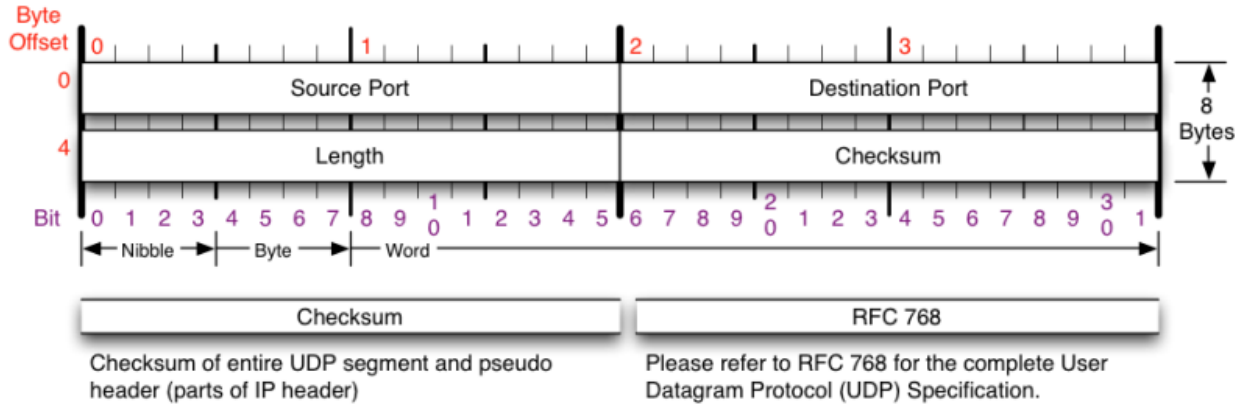
Checksum

For Project 3a, you are not required to check the checksum value of packets. For Project 3b, you need to understand how to calculate the TCP checksum.

Unlike IPv4 checksum, which only covers the IP header, TCP checksum calculation is more complex. It is calculated with a TCP "pseudo header" and the payload data.

Wikipedia has a detailed description of the pseudo header.
http://en.wikipedia.org/wiki/Transmission_Control_Protocol#TCP_checksum_for_IPv4
http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader.htm

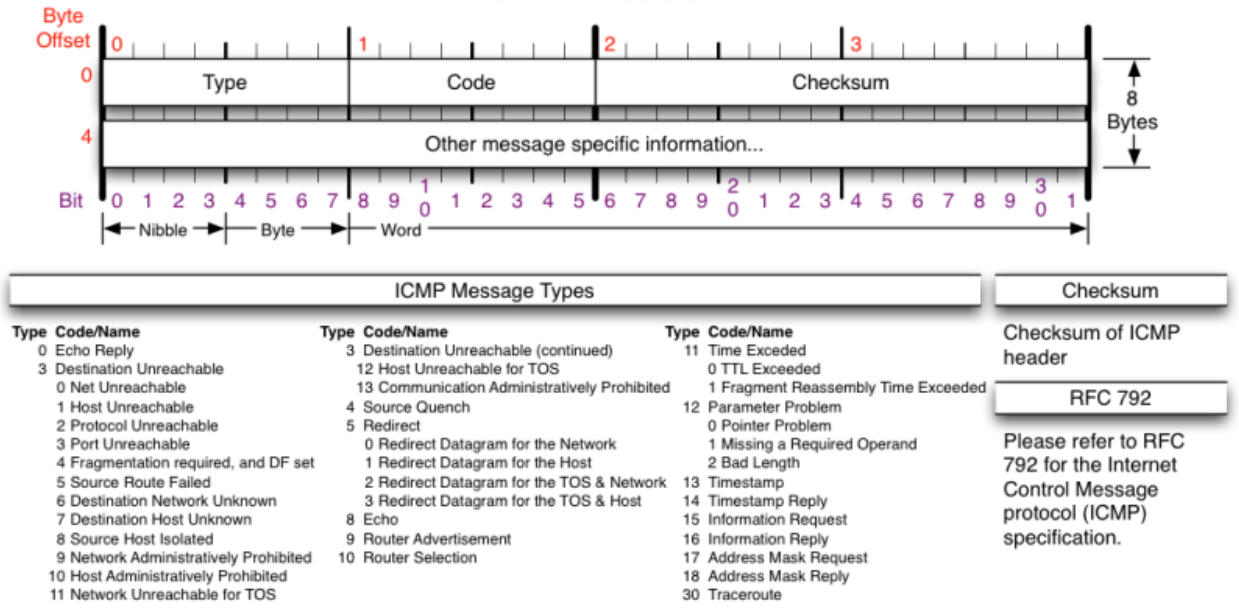
UDP Header



http://nmap.org/book/images/hdr/MJB-UDP-Header-800x264.png

UDP header is simpler than TCP, since it is designed as a basic wrapper for raw IP packets. The offsets of Source Port and Destination Port are the same as in TCP.

ICMP Header



http://nmap.org/book/images/hdr/MJB-ICMP-Header-800x392.png

The IP protocol defines the data plane of the Internet; IP packets carry data among end hosts and routers.

ICMP is a swiss-army-knife protocol that is specially designed to supplement the functionality of IP (e.g., diagnostica and error reporting).

Like TCP and UDP, ICMP is implemented on top of IP, thus the ICMP header begins at the end of the IPv4 header. The format of a ICMP packet greatly varies according to its type. For Project 3a, you will need to only examine the 1-byte “Type” field.

DNS Packets

DNS can be implemented on both TCP and UDP, but most implementations primarily use UDP. For the project, we only consider UDP-based DNS packets with destination port 53. All communications inside of the domain name system protocol are carried in a single format called a message. The top level format of message is divided into 5 sections (some of which are empty in certain cases) shown below:

```
+---------------------+
|        Header       |
+---------------------+
|       Question      |    the question for the name server
+---------------------+
|        Answer       |    RRs answering the question
+---------------------+
|      Authority      |    RRs pointing toward an authority
+---------------------+
|      Additional     |    RRs holding additional information
+---------------------+
```

RR stands for “resource record”. In this project, we only care about RR records with A (IPv4) or AAAA (IPv6) type. While the VM is configured to disable IPv6, the DNS resolver library may still generates AAAA-type queries if an A-type query fails.

Header

The header contains the following fields:

```
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|  Opcode   |AA|TC|RD|RA|  zero  |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Long story short, this is 16 bits (two bytes) by 6 rows, for a total of 12 bytes. For the project, you should examine the QDCOUNT field, which specifies the number of question entries in the QUESTION section. The spec documents states that we only consider DNS messages with QDCOUNT == 1.

QUESTION Section Format

The question section is used to carry the "question" in most queries, i.e., the parameters that define what is being asked. The section contains QDCOUNT (usually 1) entries, each of the following format:

```
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                               |
/                     QNAME                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     QTYPE                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QCLASS                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

- QNAME: a domain name represented as a sequence of labels, where each label consists of a length byte followed by that number of bytes. The domain name terminates with the zero length byte. Note that this field may be an odd number of bytes; no padding is used, so the following two fields may not be 16-bit aligned.
- QTYPE: a two-byte code which species the type of the query. The values for this eld include all codes valid for a TYPE field, together with some more general codes which can match more than one type of RR.
- QCLASS; a two-byte code that species the class of the query. For example, the QCLASS field is IN (1) for the Internet.

An example of what QNAME will look like:

```
03 77 77 77 06 67 6f 6f 67 6c 65 03 63 6f 6d 00
    w  w  w     g  o  o  g  l  e     c  o  m
```

You will primarily be interested in A records, which map hostname to IPv4 address (QTYPE == 1), and AAAA records, which map hostname to IPv6 address (QTYPE == 28).

Network Testing Tools

tcpdump / Wireshark

As you go about developing your firewall you might find it useful to observe the packets arriving at the network interface. Amongst other things observing packet data is helpful for debugging (you can try and determine properties of packets not being processed correctly), making sure that your firewall is actually being tested and just determining the kinds of packets generated by a variety of applications.

Packet sniffers are commonly used to accomplish these tasks, your VM has two of these installed: `Wireshark` and `tcpdump`. `Wireshark` is graphical, while `tcpdump` is a command line tool. Both are capable of filtering packets and are almost equally powerful. We briefly describe both below and point to a few sources of information online. We strongly encourage you to look through tutorial and other documentation.

Normally The both tools require root privileges to run, since packets contain security-critical information. However, in the provided VM, you don't need to do "`sudo`" every time to run them (we did something special for you).

When you run tcpdump/Wireshark on your own machine, you will see a lot of packets, since there are many background applications that connect to the Internet. In the VM, we disabled most such background applications, so you will see much fewer "noise" packets.

tcpdump

`tcpdump` is a command line packet sniffer, which prints out a description of packets going through a network interface. By default `tcpdump`'s description of a packet is dependent upon the protocol, for TCP packets it will print a description like: "src > dst: flags data-seqno ack window urgent options". As an example, consider the following output:

```
$ tcpdump -n -i int -vv port 53
tcpdump: listening on int, link-type EN10MB (Ethernet), capture size 65535 bytes
17:32:33.567292 IP (tos 0x0, ttl 64, id 31258, offset 0, flags [none], proto UDP (17),
length 73)
    10.0.2.15.49827 > 8.8.8.8.53: [udp sum ok] 26425+ [1au] A? www.berkeley.edu. ar: .
OPT UDPsize=4096 (45)
17:32:33.637730 IP (tos 0x0, ttl 64, id 8, offset 0, flags [none], proto UDP (17), length
110)
    8.8.8.8.53 > 10.0.2.15.49827: [udp sum ok] 26425$ q: A? www.berkeley.edu. 2/0/1
www.berkeley.edu. CNAME www.w3.berkeley.edu., www.w3.berkeley.edu. A 169.229.216.200 ar:
. OPT UDPsize=512 (82)
```

What does the "-n -i int -vv port 53" mean?

- `-n:` Normally tcpdump will try to convert numeric addresses into human-friendly strings (e.g., IP address 8.8.8.8 → google-public-dns-a.google.com, port number 53 → DNS, etc.). The "`-n`" option prevents this behavior.

- `-i int`: specifies the interface to monitor.
    - For this project, you are interested in `int` and `ext`.
- `-vv`: specifies the entire payload should be decoded.
- `port 53`: specifies an optional filter, in this case stating that we only want to capture packets with TCP or UDP source or destination port 53.
    - For more examples of filter expressions, try "`man pcap-filter`" in the VM or refer to this: http://wiki.wireshark.org/CaptureFilters

While such interpreted records can help determine the kind of packets being sent, it is often useful to just see raw packet data. This can be accomplished using the '`-X`' flag which prints raw bytes in hex and ascii, side-by-side, for instance:

```
$ tcpdump -n -i int -X
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on int, link-type EN10MB (Ethernet), capture size 65535 bytes
17:40:14.501879 IP 10.0.2.15.46031 > 8.8.8.8.53: 17322+ [1au] A? www.berkeley.edu. (45)
       0x0000:  4500 0049 7a1d 0000 4011 e468 0a00 020f  E..Iz...@..h....
       0x0010:  0808 0808 b3cf 0035 0035 ad8c 43aa 0120  .......5.5..C...
       0x0020:  0001 0000 0000 0001 0377 7777 0862 6572  .........www.ber
       0x0030:  6b65 6c65 7903 6564 7500 0001 0001 0000  keley.edu.......
       0x0040:  2910 0000 0000 0000 00                   ).........
```

`tcpdump` has several other useful options. We recommend reading through the man pages ("`man tcpdump`" in the VM) or looking at

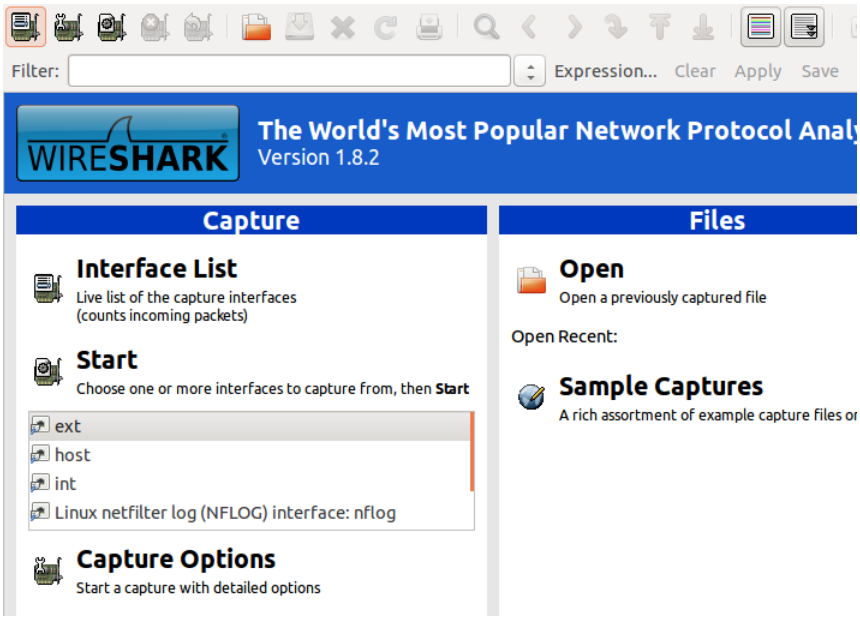
- http://www.thegeekstuff.com/2010/08/tcpdump-command-examples/
- http://www.danielmiessler.com/study/tcpdump/
- and Googling around for more information.

Wireshark

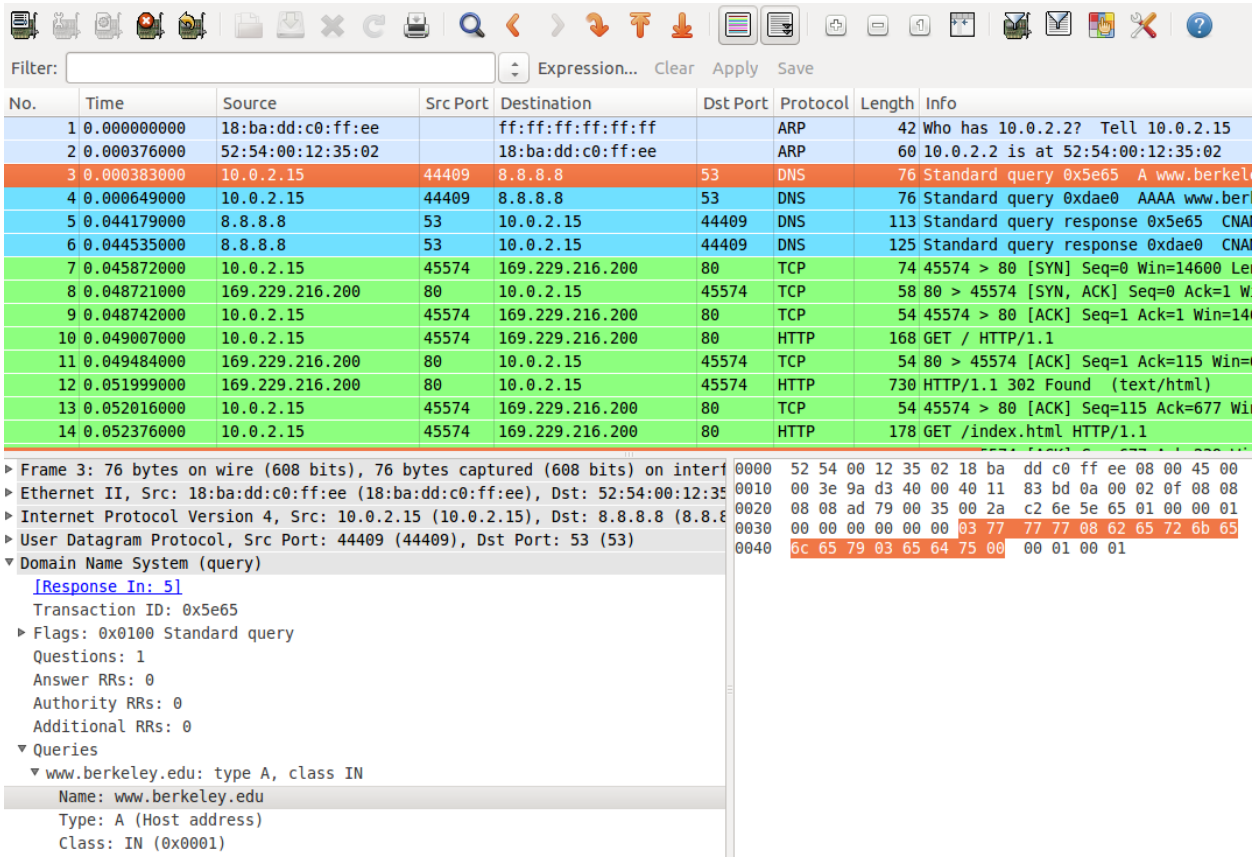
`Wireshark` provides a graphical interface for capturing packets similar to what is allowed by `tcpdump`. You can start `Wireshark` by running "`wireshark &`" from the command line, or by clicking the following icon below the desktop screen:

Tip: You may want to launch two instances of Wireshark, to monitor both int and ext interfaces at the same time. This can be useful to verify your firewall's behavior.

Once started, the window will look like this:



Choose a network interface (`ext` or `int` on the list box), and click "Start". Wireshark will start capturing packets on the interface and display them:

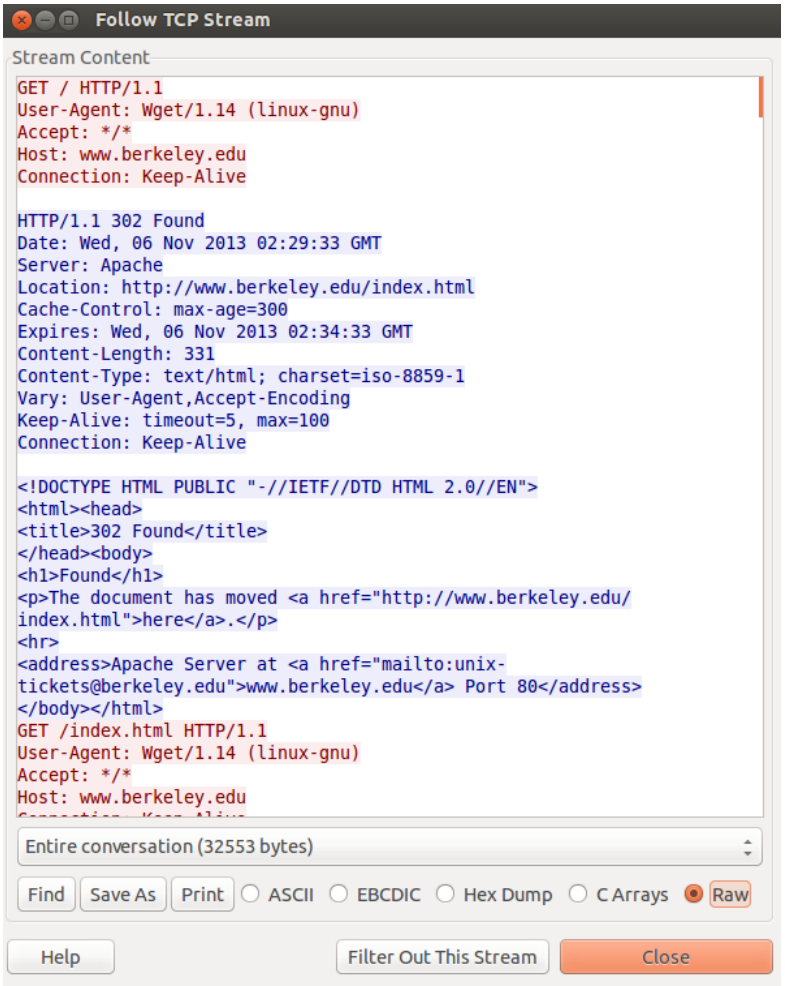


The screen shows three panels. The top one is the list of captured packets. If you are seeing too many

packets, you can apply a display filter to show interested packets only. The syntax of Wireshark display filter is different from that of tcpdump. Refer to this link: http://wiki.wireshark.org/DisplayFilters

Click on a packet you want to inspect. On the bottom left panel, you will see the detailed information of the decoded packet, for each network layer (In the above example, Ethernet, IP, UDP, and DNS). When you click on one of the entries in the panel, the bottom right panel will show how the selected part of the packet corresponds to the packet binary data. In the above screenshot, it represents how "www.berkeley.edu" is represented in the QNAME field.

One of the most useful feature of `Wireshark` is that it can reconstruct a whole TCP stream from individual TCP packets. You will heavily rely on this feature for Project 3b. Right-click on a TCP packet, and choose "Follow TCP Stream". The display filter will be automatically set for the TCP connection, and the following popup window will appear. The example below is from the packets with "`wget www.berkeley.edu`"



`tcpdump` and `Wireshark` are some of the ultimate tools that every programmer should know how to use. There are many video tutorials on YouTube. Try them out!

nslookup / dig

Both nslookup and dig performs DNS lookups, so you can utilize them to generate DNS query packets for testing DNS rule matching. nslookup is deprecated, but it is still widely used. Here, we briefly introduce how to use dig.

```
dig [@server] [options] query
```

You can optionally specify a DNS resolver (e.g, `@8.8.4.4`). If unspecified, the default DNS server of the system (8.8.8.8 in the provided VM) will be used. The destination IP address of the DNS packet will be that of the DNS resolver. The "query" is the name of the resource record to be looked up. Since both `nslookup` and `dig` generate a DNS query for `A` records, the query should be a domain name.

The following example shows the result of "`dig @75.75.75.75 www.berkeley.edu`". 75.75.75.75 is a DNS server operated by Comcast. What it shows is basically the decoded information of the DNS response packet.

```
; <<>> DiG 9.8.3-P1 <<>> @75.75.75.75 www.berkeley.edu
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58999
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.berkeley.edu.         IN     A

;; ANSWER SECTION:
www.berkeley.edu.    154    IN     CNAME  www.w3.berkeley.edu.
www.w3.berkeley.edu.145    IN     A      169.229.216.200
```

From the result, you see that `www.berkeley.edu` is just an alias of `www.w3.berkeley.edu` (the `CNAME` record), and its IP address is `169.229.216.200` (the `A` record).

`dig` supports a variety of options. You may find the following two options useful.

- `-t AAAA`: Ask for an `AAAA` (IPv6 address) record, instead of `A`.
- `+trace`: Make iterative queries, instead of recursive queries.

wget / curl

`wget` is used for non-interactive downloading of files via HTTP or FTP. The most basic way of using wget is just "`wget http://foo.com/bar/baz`" which will just download the file to the current directory. A nice feature is that a download bar will be shown to portray the progress, as well as speed and predicted time until completion. `wget` is very useful for Bells and Whistles 2 and 3 of Project 3a.

For Project 3b, `wget` can be a good alternative to Firefox for generating HTTP test traffic, because of its streamlined behavior (e.g., you don't need to empty the local cache of Firefox every time).

Various options you can put on the command line are:

- `-O [output file]`: This allows specifying the output location. Note that this is not `-o` (lowercase), which is writing debug messages to a log file (and likely not what you want to do).
- `-p`: Download recursively. This allows downloading an entire page (e.g., the HTML file and its embedded images) instead of just a single file.
- `-nd`: Do not create directory hierarchies when downloading recursively.
- `-nc`: Do not clobber. This is to preserve any previous instances of the same file. A new copy, in that case, will be named filename.N, where N is the Nth copy of the same file.
- `-c`: Continue. This is to continue downloading a partially downloaded file.

`curl` is a similar to `wget` in its purpose, but it supports more various protocols. If you are a Mac user, you may be more familiar with curl than wget, as it is installed by default. For differences between `curl` and `wget`, read this article: http://daniel.haxx.se/docs/curl-vs-wget.html

nc

`nc` (short for netcat) is a command-line tool that can be used for various socket operations. You will find this tool very useful to generate TCP/UDP packets for Project 3a; "It can open TCP connections, send UDP packets, listen on arbitrary TCP and UDP ports, [...]" [manpage].

By default nc will use TCP as its transport protocol. The basic usage is "`nc` [destination] [port]", where the destination can be either an IP address or a domain name. `nc` will initiate a TCP connection to the specified destination/port, which triggers TCP 3-way handshake. Once connected, what you type (via standard input) will be transferred to the destination via TCP packets, and the response will be displayed on the screen (via standard output). You can specify the -u flag to use UDP, instead of TCP.

Do not play with the port scanning function of `nc`. You may get into trouble for doing this (network abuse).