Packet Reading

Prepared by:

William Zereneh
zereneh@scs.ryerson.ca
January 2010

Packet Reading

Manually Inspecting Packet Fields for Analysis!
Why?

“There will be times that you WILL have to manually evaluate the content of packets to identify what they are doing”

Binary Conversion

- To calculate a number, you take the base of the numbering system and raise it to the power of the column.
  - First Column is 0, next is 1, etc.
  - 256 base 10
    - $(2x10^2)+(5x10^1)+(6x10^0)=256$
  - 1001 base 2
    - $(1x2^3)+(0x2^2)+(0x2^1)+(1x2^0)=9$ base 10
  - 11111001 base 2 equals F9 base 16

Hexadecimal Representation

2^3	2^2	2^1	2^0					2^3	2^2	2^1	2^0			Hex
0	0	0	0	=	0			1	0	0	0	=	8	
0	0	0	1	=	1			1	0	0	1	=	9	
0	0	1	0	=	2			1	0	1	0	=	10	(a)
0	0	1	1	=	3			1	0	1	1	=	11	(b)
0	1	0	0	=	4			1	1	0	0	=	12	(c)
0	1	0	1	=	5			1	1	0	1	=	13	(d)
0	1	1	0	=	6			1	1	1	0	=	14	(e)
0	1	1	1	=	7			1	1	1	1	=	15	(f)

Five Tips for Decoding Packets

- Offset from beginning of the packet start at 0
- Four bits = 1 hex character
- Fields can be any length
  - From one bit to many bytes
  - Fixed length, or variable length
- Fields in one protocol identify the length and contents of others
- Computers use header references too!

Calculating Variable Length Fields

- TCP options and length of payload are variable width fields
- We calculate field length using other header information and fixed lengths

TCP Options Length = (TCP header Length – Min. TCP Header Length)

Length of Packet Payload = IP Total Length – (IP Header Length + TCP Header Length)

Packet Reading - tcpdump

What is tcpdump?

- Simply dumps traffic on a network
- Was officially maintained by the Lawrence Berkeley Lab
- Now maintained through a collective effort
- Command-line tool for monitoring (sniffing) network traffic
- Universally available and used on many platforms
- Free – Open Source Software/GPL

Getting Started with tcpdump

The Good:

- Universally available and used, even on Windows
- Provides audit trail of network
- Non-assuming, you do your own interpretation

The Bad:

- Only collects 68 bytes of data from network with default settings
- Does not keep track of traffic flow (state)
- Does not scale well on large networks

Default output

To run tcpdump:

#> tcpdump

Sample Output:

13:16:37.348336 IP 172.16.210.131 > od-in-f104.google.com: ICMP echo request, id 55057, seq 1, length 64

tcpdump will listen on first interface (eth0) and dump packets to console

Output in Hexadecimal

To run tcpdump:

#> tcpdump -x

Output:

13:28:05.783671 IP 172.16.210.131 > qy-in-f104.google.com: ICMP echo request, id 4626, seq 1, length 64

```
0x0000:  4500 0054 0000 4000 4001 1630 ac10 d283
0x0010:  4a7d 5b68 0800 db92 1212 0001 2518 c949
0x0020:  25f5 0b00 0809 0a0b 0c0d 0e0f 1011 1213
0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
0x0050:  3435
```

IP Header

ICMP Header

tcpdump – Default snaplen

- The default snapshot length is 68 bytes, meaning only 68 bytes will be captured
- Usually enough to capture IP header, embedded protocol header and some data.
- Then why do we only see **54** bytes of data from tcpdump?
- Frame header: 14 bytes + 4 byte for CRC that are not captured by tcpdump
- Increase snaplen by using tcpdump option **–s**

  Example: tcpdump –s 0

  -s zero will capture entire Ethernet header and IP packet

Frame Header

13:47:03.474681 00:50:56:f4:44:da > 00:0c:29:98:5d:11, ethertype IPv4 (0x0800), **length 98**: 74.125.91.104 > 172.16.210.131: ICMP echo reply, id 2095, seq 1, length 64

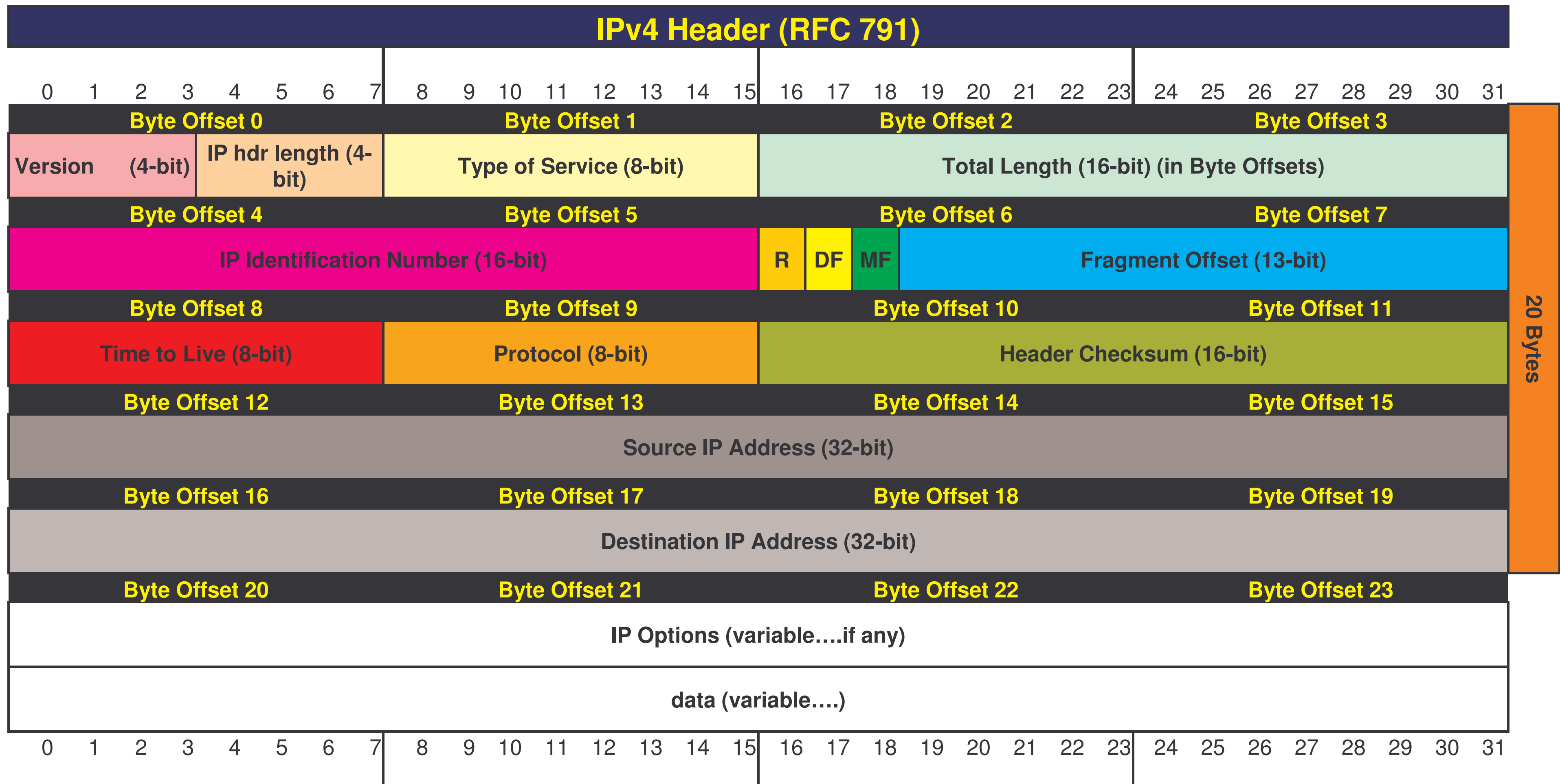
```
0x0000:  000c 2998 5d11 0050 56f4 44da 0800 4500
0x0010:  0054 2652 0000 8001 efdd 4a7d 5b68 ac10
0x0020:  d283 0000 4778 082f 0001 176e ca49 dd9c
0x0030:  0600 0809 0a0b 0c0d 0e0f 1011 1213 1415
0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425
0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435
```

Frame Header = 14 bytes

IP Header = 20 bytes

ICMP Header = 8 bytes

IP Header Length Fields



4-bit IP Header length multiply by 4 to convert to bytes
16-bit IP datagram total length
13-bit Fragment offset length multiply by 8 to convert to bytes

Decoding IP header (1)

Byte 0
Version 4, IHL 5

Byte 1
ToS 0

```
0x0000:  4500 00f4 4a99 0000 8011 f2b8 ac10 d202
0x0010:  ac10 d283 0035 aaf4 00e0 7d0c 7a6a 8180
0x0020:  0001 0001 0004 0004 0331 3034 0331 3631
0x0030:  0332 3333 0236 3407 696e 2d61 6464 7204
0x0040:  6172 7061 0000 0c00 01c0 0c00 0c00 0100
0x0050:  0000
```

Byte 0 contains both Version and Initial Header Length.
IP Version Upper Nibble byte 0: 4 – IPv4
IHL Lower Nibble byte 0: 5 32-bite words
ToS Byte 1: 0 not set

IP Header length

0x0000: 4500 00f4 4a99 0000 8011 f2b8 ac10 d202
0x0010: ac10 d283 0035 aaf4 00e0 7d0c 7a6a 8180
0x0020: 0001 0001 0004 0004 0331 3034 0331 3631
0x0030: 0332 3333 0236 3407 696e 2d61 6464 7204
0x0040: 6172 7061 0000 0c00 01c0 0c00 0c00 0100
0x0050: 0000

IP Header Length = 5 or 5 32-bit words
5*4 bytes = 20 bytes in length

Note: Minimum IP Header length is 20 bytes

Are there any IP options?

IP Options

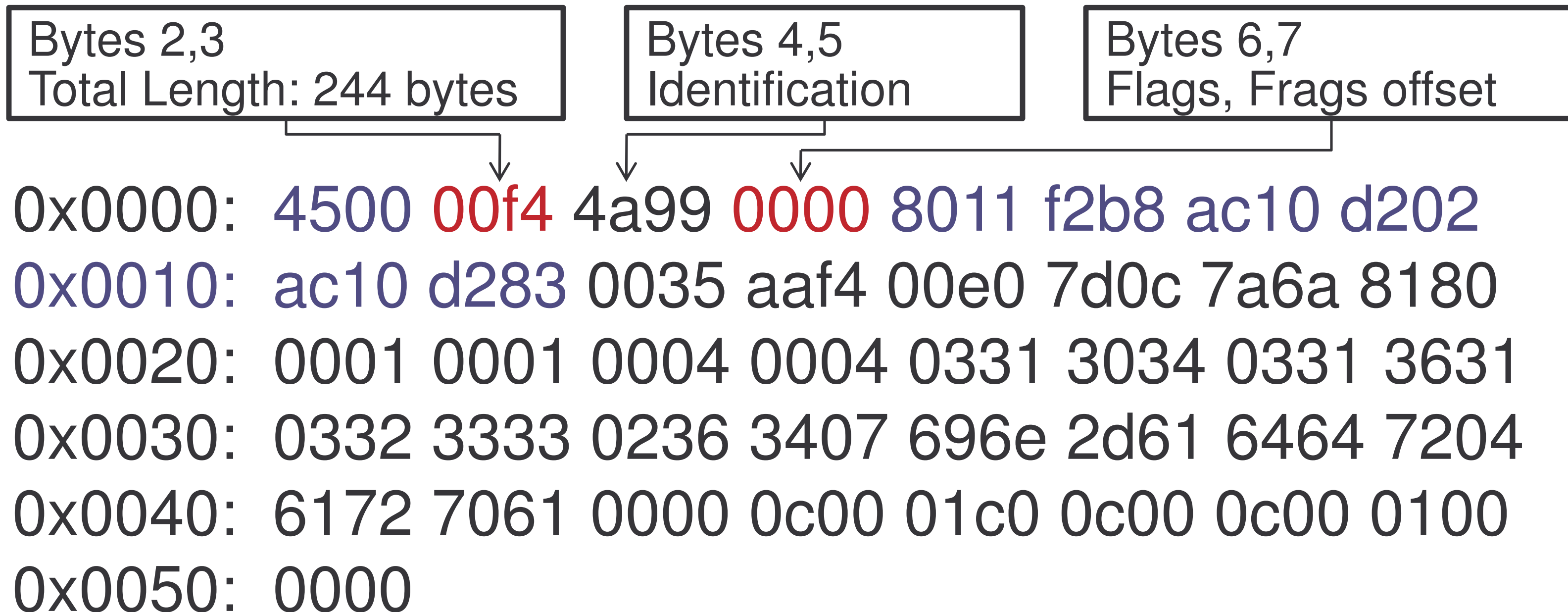
0x0000: 4700 0030 7276 0000 4006 74c9 7f00 0001
0x0010: 7f00 0001 8907 087f 0000 0200 091d 0000
0x0020: 0052 ea5d 1176 204e 5000 0200 8a51 0000

IP Header Length = 7 or 7 32-bit words
7*4 bytes = 28 bytes in length

Type:8bits = 0x89 = 137 decimal
Length:8bits = 07 bytes
Pointer:8bits = 08
Route data: variable = 127.0.0.2

Note: Strict source route - RFC 791

Decoding IP header (2)



Byte 2,3 Total Length: 0xf4 = 244 bytes
Bytes 4,5 IPID: 4a99
Bytes 6,7 IP Flags 3 bits and 13-bit Fragments offset. An example will be shown in slide Fragmentation Total Length

IP Datagram Length

0x0000: 4700 0030 7276 0000 4006 74c9 7f00 0001
0x0010: 7f00 0001 8907 087f 0000 0200 091d 0000
0x0020: 0052 ea5d 1176 204e 5000 0200 8a51 0000

IP Datagram Length = 30 = 48 bytes in length

Note: 2 bytes frame check sum missing from dump

Fragmentation Total Length

0x0000: 4500 0024 0043 2000 4006 5c8f 7f00 0001
0x0010: 7f00 0001 0b4c 0000 6e4a 9e07 0279 5755
0x0020: 5000 0200

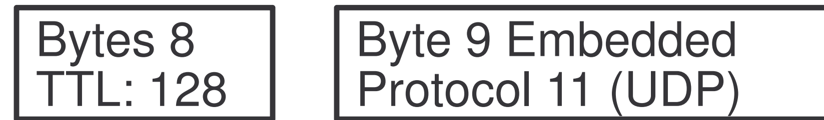
0x0000: 4500 0022 0043 0002 4006 7c8f 7f00 0001
0x0010: 7f00 0001 5554 0000 4672 6167 6d65 6e74
0x0020: 6564

Bytes 6, 7 offset zero of ip header contains:
3 bits flags field - 0x2 = 0010 binary – first bit is set meaning “More Fragments”

13 Fragment offset – zero offset meaning first fragment

Fragment offset length 2^{13} = 8192 bytes
All possible datagram length 2^{16} = 65,536 bytes
65,536 / 8192 = 8
Therefore, fragment offset must be multiplied by 8

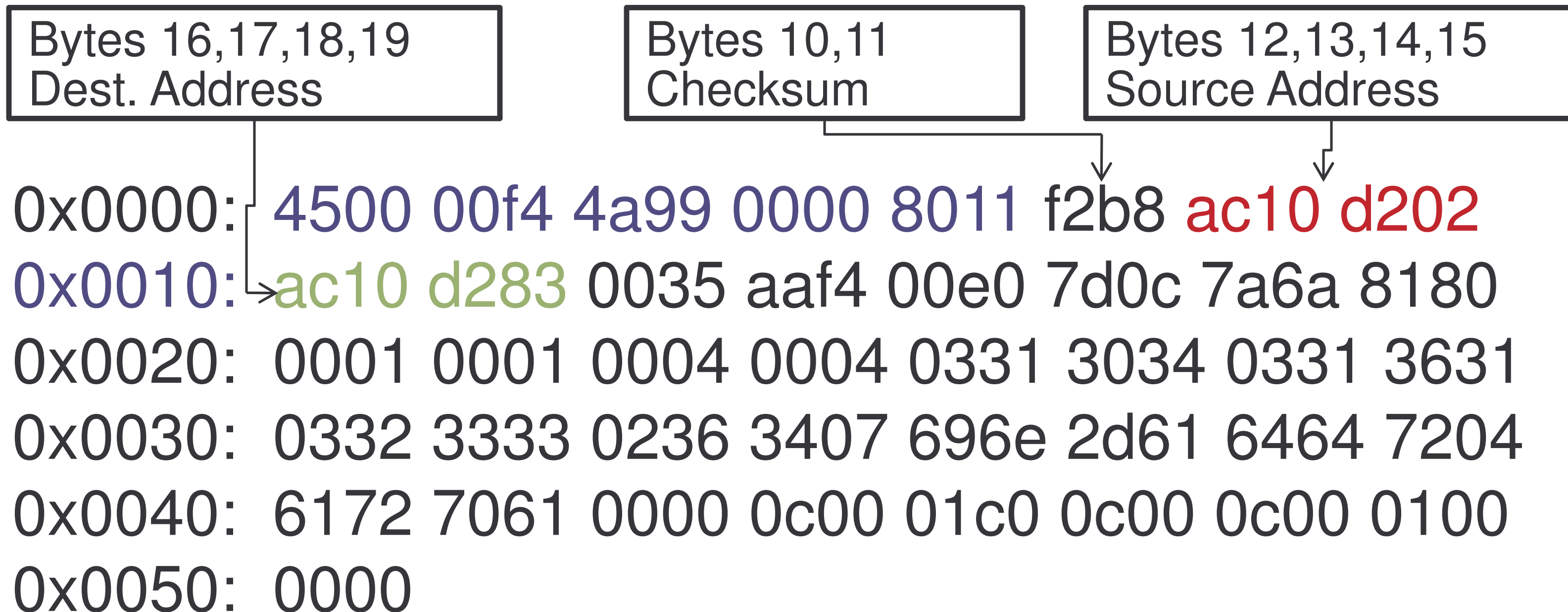
Decoding IP header (3)



0x0000: 4500 00f4 4a99 0000 8011 f2b8 ac10 d202
0x0010: ac10 d283 0035 aaf4 00e0 7d0c 7a6a 8180
0x0020: 0001 0001 0004 0004 0331 3034 0331 3631
0x0030: 0332 3333 0236 3407 696e 2d61 6464 7204
0x0040: 6172 7061 0000 0c00 01c0 0c00 0c00 0100
0x0050: 0000

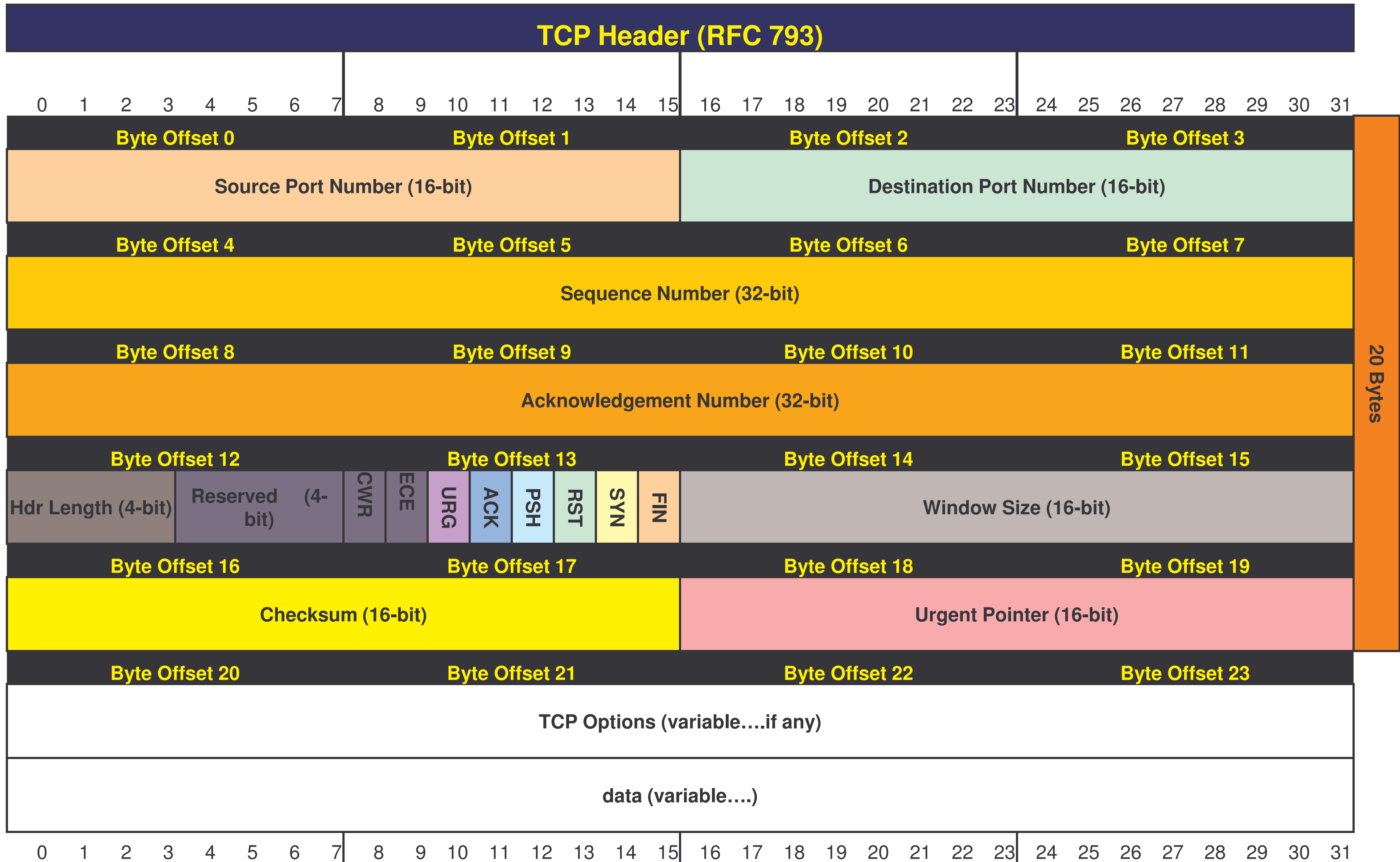
Byte 8 Time To Live: 0x80 = 128
Bytes 9 Embedded Protocol: 0x11 = 17 decimal UDP

Decoding IP header (4)

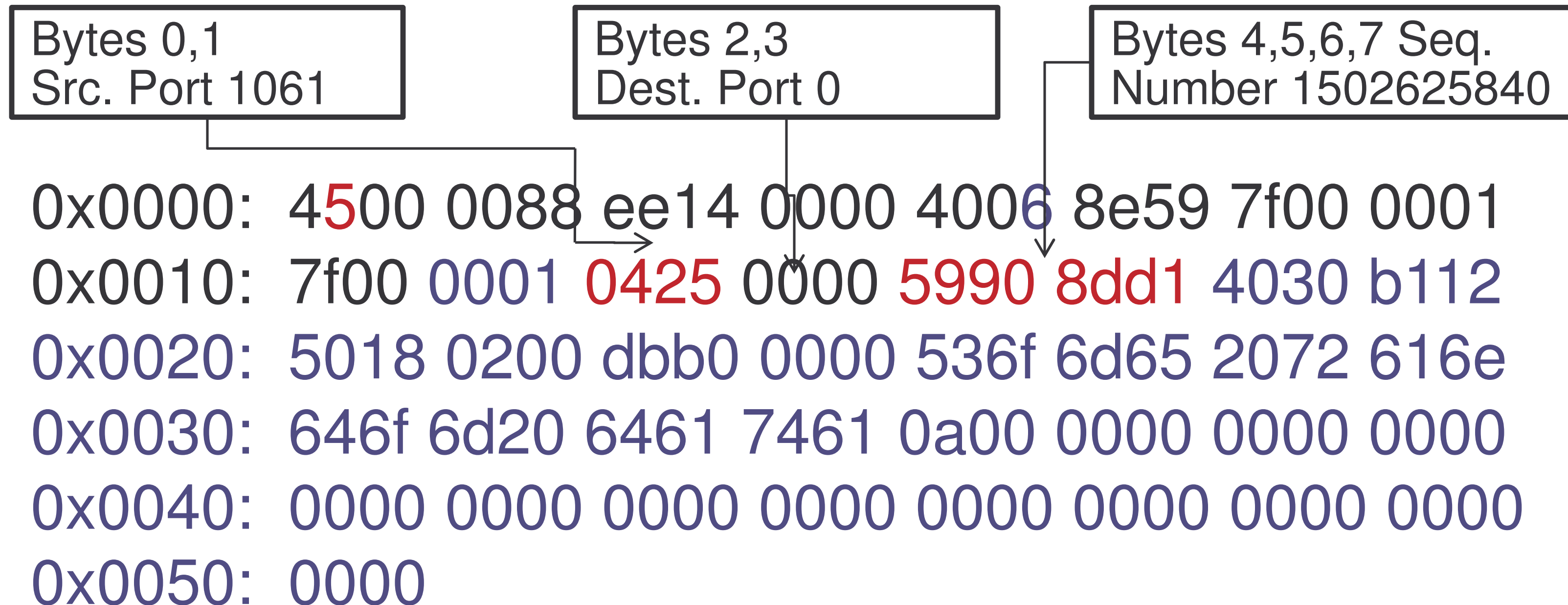


Byte 10,11 Checksum: f2b8
Bytes 12,13,14,15 Source address = ac10 d202 – 172.16.210.2
Bytes 16,17,18,19 Dest. address = ac10 d283 – 172.16.210.131

TCP Header



Decoding TCP header (1)



Bytes 0,1 Source Port: 0425 - 1061
Bytes 2,3 Destination Port: 0000
Byte offset 4, length 4 bytes Sequence number: 5990 8dd1 - 1502625840

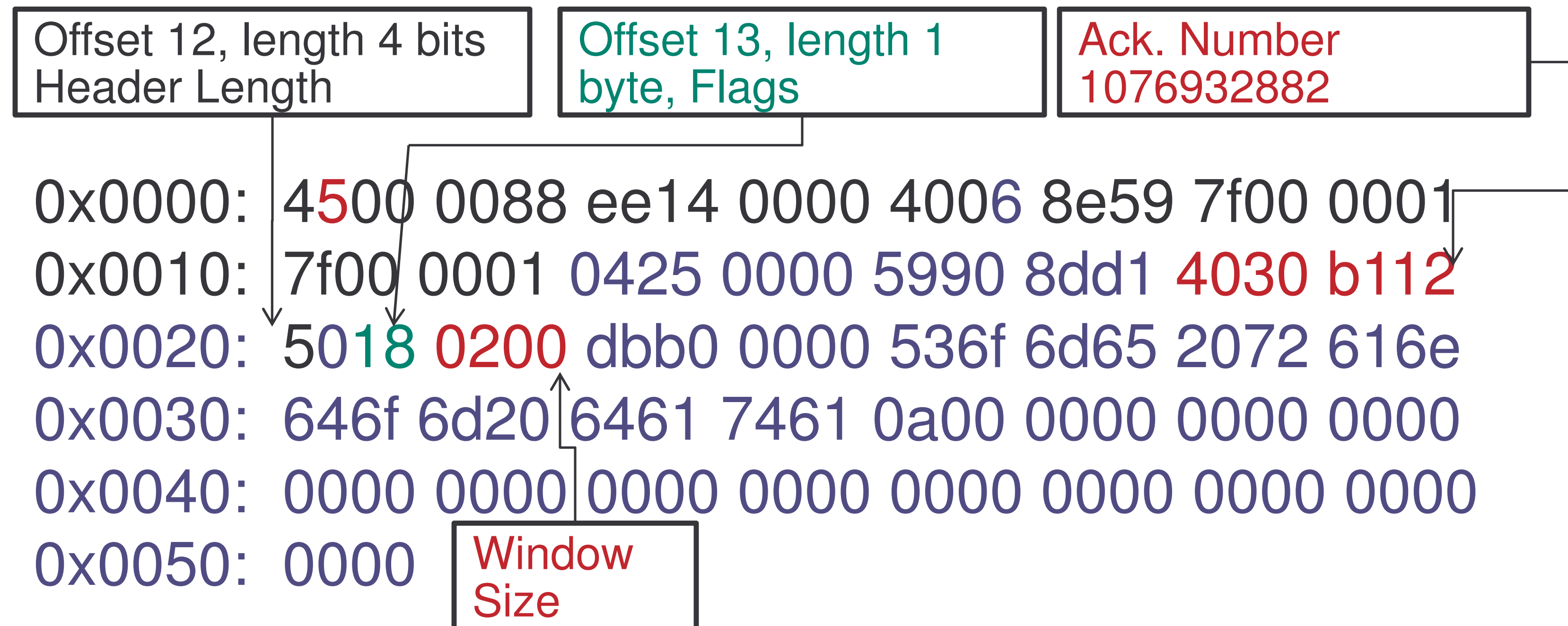
TCP Header Length

```
0x0000:  4500 0088 ee14 0000 4006 8e59 7f00 0001
0x0010:  7f00 0001 0425 0000 5990 8dd1 4030 b112
0x0020:  5000 0200 dbb0 0000 536f 6d65 2072 616e
0x0030:  646f 6d20 6461 7461 0a00 0000 0000 0000
0x0040:  0000 0000 0000 0000 0000 0000 0000 0000
0x0050:  0000
```

The TCP header length is found in the high-order nibble of the 12th byte offset in the TCP Header expressed as 32-bit word and must be multiplied by 4 to convert to bytes.

Needed to calculate where the TCP header stops, and where data starts.

Decoding TCP header (2)



Byte offset 8, length 4 bytes Ack. Number: 4030 b112 - 1076932882
Byte offset 12, length 4 bits Header Length: 5 32-bit words
Byte offset 13, length 1 byte TCP Flags: 18 – 0001 1000 = PSH, ACK
Byte offset 14, length 2 byte Windows Size: 0200 - 512

TCP Header Length with options

```
0x0000:  4500 0094 95f7 0000 4006 e66a 7f00 0001
0x0010:  7f00 0001 0a72 0000 5258 3d2d 2e30 da24
0x0020:  8000 0200 5000 0000 0101 080a ef2e feeb
0x0030:  0000 0000 5443 5020 7061 636b 6574 2077
0x0040:  6974 6820 6f70 7469 6f6e 730a 0000 0000
0x0050:  0000
```

TCP Header length = 8 * 4 = 32
Minimum TCP header length = 20
Options length = 32 – 20 = 12 starting from 20th TCP byte offset
01 01 nop nop
08 – Timestamp followed by timestamp - 080a ef2e feeb
0000 0000 – Padding

Decoding TCP header (3)

Offset 16, length 2 bytes Checksum

Offset 18, length 2 bytes, Urgent Pointer

Offset 20, length 96 bytes Data

```
0x0000:  4500 0088 ee14 0000 4006 8e59 7f00 0001
0x0010:  7f00 0001 0425 0000 5990 8dd1 4030 b112
0x0020:  5018 0200 dbb0 0000 536f 6d65 2072 616e
0x0030:  646f 6d20 6461 7461 0a00 0000 0000 0000
0x0040:  0000 0000 0000 0000 0000 0000 0000 0000
0x0050:  0000
```

Byte offset 16, length 2 bytes Checksum: dbb0 - 56240
Byte offset 18, length 2 bytes Urgent pointer 0000
Byte offset 20, length 96 bytes Data: “Some random data” + padding.

(TCP HL – Min. TCP HL) = TCP Options Length
(5*4) – 20 = 0 NO Options

IP TL – (IHL + TCP HL) = Payload Length
136 – ((5*4) + (5*4)) = 96 bytes - Rest of padding is not showing/was not captured

Header Offset and Length

Header Offset Shortcuts

<table>
<tr><th colspan="2">Field</th><th>Length (bits)</th><th colspan="2">TCPDUMP Filter</th><th colspan="5">Notes</th></tr>
<tr><td colspan="2">IP Header Length</td><td>4</td><td colspan="2">ip[0] &0x0F</td><td colspan="5">Remember to use a 4 byte multiplier to find header length in bytes</td></tr>
<tr><td colspan="2">IP Packet Length</td><td>16</td><td colspan="2">ip[2:2]</td><td colspan="5">The is no multiple for this length field</td></tr>
<tr><td colspan="2">IP TTL</td><td>8</td><td colspan="2">ip[8]</td><td colspan="5"></td></tr>
<tr><td colspan="2">IP Protocol</td><td>8</td><td colspan="2">ip[9]</td><td colspan="5"></td></tr>
<tr><td rowspan="4"></td><th>D</th><th>Hex</th><th>Proto</th><th>D</th><th>Hex</th><th>Proto</th><th>D</th><th>Hex</th><th>Proto</th></tr>
<tr><td>1</td><td>0x01</td><td>ICMP</td><td>9</td><td>0x09</td><td>IGRP</td><td>47</td><td>0x2F</td><td>GRE</td></tr>
<tr><td>2</td><td>0x02</td><td>IGMP</td><td>17</td><td>0x11</td><td>UDP</td><td>50</td><td>0x32</td><td>ESP</td></tr>
<tr><td>6</td><td>0x06</td><td>TCP</td><td>47</td><td>0x2F</td><td>GRE</td><td>51</td><td>0x33</td><td>AH</td></tr>
<tr><td colspan="2">IP Address - Src</td><td>32</td><td colspan="2">ip[12:4]</td><td colspan="5"></td></tr>
<tr><td colspan="2">IP Address - Dst</td><td>32</td><td colspan="2">ip[16:4]</td><td colspan="5"></td></tr>
<tr><td colspan="2" rowspan="2">IP Fragmentation</td><td>flag=3</td><td colspan="7">ip[6] &0x20 = 0x20 More Fragment bit is set.</td></tr>
<tr><td>offset=13</td><td colspan="7">ip[6:2] &0x1fff != 0x000 fragment offset in not 0</td></tr>
<tr><td colspan="2">ICMP Type</td><td>8</td><td colspan="2">icmp[0]</td><td colspan="5"></td></tr>
<tr><td colspan="2">ICMP Code</td><td>8</td><td colspan="2">icmp[1]</td><td colspan="5"></td></tr>
<tr><td colspan="2">TCP Src Port</td><td>16</td><td colspan="2">tcp[0:2]</td><td colspan="5"></td></tr>
<tr><td colspan="2">TCP Dst Port</td><td>16</td><td colspan="2">tcp[2:2]</td><td colspan="5"></td></tr>
<tr><td colspan="2">TCP Header Length</td><td>4</td><td colspan="2">tcp[12] &0x0F</td><td colspan="5">Remember to use a 4 byte multiplier to find header length in bytes</td></tr>
<tr><td colspan="2">TCP Flags</td><td>8</td><td colspan="2">tcp[13]</td><td colspan="5"></td></tr>
<tr><td colspan="2">TCP Windows Size</td><td>16</td><td colspan="2">tcp[14:2]</td><td colspan="5"></td></tr>
<tr><td colspan="2">UDP Src Port</td><td>16</td><td colspan="2">udp[0:2]</td><td colspan="5"></td></tr>
<tr><td colspan="2">UDP Dst Port</td><td>16</td><td colspan="2">udp[2:2]</td><td colspan="5"></td></tr>
<tr><td colspan="2">UDP Header Length</td><td>16</td><td colspan="2">udp[4:2]</td><td colspan="5">The is no multiple for this length field</td></tr>
</table>

TCPDUMP filters

Two different formats to specify filters:

1. Byte displacement
   - <protocol header> [offset:length] <relation> <value>
     - ip[9] = 6 embedded protocol is TCP
     - tcp[2:2] = 80 destination port is 80
     - udp[6:2] != 0   udp checksum not zero
     - Icmp[0] = 8 echo packet

2. <macro> <value>
   - dst host www.msn.com
   - port != ssh
   - ether src MAC
   - net 10.10.10.0/24

TCPDUMP filters examples

- Look for all IP packets with TCP embedded protocol
  #> tcpdump –nnx –r <pcap.file> ‘ip[9] = 6’

- Look for all TCP packets with flags
  #> tcpdump –nnx –r <pcap.file> ‘tcp[13] != 0’

- Look for all TCP packets with Sync bit only
  #> tcpdump –nnx –r <pcap.file> ‘tcp[13] = 2’

- Look for all ICMP packets with port unreachable
  #> tcpdump –nnx –r <pcap.file> ‘icmp[0] = 3 and icmp[1] = 3’
  Or
  #> tcpdump –nnx –r <pcap.file> ‘icmp[0:2] = 0x0303’

Lab

This lab assumes that tcpdump executable is in your path, if not make sure you run tcpdump using the absolute path. Start the lab by booting into your CNSnort virtual machine and login as root/root (BAD!!!)

Please note that running as root is dangerous and should not be done in real/production environment.

Section 1:
Goal: Run tcpdump to read from a capture file. Change directory to /root/tcpdump_handson/

#> tcpdump –r section1.pcap

How many packets were displayed.
Run it again with name resolution (-n option), is it faster?

Lab Cont.

Section 2:
Goal: Run tcpdump to read from a capture file and only capture the first 3 records.

Note: -c option to specify the record count

Complete the following command to show the first 3 records.
#> tcpdump -r section1.pcap

What is the destination ip address and port number of the second record?

Lab Cont.

Section 3:

Goal: Run tcpdump to read from a capture file and display output in hex.

Note: -x option to specify hex output

Complete the following command to show the first 1 record in hex.

#> tcpdump -r section1.pcap

What are the first 2 bytes from the hex dump? Explain both.

Lab Cont.

Section 4:
Goal: Run tcpdump to read from a ‘traffic.pcap’ capture file and display output using filters.

Example: **#> tcpdump -nn -r traffic.pcap ‘udp’**

Note: Use –nn to suppress name/port resolution and ensure filters are quoted.

Write down the command and filter used to show the first 5 tcp packets from traffic.pcap file.
#> tcpdump

Lab Cont.

Section 5:

Goal: Run tcpdump to read from 'traffic.pcap' capture file and display output using filters.

Note: Use –nn to suppress name/port resolution and ensure filters are quoted.

Find all packet with embedded protocol TCP and destination port 3389

Write down the command and filter used.

#> tcpdump –nn –r traffic.pcap

How many packets found?

Lab Cont.

Section 6:

Goal: Decode the following packet and explain your finding.

```
0x0000:  4500 003c 3759 4000 4006 6f58 ac10 d283
0x0010:  480e cd68 c5cf 0050 a849 206e 0000 0000
0x0020:  a002 16d0 f1af 0000 0204 05b4 0402 080a
0x0030:  0568 1737 0000 0000 0103 0306
```

What is the embedded protocol:

List source and destination ip addresses:

List any IP Flags if present:

What is the TTL? Can you infer the type of OS that generated the packet?

Does the embedded protocol have any options? List them if available?