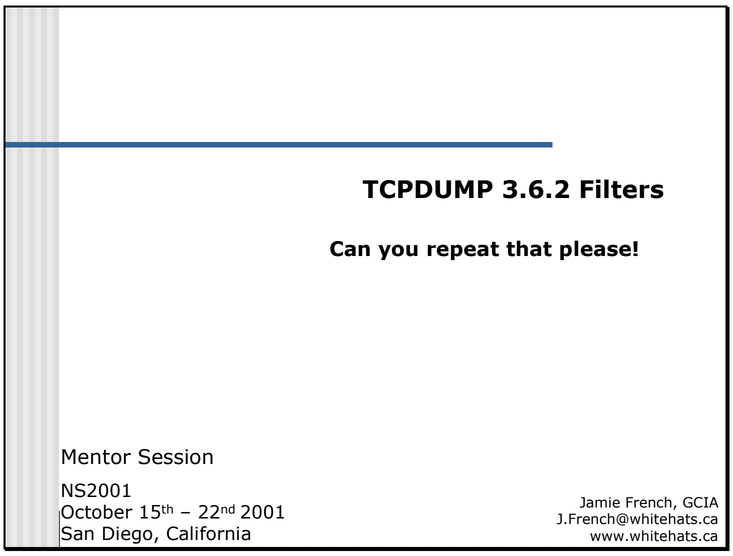
Slide 1



TCPDUMP 3.6.2 Filters

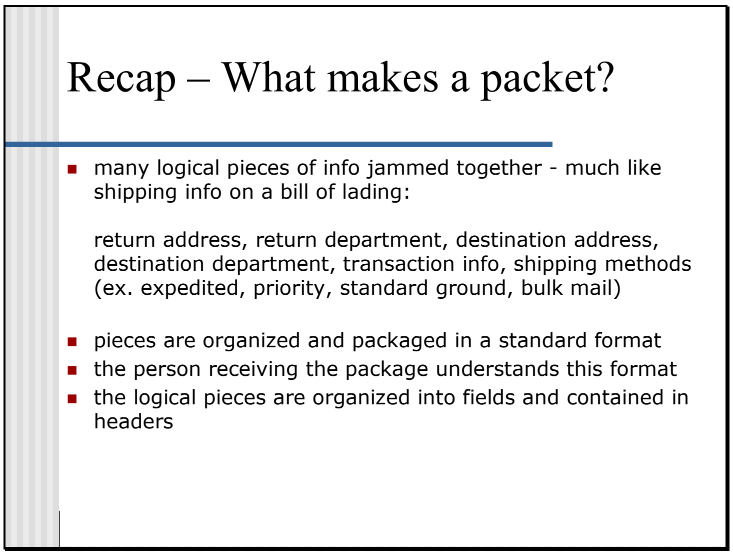
Can you repeat that please!

Mentor Session

NS2001
October 15th – 22nd 2001
San Diego, California

Jamie French, GCIA
J.French@whitehats.ca
www.whitehats.ca

Slide 2



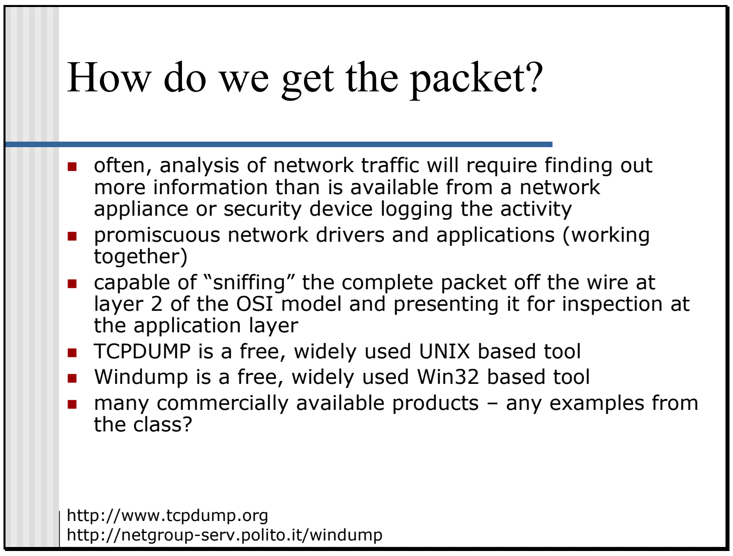
Recap – What makes a packet?

- many logical pieces of info jammed together - much like shipping info on a bill of lading:

  return address, return department, destination address, destination department, transaction info, shipping methods (ex. expedited, priority, standard ground, bulk mail)

- pieces are organized and packaged in a standard format
- the person receiving the package understands this format
- the logical pieces are organized into fields and contained in headers

Slide 3

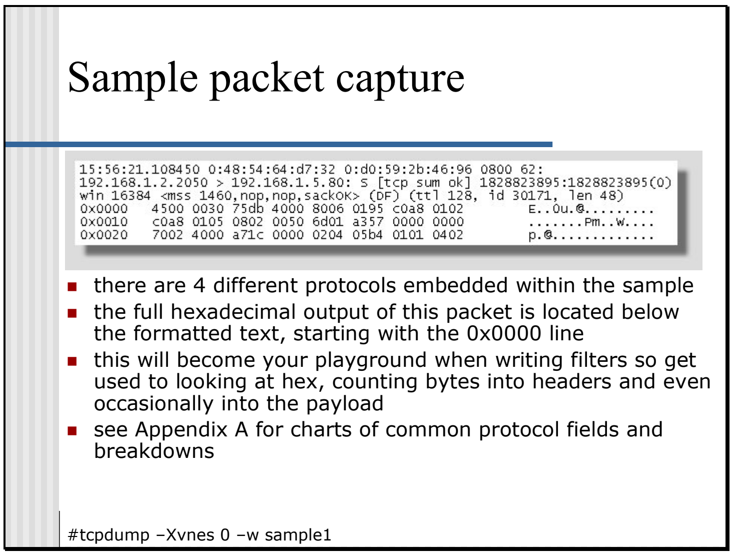


How do we get the packet?

- often, analysis of network traffic will require finding out more information than is available from a network appliance or security device logging the activity
- promiscuous network drivers and applications (working together)
- capable of “sniffing” the complete packet off the wire at layer 2 of the OSI model and presenting it for inspection at the application layer
- TCPDUMP is a free, widely used UNIX based tool
- Windump is a free, widely used Win32 based tool
- many commercially available products – any examples from the class?

http://www.tcpdump.org
http://netgroup-serv.polito.it/windump

Slide 4



Sample packet capture

```
15:56:21.108450 0:48:54:64:d7:32 0:d0:59:2b:46:96 0800 62:
192.168.1.2.2050 > 192.168.1.5.80: S [tcp sum ok] 1828823895:1828823895(0)
win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 128, id 30171, len 48)
0x0000   4500 0030 75db 4000 8006 0195 c0a8 0102        E..0u.@.........
0x0010   c0a8 0105 0802 0050 6d01 a357 0000 0000        .......Pm..W....
0x0020   7002 4000 a71c 0000 0204 05b4 0101 0402        p.@.............
```

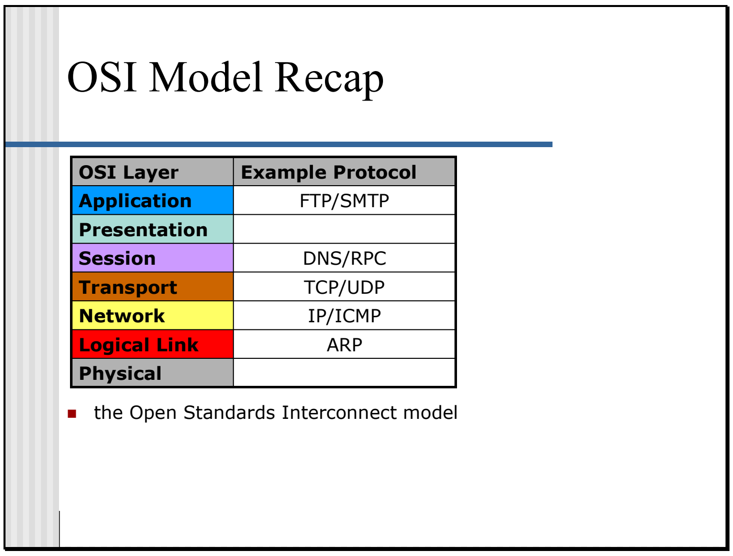
- there are 4 different protocols embedded within the sample
- the full hexadecimal output of this packet is located below the formatted text, starting with the 0x0000 line
- this will become your playground when writing filters so get used to looking at hex, counting bytes into headers and even occasionally into the payload
- see Appendix A for charts of common protocol fields and breakdowns

```
#tcpdump –Xvnes 0 –w sample1
```

The four protocols are:

IEEE 802.3	Layer 2
IP	Layer 3
TCP	Layer 4
HTTP	Layer 7

Slide 5



OSI Model Recap

OSI Layer	Example Protocol
Application	FTP/SMTP
Presentation	
Session	DNS/RPC
Transport	TCP/UDP
Network	IP/ICMP
Logical Link	ARP
Physical	

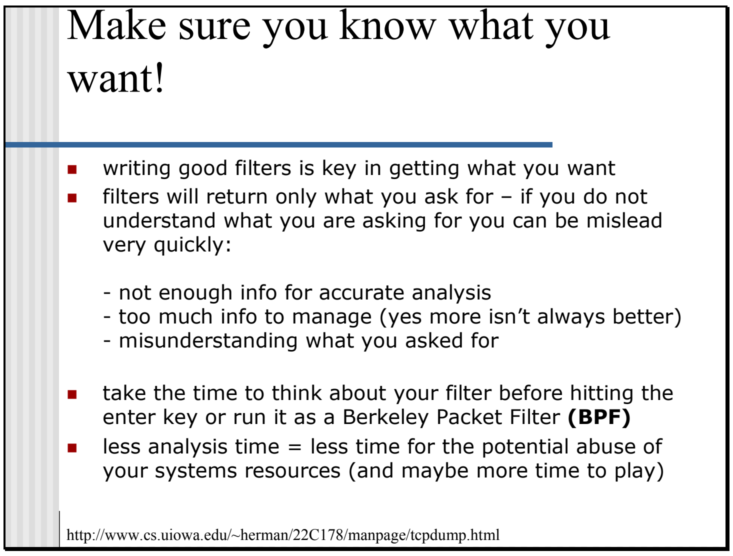
- the Open Standards Interconnect model

For our purposes, you must understand how the overall communications work. We will assume that a user sits above the application layer. To communicate with another user over the network certain steps are carried out.

Take input (any form – i.e. file, keyboard) and pass it to the application to process. The application formats the data into a standardized presentation, such as Hypertext Mark Up Language (HTML).
This is then passed to the Operating System (OS) with a requests to open up a socket. If granted, this serves as the session (link) between the application and the networking stack and its supported protocols. The network stack kicks in on the transport layer. Don't confuse this as thinking we skipped to the network layer.
The transport layer puts this data into a package and prepares it for shipping. Basically it places some necessary data in-front of and behind what was passed to it originally by the presentation layer. This added data will become known as the layer 4 header.
This data is then taken and packaged again by the network layer. This added data will become known as the layer 3 header.
Next the "package" is passed to the data link layer. At this point your data has made it to your network interface card. Another bunch of data is added which will become known as the LLC or logical link control header. In the case of Ethernet networks, people refer to it as the Ethernet header. This is then passed to the hardware and modulated or transformed into a signal that is transmitted. The method of modulation will coincide with the physical link layer (i.e. fiber optic, radio frequency). For our purposes we now have a packet and it is comprised of a stream of 1's and/or 0's. This is then placed onto the physical layer and sent on its way. All the headers added are required for other hosts OSI models to understand what to do with the packet when they receive it and to be able to figure out whether it was for them or not. The implementation of the OSI model on an OS is often referred to as the Network Stack.

The implementation of this model by TCP/IP often melds the application and presentation layers together as one function. It also has some overlap between the transport layer and the network layer. Another important point is that data is unidirectional within the layers. For instance it will not go from layer 3 to layer 2 and then back to layer 3. The data will come in or go out and continue in its current direction until it is processed or discarded.

Slide 6



Make sure you know what you want!

- writing good filters is key in getting what you want
- filters will return only what you ask for – if you do not understand what you are asking for you can be mislead very quickly:

  - not enough info for accurate analysis
  - too much info to manage (yes more isn't always better)
  - misunderstanding what you asked for

- take the time to think about your filter before hitting the enter key or run it as a Berkeley Packet Filter **(BPF)**
- less analysis time = less time for the potential abuse of your systems resources (and maybe more time to play)

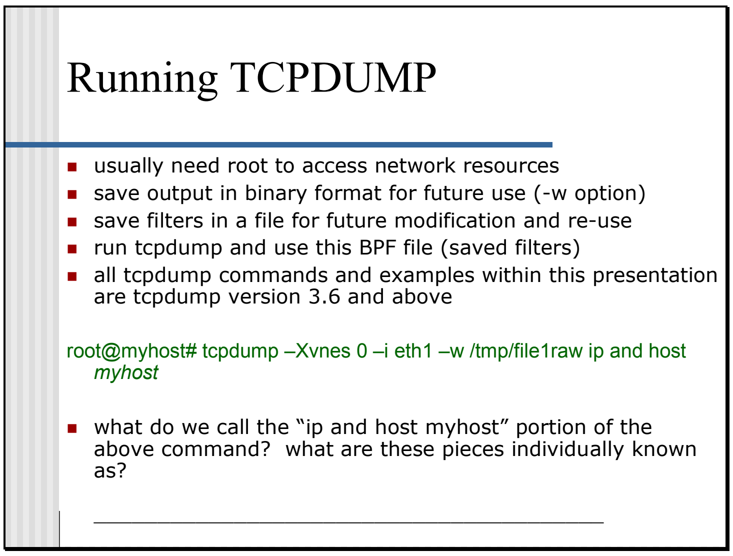
http://www.cs.uiowa.edu/~herman/22C178/manpage/tcpdump.html

From my experience, there are not any really good shortcuts to learning how to write good filters. What helps is always asking yourself the who, what, why, when, where, and how. Writing filters in a file format and then modifying these will provide the stimulus to advance beyond the basics and speed up the learning process. It may seem tedious at first but the benefits should payoff with about a 30% increase in productivity over the short term.

There are many, many options available to help isolate information specifically being sought. One excellent resource that should not be overlooked is the manual page or manpage for TCPDUMP. Almost everything in this presentation is available within the manpage.

Read it and become familiar with it!

Slide 7



Running TCPDUMP

- usually need root to access network resources
- save output in binary format for future use (-w option)
- save filters in a file for future modification and re-use
- run tcpdump and use this BPF file (saved filters)
- all tcpdump commands and examples within this presentation are tcpdump version 3.6 and above

```
root@myhost# tcpdump –Xvnes 0 –i eth1 –w /tmp/file1raw ip and host
    myhost
```

- what do we call the "ip and host myhost" portion of the above command? what are these pieces individually known as?

____

Explanation of tcpdump command with a few options, more complete option list is available in Appendix B

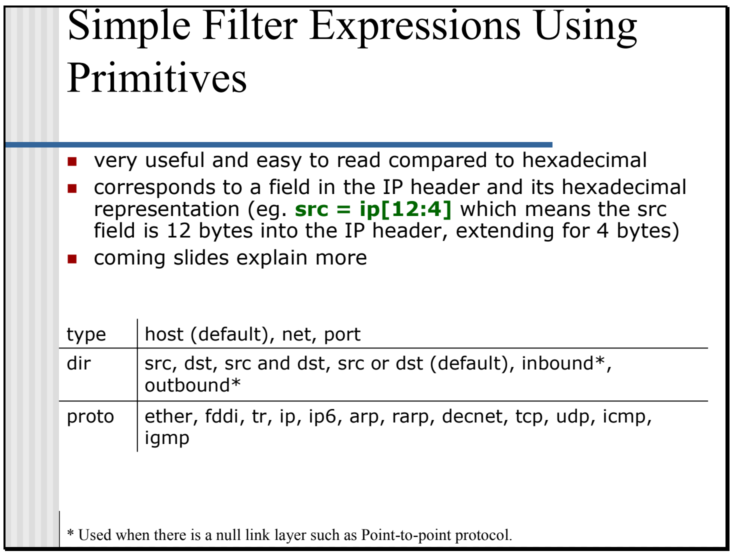
-X	Print each packet in hex and ascii
-v	Verbose output
-n	Don't convert addresses to names
-e	Print the link-level header on each dump line
-s	Snarf <u>snaplen</u> bytes of data from each packet (default is 68) – 0 means the full packet
-i	Listen on <u>interface</u> (defaults to lowest numbered interface)
-w	Write the raw packet to <u>file</u> rather than parsing and printing to stdout

TCPDUMP Options

-a	Attempt to convert network and broadcast addresses to names
-c	Exit after receiving <u>count</u> packets
-d	Dump the compiled packet-matching code in a human readable form to standard output and stop
-dd	Dump packet-matching code as a C program fragment
-ddd	Dump packet-matching code as decimal numbers (preceded with a count)
-e	Print the link-level header on each dump line
-E	Use <u>algo:secret</u> for decrypting IPsec ESP packets
-f	Print 'foreign' internet addresses numerically rather than symbolically
-F	Use <u>file</u> as input for the filter expression
-i	Listen on <u>interface</u> (defaults to lowest numbered interface)
-l	Make stdout line buffered
-n	Don't convert addresses to names
-N	Don't print domain name qualification of host names
-m	Load SMI MIB module definitions from file <u>module</u>

-O	Do not run the packet-matching code optimizer
-P	Don’t put the interface into promiscuous mode
-q	Quick output – print less protocol information
-r	Read packets from file (created with the –w option)
-R	Assume ESP/AH packets to be based on old specs
-s	Snarf snaplen bytes of data from each packet (default is 68)
-S	Print absolute, rather than relative TCP sequence numbers
-t	Don’t print a timestamp on each dump line
-tt	Print an unformatted timestamp on each dump line
-T	Force packets selected by “expressions” to be interpreted the specified type (cnfp, rpc, rtp, snmp, wb)
-v	Verbose output
-vv	Even more verbose output
-vvv	Even more verbose output
-w	Write the raw packet to file rather than parsing and printing to stdout
-x	Print each packet (less link level header) in hex
-X	Print each packet in hex and ascii

Slide 8



Simple Filter Expressions Using Primitives

- very useful and easy to read compared to hexadecimal
- corresponds to a field in the IP header and its hexadecimal representation (eg. **src = ip[12:4]** which means the src field is 12 bytes into the IP header, extending for 4 bytes)
- coming slides explain more

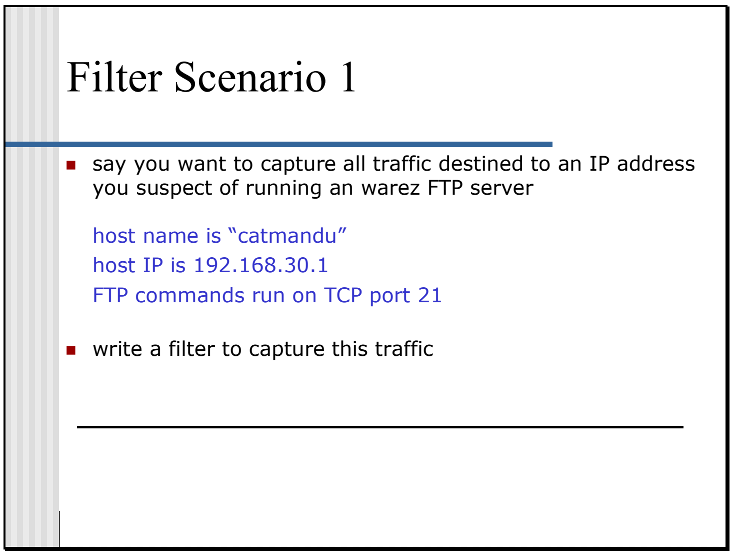
type	host (default), net, port
dir	src, dst, src and dst, src or dst (default), inbound*, outbound*
proto	ether, fddi, tr, ip, ip6, arp, rarp, decnet, tcp, udp, icmp, igmp

* Used when there is a null link layer such as Point-to-point protocol.

Some examples of primitives in use are:

tcpdump –xvn –i eth1 ip and host myhost
capture packets on interface eth1 that are IP and include "myhost" as either the source or
destination – "myhost" is resolved locally as the domain name is not fully qualified
tcpdump –xvn –i eth1 ip and host 192.168.1.1
capture packets on interface eth1 that are IP and include 192.168.1.1 as either the
source or destination
tcpdump –xvn –i eth1 tcp and net 192.168
capture packets on interface eth1 where the source or destination network
is 192.168.0.0/16
tcpdump –xvn –i eth1 src net 10 and udp and dst port 111
capture packets on interface eth1 from network 10.0.0.0/8 that are UDP and
the destination port is 111
tcpdump –xvn –i eth1 src host 10.0.0.1 and tcp or src net 172.16 and icmp
capture packets on interface eth1 from host 10.0.0.1 that are TCP or capture packets
from network 172.16.0.0/16 that are ICMP

Slide 9



Filter Scenario 1

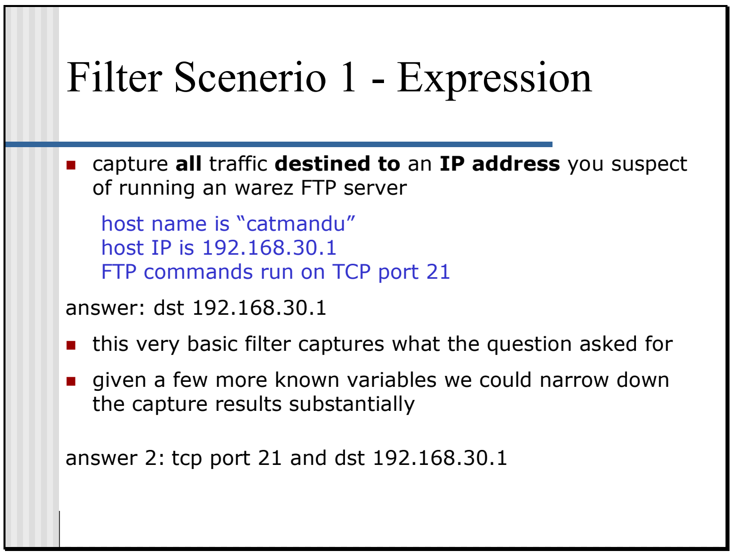
- say you want to capture all traffic destined to an IP address you suspect of running an warez FTP server

  host name is "catmandu"
  host IP is 192.168.30.1
  FTP commands run on TCP port 21

- write a filter to capture this traffic

Look at the previous slide and examples on the use of primitives if you get stuck.

Slide 10



Filter Scenerio 1 - Expression

- capture **all** traffic **destined to** an **IP address** you suspect of running an warez FTP server

  host name is "catmandu"
  host IP is 192.168.30.1
  FTP commands run on TCP port 21

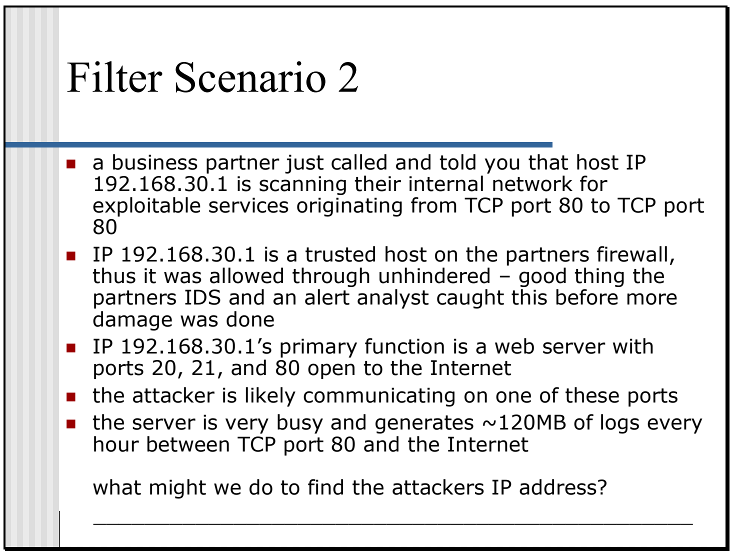
answer: dst 192.168.30.1

- this very basic filter captures what the question asked for
- given a few more known variables we could narrow down the capture results substantially

answer 2: tcp port 21 and dst 192.168.30.1

FTP runs on the TCP protocol on port 21. You may be interested in any protocol besides TCP running on port 21 too but for now we will just look at the **proto** primitive and TCP. The **direction** primitive is used to specify dst or destination and since host is the default **type** primitive we do not need to specify host before the IP address.

Slide 11



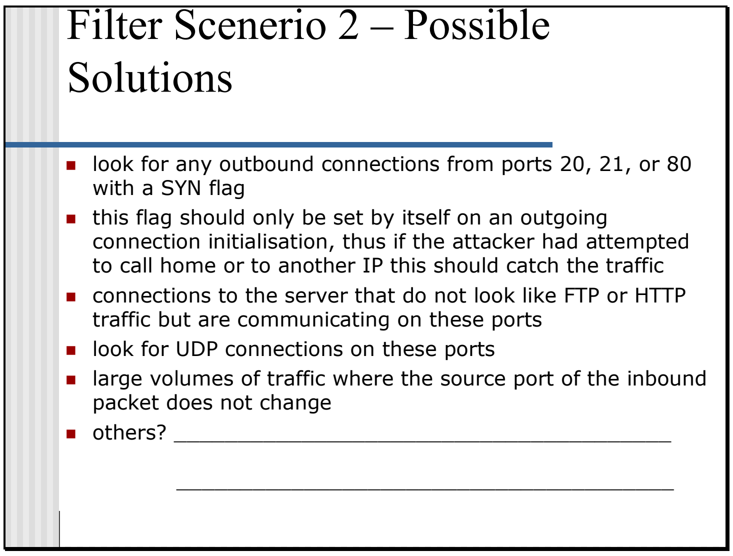
Filter Scenario 2

- a business partner just called and told you that host IP 192.168.30.1 is scanning their internal network for exploitable services originating from TCP port 80 to TCP port 80
- IP 192.168.30.1 is a trusted host on the partners firewall, thus it was allowed through unhindered – good thing the partners IDS and an alert analyst caught this before more damage was done
- IP 192.168.30.1's primary function is a web server with ports 20, 21, and 80 open to the Internet
- the attacker is likely communicating on one of these ports
- the server is very busy and generates ~120MB of logs every hour between TCP port 80 and the Internet

what might we do to find the attackers IP address?

Not all organizations security policies would allow for the tracking down of an offender once an intrusion has been identified. It is a viable option that can be explored, especially if evidence needs to be captured in order to prosecute and aid in assessing the damage. This is a call that I would suggest should be made on an incident by incident basis depending on an immediate risk assessment at the time of the noted intrusion. Just be sure to follow your organizations security policy when it comes time to make a decision.

Slide 12

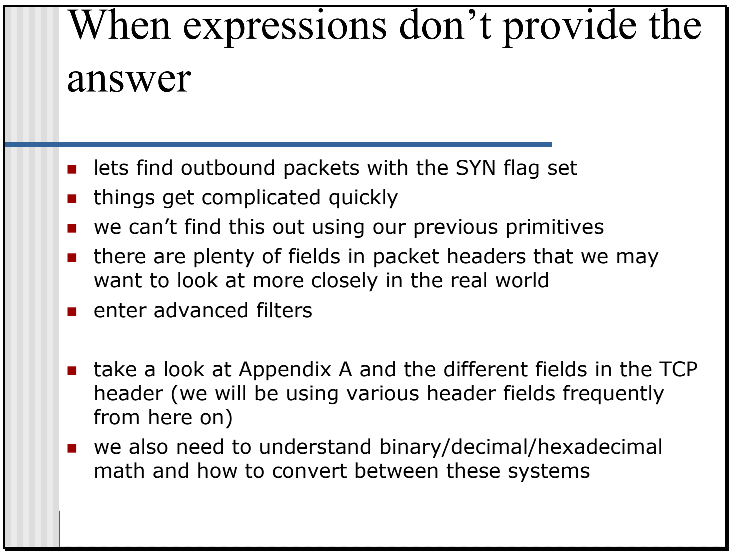


Filter Scenerio 2 – Possible Solutions

- look for any outbound connections from ports 20, 21, or 80 with a SYN flag
- this flag should only be set by itself on an outgoing connection initialisation, thus if the attacker had attempted to call home or to another IP this should catch the traffic
- connections to the server that do not look like FTP or HTTP traffic but are communicating on these ports
- look for UDP connections on these ports
- large volumes of traffic where the source port of the inbound packet does not change
- others? ____

  ____

Slide 13

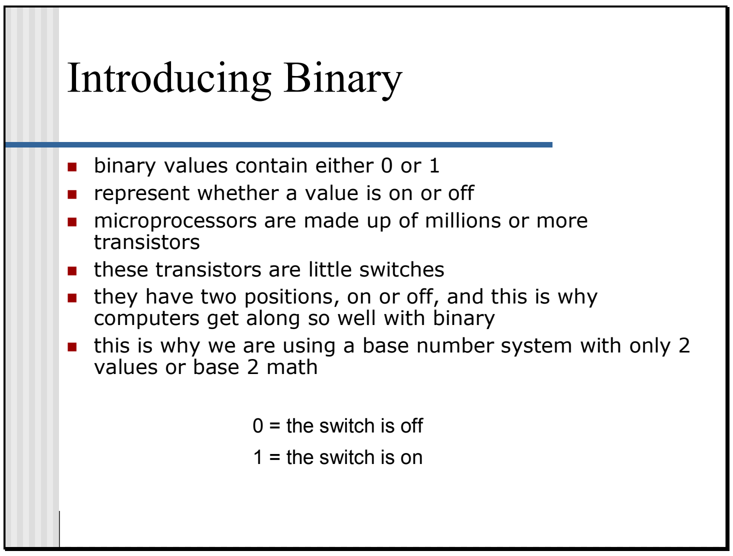


When expressions don’t provide the answer

- lets find outbound packets with the SYN flag set
- things get complicated quickly
- we can’t find this out using our previous primitives
- there are plenty of fields in packet headers that we may want to look at more closely in the real world
- enter advanced filters

- take a look at Appendix A and the different fields in the TCP header (we will be using various header fields frequently from here on)
- we also need to understand binary/decimal/hexadecimal math and how to convert between these systems

Slide 14



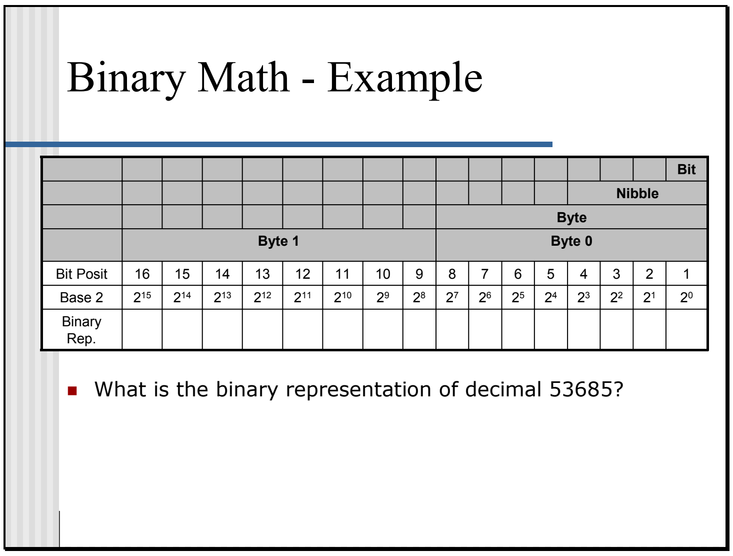
Introducing Binary

- binary values contain either 0 or 1
- represent whether a value is on or off
- microprocessors are made up of millions or more transistors
- these transistors are little switches
- they have two positions, on or off, and this is why computers get along so well with binary
- this is why we are using a base number system with only 2 values or base 2 math

0 = the switch is off

1 = the switch is on

Slide 15



Binary Math - Example

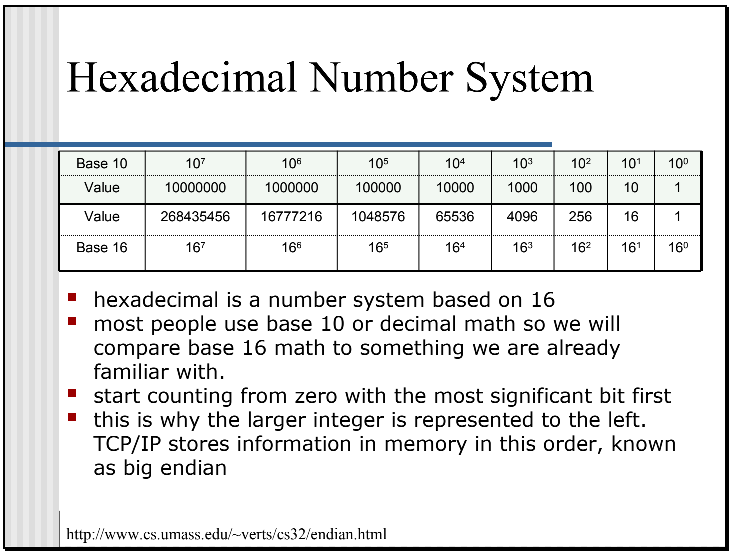
																Bit
													Nibble			
									Byte							
	Byte 1								**Byte 0**							
Bit Posit	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Base 2	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Binary Rep.																

- What is the binary representation of decimal 53685?

2 y^x 15 = 32768	Is 53685 divisible by 32768? Yes. 53685 - 32768 = 20917
2 y^x 14 = 16384	Is 20917 divisible by 16384? Yes. 20917 - 16384 = 4533
2 y^x 13 = 8192	Is 4533 divisible by 8192? No. Bit value 0. Next…
2 y^x 12 = 4096	Is 4533 divisible by 4096? Yes. 4533 - 4096 = 437
2 y^x 11 = 2048	Is 437 divisible by 2048? No. Bit value 0. Next…
2 y^x 10 = 1024	Is 437 divisible by 1024? No. Bit value 0. Next…
2 y^x 9 = 512	Is 437 divisible by 512? No. Bit value 0. Next…
2 y^x 8 = 256	Is 437 divisible by 256? Yes. 437 - 256 = 181
2 y^x 7 = 128	Is 181 divisible by 128? Yes. 181 - 128 = 53
2 y^x 6 = 64	Is 53 divisible by 64? No. Bit value 0. Next…
2 y^x 5 = 32	Is 53 divisible by 32? Yes. 53 - 32 = 21
2 y^x 4 = 16	Is 21 divisible by 16? Yes. 21 - 16 = 5
2 y^x 3 = 8	Is 5 divisible by 8? No. Bit value 0. Next…
2 y^x 2 = 4	Is 5 divisible by 4? Yes. 5 - 4 = 1
2 y^x 1 = 2	Is 1 divisible by 2? No. Bit value 0. Next…
2 y^x 0 = 1	Is 1 divisible by 1? Yes. 1 - 1 = 0.

This should give us 1101000110110101 in binary.

Slide 16



Hexadecimal Number System

Base 10	10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0
Value	10000000	1000000	100000	10000	1000	100	10	1
Value	268435456	16777216	1048576	65536	4096	256	16	1
Base 16	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0

- hexadecimal is a number system based on 16
- most people use base 10 or decimal math so we will compare base 16 math to something we are already familiar with.
- start counting from zero with the most significant bit first
- this is why the larger integer is represented to the left. TCP/IP stores information in memory in this order, known as big endian

http://www.cs.umass.edu/~verts/cs32/endian.html

For every decimal position you increase the maximum value attainable by a multiple of the base for that number system. 10^4 has a value of 10000 or 10x10x10x10. Remember that base 0 covers the numbers 0 thru 9.

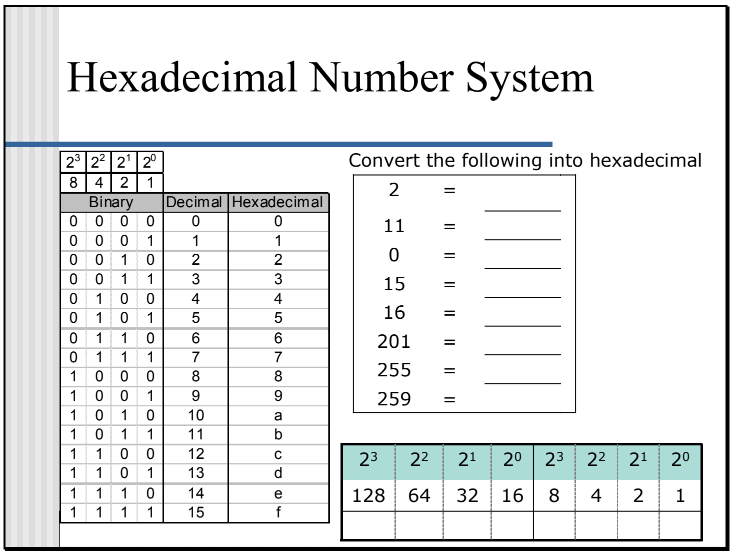
For instance if you take the number 6063 in decimal it would be broken down as follows:

1000’s = 6, 100’s = 0, 10’s = 6, Plus a remainder of 3.

The integer 3 was not high enough to increase the exponent of the 10’s therefore it is a remainder. This is because it is less than a whole unit. A unit in decimal is = 10.

The representation of decimal is flat so it would be represented as 6063 and that is why humans like base 10 math. Apply the same concepts to base 16 and we should be able to work through it.

Slide 17



Hexadecimal Number System

2^3	2^2	2^1	2^0		
8	4	2	1		
Binary				Decimal	Hexadecimal
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0	10	a
1	0	1	1	11	b
1	1	0	0	12	c
1	1	0	1	13	d
1	1	1	0	14	e
1	1	1	1	15	f

Convert the following into hexadecimal

2	=	______
11	=	______
0	=	______
15	=	______
16	=	______
201	=	______
255	=	______
259	=	

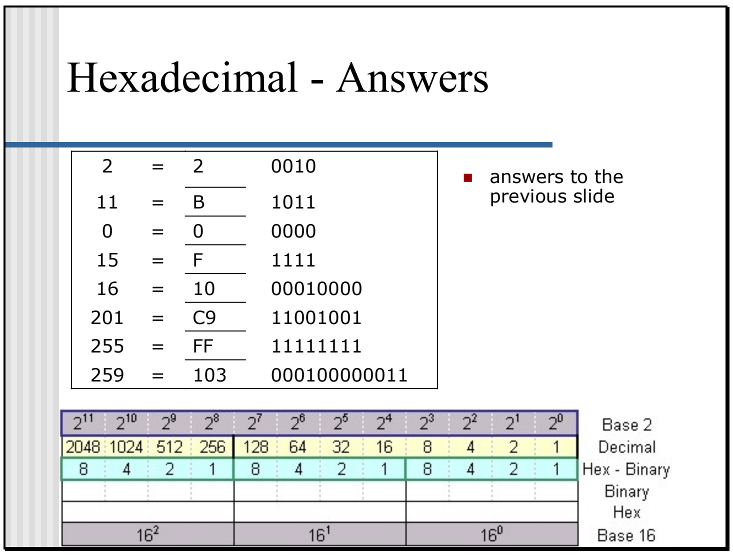
2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Since the maximum value of a base unit in hexadecimal is higher than 10 (0 thru 9) we have to use other symbols to represent the numbers 10 thru 15. The letters “a” thru “f” are used.

The maximum base value of 16 means that we can store only 4 binary 0’s or 1’s in the amount of space that would represent a hexadecimal number. See the chart in the slide showing the representation in binary and the values of each number in comparison.

Now that we know a hexadecimal number takes 4 bits to represent, we know that 2 hexadecimal numbers will take up 8 bits or 1 byte. Therefore each byte is represented by 2 hexadecimal numbers. Given this information you should be able to complete the exercise above.

Slide 18



Hexadecimal - Answers

2	=	2	0010
11	=	B	1011
0	=	0	0000
15	=	F	1111
16	=	10	00010000
201	=	C9	11001001
255	=	FF	11111111
259	=	103	000100000011

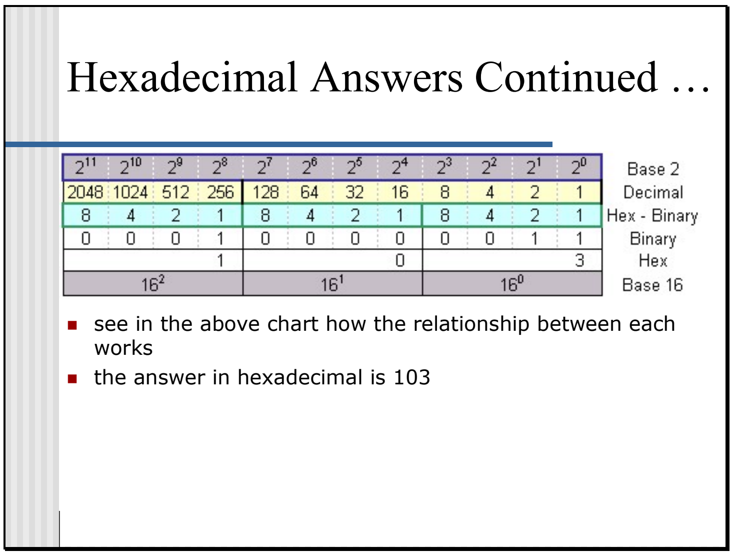
- answers to the previous slide

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Base 2
2048	1024	512	256	128	64	32	16	8	4	2	1	Decimal
8	4	2	1	8	4	2	1	8	4	2	1	Hex - Binary
												Binary
												Hex
16^2				16^1				16^0				Base 16

Some people find it easier to translate into binary and then into hexadecimal. We will try this first to help reinforce the relationships between each number system.

Fill in the work area in your book with the binary representation of each decimal number, then figure out the corresponding hexadecimal value.

Slide 19



Hexadecimal Answers Continued …

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Base 2
2048	1024	512	256	128	64	32	16	8	4	2	1	Decimal
8	4	2	1	8	4	2	1	8	4	2	1	Hex - Binary
0	0	0	1	0	0	0	0	0	0	1	1	Binary
			1				0				3	Hex
	16^2				16^1				16^0			Base 16

- see in the above chart how the relationship between each works
- the answer in hexadecimal is 103

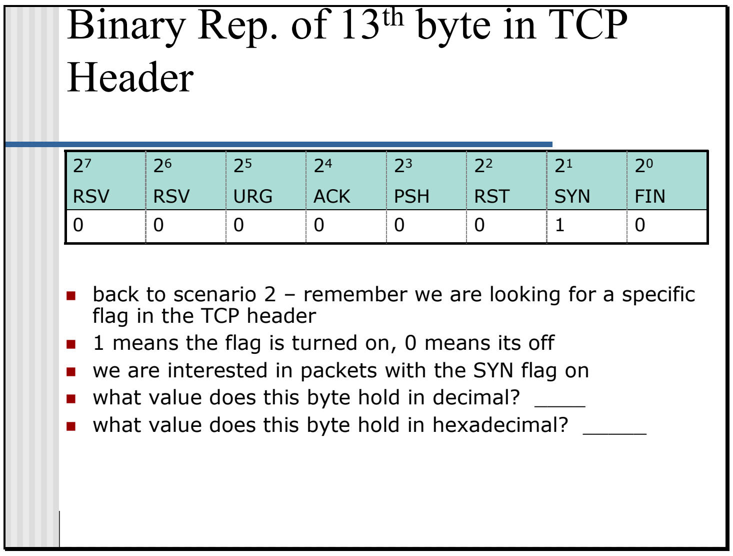
Another method of converting decimal to hexadecimal is illustrated below.

Take decimal 259 and convert it to hexadecimal. We have to find a starting point. We'll try $16^{3.}$

$16\ y^x\ 3 = 4096$ Is 259 divisible by 4096? No. Hex value 0. Next…
$16\ y^x\ 2 = 256$ Is 259 divisible by 256? Yes!
259 / 256 = 1.01171875
Binary representation is 0001 or hex 1
.01171875 * 256 = 3 (this is the remainder)
$16\ y^x\ 1 = 16$ Is 3 divisible by 16? No. Hex value 0. Next…
$16\ y^x\ 0 = 1$ Is 3 divisible by 1? Yes!
3 / 1 = 3
Binary representation is 0011 or hex 3

This should give us 100000011 in binary and 103 hex.

Slide 20

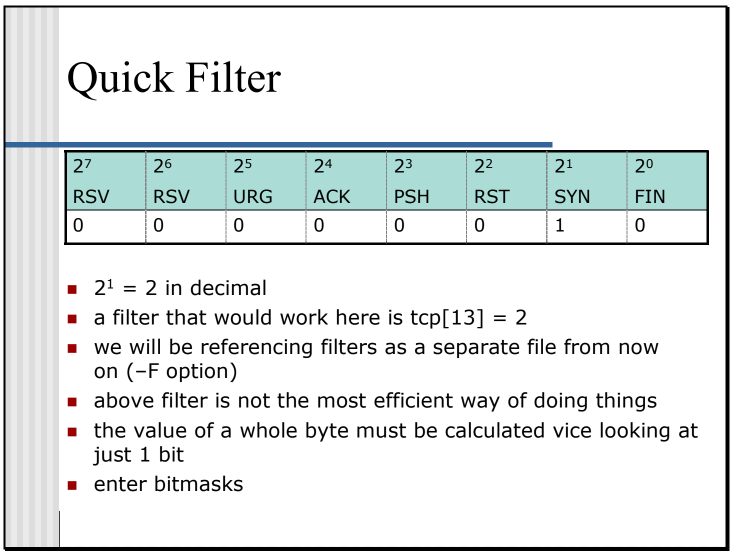


Binary Rep. of 13th byte in TCP Header

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
RSV	RSV	URG	ACK	PSH	RST	SYN	FIN
0	0	0	0	0	0	1	0

- back to scenario 2 – remember we are looking for a specific flag in the TCP header
- 1 means the flag is turned on, 0 means its off
- we are interested in packets with the SYN flag on
- what value does this byte hold in decimal? _____
- what value does this byte hold in hexadecimal? ______

Slide 21



Quick Filter

2^7 RSV	2^6 RSV	2^5 URG	2^4 ACK	2^3 PSH	2^2 RST	2^1 SYN	2^0 FIN
0	0	0	0	0	0	1	0

- 2^1 = 2 in decimal
- a filter that would work here is tcp[13] = 2
- we will be referencing filters as a separate file from now on (–F option)
- above filter is not the most efficient way of doing things
- the value of a whole byte must be calculated vice looking at just 1 bit
- enter bitmasks

Tcpdump compares values in decimal. For instance 2^4 = 16 decimal or 10 hexadecimal. We would express this in a tcpdump filter as ***tcp[13] = 16***

Breakout of the filter: ***tcp[13] = 2***

tcp because the header we want tcpdump to look at is the tcp header – this is where the flags are
[13] specifies the 13th byte into the header – we start counting from byte 0
= 2 because 2^1 = 2 and the decimal value we want for this whole byte is 2

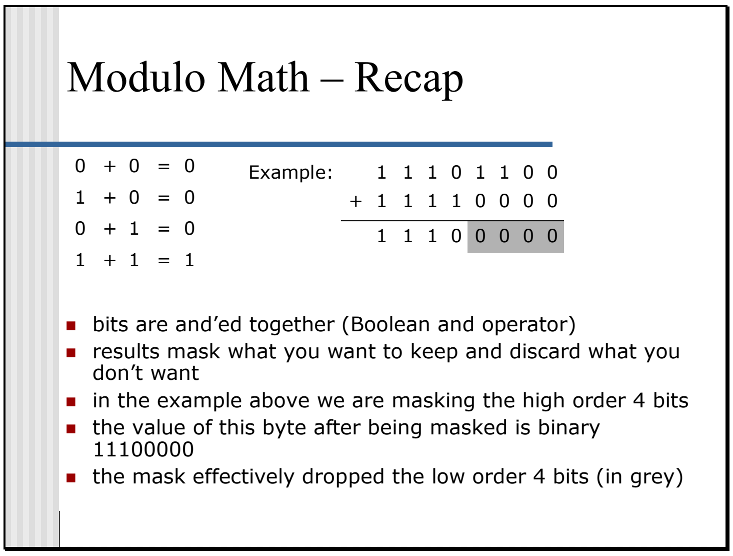
This filter would capture all traffic with only the SYN flag set. In a tcpdump command format it would look something like this:

tcpdump –Xvns 0 –i eth1 *–F /tcpdump/filters/synflagfilter1* –w /tmp/synflag1

synflagfilter1 would look like this:

tcp[13] = 2

Slide 22



Modulo Math – Recap

0	+	0	=	0
1	+	0	=	0
0	+	1	=	0
1	+	1	=	1

```
Example:    1 1 1 0 1 1 0 0
          + 1 1 1 1 0 0 0 0
          -----------------
            1 1 1 0 0 0 0 0
```

- bits are and'ed together (Boolean and operator)
- results mask what you want to keep and discard what you don't want
- in the example above we are masking the high order 4 bits
- the value of this byte after being masked is binary 11100000
- the mask effectively dropped the low order 4 bits (in grey)

Other conversions that this example is equal to are:

Original Byte:

Binary =	11101100
Hexadecimal =	EC
Decimal =	236

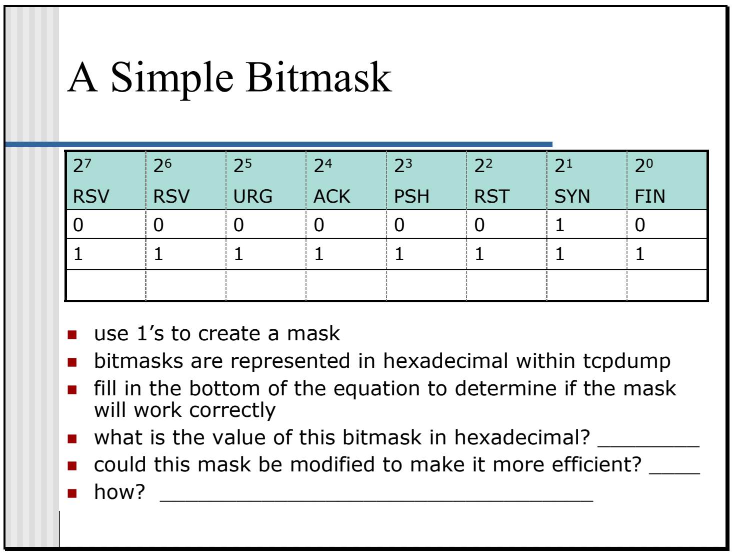
The Mask:

Binary =	11110000
Hexadecimal =	F0
Decimal =	240

Remainder after mask:

Binary =	11100000
Hexadecimal =	E0
Decimal =	224

Slide 23



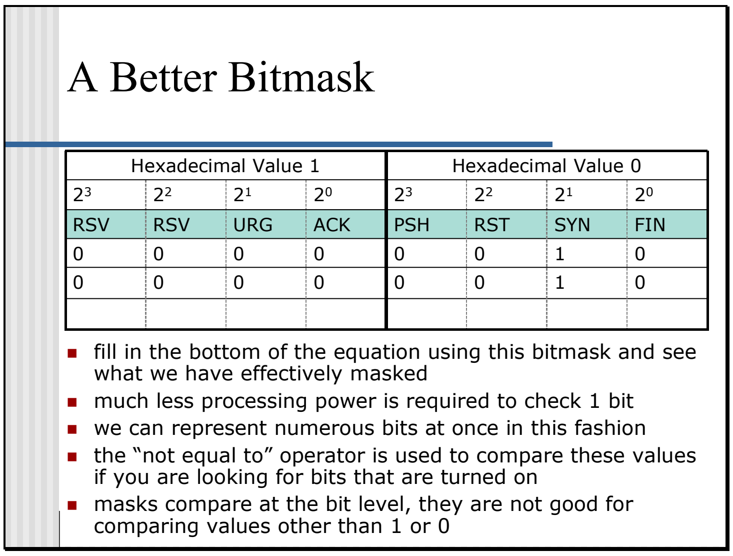
A Simple Bitmask

2^7 RSV	2^6 RSV	2^5 URG	2^4 ACK	2^3 PSH	2^2 RST	2^1 SYN	2^0 FIN
0	0	0	0	0	0	1	0
1	1	1	1	1	1	1	1

- use 1's to create a mask
- bitmasks are represented in hexadecimal within tcpdump
- fill in the bottom of the equation to determine if the mask will work correctly
- what is the value of this bitmask in hexadecimal? _________
- could this mask be modified to make it more efficient? _____
- how? ____

Binary **11111111** = **255** decimal = **ff** hexadecimal

Slide 24



A Better Bitmask

Hexadecimal Value 1				Hexadecimal Value 0			
2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
RSV	RSV	URG	ACK	PSH	RST	SYN	FIN
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0

- fill in the bottom of the equation using this bitmask and see what we have effectively masked
- much less processing power is required to check 1 bit
- we can represent numerous bits at once in this fashion
- the "not equal to" operator is used to compare these values if you are looking for bits that are turned on
- masks compare at the bit level, they are not good for comparing values other than 1 or 0

Operators:

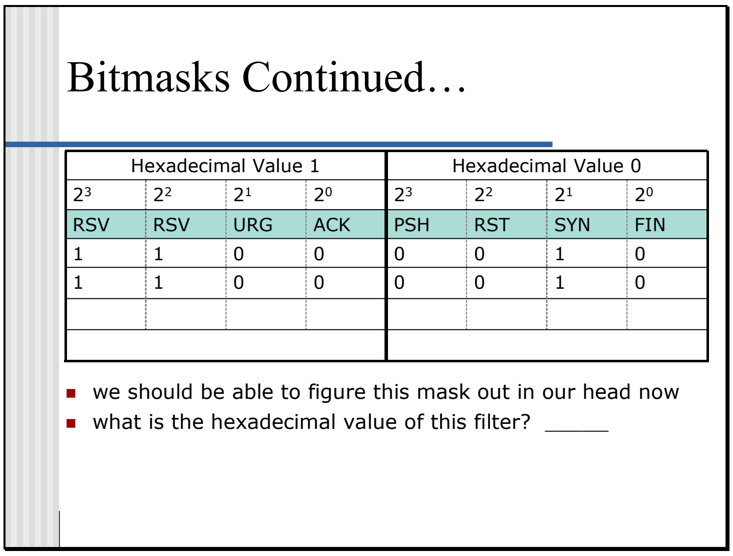
!	=	not
&&	=	and
||	=	or

tcpdump will accept either the symbolic representation or the textual operator

>	=	greater than
<	=	less than

tcpdump will accept only the symbolic representation

Slide 25



Bitmasks Continued…

Hexadecimal Value 1				Hexadecimal Value 0			
2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
RSV	RSV	URG	ACK	PSH	RST	SYN	FIN
1	1	0	0	0	0	1	0
1	1	0	0	0	0	1	0

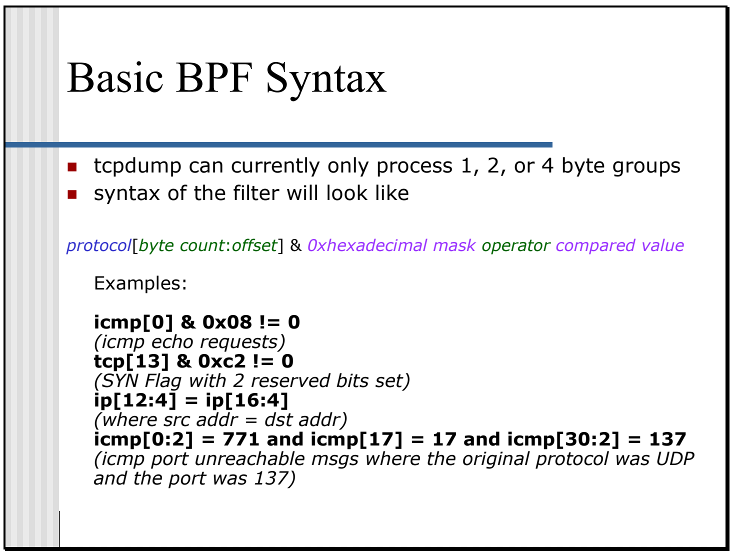
- we should be able to figure this mask out in our head now
- what is the hexadecimal value of this filter? ______

The mask we have come up with only checks these bits. If for example we use the mask in a filter as follows:

tcp[13] & 0xc2 != 0

This will capture all the packets with both reserved flags set as well as the syn flag. Don't forget that we ignored all the other bits and as long as the packet has these set it will be filtered. Thus a packet with both reserved bits, the URG, ACK, PSH and SYN flags set would be filtered as well.

Slide 26



Basic BPF Syntax

- tcpdump can currently only process 1, 2, or 4 byte groups
- syntax of the filter will look like

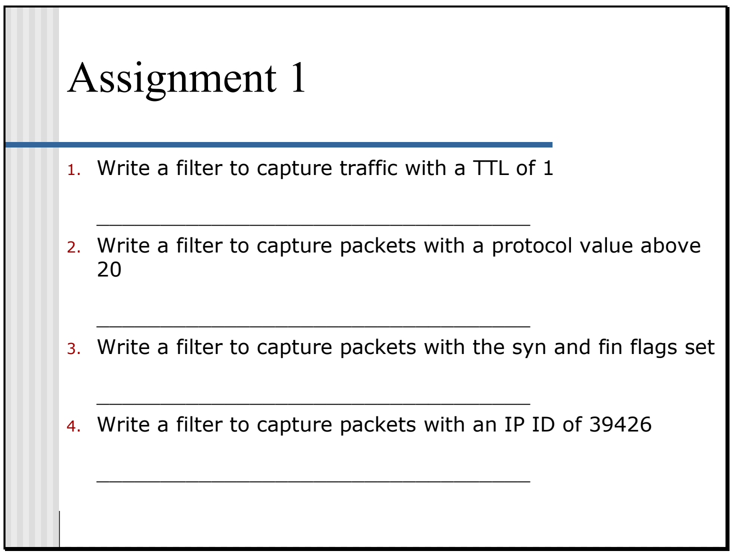
protocol[*byte count*:*offset*] & *0xhexadecimal mask operator compared value*

Examples:

icmp[0] & 0x08 != 0
(icmp echo requests)
tcp[13] & 0xc2 != 0
(SYN Flag with 2 reserved bits set)
ip[12:4] = ip[16:4]
(where src addr = dst addr)
icmp[0:2] = 771 and icmp[17] = 17 and icmp[30:2] = 137
(icmp port unreachable msgs where the original protocol was UDP and the port was 137)

Note that some of the characters used must be escaped so that they are not interpreted by the shell. See Appendix C for a short list of these characters. If they are read in using the –F option they should not have to be escaped. This is another good reason to save your filters.

Slide 27



Assignment 1

1. Write a filter to capture traffic with a TTL of 1

   ____

2. Write a filter to capture packets with a protocol value above 20

   ____

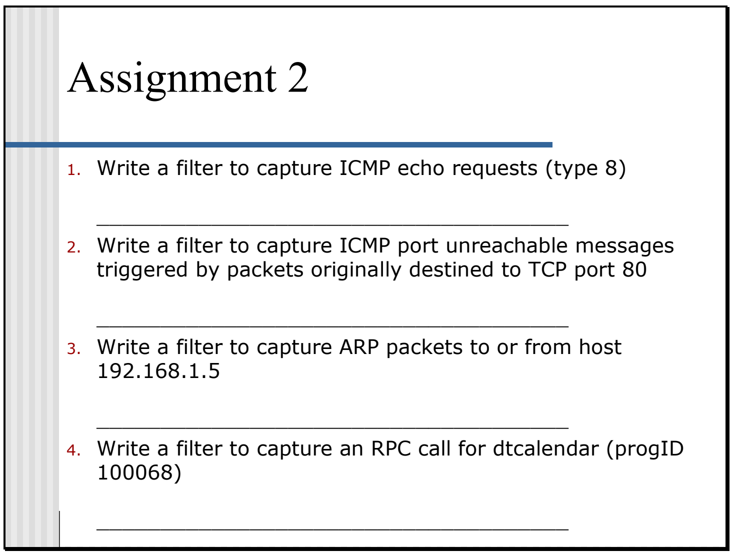
3. Write a filter to capture packets with the syn and fin flags set

   ____

4. Write a filter to capture packets with an IP ID of 39426

   ____

Slide 28



Assignment 2

1. Write a filter to capture ICMP echo requests (type 8)

   ____

2. Write a filter to capture ICMP port unreachable messages triggered by packets originally destined to TCP port 80

   ____

3. Write a filter to capture ARP packets to or from host 192.168.1.5

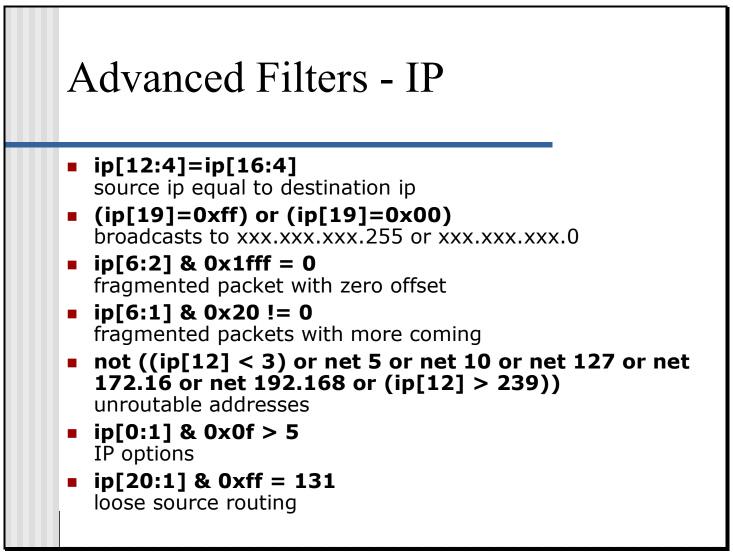
   ____

4. Write a filter to capture an RPC call for dtcalendar (progID 100068)

   ____

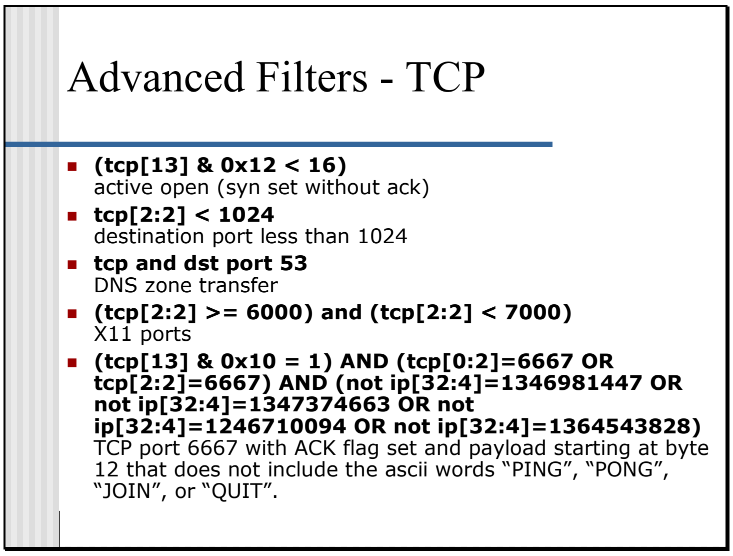
Answers in appendix D

Slide 29



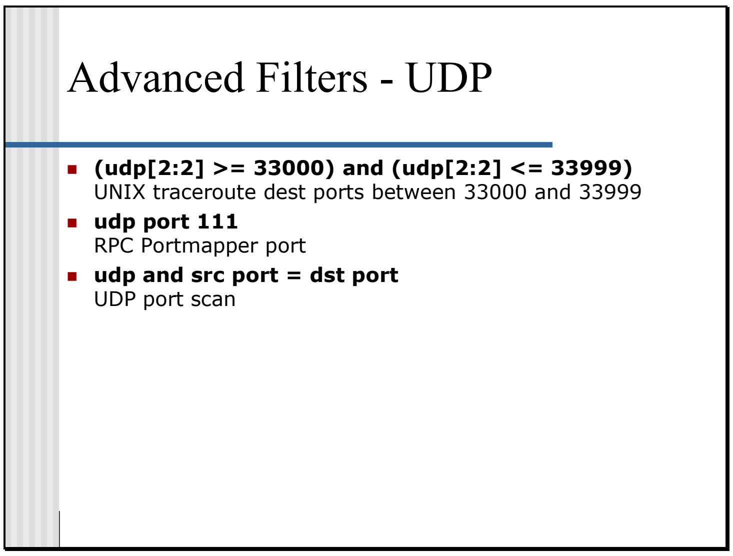
Advanced Filters - IP

- **ip[12:4]=ip[16:4]**
  source ip equal to destination ip
- **(ip[19]=0xff) or (ip[19]=0x00)**
  broadcasts to xxx.xxx.xxx.255 or xxx.xxx.xxx.0
- **ip[6:2] & 0x1fff = 0**
  fragmented packet with zero offset
- **ip[6:1] & 0x20 != 0**
  fragmented packets with more coming
- **not ((ip[12] < 3) or net 5 or net 10 or net 127 or net 172.16 or net 192.168 or (ip[12] > 239))**
  unroutable addresses
- **ip[0:1] & 0x0f > 5**
  IP options
- **ip[20:1] & 0xff = 131**
  loose source routing



Advanced Filters - TCP

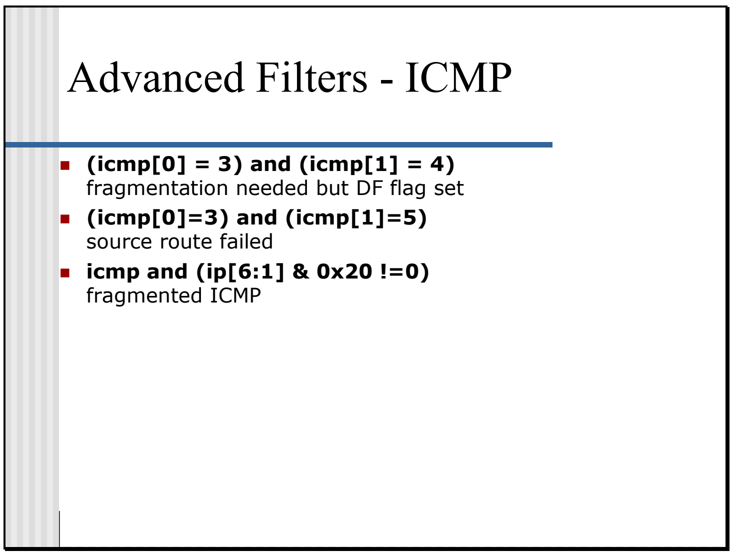
- **(tcp[13] & 0x12 < 16)**
  active open (syn set without ack)
- **tcp[2:2] < 1024**
  destination port less than 1024
- **tcp and dst port 53**
  DNS zone transfer
- **(tcp[2:2] >= 6000) and (tcp[2:2] < 7000)**
  X11 ports
- **(tcp[13] & 0x10 = 1) AND (tcp[0:2]=6667 OR tcp[2:2]=6667) AND (not ip[32:4]=1346981447 OR not ip[32:4]=1347374663 OR not ip[32:4]=1246710094 OR not ip[32:4]=1364543828)**
  TCP port 6667 with ACK flag set and payload starting at byte 12 that does not include the ascii words "PING", "PONG", "JOIN", or "QUIT".



Advanced Filters - UDP

- **(udp[2:2] >= 33000) and (udp[2:2] <= 33999)**
  UNIX traceroute dest ports between 33000 and 33999
- **udp port 111**
  RPC Portmapper port
- **udp and src port = dst port**
  UDP port scan

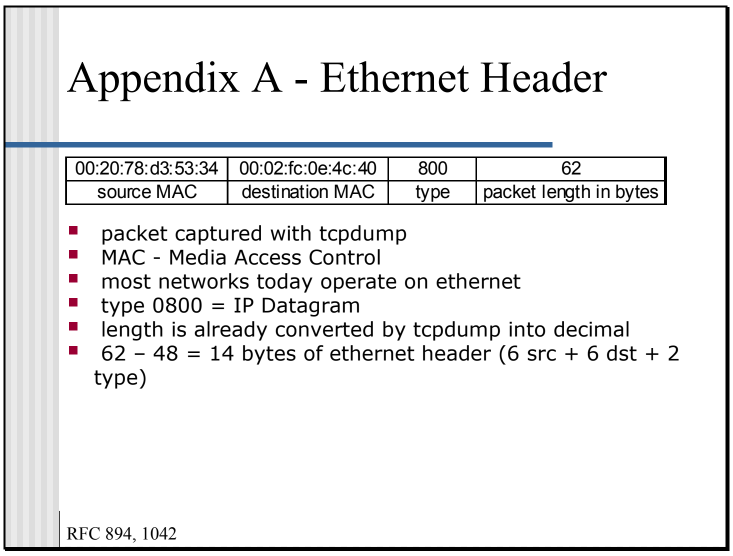
Slide 32



Advanced Filters - ICMP

- **(icmp[0] = 3) and (icmp[1] = 4)**
  fragmentation needed but DF flag set
- **(icmp[0]=3) and (icmp[1]=5)**
  source route failed
- **icmp and (ip[6:1] & 0x20 !=0)**
  fragmented ICMP

Slide 33



Appendix A - Ethernet Header

00:20:78:d3:53:34	00:02:fc:0e:4c:40	800	62
source MAC	destination MAC	type	packet length in bytes

- packet captured with tcpdump
- MAC - Media Access Control
- most networks today operate on ethernet
- type 0800 = IP Datagram
- length is already converted by tcpdump into decimal
- 62 – 48 = 14 bytes of ethernet header (6 src + 6 dst + 2 type)

RFC 894, 1042

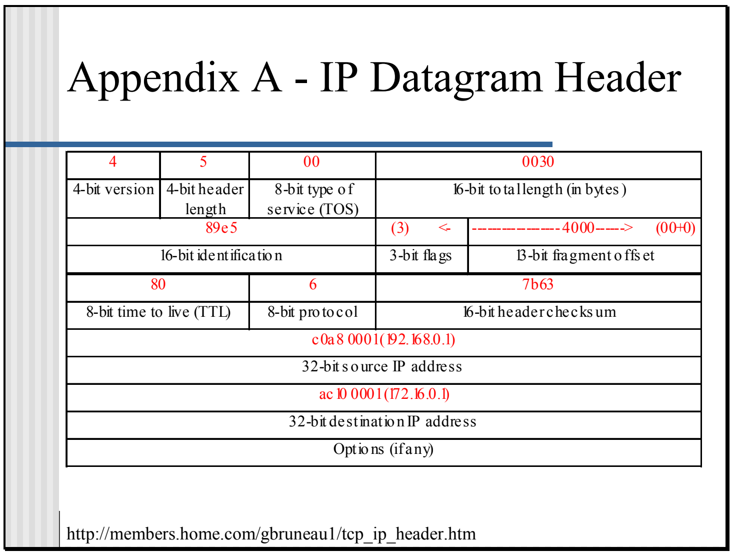
Example Packet:

08:38:29.231855 **0:20:78:d3:53:34 0:2:fc:e:4c:40 0800 62:** 192.168.0.1.2089 > 172.16.0.1.23: S [tcp sum ok] 1870179867:1870179867(0) win 16384 <mss 1436,nop,nop,sackOK> (DF) (ttl 128, id 35300, **len 48**)

```
4500 0030 89e4 4000 8006 7b64 c0a8 0001
ac10 0001 0829 0017 6f78 ae1b 0000 0000
7002 4000 27e4 0000 0204 059c 0101 0402
```

Notice the bolded len 48. This is a value calculated at layer 3 and presented by tcpdump.

Slide 34



Appendix A - IP Datagram Header

4	5	00	0030	
4-bit version	4-bit header length	8-bit type of service (TOS)	16-bit total length (in bytes)	
89e5			(3) <-	--------------4000-----> (00+0)
16-bit identification			3-bit flags	13-bit fragment offset
80		6	7b63	
8-bit time to live (TTL)		8-bit protocol	16-bit header checksum	
c0a8 0001(192.168.0.1)				
32-bit source IP address				
ac10 0001(172.16.0.1)				
32-bit destination IP address				
Options (if any)				

http://members.home.com/gbruneau1/tcp_ip_header.htm

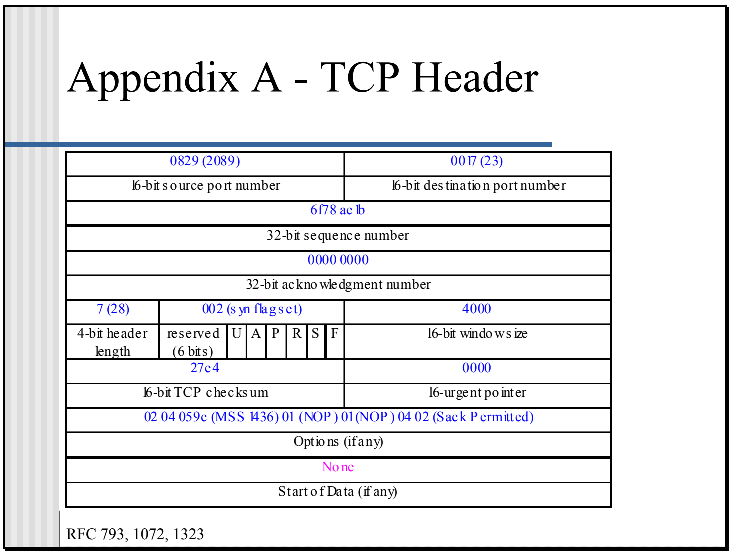
Example Packet:

08:38:29.231855 0:20:78:d3:53:34 0:2:fc:e:4c:40 0800 62: 192.168.0.1.2089 > 172.16.0.1.23: S [tcp sum ok]
1870179867:1870179867(0) win 16384 <mss 1436,nop,nop,sackOK> (DF) (ttl 128, id 35300, len 48)

4500 0030 89e4 4000 8006 7b64 c0a8 0001
ac10 0001 0829 0017 6f78 ae1b 0000 0000
7002 4000 27e4 0000 0204 059c 0101 0402

Byte 6 and 7 into the IP header present a challenge. The method of arriving at the real value of these fields is by using base 2 math. The hexadecimal value of 4000 must first be converted into binary to do this. We will look more closely at base 2 math, conversions, and binary in the coming slides.

Slide 35



Appendix A - TCP Header

0829 (2089)								0017 (23)
16-bit source port number								16-bit destination port number
6f78 ae1b								
32-bit sequence number								
0000 0000								
32-bit acknowledgment number								
7 (28)	002 (syn flag set)							4000
4-bit header length	reserved (6 bits)	U	A	P	R	S	F	16-bit window size
27e4								0000
16-bit TCP checksum								16-urgent pointer
02 04 059c (MSS 1436) 01 (NOP) 01 (NOP) 04 02 (Sack Permitted)								
Options (if any)								
None								
Start of Data (if any)								

RFC 793, 1072, 1323

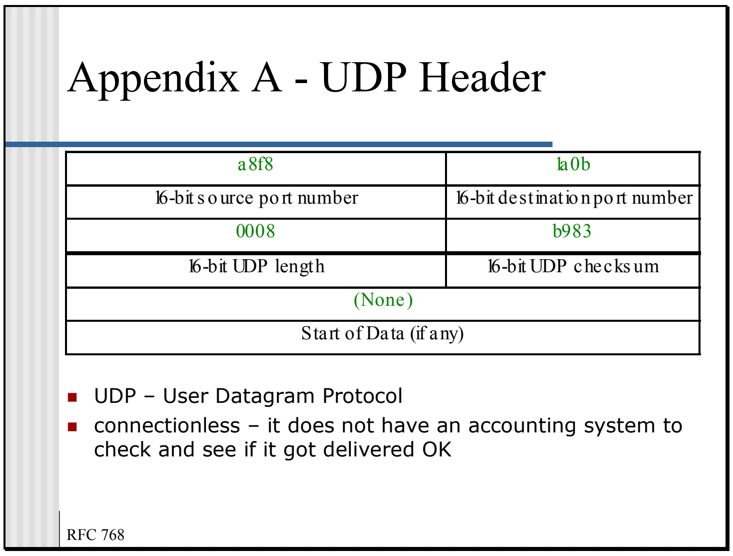
Example Packet:

08:38:29.231855 0:20:78:d3:53:34 0:2:fc:e:4c:40 0800 62: 192.168.0.1.2089 > 172.16.0.1.23: S [tcp sum ok] 1870179867:1870179867(0) win 16384 <mss 1436,nop,nop,sackOK> (DF) (ttl 128, id 35300, len 48)

```
4500 0030 89e4 4000 8006 7b64 c0a8 0001
ac10 0001 0829 0017 6f78 ae1b 0000 0000
7002 4000 27e4 0000 0204 059c 0101 0402
```

The tricky field here is the 12th and 13th byte positions. We will discuss how to break up fields that don't end on a byte boundary in the following slides.

Slide 36



Appendix A - UDP Header

a8f8	1a0b
16-bit source port number	16-bit destination port number
0008	b983
16-bit UDP length	16-bit UDP checksum
(None)	
Start of Data (if any)	

- UDP – User Datagram Protocol
- connectionless – it does not have an accounting system to check and see if it got delivered OK

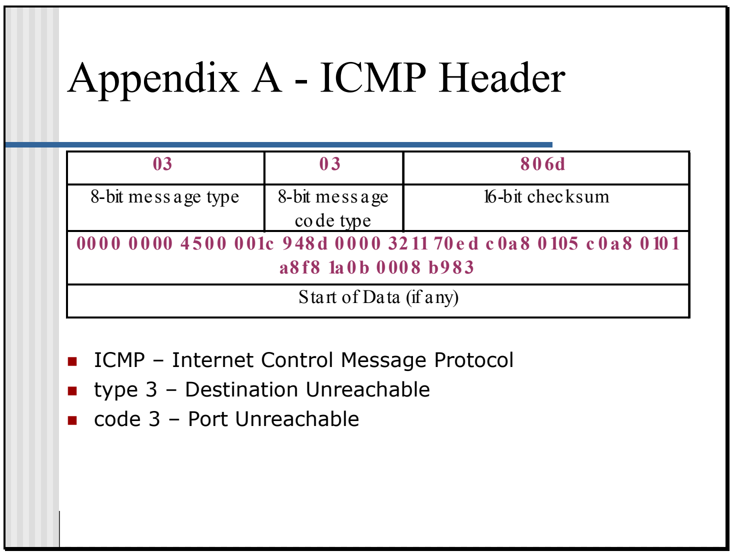
RFC 768

New Example Packet:

```
09:57:53.097634 0:d0:59:2b:46:96 0:20:78:d3:53:33 0800 42: 192.168.1.5.43256 >
192.168.1.1.6667:  [udp sum ok]
udp 0 (ttl 50, id 38029, len 28)
                        4500 001c 948d 0000 3211 70ed c0a8 0105
                        c0a8 0101 a8f8 1a0b 0008 b983
```

Notice the bold 11. This is the protocol field in the IP header. Hex 11 converted to decimal is 17 which is the protocol value of UDP.

Slide 37



Appendix A - ICMP Header

03	03	806d
8-bit message type	8-bit message code type	16-bit checksum
0000 0000 4500 001c 948d 0000 3211 70ed c0a8 0105 c0a8 0101 a8f8 1a0b 0008 b983		
Start of Data (if any)		

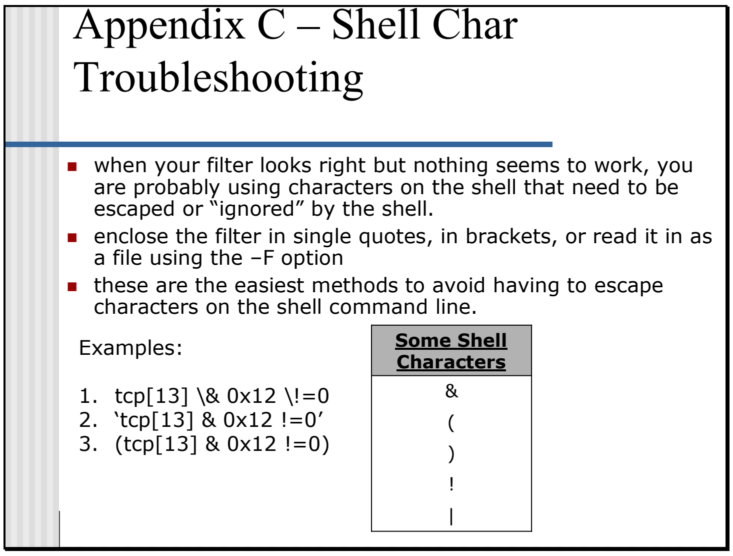
- ICMP – Internet Control Message Protocol
- type 3 – Destination Unreachable
- code 3 – Port Unreachable

New Sample Packet:

09:57:53.097998 0:20:78:d3:53:33 0:d0:59:2b:46:96 0800 70: 192.168.1.1 > 192.168.1.5: icmp: 192.168.1.1 udp port
6667 unreachable (ttl 64, id 38029, len 56)
4500 0038 948d 0000 40**01** 62e1 c0a8 0101
c0a8 0105 **0303 806d** 0000 0000 4500 001c
948d 0000 3211 70ed c0a8 0105 c0a8 0101
a8f8 1a0b 0008 b983

Notice the bold 01 in the IP header. This is the protocol value for ICMP. Also notice the original IP header and part of the UDP header are included as the data payload of the ICMP message. You can dig into this and get more information about what caused the ICMP message to be generated.

Slide 38



Appendix C – Shell Char Troubleshooting

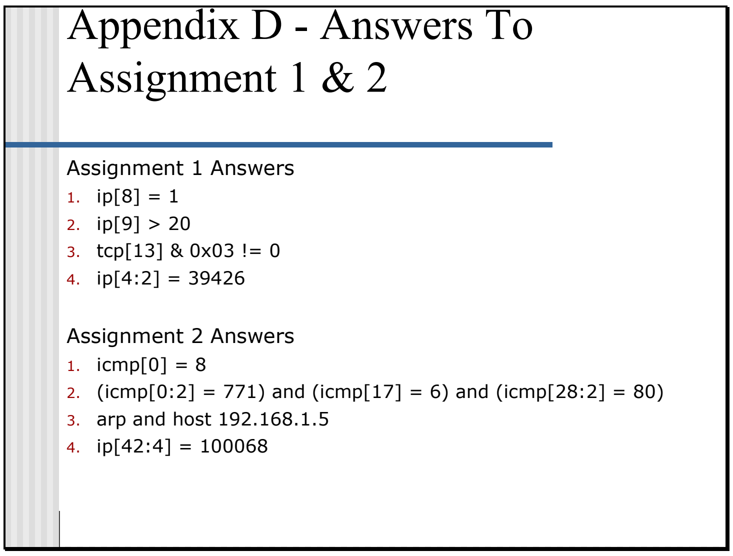
- when your filter looks right but nothing seems to work, you are probably using characters on the shell that need to be escaped or "ignored" by the shell.
- enclose the filter in single quotes, in brackets, or read it in as a file using the –F option
- these are the easiest methods to avoid having to escape characters on the shell command line.

Examples:

1. tcp[13] \& 0x12 \!=0
2. ‘tcp[13] & 0x12 !=0’
3. (tcp[13] & 0x12 !=0)

Some Shell Characters
&
(
)
!
|

Slide 39



Appendix D - Answers To Assignment 1 & 2

Assignment 1 Answers

1. ip[8] = 1
2. ip[9] > 20
3. tcp[13] & 0x03 != 0
4. ip[4:2] = 39426

Assignment 2 Answers

1. icmp[0] = 8
2. (icmp[0:2] = 771) and (icmp[17] = 6) and (icmp[28:2] = 80)
3. arp and host 192.168.1.5
4. ip[42:4] = 100068