TCPdump Basics

What we will cover:

What is/are TCPdump/WinDump?
Why use TCPdump?
Installation of TCPdump on Unix/Windows
It's installed, now what?
Changing the amount of data collected
Reading TCPdump/WinDump Output
TCP Flags in TCPdump/WinDump
Absolute & Relative Sequence Numbers
Dumping TCPdump/WinDump output in hexadecimal format
TCPdump ManPage

TCPdump is a tool we can use for packet analysis. We will not use ethereal (wireshark) because it does "too much" for us. TCPdump will keep everything "raw" and that's the way we like it!

*note: it is expected that the reader will try out all the TCPdump/WinDump commands listed in the lesson on their own computers as well as the practical exercises. You have to actually run the commands to learn how to use the tool.

What is/are TCPdump/WinDump?

TCPdump, and its cousin WinDump, is software that allows us to see inside the traffic activity that occurs on a network. TCPdump is a Unix tool used to gather data from the network, decipher the bits, and display the output in a human readable format (granted it does take a little bit of instruction to learn the TCPdump language).

Why use TCPdump?

Network traffic travels in data packets; each data packet contains the information that it needs to travel across the network. This information is contained in a TCP header. A TCP header will contain the destination and source address, state information, and protocol identifiers. The rest of the packet contains the data that is being sent. Devices that are responsible for routing read the information in these packets and send them to their correct destination. Sniffing is a process that passively monitors and captures these packets. TCPdump is a packet-sniffing tool that is used by network administrators to sniff and analyze traffic on a network. A couple of reasons for sniffing traffic on a network would be to verify connectivity between hosts, or to analyze the traffic that is traversing the network.

TCPdump and WinDump are available at:
http://www.TCPdump.org/ & http://WinDump.polito.it/

To install for Unix/Linux:

Most Linux distributions install a version of TCPdump as part of a standard operating system install. Of course, this depends on the options you choose during the installation. If a custom install is chosen, then it is possible that this package will not be available until you install it manually.

Installing TCPdump from the RPM:
To see if you have TCPdump installed on your system, type the following command from a Linux shell: rpm –q TCPdump

This should show you some output similar to the following (it may look slightly different depending on the version you have installed):

```
[root@tcp4sec root]# rpm -q TCPdump
TCPdump-3.7.2-1.9.1
```

(Note: rpm represents RedHat Package Management, the –q option represents query. The –i option represents install, the –v is for verbose and the –h is to display status in the form of a hash mark. You can find more information regarding the use of rpm by reading the rpm man page.)

If you do not have TCPdump installed you should see something like this:

```
[root@tcp4sec root]# rpm -q TCPdump
package TCPdump is not installed
```

If the package is not installed, you can get the RPM from the RedHat CD. This is probably the easiest method of installation, however, installation from the source will be covered as well. First, verify that the libpcap rpm is installed. If it is not, then install libpcap

```
rpm –ivh libpcap-0.7.2-1.i386.rpm
```

then…

```
rpm –ivh TCPdump-3.7.2-1.9.1.rpm
```

This will install the packages and the you'll be ready to use TCPdump.

Installing TCPdump using apt-get
If your distribution has apt-get you can use apt-get to install TCPdump. Apt-get is nice in that will usually install dependencies for you which is always a plus.

```
Apt-get install tcpdump
```

Or maybe

`Apt-get upgrade tcpdump` (if you already have tcpdump installed and just want to upgrade)

Installing TCPdump from the source files:
If you do not have access to the operating system CD's; an alternative way to install TCPdump is to point a web browser to http://www.TCPdump.org and find the most current version. It is important to note that libpcap must be installed prior to the installation of TCPdump. This is a library file, "which provides a packet filtering mechanism based on the

BSD packet filter (BPF)." (http://freshmeat.net/projects/libpcap/) TCPdump will not function without it. libpcap can also be found on http://www.TCPdump.org. Download the appropriate files and save them to a temporary directory. Change to the temporary directory and type:

```
tar –zxvf libpcap-0.8.3.tar.gz
```

after this extracts completely, type

```
tar –zxvf TCPdump-3.8.3.tar.gz
```

*Note: tar is an archiving program designed to store and extract files from an archive file known as a tarfile.

This will unzip the package and unpack it in one smooth operation. After you have completed this step, you will see the TCPdump-3.8.3 and the libpcap-0.8.3 directories. First, change to the libpcap-0.7.2 directory. As each process finishes, type:

```
./configure
./make  or   make
./make install  or make install
```

(Note: You must be root or have root privileges to run ./make install)

Repeat this process from within the TCPdump-3.8.3 directory. This will install libpcap and TCPdump. You should be ready to use the program at this point.

To install for Windows:
Installing WinDump for windows is much easier.
You have two choices, if you already have WinPcap installed you can just download the WinDump executable and run it from the command line. Or, you can download the installer executable that will install WinPcap for you as well. Simple enough.

OK I got it installed, now what?
On most Unix/Linux, you will need root access to run TCPdump because reading packets requires access to devices accessible to root only. On Windows, if you got it installed, you should be good to go. To run TCPdump/WinDump (for now on TCPdump and WinDump will be used interchangeably unless otherwise noted) just type:

TCPdump

```
[root@TCPIP4Sec root]# TCPdump
TCPdump: listening on eth0

0 packets received by filter
0 packets dropped by kernel
```

WinDump

```
C:\Documents and Settings\NoOne\Desktop\HackerCLI>WinDump.exe
WinDump.exe: listening on \Device\NPF_{11FB32C3-7A19-4106-949D-
6824D157C848}
```

By default, this reads all the traffic from the default network interface and spits it all the output to the console. Unless you can read really, really fast and have a really good memory this won't be the preferred output method for you. Luckily the programmers included some command line options to change the default behavior of the program.

To check out the list of interfaces available (Windows) type:

WinDump –D

```
C:\Documents and Settings\NoOne\Desktop\HackerCLI>WinDump -D
1.\Device\NPF_{F8C4038F-1462-4A50-BD0C-917801DBFC75} (Belkin 11Mbps
Wireless Notebook Network Adapter (Microsoft's Packet Scheduler) )
2.\Device\NPF_{11FB32C3-7A19-4106-949D-6824D157C848} (Intel(R) PRO/100 VE
Network Connection (Microsoft's Packet Scheduler) )
3.\Device\NPF_{886AFBE2-7B11-4F68-AE41-0D357F392C2A} (VMware Virtual
Ethernet Adapter)
4.\Device\NPF_{60315F4B-0F96-4D15-9F09-62AFF391E1A7} (VMware Virtual
Ethernet Adapter)
```

For Unix/Linux do you an

```
ifconfig
```

As root to see the available interfaces.

To select an interface type:

WinDump –i 1

```
C:\Documents and Settings\NoOne\Desktop\HackerCLI>WinDump -i 1
WinDump: listening on \Device\NPF_{F8C4038F-1462-4A50-BD0C-917801DBFC75}
```

TCPdump –i eth0

```
[root@TCPIP4Sec root]# TCPdump -i eth0
TCPdump: listening on eth0

0 packets received by filter
0 packets dropped by kernel
```

To select the type of traffic you want to watch you can just specify after your interface. For now we want to see TCP traffic.

```
[root@TCPIP4Sec root]# TCPdump -i eth0 tcp
```

```
TCPdump: listening on eth0

0 packets received by filter
0 packets dropped by kernel
```

OR

```
C:\Documents and Settings\NoOne\Desktop\HackerCLI>WinDump -i 1 tcp
WinDump: listening on \Device\NPF_{F8C4038F-1462-4A50-BD0C-917801DBFC75}
20 packets received by filter
0 packets dropped by kernel
```

Well that works ok if you just want to see TCP traffic, but as we progress we might want to look at one specific thing or a number of things that force us to create a really long filter. Luckily, someone thought of this and we don't have to type our filter every time if we don't want to. We can simply type it into a text file and just call it with the **–F filename** option of TCPdump. This way we only have to create it once, cut and paste it and we don't have to worry about fat fingering it next time we want to do the same type of analysis

```
[root@TCPIP4Sec root]# TCPdump -i eth0 -F myfilter.txt
TCPdump: listening on eth0

0 packets received by filter
0 packets dropped by kernel
[root@TCPIP4Sec root]# more myfilter.txt
tcp
[root@TCPIP4Sec root]#
```

Or

```
C:\Documents and Settings\NoOne\Desktop\HackerCLI>WinDump -i 1 -F
myfilter.txt
WinDump: listening on \Device\NPF_{F8C4038F-1462-4A50-BD0C-917801DBFC75}

21 packets received by filter
0 packets dropped by kernel
```

Ok so we have handled the filter problem but we still have all this great data whizzing by us. TCPdump gives us the option to dump the records into binary format to read later with TCPdump. We do this using the **–w filename** option.

```
[root@TCPIP4Sec root]# TCPdump -i eth0 -F myfilter.txt -w LSOoutput
TCPdump: listening on eth0

118 packets received by filter
0 packets dropped by kernel
```

Or

```
C:\Documents and Settings\NoOne\Desktop\HackerCLI>WinDump -i 1 -F
myfilter.txt -w LSOoutput
WinDump: listening on \Device\NPF_{F8C4038F-1462-4A50-BD0C-917801DBFC75}
```

```
4 packets received by filter
0 packets dropped by kernel
```

And to read that file back in we use the **–r filename** option, gee that makes sense; read = –r & write = –w.

```
[root@TCPIP4Sec root]# TCPdump -i eth0 -F myfilter.txt -r LSOoutput
09:26:25.194409 192.168.64.128.32815 > 66-193-231.87.dimenoc.com.http: S
1706723745:1706723745(0) win 5840 <mss 1460,sackOK,timestamp 308912
0,nop,wscale 0> (DF)
---EDITED-------
[root@TCPIP4Sec root]#
```

Or

```
C:\Documents and Settings\NoOne\Desktop\HackerCLI>WinDump -i 1 -F
myfilter.txt -r LSOoutput
16:42:21.831154 IP TCPIP4Sec.54473 > 192.168.1.1.5678: F
2881894966:2881894966(0) ack 28 06171 win 17520 (DF)
16:42:22.534859 IP 192.168.1.1.5678 > TCPIP4Sec.54473: F 1:1(0) ack ---
EDITED--------
C:\Documents and Settings\NoOne\Desktop\HackerCLI>
```

Changing the amount of data collected

TCPdump/WinDump has a default snapshot length of the size of datagram it collects; 68 bytes. Depending on what you are doing this may not be adequate, so they give us the option to change it. TCPdump does not collect the entire datagram. This is because of volume concerns and most of the time we are concerned with the headers and not so much the rest of the datagram. With a 68 byte header you get the framer header (14 bytes), the IP header (20 bytes), the TCP header (20 bytes), and the TCP data (14 bytes); basically the entire Ethernet header. What you don’t get in all of this is the complete payload. To alter the default snaplen you use the **TCPdump –s length** command where length is the desired number of bytes to be collected. If you want to capture an entire Ethernet frame, not including the trailer, use **TCPdump –s 1514**. This will capture the 14 byte header and the maximum transmission unit length for Ethernet of 1500 bytes.

Let’s see an example:

```
C:\Documents and Settings\NoOne\Desktop\HackerCLI>windump -i 1 –s 1514
windump: listening on \Device\NPF_{F8C4038F-1462-4A50-BD0C 917801DBFC75}
19:57:01.956298 IP TCP4Sec.1069 > 63-123-56-242.static.web.com.110: F
1009274661:10092 74661(0) ack 4089055151 win 16560 (DF)
19:57:02.842834 IP TCP4Sec.1103 > dnspool1.skynet.be.53:  229+ PTR?
242.56.123.63.in-addr.arpa. (44)

13 packets received by filter
0 packets dropped by kernel
```

Reading TCPdump/WinDump Output

Ok, so far we have learned how to grab the data off the wire now lets learn how to read the output! TCPdump has a specific format and you will have to learn how to read it. But don't worry, it's pretty easy to understand. Each protocol has its own standard output. Since this lesson is about TCP, we will cover the TCP output format.

Here is an example record:

```
20:08:41.313149 rootwars.org.1086 > 66.102.9.104.80: S
1192278531:1192278531(0) win 1638
```

- **20:08:41.313149** This is the time stamp in the format of two digits for hours, two digits for minutes, two digits for seconds, and six digits for fractional parts of a second.
- **rootwars.org** This is the source host name. The default behavior is to resolve the hostname but you can turn it off with the **TCPdump –n** option. If you don't see a DNS name the IP will appear. Something like **IP COMPUTERNAME.**
- **1086** This is the source port number or port service.
- > This is a marker to indicate direction flow going from source to destination.
- **66.102.9.104** This is the desintation host name or IP address.
- **80** This is the desination port number or maybe it will be translated ad HTTP.
- **S** This is the TCP Flag. The S represents a SYN Flag (see the next section).
- **1192278531:1192278531(0)** This is the beginning TCP sequence number: ending TCP sequence number (data bytes). Sequence nubers are used by TCP to order the data received. The initial sequence number (ISN) is selected as a unique number to mark the first byte of data. The ending sequence number is the beginning sequence plus the number of bytes being sent with this TCP segment. In this case there were zero bytes sent, the beginning and ending sequence numbers are the same.
- **win 1638** This is the receiving buffer size in bytes of rootwars.org for this connection.

TCP Flags in TCPdump/WinDump

TCP Flag	**Flag Representation**	**Flag Meaning**
SYN	S	Session establishment request which is the first part of any TCP connection (3 way handshake).
ACK	ack	Ack flag is generally used to acknowledge the receipt of data from the sender. Might be in conjunction with other flags.
FIN	F	Fin flag is generally used to indicate the sender's

		intention to gracefully terminate the sending host’s connection to the receiving host.
RESET	R	Reset flag is generally used to indicate the sender’s intention to immediately abort the existing connection wit the receiving host.
PUSH	P	Push flag is generally used to immediately “push” data from the sending host to the receiving host. This is for applications like telnet where response time is a primary concern.
URGENT	urg	Urgent flag is generally used to mean that there is “urgent” data that takes precedence over other data.
Placeholder	.	If the connections does not have a SYN, FIN, RESET, or PUSH flag set, a placeholder (a period: .) will be found after the destination port

Absolute & Relative Sequence Numbers

Sequence numbers are associated only with TCP output. TCP sequence numbers are used by the destination host to reassemble TCP traffic that arrives. Remember that TCP guarantees order where UDP does not. If TCP gets a sequence number out of order, it knows to resend the packet. These sequence numbers are decimal representations of the 32 bit field. This could get a little confusing to read so TCPdump helps us out by converting the absolute sequence number (the decimal representation of the long number) to a relative sequence numbers after the two hosts exchange their Initial Sequence Numbers (ISN).

Lets look at an example (timestamps have been removed)

```
C:\Documents and Settings\NoOne\Desktop\HackerCLI>windump -i 1 tcp
windump: listening on \Device\NPF_{F8C4038F-1462-4A50-BD0C-917801DBFC75}
TCP4SEC.2229 > 66-193-231-87.dimenoc.com.80: S 781505672:781505672(0) win
16384 <mss 1460> (DF)
```

```
IP 66-193-231-87.dimenoc.com.80 > TCP4SEC.2229: S 4106656209:4106656209(0)
ack 781505673 win 5840 <mss 1402> (DF)
IP TCP4SEC.2229 > 66-193-231-87.dimenoc.com.80: . ack 1 win 16824 (DF)
IP TCP4SEC.2229 > 66-193-231-87.dimenoc.com.80: P 1:415(414) ack 1 win
16824 (DF)
```

Ok let's explain this mess above and hopefully understand absolute and relative sequence numbers when we are done. The first two large numbers in bold represent the large absolute ISNs (**781505672:781505672** & **4106656209:4106656209**). It goes from client to server, then server back to client. We can also see the server's ack of the client's ISN (the 781505673 in bold). The third line has a bold 1, this is the first relative sequence number that TCPdump has gracefully inserted for us. This means that the client has acknowledged the previous SYN from the server with an ISN of 4106656209. The 1 acknowledgement means that the next expected relative byte to be received by the client is byte 1 or 4106656210 in absolute sequence numbers. The final line has the numbers 1 and 415 in bold to indicate that relative to the absolute sequence number of 781505672, the 1st byte through but not including the 415th byte are sent from client to server for a total of 414 bytes.

While this might not explain exactly what absolute and ISN sequence numbers are. It did hopefully explain just what the heck the funny 1:415(414) stuff means. IF you would prefer to keep the sequence numbers in absolute form use the **TCPdump –S** option.

If you are still confused on absolute sequence numbers, google.com is your friend!

Dumping TCPdump/WinDump output in hexadecimal format

TCPdump does not display all the fields of the captured data. Some fields are not available by default. If you want to see these fields, like the IP header that stores the length of the IP Header, you'll have to dump the output in Hexadecimal. The **TCPdump –x** option will dump the entire datagram with the default snaplen in hex. To interpret the TCPdump hex output you will need some extra studying and resources I will not be covering. The easiest thing to do would be to open the binary files with ethereal. Don't forget that you might need to change your snaplen to get that data you are trying to see with Ethereal as well as using the –w option to write it to a binary file.

Here's an example:

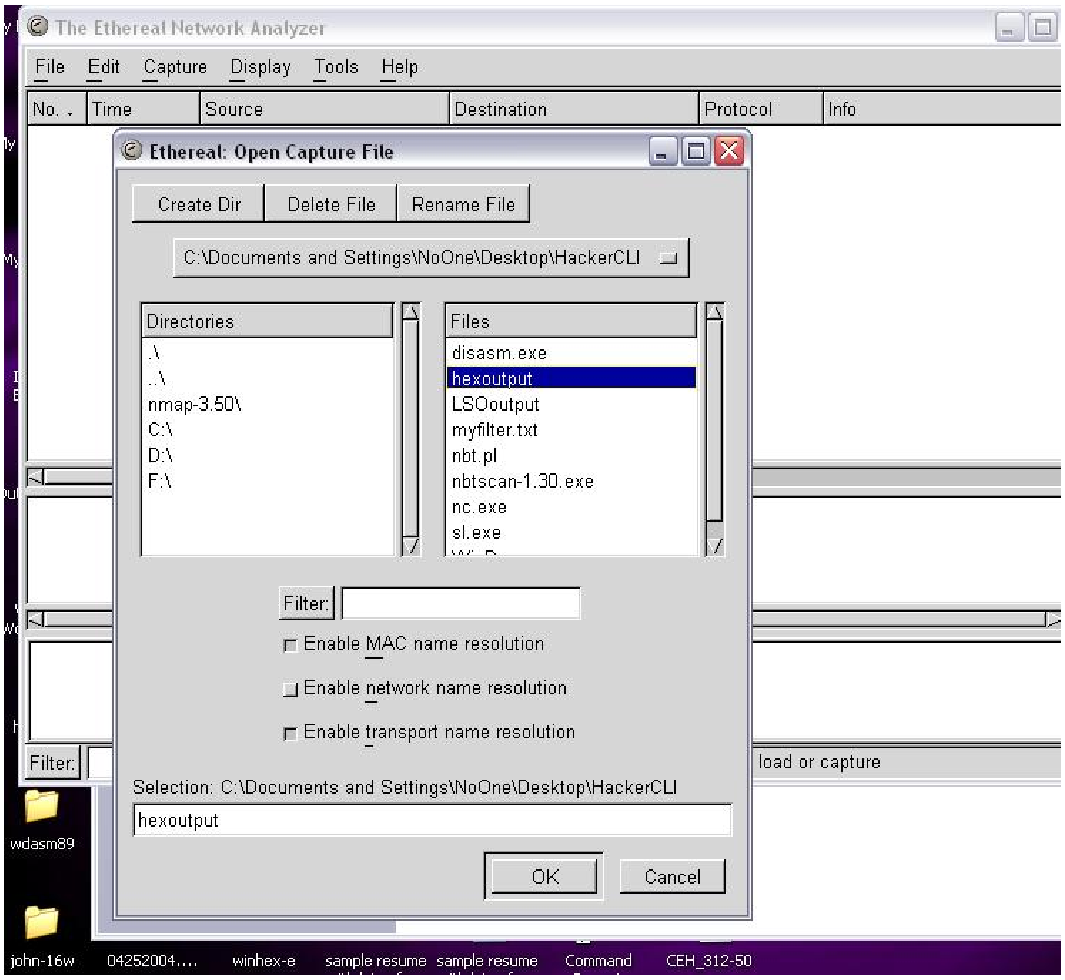
```
C:\Documents and Settings\NoOne\Desktop\HackerCLI>windump -i 1 -x
windump: listening on \Device\NPF_{F8C4038F-1462-4A50-BD0C-917801DBFC75}
18:43:52.867083 IP TCP4SEC.2145 > pop.noware.net.110: F
263898465:263898465(0) ack 5 13332375 win 16795 (DF)
                         4500 0028 5bdd 40EE 8006 b35c c0a8 0164
                         2671 0319 0861 006e 0fba c561 1e98 d497
                         5A41 419b b187 0000
18:43:53.682818 IP TCP4SEC.1038 > dnspool1.bignet.us.53:  111+ PTR?
25.3.113.38.in-addr.arpa. (42)
                         4500 0046 5bde 0000 8011 56b9 c0a8 0164
                         c3ee 0215 040e 0035 0032 5cd9 006f 0100
                         0001 0000 0000 70C4 0232 3501 3303 3131
                         3302 3338 0769 6e2d 6164 6472 EEA7 7270
```

```
6100 000c 0001 3338
```

And dumping the output to file and opening with Ethereal

```
C:\Documents and Settings\NoOne\Desktop\HackerCLI>windump -i 1 -x -w
hexoutput
windump: listening on \Device\NPF_{F8C4038F-1462-4A50-BD0C-917801DBFC75}
```

This dumps the output to a binary file, now let’s load it into Ethereal.

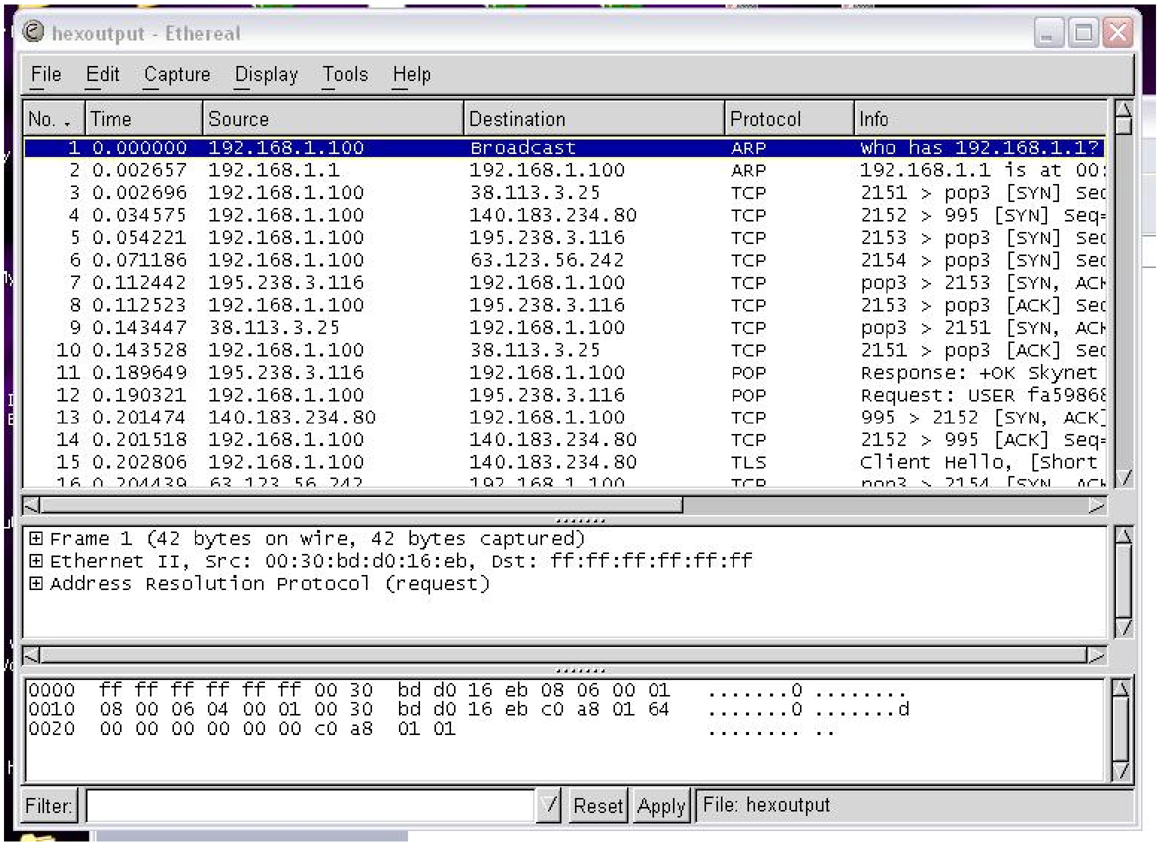

The Ethereal Network Analyzer
File Edit Capture Display Tools Help
No. Time Source Destination Protocol Info
Ethereal: Open Capture File
Create Dir Delete File Rename File
Directories
Files
Filter:
OK Cancel
load or capture
wdasm89
john-16w
04252004....
winhex-e
sample resume
sample resume
Command
CEH_312-50

C:\Documents and Settings\NoOne\Desktop\HackerCLI

Directories	Files
.\	disasm.exe
..\	hexoutput
nmap-3.50\	LSOoutput
C:\	myfilter.txt
D:\	nbt.pl
F:\	nbtscan-1.30.exe
	nc.exe
	sl.exe

Enable MAC name resolution
Enable network name resolution
Enable transport name resolution

Selection: C:\Documents and Settings\NoOne\Desktop\HackerCLI
hexoutput



hexoutput - Ethereal
File Edit Capture Display Tools Help
Filter: Reset Apply File: hexoutput

```
No. . Time     Source          Destination     Protocol Info
  1 0.000000 192.168.1.100   Broadcast       ARP      Who has 192.168.1.1?
  2 0.002657 192.168.1.1     192.168.1.100   ARP      192.168.1.1 is at 00:
  3 0.002696 192.168.1.100   38.113.3.25     TCP      2151 > pop3 [SYN] Seq
  4 0.034575 192.168.1.100   140.183.234.80  TCP      2152 > 995 [SYN] Seq=
  5 0.054221 192.168.1.100   195.238.3.116   TCP      2153 > pop3 [SYN] Seq
  6 0.071186 192.168.1.100   63.123.56.242   TCP      2154 > pop3 [SYN] Seq
  7 0.112442 195.238.3.116   192.168.1.100   TCP      pop3 > 2153 [SYN, ACK
  8 0.112523 192.168.1.100   195.238.3.116   TCP      2153 > pop3 [ACK] Seq
  9 0.143447 38.113.3.25     192.168.1.100   TCP      pop3 > 2151 [SYN, ACK
 10 0.143528 192.168.1.100   38.113.3.25     TCP      2151 > pop3 [ACK] Seq
 11 0.189649 195.238.3.116   192.168.1.100   POP      Response: +OK Skynet
 12 0.190321 192.168.1.100   195.238.3.116   POP      Request: USER fa59868
 13 0.201474 140.183.234.80  192.168.1.100   TCP      995 > 2152 [SYN, ACK]
 14 0.201518 192.168.1.100   140.183.234.80  TCP      2152 > 995 [ACK] Seq=
 15 0.202806 192.168.1.100   140.183.234.80  TLS      Client Hello, [Short
 16 0.204439 63.123.56.242   192.168.1.100   TCP      pop3 > 2154 [SYN, ACK

⊞ Frame 1 (42 bytes on wire, 42 bytes captured)
⊞ Ethernet II, Src: 00:30:bd:d0:16:eb, Dst: ff:ff:ff:ff:ff:ff
⊞ Address Resolution Protocol (request)

0000  ff ff ff ff ff ff 00 30  bd d0 16 eb 08 06 00 01   .......0 ........
0010  08 00 06 04 00 01 00 30  bd d0 16 eb c0 a8 01 64   .......0 .......d
0020  00 00 00 00 00 00 c0 a8  01 01                     ........ ..
```

Now we can see everything!

The TCPdump MAN page

WinDump - dump traffic on a network

SYNOPSIS

WinDump [**-aBdDeflnNOpqRStvxX**] [**-c** *count*] [**-F** *file*]

[**-i** *interface*] [**-m** *module*] [**-r** *file*]

[**-s** *snaplen*] [**-T** *type*] [**-w** *file*]

[**-E** *algo:secret*] [*expression*]

DESCRIPTION

TCPdump prints out the headers of packets on a network interface that match the boolean *expression*.

Under SunOS with nit or bpf: To run *TCPdump* you must have read access to */dev/nit* or */dev/bpf**. **Under Solaris with dlpi:** You must have read/write access to the network pseudo device, e.g. */dev/le*. **Under HP-UX with dlpi:** You must be root or it must be installed setuid to root. **Under IRIX with snoop:** You must be root or it must be installed setuid to root. **Under Linux:** You must be root or it must be installed setuid to root. **Under Ultrix and Digital UNIX:** Once the super-user has enabled promiscuous-mode operation using *pfconfig*(8), any user may run **TCPdump**. **Under BSD:** You must have read access to */dev/bpf**. **Under Win32:** You must have installed *WinPcap*.

OPTIONS

-a Attempt to convert network and broadcast addresses to names.

-c Exit after receiving *count* 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 *algo:secret* for decrypting IPsec ESP packets. Algorithms may be **des-cbc**, **3des-cbc**, **blowfish-cbc**, **rc3-cbc**, **cast128-cbc**, or **none**. The default is **des-cbc**. The ability to decrypt packets is only present if *TCPdump* was compiled with cryptography enabled. *secret* the ascii text for ESP secret key. We cannot take arbitrary binary value at this moment. The option assumes RFC2406 ESP, not RFC1827 ESP. The option is only for debugging purposes, and the use of this option with truly `secret' key is discouraged. By presenting IPsec secret key onto command line you make it visible to others, via *ps*(1) and other occasions.

-f Print `foreign' internet addresses numerically rather than symbolically (this option is intended to get around serious brain damage in Sun's yp server --- usually it hangs forever translating non-local internet numbers).

-F Use *file* as input for the filter expression. An additional expression given on the command line is ignored.

-i Listen on *interface*. If unspecified, *TCPdump* searches the system interface list for the lowest numbered, configured up interface (excluding loopback). Ties are broken by choosing the earliest match. In Windows *interface* can be the name of the adapter, or its number (the one reported by the -D flag).

On Linux systems with 2.2 or later kernels, an *interface* argument of ``any'' can be used to capture packets from all interfaces. Note that captures on the ``any'' device will not be done in promiscuous mode.

-l Make stdout line buffered. Useful if you want to see the data while capturing it. E.g., ``TCPdump -l | tee dat'' or ``TCPdump -l > dat & tail -f dat''.

-n Don't convert addresses (i.e., host addresses, port numbers, etc.) to names.

-N Don't print domain name qualification of host names. E.g., if you give this flag then *TCPdump* will print ``nic'' instead of ``nic.ddn.mil''.

-m Load SMI MIB module definitions from file *module*. This option can be used several times to load several MIB modules into *TCPdump*.

-O Do not run the packet-matching code optimizer. This is useful only if you suspect a bug in the optimizer.

-p *Don't* put the interface into promiscuous mode. Note that the interface might be in promiscuous mode for some other reason; hence, `-p' cannot be used as an abbreviation for `ether host {local-hw-addr} or ether broadcast'.

-q Quick (quiet?) output. Print less protocol information so output lines are shorter.

-r Read packets from *file* (which was created with the -w option). Standard input is used if *file* is ``-''.

-s Snarf *snaplen* bytes of data from each packet rather than the default of 68 (with SunOS's NIT, the minimum is actually 96). 68 bytes is adequate for IP, ICMP, TCP and UDP but may truncate protocol information from name server and NFS packets (see below). Packets truncated because of a limited snapshot are indicated in the output with ``[|*proto*]'', where *proto* is the name of the protocol level at which the truncation has occurred. Note that taking larger snapshots both increases the amount of time it takes to process packets and, effectively, decreases the amount of packet buffering. This may cause packets to be lost. You should limit *snaplen* to the smallest number that will capture the protocol information you're interested in. Setting *snaplen* to 0 means use the required length to catch whole packets.

-T Force packets selected by "*expression*" to be interpreted the specified *type*. Currently known types are **cnfp** (Cisco NetFlow protocol), **rpc** (Remote Procedure Call), **rtp** (Real-Time Applications protocol), **rtcp** (Real-Time Applications control protocol), **snmp** (Simple Network Management Protocol), **vat** (Visual Audio Tool), and **wb** (distributed White Board).

-R Assume ESP/AH packets to be based on old specification (RFC1825 to RFC1829). If specified, *TCPdump* will not print replay prevention field. Since there is no protocol version field in ESP/AH specification, *TCPdump* cannot deduce the version of ESP/AH protocol.

-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.

-v (Slightly more) verbose output. For example, the time to live, identification, total length and options in an IP packet are printed. Also enables additional packet integrity checks such as verifying the IP and ICMP header checksum.

-vv Even more verbose output. For example, additional fields are printed from NFS reply packets.

-vvv Even more verbose output. For example, telnet **SB** ... **SE** options are printed in full. With **-X** telnet options are printed in hex as well.

-w Write the raw packets to *file* rather than parsing and printing them out. They can later be printed with the -r option. Standard output is used if *file* is ``-''.

-x Print each packet (minus its link level header) in hex. The smaller of the entire packet or *snaplen* bytes will be printed.

-X When printing hex, print ascii too. Thus if **-x** is also set, the packet is printed in hex/ascii. This is very handy for analysing new protocols. Even if **-x** is not also set, some parts of some packets may be printed in hex/ascii.

Win32 specific extensions

-B	Set driver's buffer size to *size* in **KiloBytes**. The default buffer size is 1 megabyte (i.e 1000). If there is any loss of packets during the capture, the suggestion is to increase the kernel buffer size by means of this switch, since the dimension of the driver's buffer influences heavily the capture performance.

-D	Print the list of the interface cards available on the system. For every network adapter, this switch returns the *number*, the *name* and the description. The user can start the capture on a specific adapter typing 'WinDump –i *name*' or 'WinDump –i *number*'. If the machine has more than one network adapter, WinDump without parameters starts on the first network interface available on the system.
