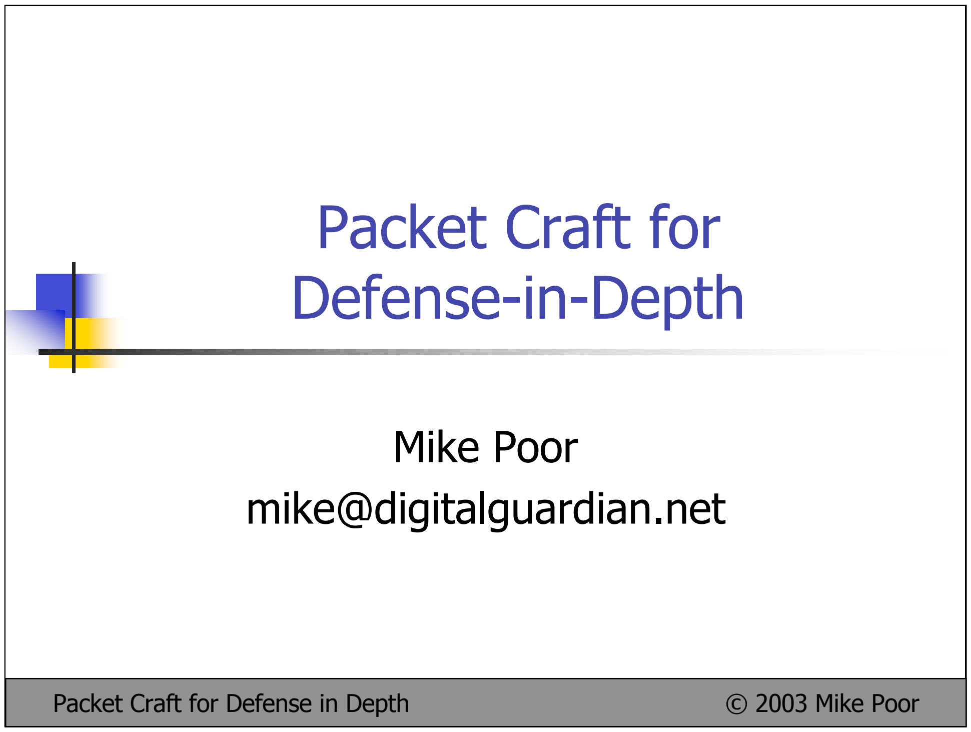


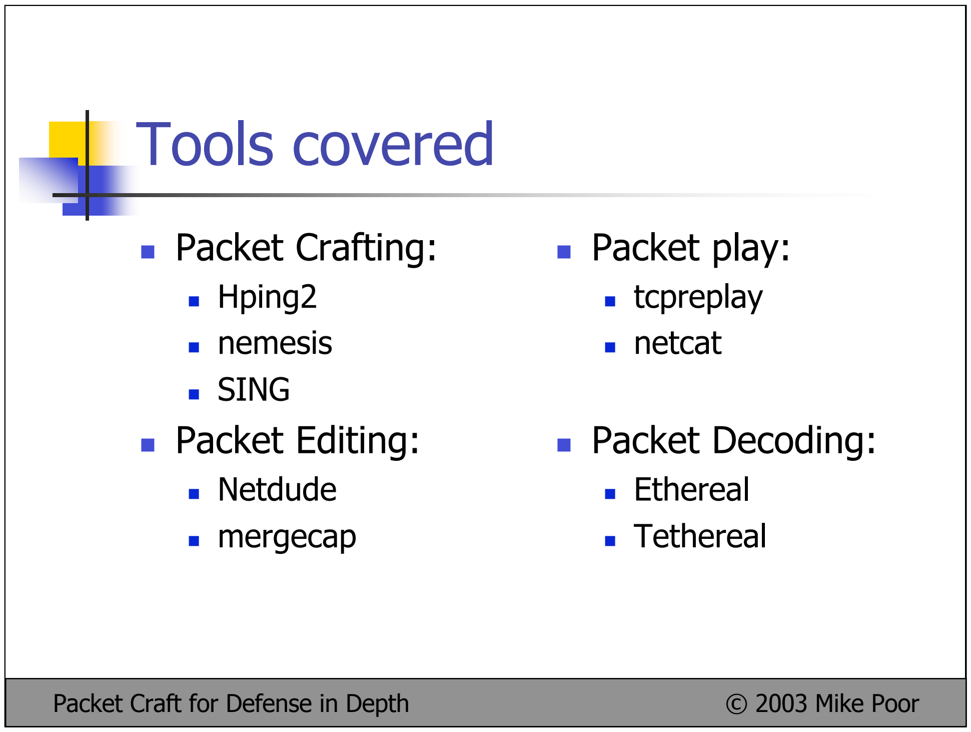
Packet Craft for Defense-in-Depth

Mike Poor
mike@digitalguardian.net

Packet Craft for Defense in Depth

Topics Covered

- Introduction to Packet Craft
- Packet Craft for testing IDS/Firewall rules
- Replaying Packet captures
- Editing Packets / Traffic
- Packet Craft for good and evil

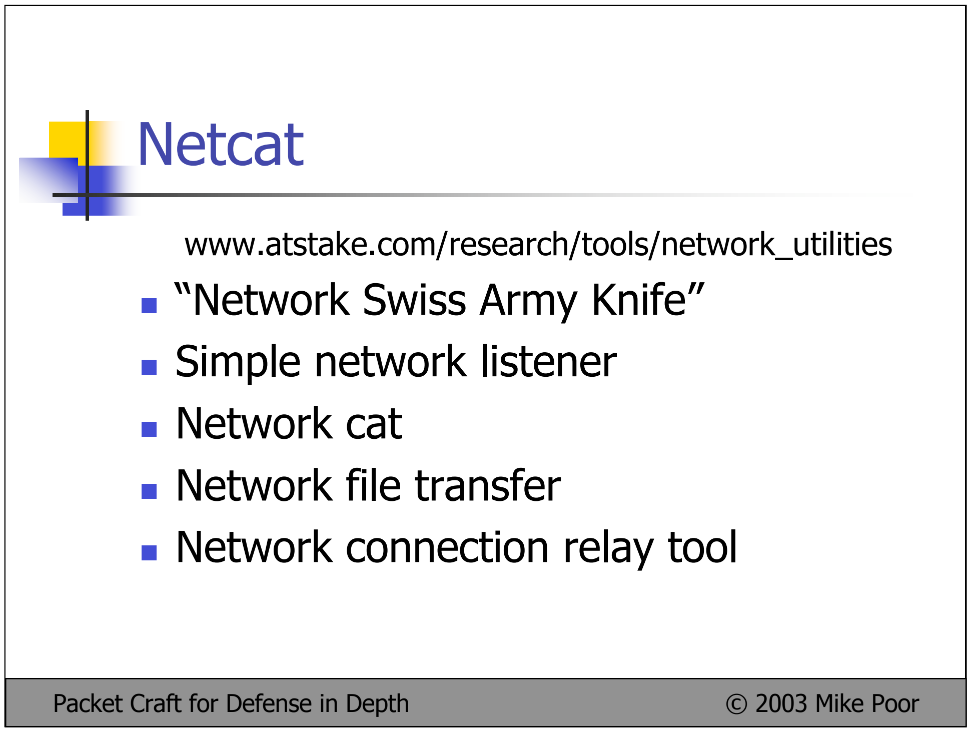


Tools covered

- Packet Crafting:
  - Hping2
  - nemesis
  - SING
- Packet Editing:
  - Netdude
  - mergecap
- Packet play:
  - tcpreplay
  - netcat
- Packet Decoding:
  - Ethereal
  - Tethereal

Packet Craft for Defense in Depth © 2003 Mike Poor

As we can see, all the tools covered in this presentation are designed for the Unix operating system. Some of these tools, like netcat, ethereal, and ngrep are available for the Windows platform, but their use under windows and other Windows based tools is beyond the scope of this presentation.



Netcat

www.atstake.com/research/tools/network_utilities

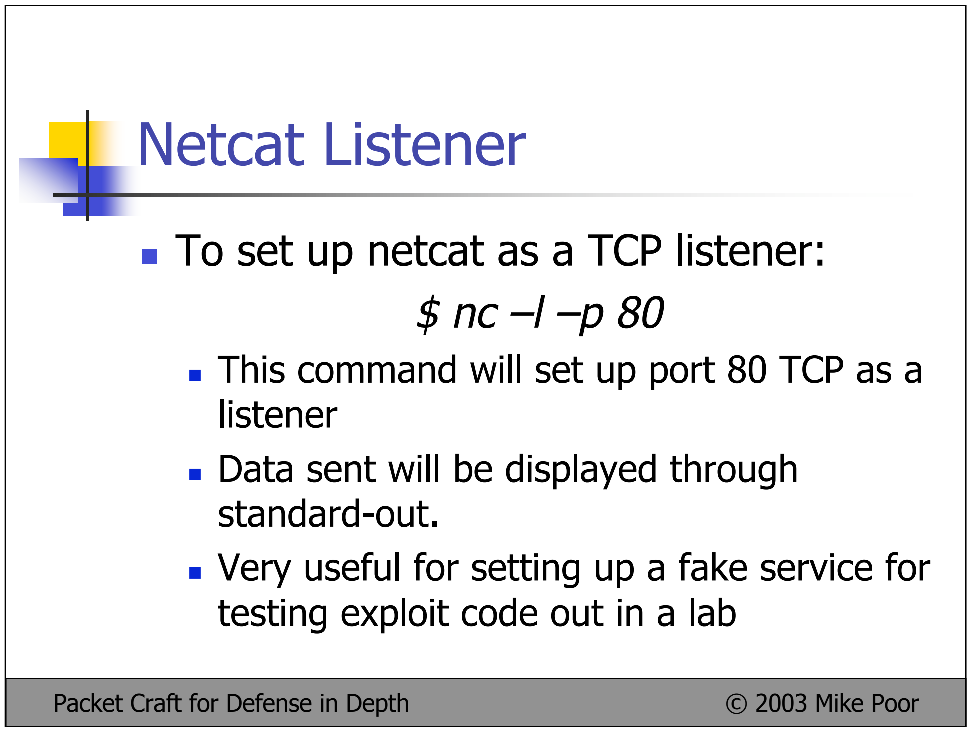
- "Network Swiss Army Knife"
- Simple network listener
- Network cat
- Network file transfer
- Network connection relay tool

Packet Craft for Defense in Depth © 2003 Mike Poor

Netcat is the ultimate handy networking tool to have on your labtop. Useful for all sorts of things, we will demonstrate some of the basics. When you are ready for ultimate netcat functionality, you can try your hand at the netcat webserver ☺

Netcat installs easily on all Unices that Ive tried it on. There is also a netcat.exe for the Windows platform.

See packages available at: www.atstake.com/research/tools/network_utilities and through the distribution of your OS.



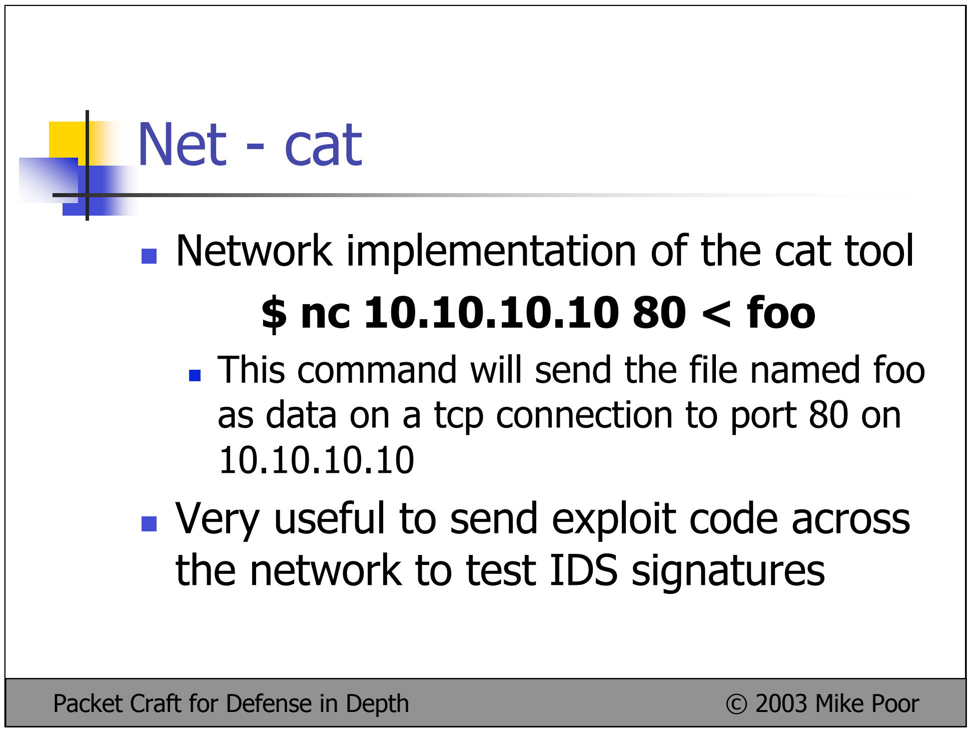
Netcat Listener

- To set up netcat as a TCP listener:

  $ nc –l –p 80

  - This command will set up port 80 TCP as a listener
  - Data sent will be displayed through standard-out.
  - Very useful for setting up a fake service for testing exploit code out in a lab

Packet Craft for Defense in Depth © 2003 Mike Poor



Net - cat

- Network implementation of the cat tool

  $ nc 10.10.10.10 80 < foo

  - This command will send the file named foo as data on a tcp connection to port 80 on 10.10.10.10
- Very useful to send exploit code across the network to test IDS signatures

Packet Craft for Defense in Depth © 2003 Mike Poor

Netcat client:

```
$ echo "this is a test" > foo
$ netcat 127.0.0.1 2000 < foo
```

Netcat listener:

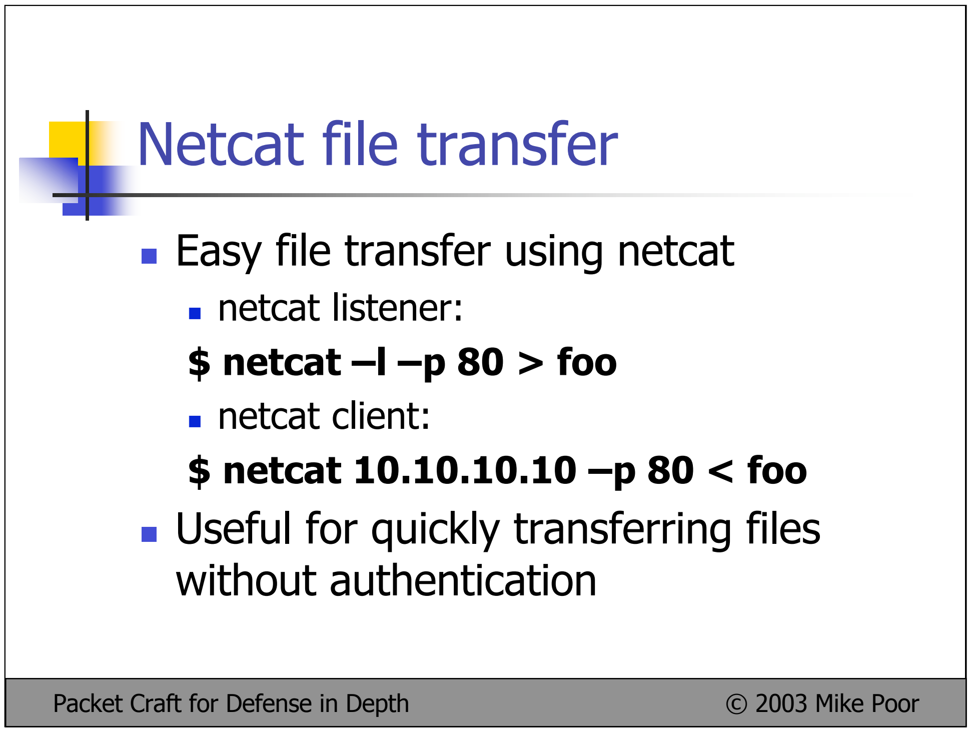
```
$ netcat -l -p 2000
this is a test
```

This example shows how we can transfer a simple text file across the network to standard-out. If we wanted to save the file off to a file, all we would have to do is execute the following command on the listener:

```
$netcat –l –p 2000 > foo
```

Netcat would take everything received on port 2000 and place it in a file named “foo”.

This becomes very useful to the IDS analyst, in order to test ids rules.



Netcat file transfer

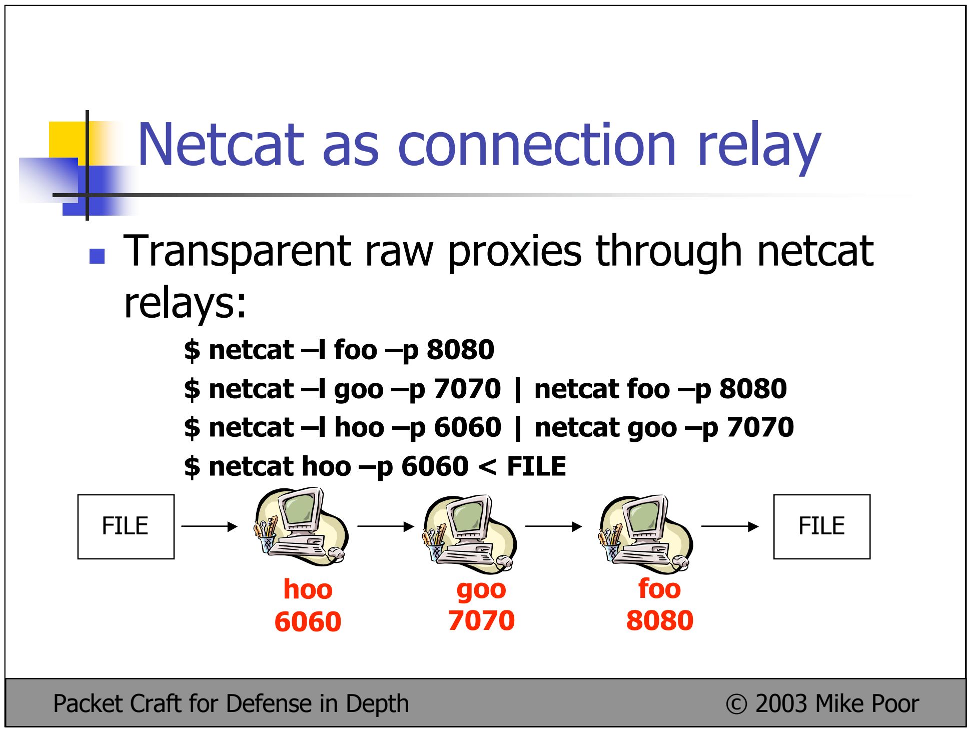
- Easy file transfer using netcat
  - netcat listener:

    $ netcat –l –p 80 > foo
  - netcat client:

    $ netcat 10.10.10.10 –p 80 < foo
- Useful for quickly transferring files without authentication

Packet Craft for Defense in Depth © 2003 Mike Poor

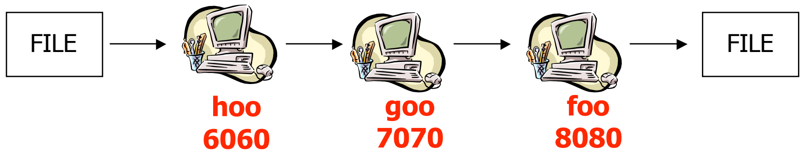
Netcat can be used to set up a simple one time file tranfer between two hosts. There is no authentication, so no need to track usernames and passwords. Bear in mind that any host that connects to the TCP port you set up to cat the file through, will receive the file after the three way handshake is complete.



Netcat as connection relay

- Transparent raw proxies through netcat relays:

```
$ netcat –l foo –p 8080
$ netcat –l goo –p 7070 | netcat foo –p 8080
$ netcat –l hoo –p 6060 | netcat goo –p 7070
$ netcat hoo –p 6060 < FILE
```



Packet Craft for Defense in Depth © 2003 Mike Poor

Here we have four machines: ioo, hoo, goo, and foo.

Our objective here is to proxy a file from ioo to foo, through hoo and goo (ouch, that was painful).

We must set these relays up in backwards order, from receiver to sender.

We start with setting up the receiving host, foo:

```
netcat –l foo –p 8080
```

Then we set up our first relay on goo:

```
netcat –l –p 7070 | netcat foo –p 8080
```

All TCP traffic going to port 7070 on goo will now be relayed to port 8080 on foo.

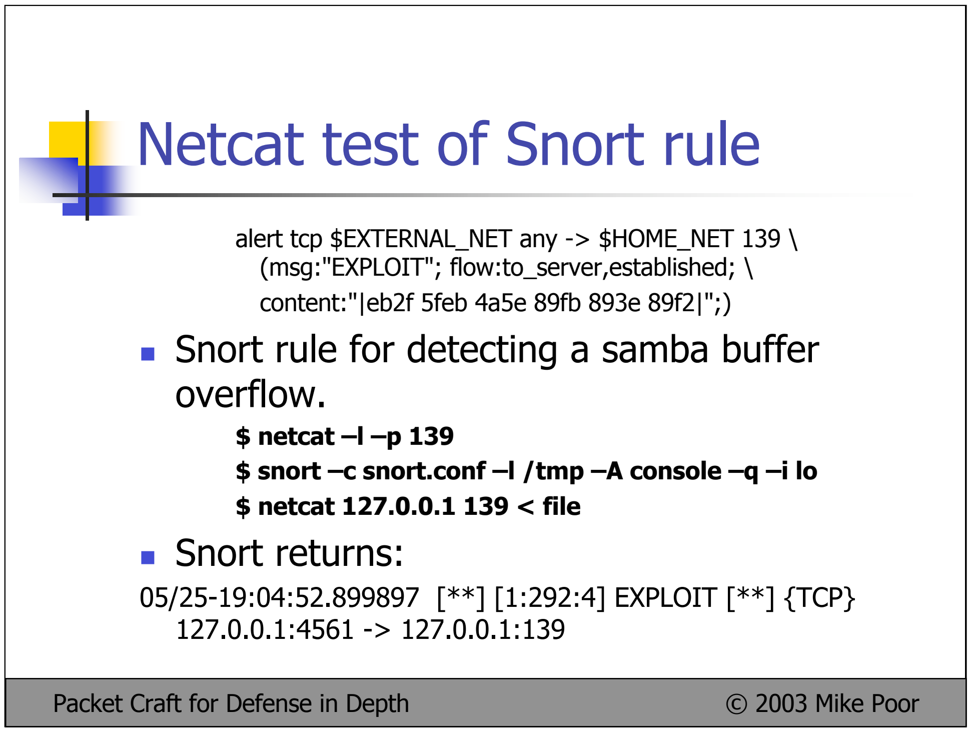
Continuing with our relay on hoo:

```
netcat –l –p 6060 | netcat goo –p 7070
```

Now everything going to port 6060 will be forwarded through goo on port 7070, to foo on port 8080.

The finale shows us cat'ing a file through our transparent relays to foo.

```
netcat hoo –p 6060 < FILE
```



Netcat test of Snort rule

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 139 \
  (msg:"EXPLOIT"; flow:to_server,established; \
  content:"|eb2f 5feb 4a5e 89fb 893e 89f2|";)
```

- Snort rule for detecting a samba buffer overflow.

```
$ netcat –l –p 139
$ snort –c snort.conf –l /tmp –A console –q –i lo
$ netcat 127.0.0.1 139 < file
```

- Snort returns:

```
05/25-19:04:52.899897  [**] [1:292:4] EXPLOIT [**] {TCP}
   127.0.0.1:4561 -> 127.0.0.1:139
```

Packet Craft for Defense in Depth © 2003 Mike Poor

We write a new rule for a linux samba overflow. If we wanted to test this rule out to see if snort will fire, we can do two things. First, we could fire the original exploit across the wire. This works well if we are in an isolated lab, and have a copy of the latest exploit. Many times this does not work that way. Security experts analyze the vulnerability and come up with proof of concept code that they do not often share. The may however share the offsets and raw bytes of a portion of the exploit in action. We can take those raw bytes and write a signature, then test it with a hex editor, netcat and snort.

First, we use a hex editor to create a file with raw bytes matching exploit signature. Once this is in place, we set up a netcat listener on port 138 TCP.

```
$ netcat –l –p 139
```

Then we fire up snort:

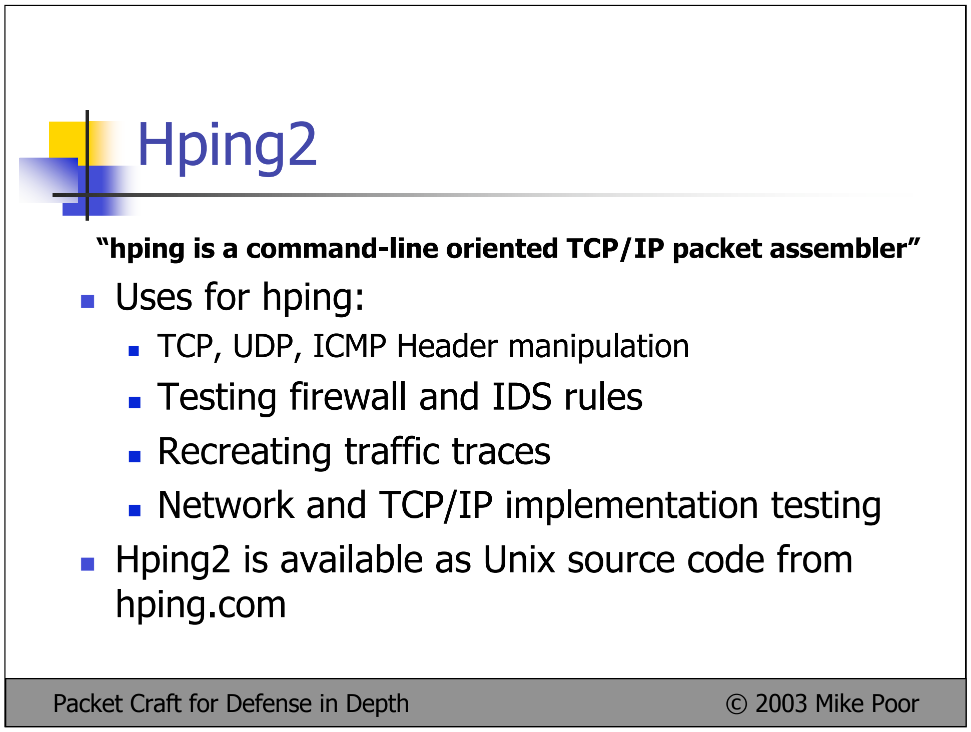
```
$ snort –c snort.conf –l /tmp –A console –q
```

We send the raw byte file to the open port:

```
$ netcat 127.0.0.1 139
```

Sure enough snort matches the following rule, and triggers an alert:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"EXPLOIT
x86 linux samba overflow"; flow:to_server,established; content:"|eb2f
5feb 4a5e 89fb 893e 89f2|"; reference:bugtraq,1816;
```



Hping2

"hping is a command-line oriented TCP/IP packet assembler"

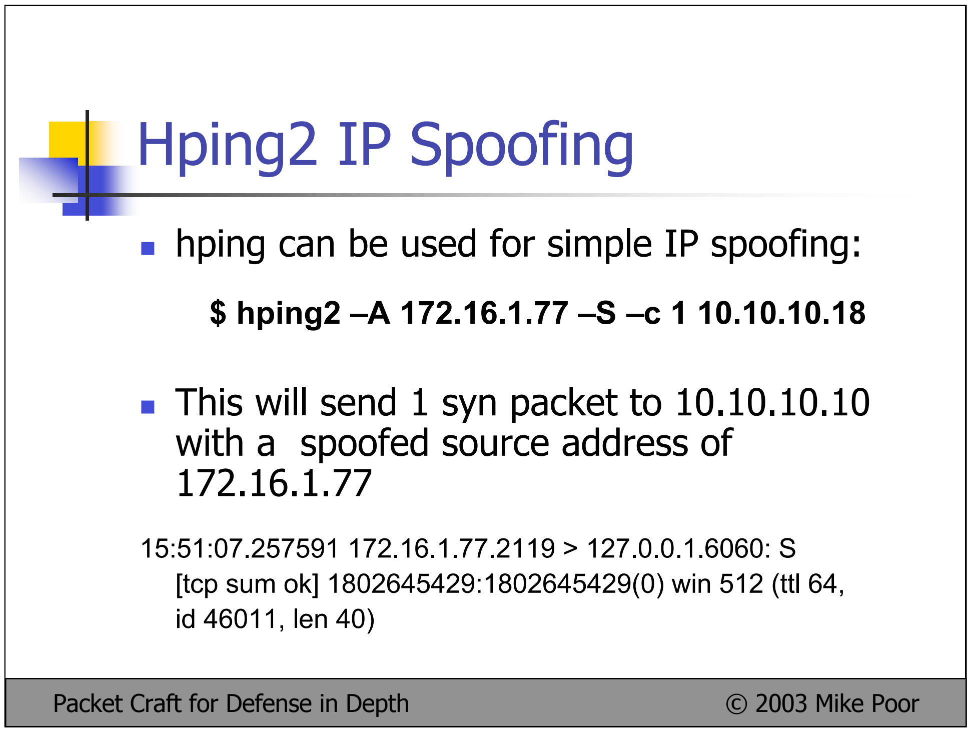
- Uses for hping:
  - TCP, UDP, ICMP Header manipulation
  - Testing firewall and IDS rules
  - Recreating traffic traces
  - Network and TCP/IP implementation testing
- Hping2 is available as Unix source code from hping.com

Packet Craft for Defense in Depth © 2003 Mike Poor

Hping is one of my favorite packet crafting tools. It is powerful, simple, and compiles cleanly. You also have to work hard to use it for evil, which in the end is a good thing. I use hping2 often when I need to test out IDS rules, or manually test a hosts responses to anomalous traffic.

From the simple task of sending a packet with a spoofed source address, to writing a complete scripted test, hping2 rises to the occasion. On a downside, hping2 does not give us much options in the payload department.

Installing hping2 is as easy as: $ ./configure && make && make install



Hping2 IP Spoofing

- hping can be used for simple IP spoofing:

  $ hping2 –A 172.16.1.77 –S –c 1 10.10.10.18

- This will send 1 syn packet to 10.10.10.10 with a spoofed source address of 172.16.1.77

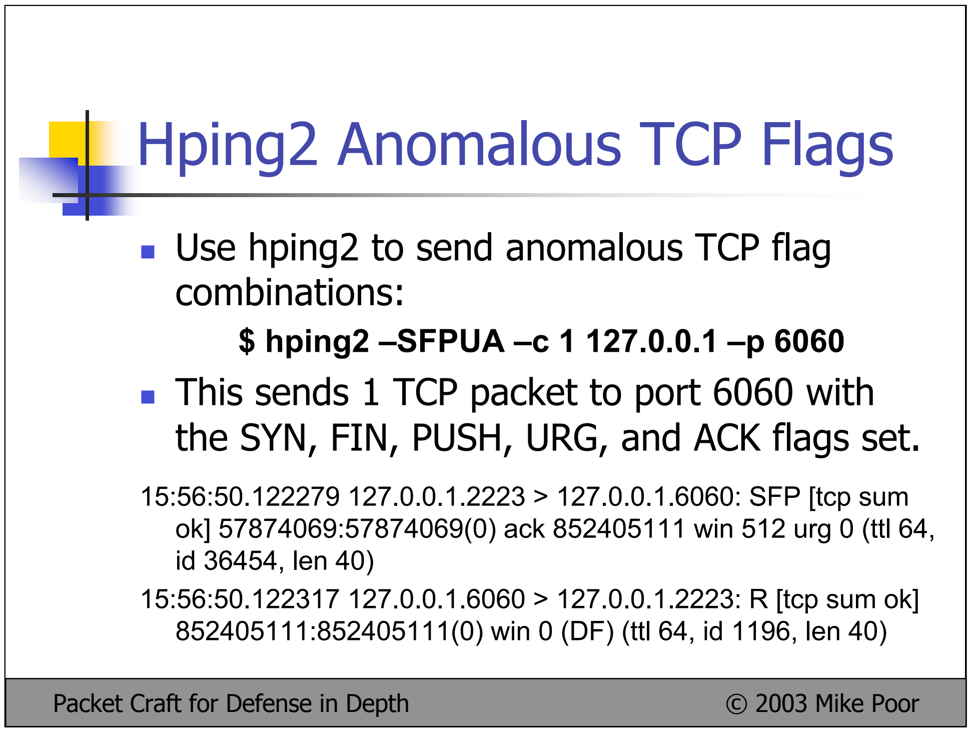
```
15:51:07.257591 172.16.1.77.2119 > 127.0.0.1.6060: S
    [tcp sum ok] 1802645429:1802645429(0) win 512 (ttl 64,
    id 46011, len 40)
```

Packet Craft for Defense in Depth © 2003 Mike Poor

In the bolded segment we see that hping is indeed spoofing the source address to be: 172.16.1.77:

```
15:48:44.273428 172.16.1.77.1041 > 127.0.0.1.6060: S [tcp sum ok]
1645313341:1645313341(0) win 512 (ttl 64, id 43574, len 40)
0x0000   4500 0028 aa36 0000 4006 a43b ac10 014d      E..(.6..@..;...M
0x0010   7f00 0001 0411 17ac 6211 7d3d 26a1 1e9a      ........b.}=&...
0x0020   5002 0200 413d 0000                          P...A=..
```

ac10014d in hex translates to 172.16.1.77



Hping2 Anomalous TCP Flags

- Use hping2 to send anomalous TCP flag combinations:

$ hping2 –SFPUA –c 1 127.0.0.1 –p 6060

- This sends 1 TCP packet to port 6060 with the SYN, FIN, PUSH, URG, and ACK flags set.

```
15:56:50.122279 127.0.0.1.2223 > 127.0.0.1.6060: SFP [tcp sum
   ok] 57874069:57874069(0) ack 852405111 win 512 urg 0 (ttl 64,
   id 36454, len 40)
15:56:50.122317 127.0.0.1.6060 > 127.0.0.1.2223: R [tcp sum ok]
   852405111:852405111(0) win 0 (DF) (ttl 64, id 1196, len 40)
```

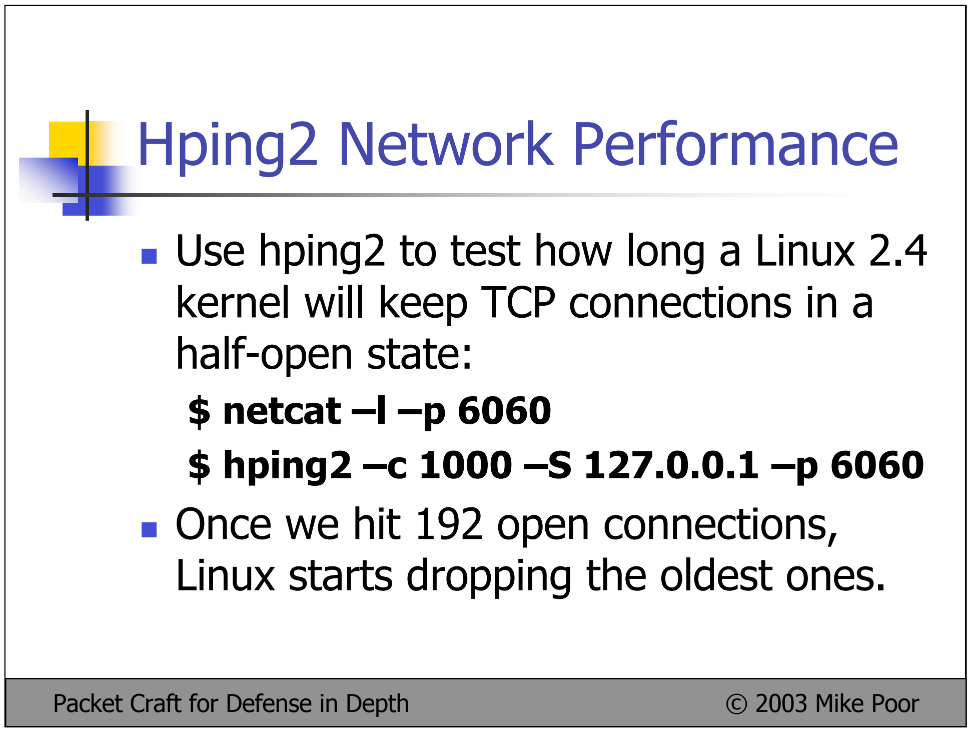
Packet Craft for Defense in Depth © 2003 Mike Poor

Issuing the command:

$ hping2 –SFPUA –c 1 127.0.0.1 –p 6060

We can see that hping2 is sending a packet with the syn, fin, push, urg, and ack flags set. My Linux machines promptly replies with a RESET flag. Note that while my machine reset the connection, which you would expect, we have now elicited a response.

```
15:56:50.122279 127.0.0.1.2223 > 127.0.0.1.6060: SFP [tcp sum ok] 57874069:57874069(0)
ack 852405111 win 512 urg 0 (ttl 64, id 36454, len 40)
15:56:50.122317 127.0.0.1.6060 > 127.0.0.1.2223: R [tcp sum ok] 852405111:852405111(0)
win 0 (DF) (ttl 64, id 1196, len 40)
```



Hping2 Network Performance

- Use hping2 to test how long a Linux 2.4 kernel will keep TCP connections in a half-open state:

```
$ netcat –l –p 6060
$ hping2 –c 1000 –S 127.0.0.1 –p 6060
```

- Once we hit 192 open connections, Linux starts dropping the oldest ones.

Packet Craft for Defense in Depth © 2003 Mike Poor

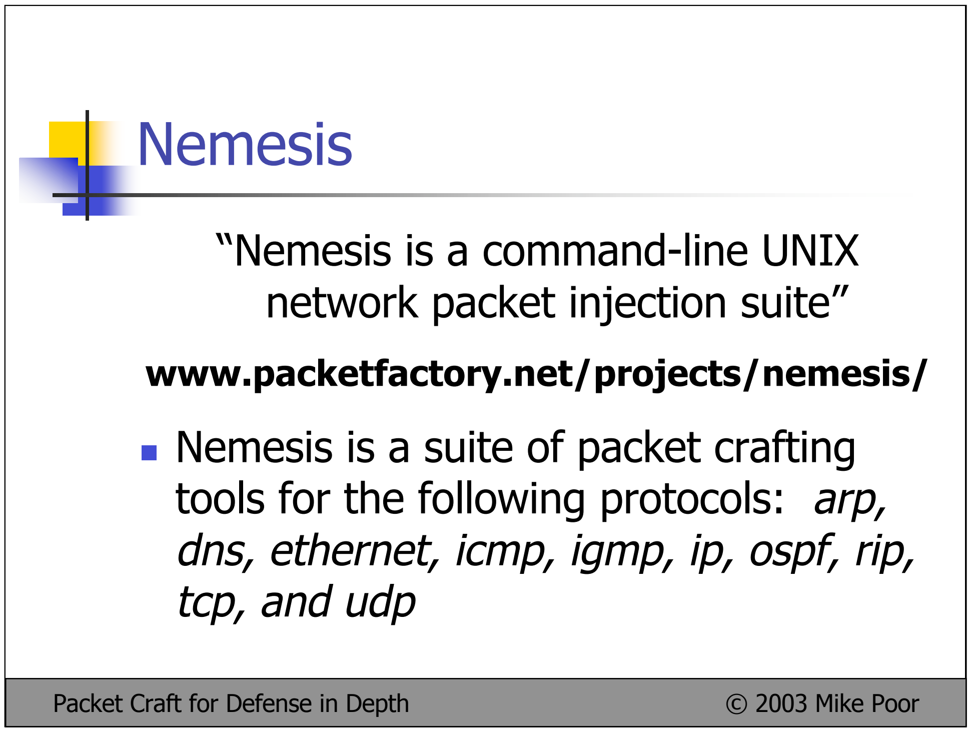
Here we can test the Linux kernels ability to withstand a barrage of half open connections to the same port.

First I set up a netcat listener on port 6060. I then use hping2 to send 1000 SYN packets to port 6060.

Once this is underway, I routinely check how many connections we have open to port 6060:

```
# netstat -an | grep 6060 | wc -l
   190
# netstat -an | grep 6060 | wc -l
   191
# netstat -an | grep 6060 | wc -l
   192
# netstat -an | grep 6060 | wc -l
   192
# netstat -an | grep 6060 | wc -l
   192
$ time hping2 –c 1000 –S 127.0.0.1 –p 6060 real    3m15.203s
```

Sure enough, after about 3 minutes, and 15 seconds, at about the time Linux reaches 191/192 half open connections, the kernel starts to drop the oldest connections.



Nemesis

"Nemesis is a command-line UNIX network packet injection suite"

www.packetfactory.net/projects/nemesis/

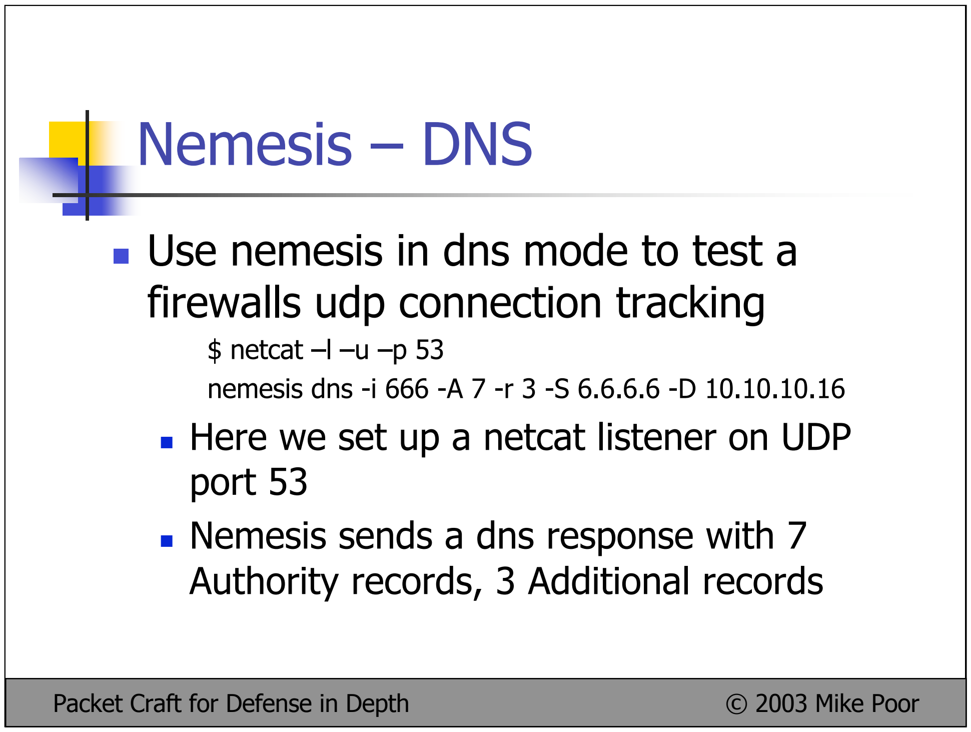
- Nemesis is a suite of packet crafting tools for the following protocols: *arp, dns, ethernet, icmp, igmp, ip, ospf, rip, tcp, and udp*

Packet Craft for Defense in Depth © 2003 Mike Poor

Nemesis is a very powerful suite of packet crafting tools. Nemesis is able to craft packets for the following protocols: arp, dns, Ethernet, icmp, igmp, ip, ospf, rip, tcp, udp.

Nemesis can be used for a wide range of packet crafting goals, from IDS and firewall testing, to recreating traffic, to scanning for live hosts using a variety of different tools. Like hping2, the facility of nemesis being a Unix command line tool, allows the user great freedom in shell scripting whatever test they wish to run.

Nemesis is far more complex then hping2. To run nemesis, you must first install libnet. You can pick up libnet here: http://www.packetfactory.net/projects/libnet/


Nemesis – DNS

- Use nemesis in dns mode to test a firewalls udp connection tracking

```
$ netcat –l –u –p 53
nemesis dns -i 666 -A 7 -r 3 -S 6.6.6.6 -D 10.10.10.16
```

  - Here we set up a netcat listener on UDP port 53
  - Nemesis sends a dns response with 7 Authority records, 3 Additional records

Packet Craft for Defense in Depth © 2003 Mike Poor

netcat –l –u –p 53 sets up a udp listener on port 53.

We fire up nemesis in dns mode to send a dns packet to 127.0.0.1, with a DNS id of 666, 7 Authority records, 3 Additional records, with a source address of 6.6.6.6

```
nemesis dns -i 666 -A 7 -r 3 -S 6.6.6.6 -D 10.10.10.16
```

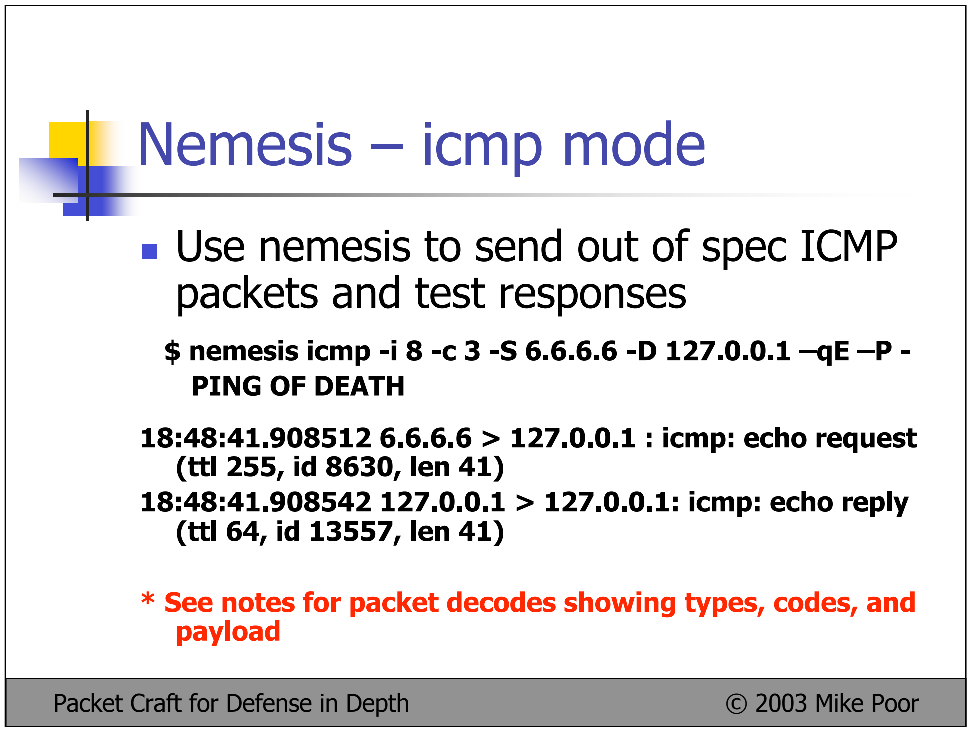
Sure enough, as seen below, the firewall (the brand remain unmentioned) lets an unsolicited UDP DNS reply through:

```
tcpdump: listening on lo

14:40:47.315192 6.6.6.6.38484 > 10.10.10.15.53: [udp sum ok]  666
[0q] [7n] [3au] ns: [|domain] [tos 0x10]  (ttl 255, id 48267, len 40)
```

Interesting to note that when the author was conducting these tests, he was using two separate versions of tcpdump. 3.6 Segfaulted each time I would read the packet with tcpdump –nnvvr foo5.dmp, while 3.7.1 had no problems.

Tethereal interprets the DNS packet as an improper one, as seen below:



Nemesis – icmp mode

- Use nemesis to send out of spec ICMP packets and test responses

```
$ nemesis icmp -i 8 -c 3 -S 6.6.6.6 -D 127.0.0.1 –qE –P -
  PING OF DEATH

18:48:41.908512 6.6.6.6 > 127.0.0.1 : icmp: echo request
   (ttl 255, id 8630, len 41)
18:48:41.908542 127.0.0.1 > 127.0.0.1: icmp: echo reply
   (ttl 64, id 13557, len 41)
```

*** See notes for packet decodes showing types, codes, and payload**

Packet Craft for Defense in Depth © 2003 Mike Poor

Here we use nemesis in icmp mode to send bogus icmp packets, with a icmp type of 8, icmp code of 3 (non existant). Here we also notice something about the ICMP protocol:

from RFC 792, page 15:

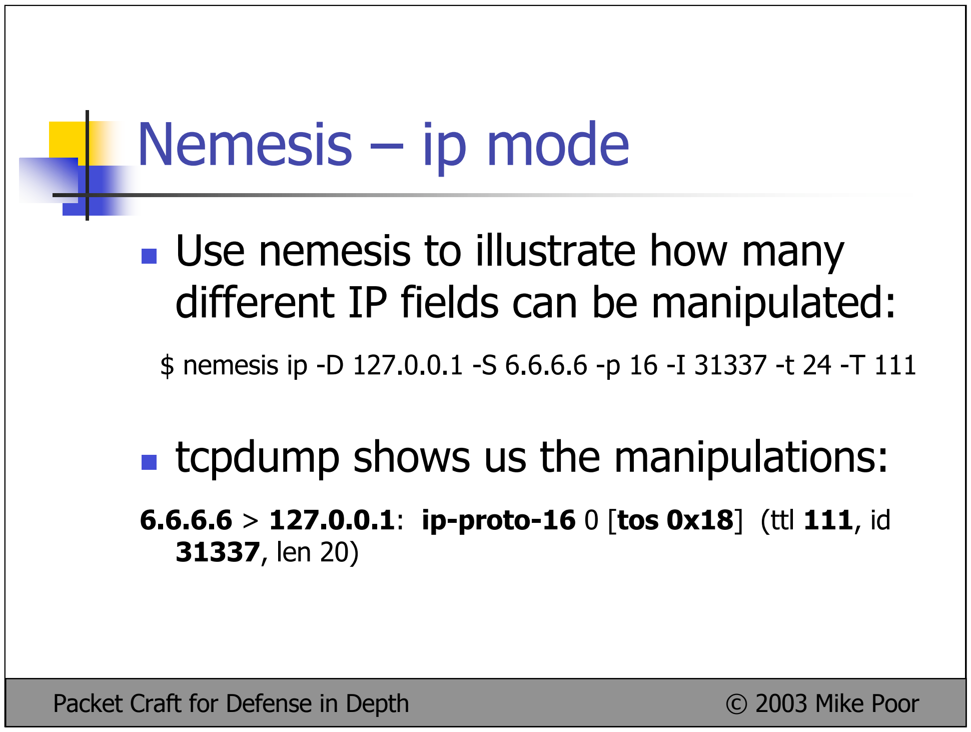
The data received in the echo request message must be returned in the echo reply message.

In essence, what happens when a host receives an Echo request is that it swaps source and destination IP fields, sets the ICMP type to 0, recalculates the checksum and replays the message on the wire.

```
nemesis icmp –qE -i 8 -c 3 -S 6.6.6.6 -D 127.0.0.1

05/25-18:48:41.908512 6.6.6.6 -> 127.0.0.1
ICMP TTL:255 TOS:0x0 ID:8630 IpLen:20 DgmLen:41
Type:8  Code:3  ID:9100   Seq:16540  ECHO
50 49 4E 47 20 4F 46 20 44 45 41 54 48          PING OF DEATH

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+
```



Nemesis – ip mode

- Use nemesis to illustrate how many different IP fields can be manipulated:

```
$ nemesis ip -D 127.0.0.1 -S 6.6.6.6 -p 16 -I 31337 -t 24 -T 111
```

- tcpdump shows us the manipulations:

6.6.6.6 > **127.0.0.1**: **ip-proto-16** 0 [**tos 0x18**] (ttl **111**, id **31337**, len 20)

Packet Craft for Defense in Depth © 2003 Mike Poor

Here we use nemesis in IP mode to generate an IP packet with multiple manipulations, just to show you how easy it is for attackers to set arbitrary values in the IP header.

Nemesis is set to send an IP datagram to 127.0.0.1 with a source address of 6.6.6.6, a protocol number of 16 (CHAOS), an IP id of 31337 (for the kiddiez), a type of service of 24 (Minimize delay), and a TTL of 111

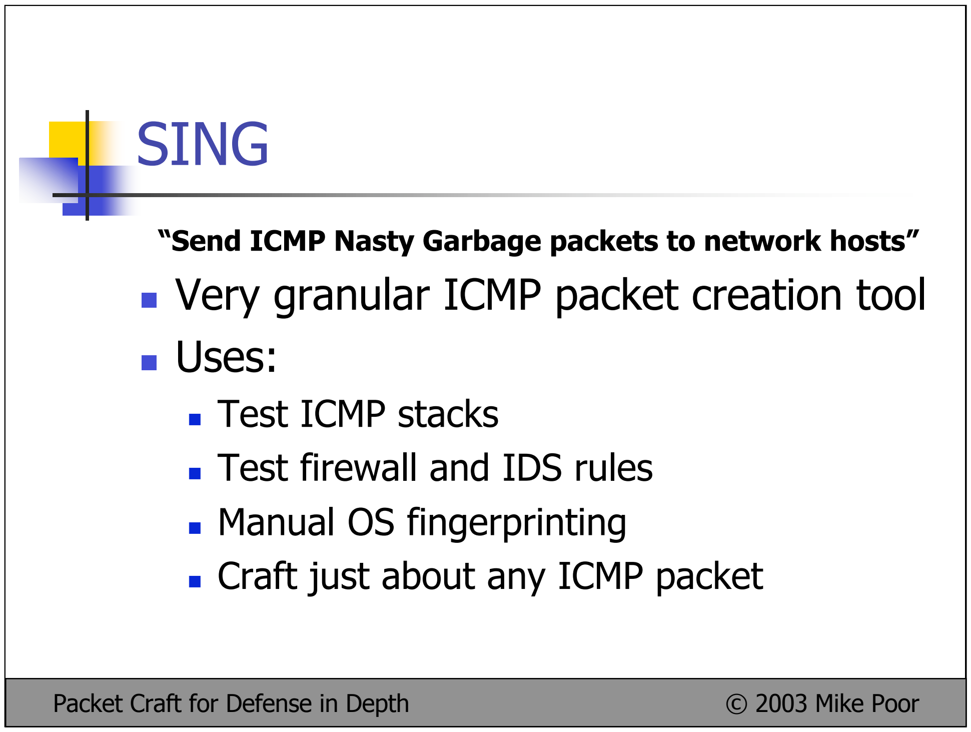
```
nemesis ip -D 127.0.0.1 -S 6.6.6.6 -p 16 -I 31337 -t 24 -T 111
```

We set up our sniffer process:

```
tcpdump –i lo –nnvvs 1514 –w nemesis1.dmp
```

The tcpdump output below is the full packet dump of the nemesis IP packet. Note that in this test, there is no payload. Another thing we note is that tcpdump does not know the name for protocol 16, CHAOS.

```
19:30:16.945350 6.6.6.6 > 127.0.0.1:  ip-proto-16 0 [tos 0x18]  (ttl 111,
id 31337, len 20)
0x0000   4518 0014 7a69 0000 6f10 464c 0606 0606       E...zi..o.FL....
0x0010   7f00 0001                                     ....
```



SING

"Send ICMP Nasty Garbage packets to network hosts"

- Very granular ICMP packet creation tool
- Uses:
  - Test ICMP stacks
  - Test firewall and IDS rules
  - Manual OS fingerprinting
  - Craft just about any ICMP packet

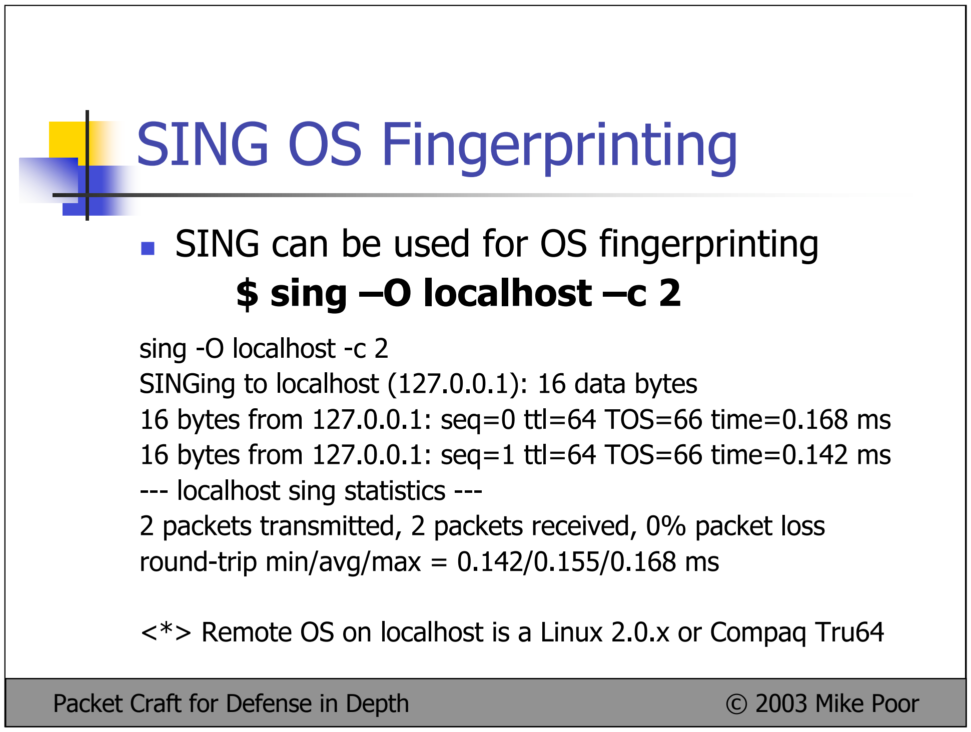
Packet Craft for Defense in Depth © 2003 Mike Poor

SING was written by: Alfredo Andres Omella, Slay <aandres@s21sec.com>

SING was essentially built to order after the research that Ofir Arkin did in: ICMP uses in scanning

(found at: http://www.sys-security.com/html/papers.html).

SING allows you to manipulate the ICMP protocol at will. It is the most granular tool for crafting ICMP packets.



SING OS Fingerprinting

- SING can be used for OS fingerprinting

$ sing –O localhost –c 2

```
sing -O localhost -c 2
SINGing to localhost (127.0.0.1): 16 data bytes
16 bytes from 127.0.0.1: seq=0 ttl=64 TOS=66 time=0.168 ms
16 bytes from 127.0.0.1: seq=1 ttl=64 TOS=66 time=0.142 ms
--- localhost sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.142/0.155/0.168 ms

<*> Remote OS on localhost is a Linux 2.0.x or Compaq Tru64
```

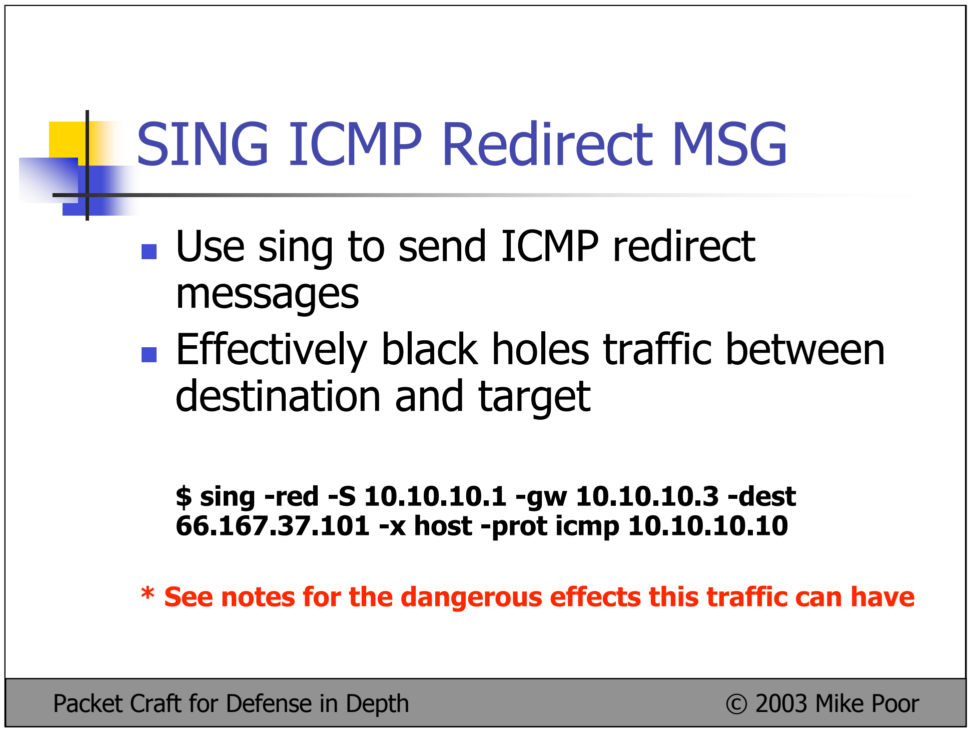
Packet Craft for Defense in Depth © 2003 Mike Poor

Here we use sing to send two packets to local host to try and determine what operating system the host is running.

```
sing -O localhost -c 2
SINGing to localhost (127.0.0.1): 16 data bytes
16 bytes from 127.0.0.1: seq=0 ttl=64 TOS=66 time=0.168 ms
16 bytes from 127.0.0.1: seq=1 ttl=64 TOS=66 time=0.142 ms
--- localhost sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.142/0.155/0.168 ms
<*> Remote OS on localhost is a Linux 2.0.x or Compaq Tru64
```

From Tethereal output, we see that SING is setting the icmp code to 22 on the pings:

```
Internet Control Message Protocol
   Type: 0 (Echo (ping) reply)
   Code: 22
   Checksum: 0x8cfd (correct)
   Identifier: 0x080a
   Sequence number: 01:00
   Data (8 bytes)
```



SING ICMP Redirect MSG

- Use sing to send ICMP redirect messages
- Effectively black holes traffic between destination and target

```
$ sing -red -S 10.10.10.1 -gw 10.10.10.3 -dest
66.167.37.101 -x host -prot icmp 10.10.10.10
```

*** See notes for the dangerous effects this traffic can have**

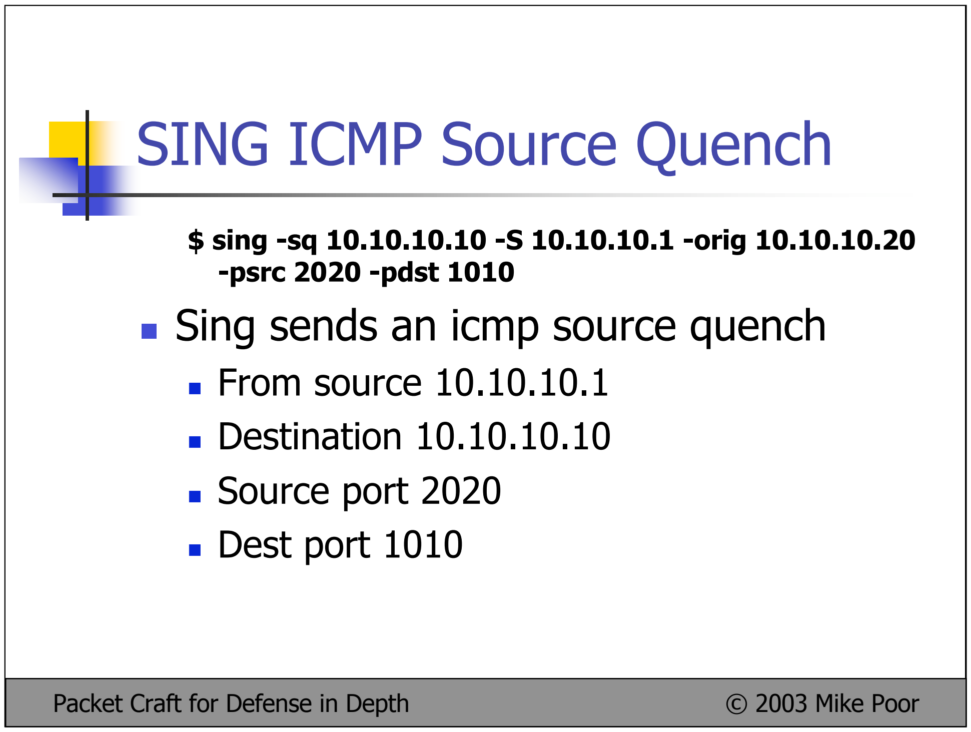
Packet Craft for Defense in Depth © 2003 Mike Poor

```
Kernel IP routing table
Destination    Gateway        Genmask        Flags   MSS Window  irtt
Iface
10.10.10.0     0.0.0.0        255.255.255.0   U       40 0        0 eth0
0.0.0.0        10.10.10.1     0.0.0.0         UG      40 0        0 eth0
```

```
ping www.digitalguardian.net
PING www.digitalguardian.net (66.167.37.101) from 10.10.10.18 :
56(84) bytes of data.
64 bytes from 66.167.37.101 : icmp_seq=1 ttl=50 time=253 ms
64 bytes from 66.167.37.101 : icmp_seq=2 ttl=50 time=252 ms
64 bytes from 66.167.37.101 : icmp_seq=3 ttl=50 time=254 ms
…
```

```
sing -red -S 10.10.10.1 -gw 10.10.10.3 -dest 66.167.37.101 -x host -
prot icmp -ip_id 1 -ip_seq 1 10.10.10.10
```

```
ping www.digitalguardian.net
PING digitalguardian.net (66.167.37.101) from 10.10.10.10 : 56(84)
bytes of data.
From ogum.digitalguardian.net (10.10.10.18): icmp_seq=1 Destination
```



SING ICMP Source Quench

```
$ sing -sq 10.10.10.10 -S 10.10.10.1 -orig 10.10.10.20
  -psrc 2020 -pdst 1010
```

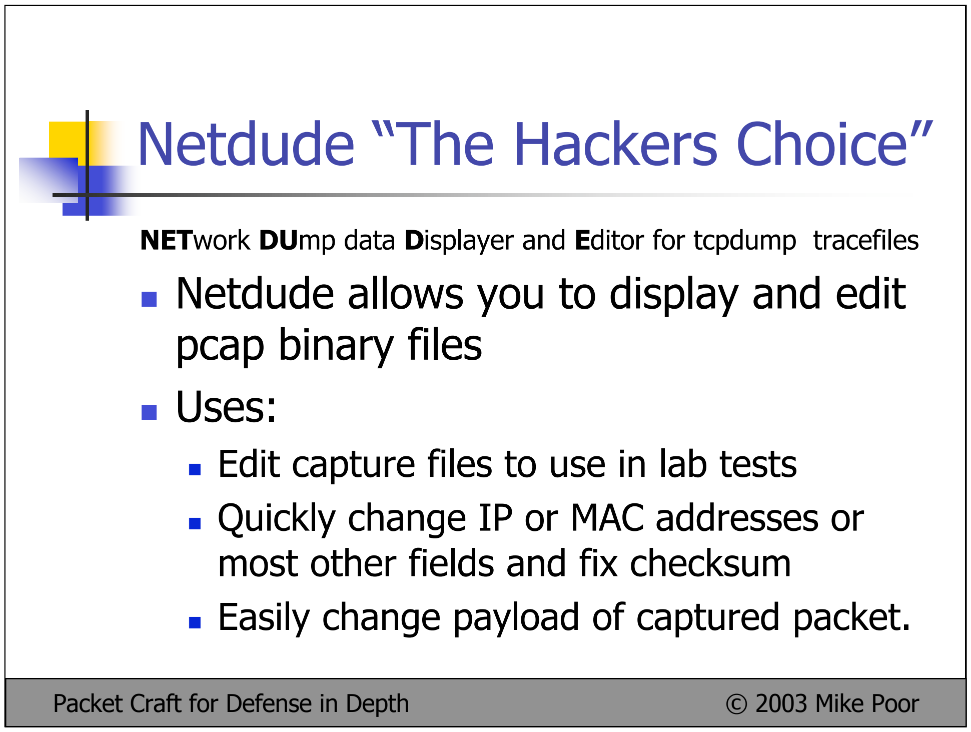
- Sing sends an icmp source quench
  - From source 10.10.10.1
  - Destination 10.10.10.10
  - Source port 2020
  - Dest port 1010

Packet Craft for Defense in Depth © 2003 Mike Poor

Hackers can spoof your gateway address and generate thousands of ICMP Source Quenches, attempting to slow down or shut down your active connections. You can use sing to recreate this traffic, or test your defenses against it.

```
sing -sq 10.10.10.10 -S 10.10.10.1 -orig 10.10.10.20 -psrc 2020 -pdst
1010
```

```
04:24:43.433429 10.10.10.1 > 10.10.10.10: icmp: source quench for
10.10.10.10.2020 > 10.10.20.1010: [|tcp] (ttl 255, id 18991, len 56)
(ttl 255, id 13170, len 56)
```



Netdude "The Hackers Choice"

NETwork **DU**mp data **D**isplayer and **E**ditor for tcpdump tracefiles

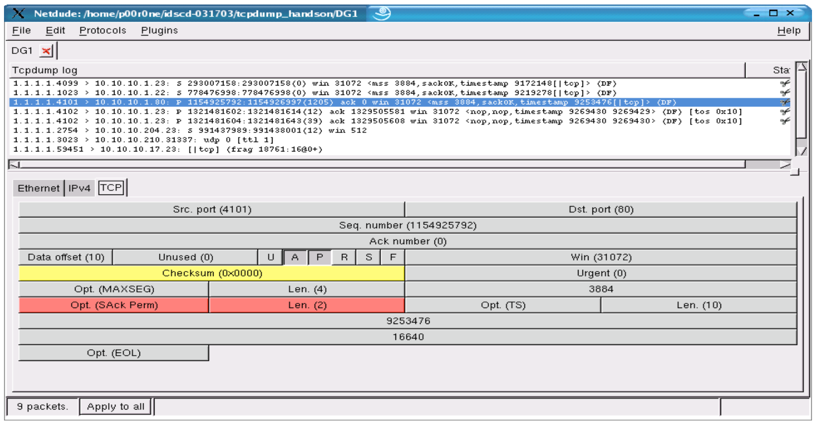
- Netdude allows you to display and edit pcap binary files
- Uses:
  - Edit capture files to use in lab tests
  - Quickly change IP or MAC addresses or most other fields and fix checksum
  - Easily change payload of captured packet.

Packet Craft for Defense in Depth © 2003 Mike Poor

The main website for the project is located at:
http://netdude.sourceforge.net

There are a few prerequisites for installing netdude. Currently, I needed GTK (Gimp Tool Kit), and tcpdump. If when installing netdude, you get an error regarding the configure script not finding GTK-config you will need to install the gtk-devel package.

Netdude



```
1.1.1.1.4099 > 10.10.10.1.23: S 293007158:293007158(0) win 31072 <mss 3884,sackOK,timestamp 9172148[|tcp]> (DF)
1.1.1.1.1023 > 10.10.10.1.22: S 778476998:778476998(0) win 31072 <mss 3884,sackOK,timestamp 9219278[|tcp]> (DF)
1.1.1.1.4101 > 10.10.10.1.80: P 1154925792:1154926997(1205) ack 0 win 31072 <mss 3884,sackOK,timestamp 9253476[|tcp]> (DF)
1.1.1.1.4102 > 10.10.10.1.23: P 1321481602:1321481614(12) ack 1329505581 win 31072 <nop,nop,timestamp 9269430 9269429> (DF) [tos 0x10]
1.1.1.1.4102 > 10.10.10.1.23: P 1321481604:1321481643(39) ack 1329505608 win 31072 <nop,nop,timestamp 9269430 9269430> (DF) [tos 0x10]
1.1.1.1.2754 > 10.10.10.204.23: S 991437989:991438001(12) win 512
1.1.1.1.3023 > 10.10.10.210.31337: udp 0 [ttl 1]
1.1.1.1.59451 > 10.10.17.23: [|tcp] (frag 18761:16@0+)
```

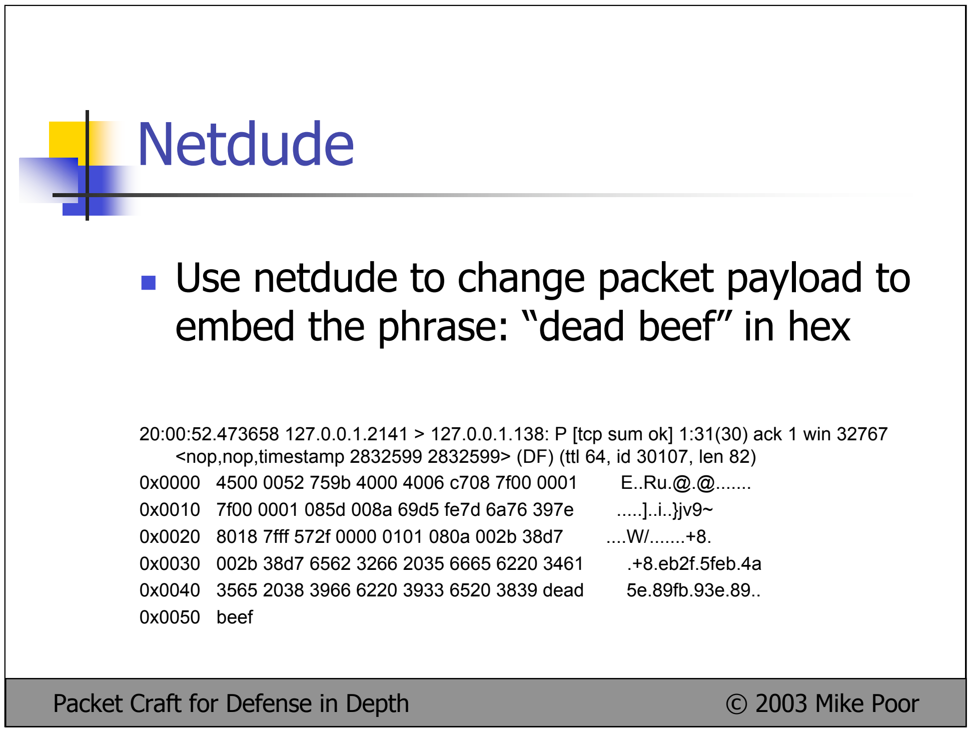
Src. port (4101)								Dst. port (80)	
Seq. number (1154925792)									
Ack number (0)									
Data offset (10)	Unused (0)	U	A	P	R	S	F	Win (31072)	
Checksum (0x0000)								Urgent (0)	
Opt. (MAXSEG)	Len. (4)							3884	
Opt. (SAck Perm)	Len. (2)							Opt. (TS)	Len. (10)
9253476									
16640									
Opt. (EOL)									

Packet Craft for Defense in Depth © 2003 Mike Poor

Screen shot of Netdude, with a tcpdump capture file loaded. Here we see the TCP fields that we can manipulate. Once the changes have been made, Netdude can recompute the checksums for the datagram (in this case, both IP and TCP checksums) if desired. This is a huge advantage over using a simple hex editor or similar.

The following protocols are currently supported for editing under Netdude:

Name	Version
ICMP	0.1.0
Ethernet	0.1.0
IPv4	0.1.0
SLL	0.1.0
TCP	0.1.0
UDP	0.1.0
LLC/SNAP	0.1.0
ARP	0.1.0
FDDI	0.1.0



Netdude

- Use netdude to change packet payload to embed the phrase: "dead beef" in hex

```
20:00:52.473658 127.0.0.1.2141 > 127.0.0.1.138: P [tcp sum ok] 1:31(30) ack 1 win 32767
     <nop,nop,timestamp 2832599 2832599> (DF) (ttl 64, id 30107, len 82)
0x0000   4500 0052 759b 4000 4006 c708 7f00 0001        E..Ru.@.@.......
0x0010   7f00 0001 085d 008a 69d5 fe7d 6a76 397e        .....]..i..}jv9~
0x0020   8018 7fff 572f 0000 0101 080a 002b 38d7        ....W/.......+8.
0x0030   002b 38d7 6562 3266 2035 6665 6220 3461        .+8.eb2f.5feb.4a
0x0040   3565 2038 3966 6220 3933 6520 3839 dead        5e.89fb.93e.89..
0x0050   beef
```

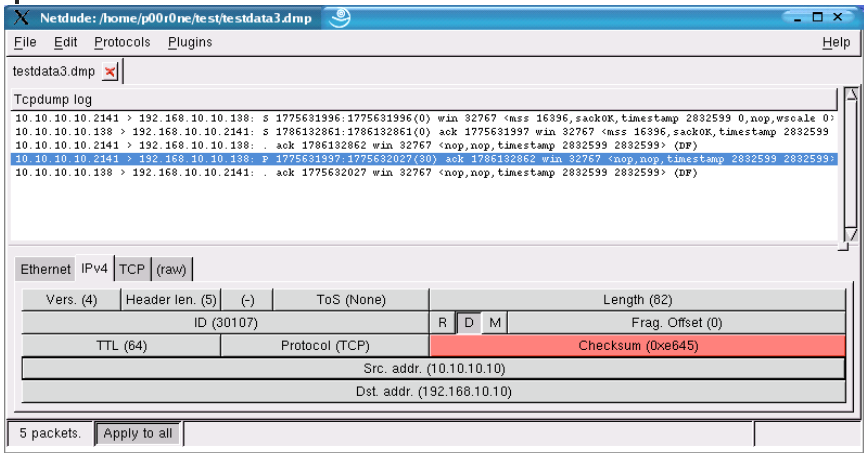
Packet Craft for Defense in Depth © 2003 Mike Poor

```
20:00:52.473658 127.0.0.1.2141 > 127.0.0.1.138: P [tcp sum ok]
1:31(30) ack 1 win 32767 <nop,nop,timestamp 2832599 2832599> (DF)
(ttl 64, id 30107, len 82)
0x0000   4500 0052 759b 4000 4006 c708 7f00 0001
E..Ru.@.@.......
0x0010   7f00 0001 085d 008a 69d5 fe7d 6a76 397e       .....]..i..}jv9~
0x0020   8018 7fff 8436 0000 0101 080a 002b 38d7       .....6.......+8.
0x0030   002b 38d7 6562 3266 2035 6665 6220 3461
.+8.eb2f.5feb.4a
0x0040   3565 2038 3966 6220 3933 6520 3839 6632
5e.89fb.93e.89f2
0x0050   0a64
```

was simply changed to:

```
20:00:52.473658 127.0.0.1.2141 > 127.0.0.1.138: P [tcp sum ok]
1:31(30) ack 1 win 32767 <nop,nop,timestamp 2832599 2832599> (DF)
(ttl 64, id 30107, len 82)
0x0000   4500 0052 759b 4000 4006 c708 7f00 0001
E..Ru.@.@.......
0x0010   7f00 0001 085d 008a 69d5 fe7d 6a76 397e       .....]..i..}jv9~
0x0020   8018 7fff 572f 0000 0101 080a 002b 38d7       ....W/.......+8.
```

Netdude



```
10.10.10.10.2141 > 192.168.10.10.138: S 1775631996:1775631996(0) win 32767 <mss 16396,sackOK,timestamp 2832599 0,nop,wscale 0>
10.10.10.10.138 > 192.168.10.10.2141: S 1786132861:1786132861(0) ack 1775631997 win 32767 <mss 16396,sackOK,timestamp 2832599
10.10.10.10.2141 > 192.168.10.10.138: . ack 1786132862 win 32767 <nop,nop,timestamp 2832599 2832599> (DF)
10.10.10.10.2141 > 192.168.10.10.138: P 1775631997:1775632027(30) ack 1786132862 win 32767 <nop,nop,timestamp 2832599 2832599>
10.10.10.10.138 > 192.168.10.10.2141: . ack 1775632027 win 32767 <nop,nop,timestamp 2832599 2832599> (DF)
```

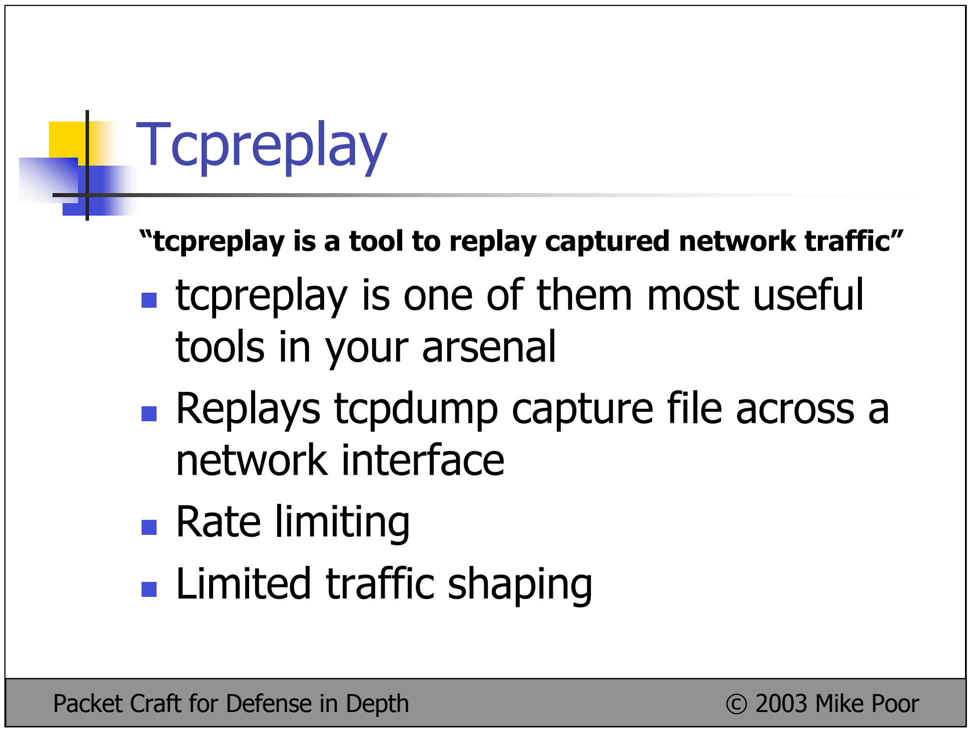
Vers. (4) | Header len. (5) | (-) | ToS (None) | Length (82)
ID (30107) | R | D | M | Frag. Offset (0)
TTL (64) | Protocol (TCP) | Checksum (0xe645)
Src. addr. (10.10.10.10)
Dst. addr. (192.168.10.10)

Packet Craft for Defense in Depth © 2003 Mike Poor

```
20:00:52.473658 0:0:0:0:0:0 0:0:0:0:0:0 0800 96: 127.0.0.1.2141 >
127.0.0.1.138: P [tcp sum ok] 1:31(30) ack 1 win 32767
<nop,nop,timestamp 2832599 2832599> (DF) (ttl 64, id 30107, len 82)
0x0000   4500 0052 759b 4000 4006 c708 7f00 0001
E..Ru.@.@.......
0x0010   7f00 0001 085d 008a 69d5 fe7d 6a76 397e        .....]..i..}jv9~
0x0020   8018 7fff 572f 0000 0101 080a 002b 38d7        ....W/.......+8.
0x0030   002b 38d7 6562 3266 2035 6665 6220 3461
.+8.eb2f.5feb.4a
0x0040   3565 2038 3966 6220 3933 6520 3839 dead
5e.89fb.93e.89..
0x0050   beef
```

Easily changed IP addresses and MAC addresses to:

```
20:00:52.473658 0:be:af:0:de:ad 0:de:ad:0:be:ef 0800 96:
10.10.10.10.2141 > 192.168.10.10.138: P [tcp sum ok] 0:30(30) ack 1
win 32767 <nop,nop,timestamp 2832599 2832599> (DF) (ttl 64, id
30107, len 82)
0x0000   4500 0052 759b 4000 4006 e644 0a0a 0a0a
E..Ru.@.@..D....
0x0010   c0a8 0a0a 085d 008a 69d5 fe7d 6a76 397e        .....]..i..}jv9~
```



Tcpreplay

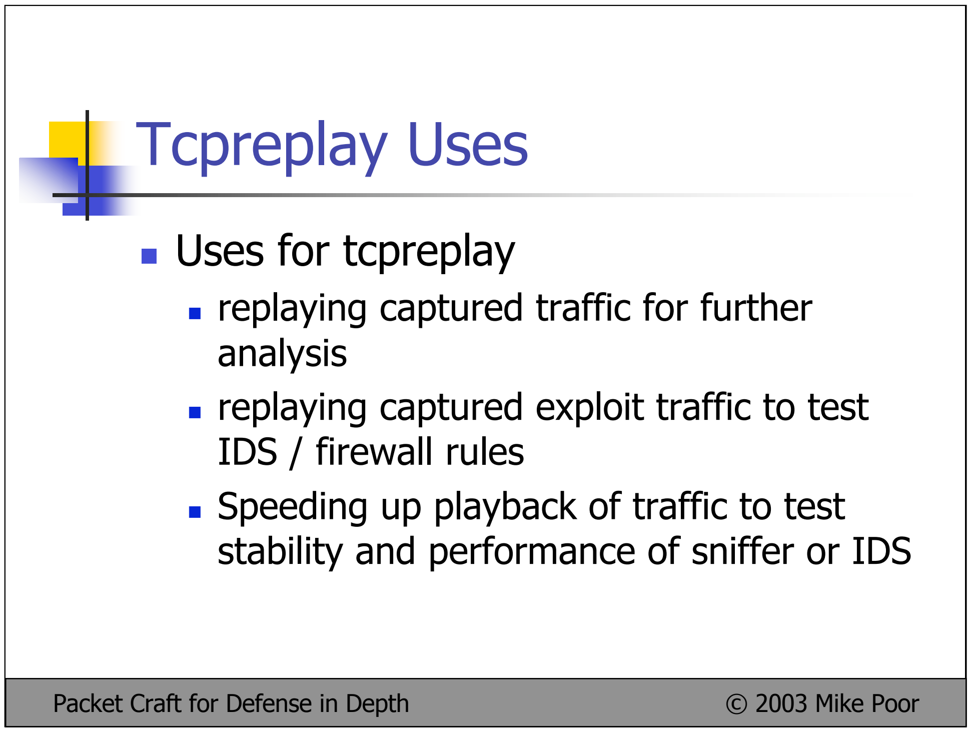
"tcpreplay is a tool to replay captured network traffic"

- tcpreplay is one of them most useful tools in your arsenal
- Replays tcpdump capture file across a network interface
- Rate limiting
- Limited traffic shaping

Packet Craft for Defense in Depth © 2003 Mike Poor

Tcpreplay can be downloaded for free from
http://sourceforge.net/projects/tcpreplay

tcpreplay v 1.4.2 requires Libnet 1.1 or higher. Libnet can be found at:
http://www.packetfactory.net/projects/libnet/

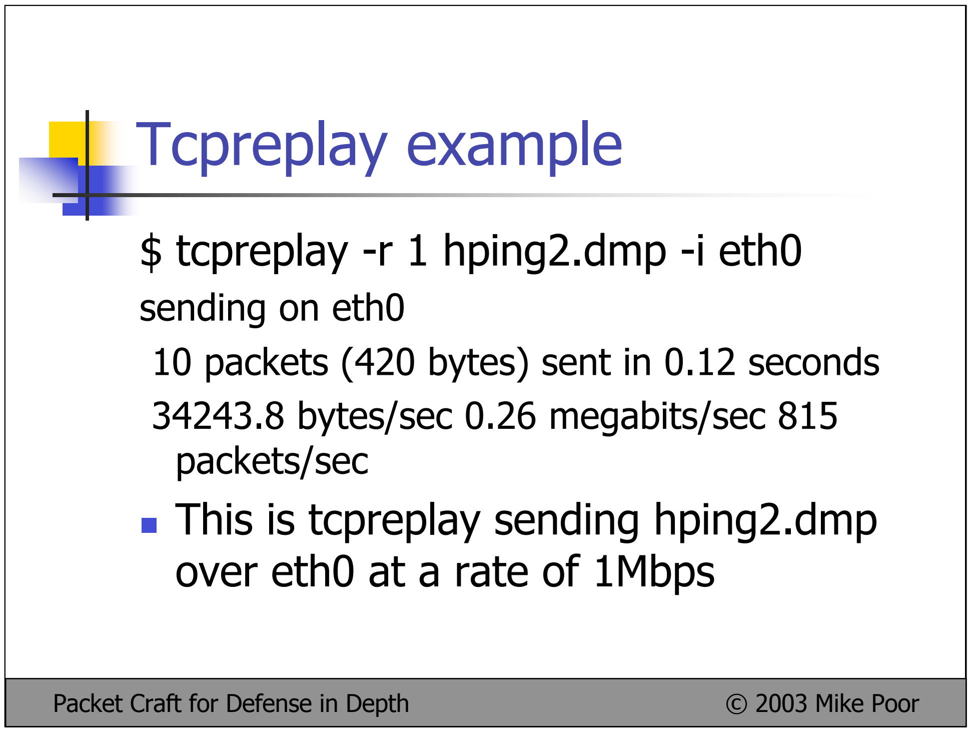


Tcpreplay Uses

- Uses for tcpreplay
  - replaying captured traffic for further analysis
  - replaying captured exploit traffic to test IDS / firewall rules
  - Speeding up playback of traffic to test stability and performance of sniffer or IDS

Packet Craft for Defense in Depth © 2003 Mike Poor

Tcpdump can be used for numerous network tasks. What I end up using it most for is to run pcaps of exploits to test an IDS. I can manage rate limits and speed to further test the IDS capability to withstand high bandwidth, or slow attack rates. You can also use tcpreplay to replay attacks on a simulated lab network, to test how production systems respond.



Tcpreplay example

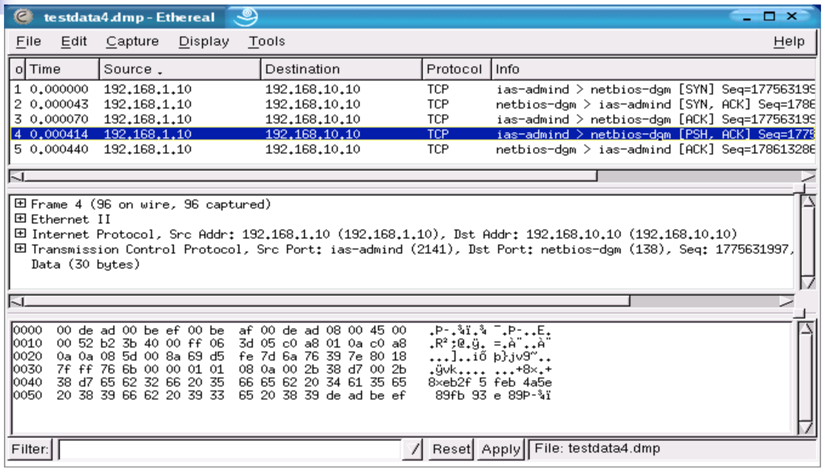
```
$ tcpreplay -r 1 hping2.dmp -i eth0
sending on eth0
 10 packets (420 bytes) sent in 0.12 seconds
 34243.8 bytes/sec 0.26 megabits/sec 815
   packets/sec
```

- This is tcpreplay sending hping2.dmp over eth0 at a rate of 1Mbps

Packet Craft for Defense in Depth © 2003 Mike Poor

```
$ tcpreplay -r 1 hping2.dmp -i eth0
sending on eth0
 10 packets (420 bytes) sent in 0.12 seconds
 34243.8 bytes/sec 0.26 megabits/sec 815 packets/sec
```

Ethereal


testdata4.dmp - Ethereal
File Edit Capture Display Tools Help
Filter: Reset Apply File: testdata4.dmp

```
o Time       Source         Destination     Protocol Info
1 0.000000   192.168.1.10   192.168.10.10   TCP      ias-admind > netbios-dgm [SYN] Seq=17756319
2 0.000043   192.168.1.10   192.168.10.10   TCP      netbios-dgm > ias-admind [SYN, ACK] Seq=178
3 0.000070   192.168.1.10   192.168.10.10   TCP      ias-admind > netbios-dgm [ACK] Seq=17756319
4 0.000414   192.168.1.10   192.168.10.10   TCP      ias-admind > netbios-dgm [PSH, ACK] Seq=177
5 0.000440   192.168.1.10   192.168.10.10   TCP      netbios-dgm > ias-admind [ACK] Seq=17861328
```

```
⊞ Frame 4 (96 on wire, 96 captured)
⊞ Ethernet II
⊞ Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 192.168.10.10 (192.168.10.10)
⊞ Transmission Control Protocol, Src Port: ias-admind (2141), Dst Port: netbios-dgm (138), Seq: 1775631997,
  Data (30 bytes)
```

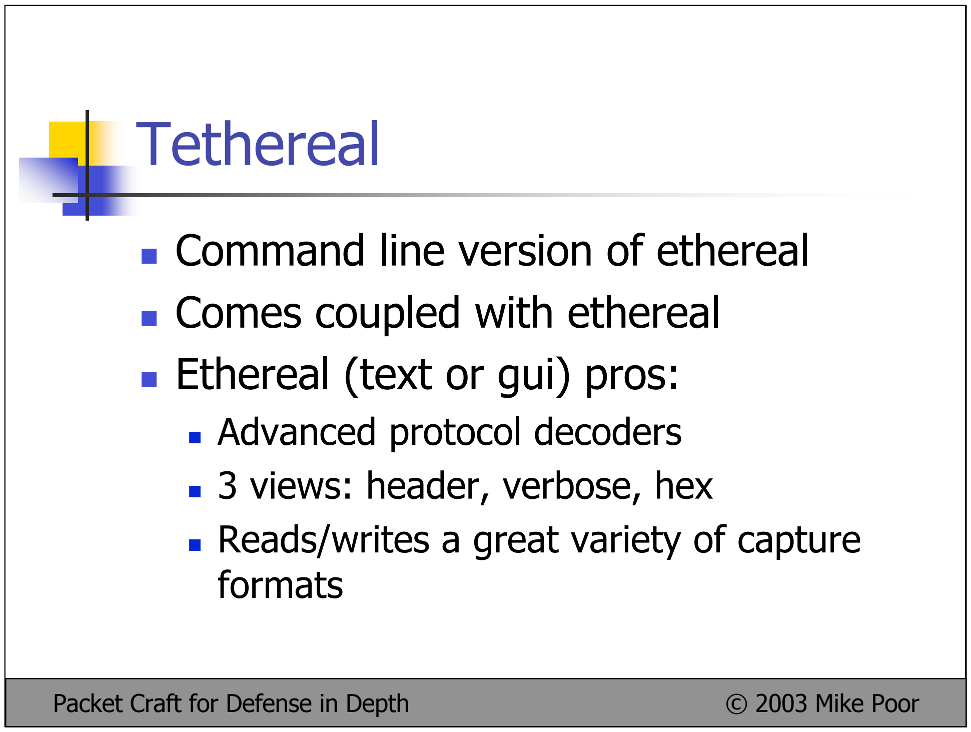
```
0000  00 de ad 00 be ef 00 be  af 00 de ad 08 00 45 00   .Þ-.¾ï.¾ ¯.Þ-..E.
0010  00 52 b2 3b 40 00 ff 06  3d 05 c0 a8 01 0a c0 a8   .R²;@.ÿ. =.À¨..À¨
0020  0a 0a 08 5d 00 8a 69 d5  fe 7d 6a 76 39 7e 80 18   ...]..iõ þ}jv9~..
0030  7f ff 76 6b 00 00 01 01  08 0a 00 2b 38 d7 00 2b   .ÿvk.... ...+8×.+
0040  38 d7 65 62 32 66 20 35  66 65 62 20 34 61 35 65   8×eb2f 5 feb 4a5e
0050  20 38 39 66 62 20 39 33  65 20 38 39 de ad be ef    89fb 93 e 89Þ-¾ï
```

Packet Craft for Defense in Depth © 2003 Mike Poor

Ethereal, grandfather of all commercial sniffers! Ethereal actually takes its good layout and ideas from the old Network General Sniffer. Ethereal is available for free, for a variety of platforms from http://www.ethereal.com

The top pane shows basic packet header and protocol information. The middle pane is almost identical to tcpshow output, where you see packet information printed out verbosely by layer. The final pane is a packet dump in hexadecimal.

Ethereal has a phenomenal filter language. There are currently 381 supported protocols and media in ethereal. If you run it, chances are, ethereal can decode it.



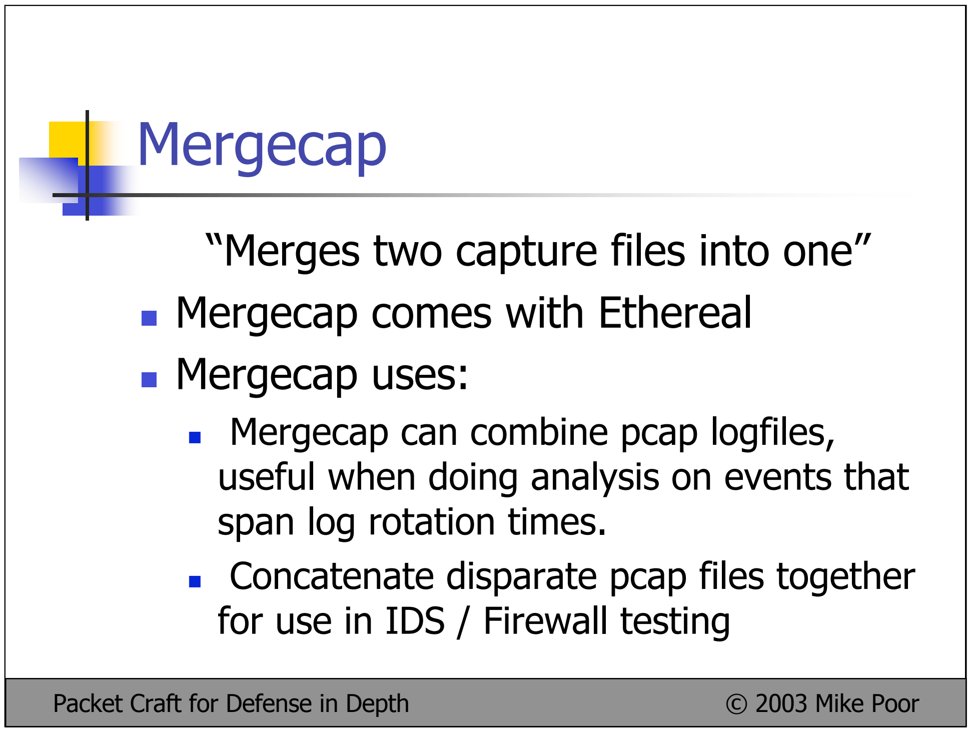
Tethereal

- Command line version of ethereal
- Comes coupled with ethereal
- Ethereal (text or gui) pros:
  - Advanced protocol decoders
  - 3 views: header, verbose, hex
  - Reads/writes a great variety of capture formats

Packet Craft for Defense in Depth © 2003 Mike Poor

Ethereal and Tethereal are effectively the same tool. One is the GUI version of the other. The advantage of using Tethereal over Ethereal is that you can run it as root without running X windows as root, as well being able to use the unix text processing tools to parse the data.

Ethereal and Tethereal should be used, in my opinion, as packet replay and analysis tools, not as a packet capture tool. In my experience, Ethereal is far slower then tcpdump or snort for packet capture.



Mergecap

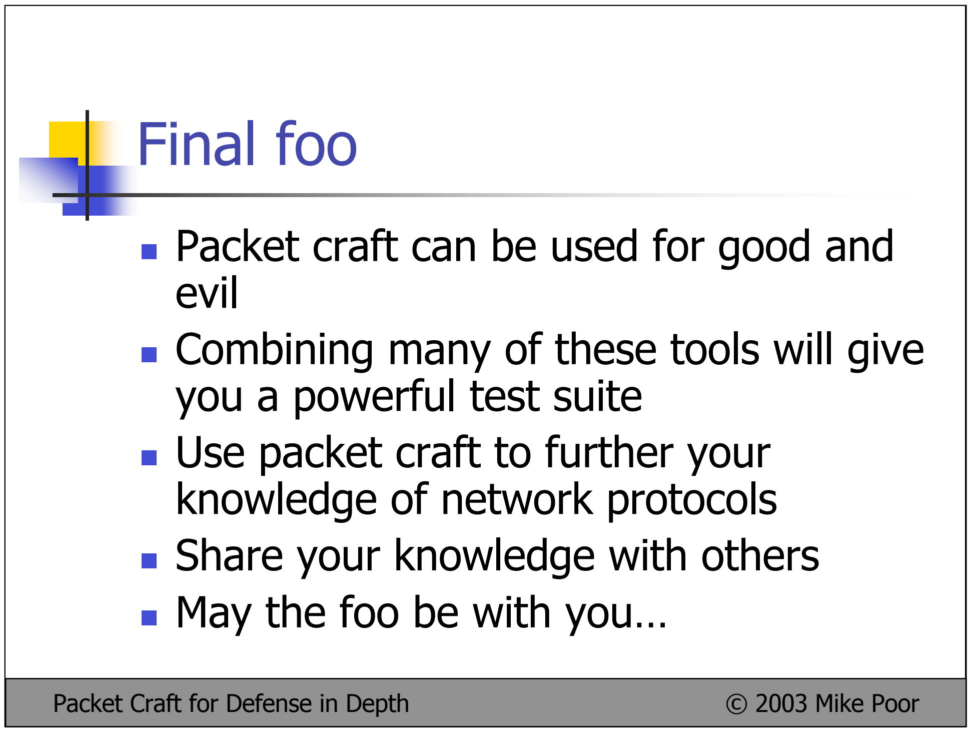
"Merges two capture files into one"

- Mergecap comes with Ethereal
- Mergecap uses:
  - Mergecap can combine pcap logfiles, useful when doing analysis on events that span log rotation times.
  - Concatenate disparate pcap files together for use in IDS / Firewall testing

Packet Craft for Defense in Depth © 2003 Mike Poor

Mergecap comes with Ethereal and Tethereal. Mergecap is a very handy tool. It has come most in handy when combining pcap files together in order to conduct analysis on an event that spans log rotation times.

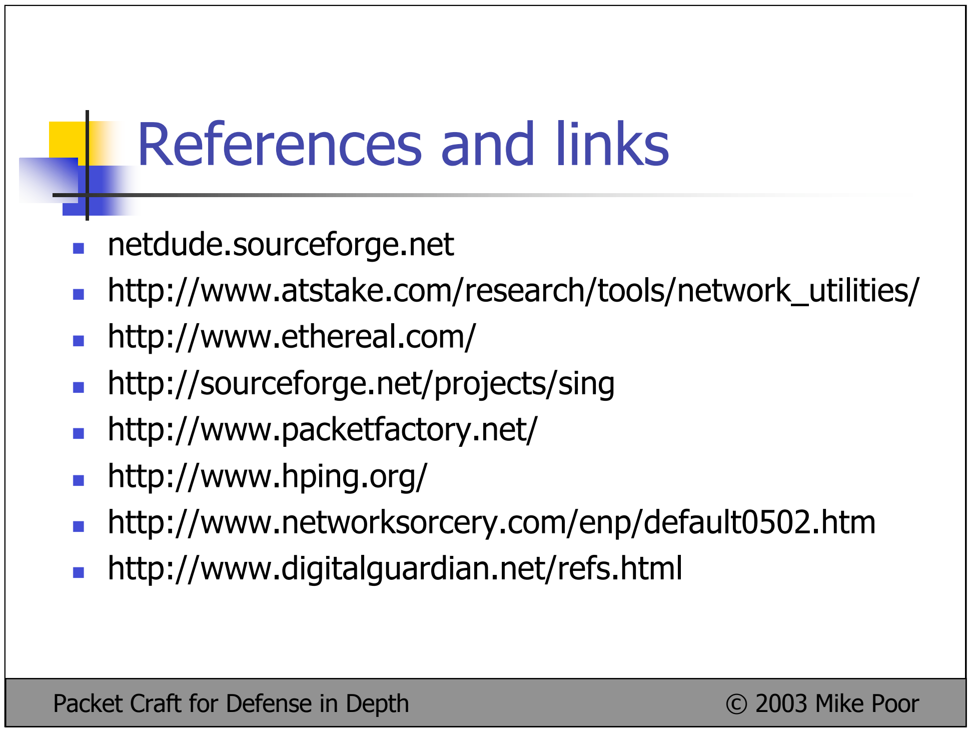
For example: You rotate your IDS logs every hour. Suppose an incident began at 14:59 and ended at 15:01. Your logs would be in two separate pcap files. The easy way to merge them: $ mergecap –w combined.pcap 1400.pcap 1500.pcap



Final foo

- Packet craft can be used for good and evil
- Combining many of these tools will give you a powerful test suite
- Use packet craft to further your knowledge of network protocols
- Share your knowledge with others
- May the foo be with you…

Packet Craft for Defense in Depth © 2003 Mike Poor



References and links

- netdude.sourceforge.net
- http://www.atstake.com/research/tools/network_utilities/
- http://www.ethereal.com/
- http://sourceforge.net/projects/sing
- http://www.packetfactory.net/
- http://www.hping.org/
- http://www.networksorcery.com/enp/default0502.htm
- http://www.digitalguardian.net/refs.html

Packet Craft for Defense in Depth © 2003 Mike Poor

Thanks and good night!