

Packet Crafting

OpenFest, Sofia
November, 2013



Disclaimer

!!! Use at your own risk !!!



Motivation



Motivation (2)



Agenda

- ping(8), traceroute(8), telnet(1), nmap(1)
- ng_source(4), tcpdump(1)
- bridge(4), vlan(4), svlan(4), trunk(4)/lagg(4)
- yersinia(8)
- nemesis(1), libnet
- hyenae
- pierf, Scapy
- netmap(4)
- iperf
- PF_PACKET, etc... sockets, C code
- OS network stack & daemons



Let's start

- ping(8)
 - options: flood, quiet, multicast, source route, etc
- traceroute(8)
 - incrementing IP TTL probes
 - UDP, but may be random IP protocol number
- telnet(1)
 - TCP port 23, unless custom specified
- nmap(1)
 - network security scanner
- and more
 - arping, nslookup/host/dig, ...




```
RX packets:268 errors:0 dropped:0 overruns:0 frame:0
TX packets:268 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:18564 (18.5 KB) TX bytes:18564 (18.5 KB)
```

```
sotir@sotir-HP-550:~$ nslookup yahoo.com
;; connection timed out; no servers could be reached
```

```
sotir@sotir-HP-550:~$ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 127.0.1.1
```

```
sotir@sotir-HP-550:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 3999ms
```

```
sotir@sotir-HP-550:~$ traceroute 8.8.8.8
The program 'traceroute' can be found in the following packages:
* inetutils-traceroute
* traceroute
Try: sudo apt-get install <selected package>
sotir@sotir-HP-550:~$
```



Update information

Incomplete Language Support

The language support files are incomplete. You can install the missing files by clicking the "Run this action now" button and following the instructions. If you would like to use the default language support instead (click the "System Settings..." button in the top right corner of this window).

Run this action now

Close

ng_source(4)

- *tcpdump(8)*

```
0x0000: 0a00
11:40:23.411341 00:e0:0c:11:95:22 > 00:00:00:77:77:78, ethertype MPLS unicast (0x8847), length 82: MPLS (label 28673,
0x0000: 0000 0077 7778 00e0 0c11 9522 8847 0700
0x0010: 11ff 0100 5e02 020b 0040 9530 450b 8100
0x0020: 0001 0800 4500 002e fdc5 0000 4011 35b9
0x0030: 6401 0132 e002 020b f28f 4908 001a 9d1b
0x0040: 476f 7420 4d75 6c74 6963 6173 743f 213f
0x0050: 0a00
```

- there's wireshark too

- *netgraph(3)*

- graph based kernel networking subsystem of FreeBSD

- *ng_ether(4)*

- a node automatically created for every Ethernet interface in the system

ng_source(4) contd.

```
#!/bin/sh
#####
# TODO: insert some nice comment here
#####
ECHO=/bin/echo
NGHOOK=/usr/sbin/nghook
HOOKNAME=input

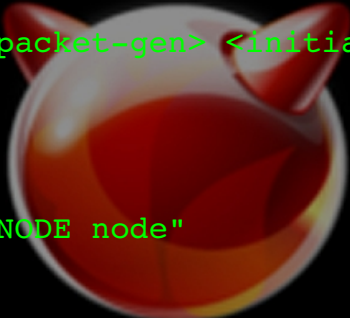
usage () {
$ECHO      "Usage: $0 <ng-node> <count> <packet-gen> <initial-arg> <increment-gen>"
}

...

$ECHO "Injecting $LIMIT packets into $NGNODE node"

# Loop from 1 to 255
while [ "$var1" -le $LIMIT ]
do
DSTIP=${ARGS}
$ECHO "Injecting packet with DST IP ${DSTIP}"

${PACKGEN} ${DSTIP} | ${NGHOOK} ${NGNODE} ${HOOKNAME}
let "var1 = var1 + 1" >> /dev/null # let "var1+=1"
ARGS=`${INCGEN} ${DSTIP}`
done
exit 0
```



Ether-like interfaces

- bridge(4)
- vlan(4), svlan(4)
- trunk(4)/lagg(4)
- tap(4)



Yersinia

- <http://www.yersinia.net>
- Supported protocols
 - STP, CDP, DTP, DHCP, HSRP, IEEE 802.1Q, IEEE 802.1X, ISL, VTP
- UI
 - GTK, ncurses, command line, netclient
- predefined scenarios
- netclient
 - TCP port 12000
 - passwd/enable passwd : root/tomac

nemesis(1)

NEMESIS ==- The NEMESIS Project Version 1.4 (Build 26)

NEMESIS Usage:

```
nemesis [mode] [options]
```

NEMESIS modes:

```
arp  
dns  
ethernet  
icmp  
igmp  
ip  
ospf (currently non-functional)  
rip  
tcp  
udp
```



```
gcc -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT hyenae-hsrp.o -MD -MP -MF .deps/hyenae-hsrp.Tpo -c -o hyenae-hsrp.o hyenae-hsrp.c
mv -f .deps/hyenae-hsrp.Tpo .deps/hyenae-hsrp.Po
gcc -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT hyenae-attack.o -MD -MP -MF .deps/hyenae-attack.Tpo -c -o hyenae-attack.o hyenae-attack.c
mv -f .deps/hyenae-attack.Tpo .deps/hyenae-attack.Po
gcc -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT hyenae-patterns.o -MD -MP -MF .deps/hyenae-patterns.Tpo -c -o hyenae-patterns.o hyenae-patterns.c
mv -f .deps/hyenae-patterns.Tpo .deps/hyenae-patterns.Po
gcc -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT hyenae-config.o -MD -MP -MF .deps/hyenae-config.Tpo -c -o hyenae-config.o hyenae-config.c
mv -f .deps/hyenae-config.Tpo .deps/hyenae-config.Po
gcc -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT hyenae-base.o -MD -MP -MF .deps/hyenae-base.Tpo -c -o hyenae-base.o hyenae-base.c
mv -f .deps/hyenae-base.Tpo .deps/hyenae-base.Po
gcc -g -O2 -o hyenae hyenae.o hyenae-assistant.o hyenae-remote.o hyenae-common.o hyenae-protocol.o hyenae-eth.o hyenae-arp.o hyenae-pppoe.o hyenae-ip.o hyenae-icmp.o hyenae-tcp.o hyenae-udp.o hyenae-dns.o hyenae-bootp.o hyenae-dhcp.o hyenae-hsrp.o hyenae-attack.o hyenae-patterns.o hyenae-config.o hyenae-base.o -lpcap -ldnet -lpcap -ldnet -lpcap -ldnet -lpcap -ldnet -lpcap -ldnet
hyenae-base.o(.text+0x1077): In function `hy_output':
/home/syrinx/hyenae-0.36-1/src/hyenae-base.c:78: warning: vsprintf() is often misused, please use vsnprintf()
hyenae-assistant.o(.text+0x79): In function `hy_assistant_handle_tcp_syn_flood':
/home/syrinx/hyenae-0.36-1/src/hyenae-assistant.c:605: warning: strcpy() is almost always misused, please use strncpy()
hyenae-base.o(.text+0xd42): In function `hy_random':
/home/syrinx/hyenae-0.36-1/src/hyenae-base.c:421: warning: rand() isn't random; consider using arc4random()
hyenae-base.o(.text+0x1144): In function `hy_initialize':
/home/syrinx/hyenae-0.36-1/src/hyenae-base.c:47: warning: srand() seed choices are invariably poor
hyenae-base.o(.text+0x10d0): In function `hy_output':
/home/syrinx/hyenae-0.36-1/src/hyenae-base.c:88: warning: strcat() is almost always misused, please use strlcat()
hyenae-assistant.o(.text+0x46d): In function `hy_assistant_handle_dns_query_flood':
/home/syrinx/hyenae-0.36-1/src/hyenae-assistant.c:807: warning: sprintf() is often misused, please use snprintf()
gcc -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT hyenaed.o -MD -MP -MF .deps/hyenaed.Tpo -c -o hyenaed.o hyenaed.c
mv -f .deps/hyenaed.Tpo .deps/hyenaed.Po
gcc -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT hyenaed-daemon.o -MD -MP -MF .deps/hyenaed-daemon.Tpo -c -o hyenaed-daemon.o hyenaed-daemon.c
mv -f .deps/hyenaed-daemon.Tpo .deps/hyenaed-daemon.Po
gcc -g -O2 -o hyenaed hyenaed.o hyenaed-daemon.o hyenae-common.o hyenae-protocol.o hyenae-eth.o hyenae-arp.o hyenae-pppoe.o hyenae-ip.o hyenae-icmp.o hyenae-tcp.o hyenae-udp.o hyenae-dns.o hyenae-bootp.o hyenae-dhcp.o hyenae-hsrp.o hyenae-attack.o hyenae-patterns.o hyenae-config.o hyenae-base.o -lpcap -ldnet -lpcap -ldnet -lpcap -ldnet -lpcap -ldnet -lpcap -ldnet
hyenae-base.o(.text+0x1077): In function `hy_output':
/home/syrinx/hyenae-0.36-1/src/hyenae-base.c:78: warning: vsprintf() is often misused, please use vsnprintf()
/usr/lib/libdnet.so.1.0: warning: strcpy() is almost always misused, please use strncpy()
hyenae-base.o(.text+0xd42): In function `hy_random':
/home/syrinx/hyenae-0.36-1/src/hyenae-base.c:421: warning: rand() isn't random; consider using arc4random()
hyenae-base.o(.text+0x1144): In function `hy_initialize':
/home/syrinx/hyenae-0.36-1/src/hyenae-base.c:47: warning: srand() seed choices are invariably poor
hyenaed-daemon.o(.text+0x8f9): In function `hy_attack_to_string':
/home/syrinx/hyenae-0.36-1/src/hyenaed-daemon.c:757: warning: strcat() is almost always misused, please use strlcat()
hyenaed-daemon.o(.text+0x134):/home/syrinx/hyenae-0.36-1/src/hyenaed-daemon.c:546: warning: sprintf() is often misused, please use snprintf()
$
```

hyenae

• No *BSD packages available on SF
• assumes /usr/include & /usr/lib

hyenae (contd.)

- “ERROR: Root privileges required”
- Supported protocols
 - ARP, PPPoE, ICMP, TCP, UDP, DNS, DHCP, HSRP
- UI
 - command line, QT-frontent
- Example



```
$ sudo hyenae -a pppoe-discover -i em0 -s 01:00:82:00:00:c2 -d  
01:00:82:00:00:c3  
* Initializing  
* Opening network interface (em0)  
* Launching attack
```

Press any key to stop

Scapy

- “Powerful interactive packet manipulation program in python”
- Python interpreter disguised as a Domain Specific Language
- Fast Packet Designing
- Interactive packet and result manipulation
- Fast packet generator ? :dd



Scapy - ls()

```
Welcome to Scapy (2.2.0)
```

```
>>>
```

```
>>> ls()
```

```
ARP : ARP
```

```
ASN1_Packet : None
```

```
BOOTP : BOOTP
```

```
CookedLinux : cooked linux
```

```
DHCP : DHCP options
```

```
DHCP6 : DHCPv6 Generic Message)
```

```
DHCP6OptAuth : DHCP6 Option - Authentication
```

```
DHCP6OptBCMCSDomains : DHCP6 Option - BCMCS Domain Name List
```

```
DHCP6OptBCMCSservers : DHCP6 Option - BCMCS Addresses List
```

```
DHCP6OptClientFQDN : DHCP6 Option - Client FQDN
```

```
DHCP6OptClientId : DHCP6 Client Identifier Option
```

```
DHCP6OptDNSDomains : DHCP6 Option - Domain Search List option
```

```
DHCP6OptDNSServers : DHCP6 Option - DNS Recursive Name Server
```

```
DHCP6OptElapsedTime : DHCP6 Elapsed Time Option
```

```
DHCP6OptGeoConf :
```

```
DHCP6OptIAAddress : DHCP6 IA Address Option (IA_TA or IA_NA suboption)
```

```
...
```

```
<6 more screens of output>
```

Scapy - lcs()

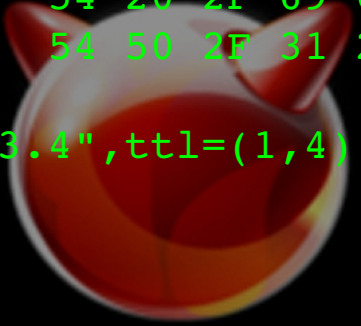
```
>>> lcs()
bind_layers      : Bind 2 layers on some specific fields' values
corrupt_bits     : Flip a given percentage or number of bits from a string
corrupt_bytes   : Corrupt a given percentage or number of bytes from a string
defrag          : defrag(plist) -> ([not fragmented], [defragmented],
defragment      : defrag(plist) -> plist defragmented as much as possible
etherleak       : Exploit Etherleak flaw
fragment        : Fragment a big IP datagram
fuzz            : Transform a layer into a fuzzy layer by replacing some default values by
random objects
hexdiff         : Show differences between 2 binary strings
hexdump        : --
hexedit        : --

rdpcap         : Read a pcap file and return a packet list
send           : Send packets at layer 3
sendp         : Send packets at layer 2
sendpfast     : Send packets at layer 2 using tcpreplay for performance
sniff         : Sniff packets
split_layers   : Split 2 layers previously bound
sr            : Send and receive packets at layer 3
srbt         : send and receive using a bluetooth socket
srflood       : Flood and receive packets at layer 3
srloop       : Send a packet at layer 3 in loop and print the answer each time
srp          : Send and receive packets at layer 2
traceroute    : Instant TCP traceroute
wrpcap       : Write a list of packets to a pcap file

<some commands ommitted, such as wireshark>
```

Scapy – sending a packet

```
Welcome to Scapy (2.2.0)
>>> a=Ether()/IP(dst="www.slashdot.org")/TCP()/ "GET /index.html
HTTP/1.0 \n\n"
>>> hexdump(a)
0000  00 22 68 5A AA E4 74 E5  0B E3 39 68 08 00 45 00  ."hZ..t...9h..E.
0010  00 43 00 01 00 00 40 06  41 3B AC 14 00 12 D8 22  .C....@.A;....."
0020  B5 30 00 14 00 50 00 00  00 00 00 00 00 00 50 02  .0...P.....P.
0030  20 00 84 38 00 00 47 45  54 20 2F 69 6E 64 65 78  ..8..GET /index
0040  2E 68 74 6D 6C 20 48 54  54 50 2F 31 2E 30 20 0A  .html HTTP/1.0 .
0050  0A  .
>>> sendp(Ether()/IP(dst="1.2.3.4",ttl=(1,4)), iface="wlan0")
....
Sent 4 packets.
```



Scapy – writing your own class

- TBD
- (homework)



netmap(4)

- fast packet I/O framework
- reduce syscalls to improve packet processing performance
- in many cases, shared buffers between user and kernel space
- `open("/dev/netmap")`
- `ioctl(.., NIOCREGIF, ..)` to bind to an interface
- fill in available buffers with data
- non-blocking `ioctl(.., NIOCTXSYNC) / ioctl(.., NIOCRXSYNC)` to transmit/receive data
- supported drivers - `em(4)`, `igb(4)`, `ixgbe(4)`, `re(4)`



```
shteryana@aphrodite:/usr/src/tools/tools/netmap %
```

```
shteryana@aphrodite:/usr/src/tools/tools/netmap % ./pkt-gen
```

```
main [1461] missing ifname
```

```
Usage:
```

netmap(4)'s pkt-gen- example

```
pkt-gen arguments
```

- i interface interface name
- f function tx rx ping pong
- n count number of iterations (can be 0)
- t pkts_to_send also forces tx mode
- r pkts_to_receive also forces rx mode
- l pkts_size in bytes excluding CRC
- d dst-ip end with %n to sweep n addresses
- s src-ip end with %n to sweep n addresses
- D dst-mac end with %n to sweep n addresses
- S src-mac end with %n to sweep n addresses
- a cpu_id use setaffinity
- b burst size testing, mostly
- c cores cores to use
- p threads processes/threads to use
- T report_ms milliseconds between reports
- P use libpcap instead of netmap
- w wait_for_link_time in seconds

```
shteryana@aphrodite:/usr/src/tools/tools/netmap %
```

netmap(4) - example

```
char *buf;
int fd;
struct pollfd      fds;
struct nmreq       nmr;
struct netmap_if *nifp;

fd = open("/dev/netmap", O_RDWR);

strcpy(nmr.nr_name, "em0", sizeof(nmr.nr_name));
nmr.nr_version = NETMAP_API;
ioctl(ng.fds.fd , NIOCGINFO, &nmr);

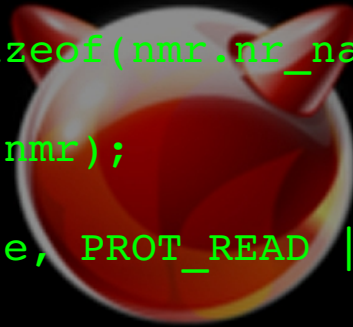
buf = mmap(NULL, nmr.nr_memsize, PROT_READ | PROT_WRITE,
           MAP_SHARED, fd, 0));

ioctl(fd , NIOCREGIF, &nmr);

fds.events = POLLOUT | POLLWRNORM | POLLWRBAND;
fds.fd = fd;

/* prepare packet in buf */
poll(&fds, 1, INFTIM);

ioctl(fd, NIOCTXSYNC, NULL); /* optional */
```



iperf

- commonly used network testing tool
- creates UDP/TCP streams to measure throughput
- iperf (July, 2010) vs iperf3 (March, 2013)
- client and server
- uni-/bi-directional tests



PF_PACKET socket & friends

Chapter 29. Datalink Access	
29.1	Introduction 787
29.2	BSD Packet Filter (PF_PACKET) 789
29.3	Datalink Provider Interface (DLPI) 790
29.4	Linux: SOCK_PACKET and PF_PACKET 791
29.5	libpcap: Packet Capture Library 792
29.6	libnet: Packet Creation and Injection Library 793
29.7	Examining the UDP Checksum Field 793
29.8	Summary 815

Chapter 30. Client/Server Design Alternatives	
30.1	Introduction 817
30.2	TCP Client Alternatives 819
30.3	TCP Test Client 820
30.4	TCP Iterative Server 821
30.5	TCP Concurrent Server, One Child per Client 822
30.6	TCP Preforked Server, No Locking Around accept 826
30.7	TCP Preforked Server, File Locking Around accept 832
30.8	TCP Preforked Server, Thread Locking Around accept 835
30.9	TCP Preforked Server, Descriptor Passing 836

29.3 Datalink Provider Interface (DLPI)

SVR4 provides datalink access through DLPI. DLPI is a protocol-independent interface designed by AT&T that interfaces to the service provided by the datalink layer [Unix International 1991]. Access to DLPI is by sending and receiving STREAMS messages.

There are two styles of DLPI. In one style, there is a single device to open, and the desired interface is specified using a DLPI DL_ATTACH_REQ request. In the other style, the application simply opens the device (e.g., /dev/lx0). But for efficient operation, two additional STREAMS modules are normally pushed onto the stream: pfmod, which performs packet filtering within the kernel, and bufmod, which buffers the data destined for the application. We show how to use DLPI in Section 29.3.2.

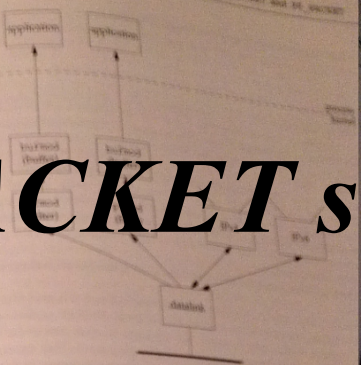


Figure 29.2 Packet capture using DLPI, pfmod, and bufmod

Conceptually, this is similar to the way that the libpcap library works. The libpcap library supports filtering within the kernel, and bufmod, which buffers the data destined for the application, and the amount of data and number of seconds supporting a snapshot length and a read timeout.

One interesting difference, however, is the type of pseudomachine supported by the BPF and pfmod filters. The BPF filter is a directed acyclic control flow graph (CFG) while pfmod uses a Boolean expression. The Boolean expression is evaluated on a register machine while the CFG is evaluated on a stack machine. (See the work of McCarrie and Jacobsen [1993].) BPF is normally 5 to 20 times faster than the Boolean expression filter, depending on the complexity of the filter.

Another difference is that BPF always makes the filtering decision before copying the packet to user space. In contrast, the filter will discard, depending on the BPF implementation, the packet only if it is going to give it to user space. This is a difference.

There are two methods of receiving packets from the datalink layer: an original method, which is more widely available but less flexible, and a newer method, which is more flexible but less widely available. The newer method, which introduces more flexibility, is to create a socket of family PF_PACKET. To obtain sufficient privileges (similar to creating a raw socket), and the socket must be a non-stream value specifying the libnet: Packet Creation and Injection Library. The second argument to socket() can be SOCK_PACKET with the link-layer header removed, or SOCK_PACKET with the link-layer header. SOCK_PACKET sockets only return the complete packet to receive all frames from the datalink, receive.

29.4 Datalink Access

If you want to use IP4 for the filter, you must use the SOCK_PACKET socket type. Other constants for the final argument are ETH_P_ARP and ETH_P_IPv6. Specifying a protocol of ETH_P_IP tells the datalink which frame types to pass to the socket for the frames the datalink receives. If the datalink supports a promiscuous mode (e.g., an Ethernet), then the device must be set into promiscuous mode. This is done via the SOCK_PACKET socket option. The SOCK_PACKET socket option is a packet structure that is defined by the SOCK_PACKET structure. The SOCK_PACKET structure is defined by the SOCK_PACKET structure. Other constants for the final argument are ETH_P_ARP and ETH_P_IPv6. Specifying a protocol of ETH_P_IP tells the datalink which frame types to pass to the socket for the frames the datalink receives. If the datalink supports a promiscuous mode (e.g., an Ethernet), then the device must be set into promiscuous mode. This is done via the SOCK_PACKET socket option. The SOCK_PACKET socket option is a packet structure that is defined by the SOCK_PACKET structure. The SOCK_PACKET structure is defined by the SOCK_PACKET structure.

Some differences are evident when comparing this Linux feature to BPF. The Linux feature provides no kernel buffering and no filtering is available on newer systems (via the SO_ATTACH_FILTER socket option). It is a normal socket receive buffer, but multiple frames cannot be buffered together and passed to the application with a single read. This increases the overhead involved in copying the potentially voluminous amounts of data from the kernel to the application.

SOCK_PACKET provides no filtering by device. (PF_PACKET sockets can be linked to a device by calling bind()) If ETH_P_IP is specified in the call to socket(), then all IPv4 packets from all devices (Ethernet, PPP links, SLIP links, and the loopback device, for example) are passed to the socket. A generic socket address structure is returned by recvfrom(), and the sa_data member contains the device name (e.g., eth0). The application must then discard data from the device to which it is not interested. The problem again is too much data can be returned to the application, which can get in the way when monitoring a high-speed network.

29.5 libpcap: Packet Capture Library

The packet capture library, libpcap, provides implementation-independent access to the underlying packet capture facility provided by the OS. Currently, it supports only the reading of packets (although adding a few lines of code to the library lets one write datalink packets too on some systems). See the next section for a description of another library that supports not only writing datalink packets, but also constructing arbitrary packets.

libnet: Packet Creation and Injection Library

libnet provides an interface to craft and inject arbitrary packets into the network. It provides both raw socket and datalink access modes in an implementation-independent manner. The library hides many of the details of crafting the IP and UDP or TCP packets and provides a simple and portable access to the underlying network. The library is available from <http://www.libnet.org/>.

The library is available from <http://www.libnet.org/>. The library is available from <http://www.libnet.org/>. The library is available from <http://www.libnet.org/>.

Examining the UDP Checksum Field

Enabling or disabling UDP checksums is normally done on a system-wide basis. We will build our own UDP datagram (the DNS query) and we will also show the same code using libnet. We could use a raw socket to send the query, but we want to show how to use the IP, UDP, and build a complete IP datagram.

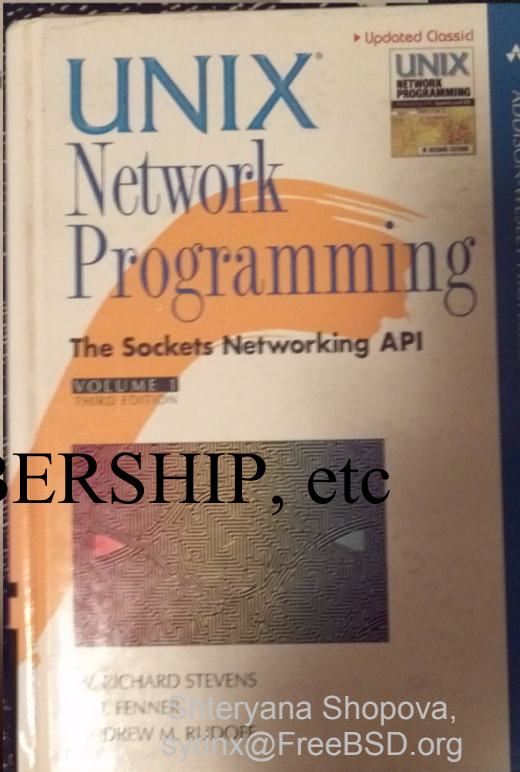
SOCK_DGRAM or SOCK_RAW modes

IP_OPTIONS

Chapter 27. IP Options	
27.1	Introduction 709
27.2	IPV4 Options 711
27.3	IPV4 Source Route Options 719
27.4	IPV6 Extension Headers and Destination Options 725
27.5	IPV6 Hop-by-Hop Options and Destination Options 731
27.6	IPV6 Routing Header 732
27.7	IPV6 Sticky Options 733
27.8	Historical IPV6 Advanced API 732
27.9	Summary 733

Raw Sockets

Chapter 28. Raw Sockets	
28.1	Introduction 735
28.2	Raw Socket Creation 736
28.3	Raw Socket Input 737
28.4	Raw Socket Output 739
28.5	Raw Socket Program 741
28.6	traceroute Program 755
28.7	An ICMP Message Daemon 769
28.8	Summary 786



PF_PACKET

- Linux-only

- SOCK_DGRAM or SOCK_RAW modes

BPF

socket options

IP_TOS, IP_TTL,

MCAST_JOIN/LEAVE_GROUP,

IP_ADD/DROP_(SOURCE)_MEMBERSHIP, etc

References

<http://www.kohala.com/start/unpv12e.html>

<http://www.unpbook.com/src.html>

<http://www.secdev.org/projects/scapy/>

<http://info.iet.unipi.it/~luigi/netmap>

http://www.secdev.org/conf/scapy_pacsec05.pdf

<http://docs.python.org/2/tutorial/introduction.html>

<http://www.secdev.org/projects/scapy/doc/usage.html#interactive-t>

Questions?



Thank you!



* sources from this presentation available at <http://people.freebsd.org/~syrinx/pktgen/>