VPN Tunneling Lab

2018级 信息安全 管箫 18307130012

Task 1: Network Setup

我们将需要三台虚拟机。

Server

ens33（Internet)

ip : 192.168.61.138 mac： 00:0c:29:01:41:ae

ens38（Internel)

ip： 192.168.226.1 mac： 00:0c:29:01:41:b8

Host U

ens33（Internet)

ip： 192.168.61.139 mac： 00:0c:29:a3:8a:e6

Host V

ens33（Internel)

ip： 192.168.226.101 mac： 00:0c:29:aa:55:ad

Test

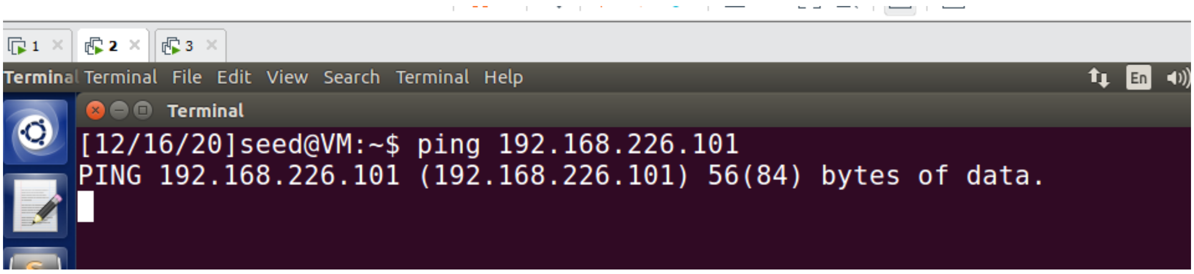
Host U can communicate with VPN Server

```
[12/16/20]seed@VM:~$ ping 192.168.61.138
PING 192.168.61.138 (192.168.61.138) 56(84) bytes of data.
64 bytes from 192.168.61.138: icmp_seq=1 ttl=64 time=0.970 ms
64 bytes from 192.168.61.138: icmp_seq=2 ttl=64 time=0.526 ms
64 bytes from 192.168.61.138: icmp_seq=3 ttl=64 time=0.403 ms
```

VPN Server can communicate with Host V

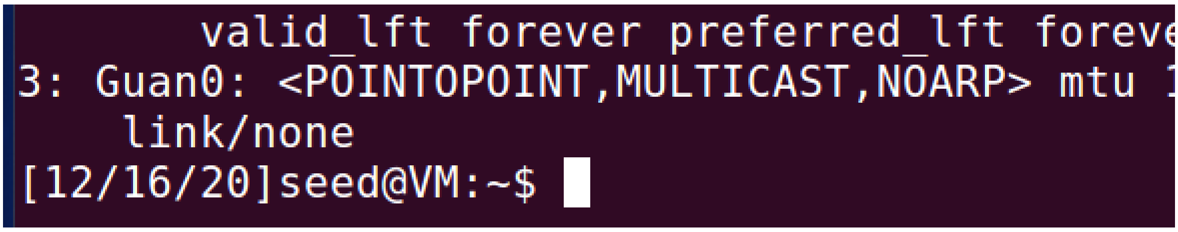
```
Terminal
[12/16/20]seed@VM:~$ ping 192.168.226.101
PING 192.168.226.101 (192.168.226.101) 56(84) bytes of data.
64 bytes from 192.168.226.101: icmp_seq=1 ttl=64 time=0.516 ms
64 bytes from 192.168.226.101: icmp_seq=2 ttl=64 time=0.348 ms
```

Host U should not be able to communicate with Host V

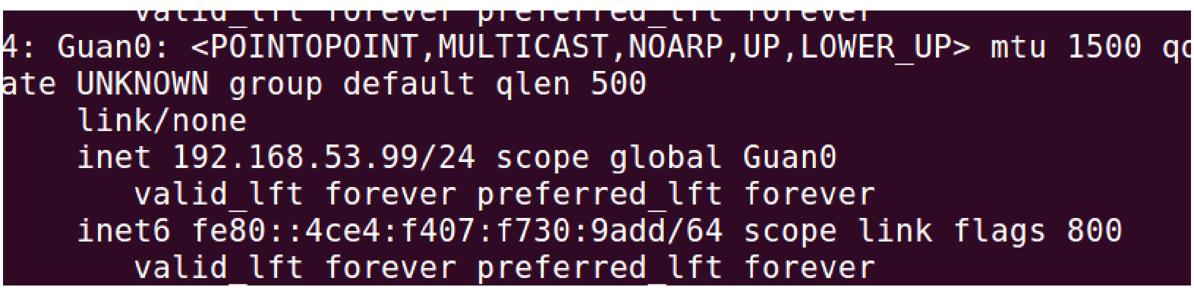


Task 2: Create and Configure TUN Interface

Task 2.a: Name of the Interface

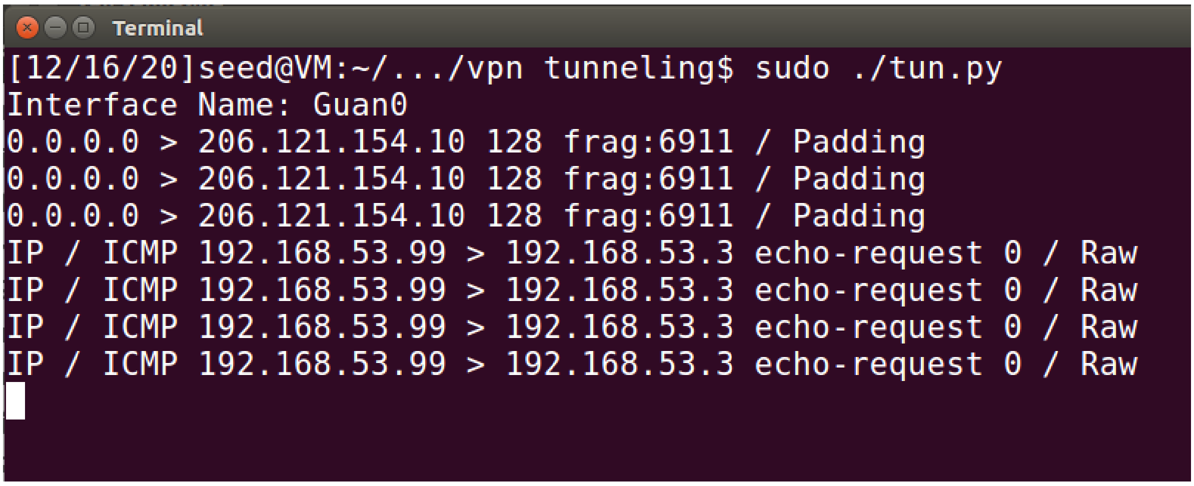


Task 2.b: Set up the TUN Interface



可以观察到该虚拟设备拥有了对应的ipv4和ipv6子网归属以及被启动。

Task 2.c: Read from the TUN Interface



ping 192.168.53.3

程序打印出了发送的icmp包的基本信息，这是因为icmp包的目标网段在TUN的网段中，经由TUN被路由。

ping 192.168.60.3

程序没有输出，这是因为这个地址不属于192.168.53.99/24网段，不由程序路由。

Task 2.d: Write to the TUN Interface

spoof icmp reply

```
Terminal
[12/16/20]seed@VM:~$ ping 192.168.53.2 -c 1
PING 192.168.53.2 (192.168.53.2) 56(84) bytes of data.
8 bytes from 192.168.53.2: icmp_seq=1 ttl=64 (truncated)

--- 192.168.53.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2147483.647/0.000/0.000/0.000 ms
```

```python
while True:
    # Get a packet from the tun interface
    packet = os.read(tun, 2048)
    ip = IP(packet)
    print(ip.summary())
    if ip.proto == 1:
        icmp = ip[ICMP]
        print(icmp.type)
        if icmp.type == 8:
            print("reply")
            newip = IP(src=ip.dst, dst=ip.src)
            newicmp = ICMP(type = 0, id = icmp.id, seq = icmp.seq)
            newpkt = newip/newicmp
            newpkt.show()
            os.write(tun, bytes(newpkt))
```

arbitrary data

```
Traceback (most recent call last):
  File "./tun.py", line 41, in <module>
    os.write(tun, b'12345')
OSError: [Errno 22] Invalid argument
[12/16/20]seed@VM:~/.../vpn tunneling$
```

出错，程序跳出

Task 3: Send the IP Packet to VPN Server Through a Tunnel

The server program tun_server.py

Implement the client program tun client.py

```
while True:
    # Get a packet from the tun interface
    packet = os.read(tun, 2048)
    if True:
        # Send the packet via the tunnel
        ssock.sendto(packet, (192.168.61.138, 9090))
```

Testing

IP address belonging to the 192.168.53.0/24

显示从Host U的外网地址向Server的外网地址的12345端口发送了包，内容是192.168.53.99向192.168.53.3的包。

```
192.168.61.139:42531 --> 192.168.61.138:12345
 Inside: 192.168.53.99 --> 192.168.53.3
192.168.61.139:42531 --> 192.168.61.138:12345
 Inside: 192.168.53.99 --> 192.168.53.3
192.168.61.139:42531 --> 192.168.61.138:12345
 Inside: 0.0.0.0 --> 115.182.40.138
192.168.61.139:42531 --> 192.168.61.138:12345
 Inside: 192.168.53.99 --> 192.168.53.3
192.168.61.139:42531 --> 192.168.61.138:12345
 Inside: 192.168.53.99 --> 192.168.53.3
192.168.61.139:42531 --> 192.168.61.138:12345
 Inside: 192.168.53.99 --> 192.168.53.3
192.168.61.139:42531 --> 192.168.61.138:12345
 Inside: 192.168.53.99 --> 192.168.53.3
```

IP address in the 192.168.60.0/24

```
Terminal
[12/16/20]seed@VM:~/.../VPN Tunnel$ sudo ./tun_server.py
192.168.61.139:44043 --> 192.168.61.138:12345
 Inside: 0.0.0.0 --> 168.237.64.105
192.168.61.139:44043 --> 192.168.61.138:12345
 Inside: 0.0.0.0 --> 168.237.64.105
192.168.61.139:44043 --> 192.168.61.138:12345
 Inside: 0.0.0.0 --> 168.237.64.105
192.168.61.139:44043 --> 192.168.61.138:12345
 Inside: 192.168.53.99 --> 192.168.226.2
192.168.61.139:44043 --> 192.168.61.138:12345
 Inside: 192.168.53.99 --> 192.168.226.2
192.168.61.139:44043 --> 192.168.61.138:12345
 Inside: 192.168.53.99 --> 192.168.226.2
```

Task 4: Set Up the VPN Server

	Time	Source	Destination	Protocol
61	2020-12-16 19:36:26.3179334…	192.168.53.99	192.168.226.101	ICMP
64	2020-12-16 19:36:26.3185908…	192.168.53.99	192.168.226.101	ICMP
65	2020-12-16 19:36:26.3190094…	192.168.226.101	192.168.53.99	ICMP
66	2020-12-16 19:36:26.3190161…	192.168.226.101	192.168.53.99	ICMP
68	2020-12-16 19:36:27.3401599…	192.168.53.99	192.168.226.101	ICMP
69	2020-12-16 19:36:27.3401688…	192.168.53.99	192.168.226.101	ICMP
70	2020-12-16 19:36:27.3405796…	192.168.226.101	192.168.53.99	ICMP
71	2020-12-16 19:36:27.3405855…	192.168.226.101	192.168.53.99	ICMP
73	2020-12-16 19:36:28.3645546…	192.168.53.99	192.168.226.101	ICMP
74	2020-12-16 19:36:28.3645652…	192.168.53.99	192.168.226.101	ICMP
75	2020-12-16 19:36:28.3649895…	192.168.226.101	192.168.53.99	ICMP
76	2020-12-16 19:36:28.3649963…	192.168.226.101	192.168.53.99	ICMP
78	2020-12-16 19:36:29.3882938…	192.168.53.99	192.168.226.101	ICMP
79	2020-12-16 19:36:29.3883035…	192.168.53.99	192.168.226.101	ICMP
80	2020-12-16 19:36:29.3887530…	192.168.226.101	192.168.53.99	ICMP
81	2020-12-16 19:36:29.3887600…	192.168.226.101	192.168.53.99	ICMP

Task 5: Handling Traffic in Both Directions

ping

1	2020-12-16 21:15:31.9747429…	192.168.61.139	192.168.61.138	UDP	128	12345 → 12345 Len=84
2	2020-12-16 21:15:32.1942195…	192.168.53.99	192.168.226.101	ICMP	100	Echo (ping) request id=0x2e07, seq=1/256, ttl=64 (reply in 3)
3	2020-12-16 21:15:32.1946067…	192.168.226.101	192.168.53.99	ICMP	100	Echo (ping) reply id=0x2e07, seq=1/256, ttl=64 (request in 2)
4	2020-12-16 21:15:32.1946208…	192.168.226.101	192.168.53.99	ICMP	100	Echo (ping) reply id=0x2e07, seq=1/256, ttl=63
5	2020-12-16 21:15:32.4204463…	192.168.61.138	192.168.61.139	UDP	128	12345 → 12345 Len=84
6	2020-12-16 21:15:32.9761889…	192.168.61.139	192.168.61.138	UDP	128	12345 → 12345 Len=84
7	2020-12-16 21:15:33.1978994…	192.168.53.99	192.168.226.101	ICMP	100	Echo (ping) request id=0x2e07, seq=2/512, ttl=64 (reply in 8)
8	2020-12-16 21:15:33.1982532…	192.168.226.101	192.168.53.99	ICMP	100	Echo (ping) reply id=0x2e07, seq=2/512, ttl=64 (request in 7)
9	2020-12-16 21:15:33.1982677…	192.168.226.101	192.168.53.99	ICMP	100	Echo (ping) reply id=0x2e07, seq=2/512, ttl=63
10	2020-12-16 21:15:33.4206688…	192.168.61.138	192.168.61.139	UDP	128	12345 → 12345 Len=84
11	2020-12-16 21:15:33.9771127…	192.168.61.139	192.168.61.138	UDP	128	12345 → 12345 Len=84
12	2020-12-16 21:15:34.1940093…	192.168.53.99	192.168.226.101	ICMP	100	Echo (ping) request id=0x2e07, seq=3/768, ttl=64 (reply in 13)
13	2020-12-16 21:15:34.1945020…	192.168.226.101	192.168.53.99	ICMP	100	Echo (ping) reply id=0x2e07, seq=3/768, ttl=64 (request in 12)
14	2020-12-16 21:15:34.1945165…	192.168.226.101	192.168.53.99	ICMP	100	Echo (ping) reply id=0x2e07, seq=3/768, ttl=63
15	2020-12-16 21:15:34.4164505…	192.168.61.138	192.168.61.139	UDP	128	12345 → 12345 Len=84

telnet

	Time	Source	Destination	Protocol	Length	Info
16	2020-12-16 21:16:33.7176064…	192.168.61.138	192.168.61.139	UDP	96	12345 → 12345 Len=52
17	2020-12-16 21:16:33.7188684…	192.168.61.139	192.168.61.138	UDP	96	12345 → 12345 Len=52
18	2020-12-16 21:16:33.9411879…	192.168.53.99	192.168.226.101	TCP	68	56584 → 23 [ACK] Seq=1828932558 Ack=356166295 Win=29312
19	2020-12-16 21:16:33.9416306…	192.168.226.101	192.168.53.99	TELNET	107	Telnet Data ...
20	2020-12-16 21:16:33.9416460…	192.168.226.101	192.168.53.99	TCP	107	[TCP Retransmission] 23 → 56584 [PSH, ACK] Seq=356166295
21	2020-12-16 21:16:34.1565355…	192.168.61.138	192.168.61.139	UDP	135	12345 → 12345 Len=91
22	2020-12-16 21:16:34.1575611…	192.168.61.139	192.168.61.138	UDP	96	12345 → 12345 Len=52
23	2020-12-16 21:16:34.1581608…	192.168.61.139	192.168.61.138	UDP	171	12345 → 12345 Len=127
24	2020-12-16 21:16:34.3777158…	192.168.53.99	192.168.226.101	TCP	68	56584 → 23 [ACK] Seq=1828932558 Ack=356166334 Win=29312
25	2020-12-16 21:16:34.8214938…	192.168.53.99	192.168.226.101	TELNET	143	Telnet Data ...
26	2020-12-16 21:16:34.8219541…	192.168.226.101	192.168.53.99	TCP	68	23 → 56584 [ACK] Seq=356166334 Ack=1828932633 Win=29056
27	2020-12-16 21:16:34.8219696…	192.168.226.101	192.168.53.99	TCP	68	[TCP Dup ACK 26#1] 23 → 56584 [ACK] Seq=356166334 Ack=18
28	2020-12-16 21:16:34.8221938…	192.168.226.101	192.168.53.99	TELNET	71	Telnet Data ...
29	2020-12-16 21:16:34.8222007…	192.168.226.101	192.168.53.99	TCP	71	[TCP Retransmission] 23 → 56584 [PSH, ACK] Seq=356166334
30	2020-12-16 21:16:35.0404933…	192.168.61.138	192.168.61.139	UDP	96	12345 → 12345 Len=52
31	2020-12-16 21:16:35.0409664…	192.168.61.138	192.168.61.139	UDP	99	12345 → 12345 Len=55
32	2020-12-16 21:16:35.0423879…	192.168.61.139	192.168.61.138	UDP	99	12345 → 12345 Len=55
33	2020-12-16 21:16:35.2617344…	192.168.53.99	192.168.226.101	TELNET	71	Telnet Data ...
34	2020-12-16 21:16:35.2668550…	192.168.226.101	192.168.53.99	TELNET	71	Telnet Data ...
35	2020-12-16 21:16:35.2668731…	192.168.226.101	192.168.53.99	TCP	71	[TCP Retransmission] 23 → 56584 [PSH, ACK] Seq=356166337
36	2020-12-16 21:16:35.4816889…	192.168.61.138	192.168.61.139	UDP	99	12345 → 12345 Len=55

flow

应用程序（Host U） --> tun （Host U） --> ens33 （Host U） --> ens33 （Server） --> socket （Server） --> ens38 （Server） --> ens33 （Host V） --> 应用程序 （Host V） --> ens33 （Host V） -->ens38 （Server） --> tun （Server） --> ens33 （Server） --> ens33 （Host U） -->socket （Host U） -->tun （Host U） -->应用程序 （Host U）

Task 6: Tunnel-Breaking Experiment

当断开tun连接时，在telnet中输入的字符将不可见。重新建立tun连接后，输入的字符将会出现。

观察wireshark抓包记录。

14	2020-12-16 21:27:50.1513074…	127.0.0.1	127.0.0.1
15	2020-12-16 21:27:50.1513106…	127.0.0.1	127.0.0.1
16	2020-12-16 21:27:50.1532621…	127.0.0.1	127.0.0.1
17	2020-12-16 21:27:50.1532654…	127.0.0.1	127.0.0.1
18	2020-12-16 21:27:50.1553562…	127.0.0.1	127.0.0.1
19	2020-12-16 21:27:50.1553590…	127.0.0.1	127.0.0.1
20	2020-12-16 21:27:50.1573584…	127.0.0.1	127.0.0.1
21	2020-12-16 21:27:50.1573614…	127.0.0.1	127.0.0.1
22	2020-12-16 21:27:50.1593274…	127.0.0.1	127.0.0.1
23	2020-12-16 21:27:50.1593324…	127.0.0.1	127.0.0.1
24	2020-12-16 21:27:50.1612183…	127.0.0.1	127.0.0.1
25	2020-12-16 21:27:50.1612208…	127.0.0.1	127.0.0.1
26	2020-12-16 21:27:50.1632666…	127.0.0.1	127.0.0.1
27	2020-12-16 21:27:50.1632689…	127.0.0.1	127.0.0.1

可以观察到在tun连接断开期间，产生了大量tcp重传。这暗示我们tcp连接并没有断开。

Task 7: Routing Experiment on Host V

```
[12/16/20]seed@VM:~$ sudo ip route add 192.168.53.0/24 dev ens33 via 192.168.226
.1
[12/16/20]seed@VM:~$ ip route list
169.254.0.0/16 dev ens33  scope link  metric 1000
192.168.53.0/24 via 192.168.226.1 dev ens33
192.168.226.0/24 dev ens33  proto kernel  scope link  src 192.168.226.101  metri
c 100
```

`sudo ip route add 192.168.53.0/24 dev ens33 via 192.168.226.1`

Task 8: Experiment with the TUN IP Address

Where are the packets dropped?

Server

	Time	Source	Destination	Protocol	Length	Info
1	2020-12-16 21:41:24.8919366…	::1	::1	UDP	64	56810 → 60757 Len=0
2	2020-12-16 21:41:26.9016027…	192.168.61.139	192.168.61.138	UDP	128	12345 → 12345 Len=84
3	2020-12-16 21:41:27.1256353…	192.168.30.99	192.168.226.101	ICMP	100	Echo (ping) request id=0x3335, seq=1/256, ttl=64 (no res
4	2020-12-16 21:41:27.9151846…	192.168.61.139	192.168.61.138	UDP	128	12345 → 12345 Len=84
5	2020-12-16 21:41:28.1302946…	192.168.30.99	192.168.226.101	ICMP	100	Echo (ping) request id=0x3335, seq=2/512, ttl=64 (no res
6	2020-12-16 21:41:28.9402636…	192.168.61.139	192.168.61.138	UDP	128	12345 → 12345 Len=84
7	2020-12-16 21:41:29.1575879…	192.168.30.99	192.168.226.101	ICMP	100	Echo (ping) request id=0x3335, seq=3/768, ttl=64 (no res
8	2020-12-16 21:41:32.0431923	Vmware_a3:8a:e6		ARP	62	Who has 192.168.61.138? Tell 192.168.61.139

Host V

	Time	Source	Destination	Protocol	Length	Info
1	2020-12-16 18:55:53.3932580…	192.168.30.99	192.168.226.101	ICMP	100	Echo (ping) request id=0x3335, seq=1/256, ttl=
2	2020-12-16 18:55:54.3977751…	192.168.30.99	192.168.226.101	ICMP	100	Echo (ping) request id=0x3335, seq=2/512, ttl=
3	2020-12-16 18:55:55.4256430…	192.168.30.99	192.168.226.101	ICMP	100	Echo (ping) request id=0x3335, seq=3/768, ttl=

可以观察到Server收到了数据并且进行了转发，Host V也收到了数据，但是并没有进行回应

Why are the packets dropped?

这是因为一个叫做“反向路径过滤”的机制存在。

```
The default behavior of Linux is to consider asymmetric routing suspicious and
therefore to drop any packet whose source IP address is not reachable through the
device the packet was received from, according to the routing table.
——《Understanding Linux Network Internals》
```

在本案例中，Host V反向查找路由，发现收到的包的上一跳路由器和响应包所对应的默认路由器不是同一个，反向路径检查失败，所以抛弃了回应包。

How to solve this problem?

在Host V上增加一条路由记录：

`sudo ip route add 192.168.30.0/24 via 192.168.226.1`

Task 9: Experiment with the TAP Interface

Ping

```
Ether / ARP who has 192.168.53.6 says 192.168.53.99
Ether / IP / UDP / DNS Qry "b'_nfs._tcp.local.'"
Ether / ARP who has 192.168.53.6 says 192.168.53.99
Ether / ARP who has 192.168.53.6 says 192.168.53.99
Ether / IPv6 / UDP / DNS Qry "b'_nfs._tcp.local.'"
Ether / ARP who has 192.168.53.6 says 192.168.53.99
Ether / ARP who has 192.168.53.6 says 192.168.53.99
Ether / ARP who has 192.168.53.6 says 192.168.53.99
Ether / ARP who has 192.168.53.6 says 192.168.53.99
Ether / ARP who has 192.168.53.6 says 192.168.53.99
Ether / ARP who has 192.168.53.6 says 192.168.53.99
```

显示了ARP包的内容：`who has 192.168.53.6 says 192.168.53.99`。

这是因为当主机需要向某个同子网的主机发送数据包时，会广播ARP包请求该主机响应其mac地址，这样才能正确将数据包按照mac地址导向。

Spoof ARP

arping -I Guan0 192.168.53.33

```
[12/16/20]seed@VM:~/.../vpn tunneling$ arping -I Guar
ARPING 192.168.53.33 from 192.168.53.99 Guan0
Unicast reply from 192.168.53.33 [00:0C:29:A3:8A:E6]
Unicast reply from 192.168.53.33 [00:0C:29:A3:8A:E6]
Unicast reply from 192.168.53.33 [00:0C:29:A3:8A:E6]
Unicast reply from 192.168.53.33 [00:0C:29:A3:8A:E6]
```

```
Ether / ARP who has 192.168.53.33 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:a3:8a:e6 says 192.168.53.33
---------------------------------
---------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:a3:8a:e6 says 192.168.53.33
---------------------------------
```

arping -I Guan0 1.2.3.4