Lab1

57118233 周厚霖

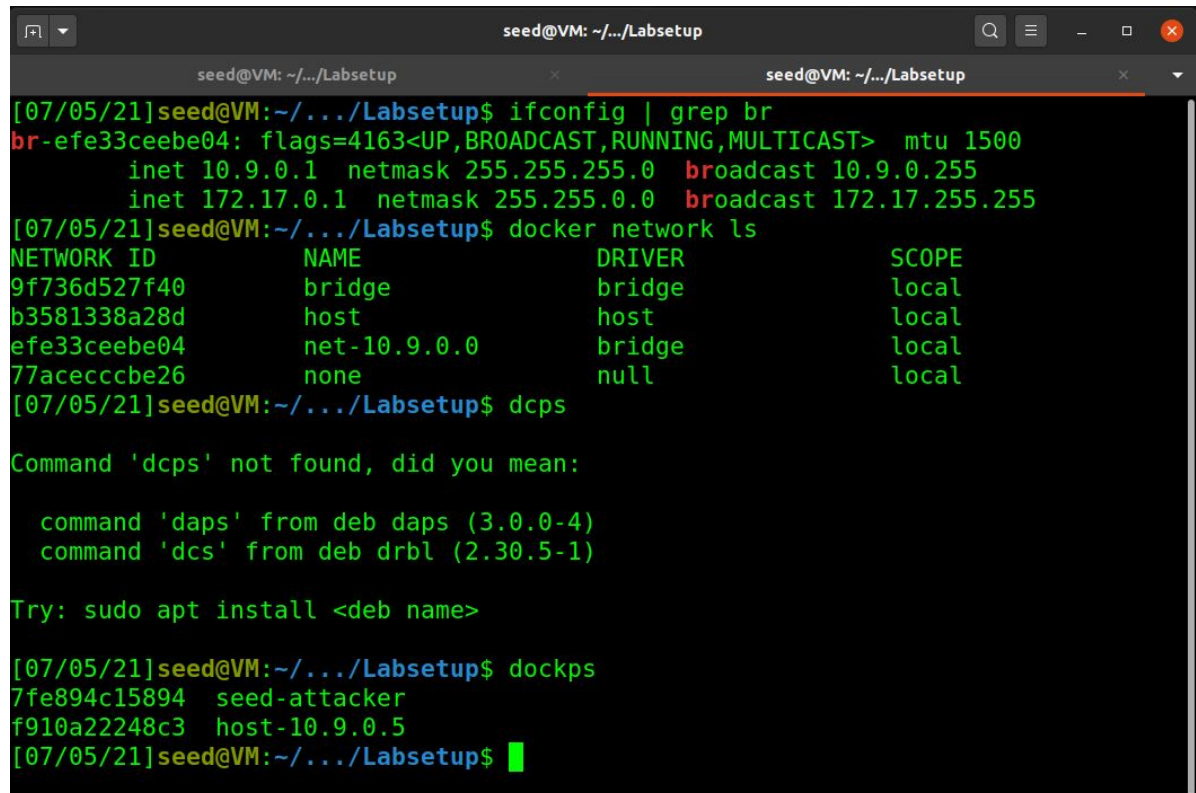## Task 1.1：Sniffing Packets

Task 1.1A

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

def print_pkt(pkt):
    pkt.show()

pkt = sinff(iface='br-efe33ceebe04', filter='icmp', prn=print_pkt)
```

启动docker后，查看网络ID。

以root权限运行sniffer.py，新开一个命令行对主机IP进行ping命令。

以seed用户运行sniffer.py时，系统会报错。

```
00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,
-./01234567'

^C[07/05/21]seed@VM:~/Desktop$
[07/05/21]seed@VM:~/Desktop$ su seed
Password:
[07/05/21]seed@VM:~/Desktop$ python3 sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 7, in <module>
    pkt = sniff(iface='br-efe33ceebe04', filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1036, in
 sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 906, in
_run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 398, i
n __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(typ
e))  # noqa: E501
  File "/usr/lib/python3.8/socket.py", line 231, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
[07/05/21]seed@VM:~/Desktop$
```

Task 1.1B

只抓取ICMP报文，见Task 1.1A所示。

捕获任何来自特定IP的TCP数据包，目的端口为23。

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-efe33ceebe04', filter='tcp port 23 and host 10.9.0.5',
prn=print_pkt)
```

利用docksh获取host的shell，telnet任意一个IP地址建立连接。

在另一处可看到tcp数据包。

捕获来自或去特定子网的数据包。可以选择任何子网，如128.230.0.0/16；不应该选择VM所绑定的子网。

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

def print_pkt(pkt):
    pkt.show()


pkt = sniff(iface='br-13a5b79724e2', filter='host 10.9.0.8', prn=print_pkt)
```

直接ping 10.9.0.8，可得到捕获的数据包。

Task 1.2：Spoofing ICMP Packets

```python
#!usr/bin/evn python3
from scapy.all import *
a = IP()
a.dst = '10.9.0.3'
b = ICMP()
p = a/b
send(p)
ls(a)
```

第一行创建了一个ICMP对象，默认类型为echo request。在第六行中，我们将a和b堆叠在一起形成了一个新对象，"/"操作符被重载，不在表示除法，而是将b添加为a的有效负载字段，并相应地修改a的字段。最终我们得到一个表示ICMP数据包的新对象，报文重组后，向子网内的一个IP发送数据包，打开Wireshark可观测发送数据包和响应数据包。

```
[07/05/21]seed@VM:~/Desktop$ sudo python3 spoofer.py
.
Sent 1 packets.
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             = '10.9.0.1'     (None)
dst       : DestIPField               = '10.9.0.5'     (None)
options   : PacketListField           = []             ([])
[07/05/21]seed@VM:~/Desktop$
```


Task 1.3: Traceroute

```python
#!usr/bin/evn python3
from scapy.all import *
a = IP()
b = ICMP()
a.dst = '1.2.3.4'
for i in range(30):
    a.ttl = i + 1
    p = a / b
    send(p)
```

创建一个文件trace.py，向目标IP发送 ICMP 数据包，一开始设置 TTL (Time-To-Live)值 为 1, 那么发出的 ICMP 数据包在经历一个路由结点后, 就会失活被抛弃,我们利用循环, 不断增加TTL的值，最终使得数据包到达目的地。

```
[07/05/21]seed@VM:~/Desktop$ touch trace.py
[07/05/21]seed@VM:~/Desktop$ vi trace.py
[07/05/21]seed@VM:~/Desktop$ sudo python3 trace.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

从Wireshark中我们可以观察到，途径的IP有172.20.10.1，172.20.73.13，172.18.0.5，最后到达1.2.3.4，即目的地。

Task 1.4: Sniffing and-then Spoofing
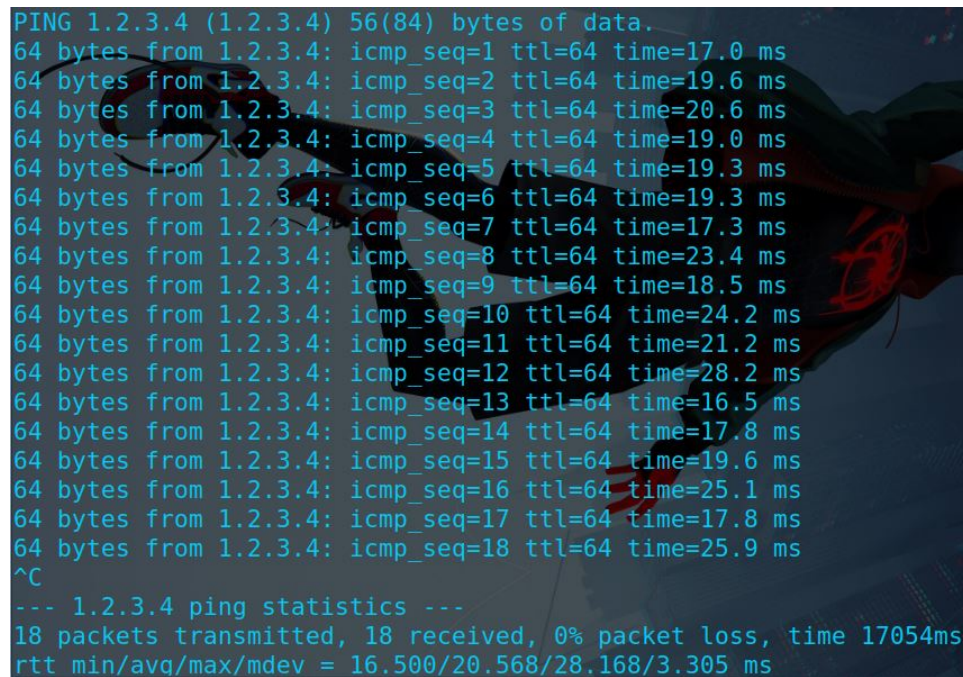
```python
#!/usr/bin/evn python3
from scapy.all import *

def spoof_pkt(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load
        newpkt = ip/icmp/data
        send(newpkt)

pkt = sniff(filter='icmp', prn=spoof_pkt)
```
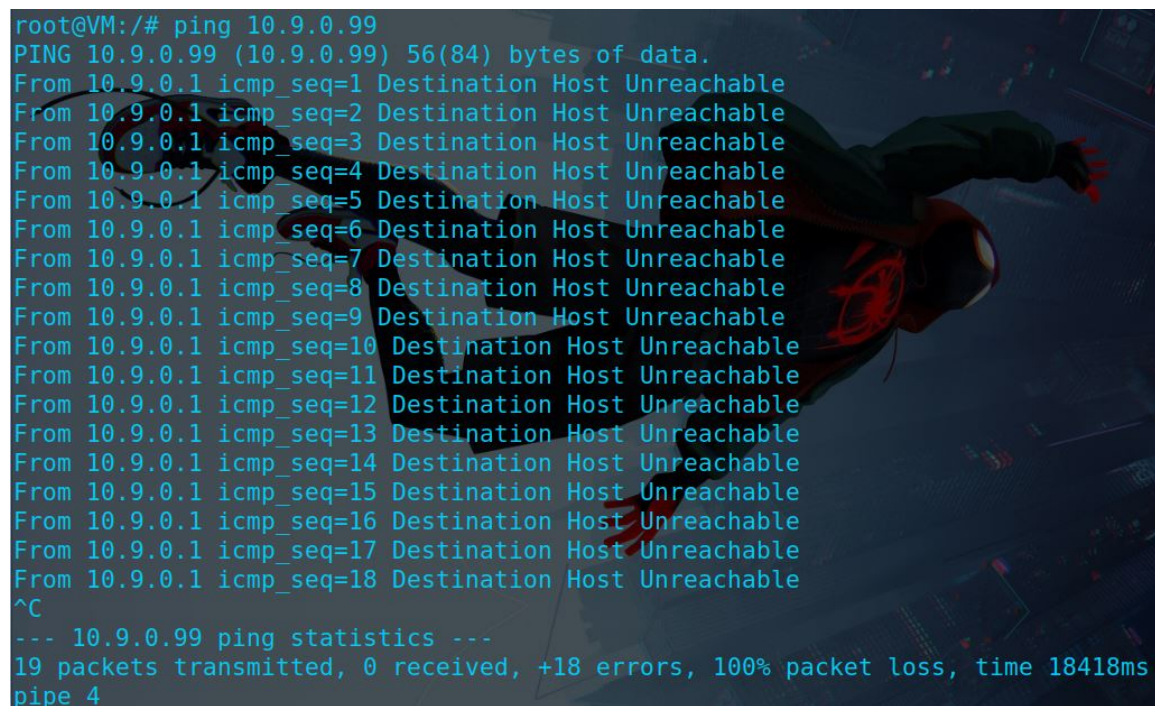
ping 1.2.3.4 # a non-existing host on the Internet

```
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=17.0 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=19.6 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=20.6 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=19.0 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=19.3 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=19.3 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=17.3 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=23.4 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=18.5 ms
64 bytes from 1.2.3.4: icmp_seq=10 ttl=64 time=24.2 ms
64 bytes from 1.2.3.4: icmp_seq=11 ttl=64 time=21.2 ms
64 bytes from 1.2.3.4: icmp_seq=12 ttl=64 time=28.2 ms
64 bytes from 1.2.3.4: icmp_seq=13 ttl=64 time=16.5 ms
64 bytes from 1.2.3.4: icmp_seq=14 ttl=64 time=17.8 ms
64 bytes from 1.2.3.4: icmp_seq=15 ttl=64 time=19.6 ms
64 bytes from 1.2.3.4: icmp_seq=16 ttl=64 time=25.1 ms
64 bytes from 1.2.3.4: icmp_seq=17 ttl=64 time=17.8 ms
64 bytes from 1.2.3.4: icmp_seq=18 ttl=64 time=25.9 ms
^C
--- 1.2.3.4 ping statistics ---
18 packets transmitted, 18 received, 0% packet loss, time 17054ms
rtt min/avg/max/mdev = 16.500/20.568/28.168/3.305 ms
```

可以看到当我们ping一个网络上不存在的IP时，由于伪造报文，我们仍可以接收到响应。
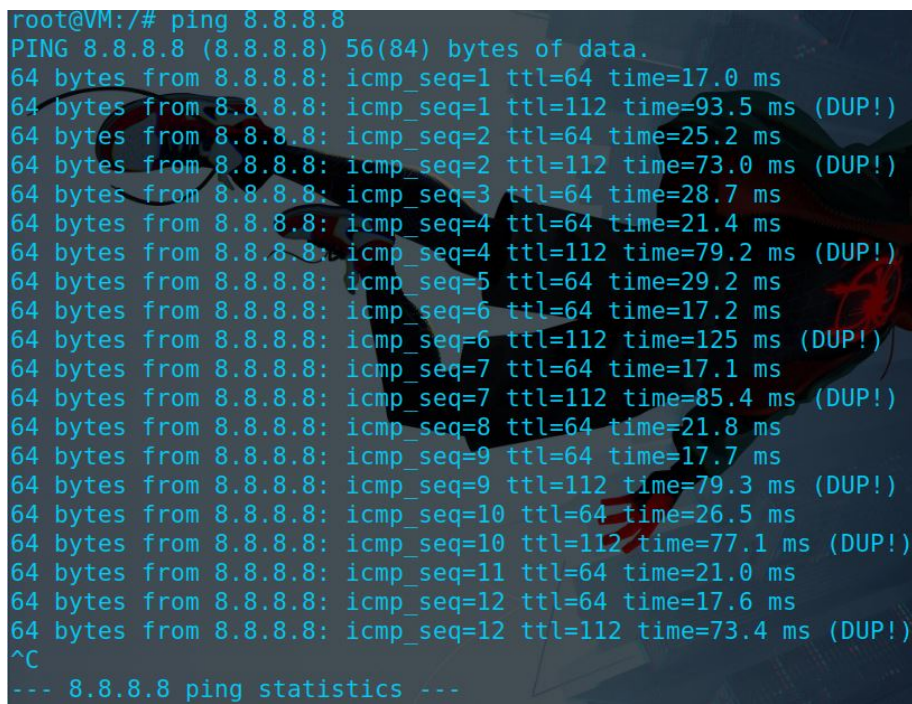
ping 10.9.0.99 # a non-existing host on the LAN

```
root@VM:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Host Unreachable
From 10.9.0.1 icmp_seq=2 Destination Host Unreachable
From 10.9.0.1 icmp_seq=3 Destination Host Unreachable
From 10.9.0.1 icmp_seq=4 Destination Host Unreachable
From 10.9.0.1 icmp_seq=5 Destination Host Unreachable
From 10.9.0.1 icmp_seq=6 Destination Host Unreachable
From 10.9.0.1 icmp_seq=7 Destination Host Unreachable
From 10.9.0.1 icmp_seq=8 Destination Host Unreachable
From 10.9.0.1 icmp_seq=9 Destination Host Unreachable
From 10.9.0.1 icmp_seq=10 Destination Host Unreachable
From 10.9.0.1 icmp_seq=11 Destination Host Unreachable
From 10.9.0.1 icmp_seq=12 Destination Host Unreachable
From 10.9.0.1 icmp_seq=13 Destination Host Unreachable
From 10.9.0.1 icmp_seq=14 Destination Host Unreachable
From 10.9.0.1 icmp_seq=15 Destination Host Unreachable
From 10.9.0.1 icmp_seq=16 Destination Host Unreachable
From 10.9.0.1 icmp_seq=17 Destination Host Unreachable
From 10.9.0.1 icmp_seq=18 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
19 packets transmitted, 0 received, +18 errors, 100% packet loss, time 18418ms
pipe 4
```

对于局域网内不存在的主机，先利用ARP进行MAC地址询问，由于一直得不到结果，所以没有ICMP报文，也就不存在报文欺骗。

ping 8.8.8.8 # an existing host on the Internet

```
root@VM:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=17.0 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=112 time=93.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=25.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=112 time=73.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=28.7 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=21.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=112 time=79.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=29.2 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=64 time=17.2 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=112 time=125 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=7 ttl=64 time=17.1 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=112 time=85.4 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=8 ttl=64 time=21.8 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=64 time=17.7 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=112 time=79.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=10 ttl=64 time=26.5 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=112 time=77.1 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=11 ttl=64 time=21.0 ms
64 bytes from 8.8.8.8: icmp_seq=12 ttl=64 time=17.6 ms
64 bytes from 8.8.8.8: icmp_seq=12 ttl=112 time=73.4 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
```

对于网络上存在的主机，我们可以看到每个序列号的报文都存在一个重复报文，我们可以知道TTL=64，且时间较短的那个报文是伪造的报文。