Chapter 1: Packet Sniffing and Spoofing

57118132 吴嘉琪

852809049@qq.com

Southeast University — 2021 年 7 月 8 日

Task 1.1:Sniffing Packets

The objective of this task is to learn how to use Scapy to do packet sniffing in Python programs.

Getting the network interface name. When we use the provided Compose file to create containers for this lab, a new network is created to connect the VM and the containers. The IP prefix for this network is 10.9.0.0/24, which is specified in the docker-compose.yml file. The IP address assigned to our VM is 10.9.0.1. We need to find the name of the corresponding network interface on our VM, because we need to use it in our programs. The interface name is the concatenation of br- and the ID of the network created by Docker. When we use ifconfig to list network interfaces, we will see quite a few. Look for the IP address 10.9.0.1.

```
$ ifconfig
br-e82d66e7967a: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
```

Tast 1.1A The code above will sniff the packets on the br-c93733e9f913 interface.

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

def print_pkt(pkt):
    pkt.show()
pkt = sniff(iface='br-e82d66e7967a',filter='icmp', prn=print_pkt)
```

Run the program with the root privilege and demonstrate that you can indeed capture packets.Then do the following

```
$ ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.083 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.095 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.068 ms
```

The experimental results are

```
$ sudo python3 sniffer.py
###[ Ethernet ]###
```

```
  dst       = 02:42:0a:09:00:05
  src       = 02:42:72:f8:fc:62
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 19340
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xdb05
     src       = 10.9.0.1
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xa4e7
        id        = 0x1
        seq       = 0x1
```

run the program again, but without using the root privilege.The experimental results are

```
$ python3 sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 13, in <module>
    pkt = sniff(filter='icmp', prn=print_pkt)
        ......
PermissionError: [Errno 1] Operation not permitted
```

Tast 1.2A

1.Capture only the ICMP packet

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

def print_pkt(pkt):
    pkt.show()
pkt = sniff(iface='br-e82d66e7967a',filter='icmp', prn=print_pkt)
```

Run the program with the root privilege and demonstrate that you can indeed capture packets.Then do the following

```
$ ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.083 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.095 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.068 ms
```

The experimental results are

```
$ sudo python3 sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:72:f8:fc:62
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 19340
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xdb05
     src       = 10.9.0.1
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xa4e7
        id        = 0x1
        seq       = 0x1
```

2.Capture any TCP packet that comes from a particular IP and with a destination port number 23

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

def print_pkt(pkt):
    pkt.show()
pkt = sniff(filter='tcp and src host 10.9.0.1 and dst port 23', prn=print_pkt)
```

Run the program with the root privilege and demonstrate that you can indeed capture packets.Then do the following

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

ip=IP()
ip.src='10.9.0.1'
ip.dst='10.9.0.5'
tcp=TCP()
tcp.dport=23
send(ip/tcp)
```

The experimental results are

```
$ sudo python3 sender.py.
Sent 1 packets.

$ sudo python3 sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:72:f8:fc:62
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 40
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x66b8
     src       = 10.9.0.1
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
        sport     = ftp_data
        dport     = telnet
        seq       = 0
        ack       = 0
        dataofs   = 5
        reserved  = 0
```

```
        flags     = S
        window    = 8192
        chksum    = 0x7ba0
        urgptr    = 0
        options   = []
```

3.Capture packets comes from or to go to a particular subnet. Y ou can pick any subnet, such as128.230.0.0/16; you should not pick the subnet that your VM is attached to.

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

def print_pkt(pkt):
    pkt.show()
pkt=sniff(filter='dst net 10.9.0.0/24',prn=print_pkt)
```

Run the program with the root privilege and demonstrate that you can indeed capture packets.Then do the following

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

send(IP(dst='10.9.0.0/24'))
```

The experimental results are

```
$ sudo python3 sender.py.
Sent 1 packets.

$ sudo sudo python3 sniffer.py
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 02:42:72:f8:fc:62
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = 02:42:72:f8:fc:62
     psrc      = 10.9.0.1
     hwdst     = 00:00:00:00:00:00
     pdst      = 10.9.0.0

###[ Ethernet ]###
```

```
  dst       = ff:ff:ff:ff:ff:ff
  src       = 02:42:72:f8:fc:62
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 20
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = hopopt
     chksum    = 0x66d7
     src       = 10.9.0.1
     dst       = 10.9.0.0
     \options   \

###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 02:42:72:f8:fc:62
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = 02:42:72:f8:fc:62
     psrc      = 10.9.0.1
     hwdst     = 00:00:00:00:00:00
     pdst      = 10.9.0.2

###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 02:42:72:f8:fc:62
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 20
```

```
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = hopopt
     chksum    = 0x66d5
     src       = 10.9.0.1
     dst       = 10.9.0.2
     \options   \

###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 02:42:72:f8:fc:62
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = 02:42:72:f8:fc:62
     psrc      = 10.9.0.1
     hwdst     = 00:00:00:00:00:00
     pdst      = 10.9.0.3

###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 02:42:72:f8:fc:62
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 20
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = hopopt
     chksum    = 0x66d4
     src       = 10.9.0.1
     dst       = 10.9.0.3
```

```
    \options    \
```

Task 1.2:Spoofing ICMP Packets

Run the program with the root privilege and demonstrate that you can indeed capture packets.Then do the following

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

send(IP(dst='10.9.0.5')/ICMP())
```

Then see packets back with Wireshark:

	Time	Source	Destination	Protocol	Length	Info
1	2021-07-05 12:1…	02:42:72:f8:fc:62	Broadcast	ARP	42	Who has 10.9.0.5? Tell
2	2021-07-05 12:1…	02:42:0a:09:00:05	02:42:72:f8:fc:62	ARP	42	10.9.0.5 is at 02:42:0
3	2021-07-05 12:1…	10.9.0.1	10.9.0.5	ICMP	42	Echo (ping) request i
4	2021-07-05 12:1…	10.9.0.5	10.9.0.1	ICMP	42	Echo (ping) reply i
5	2021-07-05 12:1…	fe80::42:72ff:fef8:…	ff02::fb	MDNS	180	Standard query 0x0000
6	2021-07-05 12:1…	10.9.0.1	224.0.0.251	MDNS	160	Standard query 0x0000

Task 1.3:Traceroute

Run the program with the root privilege

```
from scapy.all import *

ans,unans=sr(IP(dst='www.souhu.com', ttl=(4,100))/TCP(flags=0x2))
for snd,rcv in ans:
        print(snd.ttl, rcv.src, isinstance(rcv.payload,TCP))
```

The experimental results are

```
$ sudo python3 tarcerout.py
Begin emission:
Finished sending 97 packets.
.*******^C
Received 8 packets, got 7 answers, remaining 90 packets
4 202.97.95.138 False
5 219.158.6.245 False
6 219.158.9.37 False
7 219.158.7.130 False
8 219.158.8.186 False
9 47.74.46.59 True
10 47.74.46.59 True
```

Task 1.4:Sniffing and-then Spoofing

Run the program with the root privilege

```
from scapy.all import*
```

```
def print_pkt(pkt):
        a=IP(src=pkt[IP].dst,dst=pkt[IP].src)
        b=ICMP(type='echo-reply',code=0,id=pkt[ICMP].id,seq=pkt[ICMP].seq)
        c=pkt[Raw].load
        send(a/b/c)
pkt=sniff(filter='icmp[icmptype]==icmp-echo',prn=print_pkt)
```

Then ping 1.2.3.4, a non-existing host on the Internet, and the result is

```
$ ping 1.2.3.4
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=17.7 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=20.2 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=18.3 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=21.3 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=20.1 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=19.0 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=17.2 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=23.5 ms
```

Then ping 10.9.0.99, a non-existing host on the LAN. The address in the network segment does not pass through the network interface, can not be sniffed and sent spoofing message, so the address is unreachable,and the result is

```
$ 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
```

Then ping 8.8.8.8, an existing host on the Internet.In addition to the forged reply message, the real message returned is also received, so the duplicate message is displayed, And the result is

```
$ 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=15.9 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=111 time=112 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=17.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=17.4 ms
```

```
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=17.3 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=17.2 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=111 time=127 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=6 ttl=64 time=17.3 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=111 time=61.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=7 ttl=64 time=25.4 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=64 time=25.0 ms
```