Lab3：Packet Sniffing and Spoofing Lab

57117219尚林灏

Task1

root权限运⾏程序后

###[ 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

qname = 'detectportal.firefox.com.' qtype = A qclass = IN

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.'

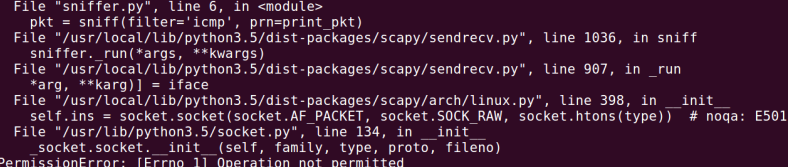
type = A rclass = IN ttl = 38 rdlen = None rdata = 184.28.98.108

###[ 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

普通⽤户权限运⾏程序报错：



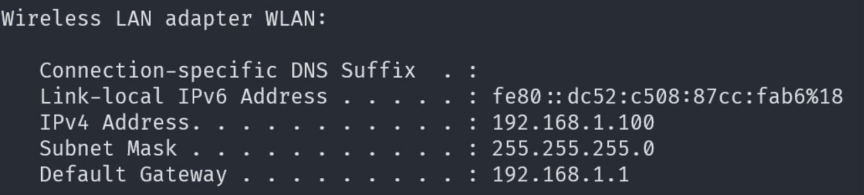
报错的原因在于普通⽤户没有权限创建socket。

捕获ICMP报⽂

filter与原代码⼀致，输出也与上⾯⼀样。

从特定IP发出的，⽬的端⼝为23的TCP包

ipconfig 命令获得宿主机的Windows系统的IP为 192.168.1.100 :



filter写为 "src host 192.168.1.100 and tcp dst port 23"。

ifconfig命令获得虚拟机的Ubuntu系统的IP为 192.168.1.103

在虚拟机中的程序中输出，成功捕获

###[ 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

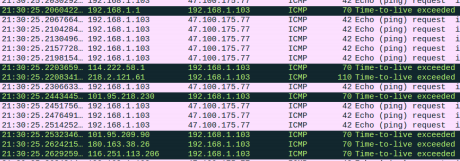
chksum = 0x5ee7

urgptr = 0

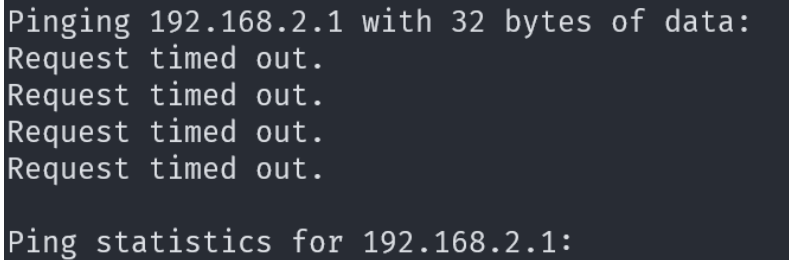
options = [('MSS', 1460), ('NOP', None), ('WScale', 8), ('NOP',

None), ('NOP', None), ('SAckOK', b'')]

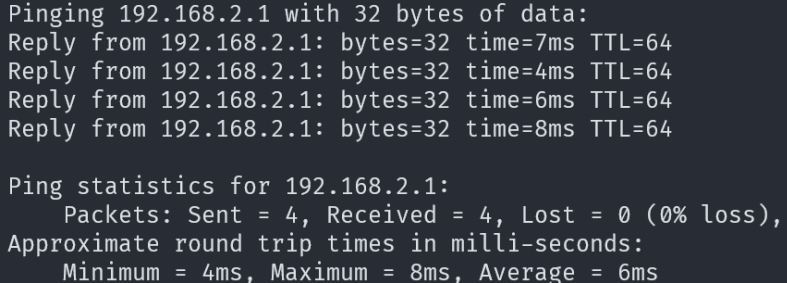
⽤Scapy来估计虚拟机与⽬标地址之间的路由器跳数。通过⼀个⽆限循环，每次将TTL递增，然后使⽤Wireshark查看



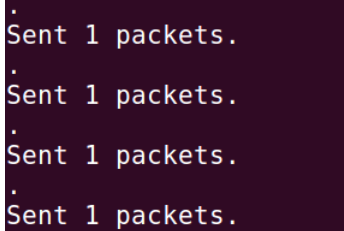
通过捕获ICMP报⽂，将源宿地址对调，设置ICMP类型为Reply，发出后，伪造出ICMP的reply。



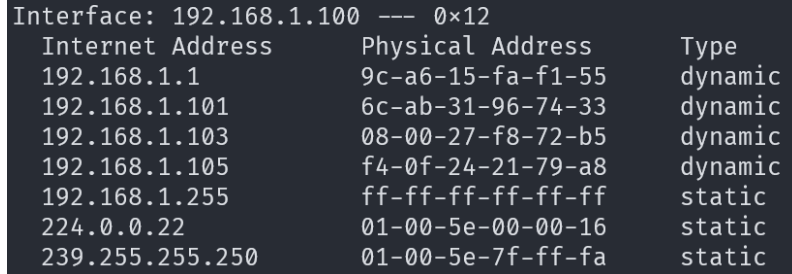
在虚拟机中运⾏脚本，再次在宿主机中进⾏相同的操作



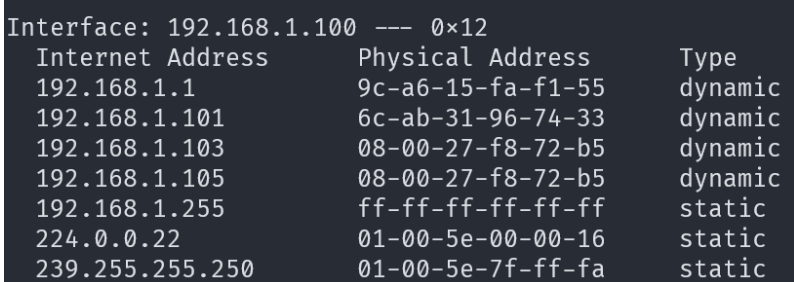
在虚拟机中



在宿主机中使⽤arp -a查看ARP缓存表



向宿主机的IP地址发送ARP请求报⽂，将源地址设为想要污染的IP地址192.168.1.105 。运⾏一段时间后，在宿主机中查看。



实验成功

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)

⼿动将UDP报⽂分⽚，其过程为：⾸先计算UDP报⽂总⻓度，为UDP头部⻓度8字节+载荷96字节，共104字节。第⼀⽚IP报⽂的⽚偏移量 frag 为 0 ， flags 为 1 ，表明接下来还有分⽚。第⼀⽚IP报⽂包含UDP⾸部和前32个字节的载荷。第⼆⽚IP报⽂的⽚偏移量为第⼀⽚IP报⽂载荷/8，也就是5，其余不变，同时不再包含UDP⾸部。第三⽚IP报⽂的⽚偏移量为第⼀、⼆⽚IP报⽂载荷之和/8，也就是9，同时 flags 设置为 0 ，表明后⾯不再有分⽚。然后，在 192.168.1.106 的系统中使⽤sudo nc -lu 9090命令

将第⼆⽚报⽂的⽚偏移量设置为4，第三⽚相应设置为8，UDP报⽂的⻓度相应设置为96，第⼆⽚报⽂的前8个字节与第⼀⽚报⽂的后8个字节重合。将第⼆⽚报⽂的载荷中的A全部改为B。

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)

次运⾏脚本，在服务器中收到的前24个字符是 A ，然后跟着32个 B ，接着是32个 A 。这说明，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。这是因为，内核重组IP报⽂是在获得全部IP报⽂之后才进⾏的。

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。当分⽚总⻓超过 0xFFFF 后，设置其 flags 为 0 。此时，使⽤ nc 架起的UDP服务器崩溃了。改写脚本，不再发送第⼆⽚分⽚，⽽是只发送第⼀⽚、第三⽚分⽚，并不断改变id通过这种⽅案，服务器的内存占⽤急剧升⾼。

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)