# 环境准备

sudo sysctl -w kernel.randomize\_va\_space=0

编译指令为

gcc -fno-stack-protector -z execstack -o test test.c

修改/bin/sh符号链接为zsh

sudo ln -sf /bin/zsh /bin/sh

# 实验

## Task1

编译并修改为setuid程序

gcc -g -DBUF\_SIZE=24 -o stack -z execstack -fno-stack-protector stack.c

sudo chown root stack

sudo chmod 4755 stack

创建badfile文件，供stack读取。

在64位机上编译，需要在gcc指令加上-m32

最终结果



## Task2

执行所给出的python代码。

需要对代码进行修改。

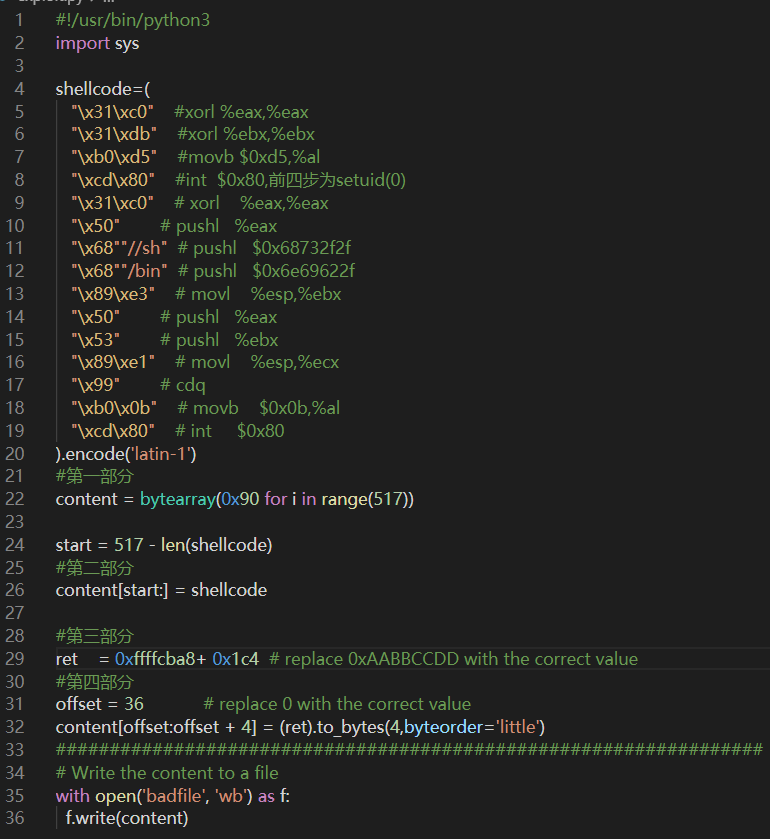
首先是返回地址的确定，即填充的字符串中，哪一部分最终会pop eip。

可以通过pwntools的cyclic确定，最终确定为36。

修改其为我们想要的地址，脚本中将shellcode填充到最末尾，前面用nop指令填充，因此给一个比较高的地址即可。

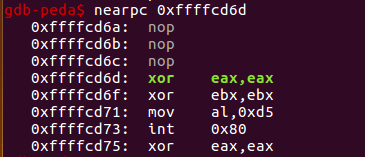
Bof函数中，通过gdb调试确定ebp地址，返回地址比ebp大是必要的，在ebp基础上增加即可。

而后运行，可以拿到shell。



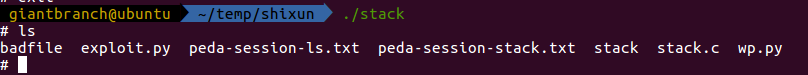
通过调试，获得bof函数的ebp为0xffffcba8。

而后stack命令打印栈，找到代码位置。Shellcode部分不是0x90，比较容易发现大体位置，确定位置需要测试，最终如下图



计算出和前面获得的ebp偏移，取0x1c4为偏移。

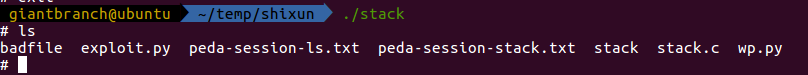
运行即可。



## Task3

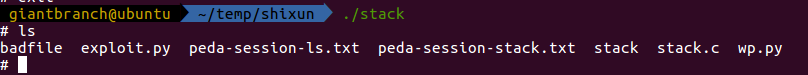
修改stack程序检查更为严格，而后修改python脚本，修改shellcode指令，再次攻击成功。

通过添加修改uid的shellcode来绕过对uid的检查，使得获得的shell具有特权。



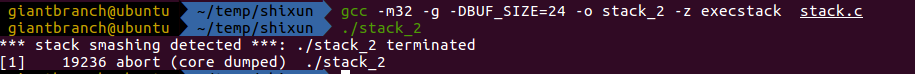
## Task4

开启ASLR，而后不改变脚本，不断运行程序，有概率恰巧脚本中的攻击地址，即我们修改的指令返回地址是可以进行攻击的。



## Task5

开启canary，即stackguard后会报错，因为我们覆盖了canary值，导致程序从栈帧返回的时候。无法通过检测而报错。



这时候就需要对canary值进行泄露，然后覆盖为相同值。

## Task6

栈上代码不可执行后，我们写入shellcode也没有执行权限无法执行，这个时候就要对libc进行攻击，即返回到代码段，并执行代码段的一串代码获得shell。

