“软件安全”实验报告

**.**

**班 级：　　　 网安2104班　　 　.**

**姓名：　　　　　邬雪菲　　　　　.**

**学号：　　　　U202112131　　　　.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **项目** | **撰写规范** | **实验过程** | **问题分析与小结** | **总分** | **教师签字** |
| 分值 | 20 | 50 | 30 | 100 |  |
| 评分 |  |  |  |  |  |

# 目　录

[目　录 I](#_Toc151928771)

[1 Linux平台漏洞攻防工具使用 1](#_Toc151928772)

[1.1 实验目的 1](#_Toc151928773)

[1.2 实验要求 1](#_Toc151928774)

[1.3 实验环境 1](#_Toc151928775)

[1.4 实验过程记录 1](#_Toc151928776)

[2 栈保护机制绕过与漏洞利用 1](#_Toc151928777)1

[2.1 实验目的 1](#_Toc151928778)1

[2.2 实验要求 1](#_Toc151928779)1

[2.3 实验环境 1](#_Toc151928780)1

[2.4 实验过程记录 1](#_Toc151928781)1

[2.实验遇到的难点与问题分析 1](#_Toc151928782)6

[3.实验小结 1](#_Toc151928783)7

[4.课程意见与建议 1](#_Toc151928784)8

# Linux平台漏洞攻防工具使用

## 实验目的

* 掌握Linux进程的原理；
* 了解ELF可执行文件格式与载入原理；
* 掌握知名反汇编工具的工作原理与使用方法；

## 实验要求

* 编写代码独立解析ELF文件，获取ELF文件元信息；
* 需独立使用进程分析工具分析给定ELF的进程地址空间。
* 以小组为单位完成对ELF程序的逆向分析后获取flag

## 实验环境

* 实验平台：pwn.hust.college
* 操作系统：Ubuntu 20.04
* 分析工具：IDA free version

## 实验过程记录

### 实验内容一 编写代码读取ELF元信息

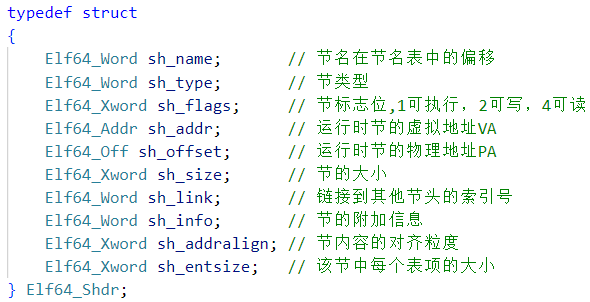
要求不使用外部库或工具实现ELF文件元信息的读取：段在文件中的偏移量、段虚拟地址、文件中段的长度、内存中段的长度、节名、节在执行时的虚拟地址、节在文件中的偏移量、节长度等等。

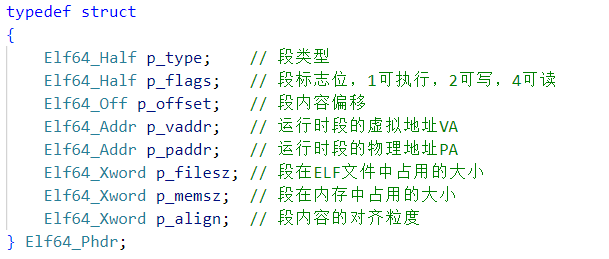
想要读取文件各段和各节的信息，就要从文件头中获取程序头表Program Header Table和节头表Section Header Table的信息。64位系统中ELF文件头为64字节，结构如下所示：



文件头处于ELF文件最开始的位置，通过读取文件头中对应偏移处的内容，可以获得程序头表与节头表的偏移地址、表项大小、表项数，由此即可定位到各个程序头表项和节头表项，进一步读取各段各节的信息。

节头表和程序头表的结构分表如下所示：





由上述的三个结构信息，即可完成ELF元信息的读取。本文使用c++语言来编写程序。

打开文件，读取文件头信息的代码如下：



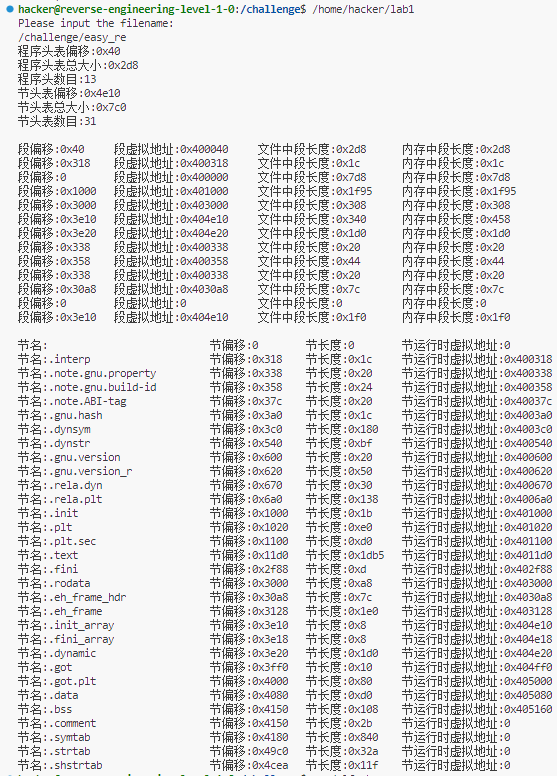
读取程序头表，直接从程序头表偏移处读文件，放入程序头表结构中，根据结构成员即可输出对应的信息，如果需要其他的成员信息直接添加输出即可：



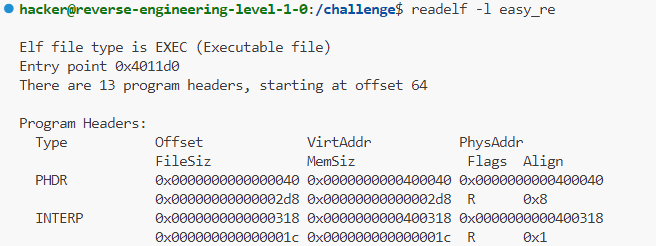
节头表的读取和程序头表类似，不同的是要读取节名，节名并不直接存放在结构中，而是给出该节名在节名表.shstrndx中的偏移，由此间接读取节名：

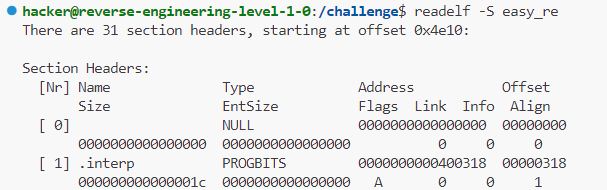


在实验在线平台中，程序的编译运行结果如下：



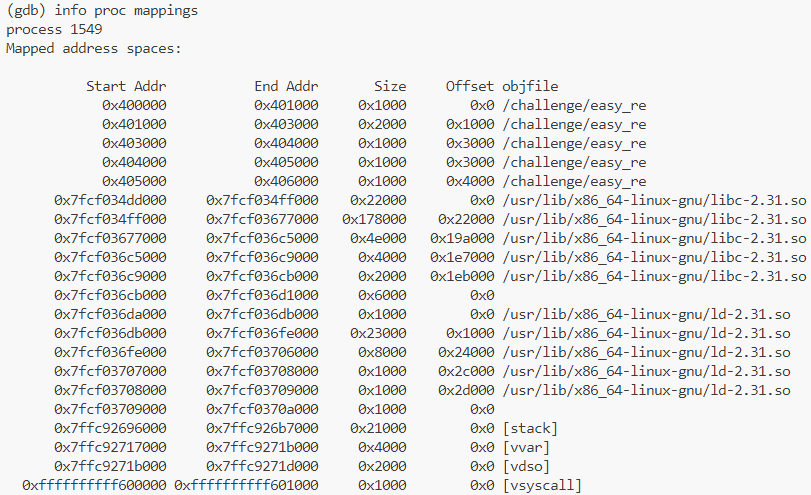
readelf -S可查询节信息，readelf –l 查看段信息，readelf -h可查看ELF文件头信息。使用命令查看ELF文件元信息，与cpp程序得到的结果一致，说明程序设计成功。





### 实验内容二 查看ELF文件进程地址空间

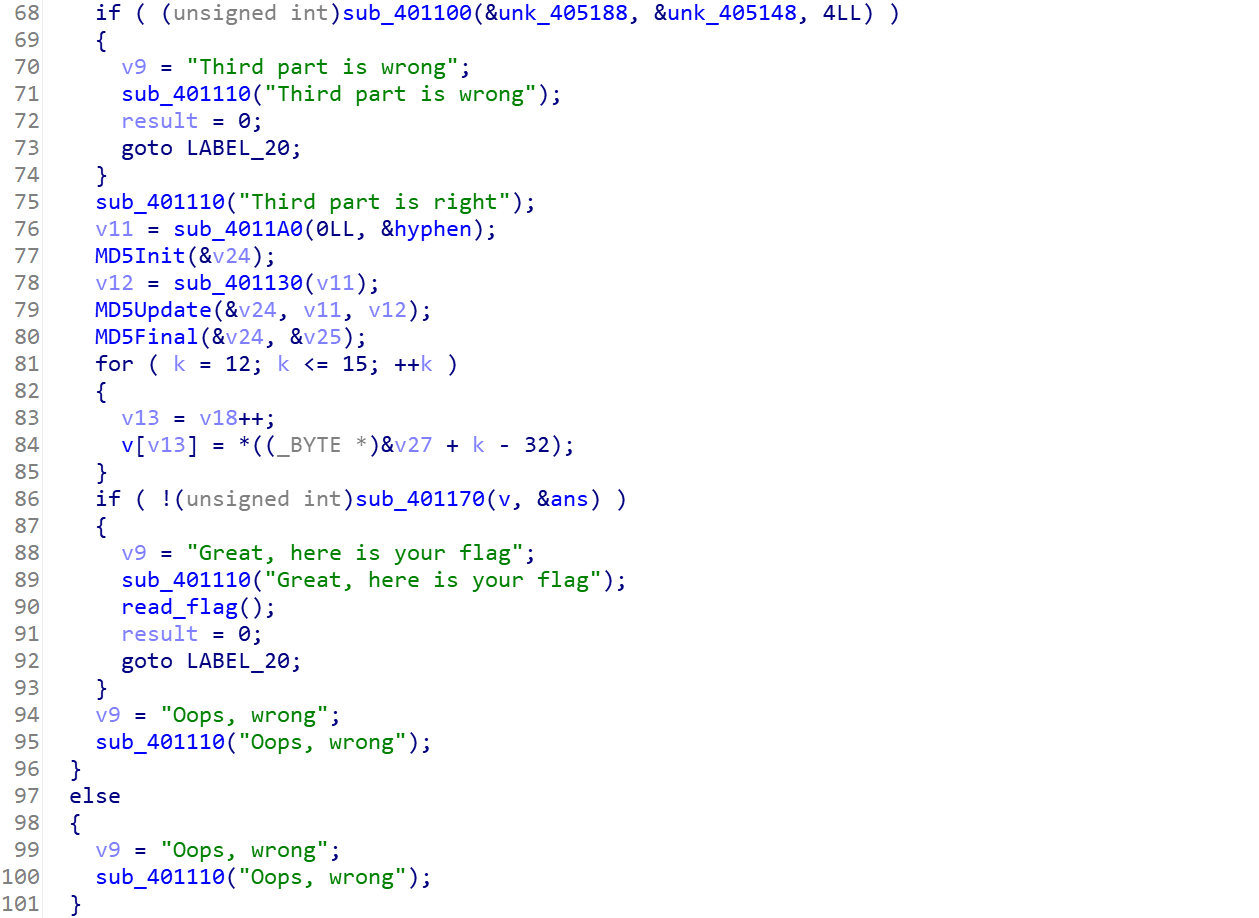
使用gdb调试ELF文件，并使用info proc mappings命令查看进程地址空间如下：



### 实验内容三 逆向分析ELF文件获取license

使用IDA静态分析可执行文件，tab反汇编得到main函数c代码。



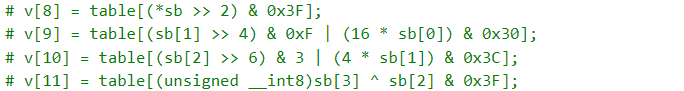


通过反汇编得到的c代码，很容易就能分析清楚程序逻辑。但是也不能过于依赖反汇编得到的c代码，因为这不是源码，有些分析结果可能不是那么地准确，甚至有误导性，而且逆向的过程中常常是根据汇编原码分析。此处观察main函数，并双击跟踪关键函数或变量分析，得知程序首先将输入的字符串由-隔断为4个部分，先对第二部分进行检查，再检查第三部分，然后使用MD5加密第四部分，再将后三个部分的操作结果放入暂存处，最后将四个部分与目标结果比对，一致则输出flag。比较细节的点是程序用到了字符表table，有些操作的结果不直接作为结果字符，而是作为字符表table的索引号。下面是具体分析。

第二部分实际上是一个异或移位操作，每次当前字符的高六位与前一个字符的高六位进行异或操作后与0x3F异或得到字符表的索引值，再取出字符。目标的最终结果为xw0t。两个解题思路，一个是直接由结果逆向，因为异或和移位操作可逆；另一种是直接模拟操作过程穷举。由于只需关注字符的高六位，有多个可行解，加上穷举量很小，本文直接使用python脚本爆破，得到一个可行解为xTpH，代码如下：



然后是第三部分，其实也是一个加密操作，不过比第二个部分的操作要复杂许多，实验指导上提示使用python的z3库约束求解。本文还是习惯采用直接爆破的方式，就没有使用额外的库。目标的字符串为pL97。具体的加密过程如下所示：



此加密过程是递进式的层层相扣，有些输入可能只满足部分字符，要到内层循环才能判断出来，然后返回上一层继续枚举，所以四层循环的穷举量会比第二部分多得多，代码如下：



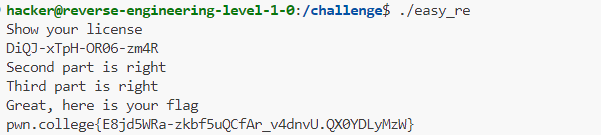
最后分析出第三部分密码为OR06

第四部分思路也是简单的爆破，直接穷举四位密码进行MD5加密，取结果最后8个十六进制位和目标密文串比对即可。由于需要进行MD5加密，耗时稍长了点，几十秒才爆破出结果zm4R，代码如下：





第一部分的比较要到最后和第四部分一起进行，直接在源码中给出了明文，存在ans首部，为DiQJ。所以最终得到一个可行license为DiQJ-xTpH-OR06-zm4R，在平台上测试，成功得到flag，如下所示：



# 栈保护机制绕过与漏洞利用

## 实验目的

* 掌握反汇编代码分析工具分析原理与使用方法；
* 掌握栈溢出和 Stack Canary 的原理；
* 掌握栈溢出漏洞的利用技巧和 Stack Canary 的绕过手法；
* 理解栈溢出漏洞的防范措施；

## 实验要求

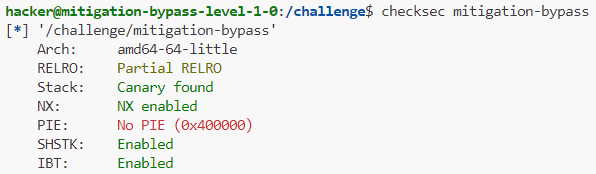
* 调试与掌握函数反汇编执行流程
* 定位程序溢出点
* Stack Canary 爆破绕过，劫持程序控制流；
* 编写 ROP 链，获取具有任意命令执行功能的 Shell

## 实验环境

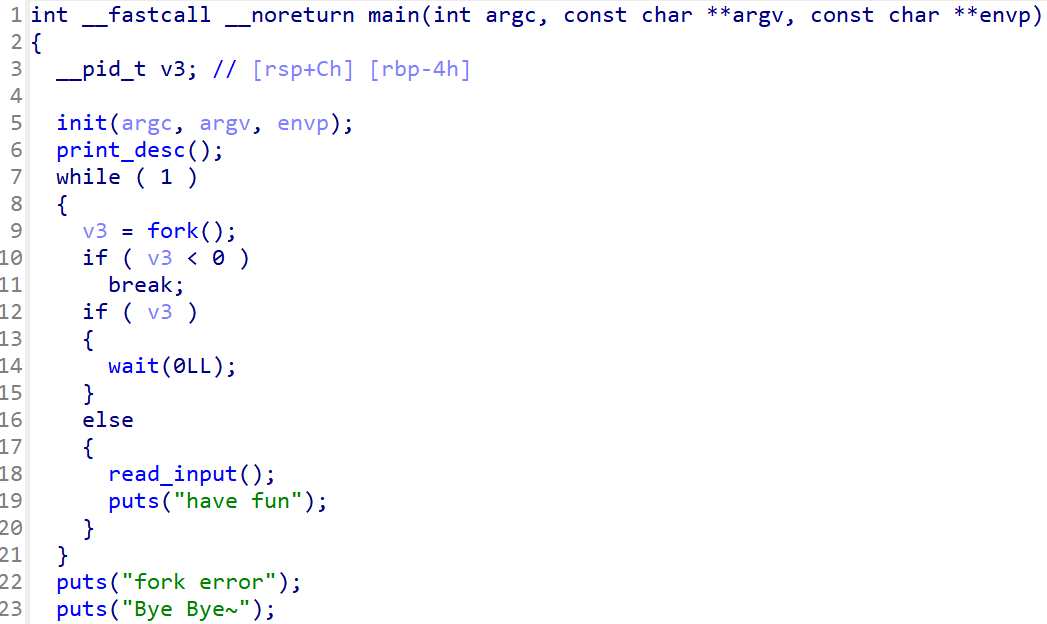
* 实验平台：pwn.hust.college
* 操作系统：Ubuntu 20.04
* 溢出软件：mitigation-bypass
* 溢出工具：pwndbg，vscode，XFCE桌面，bash命令行

## 实验过程记录

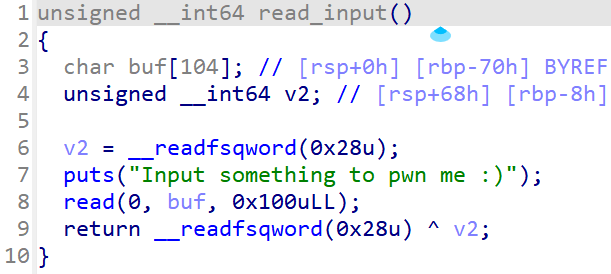
首先查看程序的保护机制，checksec查看到开启了栈保护canary和NX机制

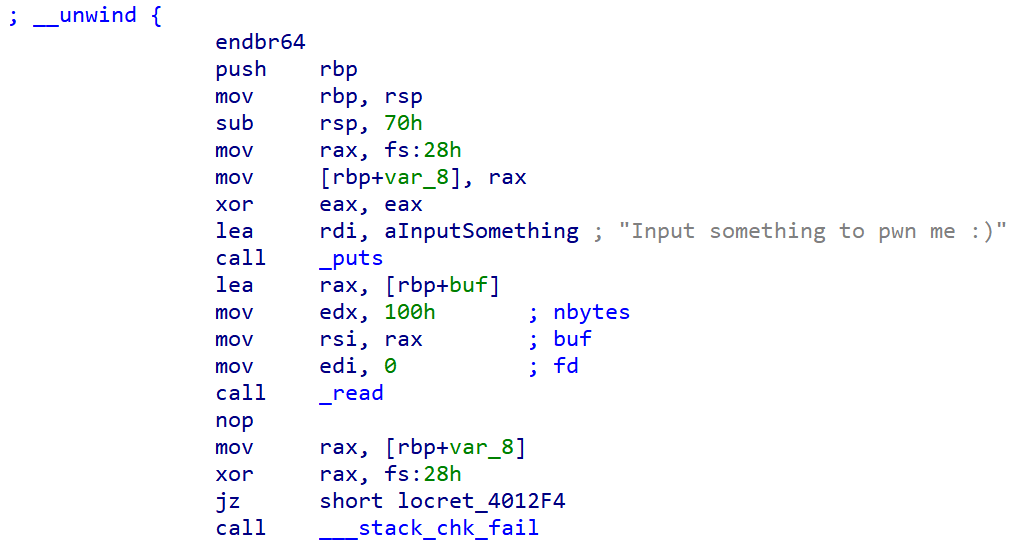


然后分析程序逻辑，使用IDA反汇编为c代码查看，可见程序循环fork子进程，接收用户的输入并返回提示。

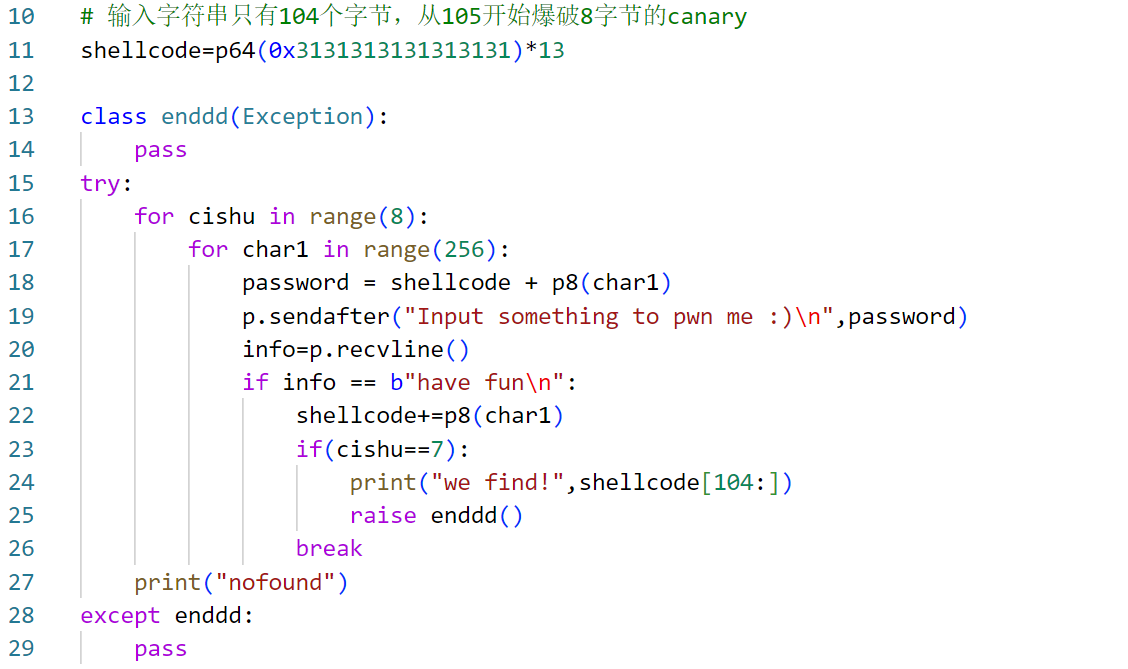


查看read\_input函数，buf大小只有104，但是read函数最多可读入256字节，存在栈溢出漏洞。但是由于开启了canary栈溢出保护，进入read\_input函数时会放置fs:28h处的canary到栈底-8字节处，退出函数时先检查栈上canary是否保持一致，如果变化则报错提示。





由于子进程的canary与父进程一致，且子进程触发栈溢出警告不会终止父进程，所以可以利用循环fork子进程输入的特点逐字节爆破canary，具体代码如下所示：



canary的最低字节一定是\x00，本意是防止被read等函数直接读出来，此处可以不用枚举该字节，不过由于是从0开始枚举所以就没什么关系。

下一步是考虑如何赋予root权限劫持控制流获取flag，由于开启了NX保护，所以无法在栈上写入shellcode执行。那么考虑构造ROP链，修改read函数的返回地址劫持控制流，调用libc动态库中的setuid(0)函数设置root权限并使用system(/bin/sh）获取shell执行。虽然无法确定libc库链接到程序中的基地址，但是每个函数在libc中的相对位置是确定的，可以通过泄露libc基地址加上函数的相对地址获取函数实际地址实现调用。

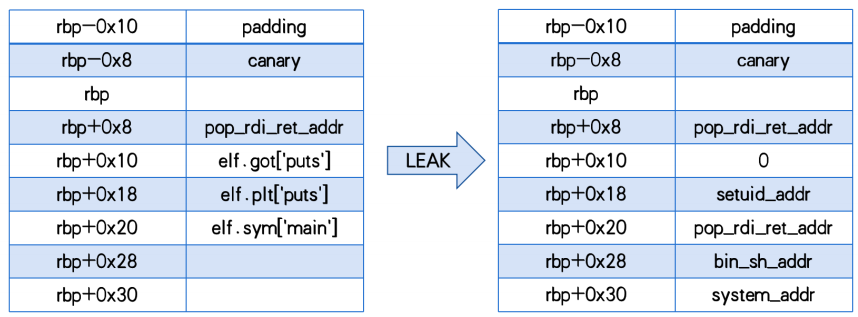
因为本题libc文件已知，所以可以直接使用命令或者pwntools获取具体函数的偏移地址。由理论课中GOT表和PLT的知识可知，当调用动态库中的函数时，指令跳转到plt表查看对应的got表是否已经被重定位，如果重定位完成则跳转到目标地址执行，否则调用动态连接器完成重定位地址填入got表再跳转。可以看到程序中有libc库函数puts，那么可以通过调用puts函数即call puts\_plt更新puts真实地址到got表中并打印出来，从而计算出libc基地址。这里利用了got表延迟加载的机制。

此时要注意32位linux是通过栈传递参数的，而64位linux前六个参数是使用rdi, rsi, rdx, rcs, r8, r9 传递的。由于不是使用栈，所以构建栈上的ROP时，需要把要泄漏的地址puts\_got\_addr放入rdi寄存器中。此时程序跳转到pop rdi的位置执行，栈指针出栈后接下来执行pop rdi，将栈指针当前所指弹出，存入rdi中。这样一来，成功将puts\_got\_addr放入了rdi中。执行后sp继续下移一帧，指向了puts\_plt地址，完成了puts(puts\_got\_addr)的调用。

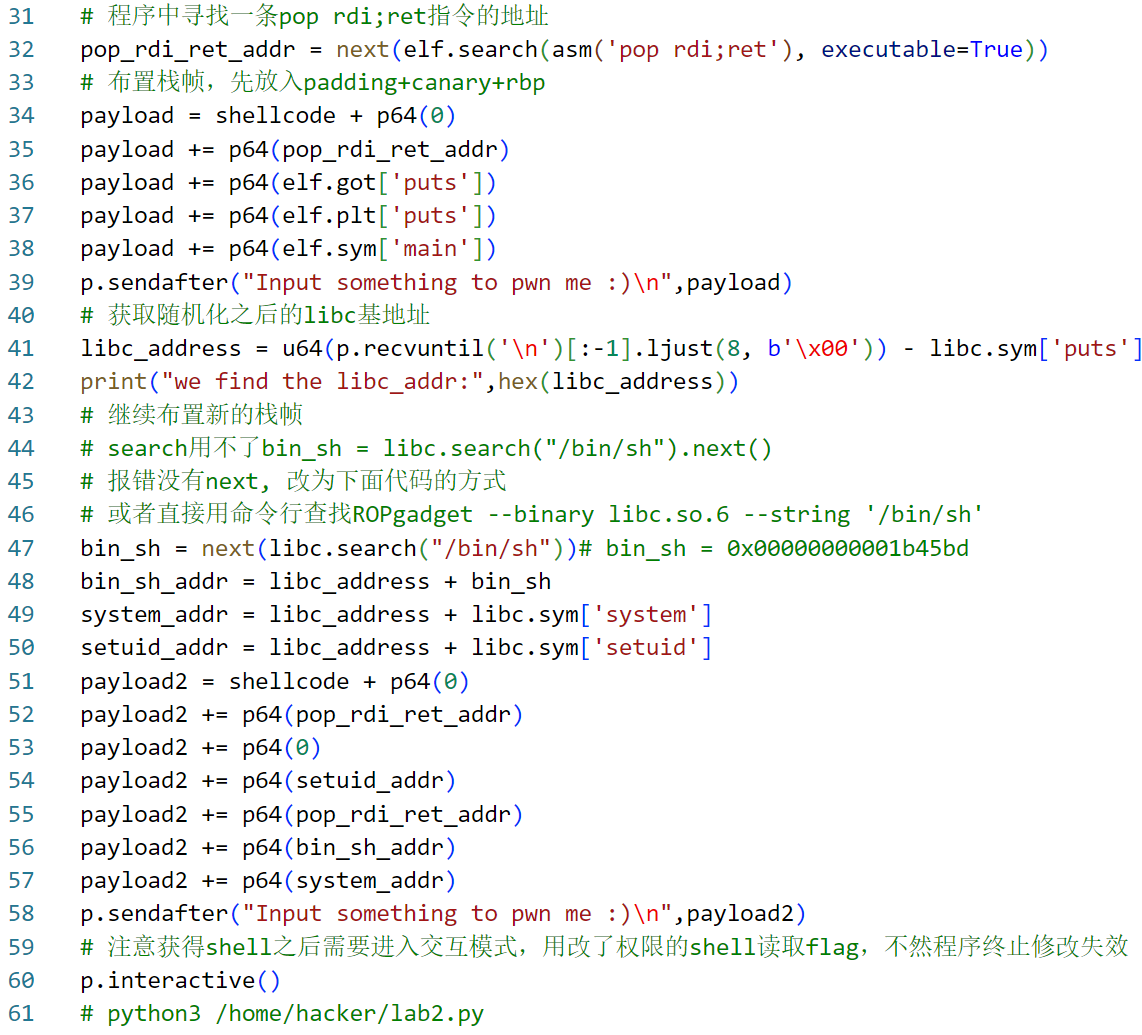
可以使用pwntools在程序中寻找pop\_rdi\_ret这样完美的gadget。

泄露出地址后还需要返回main函数，因为程序每次加载动态库的基地址都不一致，所以需要在一次运行中完成所有操作。得到了基地址便可以填充setuid(0)和system(/bin/sh）达到控制目的，然后再转入交互模式，输入cat /flag命令执行打印flag操作即可。

最终的两次栈帧布局如下所示：



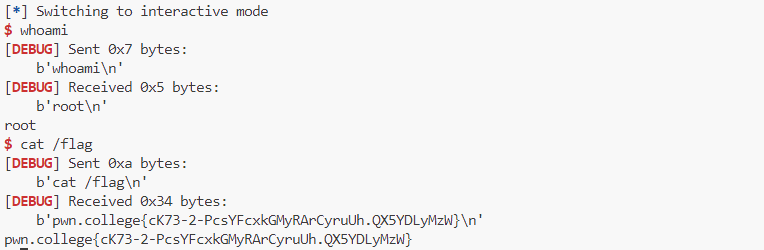
具体实现代码如下，分两次发送payload



执行脚本，结果如下，成功爆破canary并泄露基地址，最后获取root权限执行shell







# 2.实验遇到的难点与问题分析

1. 实验一：

实验一手搓代码读取ELF文件、使用命令分析ELF程序进程地址空间的部分没遇到什么问题，但是逆向分析密码部分一开始进行得不是很顺利。

首先是因为当时理论课的进度并没有很后面，课后作业的难度也不大，对逆向分析的接触还不够，无论是使用IDA静态分析还是gdb动态调试都非常不熟练，导致我有种无从下手的感觉。特别是在分析第四部分license的地方卡了很久，因为IDA的反汇编得到的c代码并不是那么准确，而且函数名和变量名都很混乱，加上我对c函数如strcmp的底层实现并不熟悉。所以我误以为最后的比较只是对第一部分进行检查，而没有想到是对整个部分的检查，后面存储的非可见字符其实就是MD5的比对值，而我却以为第四部分的比较放在某个比较深的函数中了。最后是反复用gdb调试才清楚了程序的逻辑，这个过程也让我真正开始学会使用gdb进行调试分析。

在做完了整个软件安全的平台作业后，再回过头来看第一次实验，简直不能再简单了，但对于当时基本没接触过逆向的我来说，真的是每一步都会遇到很多问题，IDA和gdb的使用，阅读理解汇编代码等等。所以我觉得这个实验最大的难点就是入门，是如何在短时间内接触一个全新的领域并摸清楚它的基本套路。这个实验基本算是我真正接触的第一道逆向分析题目，前期的作业题都没有涉及到完整的逆向分析流程，所以我还是处于未入门的状态，导致每一步的推进都有点无头苍蝇的感觉。

还有一点小问题就是python代码的编写部分，由于对字符串类型和比对操作不熟悉，有时候经常会出现代码无法实现预期功能的情况，三个脚本编写时都遇到了这个问题，好在通过网络搜索和多方案尝试解决了。

1. 实验二：

实验二得益于讲解清晰的实验指导书，加上理论课中对ELF文件结构的学习掌握，没有遇到什么大问题。难点都在指导书中给出了提示信息，一是canary的逐字节爆破，二是ROP链的两次构造。

canary逐字节爆破的思路很清晰。程序循环fork子进程读取用户输入，由canary保护机制检测到栈溢出后会报错，但不会终止父进程，而是继续循环fork输入的子进程，而且子进程与父进程的canary一致，由此产生了可利用的漏洞，即通过逐字节爆破泄漏canary最终绕过栈溢出保护，得以进行下一步的ROP链控制。难点在于爆破脚本编写，不过实验一后也有了编写穷举脚本的经验，没遇到什么困难。

ROP链的构造则花了一点时间来理解链的执行过程，主要是got表延迟加载的利用的理解，这部分和理论课契合得很好，所以也还算顺利。还有一个特殊的点是实验平台是通过设置uid来赋予权限的，所以还需要额外执行setuid(0)操作赋予root权限，如果指导书中没有给出这部分的说明那实验难度就大大上升了，因为我在阅读ROP链的相关介绍和教程时都没有提到这一点，这算是实验平台的特色吧。编写代码则没有什么困难，直接按照链填充。

# 3.实验小结

1. 实验一：

首先是手动编写代码读取ELF文件的元信息，让我对ELF的结构了解更清晰，也清楚了ELF文件的链接视图和执行视图的不一致之处，在理论课粗浅了解的基础上通过实操的形式巩固加深了这一块的知识，收获良多。

对ELF进程地址空间的分析也让我清楚了ELF文件从静态的程序转变为动态的进程，需要将各部分映射到内存中，包括程序各段、链接的各个动态库，以及栈和堆的加载布局等等。这部分也和本学期的操作系统原理课程相呼应，使我对两门课中的相关知识点都有了更深的体会。

对ELF程序进行逆向分析获取密码，让我真正开始了解pwn逆向分析的解题过程，并初步学会使用IDA静态分析，gdb静态分析，爆破脚本的编写等等逆向分析基操，收获非常大。

2. 实验二：

得益于实验指导书详实的讲解，我真正了解到构造ROP链的原理和思路，对于canary栈保护机制也有了实践上的体会，而不是仅仅停留在理论课泛泛理解的基础上。对于课程平台的核心原理也有了了解，对于之后分析更多形式的逆向题目有着非常大的帮助。

# 4.课程意见与建议

首先非常感谢老师和助教们提供了如此精彩的一门课，大量实操的形式使得理论课不再是枯燥的纸上谈兵，也让同学们对逆向分析以及安全方向产生了十分浓厚的兴趣。不过我认为这门课还能够更好，下面是我觉得可以完善的几个点。

一是课程安排的时间。了解到上一届的软安是安排在大二下学期，也就是在大二上学习完汇编之后。软安这门课无论是前半部分的逆向分析还是后半部分的病毒分析，都需要接触和使用大量的汇编代码，而且实际上这门课也不需要其它课程的前置知识，但又可以极大地激发同学们对逆向对安全的兴趣，对之后的研究和学习有着一定影响。所以我认为可以考虑把这门课前置，或者说和汇编课程的安排匹配。

二是课程作业的设置。虽然老师已经通过问卷的形式征集了大家对课程平台的建议，我还是在这里再阐述一遍吧。课程作业基本都是以逆向分析的题目为主，而课程理论部分的内容实际上并不是都可以用逆向呈现的，可以适当增加一些文字题目如思考题。平台上的作业题也缺乏一定的先导知识，而且64位环境的相关教程和介绍比32位要少很多，建议每个模块前增加一个专门的入门板块共同学们学习了解，而不是只用短短一句话描述关卡特点，这样对大部分未接触过逆向分析的同学很不友好，本来就没有经过系统的了解学习，还要花费大量时间在混乱的网络中查找相对清晰的教程学习并理解，甚至都不知道该提问哪一部分。类似实验一实验二的指导就很好。此外，可以尽量减少每个类型的作业题，留一两道经典的就够了。

三是实验。实验一实验二可以设置成教学演示模块帮助同学们入门逆向，然后采用形式类似但是讲解更少的题目作为拓展实验，我甚至觉得有些作业题难度远大于实验题，可以替换一下，或者直接加到实验中去，给出更详细的提示。比如ELF CrakeMe模块对理解ELF文件很有帮助，可拓展为实验；ptmalloc堆分配那题也可以加到实验中并附加对tcache的介绍；ROP链的构造可以替换成保护全开的题目，加大难度。

还有就是课程ppt的内容可以丰富一些，前半部分的ppt有些简略了，即时当时上课或者做练习后清楚了某个知识点，考前也可能记不太清了，这门课本身涉及的知识点又非常多且细节。PPT上的提示又太少，自己做的笔记也不太规范，导致复习效率降低。

最后，希望这门课越来越好！