编号：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验 | 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 总评 | 教师签名 |
| 成绩 |  |  |  |  |  |  |  |  |  |  |

武汉大学国家网络安全学院

课程实验(设计)报告

题 目： Heap-overflow

专业(班)： 信安5班

学 号： 2022302181161

姓 名： 王亚鹏

课程名称： 软件安全

任课教师： 赵磊

2024年 11 月 16日

**目录**

[堆溢出实验 1](#_Toc182680683)

[1实验名称 1](#_Toc182680684)

[2实验原理 1](#_Toc182680685)

[3实验步骤 3](#_Toc182680686)

[4 实验结果 8](#_Toc182680687)

[5实验心得 9](#_Toc182680688)

## 堆溢出实验

### 1实验名称

**Heap\_overflow（堆溢出实验）**

### 2实验原理

通过构造特定的数据破坏堆的内存结构，进而达到执行任意代码的目的

**1. 堆内存管理基础**

堆是由程序动态分配和管理的内存区域。堆内存管理涉及以下三类操作：

* **堆块分配**：分配一块连续的内存，用于存储数据。
* **堆块释放**：释放不再需要的内存，使其可以再次分配。
* **堆块合并**：当释放的堆块相邻时，合并它们以减少内存碎片。

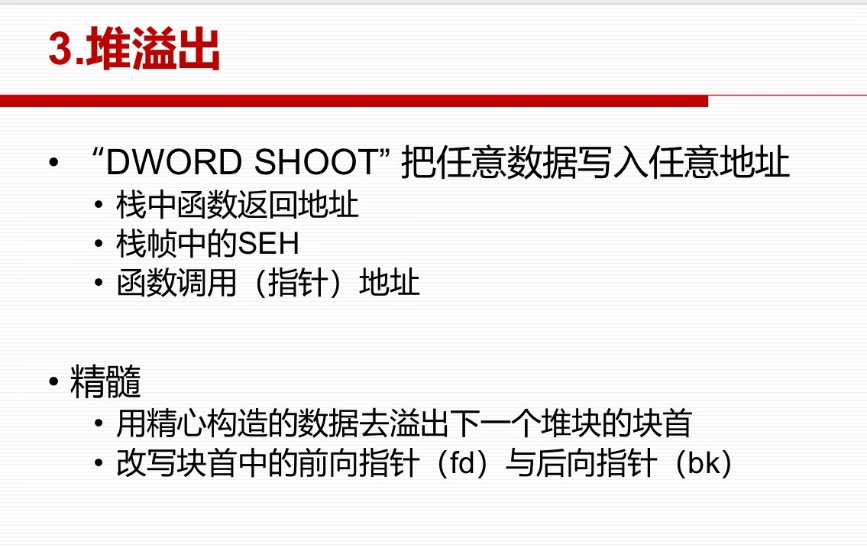
在堆管理过程中，堆内存的数据结构通常包含以下指针：

* **前向指针 (fd)**：指向链表中的下一个堆块。
* **后向指针 (bk)**：指向链表中的上一个堆块。

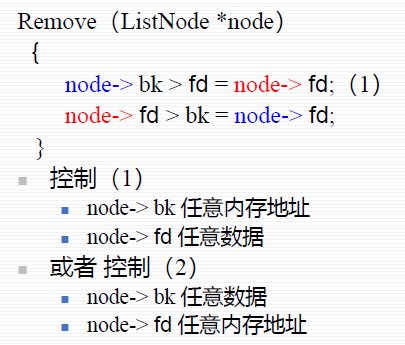
**2. 堆溢出原理**

堆溢出发生在程序未正确检查用户输入时，将用户输入超出分配空间的大小写入堆内存，从而覆盖临近的内存区域。通过精心构造的溢出数据，可以：

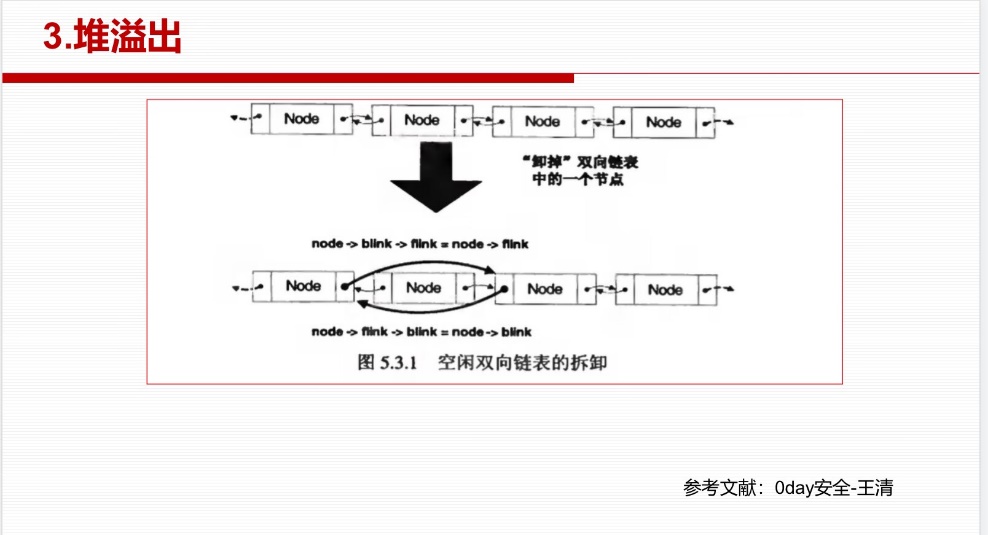
1. **修改临近堆块的头部**：破坏堆块的fd和bk指针。
2. **伪造堆块**：构造一个伪造的堆块结构，使其包含恶意指针。
3. **劫持程序控制流**：利用释放、合并堆块的过程触发伪造的指针，将执行流重定向到攻击者指定的位置（如shellcode）

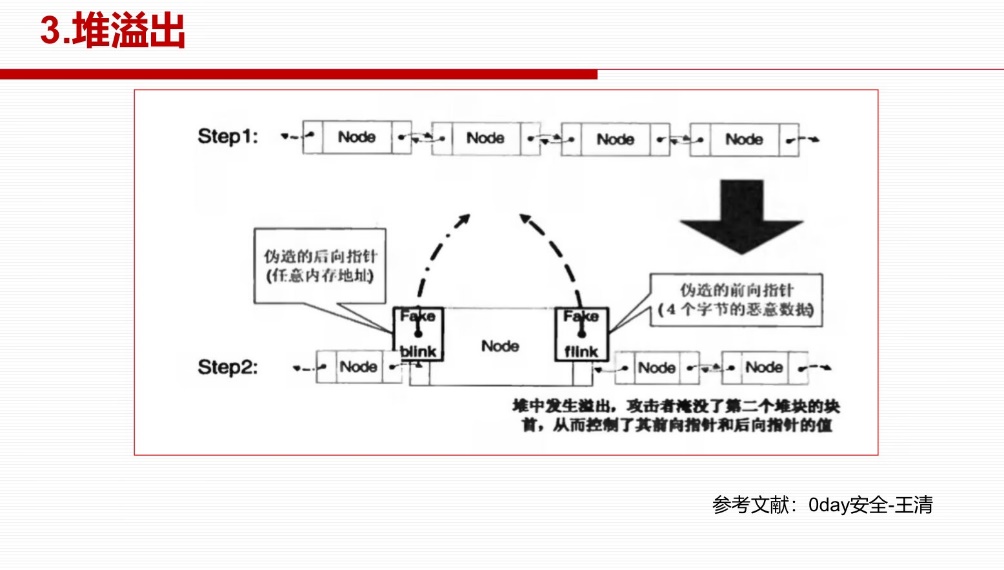


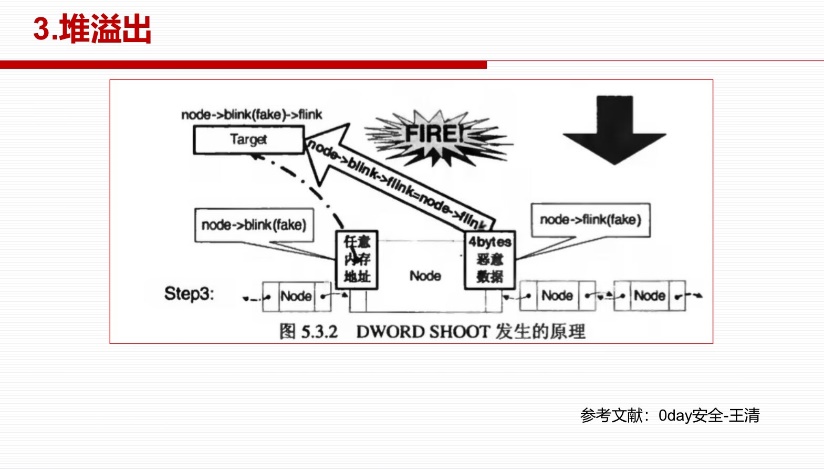
以堆块释放为例，简述写入的过程，由于我们释放堆块要涉及到删除链表中的结点，因此会进行以下操作：



图示过程如下（参见课程群内的PPT）：



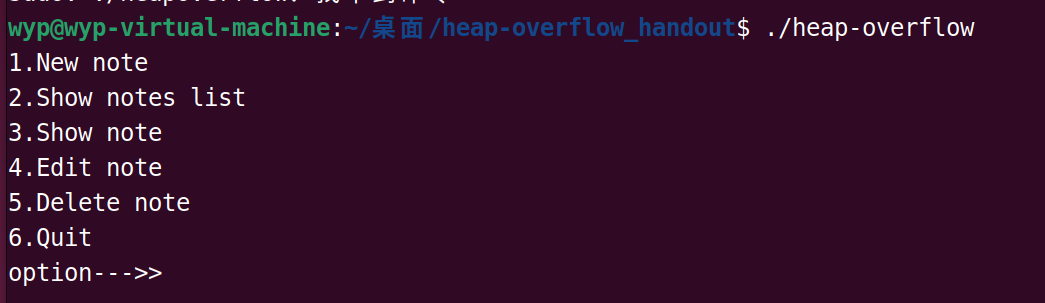




### 3实验步骤

**1.程序分析**

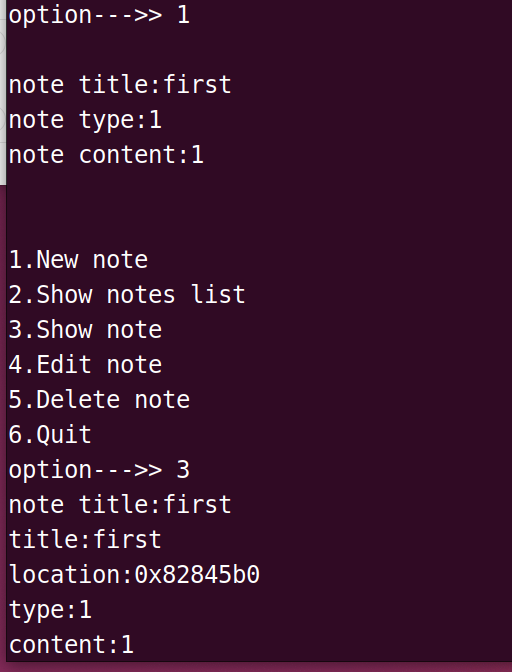
首先运行heap-overflow程序



可以看到出现了6个不同的选项

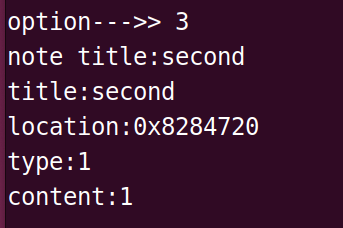
6个选项代表着利用该程序模拟堆块的操作，如添加堆块，释放堆块，编辑堆块，删除堆块等。根据堆溢出的原理，猜测利用1,4,5的过程可能发生溢出

如下图所示，先创建一个堆块，然后使用选项3来查看指定堆块的信息



可以看到它的地址、大小、内容

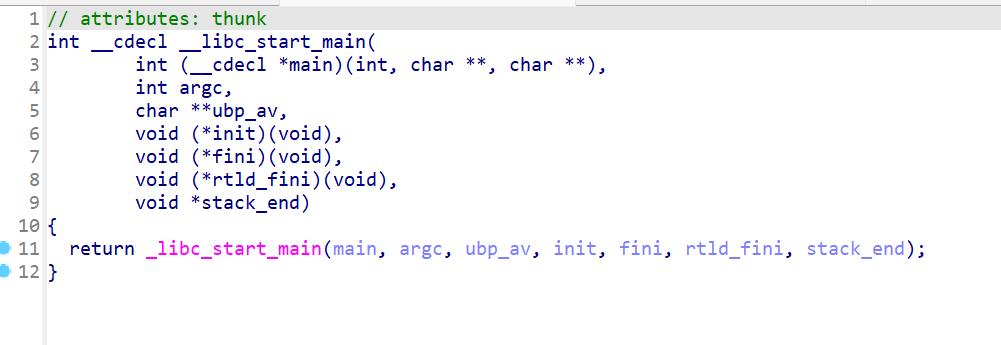
再查看下一个堆块的信息



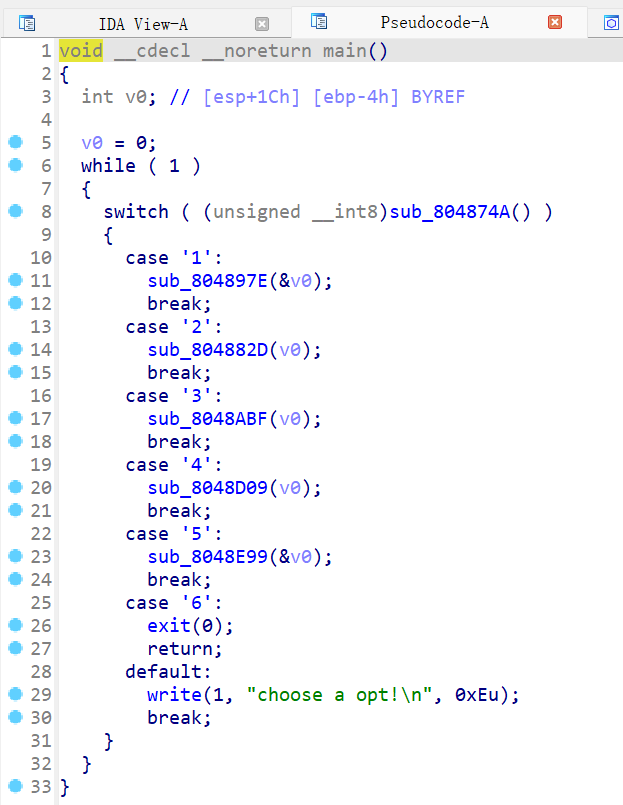
这两个堆块我是相继创建的相邻堆，通过它们的地址差值可以算出两个连续堆块之间的距离是0x170

**2.源码分析**

接下来使用IDA Pro反汇编该文件的源码



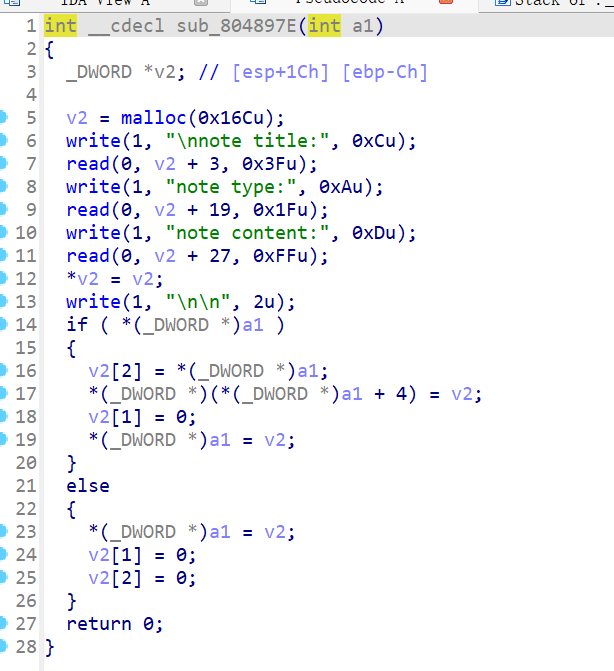
进入IDA Pro之后使用F5进行反编译生成C语言文件，看到如图所示的界面，并没有显示什么有效信息，这时我们双击第一个参数main，就能出现main函数的反编译结果如下



main函数对应的应该就是主界面的几个选项，重点观察1,4,5,这是根据溢出特点猜测的有问题的地方

**选项1，创建堆块**

反编译结果如下：



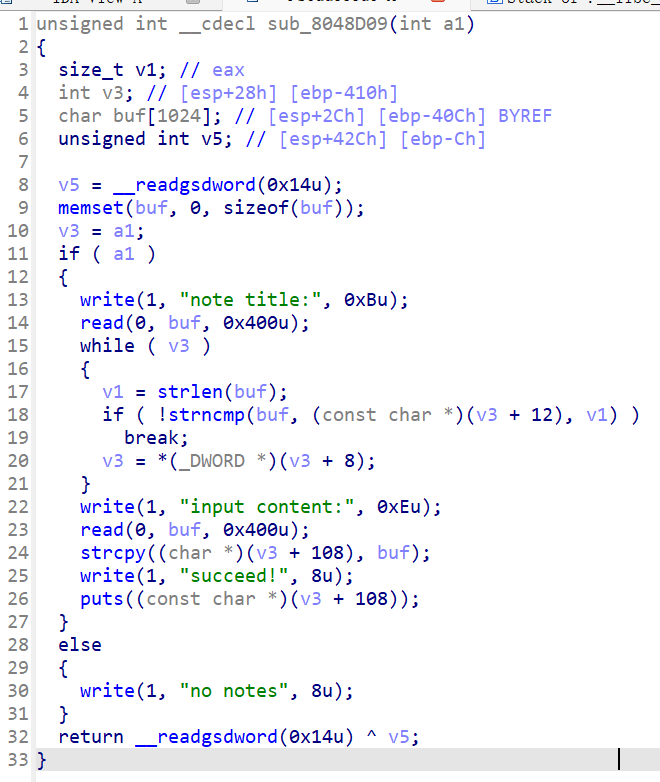
可以看到首先为v2分配了0x16C大小的空间

接下来如果链表非空的话，则将v2插入链表头

如果链表为空，则将v2作为链表头

**选项4，编辑堆块**

反编译结果如下：



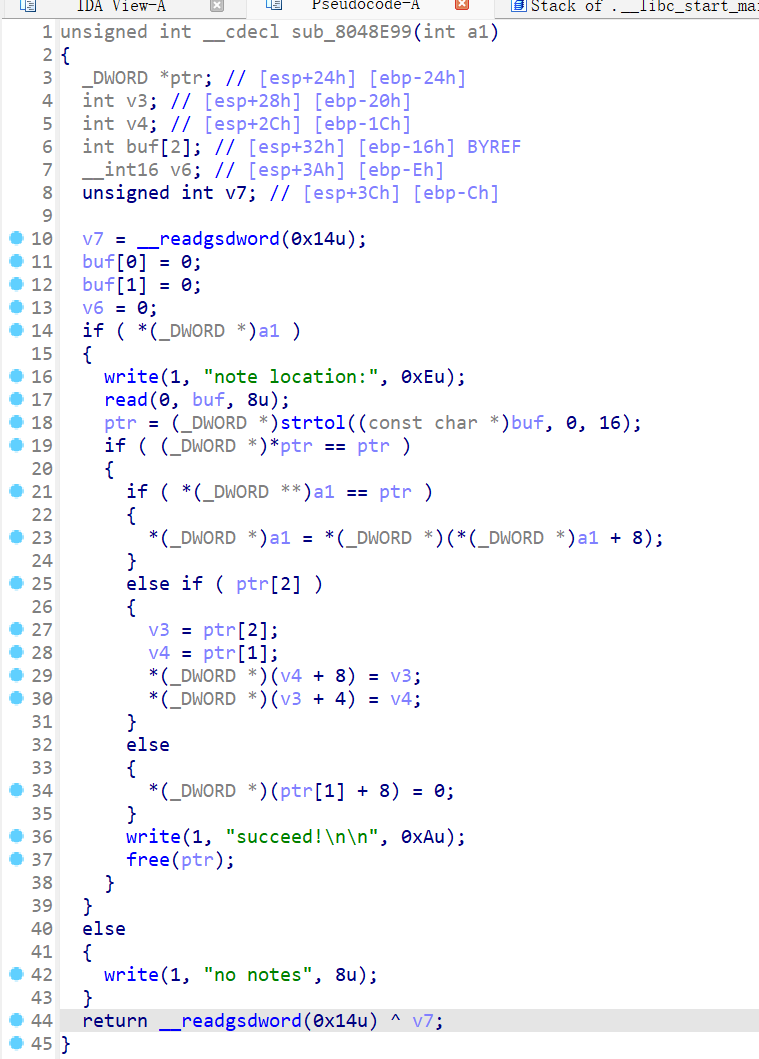
首先开辟一个大小为0x400的缓冲区buf，用于存储用户输入的数据

然后通过循环找到与用户输入的堆块

接下来将用户的输入拷贝到当前堆块内容偏移为108的位置处，此处编辑堆块可以被我们用于将shellcode写入，覆盖堆块的前向后向指针

**选项5，删除堆块**

反编译结果如下：



如果存在堆块，则将输入读取的要删除的块地址信息读取到buf中，块地址解析为16进制数存储在buf中

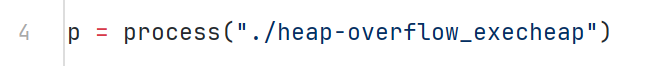
如果ptr是链表头，则更新链表头为下一个节点

如果链表头是中间结点，设v2是ptr的后结点，v3是ptr的前结点，然后让ptr前节点的后结点为v2，ptr后结点的前节点为v3

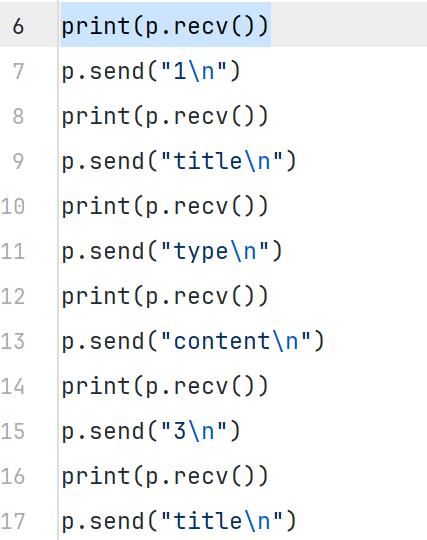
如果ptr是最后一个结点，则将前一个结点的后指针置空

这里会发生堆溢出

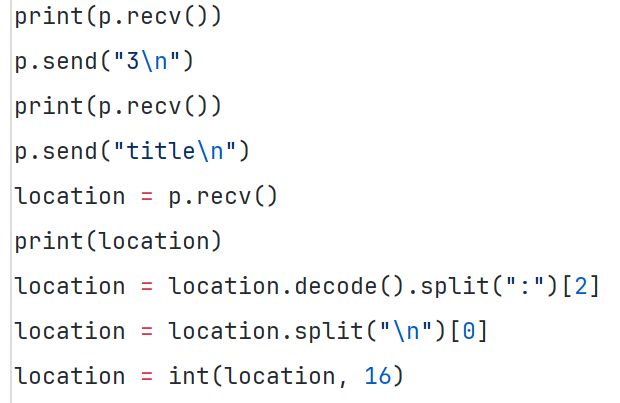
**3.脚本攻击**



脚本通过pwntools的process函数启动目标程序，与之交互

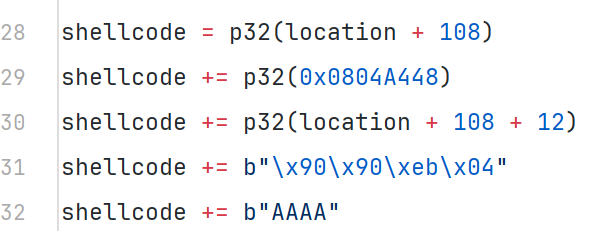


接着利用程序的功能选项，创建一个堆块，并填写标题、类型和内容



之后利用程序的查看功能，提取堆块的内存地址

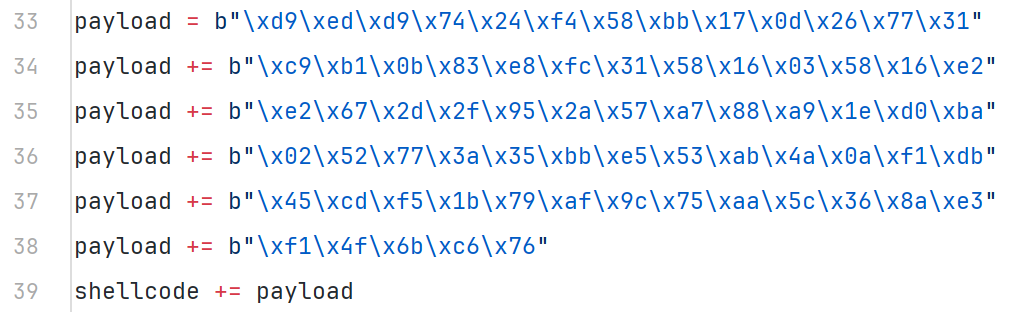
地址的获取是后续伪造堆块的基础



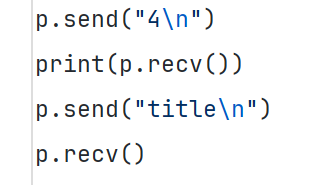
经查找资料，这是在伪造堆块地址

**伪造堆块内容**：

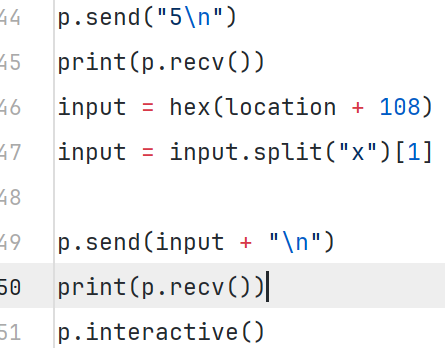
* fd 指针：被设置为目标地址（可能是free函数的GOT表地址）。
* bk 指针：指向攻击者的Shellcode地址。
* 利用NOP滑板（0x90）确保指令能跳转到有效的Shellcode。



这是执行Shell的机器指令代码（Shellcode）



通过编辑堆块功能，将伪造堆块的数据写入指定堆块位置



释放伪造堆块：释放堆块时，堆管理系统会使用fd和bk指针执行链表操作。

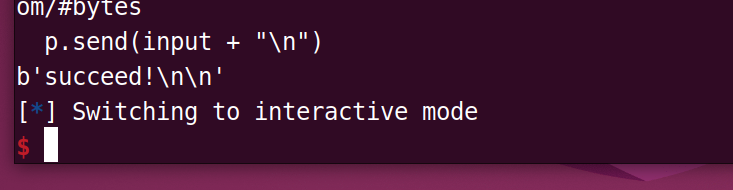
劫持控制流：释放操作时跳转到伪造的fd指针（攻击者的Shellcode）。

**攻击流程总结**

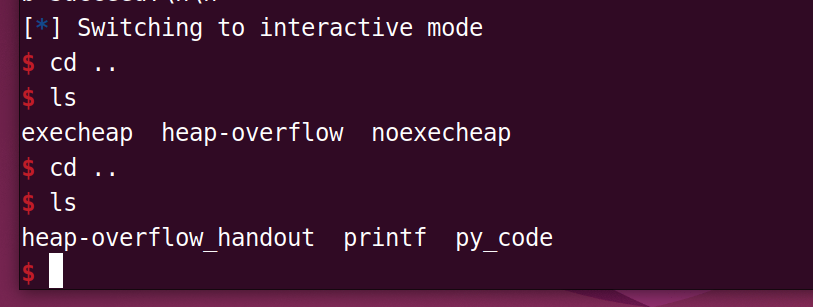
1. **创建并获取堆块地址**：通过程序功能创建堆块并提取其内存地址。
2. **伪造堆块**：构造含有恶意fd和bk指针的伪造堆块，注入攻击者的Shellcode。
3. **释放伪造堆块**：触发释放操作，劫持程序控制流，执行Shellcode。
4. **攻击效果**：成功获得Shell，控制目标程序。

### 4 实验结果

运行脚本，结果如图：



看到了输入命令行的地方，我们可以输入一些指令进行尝试



运行无误，成功窃取了shell

**攻击的核心过程解释如下：**

1. **程序提供的堆块操作功能**：
   * 程序支持对堆块的常见操作，如创建、查看、编辑和释放。
   * 攻击者利用了编辑和释放堆块的功能，通过堆溢出伪造堆块并劫持释放逻辑。
2. **攻击者操作的关键步骤**：
   * **创建堆块并获取地址**： 攻击者通过程序的功能创建一个堆块，并通过查看功能获取其内存地址。
   * **伪造堆块内容**： 利用编辑功能，在堆块的用户数据区域（location + 108）伪造一个假堆块。
   * **释放伪造堆块**： 攻击者传入伪造堆块的地址，触发释放逻辑，从而执行伪造块中的Shellcode。
3. **伪造数据的内容**：
   * **伪造的堆块头部**： 伪造的堆块头部包含以下内容：
     + **location + 108**：伪造堆块的地址，满足合法性检查。
     + **fd 和 bk 指针**：
       - fd 被设置为目标地址（如free函数的GOT表入口）。
       - bk 被设置为Shellcode的地址。
   * **Shellcode**：
     + 被注入到伪造堆块中，确保释放时执行攻击者的代码

攻击成功的关键在于：

* 程序对堆块合法性验证不足。
* 堆溢出导致的内存覆盖。
* 释放逻辑中对伪造指针的信任。

### 5实验心得

对堆溢出的攻击原理和实现过程有了深入的理解。这次实验不仅了解了堆内存管理的基础知识，同时进一步熟悉了漏洞利用的具体步骤和工具的使用。理论部分让我知道如何利用堆溢出修改指针（如fd和bk），并通过伪造堆块劫持程序的执行流。在实际操作中深刻体会到漏洞利用需要对内存布局、指针地址等细节有非常准确的把握。

**1. 工具使用**

IDA Pro：

通过反汇编和逆向分析程序，定位到主函数的调用流程，观察到程序的堆操作逻辑。

虽然最开始反编译结果没有直接显示主逻辑，但通过分析\_\_libc\_start\_main，成功跟踪到了主函数

Pwntools：

一个功能强大的Python库，用于漏洞利用的脚本编写。通过该工具，实现了自动化的堆块创建、编辑和伪造，最终成功劫持程序控制流

**2. 安全机制的重要性**

ASLR **(地址随机化)：**

通过随机化内存地址，增加了溢出攻击的复杂性。，没有ASLR的情况下，才得以固定堆块的内存地址，实验才能够进行

地址随机化让我认识到，虽然漏洞可能存在，但现代系统中多层次的保护机制极大地提高了漏洞利用的难度。

**3. 攻击过程**

伪造堆块的构造逻辑： 刚开始时，对伪造堆块的内容（如fd和bk的指向）理解不透彻，通过反复调试还有上网查找资料、请教同学，最终搞懂了原理