# # Lab06 栈溢出实验详解

本实验需要环境：Linux x64，pwntools软件，gdb安装pwndbg插件，gcc。

编译时，需要加参数 -O0 -fno-stack-protector -fno-pie。

系统要关闭ASLR：echo 0 >/proc/sys/kernel/randomize\_va\_space

## ## 问题1：给出代码，靠溢出劫持返回地址

我产生的二进制文件，需要将原代码中的+16改为+24，将+8改为+7。

本题的核心在于使用GDB等调试工具看到栈布局和可执行文件Text段的布局情况（也就是汇编代码）。

原理异常暴力，就是欺负关了缓解措施的操作系统拿你程序员没办法。

## ## Extra：写一个密码验证，劫持返回地址强行通过

同样，靠观察汇编二进制文件和栈的情况，硬编码一些Offset。同理，看我的可执行程序。

## ## 演示步骤

首先，安装pwndbg，打开GDB调试对应二进制程序。

对于第一个问题，我想要跳过func调用后，对x的那个赋值语句，观察运行时func函数的返回地址：

电脑萤幕的截图

描述已自动生成

发现是位于rbp（栈底指针）所指向的下面是回到主函数的地址main+35，我要靠上面栈里面的东西向下增长覆盖掉这个值，让它回去的时候直接跳过x=1赋值语句，运行过程中，注意DISASM汇编部分，发现跳过x=1后的语句位置为main+42，也就是说源代码中的+8应该变为+7。

为什么+16要变成+24，是因为：当+16时，栈中rax会指向rbp和rsp一样的位置，也就是说少了一个+8来跨越上面压栈的原rbp（现在rbp所指的db70），这整个func的逻辑就是使rax作为指向返回地址的指针，然后暴力修改其指向的内容（也就是返回地址）。

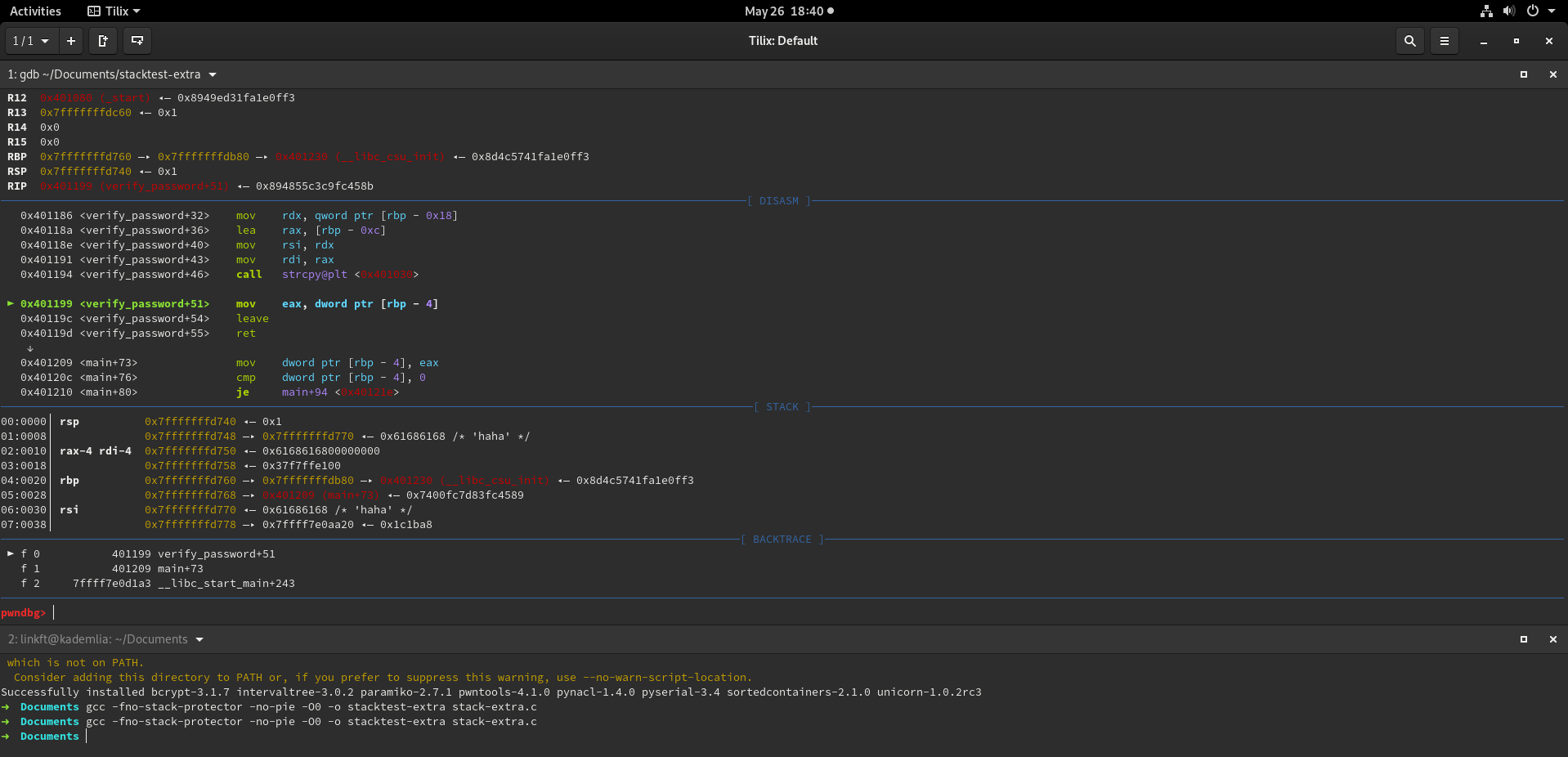
那么原来的+16都跨越了什么东西？我观察栈发现，它其实只跨越了char数组（大小为8B），剩下8B都是什么东西？可能是编译器填充的00空挡，也可能是为了内存对齐所做的补全（1B变2B）。

第二题的思路则要根据我写的代码来看，还是同样道理，观察汇编DISASM的布局情况，发现要跳转的地址，succeed函数地址在main+99（0x401223），我先判断我写的程序必定存在缓冲区溢出漏洞，于是先输入30个A+123等占位字符看栈的情况，主要是利用strcpy不检查缓冲区大小的问题。

手机屏幕的截图

描述已自动生成这里需要多次实验，因为strcpy似乎没有遵照一般要求进行升栈移动rbp的操作！我也一眼看不出由检测函数持有的buffer缓冲区是哪个。最简单粗暴的办法就是输入一堆让它必定溢出，观察123出现在d768这个地址的情况（前面有几个h？）来判断应该填写多少偏移量。

由于目标跳转地址0x401223转为ascii存在不可见字符，而且存在小端序转换问题，所以使用pwntools来对它进行输入。看exp.py。



现在d750里面的就是strcpy用的缓冲区，当然这是我们后来慢慢观察出来的，我们要攻击他让它溢出，覆盖写入下面d768的返回主函数的返回地址，让它必定直接跳转到succeed函数上，succeed函数在main+99也就是0x401223。