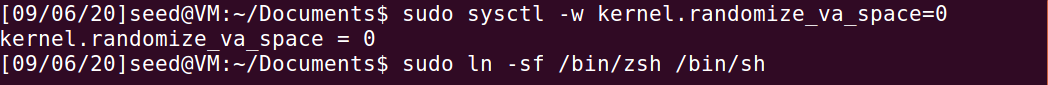
61517304 刘余文

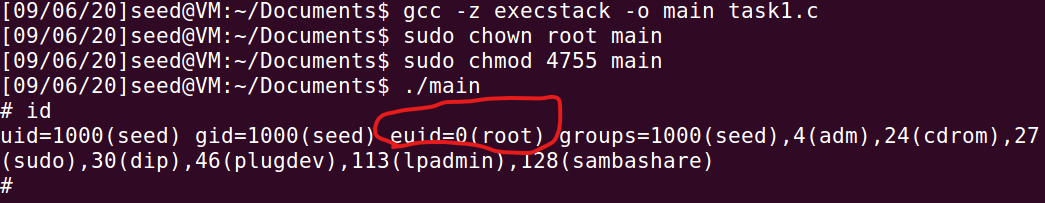
Buffer Overflow Vulnerability Lab

1. 关闭防护措施
   1. 关闭地址随机
   2. 关闭栈保护
   3. 使用可执行栈
   4. 重新配置/bin/sh的链接



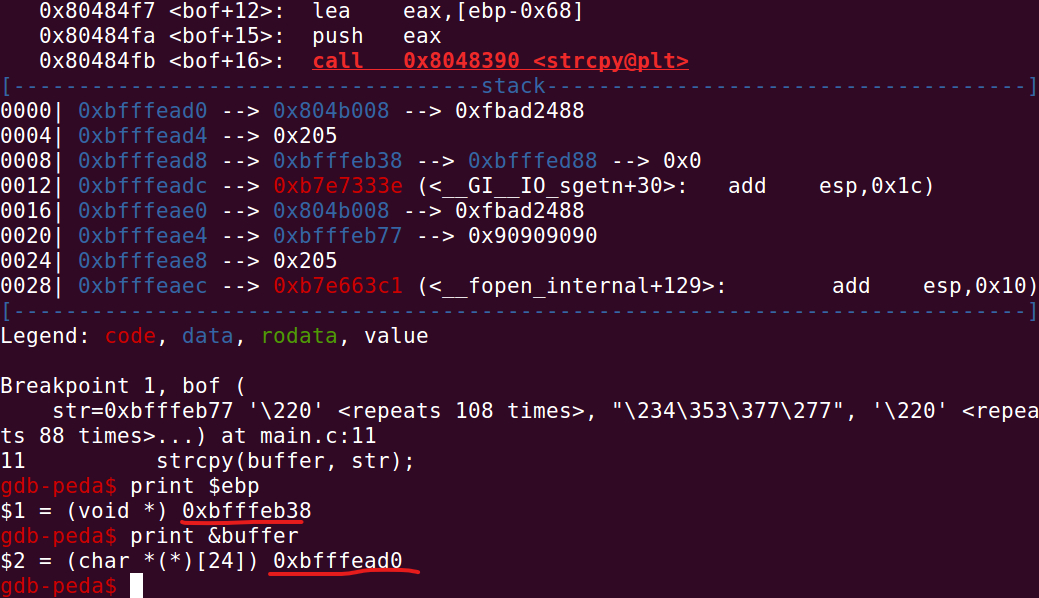
1. 运行Shellcode

编译并运行相关代码即可，注意需要使用可执行栈。

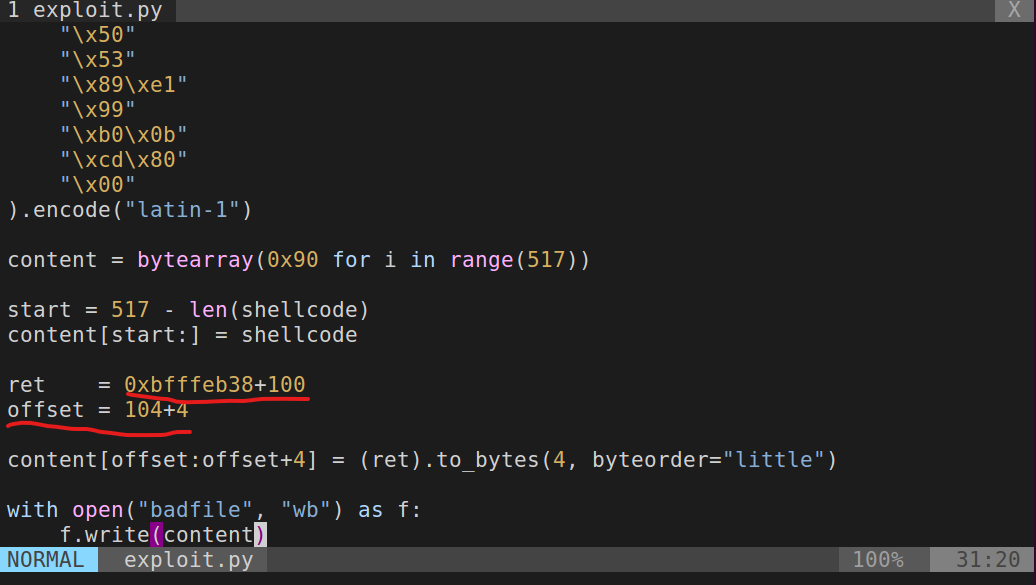


可以发现，虽然是字符串，一样也能实现Shellcode

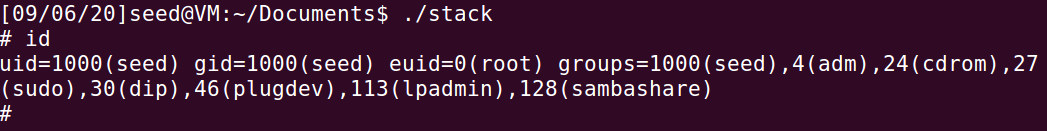
1. 完成缓冲区溢出
   1. 首先编译完成有缓冲区溢出的程序，并使用gdb找出bof函数中的ebp地址和buffer的地址



* 1. 构造出badfile的文件，其中，bof函数的栈的返回地址的位置在$ebp+4的位置，而$ebp和&buffer差104，因此自定义bof返回地址为$ebp+100，bof返回地址的位置偏差为104+4，如此构造成badfile

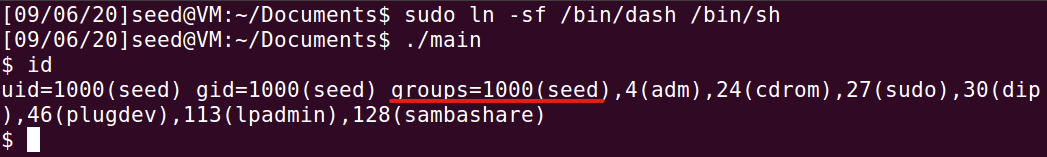


* 1. 设为Set-UID程序，运行发现获得root权限的shell

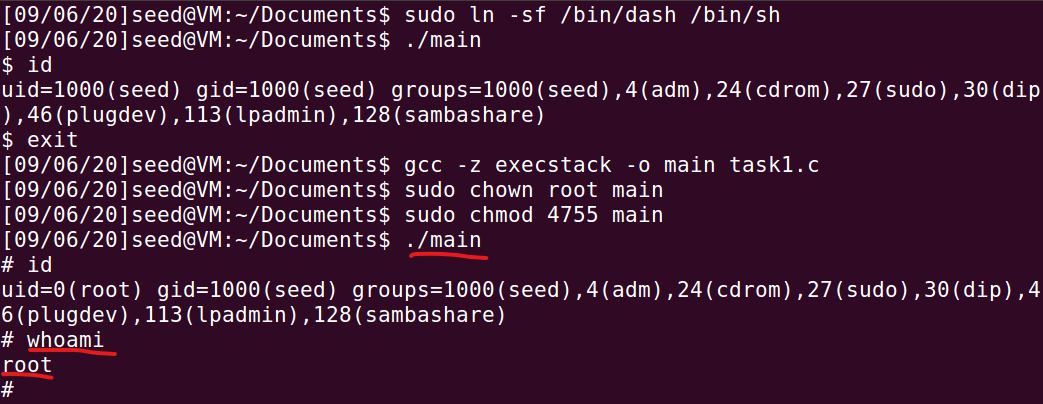


1. 突破防御

将/bin/sh重新链接回/bin/dash，在运行2中的代码，发现已经自动降级

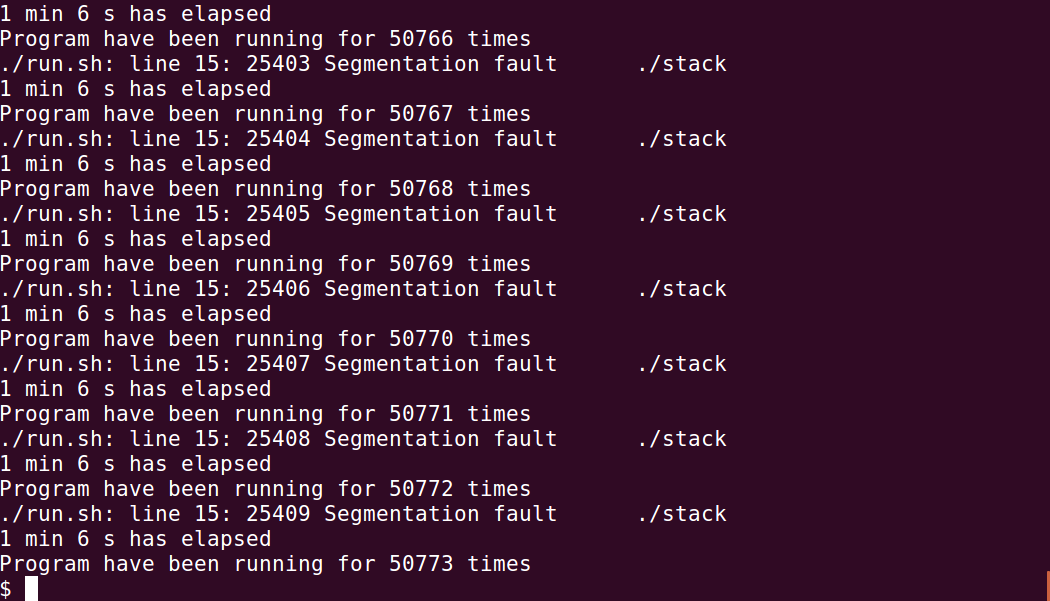


但是加上前端setuid(0)的汇编代码后，重新获得root权限



1. 突破地址随机

突破地址随机的办法，最常见的就是直接爆破，用固定的地址覆写，但是比较难命中，此处直接运行sh脚本，查看结果。



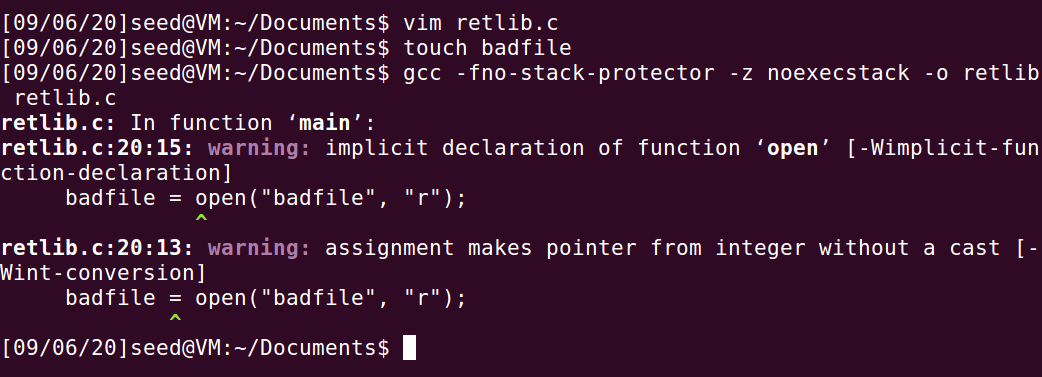
发现50773次尝试后成功爆破到指定位置

Return-to-libc Attack Lab

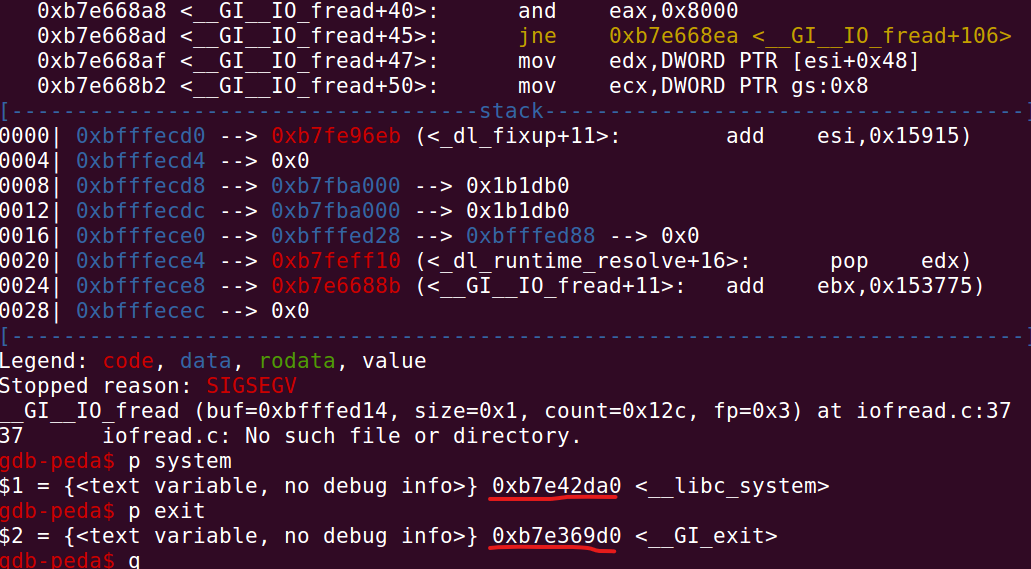
1. 关闭防护措施

同上个实验

1. 找到libc的函数地址
   1. 编译程序

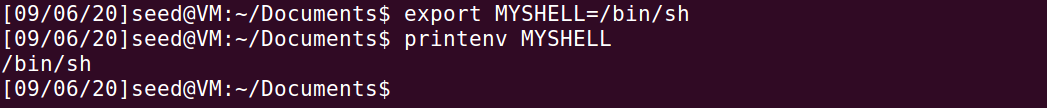


* 1. 使用gdb找到system函数和exit函数的执行地址

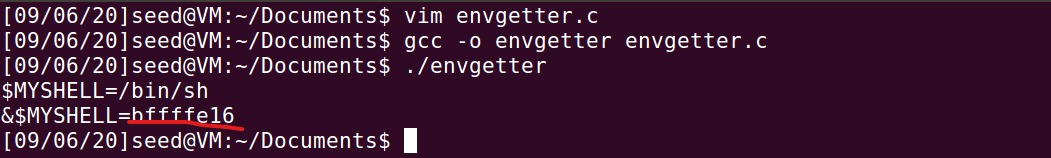


1. 将shell的字符串放进内存中

采用的方法是使用环境变量



在代码中获取环境变量



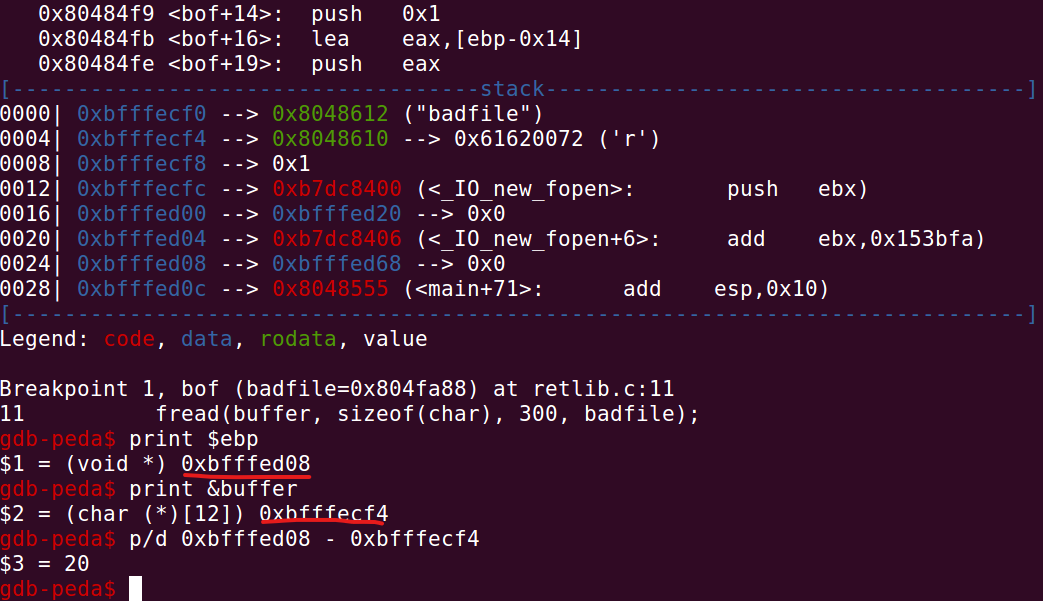
1. 完成攻击

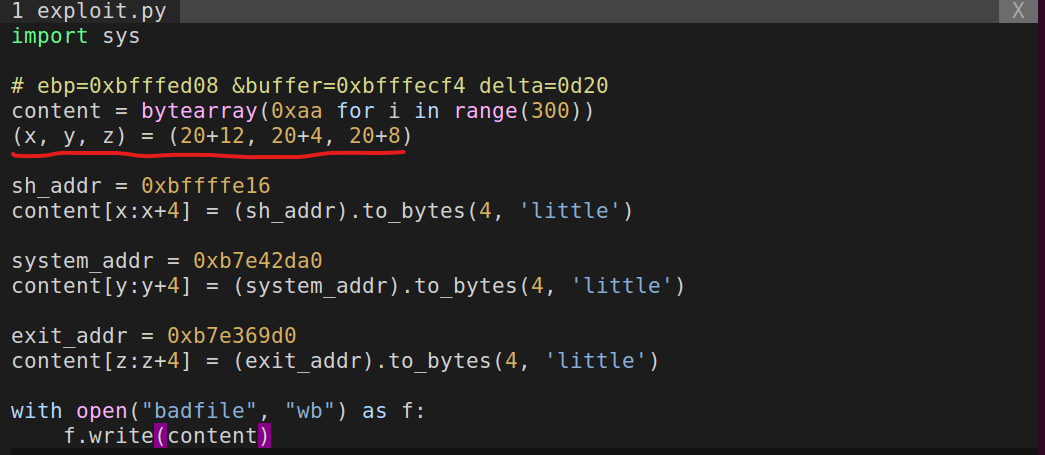
获得$ebp，那么返回地址位置就应该是$ebp+4，因为要调用system函数，所以放置system的地址；system调用完成后的返回地址存储在$ebp+8中，因此此处放置exit的地址；system调用的函数为/bin/sh，为一个字符串，作为参数应该放在$ebp+12中，因此此处放置”/bin/sh”的地址

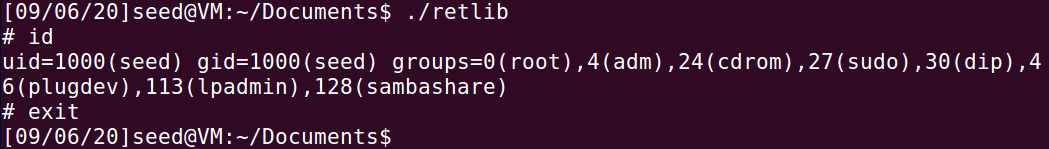
运行代码后发现成功获得root权限的shell

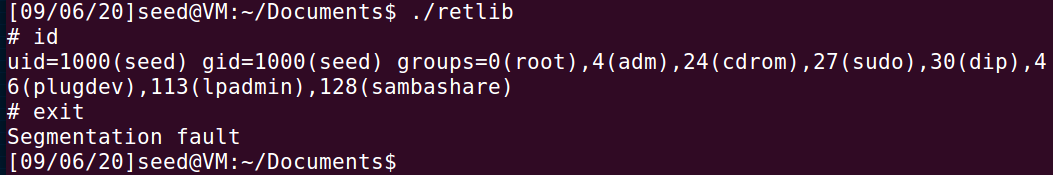
攻击变种1：不加exit会导致退出时发生段错误

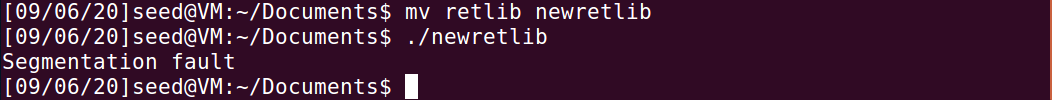
攻击变种2：该名称会导致环境变量的改变，导致段错误











1. 打开随机地址

尝试使用爆破，但是不行，一直段错误，分析原因，可能是其中多个变量在随机变化，导致情况太多无法通过爆破完成

使用gdb调试发现system、exit函数的地址都改变了，环境变量的地址倒是没有改变

