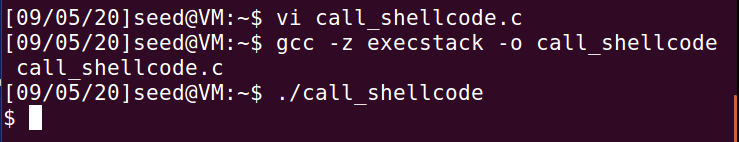
Task 1: Running Shellcode



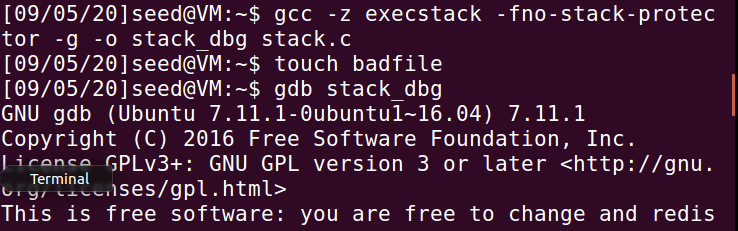
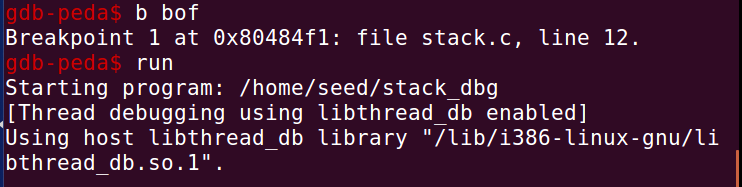
运行手册中的程序call\_shellcode.c，并关闭栈不可执行和栈保护，成功调用了shell。

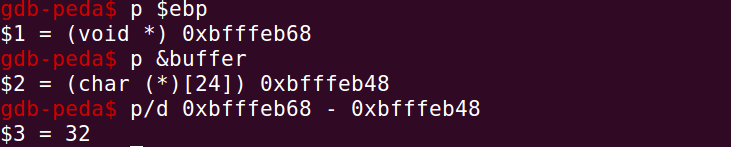
Task 2：Exploiting the Vulnerability

首先我们让sh指向zsh而不是dash。

关闭地址随机化、栈不可执行和栈保护。

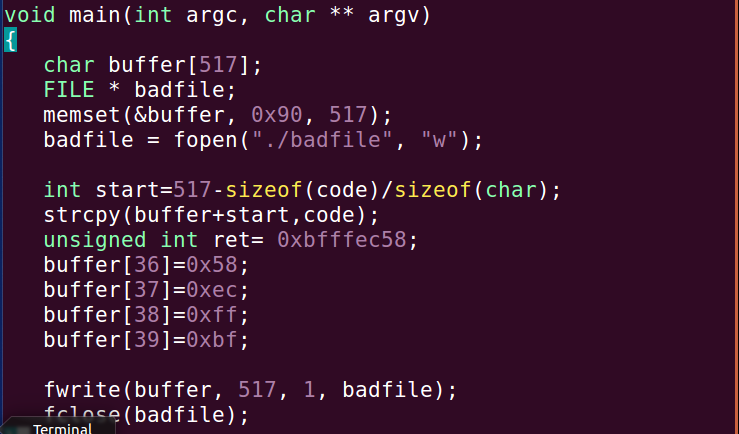
编译，进入gdb。

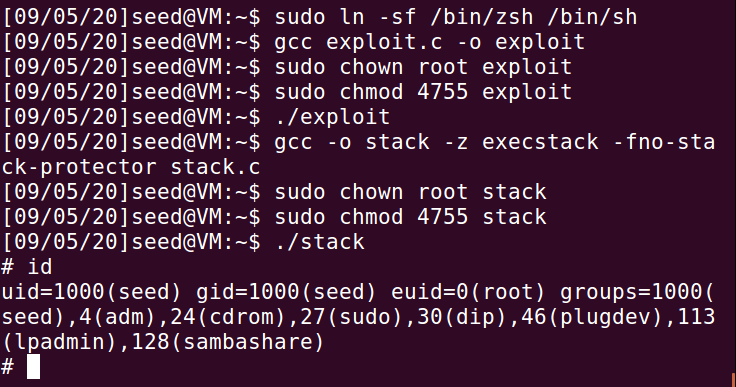
 使用 b bof命令对bof设置断点，然后run。



使用p $ebp查看ebp寄存器此时的值，使用p &buffer查看buffer数组的地址。ebp寄存器此时的值再加4就是bof函数返回地址所在的内存地址，通过p/d就可以计算出buffer与bof返回地址之间的距离为32+4=36

因此，exploit.c的程序更改为：

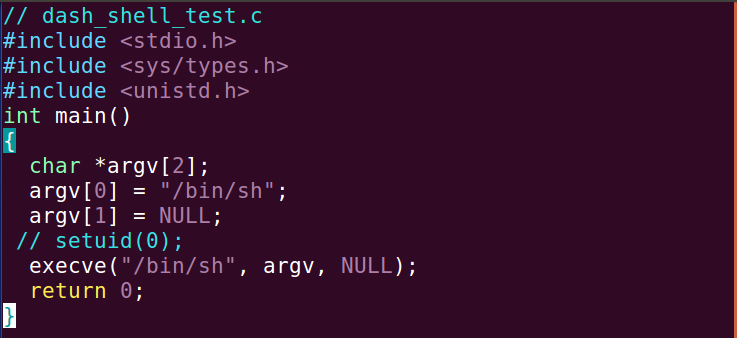


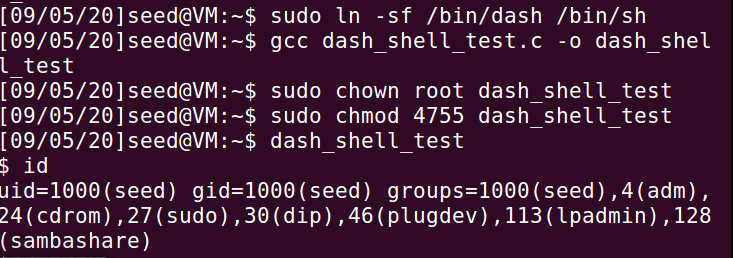
将stack变成root所有的Set-UID程

观察运行结果，我们成功得到了shell，并且输入id可以查看到，我们成功拿到了root用户权限。

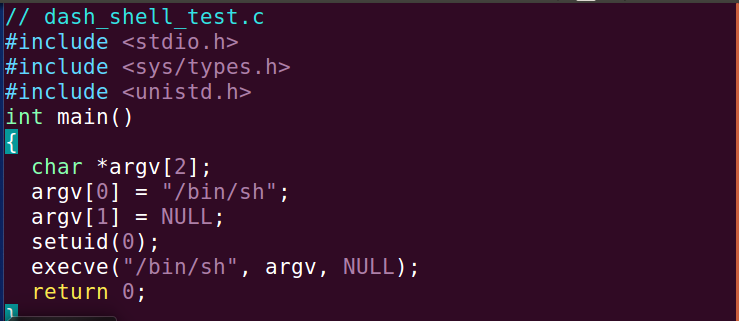
Task 3: Defeating dash’s Countermeasure

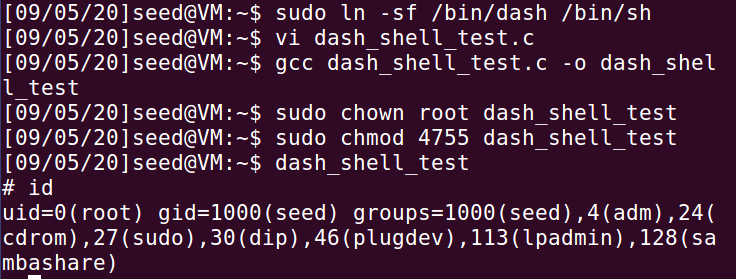
已知此时使用的是dash，当RUID不等于EUID时，dash会给程序会自动降权，复制手册中dash\_shell\_test代码编译，改为root的Set-UID程序运行，运行，无法提权。

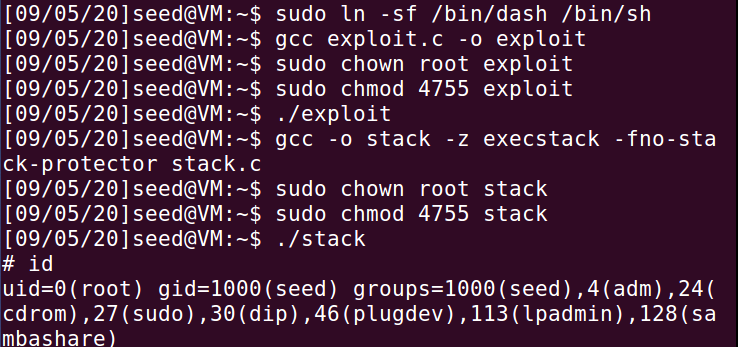
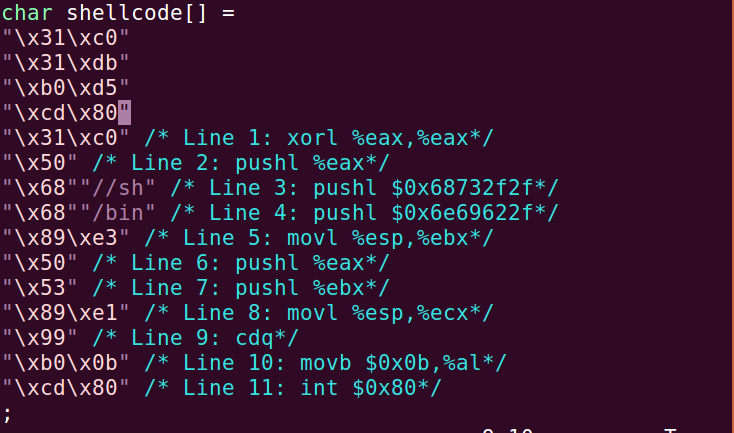




接着按手册要求修改代码，setuid(0)取消注释后再次编译，运行，由于此时RUID等于EUID，成功提权。



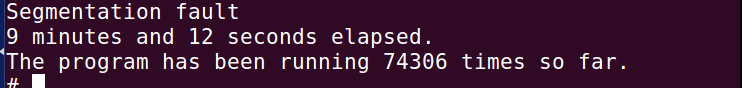


按手册要求，在调用execve之前添加新的shellcode，将RUID置为零，从而绕过dash对UID的判断，在exploit.c和exploit.py中添加新的shellcode，重新运行生成badfile，实现栈溢出并成功提权至root

分析dash源代码我们可以发现当用户EUID与RUID不相同时，dash会自动将程序权限降级，但setuid=0这个命令就是将RUID也设置为root，这样两者相同，就不会出现降级问题。

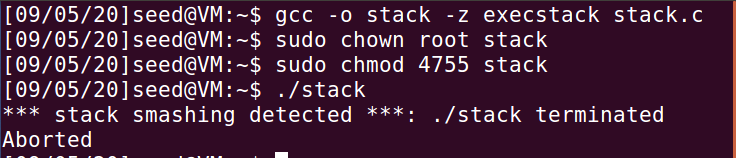
Task 4: Defeating Address Randomization

按手册编译运行运行程序，在9分12秒后获得结果，运行74306次。



Task 5: Turn on the StackGuard Protection

再次运行stack，将-fno-stack-protector选项关闭，栈保护开启，此时运行stack.c发现报错，并且精确说明错误原因是被栈溢出攻击。



Task 6：Turn on the Non-executable Stack Protection

选择-z noexecstack选项之后运行stack，发现再次报错

