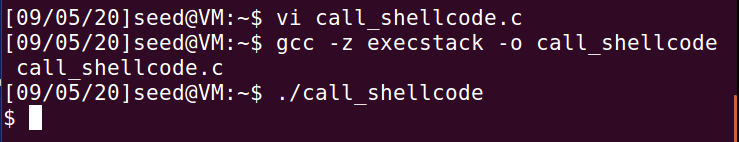
**Buffer Overflow Vulnerability Lab**

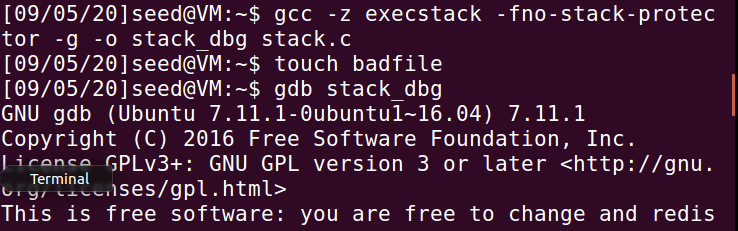
——57118125施祺

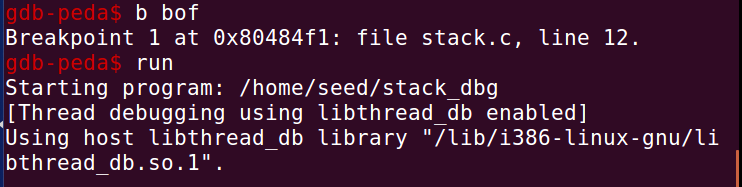
**Task 1: Running Shellcode**

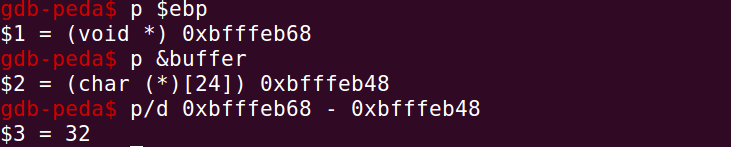


运行手册中的程序call\_shellcode.c，成功调用了shell

**Task 2：Exploiting the Vulnerability**

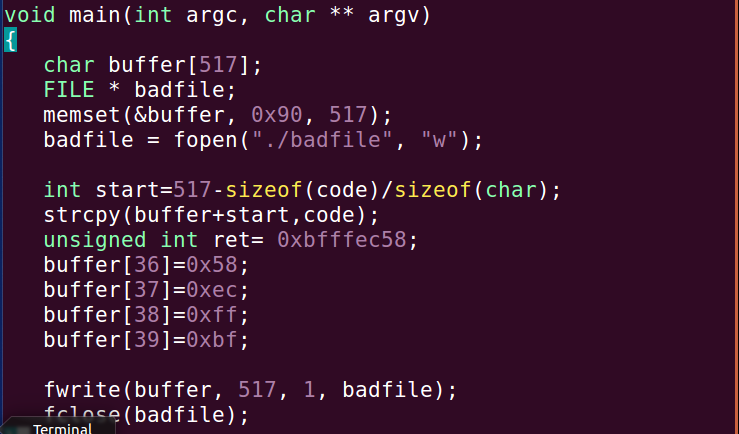
使用 b bof命令对bof设置断点，然后run，在程序暂停时，查看相关数据





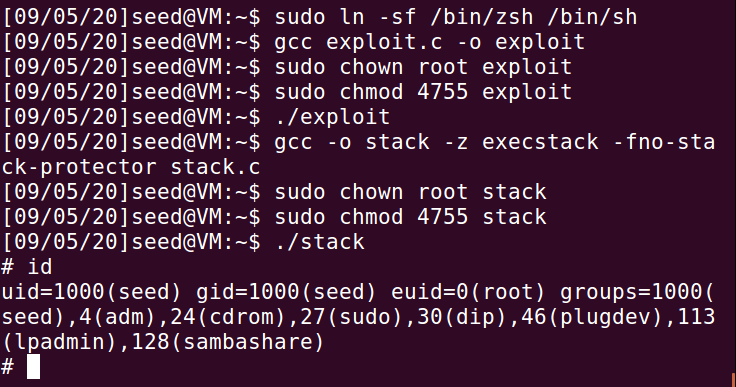
使用p $ebp查看ebp寄存器此时的值，使用p &buffer查看buffer数组的地址。ebp寄存器此时的值再加4就是bof函数返回地址所在的内存地址，通过p/d就可以计算出buffer与bof返回地址之间的距离为32+4=36

因此，exploit.c的程序更改为：



将0x90填满buffer，在buffer的尾部填充shellcode，在第36-39字节处，填写的是我们希望bof函数返回的地址。因此，我们选择一个与ebp值相距较近的，不包含0x00的地址0xbfffec58，按小端序填入buffer。

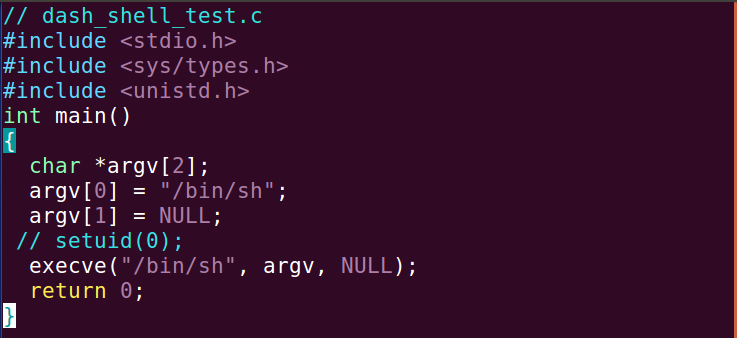
之后再次编译stack.c。在编译之前，我们要先更改路径，将sh更改为zsh。

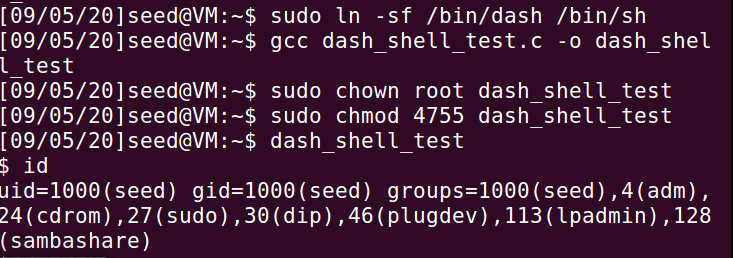


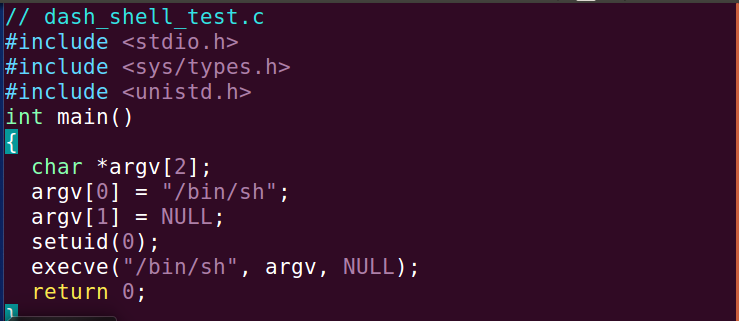
观察运行结果，我们成功得到了shell，并且使用id可以查看到当前RUID为1000，euid=0，也就是成功拿到了root用户权限。

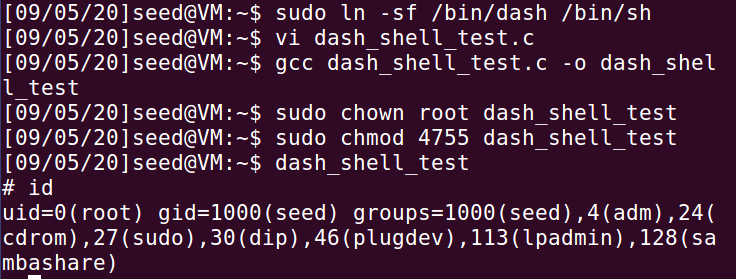
**Task 3: Defeating dash’s Countermeasure**

首先我们要分析setuid(0)在dash\_shell\_test的作用，为此我们进行一个对比实验。



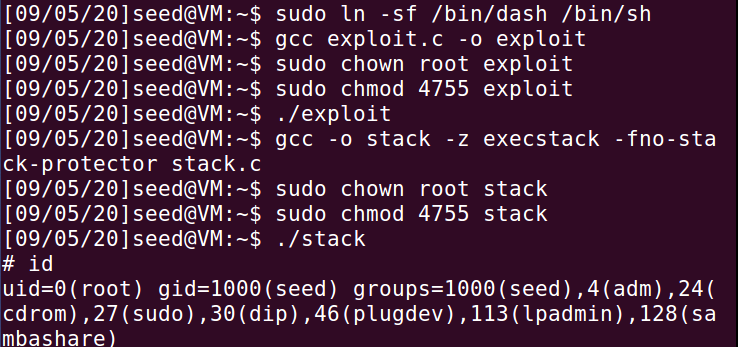
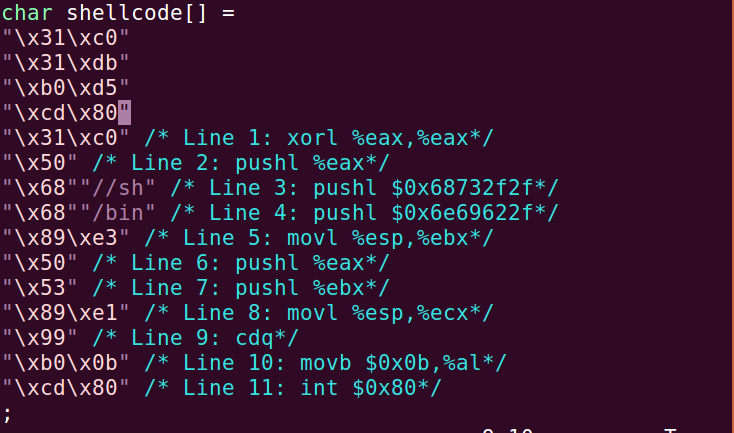






将setuid(0)注释后，我们发现此时无法获得root权限，加入setuid(0)后，成功获得了root权限。

再次基础上，修改exploit.c，再次运行Task 2中的实验，发现也是成功获得了root权限。

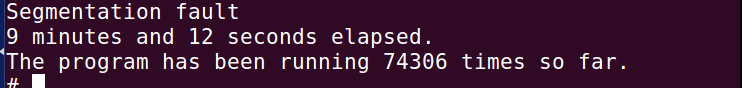


分析dash源代码我们可以发现当用户EUID与RUID不相同时，dash会自动将程序权限降级，但setuid(0)这个命令就是将RUID也设置为root，这样两者相同，就不会出现降级问题。



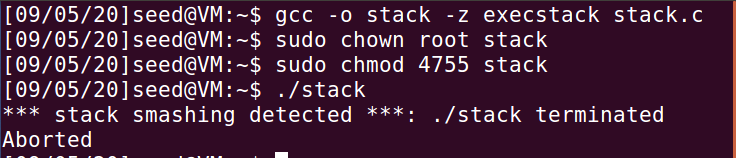
**Task 4: Defeating Address Randomization**

运行脚本，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

