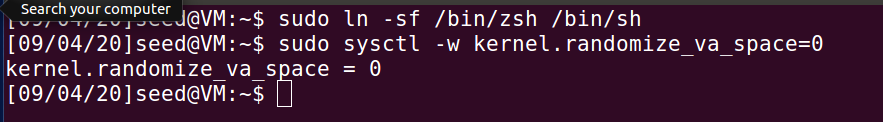
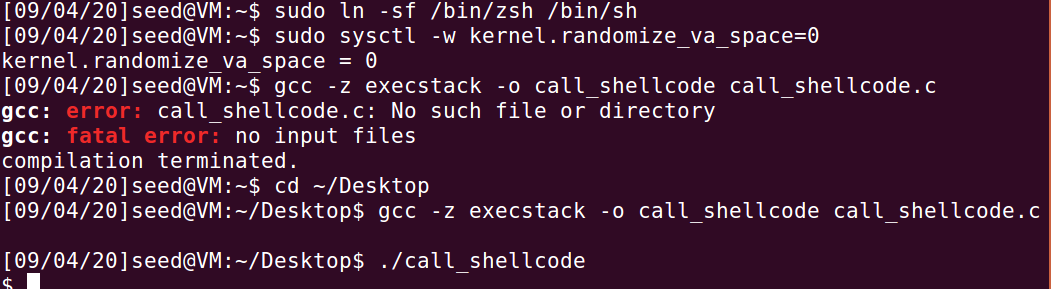
**Buffer Overﬂow Vulnerability Lab**

**Task 1：Running Shellcode**



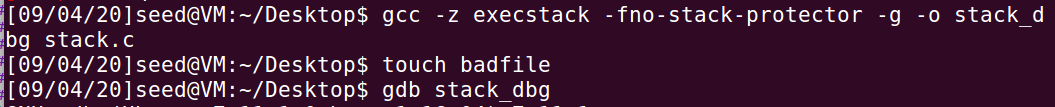
关闭地址随机化，并将/bin/sh链接到/bin/zsh

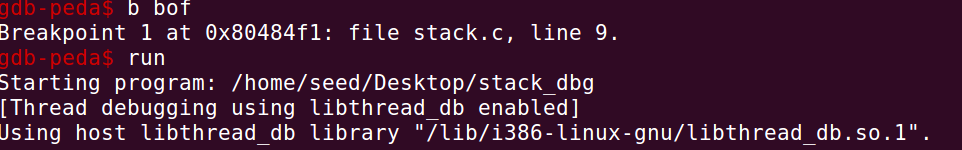


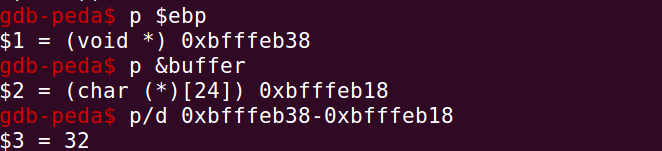
运行编译程序后，zsh被调用

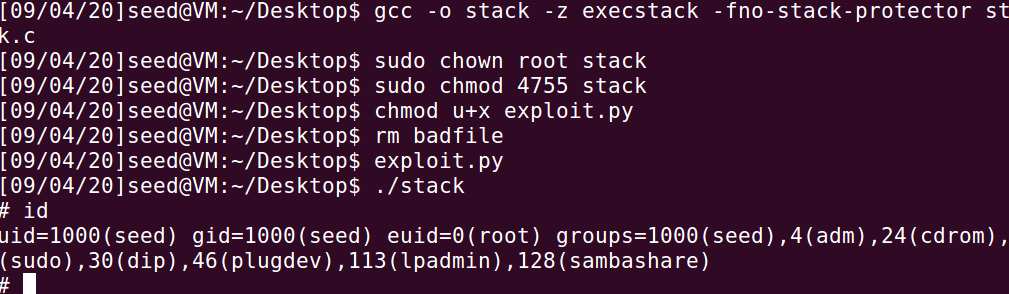
**Task 2：Exploiting the Vulnerability**

使用gdb设置断点，找到ebp的地址。在bof（）处设置断点，继续运行。通过buffer数组首地址和%ebp的地址，发现相差32字节。Offset=32+4=36。修改exploit.py代码，并执行，生成个badfile文件。重新编译stack.c，发现已经获得root权限。栈溢出漏洞利用成功能。







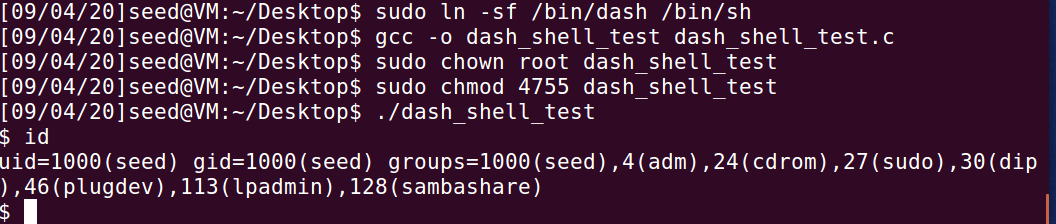


**Task3: Defeating dash’s Countermeasure**

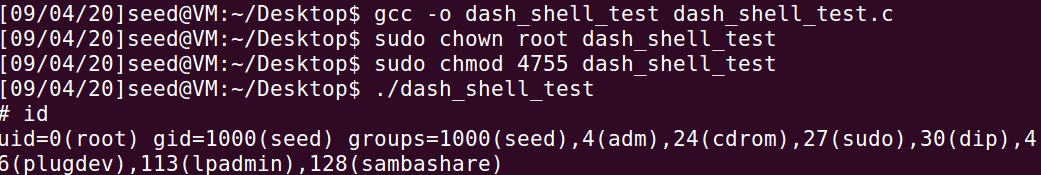
将/bin/sh 软链接至/bin/dash

****

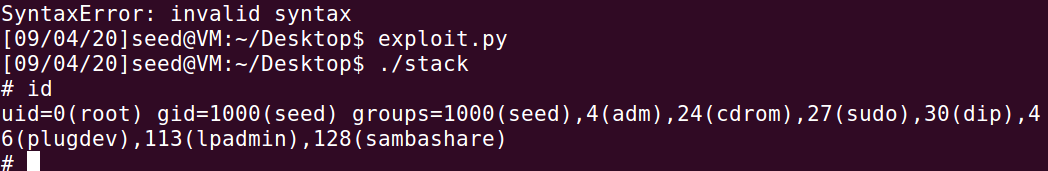
运行程序，发现用户为seed

****

取消注释后，发现用户变回root

****

更改exploit.py后，再次运行stack，成功获得root权限

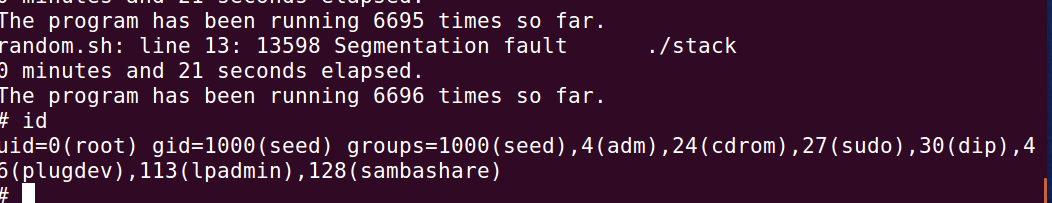
****

**Task 4: Defeating Address Randomization**

打开随机地址

****

21秒后，成功通过暴力破解方式，得到root权限

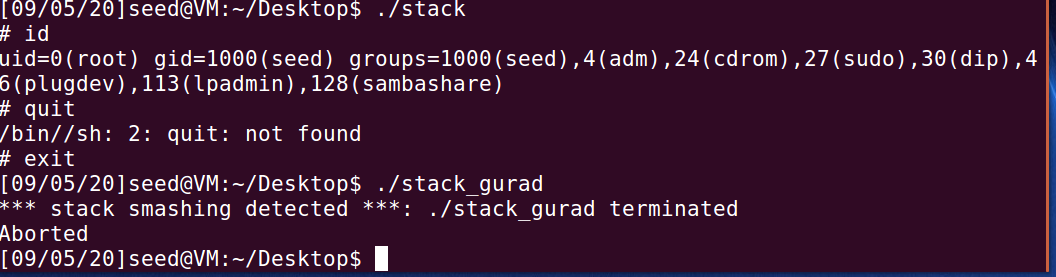
****

**Task 5: Turn on the StackGuard Protection**

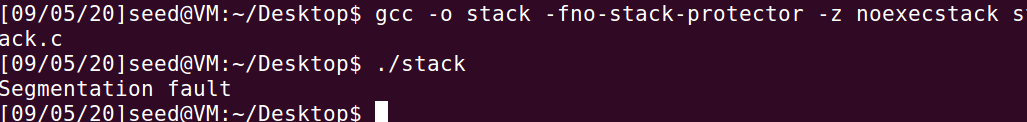
**关闭地址随机化。**

****

**在打开stackguard的情况下，编译得到stack\_gurad ，栈溢出会被检测到并且程序被aborted**

****

**Task6: Turn on the Non-executable Stack Protection**

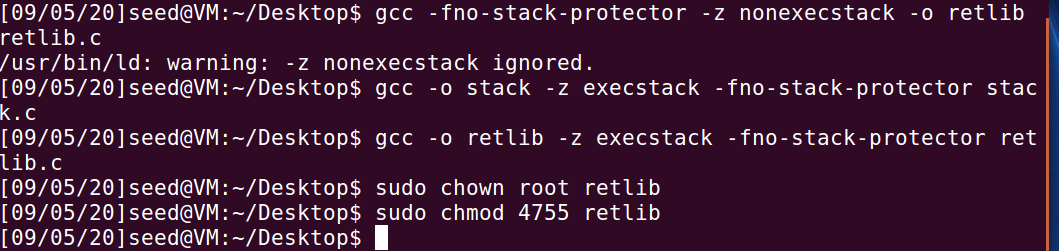
****

打开不可执行栈的保护机制后，可以看到我们不再能执行stack程序，因为已经被检测出栈溢出。

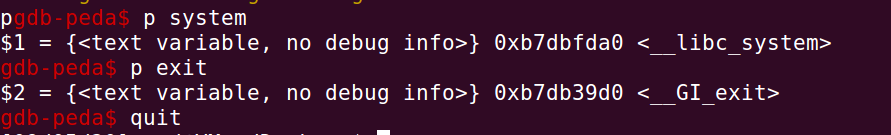
**Return-to-libc Attack Lab**

**Task1：**

1. **关闭保护，编译retlib.c 并设置为特权程序。**

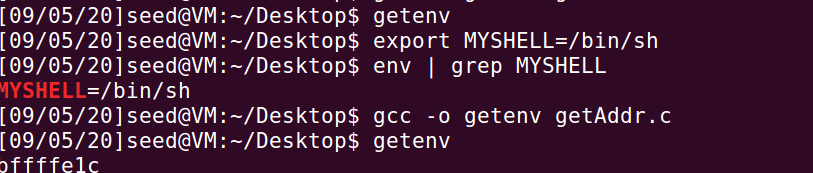
****

1. **利用gdb调试找到system和exit的地址**

****

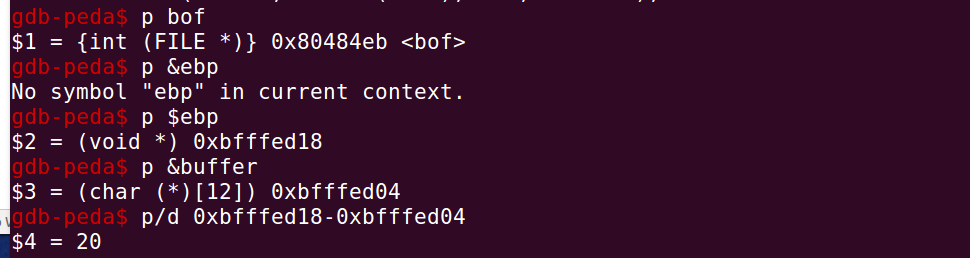
**Task2：Putting the shell string in the memory**

利用报告中的程序，找到MYSHELL的地址

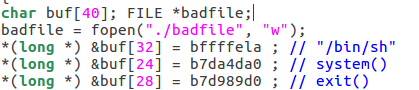
****

**Task3：Exploiting the buffer-overﬂow vulnerability**

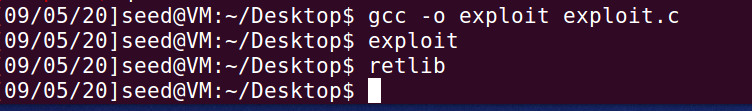
1. **查找ebp和buffer的位置**

****

1. **修改漏洞攻击代码**

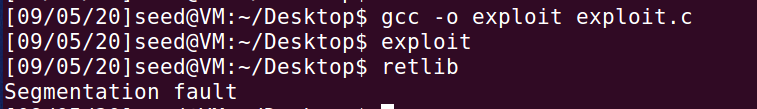
****

1. **编译exploit.c 再运行，发现并没有#出现**



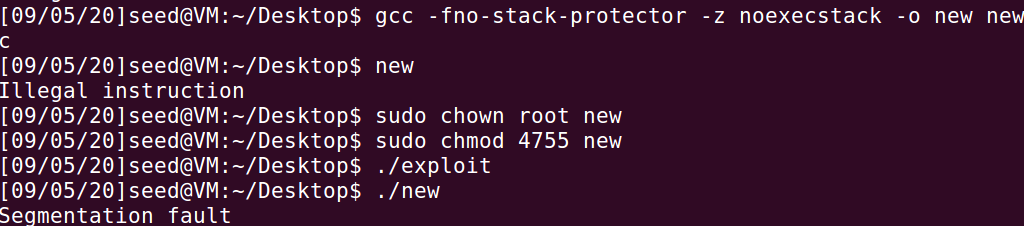
**Attack variation1**

注释掉exit（）地址，重新编译执行，攻击失败



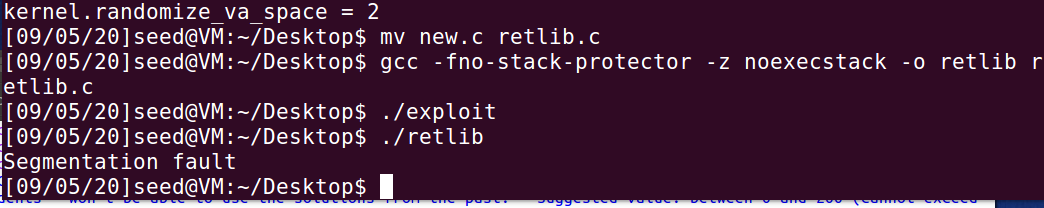
**Attack variation2**

修改文件名，攻击失败

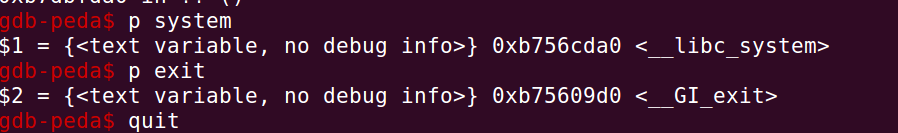
****

**Task 4： Turning on address randomization**

打开地址保护，攻击失败。

****

System 和exit的地址改变。但X/Y/Z 由 ebp 的偏移计算而来，因而不会单独变化

****