Lab2

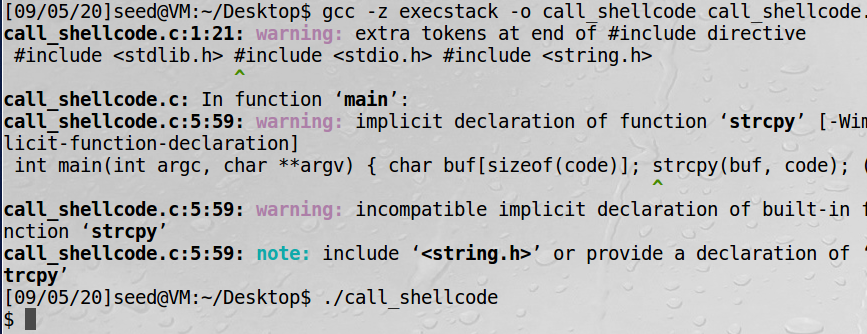
张镇57117126

1. 缓冲区溢出漏洞

Task1.

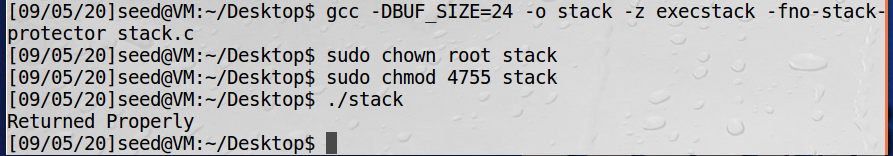
1. 运行shellcode

运行并编译实验手册中的shellcode.c的代码，可以看到终端进入了一个新的shell，说明代码运行成功。



1. 缓冲区溢出漏洞

按照实验手册命令编译运行，结果如下图：

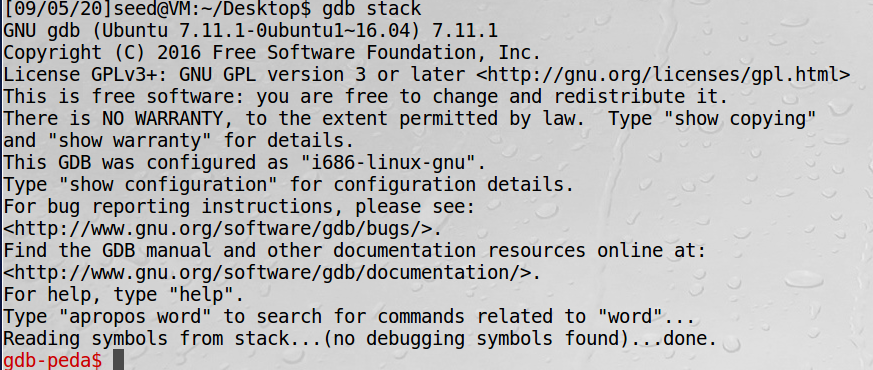


Task2

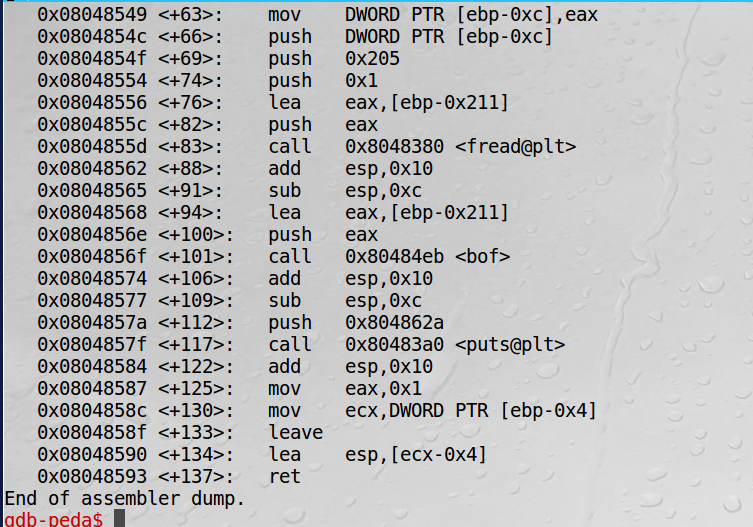
实验过程：

思路是寻找到stack中str的内存基址，之后再将str的前100个字节覆盖为shellcode的内存地址(包含nop指令，即什么都不做)，str的第100-300个字节覆盖为NOP，第300个字节以后填充shellcode的二进制码形式。这样做的基本思想是在字节对齐的情况下通过大量重复shellcode地址淹没return address地址处的内存数据。

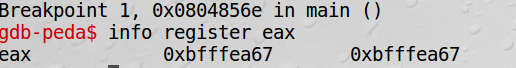
首先用gdb反编译stack文件，寻找str的起始地址：



对main函数的反编译：

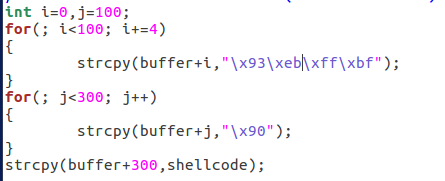


0804856e处的指令push eax代表传入参数str，这说明eax寄存器保存了str的地址。我们在这里设置一个断点，并且运行程序。

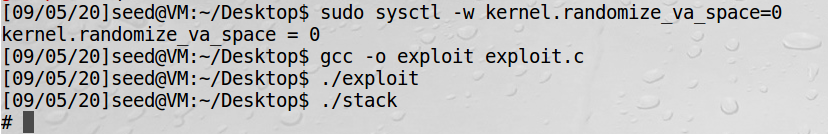


str的地址为bfffea67

对exploit.c文件进行补充：



编译并运行，结果如下图：



成功地利用了缓冲区溢出漏洞使得函数返回时跳转到了预先设定好的shellcode处。缓冲区溢出的核心思想是定位return address的内存地址，并利用已知的地址去覆盖到，这个已知的地址可以帮助我们快速转至shellcode处，执行我们的shellcode。

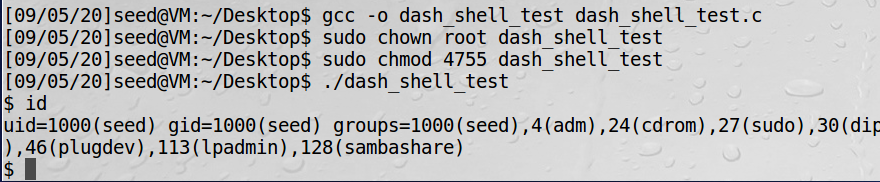
Task3 Defeating dash’sCountermeasure

实验过程：

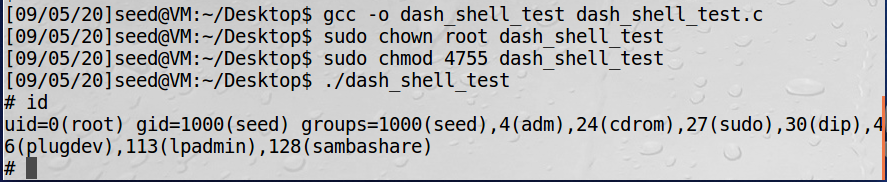
首先将/bin/sh软连接到/bin/dash：

C:\Users\lenovo\AppData\Roaming\Tencent\Users\345312414\QQ\WinTemp\RichOle\]38Q32D}~QWBV$_XHAZ75IY.png

注释掉setuid语句情况下运行并编译dash\_shell\_test.c：

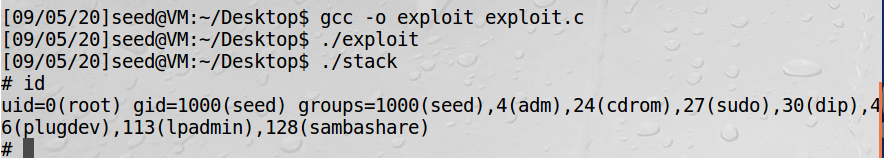


下面不注释掉setuid语句情况下运行并编译dash\_shell\_test.c



setuid可以使程序uid变为0，获得root权限。

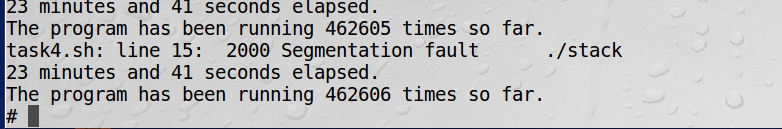
修改exploit.c中的shellcode，重新进行task2中步骤，如下图：



Task4

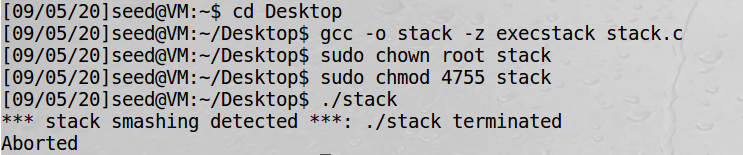
首先让地址重新随机化：

C:\Users\lenovo\AppData\Roaming\Tencent\Users\345312414\QQ\WinTemp\RichOle\)RRT52%)K{HATHNG$)$MJ67.png



Task5

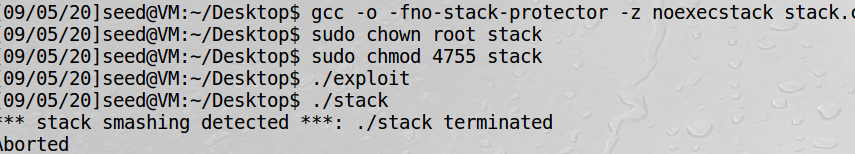
打开栈保护机制，再次编译运行stack程序：



可以看到程序检测到异常，停止运行。

Task6

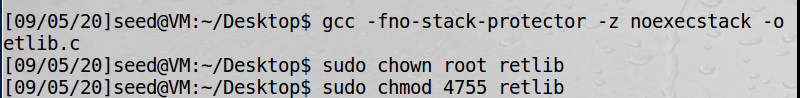
观察栈的不可执行的机制效果，重新编译Task2：



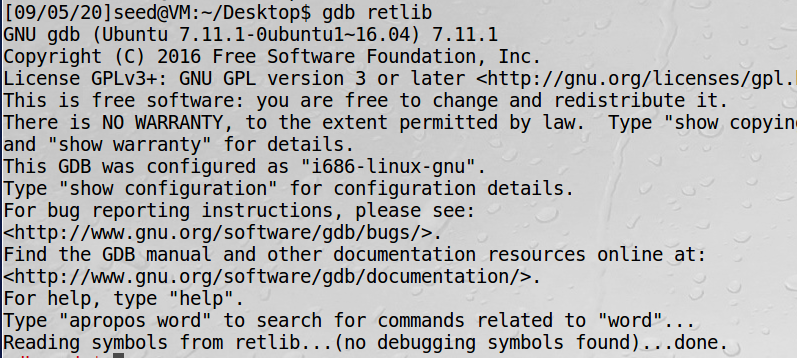
1. Return-to-libc Attack Lab

Task1

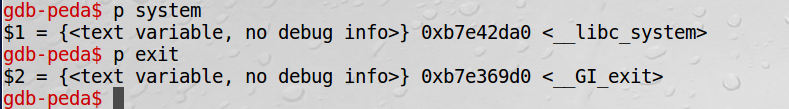
根据实验手册上的，运行并编译retlib.c：



对retlib用gdb进行调试：

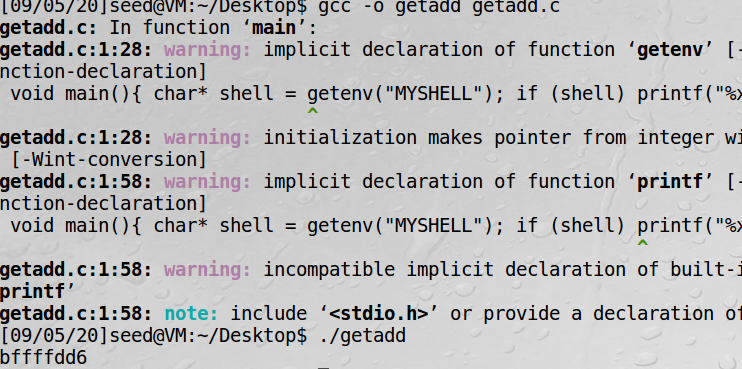


运行程序，并用p system和p exit找地址：



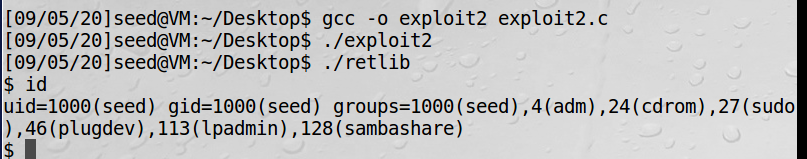
Task2

实验过程：将字符串/bin/sh放入内存，并且寻找到它的地址。



Task3

再编译运行pdf上的代码：



缓冲区的漏洞成功被利用。

Task4

开启地址随机化：

C:\Users\lenovo\AppData\Roaming\Tencent\Users\345312414\QQ\WinTemp\RichOle\(HG6NUCB@0]U]}%EL7RMYZ8.png

再次运行retlib

C:\Users\lenovo\AppData\Roaming\Tencent\Users\345312414\QQ\WinTemp\RichOle\$KPH55_~HCKHSYRD[IHCAXV.png

发生了段错误，说明地址随机化原先寻找到的地址都无效了。