# 补丁实验

## 1.编写漏洞代码并编译

#include<unistd.h>

#include<stdlib.h>

#include<stdio.h>

#include<math.h>

ssize\_t output\_something();

int main()

{

char buf[10];

puts("Input a char(c or q): ");

gets(buf);

if(buf[0]=='c')output\_something();

else if(buf[0]=='q')exit(0);

else

{

puts("Something error\n");

return 1;

}

return 0;

}

ssize\_t output\_something()

{

char buf[5];

puts("Please input your address:");

read(0,&buf,10);

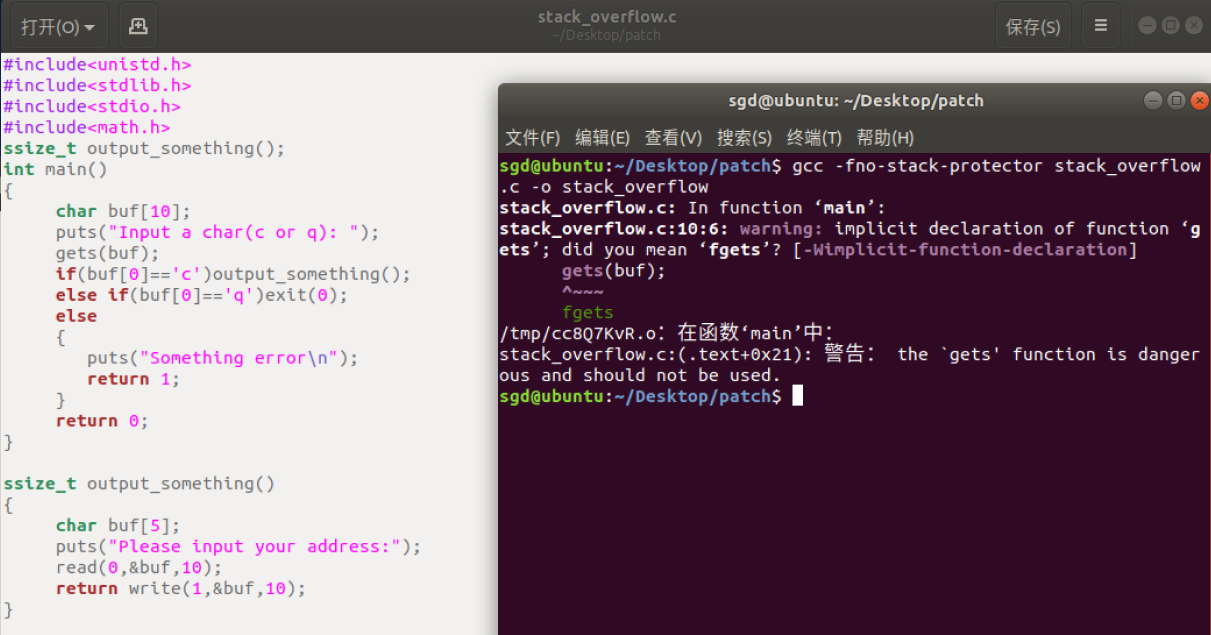
return write(1,&buf,10);

}

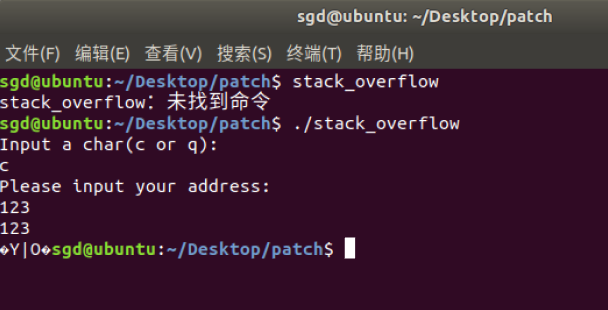
代码main函数中用4字节大小的数组buf保存用户输入的选项，但使用了gets函数获取输入未进行边界限制，可能导致栈溢出。

output\_something函数用5字节大小的buf数组缓存用户接下来输入的字符串。但是接下来的read要读取10个字节，超过buf大小，可能造成栈溢出。write函数输出50字节，超过buf大小，可能导致栈中数据泄露。

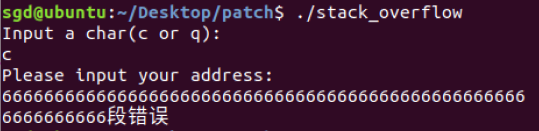
编译gcc -fno-stack-protector stack\_overflow.c -o stack\_overflow



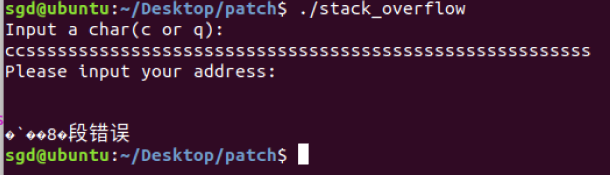
之后执行：



read溢出:

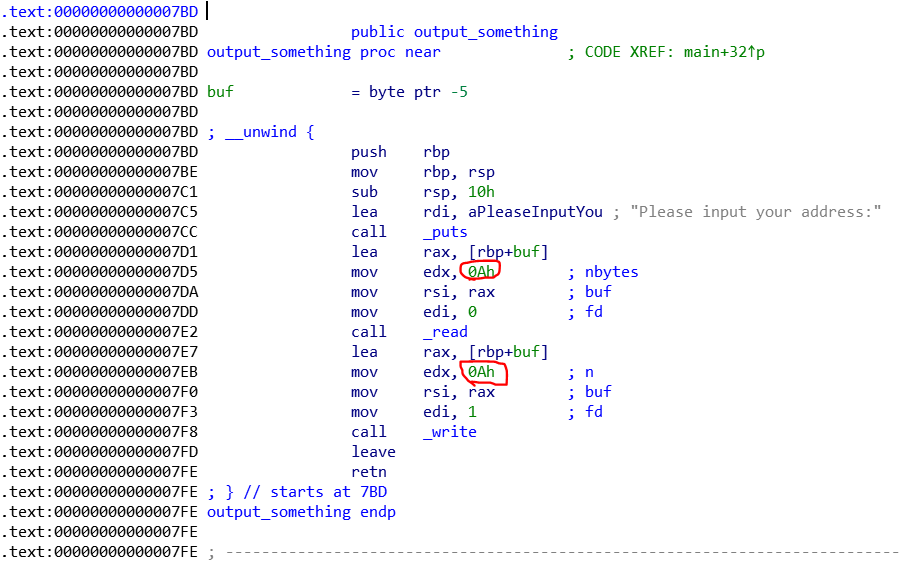


gets栈溢出：

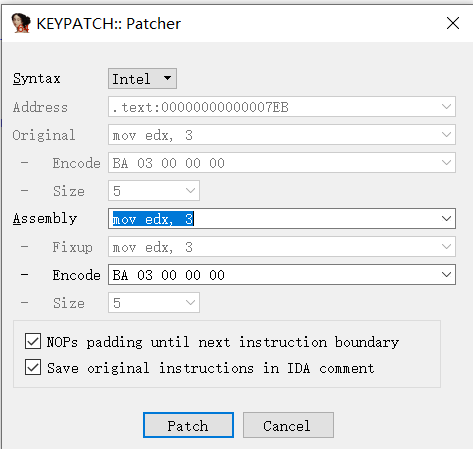


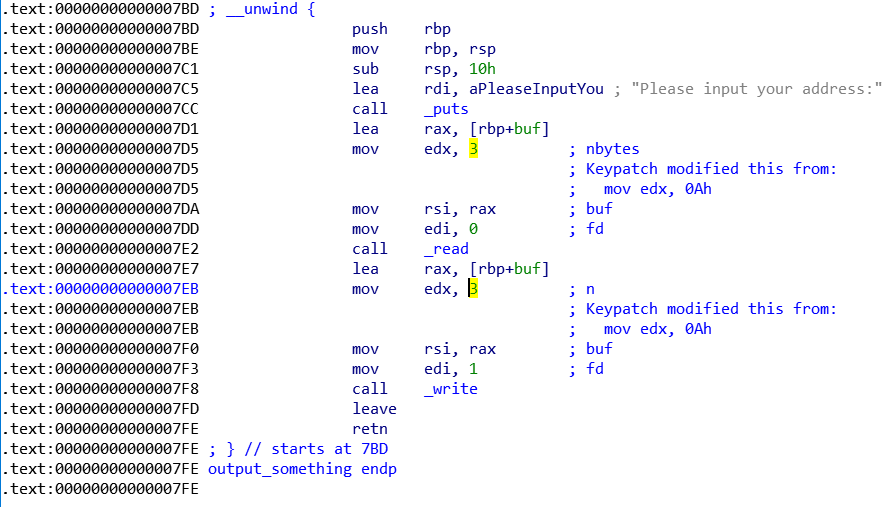
## 2.修补read函数和write函数的漏洞

用ida64打开程序stack\_overflow，在output\_something函数中找到调用read、write函数的位置，发现读取和显示的字节数为0A（10）字节。

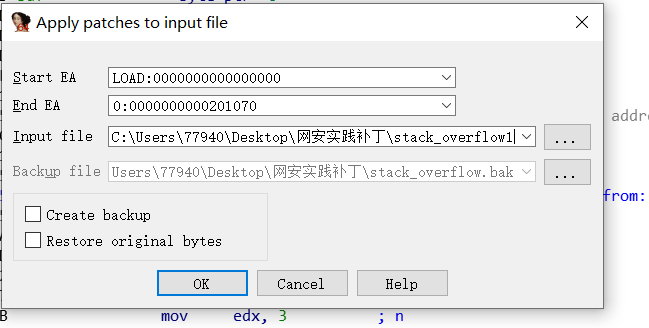


将这里的0Ah修改成小于5字节的数值。

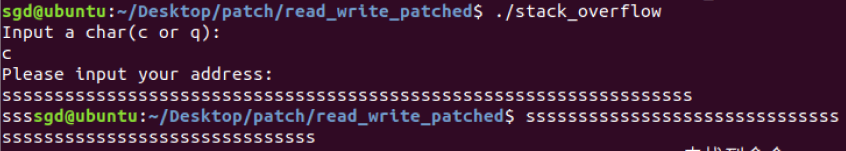




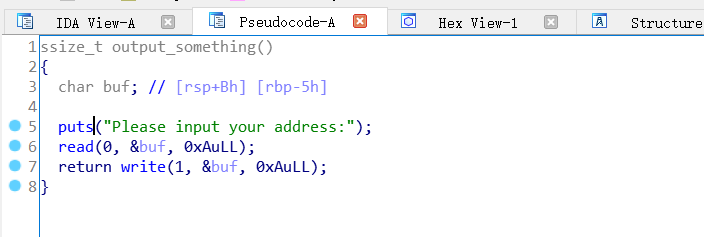
将修补后的文件保存：



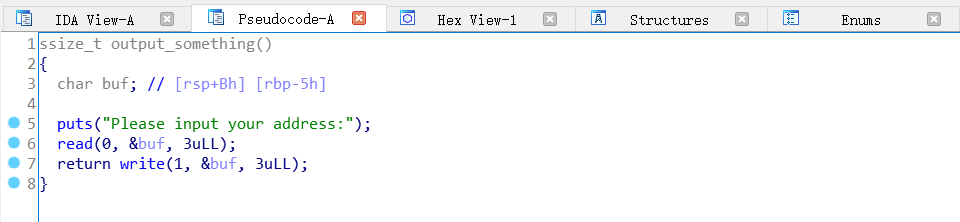
再次测试，发现read、write的漏洞被修补



修改漏洞前的代码



修改漏洞后的代码：



## 3.gets漏洞修补：

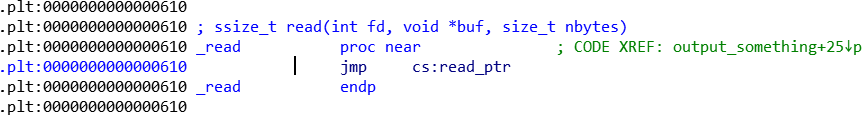
lea rax, [rbp+buf]

mov edx, 3

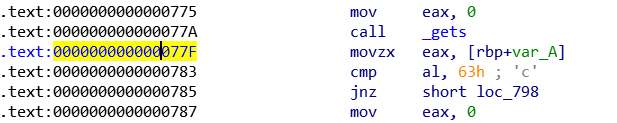
mov rsi, rax

mov edi, 0

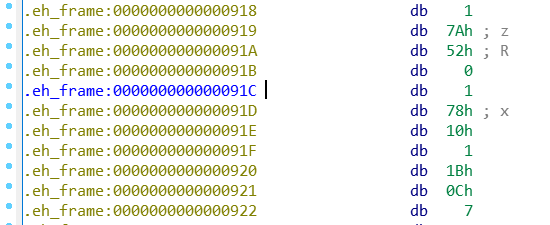
查询read函数的地址，位于0x000610



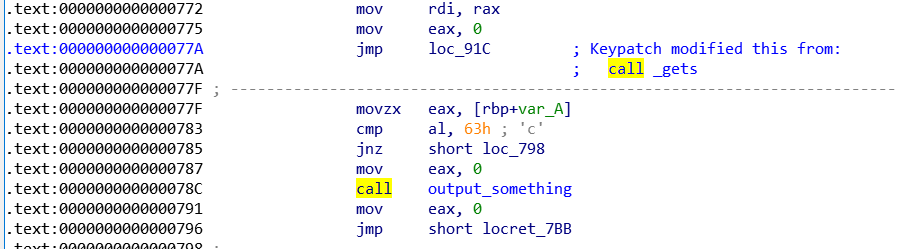
在main函数中查询gets执行完的下一条地址，位于0x00077F



选择eh\_frame的0x00091C处填写漏洞修补代码



将call \_gets处修改为jmp 0x00091C



lea rax, [rbp+s1]

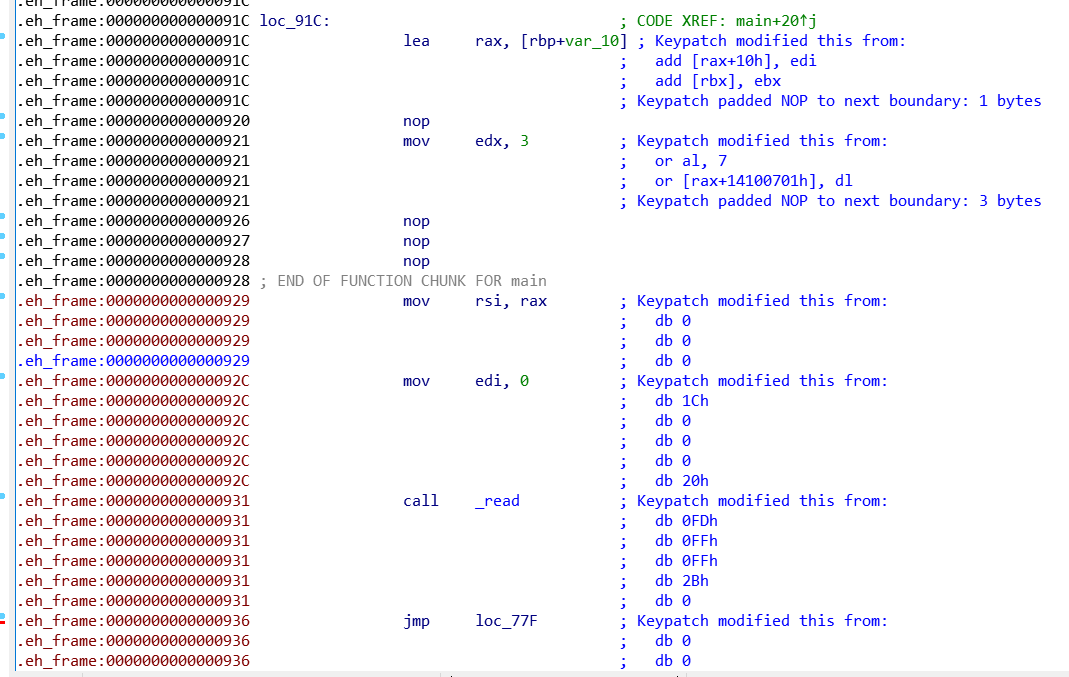
mov edx, 3

mov rsi, rax

mov edi, 0

call 0x000610

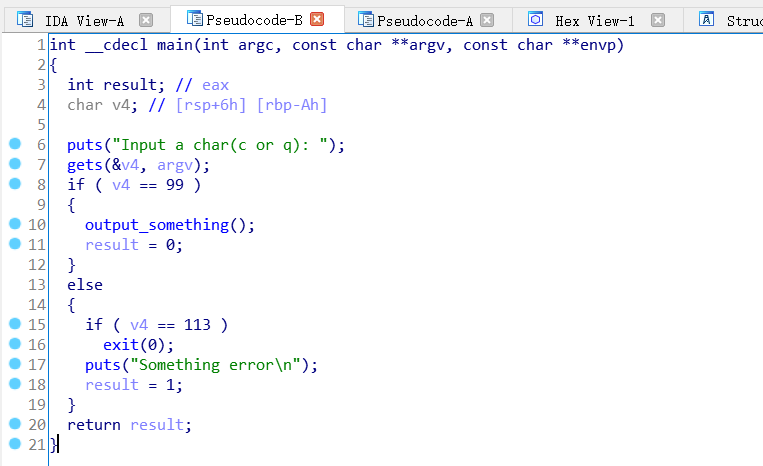
jmp 0x00077F



将修改保存到文件，继续测试，发现gets的栈溢出漏洞已修补



（1）修改gets漏洞前的代码



（2）修改gets漏洞后的代码

