dictor的结构非常简单：

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  char buf; // [rsp+0h] [rbp-100h]

  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  alarm(0x1Eu);
  read(0, &buf, 0xFFuLL);
  printf(&buf, &buf);
  return 0LL;
}
```

printf这里明确地存在格式化串漏洞。
格式化串漏洞可以做两件事： 任意读/写一次.

控制RIP

首先是要想办法控制RIP：
调试看到利用格式化串可以改掉栈上指针指向的值，这就意味着如果知道一个地址，那么就可以把它放到栈上修改掉它指向的值.

但是这样是无法修改main函数的rbp的，因为它的地址我们不知道.
所以接下去看在fini_array里面找到了一个预留的函数指针，那么用格式化串写那个就可以控制执行流了。

```
00400776 ; Attributes: bp-based frame
00400776
00400776 sub_400776      proc near               ; DATA XREF: .fini_array:0000000000600DE0↓o
00400776 ; __unwind {
00400776                 push    rbp
00400777                 mov     rbp, rsp
0040077A                 mov     rdx, cs:qword_601050
00400781                 mov     eax, 0
00400786                 call    rdx ; qword_601050
00400788                 nop
00400789                 pop     rbp
0040078A                 retn
0040078A ; } // starts at 400776
0040078A sub_400776      endp
0040078A
```

ok,劫持执行流的同时可以泄露libc地址，然后跳回main.

跳回start_, 使printf调malloc

为什么要这样操作，其实是因为没办法再调一次fini*array*了.
*fini*array实际上是在libc*start*main的init函数中被作为参数传入的，和exit相关.
但是我们跳回main，再度执行到exit时不会触发这个函数指针了。

那么，也就没有办法再度劫持执行流。

所以，控制执行流跳回start，这里由于题目设计，形成了一个循环。

如何再度劫持执行流？
那就利用printf在打印大量(具体数目不清楚)时会调用malloc来做就ok了。
由于第一步泄露了libc，这里改掉malloc_hook.

使printf打印大量内容。就能够劫持执行流了。