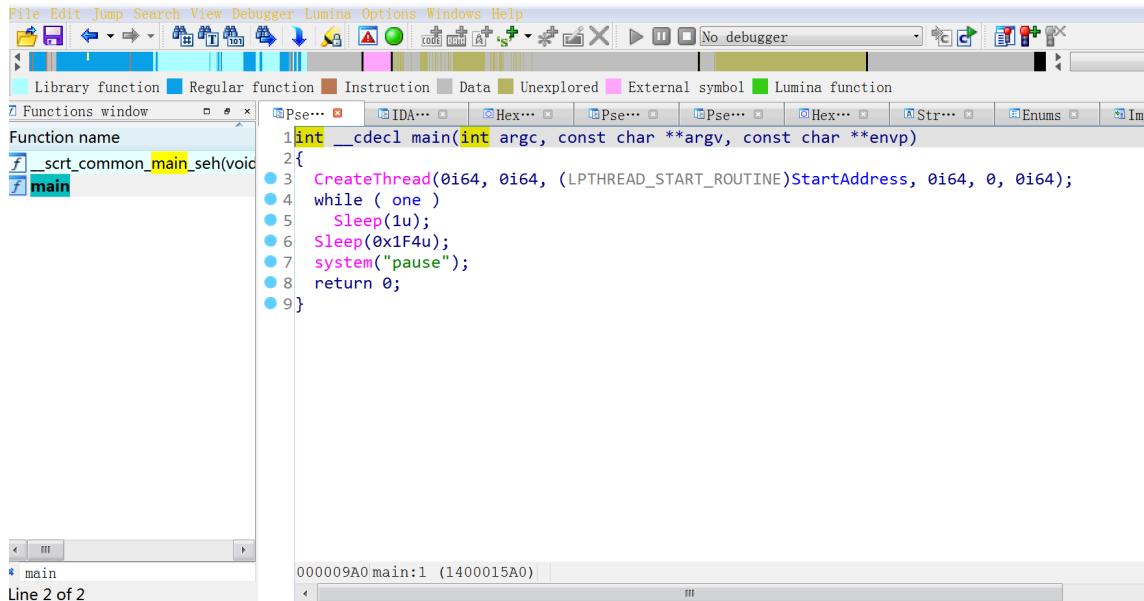


Level-空（什么时候能做派蒙啊）

拿到题先看看，程序逻辑是输入一串字符，如果正确则输出“重庆没水没电没网大学”，如果正确则输出“好”，看起来像是普通的re签到题，不管，丢ida，找到main函数（ctrl+f寻找），无脑f5开始研究。

## 静态调试部分（这一段其实基本都是踩的坑，可以忽略绝大部分直接看动态调试部分）



【ps：找main函数时候其实踩了巨多坑。我用的是ida7.0，它并不能自动识别main函数，但其实也有办法，start跟进就好了，奈何我菜，研究了很多天没有一点头绪（因为这个main函数里面感觉也很绕（由于本来就不应该静态调试（这也是后话了）））。直到后来问了大佬于是更新了7.5版本这个问题才得以解决】

While ( ) 括号后面原本是一串指令，跟进后显示数值为1，可以理解为一个死循环。无限执行sleep函数。

```
.data:00000000140005034 dw 01h ; DATA XREF: __SCRT_15_1
.data:00000000140005034 one dd 1 ; DATA XREF: main+23↑r
.data:00000000140005034 ; main+3B↑r
```

这个sleep函数，参考百度上的定义：Sleep函数可以使计算机程序（进程，任务或线程）进入休眠，使其在一段时间内处于非活动状态，括号后的数则是休眠的时间。

也就是在休眠这么一段时间后中止程序。

所以 create thread ( ) 以后的句子可以理解为没什么用了。我们就把重心放在这个create thread函数上。

create thread ()，百度后知这是一个建立新线程的函数，其中一个参数是在新线程中需要被执行的函数。也就是这个startaddress。【ps: 0i64也就是0，create thread在被调用的时候需要有六个参数，这里把其他几个设置成0，可能是没必要吧。。。】我们跟进它。

```
1 __int64 __fastcall StartAddress(LPVOID lpThreadParameter)
2 {
3     DWORD (__stdcall *i)(LPVOID); // rax
4     DWORD flOldProtect; // [rsp+48h] [rbp+10h] BYREF
5
6     VirtualProtect(loc_1400011A0, (char *)main - (char *)loc_1400011A0, 0x40u, &flOldProtect);
7     for ( i = loc_1400011A0; (unsigned __int64)i < (unsigned __int64)main; i = (DWORD (__stdcall
8         *(_BYTE *)i ^= 0x44u;
9     CreateThread(0i64, 0i64, loc_1400011A0, 0i64, 0, 0i64);
10    return 0i64;
11 }
```

第一句定义函数基本信息后。第三四没有看懂，也许是某种定义。不是关键句。

第六句，virtual protect。百度得“VirtualProtect，是对应 Win32 函数的逻辑包装函数，它会在调用处理程序的虚拟位置空间里，变更认可页面区域上的保护”，对于这句话我并不能深刻理解，但是我们可以知悉这个函数的调用参数的含义。

loc\_1400011A0 目标地址起始位置

(char \*)main - (char \*)loc\_1400011A0 内存大小

后面两个可以忽略。

这一句大概就是针对于从main函数地址到loc\_1400011A0地址的内存属性进行改变。具体目的可能是为了实现保护吧。

也不是关键的东西

第七句是一个循环体。

```
i = loc_1400011A0; (unsigned __int64)i < (unsigned __int64)main; i = (DWORD (__stdcall *) (LPVOID))
```

i=某个数（稍后分析），当i小于main (int)，i对0x44进行异或。

第九句再次创建了一个新的线程，但是这次调用的不是startaddress而是loc\_1400011A0，最后return 0。

我不是很能理解第九句的作用，所以让我们跟进loc\_1400011A0看看这一个指令集的具体作用。

跟进后发现这一指令位于text段。查看交叉引用发现不止startaddress对其进行了引用。

```
40003BB4  RUNTIME_FUNCTION <rva loc_1400011A0, rva sub_140001299, \
```

但这不好。我并不能看懂。百度过后我们知道它和动态代码有关，但我选择战略性忽略。

```

ext:000000001400011A0 loc_1400011A0: ; DATA XREF: StartAddr
ext:000000001400011A0 ; .rdata:00000000140003
ext:000000001400011A0 add al, 17h
ext:000000001400011A2 or al, 0C5h
ext:000000001400011A4 test al, 0D4h
ext:000000001400011A6 db 44h, 44h
ext:000000001400011A6 or al, 0CFh
ext:000000001400011AB sbb al, 5Bh ; '['
ext:000000001400011AE db 44h
ext:000000001400011AE or al, 0C9h
ext:000000001400011B2 push rcx
ext:000000001400011B3 jnz short loc_14000121B
ext:000000001400011B5 db 44h, 44h
ext:000000001400011B5 adc rax, [r15+r14*2+554B84h]
ext:000000001400011B5 : -----

```

直接看汇编指令。  
首先明确：我们不知道al里的值，设为a。

a与17h进行与运算得b  
b与0c5h进行或运算得c  
Test运算不会改变al的值，忽略。  
C与0cfh或运算得d  
D与5bh进带借位减法得e  
E与0c9进行或运算得f  
之后就没有针对al的运算指令了。  
看下一句  
push rcx后条件转移到loc\_14000121B,

```

000121B loc_14000121B: ; CODE XREF: .text:000000001400011B5
000121B mov esi, 858ECF46h
0001220 lodsd
0001221 pop rbx
0001222 xchg eax, r13d
0001224 leave
0001224 : -----

```

Mov 858ECF46h至esi,  
lodsd指令指从 [esi]加载到 eax  
pop rbx 出栈  
交换eax和r13d数据  
清栈。  
由于已经跳转，战略性忽略adc rax, [r15+r14\*2+554B84h]这一条指令。（看不懂）（）

解决i的问题需要一个多元一次方程，这里我们先放一放。

来看main，这里main是一个int类型数，作为一个c语言分数不尽人意的菜鸡，我姑且猜测它为main函数的返回值，也就是0。

那么再重新看之前的循环体。

“l=某个数（稍后分析），当l小于main（int），i对0x44进行异或。”

l=a=f，当lf小于0，

F对0x44异或得x

F再次进行之前的运算分别得g, h, i, j, k, 但是没啥用。

代码到这里就快分析完了。但是我们仍未知道那天我们寻找的字符是多少，我们甚至没有寻找哪怕printf或者scanf或者getchar。

我们可以肯定这些东西被巧妙地隐藏起来了。

## Ida字符串搜索“give”

```

00033D0 aBadcast      db 'Bad cast',0
00033E1          align 8
00033E8 aGiveMeSomeWord db 'Give me some words > ',0
00033FE          align 20h
0003400 aRedrock      db 'Redrock',0
0003408 aChongqingNowat db 'Chongqing No Water No Electric No Network University'
000343D          align 20h
0003440 aNice        db 'Nice!',0
000344C          db 0

```

可以看到程序显示的字符都在这些只读数据段中，但是没有任何的交叉引用。

然后我没有思路，直到hint提示：smc

?

## What is smc?

SMC (Self-Modifying Code) (自解码)，可以在一段代码执行前对它进行修改。常常利用这个特性，把代码以加密的形式保存在可自行文件中，然后在程序执行的时候进行动态解析。这样我们在采用静态分析时，看到的都是加密的内容，从而阻断了静态调试的可能性。

[illegible]

在坚信我没有白忙活的同时，我开始学习 动态调试。

## 动态调试部分

从天而降的hint总算是给弹尽粮绝的可怜人一点希望了。

**（我对于）这道题的动态分析部分的思路就是，先找到被smc代码加密的关键函数**

姑且猜测这个函数就是之前跟进以后内容非常诡异的loc 1400011A0

我们就地取材，就用ida自带的动态调试器进行动态调试。

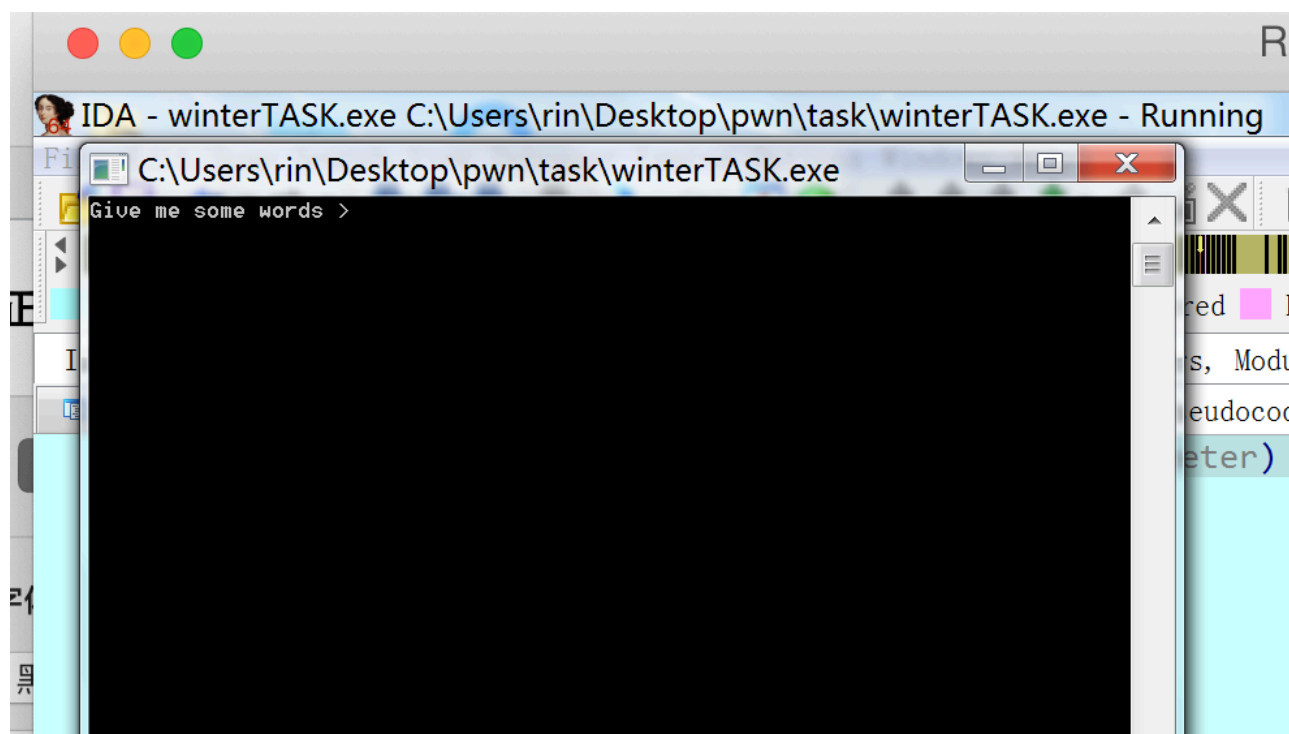
```

4  DWORD f10IdProtect; // [rsp+48h] [rbp+10h] BYTE
5
6  VirtualProtect(sub_1400011A0, (char *)main - (char *)sub_1400011A0, 0x40u, &f10IdProtect);
7  for ( i = sub_1400011A0; (unsigned __int64)i < (unsigned __int64)main; i = (__int64 (__fast
8  *(_BYTE *)i ^ 0x44u;
9  CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)sub_1400011A0, 0i64, 0, 0i64);
10 return 0i64;
11}

```

(并且在动态调试器中由于程序已经被执行，所以代码会和未执行前的静态代码有区别)

直接对第九句下断点，查看运行情况



可以看到（我们熟悉的）主程序已经被执行一部分了，所以我们想要寻找的被加密的函数应该就藏在第九句。我们的猜测似乎被验证了一部分

```

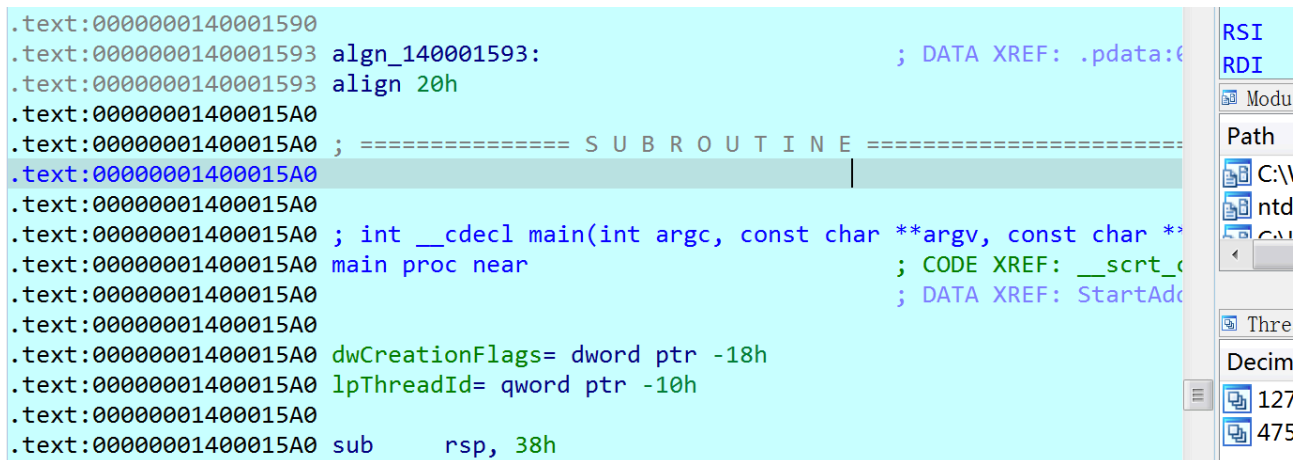
.text:0000000014000119F align 20h
.text:000000001400011A0
.text:000000001400011A0 ; DWORD __stdcall loc_1400011A0(LPVOID lpThreadParameter)
.text:000000001400011A0 loc_1400011A0: ; DATA XREF: StartAddr
.text:000000001400011A0 ; .rdata:000000001400011A0
.text:000000001400011A0 add     al, 17h
.text:000000001400011A2 or      al, 0C5h
.text:000000001400011A4 test    al, 0D4h
.text:000000001400011A6 db      44h, 44h
.text:000000001400011A6 or      al, 0CFh
.text:000000001400011AB sbb     al, 5Bh ; '['
.text:000000001400011AE db      44h
.text:000000001400011AE or      al, 0C9h
.text:000000001400011B2 push    rcx
.text:000000001400011B3 jnz     short loc_14000121B

```

继续跟进loc\_1400011A0

继续我们的猜测，我们缺少一个能够加密我们输入字符串并将其加密后与原本的字符串进行对比然后输出重庆balabala大学的函数。

由于smc代码对该段进行了加密，所以ida无法将其正确识别成某个单独的函数。所以当我们获得了函数的开始的地址，我们得再找到它的结束地址。



```
.text:00000000140001590
.text:00000000140001593 align_140001593: ; DATA XREF: .pdata:0
.text:00000000140001593 align 20h
.text:000000001400015A0
.text:000000001400015A0 ; ===== S U B R O U T I N E =====
.text:000000001400015A0
.text:000000001400015A0 ; int __cdecl main(int argc, const char **argv, const char **
.text:000000001400015A0 main proc near ; CODE XREF: __scrt_0
.text:000000001400015A0 ; DATA XREF: StartAdd
.text:000000001400015A0
.text:000000001400015A0 dwCreationFlags= dword ptr -18h
.text:000000001400015A0 lpThreadId= qword ptr -10h
.text:000000001400015A0
.text:000000001400015A0 sub rsp, 38h
```

如图000000001400015A0后为main。

所以这个函数其实是在main执行之前就执行了吗，还是说ida里面数据段的排布是混乱的？

在经过对000000001400011A0和000000001400015A0之间的一些奇怪的函数的测试之后，我们终于发现如果把这个地址区间的数据转换成一个完整的函数，那么奇迹就会发生。

因为ida显示的是它瞎识别以后的结果，我们先把这一段数据变为undefine后，手动将它转为code，再在开头创建函数

F5发现可以反汇编



```

ext:00000001400011A0 ; DWORD __stdcall sub_1400011A0(LPVOID lpThreadParameter, int)
ext:00000001400011A0 sub_1400011A0 proc near ; DATA XREF: St
ext:00000001400011A0 ; .rdata:00000001
ext:00000001400011A0 var_78= xmmword ptr -78h
ext:00000001400011A0 var_68= xmmword ptr -68h
ext:00000001400011A0 var_58= xmmword ptr -58h
ext:00000001400011A0 var_48= xmmword ptr -48h
ext:00000001400011A0 var_38= xmmword ptr -38h
ext:00000001400011A0 var_28= xmmword ptr -28h
ext:00000001400011A0 var_18= dword ptr -18h
ext:00000001400011A0 arg_0= qword ptr 8
ext:00000001400011A0 arg_8= qword ptr 10h
ext:00000001400011A0
ext:00000001400011A0 push rbp
ext:00000001400011A2 sub rsp, 90h
ext:00000001400011A9 mov rcx, cs:?.cout@std@@@3V?$.basic_ostream@DU?$char_t
ext:00000001400011B0 lea rdx, aGiveMeSomeWord ; "Give me some
ext:00000001400011B7 xorps xmm0, xmm0
ext:00000001400011BA xor eax, eax
ext:00000001400011BC movups [rsp+98h+var_78], xmm0
ext:00000001400011C1 mov [rsp+98h+var_18], eax
ext:00000001400011C8 movups [rsp+98h+var_68], xmm0
ext:00000001400011CD movups [rsp+98h+var_58], xmm0

```

```

int v2; // ebx
__int64 v3; // rax
__int64 v4; // rax
const char *v5; // rdx
__int64 v6; // rax
__int128 v8[6]; // [rsp+20h] [rbp-78h] BYREF
int v9; // [rsp+80h] [rbp-18h]

v8[0] = 0i64;
v9 = 0;
v8[1] = 0i64;
v8[2] = 0i64;
v8[3] = 0i64;
v8[4] = 0i64;
v8[5] = 0i64;
sub_140001760(std::cout, "Give me some words > ");
sub_140001A00(std::cin, v1, v8);
if ( LODWORD(v8[0]) == 1347768643 && BYTE4(v8[0]) == 84 )
{
    v2 = rand() % 10;
    if ( v2 + rand() % 10 > 30 )
    {
        v3 = sub_140001760(std::cout, "Redrock");
        std::ostream::operator<<(v3, sub_140001930);
    }
}
else
{
    v4 = -1i64;
    do
    {
        ++v4;
        while ( *((_BYTE *)v8 + v4) );
        if ( v4 != 8
            || 870732 * SBYTE5(v8[0])
            + 620576 * SBYTE6(v8[0])
            + 687392 * SBYTE3(v8[0])

```

至此我们终于找到了最关键的函数。  
通过分析这个函数我们就能知道程序加密字符串的方式，并且解出字符串。

研究程序后，大致认为是输入一个八位的字符，把字符存入v8这个数组。  
设a1a2a3a4a5a6a7a8,这八个数经过函数中的运算后符合方程式，则输出重庆balabla大学。

```
while ( (BYTE) v8[0] );  
if ( v4 != 8  
|| 870732 * SBYTE5(v8[0])  
+ 620576 * SBYTE6(v8[0])  
+ 687392 * SBYTE3(v8[0])  
+ 790701 * SBYTE4(v8[0])  
- 264980 * SLOBYTE(v8[0])  
- 558068 * SBYTE1(v8[0])  
- 940616 * SBYTE7(v8[0])  
- 805665 * SBYTE2(v8[0]) != -1990197  
|| 242625 * SBYTE5(v8[0])  
+ 230650 * SBYTE4(v8[0])  
+ 460946 * SBYTE3(v8[0])  
+ 269419 * SLOBYTE(v8[0])  
+ 630862 * SBYTE1(v8[0])  
- 24920 * SBYTE7(v8[0])  
- 482510 * SBYTE2(v8[0])  
- 580412 * SBYTE6(v8[0]) != 50416313  
|| 340464 * SBYTE2(v8[0])  
+ 719348 * SBYTE3(v8[0])  
+ 240775 * SBYTE7(v8[0])  
+ -207754 * SBYTE1(v8[0])  
- 470557 * SLOBYTE(v8[0])  
- 719143 * SBYTE5(v8[0])  
- 114858 * SBYTE6(v8[0])  
- 13126 * SBYTE4(v8[0]) != -8167199  
|| 639378 * SBYTE5(v8[0])  
+ 903739 * SBYTE2(v8[0])  
+ 577554 * SBYTE3(v8[0])  
+ 107894 * SBYTE7(v8[0])  
- 860840 * SLOBYTE(v8[0])  
- 657457 * SBYTE6(v8[0])  
- 358459 * SBYTE4(v8[0])  
- 179591 * SBYTE1(v8[0]) != 8821640  
|| 1848 * SBYTE1(v8[0])  
+ 693461 * SBYTE2(v8[0])  
+ 862506 * SBYTE7(v8[0])
```



```

1  from z3 import *
2  a1, a2, a3, a4, a5, a6, a7, a8 = Ints('a1 a2 a3 a4 a5 a6 a7 a8')
3  x = Solver()
4  x.add(209448 * a2
5      + 679897 * a1
6      + 138284 * a8
7      - 982280 * a5
8      - 127647 * a3
9      - 157768 * a7
10     - 679219 * a6
11     - 907473 * a4 == -85049773)
12
13  x.add(773383 * a2
14      + 154384 * a8
15      + 714136 * a6
16      + -592212 * a7
17      - 858737 * a5
18      - 263859 * a1
19      - 130037 * a4
20      - 997096 * a3 == -80151959)
21  x.add(1848 * a2
22      + 693461 * a3
23      + 862506 * a8
24      - 208232 * a7
25      - 664100 * a5
26      - 192669 * a6
27      - 354894 * a1
28      - 644389 * a4 == -31320696)
29  x.add(639378 * a6
30      + 903739 * a3
31      + 577554 * a4
32      + 107894 * a8
33      - 860840 * a1
34      - 657457 * a7
35      - 358459 * a5
36      - 179591 * a2 == 8821640)
37  x.add(802559 * a5
38      + 448649 * a6
39      + 190047 * a2
40      + -256560 * a8

```

```

Give me some words > MAGA2020
Chongqing No Water No Electric No Network University
请按任意键继续. . .

```

运用神奇的z3-solver（这玩意真难装）

解出八个变量的值，依次排序后对应asc2表推出输入字符为MAGA2020（77 65 71 65 50 48 50 48）

输入后得到期盼的答案。

重庆  
没水  
没电  
没网  
大学。

总结（唠叨）：

基于我上学期都去爬墙学web还学的特别烂的情况下我觉得level空真是贼难，虽然之前对二进制安全有过一小段时间的研究（主要都是去啃汇编指令了），但re是真没怎么接触，题做的也都是pwn类型的。所以对于做题中碰到的难点（我认为的）最初都是一头雾水。好在问了请教了一些大佬之后山回路转（）荧妹妹我是真的做不出了，汇编指令我可能还需要再研究一段时间再patch，大致的思路就是直接jmp到重庆没水没电大学或者其他啥的（），但是level派蒙我还是有点信心的（虽然还没有做）毕竟栈溢出老朋友了。