

[开头的唠嗑：

「大部分都是倒垃圾，你可以甚至能看到各种c库函数等等等的百度结果。但是之后我会用md做一个精简版（前提是我做出来了）」

荧

要让任意字符长度小于20的字符串输出，思路大致是
利用hint中的“无脑jmp”，无条件跳转到输出目标字符串的指令。

不管，搞清楚代码逻辑是第一。

国际惯例，一句一句分析（ps：上一个level其实我并没有把代码逻辑完全搞懂，只是因为群里说“八元一次方程”泄露了天机，我想也没想直接把那一堆丢进z3了）。

```
1 __int64 __fastcall sub_1400011A0(LPVOID lpThreadParameter)
2 {
3     __int64 v1; // rdx
4     int v2; // ebx
5     __int64 v3; // rax
6     __int64 v4; // rax
7     const char *v5; // rdx
8     __int64 v6; // rax
9     __int128 v8[6]; // [rsp+20h] [rbp-78h] BYREF
10    int v9; // [rsp+80h] [rbp-18h]
11
12    v8[0] = 0i64;
13    v9 = 0;
14    v8[1] = 0i64;
15    v8[2] = 0i64;
16    v8[3] = 0i64;
17    v8[4] = 0i64;
18    v8[5] = 0i64;
19    sub_140001760(std::cout, "Give me some words > ");
20    sub_140001A00(std::cin, v1, v8);
21    if ( LODWORD(v8[0]) == 1347768643 && BYTE4(v8[0]) == 84 )
22    {
23        v2 = rand() % 10;
24        if ( v2 + rand() % 10 > 30 )
25        {
26            v3 = sub_140001760(std::cout, "Redrock");
27            std::ostream::operator<<(v3, sub_140001930);
28        }
29    }
30    else
31    {
32        v4 = -1i64;
33        do
34            ;
```

首先是一堆定义变量，

1—18 int v123459 char v5 array v8 v8初始化数组，0000000

19 打印give me some words

20 sub_140001A00(std::cin, v1, v8)

百度得「std::cin「标准输入流（cin）：

预定义的对象 cin 是 istream 类的一个实例。cin 对象附属到标准输入设备，通常是键盘」

（我到现在才发现这是c++不是c，我自裁了。）

也就是输入字符串到储存到v8（那v1是干什么的）

下一句 一个if else句

lodword我还以为是lod+word，搜了以后发现是low dword。。。。

我到目前为止还不是很理解qword，qword等等等等字符类型。也许过十分钟就知道了。

刚刚百度了， dword就是四字节的无符号短整形， q就是八字节。

If条件内容：当lodwordv8[0]=1347768643和byte4v8[0]=84都为真时，返回值为真，否则为假。

设返回值为真，则令v2为0—9中的一个随机数。若v2再加这个随机数大于30，则输出红岩，（ps：

下一句没看出是什么意思）

如果返回值为假，则无明显变化（）。

跳出里面的那个if条件句，看最外面的if，如果返回值为假，

那么令v4为-1

， ++v4开始循环（如果输入数值不符合方程，则输出nice，符合的话进入else中的v5的输出，也就是重庆没水没电没网大学），【while中的循环条件应该是一个表示true的数字也就是无限循环。】

（猜的）

最后结束程序。

代码逻辑基本分析出来了，那么该考虑如何进行重庆字符串的输出（当输入字符长度大于20）

首先字符串cqwnennu是在运算中被定义的，如果我们执行思路2，就一定不能跳过这个运算。也许我们可以先让这个运算执行，虽然输入的字符完全不符合方程，但是在程序输出nice之前jmp成输出v5这一句指令，也就是输出重庆没水大学。

也就是要把nice的指令nop掉再修改条件判断语法？

也不对，如果字符超过20应该输出的就是nice，也就是先条件判断。

如果字符小于20，执行后jmp到输出v5，如是大于20，则输出nice。

所以只需要对一些指令进行patch

然而在对程序patch之前。我们还面临一个新的问题。之前动态调试的时候smc自解密没有任何问题，但是如何将这种解密后的状态保存下来。又是一个问题。上一个level我全程没有考虑到这个问题，程序其实一直都是在running状态下的。这次要对程序打补丁。必须先要把没有加密代码的程序做出来。

这是个大问题。在查询各种奇怪的wp以及询问re师傅后，我们可以知道 (https://blog.csdn.net/qq_41923479/article/details/80377708，这篇文章帮助很大)

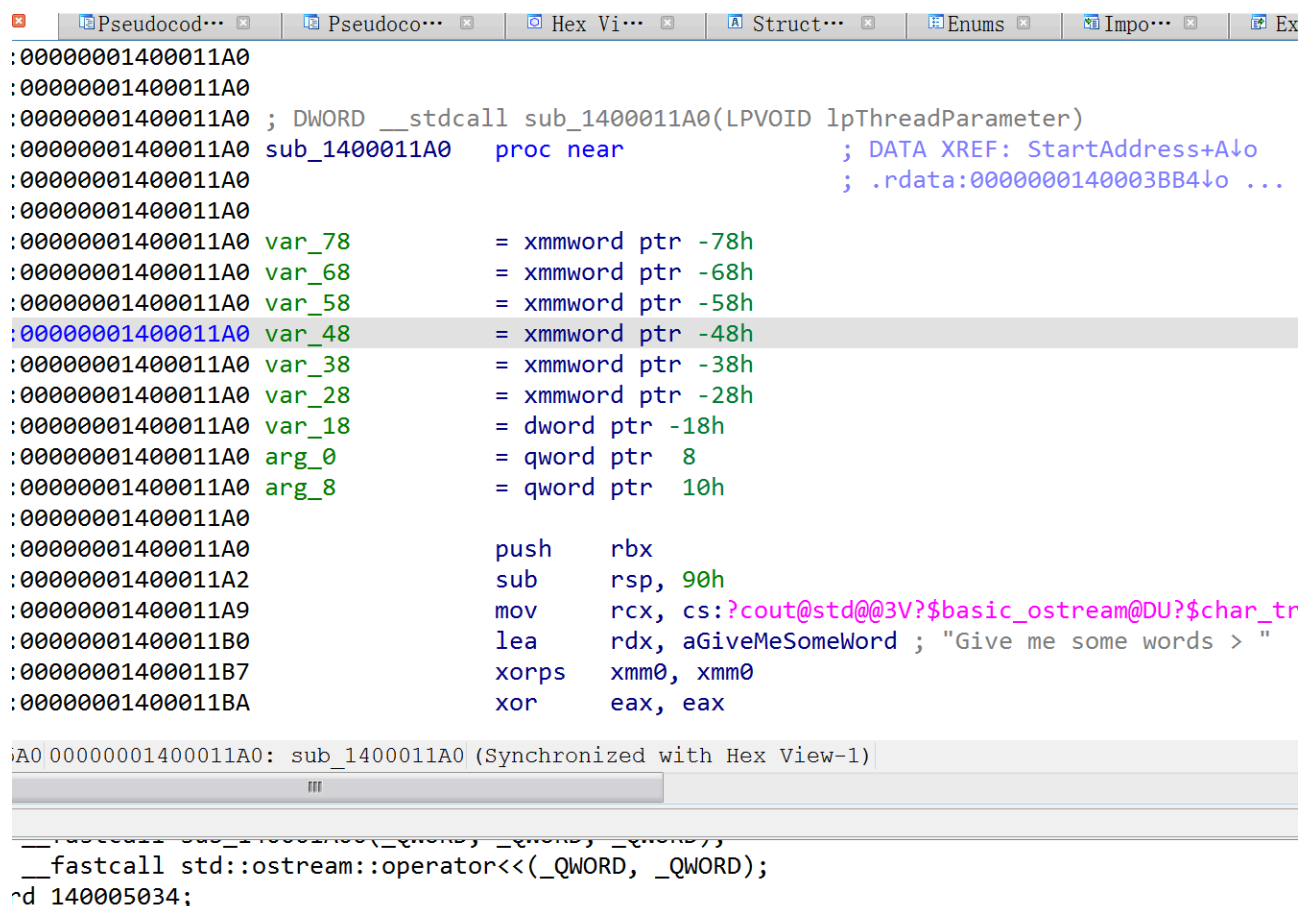
把startaddress函数中的异或改成异或0———原本是异或44，结果为0，现在结果为i本身只要运用ida脚本使用异或解密，就可以解开。

```
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)i + 1)
8      *(_BYTE *)i = |*(_BYTE *)i;
9  CreateThread(0i64, 0i64, loc_1400011A0, 0i64, 0, 0i64);
10 return 0i64;
11 }
```

首先寻找解密的起始地址，就是整个loc_1400011A0的开头结尾 00000001400011A0和
00000001400015A0
开始改脚本（）

```
1  #include <idc.idc>
2
3  static main()
4  {
5      auto addr = 0x001400011A0;    //这里填入要解密字符串的起始地址
6      auto i = 0;
7      for(i=0;addr+i<0x001400015A0;i++)    //循环结束的条件为字符串的结束地址
8      {
9          PatchByte(addr+i,Byte(addr+i)^0x44);    //异或的数字根据情况修改
10     }
11 }
```

放进ida执行。



The screenshot shows the IDA Pro interface with the assembly view of the function sub_1400011A0. The assembly code is as follows:

```
:00000001400011A0
:00000001400011A0
:00000001400011A0 ; DWORD __stdcall sub_1400011A0(LPVOID lpThreadParameter)
:00000001400011A0 sub_1400011A0 proc near ; DATA XREF: StartAddress+A↓o
:00000001400011A0 ; .rdata:0000000140003BB4↓o ...
:00000001400011A0
:00000001400011A0 var_78 = xmmword ptr -78h
:00000001400011A0 var_68 = xmmword ptr -68h
:00000001400011A0 var_58 = xmmword ptr -58h
:00000001400011A0 var_48 = xmmword ptr -48h
:00000001400011A0 var_38 = xmmword ptr -38h
:00000001400011A0 var_28 = xmmword ptr -28h
:00000001400011A0 var_18 = dword ptr -18h
:00000001400011A0 arg_0 = qword ptr 8
:00000001400011A0 arg_8 = qword ptr 10h
:00000001400011A0
:00000001400011A0 push rbx
:00000001400011A2 sub rsp, 90h
:00000001400011A9 mov rcx, cs:?.cout@std@@3V?$.basic_ostream@DU?$.char_tr
:00000001400011B0 lea rdx, aGiveMeSomeWord ; "Give me some words > "
:00000001400011B7 xorps xmm0, xmm0
:00000001400011BA xor eax, eax
```

Below the assembly view, the hex view shows the memory address 0A0 00000001400011A0, which is synchronized with the Hex View-1 window. The hex view shows the memory contents at this address, which are all zeros.

执行的时候脚本总是解不完全。有一小段无法直接转换成代码。手动调了好几次总归是把他弄成可以反编译的函数了

```
10 v6[0] = 0i64;
11 v7 = 0;
12 v6[1] = 0i64;
13 v6[2] = 0i64;
14 v6[3] = 0i64;
15 v6[4] = 0i64;
16 v6[5] = 0i64;
17 sub_140001760(std::cout, "Give me some words > ");
18 sub_140001A00(std::cin, v1, v6);
19 v2 = -1i64;
20 do
21     ++v2;
22 while ( *((_BYTE *)v6 + v2) );
23 if ( v2 == 20
24     && (870732 * SBYTE5(v6[0])
25         + 620576 * SBYTE6(v6[0])
26         + 687392 * SBYTE3(v6[0])
27         + 790701 * SBYTE4(v6[0])
28         - 264980 * SLOBYTE(v6[0])
29         - 558068 * SBYTE1(v6[0])
30         - 940616 * SBYTE7(v6[0])
31         - 805665 * SBYTE2(v6[0]) != -1990197
32         || 242625 * SBYTE5(v6[0])
000005BC sub_1400011A0:10 (1400011BC)
__int64 __fastcall sub_1400011A0(_QWORD, _QWORD, _QWORD);
__int64 __fastcall std::ostream::operator<<(_QWORD, _QWORD);
int dword_140005034;
ait...ok
```

(图这里已经把字符判断的数改成了20)

解决了第一步
，来看如何对这个文件打补丁。

我的思路还是先看伪代码，确定要jmp的地方，然后再寻找与其对应的汇编指令。

思路还是把判断是否是nice的地方强制跳转到输出重庆三无大学的地方。

「呜呜这里也好难。我还是太菜了」

先找到判断输入字符长度的指令

```
• t:00000001400012A1      mov     [rsp+98h+arg_8], rdi
• t:00000001400012A9      cmp     rax, 14h
• t:00000001400012AD      jnz     loc_14000154B
• t:00000001400012B3      movsx   edi, bYTE ptr [rsp+98h+var_78]
```

这里

比较14h (20u) 和输入字符串长度，目的是如果字符串长度大于20则输出nice 。

```
• t:00000001400012A1      mov     [rsp+98h+arg_8], rdi
• t:00000001400012A9      cmp     rax, 14h
• t:00000001400012AD      jge     loc_14000154B
• t:00000001400012B3      movsx   edi, bYTE ptr [rsp+98h+var_78]
```

把jnz改成jge，大于等于即转移到输出nice

字符串长度判定完成了。

接下来任务就是跳转输出重庆

```
00001400011F7 movsx    ecx, byte ptr [rsp+98h+var_78+4]
0000140001204 jnz      short loc_140001283
0000140001206 cmp     cl, 54h ; 'T'
0000140001209 jnz      short loc_140001283
-----
```

这里的loc_140001283对应的伪代码其实就是执行后面的解密。jnz条件运算是针对于输出redrock的条件的，如果这里直接jmp会使程序简化

```
t:000000001400011F7 cmp     dword ptr [rsp+98h+var_78], 50555143h
t:000000001400011FF movsx    ecx, byte ptr [rsp+98h+var_78+4]
t:00000000140001204 jmp      short loc_140001283
t:00000000140001206 ; -----
```

```
15 v6[4] = 0104,
16 v6[5] = 0164;
17 sub_140001760(std::cout, "Give me some words > ");
18 sub_140001A00(std::cin, v1, v6);
19 v2 = -1i64;
20 do
21     ++v2;
22 while ( *((_BYTE *)v6 + v2) );
23 if ( v2 < 20
24     && (870732 * SBYTE5(v6[0])
25         + 620576 * SBYTE6(v6[0])
26         + 687392 * SBYTE3(v6[0])
27         + 790701 * SBYTE4(v6[0])
28         - 264980 * SLOBYTE(v6[0])
29         - 558068 * SBYTE1(v6[0])
30         - 940616 * SBYTE7(v6[0])
```

jnz改jmp后的伪代码如下

到现在改了两处。一处是字符判定，一处是把和level派蒙有关的内容删去了。

剩下的我们需要使“无论输什么数，都跳转到重庆”

```
00000000140001315          imul    eax, r11d, 0D494Ch
0000000014000131C          add     ecx, eax
0000000014000131E          cmp     ecx, 0FFE1A1CBh
00000000140001324          jmp     loc_140001520
00000000140001324 ; -----
00000000140001329          db      0
0000000014000132A ; -----
```

可以直接把运算后输出jmp成重庆

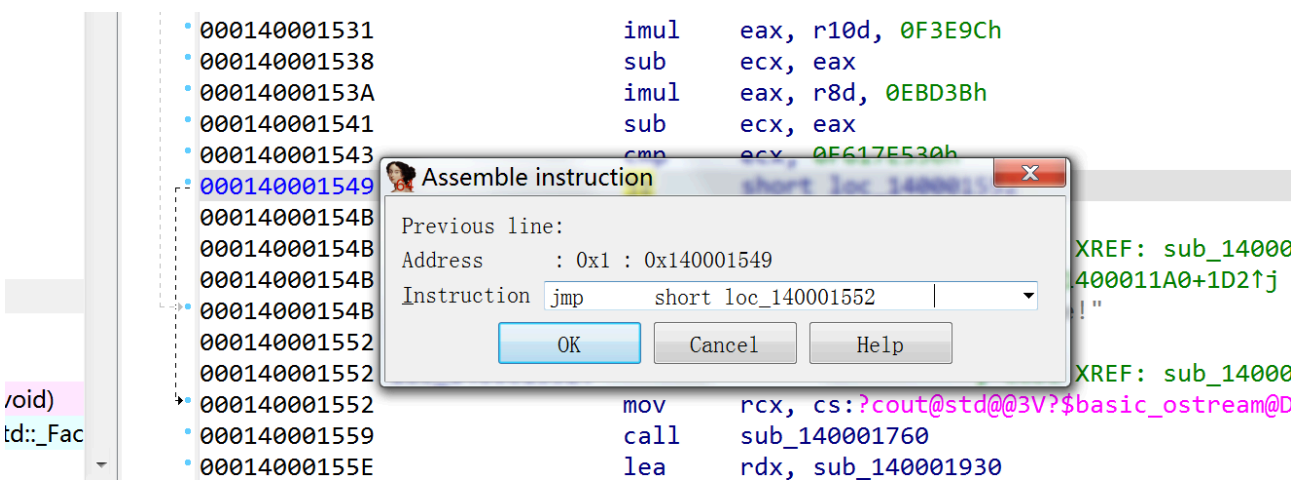
这样看看伪代码。
还是有一大段运算。怎么样把他们jmp掉是一个问题。

```
19 v2 = -1i64;  
20 do  
21 ++v2;  
22 while ( *((_BYTE *)v6 + v2) );  
23 if ( v2 >= 20  
24 || (v3 = "Chongqing No Water No Electric No Network University",  
25 802559 * SBYTE4(v6[0])  
26 + 620576 * SBYTE6(v6[0])  
27 + 687392 * SBYTE3(v6[0])  
28 + 790701 * SBYTE4(v6[0])  
29 - 264980 * SLOBYTE(v6[0])  
30 - 558068 * SBYTE1(v6[0])  
31 - 940616 * SBYTE7(v6[0])  
32 - 805665 * SBYTE2(v6[0])  
33 - 999068 * SBYTE2(v6[0])  
34 - 965947 * SBYTE6(v6[0]) != -166206160) )  
35 {  
36 v3 = "Nice!";  
37 }  
38 v4 = sub_140001760(std::cout, v3);  
39 std::ostream::operator<<(v4, sub_140001930);  
40 dword_140005034 = 0;  
41 return 0i64;
```

还是要找到对应的汇编指令

```
00151A          imul     eax, edx, 0ACE00h  
001520  
001520 loc_140001520:          ; CODE XREF: sub_1400011A0+184↑j  
001520          ; sub_1400011A0+358↑j  
001520          lea      rdx, aChongqingNowat ; "Chongqing No Water No Electric No  
001527          sub      ecx, eax  
001529          imul     eax, ebx, 0C3EFFh  
00152F          add      ecx, eax  
001531          imul     eax, r10d, 0F3E9Ch  
001538          sub      ecx, eax  
00153A          imul     eax, r8d, 0EBD3Bh  
001541          sub      ecx, eax  
001543          cmp      ecx, 0F617E530h  
001549          jz       short loc_140001552  
00154B
```

如图。最后一个指令，如果运算未满足条件就不跳转，为了把这一段运算jmp掉，把jz改成jmp



The screenshot shows a debugger window with assembly code. The code is as follows:

```
000140001531      imul     eax, r10d, 0F3E9Ch
000140001538      sub      ecx, eax
00014000153A      imul     eax, r8d, 0EBD3Bh
000140001541      sub      ecx, eax
000140001543      cmp      ecx, 0F617F530h
000140001549      jmp      short loc_140001552
00014000154B
00014000154B
00014000154B
000140001552      mov      rcx, cs:?.cout@std@@3V?$basic_ostream@D
000140001559      call     sub_140001760
00014000155E      lea      rdx, sub_140001930
```

An "Assemble instruction" dialog box is open, showing the previous line's address as 0x1 : 0x140001549 and the instruction as `jmp short loc_140001552`. The dialog box has "OK", "Cancel", and "Help" buttons.

On the right side of the assembly code, there are cross-references (XREF) listed:

- XREF: sub_1400011A0+1D2↑j
- XREF: sub_1400011A0+1D2↑j
- XREF: sub_1400011A0+1D2↑j

看看现在的代码

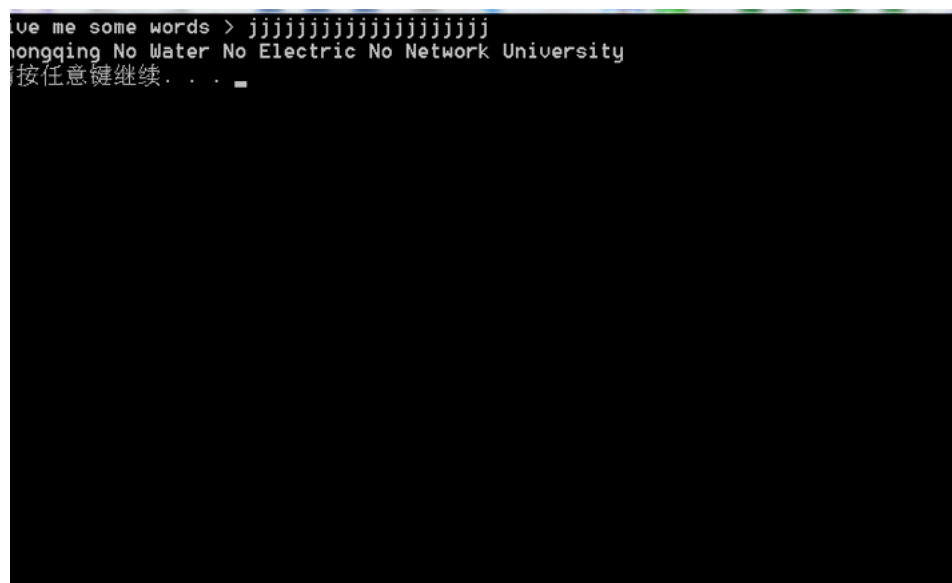
感觉挺靠谱。
但是实际测试发现输出还是nice

```
11 v7 = 0;
12 v6[1] = 0i64;
13 v6[2] = 0i64;
14 v6[3] = 0i64;
15 v6[4] = 0i64;
16 v6[5] = 0i64;
17 sub_140001760(std::cout, "Give me some words > ");
18 sub_140001A00(std::cin, v1, v6);
19 v2 = -1i64;
20 do
21     ++v2;
22 while ( *((_BYTE *)v6 + v2) );
23 if ( v2 >= 20 )
24     v3 = "Nice!";
25 else
26     v3 = "Chongqing No Water No Electric No Network University";
27 v4 = sub_140001760(std::cout, v3);
28 std::ostream::operator<<(v4, sub_140001930);
29 dword_140005034 = 0;
30 return 0i64;
31 }
```

```
xl:000000001400012A1      mov     [rsp+98h+arg_8], rdi
xt:000000001400012A9      cmp     rax, 14h
xt:000000001400012AD      jmp     loc_14000154B
vt:000000001400012AD
```

心急如焚地研究了一下啥也没发现，然后开始魔改程序。先是把字符判断给jmp了。结果重庆都没了
然后改成jz，字符长度等于20.试试看倒是成功了。见鬼了，为什么小于就失败》

Jle输入长于20个字符也输出nice。



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

External sy
St

ktop\1_repair.
ble for AMD64
n 27 12:13:29
001000)
: 00001AB3 (
: 00001C00 (
: 00000400
e Readable

.text:0000000140001000 ; Alignment : default
.text:0000000140001000 ; OS type : MS Windows
.text:0000000140001000 ; Application type: Executable
.text:0000000140001000 ; =====
00000400 0000000140001000: .text:0000000140001000 (Synchronized with RIP)

Hex View-1
00000140001280 02 00 00 48 8D 54 24 20 48 C7 C0 FF FF FF FF 90 ...HE
00000140001290 48 FF C0 80 3C 02 00 75 F7 48 89 B4 24 A0 00 00 H.纭.
000001400012A0 00 48 89 BC 24 A8 00 00 00 48 83 F8 14 0F 8D 98 .H壩.

好家伙，昨天晚上试怎么都不成功，今天下午一试发现是patch没有完全保存，jge的地方还是jnz，还有一个本来要改成jmp的地方也变成了jnz，然后重新改回来了，直接测试，成功。好家伙