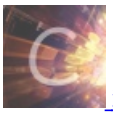
复盘网鼎杯Re-Signal Writeup

原创

Chauncy. 于 2020-05-17 11:41:53 发布 322 收藏
分类专栏： 程序逆向之美 文章标签： 安全

本文链接：https://blog.csdn.net/shipsail/article/details/106164969

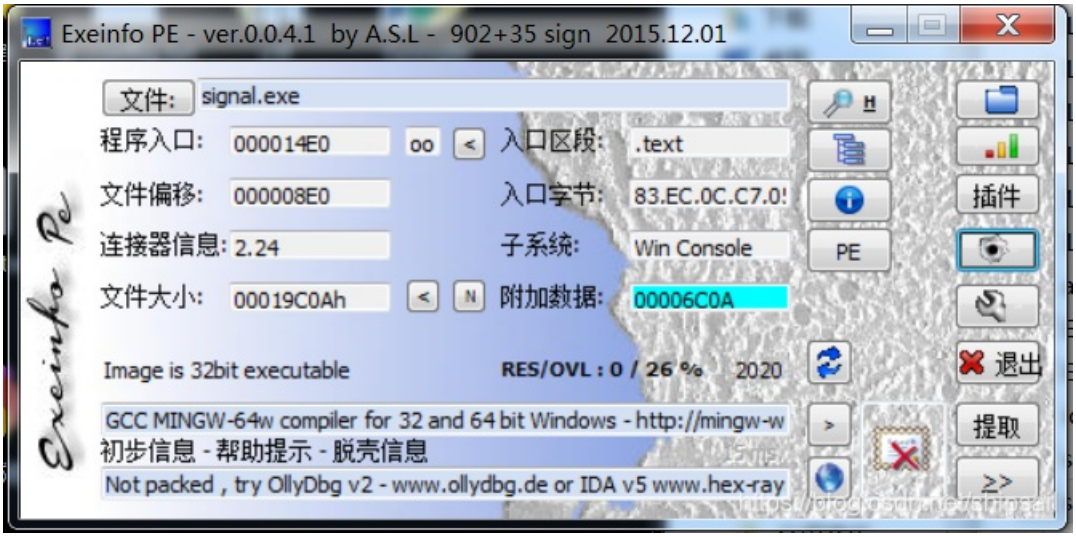


Signal

0x0 引言

比赛时没能做出来这道题，当时我使用OD进行动态调试，分析出启动程序时初始化了0x72数组，并且含有一个12分支的switch被嵌套在循环中，最终没能搞定，今天就复盘这道题。

0x1 查壳



无壳，并且编译时未启用ASLR技术，挺友好哈。

0x2 分析 main

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    int v4; // [esp+18h] [ebp-1D4h]
4
5    __main();
6    qmemcpy(&v4, &unk_403040, 0x1C8u);
7    vm_operad(&v4, 114);
8    puts("good,The answer format is:flag {}");
9    return 0;
10 }
```


这次学聪明了，先打开反汇编辅助插件查看伪代码（这对理解汇编逻辑有极大的帮助）。

可以看到在 `main` 函数中先调用了 `__main()`
随后复制内存到变量v4，随后v4被传入 `vm_operad()`
容易想到，&unk_403040 是通过 `__main()` 得到的，随后的 `vm_operad()` 则是在计算flag，并在puts()中输出。

0x3 瞧一瞧 __main()

GO TO：0x0040176F
通过OD动态调试，F8步过发现寄存器中没有任何的变化，所以我认为之后的操作与__main()函数是无关的。

0x4 内存复制

```
.text:00401760                push    ebp
.text:00401761                mov     ebp, esp
.text:00401763                push    edi
.text:00401764                push    esi
.text:00401765                push    ebx
.text:00401766                and     esp, 0FFFFFFF0h
.text:00401769                sub     esp, 1E0h
.text:0040176F                call    ___main
.text:00401774                lea     eax, [esp+18h]
.text:00401778                mov     ebx, offset unk_403040
.text:0040177D                mov     edx, 72h ; 'r'
.text:00401782                mov     edi, eax
.text:00401784                mov     esi, ebx
.text:00401786                mov     ecx, edx
.text:00401788                rep movsd
```


`rep movsd` 根据百科上的定义，赋值edx次，即0x72次
其中esi指向的地址为0x403040，如下

```
00403030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00403040  0A 00 00 00 04 00 00 00  10 00 00 00 08 00 00 00  ................
00403050  03 00 00 00 05 00 00 00  01 00 00 00 04 00 00 00  ................
00403060  20 00 00 00 08 00 00 00  05 00 00 00 03 00 00 00  ·...............
00403070  01 00 00 00 03 00 00 00  02 00 00 00 08 00 00 00  ................
00403080  0B 00 00 00 01 00 00 00  0C 00 00 00 08 00 00 00  ................
00403090  04 00 00 00 04 00 00 00  01 00 00 00 05 00 00 00  ................
004030A0  03 00 00 00 08 00 00 00  03 00 00 00 21 00 00 00  ............!...
004030B0  01 00 00 00 0B 00 00 00  08 00 00 00 0B 00 00 00  ................
004030C0  01 00 00 00 04 00 00 00  09 00 00 00 08 00 00 00  ................
004030D0  03 00 00 00 20 00 00 00  01 00 00 00 02 00 00 00  ····.···........
004030E0  51 00 00 00 08 00 00 00  04 00 00 00 24 00 00 00  Q...........$...
004030F0  01 00 00 00 0C 00 00 00  08 00 00 00 0B 00 00 00  ................
00403100  01 00 00 00 05 00 00 00  02 00 00 00 08 00 00 00  ................
00403110  02 00 00 00 25 00 00 00  01 00 00 00 02 00 00 00  ····%...........
00403120  36 00 00 00 08 00 00 00  04 00 00 00 41 00 00 00  6...........A...
```

再来看反汇编代码。

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    int v4; // [esp+18h] [ebp-1D4h]
4
5    __main();
6    qmemcpy(&v4, &unk_403040, 0x1C8u);
7    vm_operad(&v4, 114);
8    puts("good,The answer format is:flag {}");
9    return 0;
10 }
```


内存复制，从0x403040复制到变量v4，大小为0x1C8个字节。

可能存在疑问，在汇编代码中是0x72，为啥反汇编后是0x1C8呢？

其实很简单，因为数据类型为`DWORD`占4个字节，这从上文中数据硬编码处可以看出。
`0x1C8 = 0x72 * 0x4`
还有一个很奇怪的现象，`puts`中只传入了字符串模板，而没有传入存储flag的地址...

0x5 分析 vm_operad

使用IDA自动反汇编`vm_operad(int *a1, int a2)`
int * a1传入的是之前的v4首地址，a2为常数114
经过部分的调整与修改我得到了源代码如下：

```
#include<iostream>
#include<windows.h>
#include<string>
using namespace std;


size_t  read(char *a1)
{
  size_t result; // eax

  printf("string:");
  scanf("%s", a1);
  result = strlen(a1);
  if ( result != 15 )
  {
    puts("WRONG!\n");
    exit(0);
  }
  return result;
}

int vm_operad(unsigned int *a1, int a2)
{
  int result; // eax
  char v3[100]; // [esp+13h] [ebp-E5h]
  char v4[100]; // [esp+77h] [ebp-81h]
  char v5; // [esp+DBh] [ebp-1Dh]
```

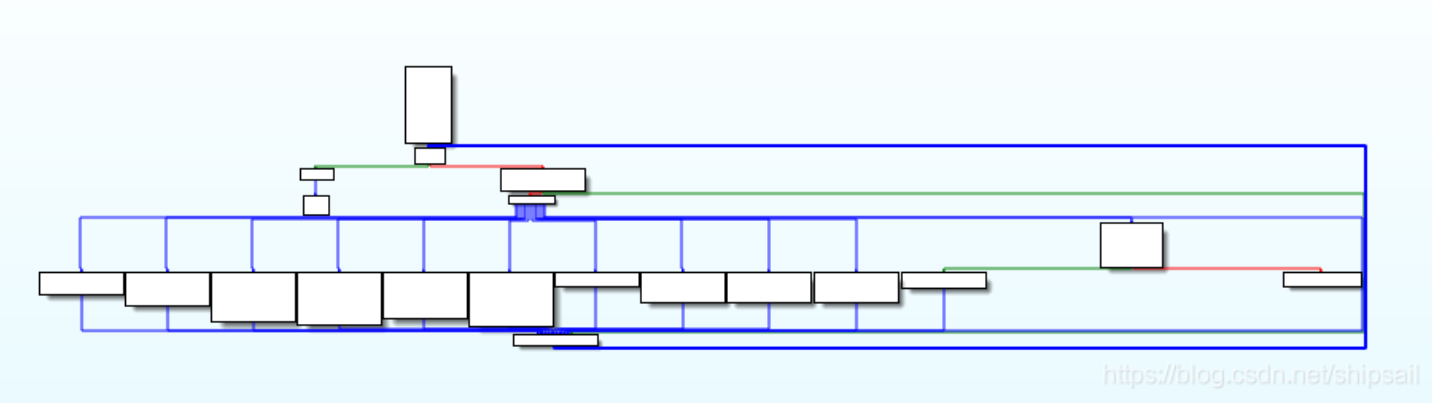
```
int v6; // [esp+DCh] [ebp-1Ch]
int v7; // [esp+E0h] [ebp-18h]
int v8; // [esp+E4h] [ebp-14h]
int v9; // [esp+E8h] [ebp-10h]
int v10; // [esp+ECh] [ebp-Ch]

v10 = 0;
v9 = 0;
v8 = 0;
v7 = 0;
v6 = 0;
while ( 1 )
{
  result = v10;
  if ( v10 >= a2 )
    return result;
  switch ( a1[v10] )
  {
    case 0:
    case 9:
      continue;
    case 1:
      v4[v7] = v5;
      ++v10;
      ++v7;
      ++v9;
      break;
    case 2:
      v5 = a1[v10 + 1] + v3[v9];
      v10 += 2;
      break;
    case 3:
      v5 = v3[v9] - LOBYTE(a1[v10 + 1]);
      v10 += 2;
      break;
    case 4:
      v5 = a1[v10 + 1] ^ v3[v9];
      v10 += 2;
      break;
    case 5:
      v5 = a1[v10 + 1] * v3[v9];
      v10 += 2;
      break;
    case 6:
      ++v10;
      break;
    case 7:
      if ( v4[v8] != a1[v10 + 1] )
      {
        printf("what a shame...");
        exit(0);
      }
      ++v8;
      v10 += 2;
      break;
    case 8:
      v3[v6] = v5;
      ++v10;
      ++v6;
      break;
```

```
      break;
    case 10:
      read(v3);
      ++v10;
      break;
    case 11:
      v5 = v3[v9] - 1;
      ++v10;
      break;
    case 12:
      v5 = v3[v9] + 1;
      ++v10;
      break;
  }
 }
}
```

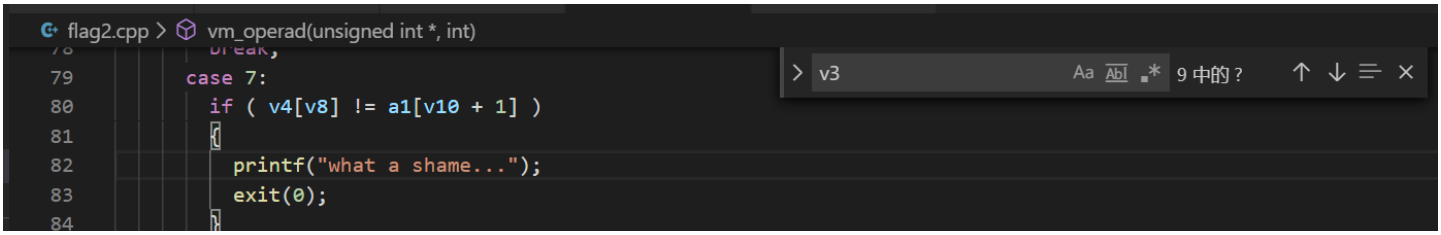
```
int main(){
    unsigned int a[] ={0x0A,0x04,0x10,0x08,0x03,0x05,0x01,0x04,0x20,0x08,0x05,0x03,0x01,0x03,0x02,0x08,0x0B,0x01
,0x0C,0x08,0x04,0x04,0x01,0x05,0x03,0x08,0x03,0x21,0x01,0x0B,0x08,0x0B,0x01,0x04,0x09,0x08,0x03,0x20,0x01,0x02,0
x51,0x08,0x04,0x24,0x01,0x0C,0x08,0x0B,0x01,0x05,0x02,0x08,0x02,0x25,0x01,0x02,0x36,0x08,0x04,0x41,0x01,0x02,0x2
0,0x08,0x05,0x01,0x01,0x05,0x03,0x08,0x02,0x25,0x01,0x04,0x09,0x08,0x03,0x20,0x01,0x02,0x41,0x08,0x0C,0x01,0x07,
0x22,0x07,0x3F,0x07,0x34,0x07,0x32,0x07,0x72,0x07,0x33,0x7,0x18,0x7,0xffffffa7,0x7,0x31,0x7,0xffffffff1,0x7,0x28,
0x7,0xffffff84,0x7,0xffffffc1,0x7,0x1e,0x7,0x7a};
    cout << sizeof(a) / sizeof(a[0]);
    vm_operad(&a,114);
    return 0;
}
```

分析到这里有点懵逼，因为不知道啥是我们要的flag，我们只是拿到了源代码，而且这代码这么多的分支，感觉和我当时用OD调试没啥大的区别。



通过可视化，可以知道，如果想反向求解存在一定的困难，因为swich分支太多啦！

通过已得到的反汇编代码，`v3` `v4` 为长度100的字符数组，由于`最后的输出中根本没有flag`，所以猜测v3 v4可能就是我们想要的flag。



```
85          ++v8;
86          v10 += 2;
87          break;
88        case 8:
89          v3[v6] = v5;
90          ++v10;
91          ++v6;
92          break;
93        case 10:
94          read(v3);
95          ++v10;
96          break;
97        case 11:
98          v5 = v3[v9] - 1;
99          ++v10;
100         break;
101       case 12:
102         v5 = v3[v9] + 1;
103         ++v10;
```



```
size_t  read(char *a1)
{
  size_t result; // eax

  printf("string:");
  scanf("%s", a1);
  result = strlen(a1);
  if ( result != 15 )
  {
    puts("WRONG!\n");
    exit(0);
  }
  return result;
}
```


`v3` 变量在 `case8` 中被改变，在`case10`中被输入，输入长度为15

```
49      switch ( a1[v10] )
50      {
51        case 0:
52        case 9:
53          continue;
54        case 1:
55          v4[v7] = v5;
56          ++v10;
57          ++v7;
58          ++v9;
59          break;
60        case 2:
```

v4 Aa Abl .* 5 中的 4


```
77          ++v10;
78          break;
79        case 7:
80          if ( v4[v8] != a1[v10 + 1] )
81          {
82            printf("what a shame...");
```

```
83          exit(0);
84        }
85        ++v8;
86        v10 += 2;
```


`v4` 在 `case 1` 中被改变， `case7` 中将他与 `a1` 元素进行比较， `a1` 中的元素是已知的，为了方便，我把 `a1` 改名为 `array` 。
这里我将printf() 和 exit() 注释，并在read()中随机输入15个字符

```
98        case 7:
99          // 输出验证信息
100         cout <<"case 7 " << int(v4[v8]) <<" " << int(array[v10 + 1]) <<endl;
101         if ( v4[v8] != array[v10 + 1] )
102         {
103           // printf("what a shame...");
104           // exit(0);
105         }
106         ++v8;
107         v10 += 2;
108         break;
```


```
7   size_t  read(char *array)
8   {
9     // string s = "123123123123123";
10    // s = "123123123123123";
11    for(int i=0;i<15;i++){
12      array[i] = rand() * 255;
13    }
14    return 15;
```


多次执行后发现规律。

```
问题  输出  终端  调试控制台                                                    1: 1

PS E:\Astdchi\contests\CTF\re> g++ .\flag2.cpp -o 1.exe
PS E:\Astdchi\contests\CTF\re> .\1.exe
case[7] v4 0            -62             22
case[7] v4 1            -9              3f
case[7] v4 2            63              34
case[7] v4 3            121             32
case[7] v4 4            60              72
case[7] v4 5            -110            33
case[7] v4 6            3               18
case[7] v4 7            -121            ffffffa7
case[7] v4 8            -82             31
case[7] v4 9            5               ffffff
case[7] v4 10           -84             28
case[7] v4 11           47              ffffff84
case[7] v4 12           82              ffffffc1
case[7] v4 13           44              1e
case[7] v4 14           89              7a
v3: -57 -3 64 125 93 -109 35 -93 -81 -32 -19 47 45 76 88
v4: -62 -9 63 121 60 -110 3 -121 -82 5 -84 47 82 44 89
89 15 15 15 15 114
try again:
1
case[7] v4:0            0               22
case[7] v4:1            71              3f
case[7] v4:2            87              34
case[7] v4:3            34              32
case[7] v4:4            43              72
case[7] v4:5            119             33
case[7] v4:6            -35             18
```

```
case[7] v4:7             55              ffffffa7
case[7] v4:8             103             31
case[7] v4:9             -35             ffffff
case[7] v4:10            -103            28
case[7] v4:11            19              ffffff84
case[7] v4:12            -47             ffffffc1
case[7] v4:13            -45             1e
case[7] v4:14            -117            7a
```


`case7` 中输出了 `v4` 每个元素的验证值 `array[v10 + 1]` 是常数，这就相当于我们已知了 `v4` 数组，而 `v4` 是在 `case 1` 中赋值为 `v5`，其中 `0xffffff` 不知道为啥显示不全，在OD中他的值为 `0xffffffff`

```
v4 = {0x22,0x3f,0x34,0x32,0x72,0x33,0x18,0xffffffa7,0x31,0xf1,0x28,0xffffff84,0xffffffc1,0x1e,0x7a};
```

现在就得着重研究 `V5`

0x6 调试计算 V5

```
#include<iostream>
#include<windows.h>
#include<string>
using namespace std;


size_t  read(char *array)
{
  for(int i=0;i<15;i++){
    array[i] = rand() * 255;
  }
  return 15;

  size_t result; // eax
  printf("string:");
  scanf("%s", array);
  result = strlen(array);
  if ( result != 15 )
  {
    puts("WRONG!\n");
    exit(0);
  }
  return result;
}

void displayV9V10(int _case ,int v9,int v10){
  return;
  printf("case %d\tv9=%d\tv10=%d\n",_case,v9,v10);
}

int vm_operad(unsigned int *array, int a2)
{
  int result; // eax
  char v3[100]; // [esp+13h] [ebp-E5h]
  char v4[100]; // [esp+77h] [ebp-81h]
  char v5; // [esp+DBh] [ebp-1Dh]
  int v6; // [esp+DCh] [ebp-1Ch]
  int v7; // [esp+E0h] [ebp-18h]
  int v8; // [esp+E4h] [ebp-14h]
  int v9; // [esp+E8h] [ebp-10h]
  int v10; // [esp+ECh] [ebp-Ch]

  v10 = 0;
```

```
v10 = 0;
v9 = 0;
v8 = 0;
v7 = 0;
v6 = 0;
while ( 1 )
{

  result = v10;
  if ( v10 >= a2 ){
    // cout << "v3: ";
    // for(int i = 0; v3[i] != '\0';i++){
    //     cout << int(v3[i]) << " ";
    // }
    // cout << endl;
    // cout << "v4: ";
    // for(int i = 0; v4[i] != '\0';i++){
    //     cout << int(v4[i]) << " ";
    // }
    // cout << endl;
    // cout << int(v5) << " "<<v6<<" "<<v7<<" "<<v8<<" "<<v9<<" "<<v10<<endl;
    return result;
  }


  switch ( array[v10] )
  {
    case 0:
    case 9:
      continue;
    case 1:
      displayV9V10(1,v9,v10);
      printf("v4[%d] = v5; \n\n\n",v7);
      v4[v7] = v5;
      ++v10;
      ++v7;
      ++v9;
      break;
    case 2:
      displayV9V10(2,v9,v10);
      printf("v5 = array[%d] + v3[%d]; \n",v10 + 1,v9);
      v5 = array[v10 + 1] + v3[v9];
      v10 += 2;
      break;
    case 3:
      displayV9V10(3,v9,v10);
      printf("v5 = v3[%d] - LOBYTE(array[%d]); \n",v9 ,v10 + 1);
      v5 = v3[v9] - LOBYTE(array[v10 + 1]);
      v10 += 2;
      break;
    case 4:
      displayV9V10(4,v9,v10);
      printf("v5 = array[%d] ^ v3[%d]; \n",v10 + 1,v9);
      v5 = array[v10 + 1] ^ v3[v9];
      v10 += 2;
      break;
    case 5:
      displayV9V10(5,v9,v10);
      printf("v5 = array[%d] * v3[%d]; \n",v10 + 1,v9);
      v5 = array[v10 + 1] * v3[v9];
```

```
        v10 += 2;
        break;
      case 6:
        ++v10;
        break;
      case 7:
        // 输出验证信息
        cout << "case[7] " << "v4:"  << v8  << "\t\t" << int(v4[v8]) << "\t\t" <<  int(array[v10 + 1]) <<endl <<
dec;
        if ( v4[v8] != array[v10 + 1] )
        {
          // printf("what a shame...");
          // exit(0);
        }
        ++v8;
        v10 += 2;
        break;
      case 8:
        displayV9V10(8,v9,v10);
        printf("v3[%d] = v5; \n",v6);
        v3[v6] = v5;
        ++v10;
        ++v6;
        break;
      case 10:
        read(v3);
        ++v10;
        break;
      case 11:
        displayV9V10(11,v9,v10);
        printf("v5 = v3[%d] - 1; \n",v9);
        v5 = v3[v9] - 1;
        ++v10;
        break;
      case 12:
        displayV9V10(12,v9,v10);
        printf("v5 = v3[%d] + 1; \n",v9);
        v5 = v3[v9] + 1;
        ++v10;
        break;
    }
  }
}



int main(){
    unsigned int a[] ={0x0A,0x04,0x10,0x08,0x03,0x05,0x01,0x04,0x20,0x08,0x05,0x03,0x01,0x03,0x02,0x08,0x0B,0x01
,0x0C,0x08,0x04,0x04,0x01,0x05,0x03,0x08,0x03,0x21,0x01,0x0B,0x08,0x0B,0x01,0x04,0x09,0x08,0x03,0x20,0x01,0x02,0
x51,0x08,0x04,0x24,0x01,0x0C,0x08,0x0B,0x01,0x05,0x02,0x08,0x02,0x25,0x01,0x02,0x36,0x08,0x04,0x41,0x01,0x02,0x2
0,0x08,0x05,0x01,0x01,0x05,0x03,0x08,0x02,0x25,0x01,0x04,0x09,0x08,0x03,0x20,0x01,0x02,0x41,0x08,0x0C,0x01,0x07,
0x22,0x07,0x3F,0x07,0x34,0x07,0x32,0x07,0x72,0x07,0x33,0x7,0x18,0x7,0xfffffa7,0x7,0x31,0x7,0xffffff,0x7,0x28,0x
7,0xffffff84,0x7,0xfffffc1,0x7,0x1e,0x7,0x7a};
    int test;
    while(1){
      unsigned int *b = new unsigned int[114];
      for(int i = 0; i < 114;i++)
        b[i] = a[i];
      vm_operad(b,114);
      delete [] b;
```

```
        delete [] b;
        cout <<"input a int to try again:"<<endl;
        cin >> test;
    }
    return 0;
}
```

打印运算`V5`的`case`和`case 1`

由于`V4`已知，所以可以求出对应的`V3`，以下为打印获取计算过程

```
v5 = array[2] ^ v3[0];
v3[0] = v5;
v5 = v3[0] - LOBYTE(array[5]);
v4[0] = v5;


v5 = array[8] ^ v3[1];
v3[1] = v5;
v5 = array[11] * v3[1];
v4[1] = v5;


v5 = v3[2] - LOBYTE(array[14]);
v3[2] = v5;
v5 = v3[2] - 1;
v4[2] = v5;


v5 = v3[3] + 1;
v3[3] = v5;
v5 = array[21] ^ v3[3];
v4[3] = v5;


v5 = array[24] * v3[4];
v3[4] = v5;
v5 = v3[4] - LOBYTE(array[27]);
v4[4] = v5;


v5 = v3[5] - 1;
v3[5] = v5;
v5 = v3[5] - 1;
v4[5] = v5;


v5 = array[34] ^ v3[6];
v3[6] = v5;
v5 = v3[6] - LOBYTE(array[37]);
v4[6] = v5;


v5 = array[40] + v3[7];
v3[7] = v5;
v5 = array[43] ^ v3[7];
v4[7] = v5;


v5 = v3[8] + 1;
```

```
v3[8] = v5;
v5 = v3[8] - 1;
v4[8] = v5;


v5 = array[50] * v3[9];
v3[9] = v5;
v5 = array[53] + v3[9];
v4[9] = v5;


v5 = array[56] + v3[10];
v3[10] = v5;
v5 = array[59] ^ v3[10];
v4[10] = v5;


v5 = array[62] + v3[11];
v3[11] = v5;
v5 = array[65] * v3[11];
v4[11] = v5;


v5 = array[68] * v3[12];
v3[12] = v5;
v5 = array[71] + v3[12];
v4[12] = v5;


v5 = array[74] ^ v3[13];
v3[13] = v5;
v5 = v3[13] - LOBYTE(array[77]);
v4[13] = v5;


v5 = array[80] + v3[14];
v3[14] = v5;
v5 = v3[14] + 1;
v4[14] = v5;
```

根据上面每一块内容写脚本计算`V3`

```cpp
#include<iostream>
#include<windows.h>
#include<string>
using namespace std;
int main(){
    unsigned long array[] ={0x0A,0x04,0x10,0x08,0x03,0x05,0x01,0x04,0x20,0x08,0x05,0x03,0x01,0x03,0x02,0x08,0x0B
,0x01,0x0C,0x08,0x04,0x04,0x01,0x05,0x03,0x08,0x03,0x21,0x01,0x0B,0x08,0x0B,0x01,0x04,0x09,0x08,0x03,0x20,0x01,0
x02,0x51,0x08,0x04,0x24,0x01,0x0C,0x08,0x0B,0x01,0x05,0x02,0x08,0x02,0x25,0x01,0x02,0x36,0x08,0x04,0x41,0x01,0x0
2,0x20,0x08,0x05,0x01,0x01,0x05,0x03,0x08,0x02,0x25,0x01,0x04,0x09,0x08,0x03,0x20,0x01,0x02,0x41,0x08,0x0C,0x01,
0x07,0x22,0x07,0x3F,0x07,0x34,0x07,0x32,0x07,0x72,0x07,0x33,0x7,0x18,0x7,0xffffffa7,0x7,0x31,0x7,0xffffffff1,0x7,
0x28,0x7,0xffffff84,0x7,0xffffffc1,0x7,0x1e,0x7,0x7a};
    unsigned long v4[] = {0x22,0x3f,0x34,0x32,0x72,0x33,0x18,0xffffffa7,0x31,0xf1,0x28,0xffffff84,0xffffffc1,0x1
e,0x7a};

    unsigned long v3[15];

    v3[0] = (v4[0] + LOBYTE(array[5])) ^ array[2];
    v3[1] = (v4[1] / array[11]) ^ array[8];
    v3[2] = (v4[2] + 1) + LOBYTE(array[14]);
    v3[3] = (v4[3] ^ array[21]) - 1;
    v3[4] = (v4[4] + LOBYTE(array[27])) / array[24];
    v3[5] = (v4[5] + 1) + 1;
    v3[6] = (v4[6] + LOBYTE(array[37])) ^ array[34];
    v3[7] = (v4[7] ^ array[43]) - array[40];
    v3[8] = (v4[8] + 1) - 1;
    v3[9] = (v4[9] - array[53]) / array[50];
    v3[10] = (v4[10] ^ array[59]) - array[56];
    v3[11] = (v4[11] / array[65])  - array[62];
    v3[12] = (v4[12] - array[71]) / array[68];
    v3[13] = (v4[13] +  LOBYTE(array[77])) ^ array[74];
    v3[14] = (v4[14] - 1) - array[80];

    cout << "flag{";
    for(int i=0;i<15;i++){
        cout <<char(v3[i]);
    }
    cout<<"}";
    return 0;
}
```

```
PS E:\Astdchi\contests\CTF\re> g++ .\get.cpp -o 2.exe
PS E:\Astdchi\contests\CTF\re> .\2.exe
flag{757515121f3d478}
```

得到 Flag

0x7 总结

这道题不像往常刷的题那样去破解算法本身，由于他分支复杂，所以应该有内在的规律。