

2021SCUCTF新生赛-逆向题WriteUp

时间：2021/11/19-2021/11/23

2021SCUCTF新生赛-逆向题WriteUp

[RE1>Welcome

[RE2]Xor

[RE3]DebugMe

[RE4]upxed

[RE5]jezpak

[RE6]EzLinearEquation

[RE7]VirtualMachine

解法一：分析虚拟机

解法二：angr梭哈

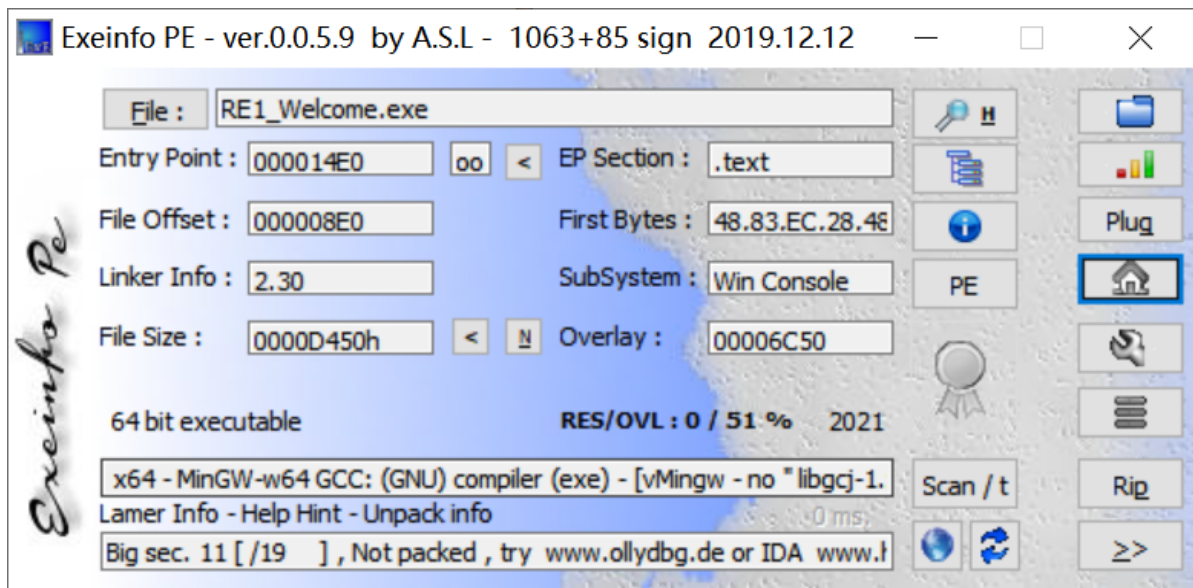
[RE8]PyCode

[RE9]Plants_Vs_Zombies

[HappyRE]ezjava

[RE1>Welcome

将文件拖入 Exeinfo PE 检查文件格式和加壳情况。64位PE文件，没有加壳：



拖入IDA后查看伪代码，分析程序逻辑，可以得到，只需要输入1919810即可输出正确的flag：

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     int v4[3]; // [rsp+2Ch] [rbp-Ch] BYREF
4
5     _main(argc, argv, envp);
6     puts("Input a correct number to rescue the Fuhrer.");
7     if ( scanf("%d", v4) != 1 )
8     {
9         puts("??");
10        exit(0);
11    }
12    if ( v4[0] != 1919810 )
13    {
14        puts("Wrong!");
15        exit(0);
16    }
17    atexit(getflag);
18    return 0;
19 }

```

```

IDA View-A
Pseudocode-B
Pseudoc

1 void __fastcall getflag()
2 {
3     puts("Correct.Flag to enter Fuhrerbunker:");
4     puts(flag);
5 }

```

```

D:\CTF\SCUCTF-XSS-2021\Review\Release\1>RE1_Welcome.exe
Input a correct number to rescue the Fuhrer.
1919810
Correct.Flag to enter Fuhrerbunker:
scuctf{w3lcm3_to_the_world_of_r3!}

```

[RE2]Xor

直接定位关键代码。中间有一段很奇怪的代码我们可能不用去仔细分析，而是通过关键代码猜测其作用。第一个if判断一个很复杂的式子是不是等于26，结合下面for循环的次数也是26可以确定第一个if是判断输入的长度是否为26。第二个if指的是将输入和key异或之后与密文enc比较，比对通过则说明输入为正确的flag：

```

8
9  _main(argc, argv, envp);
10 puts("Input your flag, and i will check it:");
11 scanf("%40s", v8);
12 v8[39] = 0;
13 v3 = v8;
14 do
15 {
16     v4 = *(_DWORD *)v3;
17     v3 += 4;
18     v5 = ~v4 & (v4 - 16843009) & 0x80808080;
19 }
20 while ( !v5 );
21 if ( (~v4 & (v4 - 16843009) & 0x8080) == 0 )
22     v5 >>= 16;
23 if ( (~v4 & (v4 - 16843009) & 0x8080) == 0 )
24     v5 -= 2;
25 if ( &v3[-__CFADD__((_BYTE)v5, (_BYTE)v5) - 3] - v8 != 26 )
26 {
27     puts("Nope.");
28     exit(1);
29 }
30 for ( i = 0i64; i != 26; ++i )
31 {
32     if ( (key[i] ^ (unsigned __int8)v8[i]) != enc[i] )
33     {
34         puts("Nope.");
35         exit(0);
36     }
37 }
38 puts("Correct! Input is your flag.");
39 return 0;

```

将 enc 和 key dump 下来丢进 [CyberChef](#) 里异或。当然这一步也可以通过写 C/Python 脚本完成：

Recipe	Input
<p>From Hex</p> <p>Delimiter: Auto</p> <p>XOR</p> <p>Key: 77356951938B45E44C079B1E3C8B7BE5DEE051F02C... HEX ▾</p> <p>Scheme: Standard <input type="checkbox"/> Null preserving</p>	<p>04561C32E7ED3E812D74E24144E409B8BF8C369F5EC140A0D6030000000000000</p>
	<p>Output</p> <p>scuctf{easy_xor_algorithm}.....</p>

[RE3]DebugMe

这题输入一个正确的数字 v22 即可输出 flag，但是 v22 的生成过程非常复杂，手动计算很困难，所以这题我们可以考虑动态调试：

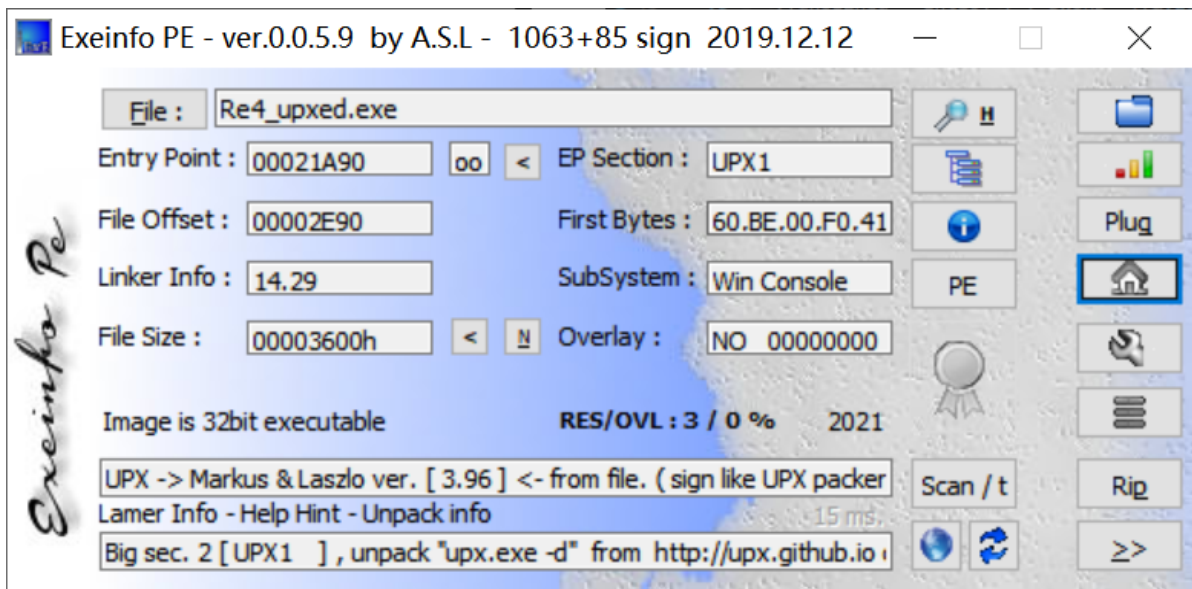
首先你需要一个Linux环境，比如说Ubuntu虚拟机。并在上面运行IDA的linux_server64:

搭建好动态调试环境之后就可以通过动调来查看v22的值了，是0x5AD，即1453：

将输入改为1453即可得到正确的flag:

[RE4]upxed

查壳，发现有upx壳：



这题说是不能直接脱壳，但直接脱壳又确实可以，挺奇怪的：

```
D:\CTF\SCUCTF-XSS-2021\Review\Release\4>Re4_upxed.exe
This is a Upx packed program. Inside it is an Encryption Algorithm.
But don't try to automatically unpack it!
You have to manually unpack it using IDA or Ollydbg.
Have fun!
=====
Input your flag:
```

用upx直接脱壳：

```
D:\CTF\SCUCTF-XSS-2021\Review\Release\4>upx -d Re4_upxed.exe -o dump_.exe
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2020
UPX 3.96w Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020

File size      Ratio      Format      Name
-----
41984 <- 13824 32.93% win32/pe dump_.exe

Unpacked 1 file.
```

找到关键代码，应该是把输入加密之后与密文进行比较：

```

8
9  __checkForDebuggerJustMyCode(&unk_41C015);
10 qmemcpy(v8, "THIS_IS_THE_KEYa", 16);
11 v6 = 0x5618435D;
12 v7[0] = -1608884855;
13 v7[1] = 1372647817;
14 v7[2] = 1872257453;
15 v7[3] = 835943406;
16 v7[4] = -1137280198;
17 puts(Buffer);
18 puts(off_41A004);
19 puts(off_41A008);
20 puts(off_41A00C);
21 puts(off_41A010);
22 j_memset(Str, 0, 0x64u);
23 puts("Input your flag:");
24 sub_41127B("%100s", (char)Str);
25 if ( j_strlen((const char *)Str) != 24 )
26 {
27     puts("Wrong!");
28     exit(0);
29 }
30 for ( i = 0; i < 3; ++i )
31 {
32     sub_41100A(&Str[i], v8);
33     if ( LODWORD(Str[i]) != (&v6 + 2 * i) || HIWORD(Str[i]) != v7[2 * i] )
34     {
35         puts("Wrong!");
36         exit(0);
37     }
38 }
39 puts("Correct! flag is your input!");
40 return 0;
41}
00001142| main 0:32 (411D42)|

```

这样看起来还有点抽象，稍作些修改，可以看到密文很明显就是v6了：

```

5 QWORD v6[4]; // [esp+154h] [ebp-38h]
6 int v7[5]; // [esp+174h] [ebp-18h] BYREF
7
8 __checkForDebuggerJustMyCode(&unk_41C015);
9 qmemcpy(v7, "THIS_IS_THE_KEYa", 16);
10 LODWORD(v6[0]) = 0x5618435D;
11 HIWORD(v6[0]) = 0xA01A5D89;
12 LODWORD(v6[1]) = 0x51D0F189;
13 HIWORD(v6[1]) = 0x6F9861AD;
14 LODWORD(v6[2]) = 0x31D37BEE;
15 HIWORD(v6[2]) = 0xBC367B3A;
16 puts(Buffer);
17 puts(off_41A004);
18 puts(off_41A008);
19 puts(off_41A00C);
20 puts(off_41A010);
21 j_memset(Str, 0, 0x64u);
22 puts("Input your flag:");
23 sub_41127B("%100s", (char)Str);
24 if ( j_strlen((const char *)Str) != 24 )
25 {
26     puts("Wrong!");
27     exit(0);
28 }
29 for ( i = 0; i < 3; ++i )
30 {
31     sub_41100A((int)&Str[i], (int)v7);
32     if ( LODWORD(Str[i]) != LODWORD(v6[i]) || HIWORD(Str[i]) != HIWORD(v6[i]) )
33     {
34         puts("Wrong!");
35         exit(0);
36     }
37 }
38 puts("Correct! flag is your input!");
39 return 0;

```

加密是很明显的TEA，密钥是之前就找到的"THIS_IS_THE_KEYa"：

```
IDA View-A Pseudocode-A Hex View-1
1 int __cdecl sub_411920(unsigned int *a1, _DWORD *a2)
2 {
3     int result; // eax
4     unsigned int i; // [esp+10Ch] [ebp-2Ch]
5     int v4; // [esp+118h] [ebp-20h]
6     unsigned int v5; // [esp+124h] [ebp-14h]
7     unsigned int v6; // [esp+130h] [ebp-8h]
8
9     __CheckForDebuggerJustMyCode(&unk_41C015);
10    v6 = *a1;
11    v5 = a1[1];
12    v4 = 0;
13    for ( i = 0; i < 0x20; ++i )
14    {
15        v4 -= 0x61C88647;
16        v6 += (a2[1] + (v5 >> 5)) ^ (v4 + v5) ^ (*a2 + 16 * v5);
17        v5 += (a2[3] + (v6 >> 5)) ^ (v4 + v6) ^ (a2[2] + 16 * v6);
18    }
19    *a1 = v6;
20    result = 4;
21    a1[1] = v5;
22    return result;
23 }
```

直接去WikiPedia抄一份TEA源码，dump密文和密钥进行解密即可：

```
#include <stdint.h>
#include <stdio.h>

void encrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0, i; /* set up */
    uint32_t delta=0x9e3779b9; /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3]; /* cache key */
    for (i=0; i < 32; i++) { /* basic cycle start */
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
    } /* end cycle */
    v[0]=v0; v[1]=v1;
}

void decrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0xc6ef3720, i; /* set up */
    uint32_t delta=0x9e3779b9; /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3]; /* cache key */
    for (i=0; i<32; i++) { /* basic cycle start */
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        sum -= delta;
    } /* end cycle */
    v[0]=v0; v[1]=v1;
}

int main(){
    uint32_t enc[6];
    enc[0] = 0x5618435D;
    enc[1] = 0xA01A5D89;
    enc[2] = 0x51D0F189;
```

```

enc[3] = 0x6F9861AD;
enc[4] = 0x31D37BEE;
enc[5] = 0xBC367B3A;
uint8_t key[20] = "THIS_IS_THE_KEYa";
for(int i = 0; i < 3; i++){
    decrypt(enc + 2 * i, (uint32_t*)key);
}
puts((char*)enc);
}

```

输出:

```
scuctf{Upx_with_TEA_Alg}
```

当然也可以手动脱壳，CTF-Wiki的脱壳教程: <https://ctf-wiki.org/reverse/windows/unpack/esp/>

[RE5]ezpak

APK文件可以用jadx打开，找到MainActivity类。可以看到这里是调用了个native函数 `check`，native函数是用C语言写的，编译成共享库（后缀为.so的文件）后塞到APK里供Java代码调用：

```

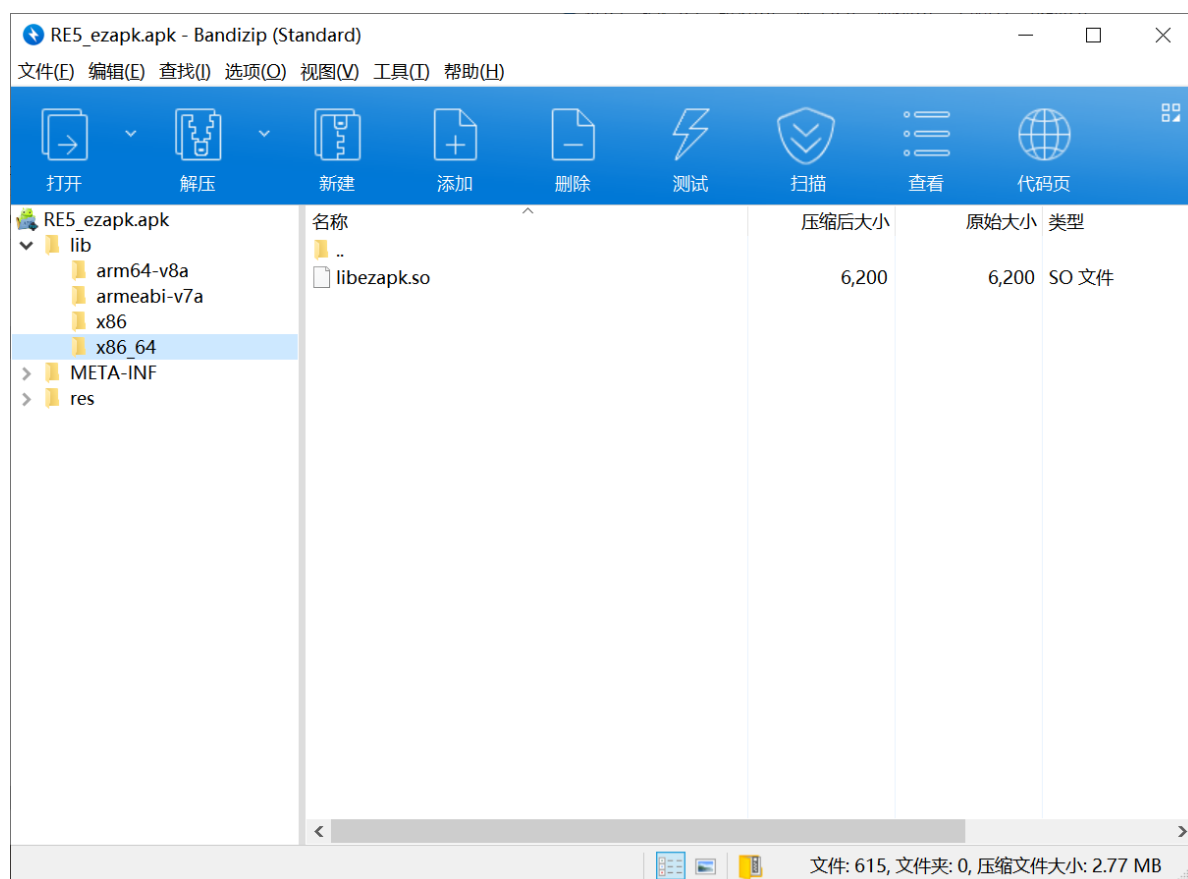
s    static {
35        System.loadLibrary("ezapk");
    }

/    /* access modifiers changed from: protected */
43p public void onCreate(Bundle bundle) {
44     super.onCreate(bundle);
45     setContentView((int) R.layout.activity_main);
46     this.password = (EditText) findViewById(R.id.password);
47     this.result = (TextView) findViewById(R.id.result);
48     Button button = (Button) findViewById(R.id.submit);
48     this.submit = button;
49     button.setOnClickListener(new MainActivity$$ExternalSyntheticLambda0(this));
    }

/    /* renamed from: Lambda$onCreate$0$cn-bluesadi-ezapk-MainActivity reason: not valid java name */
49p public /* synthetic */ void onCreate$0$cn-bluesadi-ezapkMainActivity(View view) {
50     String obj = this.password.getText().toString();
51     if (check(obj)) {
52         this.result.setText(String.format("Success!\nscuctf{%s}", new Object[]{MD5(obj)}));
53         this.result.setTextColor(getColor(R.color.green));
56         return;
    }
55     this.result.setText("Wrong password.");
56     this.result.setTextColor(getColor(R.color.red));
    }
}

```

APK本质上是一个压缩包，我们用解压缩软件打开APK，可以看到lib文件夹内有各种架构下的共享库文件：



不同架构的so文件反编译结果会有差异，可以每种架构都试一遍，选最容易看的一种。这里我选的是x86_64架构，先还原参数类型（非ARM架构需要手动导入jni.h头文件）：

```
1 bool __fastcall Java_cn_bluesadi_ezapk_MainActivity_check(JNIEnv *a1, jobject a2, jstring a3)
2 {
3     const char *v3; // rbx
4     m128i v4; // xmm1
```

第一步是将输入转存到buffer，注意这里输入的字符顺序发生了变化：

```

9  v8 = __readfsqword(0x28u);
10 input = (*a1)->GetStringUTFChars(a1, &v6);
11 if ( strlen(input) != 22 )
12     return 0;
13 *(_OWORD *)buffer = 0LL;
14 *(_QWORD *)&buffer[14] = 0LL;
15 buffer[0] = *input;
16 buffer[2] = input[1];
17 buffer[4] = input[2];
18 buffer[6] = input[3];
19 buffer[8] = input[4];
20 buffer[10] = input[5];
21 buffer[12] = input[6];
22 *(_QWORD *)&buffer[14] = *((unsigned __int8 *)input + 7);
23 *(_DWORD *)&buffer[16] = *((unsigned __int8 *)input + 8);
24 *(_WORD *)&buffer[18] = *((unsigned __int8 *)input + 9);
25 *(_WORD *)&buffer[20] = *((unsigned __int8 *)input + 10);
26 buffer[1] = input[11];
27 buffer[3] = input[12];
28 buffer[5] = input[13];
29 buffer[7] = input[14];
30 buffer[9] = input[15];
31 buffer[11] = input[16];
32 buffer[13] = input[17];
33 buffer[15] = input[18];
34 buffer[17] = input[19];
35 buffer[19] = input[20];
36 buffer[21] = input[21];

```

这一部分很容易总结出规律，可以看做是把输入分成两个部分，然后穿插：

```

for(int i = 0; i < 22; i += 1){
    for(int j = 0; j < 22; j += 2){
        buf[i + j] = input[ptr++];
    }
}

```

后面则是一个异或加密和比较的过程：

```

37 v4 = _mm_load_si128((const __m128i *)&__xmmword_860);
38 *(_m128i *)&buffer[1] = _mm_xor_si128(_mm_add_epi8(_mm_loadu_si128((const __m128i *)&buffer[1]), v4), v4);
39 buffer[17] = (buffer[17] + 17) ^ 0x11;
40 buffer[18] = (buffer[18] + 18) ^ 0x12;
41 buffer[19] = (buffer[19] + 19) ^ 0x13;
42 buffer[20] = (buffer[20] + 20) ^ 0x14;
43 buffer[21] = (buffer[21] + 21) ^ 0x15;
44 return _mm_movemask_epi8(
45     _mm_and_si128(
46         _mm_cmpeq_epi8(_mm_load_si128((const __m128i *)buffer), enc),
47         _mm_cmpeq_epi8(
48             _mm_loadu_si128((const __m128i *)&enc.m128i_i8[6]),
49             _mm_loadu_si128((const __m128i *)&buffer[6])))) == 0xFFFF;
50}

```

虽然比较零散，但可以看出来规律，归纳出来是一个这样的加密过程：

```

for(int i = 0; i < 22; i ++){
    buf[i] += i;
    buf[i] ^= i;
}

```

写出exp:

```
#include <stdio>

int main(){
    unsigned char enc[23] =
"\x79\x72\x73\x71\x7d\x6b\x63\x6c\x61\x61\x76\x79\x7d\x7f\x63\x72\x61\x6b\x92\x9b\x6c\x9d";
    for(int i = 0; i < 22; i++){
        enc[i] ^= i;
        enc[i] -= i;
    }
    int ptr = 0;
    unsigned char passwd[23] = {0};
    for(int i = 0; i < 2; i += 1){
        for(int j = 0; j < 22; j += 2){
            passwd[ptr++] = enc[i + j];
        }
    }
    printf("%s\n", passwd);
}
```

[RE6]EzLinearEquation

这一段本质上是构造了一个线性方程组，参数存储在mat数组里，这个线性方程组的解即是flag：

```
scanf("%40s", Str);
if ( strlen(Str) == 27 && !strcmp(Str, "scuctf{", 7ui64) && BYTE2(v7) == 125 )
{
    v9 = &Str[7];
    v12 = 0i64;
    for ( i = 0; i <= 18; ++i )
    {
        v12 = 0i64;
        for ( j = 0; j <= 18; ++j )
            v12 += mat[19 * i + j] * (unsigned __int8)v9[j];
        if ( v12 != enc[i] )
            goto LABEL_10;
    }
    puts("Correct!");
    result = 0;
}
else
{
```

求解方程组使用的库是z3，我之前写过一个[教程](#)。dump参数和密文，编写Python脚本求解方程即可：

```
from z3 import *
```

```
mat = [0x0000000000000010, 0x0000000000000051, 0x0000000000000002,
0x0000000000000023, 0x0000000000000023, 0x0000000000000064, 0x0000000000000008,
0x0000000000000032, 0x000000000000001B, 0x0000000000000064, 0x0000000000000034,
0x000000000000003E, 0x000000000000002E, 0x0000000000000017, 0x000000000000000A,
0x0000000000000014, 0x000000000000001D, 0x000000000000004D, 0x000000000000002A,
0x0000000000000062, 0x0000000000000040, 0x000000000000003F, 0x000000000000005D,
0x000000000000002E, 0x0000000000000013, 0x0000000000000049, 0x0000000000000022,
0x000000000000000F, 0x0000000000000063, 0x000000000000004E, 0x0000000000000058,
0x000000000000003B, 0x0000000000000060, 0x0000000000000032, 0x0000000000000043,
0x000000000000002E, 0x000000000000002A, 0x000000000000002D, 0x000000000000004A,
0x0000000000000040, 0x0000000000000063, 0x0000000000000004, 0x000000000000002B,
0x000000000000001F, 0x000000000000002D, 0x0000000000000010, 0x000000000000004B,
0x0000000000000056, 0x0000000000000007, 0x0000000000000062, 0x000000000000004B,
0x000000000000000F, 0x000000000000002F, 0x0000000000000024, 0x0000000000000043,
0x000000000000005C, 0x000000000000004A, 0x000000000000003A, 0x000000000000001C,
0x0000000000000030, 0x0000000000000025, 0x0000000000000022, 0x0000000000000005,
0x0000000000000049, 0x000000000000005A, 0x0000000000000045, 0x000000000000005F,
0x0000000000000026, 0x000000000000003D, 0x000000000000001B, 0x000000000000001C,
0x0000000000000025, 0x000000000000000C, 0x000000000000000E, 0x000000000000004C,
0x0000000000000020, 0x000000000000004D, 0x0000000000000021, 0x000000000000003E,
0x0000000000000053, 0x0000000000000020, 0x0000000000000021, 0x0000000000000056,
0x000000000000003B, 0x000000000000001F, 0x000000000000005E, 0x0000000000000051,
0x000000000000000A, 0x000000000000003C, 0x000000000000001E, 0x0000000000000047,
0x0000000000000039, 0x0000000000000021, 0x000000000000005E, 0x000000000000002F,
0x0000000000000049, 0x000000000000001A, 0x0000000000000058, 0x0000000000000062,
0x000000000000003C, 0x0000000000000019, 0x0000000000000017, 0x0000000000000036,
0x000000000000002F, 0x0000000000000016, 0x0000000000000052, 0x000000000000004B,
0x000000000000003F, 0x000000000000004A, 0x0000000000000029, 0x000000000000001A,
0x0000000000000020, 0x0000000000000002, 0x0000000000000047, 0x0000000000000062,
0x000000000000001E, 0x0000000000000040, 0x000000000000003B, 0x000000000000000D,
0x000000000000002F, 0x0000000000000061, 0x0000000000000013, 0x0000000000000038,
0x000000000000002D, 0x0000000000000023, 0x000000000000001D, 0x0000000000000051,
0x0000000000000047, 0x0000000000000058, 0x000000000000003F, 0x0000000000000039,
0x0000000000000060, 0x0000000000000001, 0x0000000000000006, 0x0000000000000004,
0x000000000000005E, 0x0000000000000034, 0x0000000000000028, 0x000000000000001B,
0x000000000000000F, 0x0000000000000012, 0x0000000000000064, 0x0000000000000009,
0x0000000000000017, 0x0000000000000036, 0x0000000000000005, 0x0000000000000048,
0x000000000000005B, 0x000000000000000D, 0x000000000000004E, 0x0000000000000031,
0x000000000000003B, 0x0000000000000050, 0x000000000000003F, 0x000000000000000A,
0x0000000000000019, 0x0000000000000042, 0x0000000000000051, 0x000000000000003A,
0x000000000000004F, 0x000000000000004C, 0x000000000000003E, 0x0000000000000016,
0x0000000000000012, 0x0000000000000019, 0x0000000000000003, 0x0000000000000032,
0x0000000000000046, 0x0000000000000023, 0x000000000000000D, 0x000000000000005D,
0x0000000000000049, 0x000000000000004D, 0x0000000000000003, 0x0000000000000058,
0x0000000000000064, 0x0000000000000030, 0x000000000000002F, 0x0000000000000037,
0x0000000000000020, 0x0000000000000044, 0x0000000000000017, 0x0000000000000052,
0x0000000000000025, 0x000000000000001E, 0x000000000000005A, 0x0000000000000017,
0x0000000000000057, 0x0000000000000058, 0x0000000000000023, 0x000000000000002A,
0x0000000000000004, 0x000000000000004F, 0x0000000000000032, 0x000000000000000D,
0x000000000000003F, 0x0000000000000008, 0x000000000000005C, 0x0000000000000023,
0x000000000000001D, 0x0000000000000053, 0x000000000000003C, 0x0000000000000005,
0x0000000000000055, 0x000000000000002F, 0x0000000000000055, 0x0000000000000017,
0x0000000000000003, 0x000000000000004A, 0x0000000000000026, 0x0000000000000053,
0x000000000000004D, 0x000000000000005A, 0x0000000000000029, 0x0000000000000036,
0x000000000000001D, 0x0000000000000035, 0x0000000000000025, 0x000000000000004D,
0x0000000000000037, 0x0000000000000031, 0x0000000000000059, 0x000000000000004C,
0x000000000000005A, 0x000000000000001C, 0x0000000000000055, 0x0000000000000052,
0x000000000000005A, 0x000000000000002B, 0x0000000000000017, 0x0000000000000048,
```

```

0x000000000000004B, 0x0000000000000008, 0x0000000000000018, 0x0000000000000057,
0x0000000000000046, 0x000000000000000F, 0x0000000000000039, 0x0000000000000042,
0x000000000000003E, 0x0000000000000061, 0x0000000000000042, 0x0000000000000060,
0x000000000000004E, 0x0000000000000038, 0x000000000000002A, 0x0000000000000058,
0x000000000000003B, 0x0000000000000051, 0x000000000000001E, 0x000000000000001A,
0x0000000000000038, 0x000000000000005E, 0x0000000000000017, 0x000000000000003E,
0x0000000000000003, 0x0000000000000057, 0x0000000000000010, 0x000000000000002E,
0x0000000000000052, 0x0000000000000017, 0x000000000000005B, 0x0000000000000042,
0x0000000000000060, 0x0000000000000063, 0x0000000000000007, 0x0000000000000003,
0x0000000000000014, 0x0000000000000036, 0x0000000000000004, 0x0000000000000019,
0x0000000000000030, 0x000000000000003C, 0x0000000000000063, 0x0000000000000047,
0x000000000000002C, 0x0000000000000026, 0x0000000000000042, 0x0000000000000019,
0x000000000000003C, 0x0000000000000005, 0x0000000000000044, 0x000000000000000F,
0x000000000000004A, 0x0000000000000051, 0x0000000000000062, 0x0000000000000033,
0x0000000000000036, 0x0000000000000058, 0x0000000000000038, 0x0000000000000042,
0x0000000000000011, 0x0000000000000014, 0x000000000000004B, 0x000000000000005A,
0x0000000000000003, 0x0000000000000062, 0x0000000000000027, 0x0000000000000019,
0x0000000000000032, 0x000000000000001D, 0x0000000000000055, 0x000000000000003D,
0x0000000000000052, 0x0000000000000034, 0x0000000000000060, 0x000000000000001E,
0x0000000000000040, 0x000000000000004B, 0x0000000000000011, 0x0000000000000009,
0x0000000000000046, 0x0000000000000056, 0x000000000000000F, 0x0000000000000036,
0x0000000000000040, 0x000000000000000F, 0x0000000000000004, 0x0000000000000042,
0x000000000000000E, 0x0000000000000029, 0x000000000000003C, 0x0000000000000030,
0x0000000000000005, 0x0000000000000057, 0x0000000000000008, 0x0000000000000035,
0x000000000000002A, 0x0000000000000012, 0x000000000000002E, 0x000000000000005B,
0x000000000000000A, 0x0000000000000048, 0x000000000000000C, 0x0000000000000002,
0x000000000000001E, 0x000000000000005A, 0x0000000000000019, 0x0000000000000060,
0x0000000000000015, 0x000000000000002A, 0x0000000000000046, 0x0000000000000006,
0x0000000000000031, 0x0000000000000007, 0x0000000000000034, 0x0000000000000018,
0x0000000000000047, 0x0000000000000005, 0x000000000000003E, 0x0000000000000023,
0x0000000000000021, 0x0000000000000056, 0x0000000000000014, 0x0000000000000040,
0x0000000000000053, 0x000000000000005F, 0x000000000000004C, 0x0000000000000040,
0x0000000000000044, 0x000000000000005F, 0x0000000000000000, 0x0000000000000000,
0x0000000000000000]
enc = [0x000000000001424A, 0x000000000001AA02, 0x000000000001962C,
0x0000000000013FF4, 0x00000000000183E3, 0x0000000000016B57, 0x0000000000017BEC,
0x00000000000133E2, 0x00000000000162AA, 0x00000000000195AC, 0x0000000000014AF1,
0x000000000001D1D0, 0x0000000000019EE2, 0x0000000000019135, 0x0000000000014E8B,
0x0000000000019ADF, 0x0000000000015600, 0x0000000000013D5A, 0x000000000001827F,
0x0000000000000000]
X = [Int('x%d' % i) for i in range(19)]
solver = Solver()
for i in range(19):
    equation = 0
    for j in range(19):
        equation += mat[i * 19 + j] * X[j]
    solver.add(equation == enc[i])
print(solver.check())
model = solver.model()
print(''.join([chr(model[X[i]].as_long()) for i in range(19)]))

```

[RE7]VirtualMachine

解法一：分析虚拟机

一道典型的虚拟机逆向题，虽然没有设置什么陷阱或者与其他技术综合，但还是有一定难度，需要有一定的汇编基础以及对操作系统的抽象架构有所了解。

首先是VM的初始化：

```
1 int __cdecl VM_init(void **a1, int a2)
2 {
3     _DWORD *v3; // [esp+D0h] [ebp-8h]
4
5     __CheckForDebuggerJustMyCode(&unk_41D02F);
6     *a1 = malloc(0x2Cu);
7     if ( !*a1 )
8     {
9         puts("Error while mallocing for VM");
10        exit(1);
11    }
12    v3 = *a1;
13    *v3 = 0;
14    v3[1] = 1;
15    v3[2] = 2;
16    v3[3] = 3;
17    v3[4] = a2;
18    v3[5] = a2;
19    v3[6] = malloc(0x1000u);
20    if ( !v3[6] )
21    {
22        puts("Error while mallocing for SP");
23        exit(1);
24    }
25    v3[7] = malloc(0x100u);
26    if ( !v3[7] )
27    {
28        puts("Error while mallocing for PWRITE");
29        exit(1);
30    }
31    v3[8] = v3[7];
32    return 0;
33 }
```

然后是VM正式开始运行，从红框中可以看出这题的VM使用了结构体，所以逆向起来会更加复杂，为了方便分析VM架构以及dump出VM指令，需要首先还原VM结构体：

```

1 void __cdecl VM_start(int a1)
2 {
3     int v1; // ecx
4     int v2; // ecx
5     int i; // [esp+D4h] [ebp-44h]
6     int v4; // [esp+E0h] [ebp-38h]
7     unsigned __int8 v5; // [esp+113h] [ebp-5h]
8     unsigned __int8 v6; // [esp+113h] [ebp-5h]
9
10    __CheckForDebuggerJustMyCode(&unk_41D02F);
11    v4 = 0;
12    while ( 1 )
13    {
14        switch ( **(_BYTE **)(a1 + 16) )
15        {
16            case 0xA:
17                sub_411398(
18                    a1 + 4 * *(unsigned __int8 *)((_DWORD *)(a1 + 16) + 1),
19                    a1 + 4 * *(unsigned __int8 *)((_DWORD *)(a1 + 16) + 2));
20                *(_DWORD *)(a1 + 16) += 3;
21                break;
22            case 0xB:
23                sub_41137F(
24                    a1 + 4 * *(unsigned __int8 *)((_DWORD *)(a1 + 16) + 1),
25                    a1 + 4 * *(unsigned __int8 *)((_DWORD *)(a1 + 16) + 2));
26                *(_DWORD *)(a1 + 16) += 3;
27                break;
28            case 0xC:
29                sub_411069(
30                    a1 + 4 * *(unsigned __int8 *)((_DWORD *)(a1 + 16) + 1),
31                    a1 + 4 * *(unsigned __int8 *)((_DWORD *)(a1 + 16) + 2));
32                *(_DWORD *)(a1 + 16) += 3;
33                break;
34            case 0xD:

```

还原结构体的方法是在 `structures` 中插入自定义的新结构体，并手动设置每个结构体变量的大小、类型等，这里我们自定义了一个 VM 结构体。还原需要根据代码推断结构体变量的作用，具体的过程就不细说了，总之是个苦差事，需要耐心和时间：

```

Pseudocode-A  Hex View-1  Structures  En
00000000 ; Ins/Del : create/delete structure
00000000 ; D/A/* : create structure member (data/ascii/array)
00000000 ; N : rename structure or structure member
00000000 ; U : delete structure member
00000000 ; [00000010 BYTES. COLLAPSED STRUCT GUID. PRESS CTRL-NUMPAD+
00000000 : -----
00000000
00000000 VM      struc ; (sizeof=0x2C, mappedto_51)
00000000 regs      dd 4 dup(?)
00000010 PC       dd ? ; offset
00000014 PC_base  dd ?
00000018 sp_       dd ? ; offset
0000001C mem       dd ?
00000020 mem_ptr    dd ?
00000024 len       dd ?
00000028 flag      dd ?
0000002C VM      ends
00000030 ; [00000010 BYTES. COLLAPSED STRUCT GUID. PRESS CTRL-NUMPAD+

```

还原结构体后的代码如图所示，还是比较清晰的：

```

66     case 0xFu:
67         rsh((int)&a1->regs[a1->PC[1]], (int)&a1->regs[a1->PC[2]]);
68         a1->PC += 3;
69         break;
70     case 0x10u:
71         rol((int)&a1->regs[a1->PC[1]], (int)&a1->regs[a1->PC[2]]);
72         a1->PC += 3;
73         break;
74     case 0x11u:
75         ror((int)&a1->regs[a1->PC[1]], (int)&a1->regs[a1->PC[2]]);
76         a1->PC += 3;
77         break;
78     case 0x12u:                                     // mov_r2r
79         a1->regs[a1->PC[1]] = a1->regs[a1->PC[2]];
80         a1->PC += 3;
81         break;
82     case 0x13u:
83         a1->regs[a1->PC[1]] = _byteswap_ulong(*(_DWORD*)(a1->PC + 2)); // mov_i2r
84         a1->PC += 6;
85         break;
86     case 0x14u:                                     // push reg
87         *a1->sp_ = a1->regs[a1->PC[1]];
88         a1->sp_ += 4;
89         a1->PC += 2;
90         break;
91     case 0x15u:                                     // push imm
92         *a1->sp_ = _byteswap_ulong(*(_DWORD*)(a1->PC + 1));
93         a1->sp_ += 4;
94         a1->PC += 5;
95         break;
96     case 0x16u:                                     // pop reg
97         v5 = a1->PC[1];
98         a1->sp_ -= 4;

```

分析出每个opcode对应的功能后，dump字节码为一种类似汇编的形式，dump脚本如下：

```

#include <stdio>
#include <stdint>
#include <stdlib.h>
#define READ_WORD(PC) ((code[PC] << 8) | code[PC + 1])
#define READ_DWORD(PC) __builtin_bswap32(*(uint32_t*)&code[PC])

uint8_t code[1176] = {
    0x23, 0x00, 0x65, 0x1B, 0x1E, 0x00, 0x13, 0x00, 0x00, 0x00, 0x00, 0x0A,
    0x1A, 0x00, 0x1C, 0x01,
    0x13, 0x00, 0x00, 0x00, 0x00, 0x57, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00,
    0x00, 0x00, 0x00, 0x72,
    0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x6F, 0x1A, 0x00,
    0x1C, 0x01, 0x13, 0x00,
    0x00, 0x00, 0x00, 0x6E, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00,
    0x00, 0x67, 0x1A, 0x00,
    0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x21, 0x1A, 0x00, 0x1C, 0x01,
    0x13, 0x00, 0x00, 0x00,
    0x00, 0x20, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x0A,
    0x1A, 0x00, 0x1C, 0x01,
    0x1E, 0x00, 0x18, 0x08, 0xFF, 0x13, 0x00, 0x00, 0x00, 0x00, 0x48, 0x1A,
    0x00, 0x1C, 0x01, 0x13,
    0x00, 0x00, 0x00, 0x00, 0x65, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00,
    0x00, 0x00, 0x6C, 0x1A,
    0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x6C, 0x1A, 0x00, 0x1C,
    0x01, 0x13, 0x00, 0x00,
    0x00, 0x00, 0x6F, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00,
    0x21, 0x1A, 0x00, 0x1C,
    0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x20, 0x1A, 0x00, 0x1C, 0x01, 0x13,
    0x00, 0x00, 0x00, 0x00,

```


[illegible]

0x00, 0x03, 0x19, 0x00, 0x1C, 0x01, 0x13, 0x02, 0x00, 0x00, 0x00, 0x7B,
0x20, 0x00, 0x02, 0x22,
0x00, 0x03, 0x1E, 0x00, 0x1C, 0x19, 0x19, 0x00, 0x13, 0x02, 0x00, 0x00,
0x00, 0x7D, 0x20, 0x00,
0x02, 0x22, 0x00, 0x03, 0x1E, 0x00, 0x1C, 0x07, 0x15, 0x00, 0x00, 0x00,
0x12, 0x15, 0x00, 0x00,
0x00, 0x01, 0x19, 0x00, 0x16, 0x01, 0x0D, 0x00, 0x01, 0x0A, 0x00, 0x01,
0x1A, 0x00, 0x14, 0x01,
0x1C, 0x01, 0x16, 0x02, 0x16, 0x03, 0x20, 0x02, 0x03, 0x24, 0x16, 0x14,
0x00, 0x13, 0x00, 0x00,
0x00, 0x00, 0x01, 0x0A, 0x02, 0x00, 0x16, 0x00, 0x14, 0x03, 0x14, 0x02,
0x23, 0x02, 0xB2, 0x1E,
0x07, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x45, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x7A, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x5F, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x56, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x71, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x7A, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x7A, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x85, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x71, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x70, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x5F, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x4D, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x79, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x7B, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x76, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x89, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x90, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x19, 0x02, 0x13, 0x03, 0x00, 0x00, 0x00, 0x89, 0x20, 0x02, 0x03,
0x22, 0x00, 0x03, 0x1C,
0x01, 0x1B, 0x1E, 0x00, 0x13, 0x00, 0x00, 0x00, 0x00, 0x0A, 0x1A, 0x00,
0x1C, 0x01, 0x13, 0x00,
0x00, 0x00, 0x00, 0x43, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00,
0x00, 0x6F, 0x1A, 0x00,
0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x72, 0x1A, 0x00, 0x1C, 0x01,
0x13, 0x00, 0x00, 0x00,
0x00, 0x72, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x63,
0x1A, 0x00, 0x1C, 0x01,
0x13, 0x00, 0x00, 0x00, 0x00, 0x74, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00,
0x00, 0x00, 0x00, 0x21,

```

    0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x0A, 0x1A, 0x00,
    0x1C, 0x01, 0x1E, 0x00,
    0x18, 0x08, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

```

```

uint32_t PC = 0;
uint32_t mem_ptr = 0;
uint32_t imm;

```

```

int main(){
    freopen("dump.txt", "w", stdout);
    while(code[PC]){
        printf("%d: ", PC);
        switch (code[PC]){
            case 0xA:
                printf("add r%d, r%d\n", code[PC + 1], code[PC + 2]);
                PC += 3;
                break;
            case 0xB:
                printf("sub r%d, r%d\n", code[PC + 1], code[PC + 2]);
                PC += 3;
                break;
            case 0xC:
                printf("mul r%d, r%d\n", code[PC + 1], code[PC + 2]);
                PC += 3;
                break;
            case 0xD:
                printf("xor r%d, r%d\n", code[PC + 1], code[PC + 2]);
                PC += 3;
                break;
            case 0xE:
                printf("lsh r%d, r%d\n", code[PC + 1], code[PC + 2]);
                PC += 3;
                break;
            case 0xF:
                printf("rsh r%d, r%d\n", code[PC + 1], code[PC + 2]);
                PC += 3;
                break;
            case 0x10:
                printf("rol r%d, r%d\n", code[PC + 1], code[PC + 2]);
                PC += 3;
                break;
            case 0x11:
                printf("ror r%d, r%d\n", code[PC + 1], code[PC + 2]);
                PC += 3;
                break;
            case 0x12:
                printf("mov r%d, r%d\n", code[PC + 1], code[PC + 2]);
                PC += 3;
                break;
            case 0x13:
                imm = READ_DWORD(PC + 2);

```

```

        if(imm >= 32 && imm <= 126){
            printf("mov r%d, %d(\'%c\')\n", code[PC + 1], imm, imm);
        }else{
            printf("mov r%d, %d\n", code[PC + 1], imm);
        }
        PC += 6;
        break;
    case 0x14:
        printf("push r%d\n", code[PC + 1]);
        PC += 2;
        break;
    case 0x15:
        printf("push %x\n", READ_DWORD(PC + 1));
        PC += 5;
        break;
    case 0x16:
        printf("pop r%d\n", code[PC + 1]);
        PC += 2;
        break;
    case 0x17:
        printf("read\n");
        PC ++;
        break;
    case 0x18:
        printf("write %d\n", code[PC + 1]);
        PC += 2;
        break;
    case 0x19:
        printf("mov r%d, mem[%d]\n", code[PC + 1], mem_ptr);
        PC += 2;
        break;
    case 0x1A:
        printf("mov mem[%d], r%d\n", mem_ptr, code[PC + 1]);
        PC += 2;
        break;
    case 0x1B:
        printf("clear_mem\n");
        PC ++;
        break;
    case 0x1C:
        printf("add_mem_ptr %d\n", code[PC + 1]);
        mem_ptr += code[PC + 1];
        PC += 2;
        break;
    case 0x1D:
        printf("sub_mem_ptr %d\n", code[PC + 1]);
        mem_ptr -= code[PC + 1];
        PC += 2;
        break;
    case 0x1E:
        printf("set_mem_ptr %d\n", code[PC + 1]);
        mem_ptr = code[PC + 1];
        PC += 2;
        break;
    case 0x1F:
        printf("mov r0, len\n");
        PC ++;
        break;

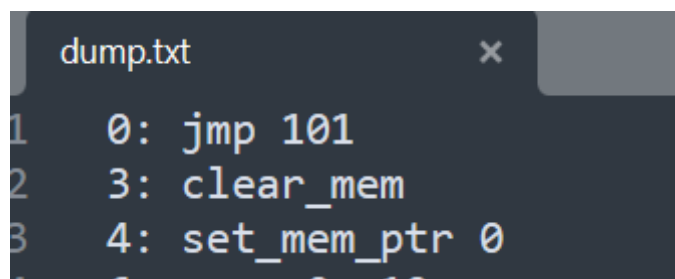
```

```

        case 0x20:
            printf("cmp r%d, r%d\n", code[PC + 1], code[PC + 2]);
            PC += 3;
            break;
        case 0x21:
            printf("je %d\n", READ_WORD(PC + 1));
            PC += 3;
            break;
        case 0x22:
            printf("jne %d\n", READ_WORD(PC + 1));
            PC += 3;
            break;
        case 0x23:
            printf("jmp %d\n", READ_WORD(PC + 1));
            PC += 3;
            break;
        case 0x24:
            printf("je_rel %d\n", PC + code[PC + 1]);
            PC += 2;
            break;
        case 0x25:
            printf("jne_rel %d\n", PC + code[PC + 1]);
            PC += 2;
            break;
        case 0x26:
            printf("jmp_rel %d\n", PC + code[PC + 1]);
            PC += 2;
            break;
        case 0x27:
            printf("debug\n");
            PC ++;
            break;
        case 0x28:
            printf("nop\n");
            PC ++;
            break;
        case 0xFF:
            printf("end\n");
            PC ++;
            break;
        default:
            printf("unknown code %d\n", code[PC]);
            PC ++;
            break;
    }
}
}

```

分析dump下来的VM指令，首先直接跳转到101位置处的指令：



```

dump.txt
1  0: jmp 101
2  3: clear_mem
3  4: set_mem_ptr 0
4  6: mem_ptr 10

```

看样子应该是一些输出信息，直接略过继续往下分析：

```
101: mov r0, 72('H')
107: mov mem[0], r0
109: add_mem_ptr 1
111: mov r0, 101('e')
117: mov mem[1], r0
119: add_mem_ptr 1
121: mov r0, 108('l')
127: mov mem[2], r0
129: add_mem_ptr 1
131: mov r0, 108('l')
137: mov mem[3], r0
139: add_mem_ptr 1
141: mov r0, 111('o')
147: mov mem[4], r0
149: add_mem_ptr 1
151: mov r0, 33('!')
157: mov mem[5], r0
159: add_mem_ptr 1
161: mov r0, 32(' ')
167: mov mem[6], r0
169: add_mem_ptr 1
171: mov r0, 87('W')
177: mov mem[7], r0
179: add_mem_ptr 1
181: mov r0, 101('e')
187: mov mem[8], r0
189: add_mem_ptr 1
191: mov r0, 108('l')
197: mov mem[9], r0
```

读取输入，判断输入的长度是不是26，以及格式是不是 `scutf{xxx}`：

```
164 528: clear_mem
165 530: read
166 531: set_mem_ptr 0
167 533: mov r0, len
168 534: mov r1, 26
169 540: cmp r0, r1
170 543: jne 3
171 546: mov r0, mem[0]
172 548: add_mem_ptr 1
173 550: mov r1, mem[1]
174 552: add_mem_ptr 1
175 554: mov r2, 115('s')
176 560: cmp r0, r2
177 563: jne 3
178 566: mov r3, 99('c')
179 572: cmp r1, r3
180 575: jne 3
181 578: mov r0, mem[2]
182 580: add_mem_ptr 1
183 582: mov r1, mem[3]
184 584: add_mem_ptr 1
185 586: mov r2, 117('u')
186 592: cmp r0, r2
187 595: jne 3
188 598: mov r3, 99('c')
189 604: cmp r1, r3
190 607: jne 3
191 610: mov r0, mem[4]
192 612: add_mem_ptr 1
193 614: mov r1, mem[5]
194 616: add_mem_ptr 1
195 618: mov r2, 116('t')
196 624: cmp r0, r2
197 627: jne 3
198 630: mov r3, 102('f')
199 636: cmp r1, r3
200 639: jne 3
201 642: mov r0, mem[6]
202 644: add_mem_ptr 1
```

```
203    646: mov r2, 123('{')
204    652: cmp r0, r2
```

接下来对 `scuctf{}` 内的输入进行循环加密，加密算法非常简单：

```
212    676: set_mem_ptr 0
213    678: add_mem_ptr 7
214    680: push 12
215    685: push 1
216    690: mov r0, mem[7]
217    692: pop r1
218    694: xor r0, r1
219    697: add r0, r1
220    700: mov mem[7], r0
221    702: push r1
222    704: add_mem_ptr 1
223    706: pop r2
224    708: pop r3
225    710: cmp r2, r3
226    713: je_rel 735
227    715: push r0
228    717: mov r0, 1
229    723: add r2, r0
230    726: pop r0
231    728: push r3
232    730: push r2
233    732: jmp 690
234    735: set_mem_ptr 7
235    737: mov r2, mem[7]
236    739: mov r3, 69('E')
237    745: cmp r2, r3
```

最后与密文做比较：


```

235 737: mov r2, mem[7]
236 739: mov r3, 69('E')
237 745: cmp r2, r3
238 748: jne 3
239 751: add_mem_ptr r 1
240 753: mov r2, mem[8]
241 755: mov r3, 112('z')
242 761: cmp r2, r3
243 764: jne 3
244 767: add_mem_ptr r 1
245 769: mov r2, mem[9]
246 771: mov r3, 95('_')
247 777: cmp r2, r3
248 780: jne 3
249 783: add_mem_ptr r 1
250 785: mov r2, mem[10]
251 787: mov r3, 86('V')
252 793: cmp r2, r3
253 796: jne 3
254 799: add_mem_ptr r 1
255 801: mov r2, mem[11]
256 803: mov r3, 113('q')
257 809: cmp r2, r3
258 812: jne 3
259 815: add_mem_ptr r 1
260 817: mov r2, mem[12]
261 819: mov r3, 112('z')
262 825: cmp r2, r3
263 828: jne 3
264 831: add_mem_ptr r 1

```

写出exp:

```

#include <stdio>

char flag[100] = "Ez_Vqzz\x85qp_My{v\x89\x90\x89";

int main(){
    for(int i = 1; flag[i - 1]; i++){
        flag[i - 1] = (flag[i - 1] - i) ^ i;
    }
    printf("scuctf{%s}", flag);
}

```

解法二：angr梭哈

之前写的一个angr教程：https://bluesadi.github.io/0x401RevTrain-Tools/angr/11_%E5%88%A9%E7%94%A8angr%E7%AC%A6%E5%8F%B7%E6%89%A7%E8%A1%8C%E6%A2%AD%E5%93%88VM%E7%B1%BBCTF%E8%B5%9B%E9%A2%98/

话不多说，先贴exp：

```
import angr

proj = angr.Project('RE7_VirtualMachine.exe', load_options={'auto_load_libs':
False})
proj.hook(addr=0x00413210, hook=angr.SIM_PROCEDURES['stubs']['Nop']()) #
__CheckForDebuggerJustMyCode
proj.hook(addr=0x004131E0, hook=angr.SIM_PROCEDURES['stubs']['Nop']()) #
j__RTC_CheckEsp
proj.hook(addr=0x00411037, hook=angr.SIM_PROCEDURES['libc']['scanf']())
proj.hook(addr=0x004110E6, hook=angr.SIM_PROCEDURES['libc']['printf']())
proj.hook_symbol('putchar', angr.SIM_PROCEDURES['libc']['putchar']())
proj.hook_symbol('exit', angr.SIM_PROCEDURES['libc']['exit']())
proj.hook_symbol('puts', angr.SIM_PROCEDURES['libc']['puts']())
proj.hook_symbol('memset', angr.SIM_PROCEDURES['libc']['memset']())
proj.hook_symbol('strlen', angr.SIM_PROCEDURES['libc']['strlen']())
proj.hook_symbol('malloc', angr.SIM_PROCEDURES['libc']['malloc']())

state = proj.factory.blank_state(addr=proj.loader.find_symbol('main_0'))
state.options.add(angr.options.ZERO_FILL_UNCONSTRAINED_MEMORY)
state.options.add(angr.options.ZERO_FILL_UNCONSTRAINED_REGISTERS)

simgr = proj.factory.simgr(state)
while len(simgr.active):
    print(simgr)
    for active in simgr.active:
        print(active.posix.dumps(1))
        if b'Correct' in active.posix.dumps(1):
            print(active.posix.dumps(0))
            exit(0)
    simgr.step()
```

有几个要注意的地方，第一是angr对PE文件的库函数识别效果很差，很多库函数都识别不出来，所以需要手动hook：


```
proj.hook(addr=0x00413210, hook=angr.SIM_PROCEDURES['stubs']['Nop']()) #
__CheckForDebuggerJustMyCode
proj.hook(addr=0x004131E0, hook=angr.SIM_PROCEDURES['stubs']['Nop']()) #
j__RTC_CheckEsp
proj.hook(addr=0x00411037, hook=angr.SIM_PROCEDURES['libc']['scanf']())
proj.hook(addr=0x004110E6, hook=angr.SIM_PROCEDURES['libc']['printf']())
proj.hook_symbol('putchar', angr.SIM_PROCEDURES['libc']['putchar']())
proj.hook_symbol('exit', angr.SIM_PROCEDURES['libc']['exit']())
proj.hook_symbol('puts', angr.SIM_PROCEDURES['libc']['puts']())
proj.hook_symbol('memset', angr.SIM_PROCEDURES['libc']['memset']())
proj.hook_symbol('strlen', angr.SIM_PROCEDURES['libc']['strlen']())
proj.hook_symbol('malloc', angr.SIM_PROCEDURES['libc']['malloc']())
```

第二点是需要用0填充未约束的寄存器和内存，否则符号执行会产生 unconstrained 状态：

```
state.options.add(angr.options.ZERO_FILL_UNCONSTRAINED_MEMORY)
state.options.add(angr.options.ZERO_FILL_UNCONSTRAINED_REGISTERS)
```

[RE8]PyCode

这题需要逆向一个用Python3.9编译的pyc:

 RE8_PyCode.cpython-39.pyc

传统的pyc反编译工具uncompyle6只支持到了Python3.8，在线反编译也只能出一部分代码：

uncompyle6 translates Python bytecode back into equivalent Python source code. It accepts bytecodes from Python version 1.0 to version 3.8, spanning over 24 years of Python releases. We include Dropbox's Python 2.5 bytecode and some PyPy bytecodes.

Google查一下，发现有一个叫做pycdc的工具，据说可以反编译Python3.9的字节码：

3 Answers

Active

Oldest

Votes

▲
7

Sadly enough, it's currently impossible. Decompile 3 has the latest pyc to py methods (decompilation), but it hasn't updated for python 3.9 yet as that update takes a very long time to create.

▼
✓

And it will most likely never happen for 3.9 (the developer of decompyle3 said that he is focusing more on his main job and that he doesn't have time to create this update as the 3.9 python update really changed the workflow, so it will be very hard and time-consuming).

🕒

So for now, the only solution is to wait, but if you want to speed things up, you can always sponsor the creator of decompile 3 (<https://github.com/sponsors/rocky>) (as he said that if you would get enough money to work more on this project, he will)

Edit:

I have recently found out that there is an alternative

I haven't used it myself, but its meant to decompile the compiled

python file (.pyc) back to humanly readable code (.py) For any python version!

You can check it out here: <https://github.com/zrax/pycdc>

这个工具既可以反编译出Python字节码也可以反编译出Python源码：

To run **pycdas**, the PYC Disassembler: `./pycdas [PATH TO PYC FILE]` The byte-code disassembly is printed to stdout.

To run **pycdc**, the PYC Decompiler: `./pycdc [PATH TO PYC FILE]` The decompiled Python source is printed to stdout. Any errors are printed to stderr.

用pycdc反编译出来的源码跟在线反编译的结果是一样的，有所残缺：

```
5676,
5520,
3504,
7550]
flag = input('Input:')
if len(flag) != 26:
    print('Wrong!')
    exit(1)
flag_arr = (lambda .0: [ ord(i) for i in .0 ])(flag)
flag_arr = (lambda .0: [ flag_arr[i] * 2 * i for i in .0 ])(range(26))
print('Success!')
print('Input is your flag!')
```

于是尝试用pycdas反编译出Python字节码，可以分析出加密流程。首先是将输入的每一个字符加上j：

```
64      GET_ITER
66      FOR_ITER          20 (to 88)
68      STORE_NAME       8: j
70      LOAD_NAME        6: flag_arr
72      LOAD_NAME        8: j
74      DUP_TOP_TWO
76      BINARY_SUBSCR
78      LOAD_NAME        8: j
80      INPLACE_ADD
82      ROT_THREE
84      STORE_SUBSCR
```

再和 26-j 异或：

```
100     LOAD_NAME        6: flag_arr
102     LOAD_NAME        8: j
104     DUP_TOP_TWO
106     BINARY_SUBSCR
108     LOAD_CONST        2: 26
110     LOAD_NAME        8: j
112     BINARY_SUBTRACT
114     INPLACE_XOR
```

最后调用一个列表生成表达式，对应pycdc反编译出来的那一部分代码：

```
CALL_FUNCTION 1
POP_TOP
LOAD_CONST    5: <CODE> <listcomp>
LOAD_CONST    6: '<listcomp>'
MAKE_FUNCTION 0
LOAD_NAME     2: flag
GET_ITER
CALL_FUNCTION 1
```

8	LOAD_GLOBAL	0: flag_arr
10	LOAD_FAST	1: i
12	BINARY_SUBSCR	
14	LOAD_CONST	0: 2
16	BINARY_MULTIPLY	
18	LOAD_FAST	1: i
20	BINARY_MULTIPLY	
22	LIST_APPEND	2
24	JUMP_ABSOLUTE	4
26	RETURN_VALUE	

根据加密流程写出exp:

```
enc = [0, 250, 444, 678, 880, 1260, 1788, 952, 2352, 1944, 1960, 1144, 2784,
2522, 2576, 3450, 3712, 4182, 5040, 5282, 4680, 3906, 5676, 5520, 3504, 7550]

flag = 's'

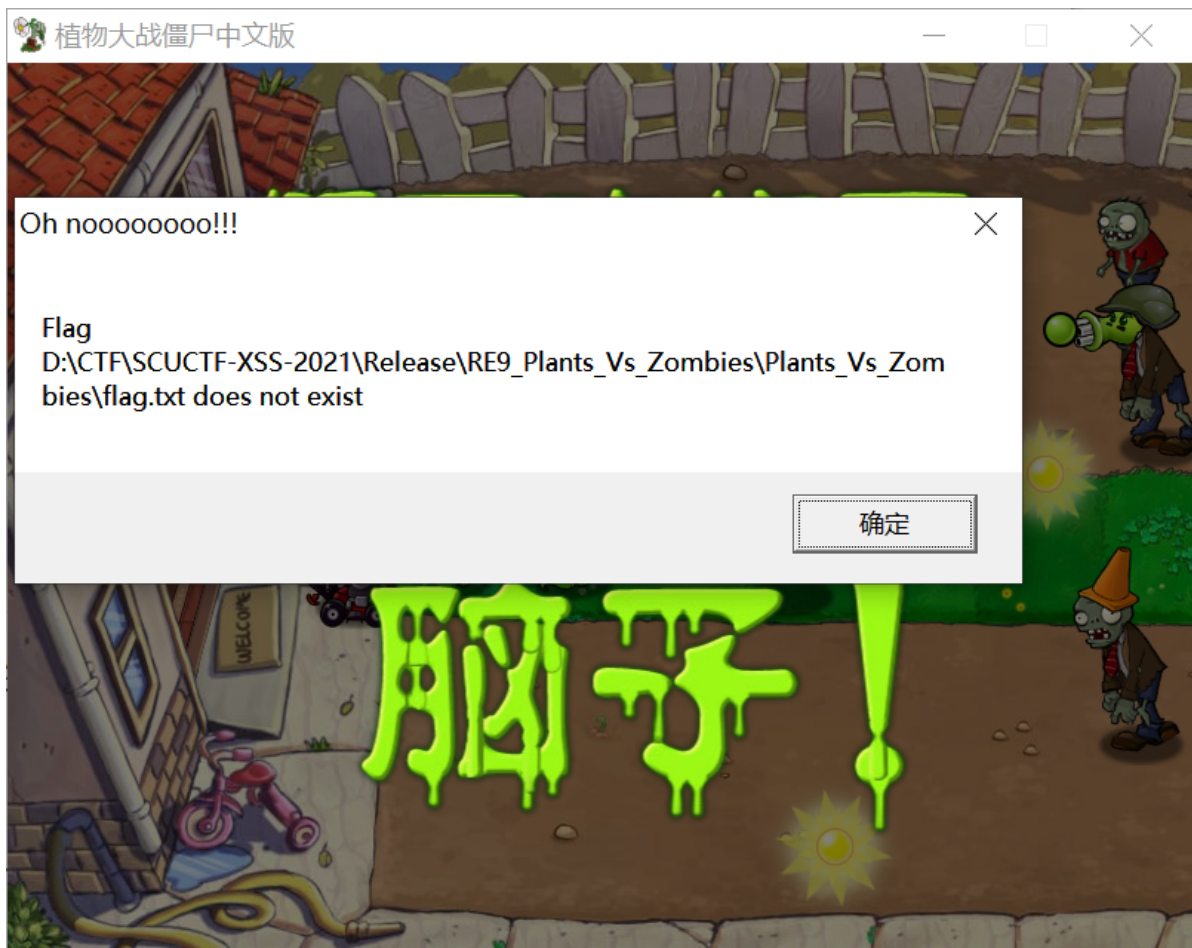
for i in range(1, 26):
    flag += chr(((enc[i] // i // 2) ^ (26 - i)) - i)

print(flag)
```

[RE9]Plants_Vs_Zombies

一道很有意思的题，这题改编自我之前用DLL注入的方法写的一个PVZ补丁：<https://github.com/bluesadi/PVZPatch>

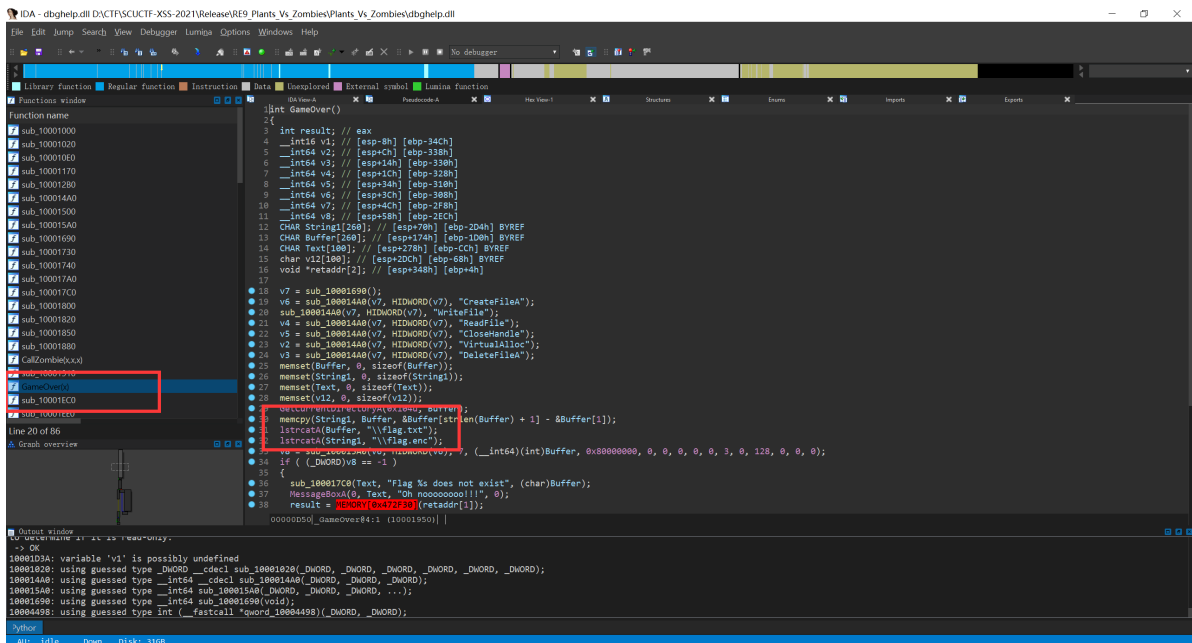
这题会在游戏失败加密PVZ目录下的flag.txt文件，加密后的文件保存为flag.enc，如果没有flag.txt文件也会提示。所以此题的关键是找出加密代码，并对flag.enc文件解密得到flag：



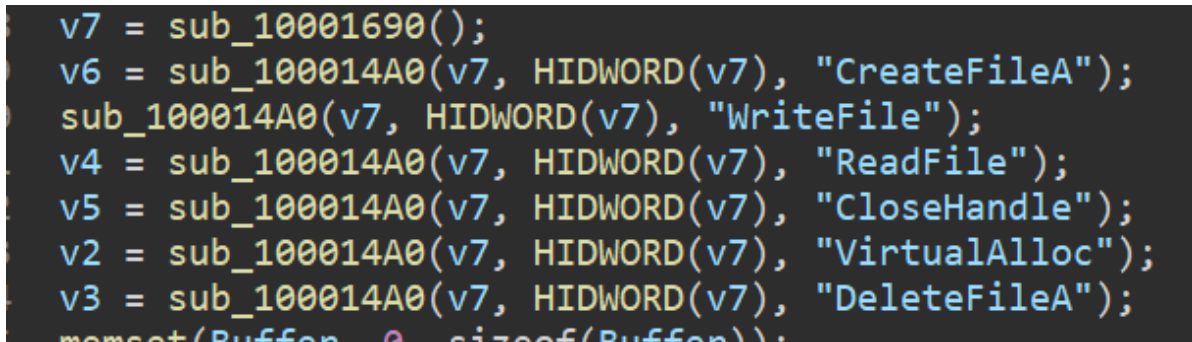
首先将PlantsVsZombies.exe拖入IDA，可以发现start函数的开头就调用了LoadLibrary函数，加载DBGHELP.DLL，start函数在正常情况下不可能出现这样的内容，说明DBGHELP.DLL是有问题的：

```
7 start      proc near      ; CODE XREF: start-18F↑j
7              ; start-171↑j
7
7 ; FUNCTION CHUNK AT .text:0061EA12 SIZE 0000018D BYTES
7 ; FUNCTION CHUNK AT .text:0061EB6D SIZE 0000000C BYTES
7
7      push    offset aDbghelpDll ; "DBGHELP.DLL"
7
7C loc_61EBEC:      ; CODE XREF: start-10↑j
7C      call    ds:LoadLibraryA
7C      call    sub_62E8F1
7      jmp     loc_61EA12
7 start      endp
7
```

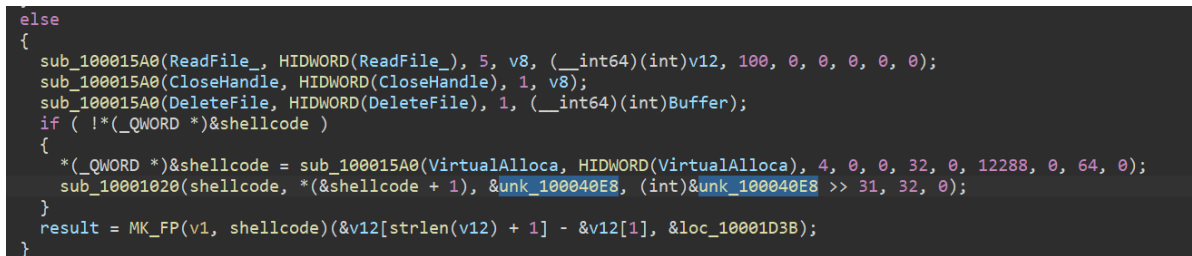
分析DBGHELP.DLL，发现有个叫做GameOver的导出函数，并且该函数中也出现了flag.txt这样的字符串，基本可以断定这个函数就是加密函数：



这里用到了天堂之门技术，在32位环境下获取64位API函数的句柄，并调用。有关天堂之门技术，我在看雪论坛上发了个[帖子](#)：



主要加密过程如下，先读取flag.txt文件的内容，然后删除文件；加载一段shellcode到内存中，并执行：



查看shellcode的内容，这段shellcode非常简单，本来是为了降低题目难度，没想到有同学直接用密码学的手段推出了加密流程（非预期解）：


```

.data:100040E8 ; -----
.data:100040E8
.data:100040E8 loc_100040E8: ; CODE XREF: .data:10004101↑j
.data:100040E8 ; DATA XREF: GameOver(x)+356↑o
.data:100040E8      dec     eax
.data:100040E9      dec     ecx
.data:100040EB      mov     al, [esi+ecx]
.data:100040ED      ror     al, 3
.data:100040F1      add     al, cl
.data:100040F3      dec     eax
.data:100040F4      cmp     ecx, 6
.data:100040F7      jnz     short loc_100040FB
.data:100040F9      xor     al, 64h
.data:100040FB loc_100040FB: ; CODE XREF: .data:100040F7↑j
.data:100040FB      mov     [esi+ecx], al
.data:100040FE      dec     eax
.data:100040FF      test    ecx, ecx
.data:10004101      jnz     short loc_100040E8
.data:10004103      push    23h ; '#'
.data:10004105      push    edx
.data:10004106      dec     eax
.data:10004107      retf

```

根据加密流程和密文写出exp:

```

#include <stdio>
#include <stdint>

// scuctf{knockin_on_heavens_gate}
//
\x6e\x6d\xb0\x6f\x92\xd1\x11\x74\xd5\xf6\x76\x78\x39\xda\xf9xfc\xdd\xfc\x1f\xbf
\x40\xe3\xc2\xe4\x86\x04\x06\x47\xaa\xc9\xcd
int main(){
    uint8_t flag[] =
"\x6e\x6d\xb0\x6f\x92\xd1\x11\x74\xd5\xf6\x76\x78\x39\xda\xf9xfc\xdd\xfc\x1f\xbf
f\x40\xe3\xc2\xe4\x86\x04\x06\x47\xaa\xc9\xcd";
    for(int i = 0; i < 31; i++){
        if(i == 6){
            flag[i] ^= 0x64;
        }
        flag[i] -= i;
        flag[i] = (flag[i] << 3) | (flag[i] >> 5);
    }
    printf("%s\n", flag);
}

```

[HappyRE]ezjava

hgg出的一道Java题，先用jd-gui打开看看，关键的CrackLicense类完全被混淆了：

Cheat Engine 7.2

文件(F) 编辑(E) 表单 D3D Java 帮助(H)

000043F0-java.exe

设置

结果: 2

地址	当前值	先前值
89F91D1D	scuctf{	
89F91D5D	scuctf{	

新的扫描 再次扫描 撤销扫描

数值:
scuctf{

扫描类型 搜索字符串 代码页 UTF-16 区分大小写 禁止随机 开启变速精灵

数值类型 字符串

内存扫描选项
All
起始 0000000000000000
停止 00007fffffffffffffff
☒ 可写 ☐ 可执行
☐ 写时拷贝
☒ 快速扫描 1 ☒ 对齐 ☐ 最后位数
☐ 扫描时暂停游戏

查看内存

手动添加地址

激活	描述	地址	类型	数值
<input type="checkbox"/>	无描述	89F91D1D	字符串[7]	scuctf{

高级选项

附加注释

内存浏览器

文件 搜索 视图 调试 工具 内核工具

java.exe+1420

地址	字节	操作码	注释
java.exe+1420	48 83 EC 28	sub rsp,28	40
java.exe+1424	E8 BF020000	call java.exe+16E8	
java.exe+1429	48 83 C4 28	add rsp,28	40
java.exe+142D	E9 72FEFFFF	jmp java.exe+12A4	
java.exe+1432	CC	int 3	
java.exe+1433	CC	int 3	
java.exe+1434	48 83 EC 28	sub rsp,28	40
java.exe+1438	E8 AF070000	call java.exe+1BEC	
java.exe+143D	85 C0	test eax,eax	
java.exe+143F	74 21	je java.exe+1462	
java.exe+1441	66 48 8B 04 2	mov eax,esi	40
subtract			

保护: 读/写 AllocationBase=82600000 基址=89F91000 长度=46F000

地址	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	DEF0123456789ABC
89F91D1D	73	63	75	63	74	66	7B	6A	76	61	76	41	67	31	65	6E	scuctf{jjvavAglen
89F91D2D	74	31	73	56	65	40	72	31	7A	79	7D	01	00	00	00	00	tlsv@rlzy}.....
89F91D3D	00	00	00	04	01	00	20	30	00	00	00	43	72	61	63	6B0...Crack
89F91D4D	65	64	20	73	75	63	63	65	73	73	66	75	6C	6C	79	3A	ed successfully:
89F91D5D	73	63	75	63	74	66	7B	6A	76	61	76	41	67	31	65	6E	scuctf{jjvavAglen
89F91D6D	74	31	73	56	65	40	72	31	7A	79	7D	05	00	00	00	00	tlsv@rlzy}.....
89F91D7D	00	00	00	51	CF	00	20	00	00	00	00	05	00	00	00	00	...Q
89F91D8D	00	00	00	AD	1D	00	20	08	00	00	00	00	00	00	00	A0
89F91D9D	1D	F9	89	01	00	00	00	00	00	00	00	04	01	00	20	10
89F91DAD	00	00	00	54	68	72	65	61	64	2D	30	00	00	00	00	00	...Thread-0.....
89F91DBD	00	00	00	05	00	00	00	00	00	00	00	01	03	00	20	D8
89F91DCD	1D	F9	89	00	00	00	00	00	00	00	00	01	00	00	00	00
89F91DDD	00	00	00	04	01	00	20	07	00	00	00	54	68	72	65	61Threa
89F91DED	64	2D	00	05	00	00	00	00	00	00	00	01	03	00	20	08	d-.....
89F91DFD	1E	F9	89	00	00	00	00	00	00	00	00	01	00	00	00	00
89F91E0D	00	00	00	04	01	00	20	08	00	00	00	54	68	72	65	61Threa
89F91E1D	64	2D	30	05	00	00	00	00	00	00	00	00	02	00	20	00	d-0.....
89F91E2D	00	00	00	01	00	00	00	00	00	00	00	F7	9B	00	20	01
89F91E3D	00	00	00	D8	07	13	8A	00	00	00	00	05	00	00	00	00