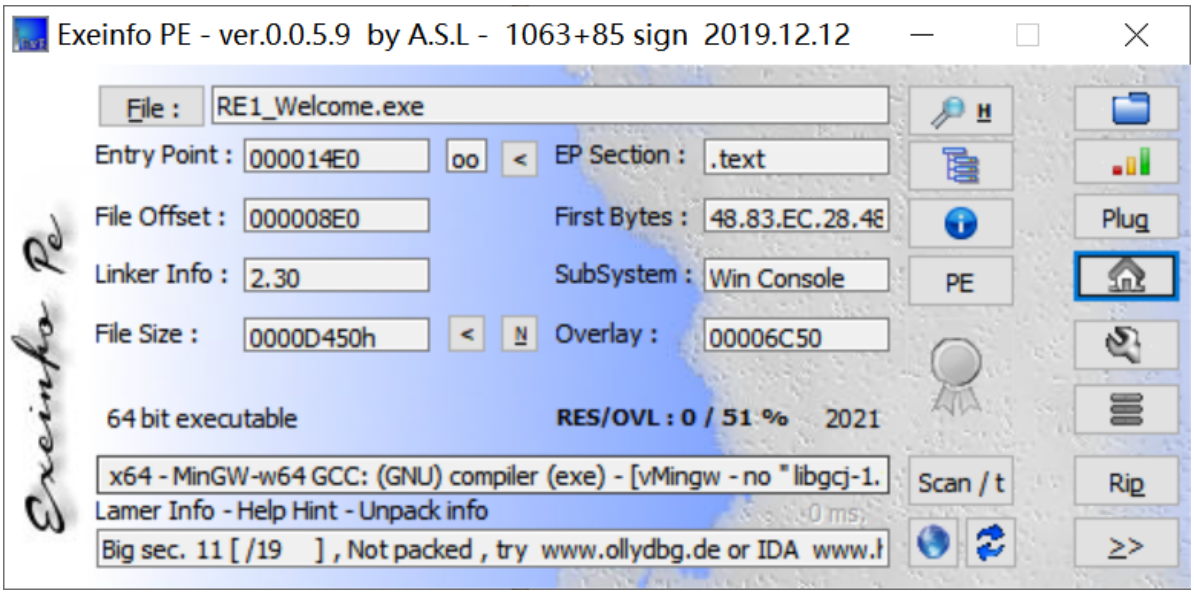
2021SCUCTF新生赛-逆向题WriteUp

时间：2021/11/19-2021/11/23

[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 }
```

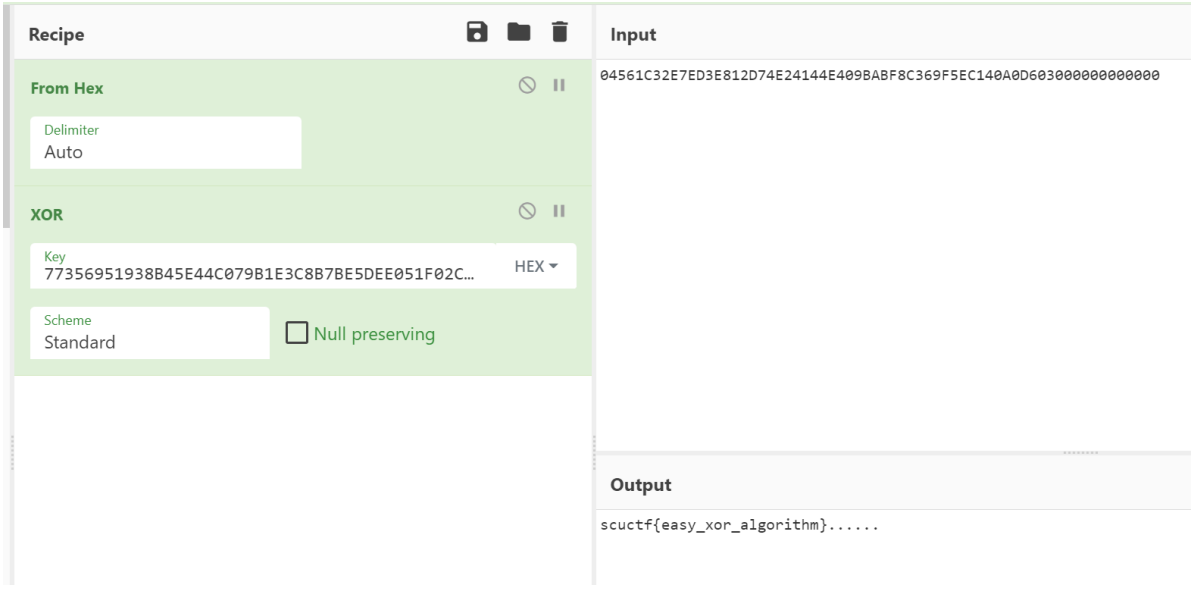
```
D:\CTF\SCUCTF-XSS-2021\Review\Release\1>RE1_Welcome.exe
Input a correct number to rescue the Fuhrer.
1919810
Correct.Flag to enter Fuhrerbunker:
scuctf{w3lcOm3_to_the_world_of_r3!}
```

[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    v3 += 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：



[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：

```
else if ( v27 == 2 )
{
  nptr = (char *)v26[1];
  v24 = strtol(nptr, 0LL, 10);
  v3 = ~((unsigned __int16)~(*(_DWORD *)(&v20 - 2) & ~((*((_DWORD *)&v20 - 4) | 0xE15) & (~(unsigned __int16)*((_DWORD *)&v20 - 4) | 0xF1EA))) & (unsig
  v4 = *((_DWORD *)&v20 - 4);
  v5 = ~(~(~(~v4 & ~((~v23 | 0x2ABDF1EA) & (v23 | 0xD5420E15))) & ~(v4 & (~v23 | 0x2ABDF1EA) & (v23 | 0xD5420E15)) & ~(~(~v4 & v3) & ~(v4 & ~v3))) & ~(
  v6 = v5 & (v5 ^ ~(v4 & ~((v4 | 0xD5420E15) & (~v4 | 0x2ABDF1EA))) & ~(~v4 & (v4 | 0xD5420E15) & (~v4 | 0x2ABDF1EA))) | (~v4 ^ (~v4 & v5 | v4 & ~(~v4
  v7 = ~((v4 | 0xD5420DCA) & (~v4 | 0x2ABDF235) & ~(~(~v4 & v6) & ~(v4 & ~v6))) & ~(~((v4 | 0xD5420DCA) & (~v4 | 0x2ABDF235)) & ~(~v4 & v6) & ~(v4 & ~v6
  v8 = ~v7 & (~v7 ^ ~(v4 & ~((v4 | 0xD5420E15) & (~v4 | 0x2ABDF1EA))) & ~(~v4 & (v4 | 0xD5420E15) & (~v4 | 0x2ABDF1EA))) | (~v4 ^ (~v4 & ~v7 | v4 & v7)
  v9 = ~(~(~v4 & v8) & ~(v4 & ~v8));
  v22 = ~(~(~(~v4 & ~((~v23 | 0x2ABDF1EA) & (v23 | 0xD5420E15))) & ~(v4 & (~v23 | 0x2ABDF1EA) & (v23 | 0xD5420E15)) & v9) & ~(~(~v4 & ~((~v23 | 0x2ABD
  if ( v24 == v22 )
  {
    puts("Correct! Here is your flag:");
    for ( i = 0; i < 34; ++i )
    {
      v10 = *((_DWORD *)&v20 - 4);
      v11 = enc[i] & (enc[i] ^ ~(v10 & ~((v10 | 0x15) & (~(_BYTE)v10 | 0xEA))) & ~(~(_BYTE)v10 & (v10 | 0x15) & (~(_BYTE)v10 | 0xEA))) | (~(_BYTE)v10 ^
      v12 = ~(~(_BYTE)v10 & v11) & ~(v10 & ~v11);
      enc[i] = ~(~((v10 | 0x57) & (~(_BYTE)v10 | 0xA8) & ~v12) & ~(~((v10 | 0x57) & (~(_BYTE)v10 | 0xA8)) & v12));
      v13 = enc[i];
      v14 = *((_DWORD *)&v20 - 4);
      v15 = v13 & (v13 ^ ~(v14 & ~((v14 | 0x15) & (~(_BYTE)v14 | 0xEA))) & ~(~(_BYTE)v14 & (v14 | 0x15) & (~(_BYTE)v14 | 0xEA))) | (~(_BYTE)v14 ^ (~(_B
      enc[i] = ~(~(~(~(_BYTE)v14 & ~((~key[i] | 0xEA) & (key[i] | 0x15))) & ~(v14 & (~key[i] | 0xEA) & (key[i] | 0x15)) & ~(~(~(_BYTE)v14 & v15) & ~(v1
      v16 = *((_DWORD *)&v20 - 4);
      v17 = enc[i] & (enc[i] ^ ~(v16 & ~((v16 | 0x15) & (~(_BYTE)v16 | 0xEA))) & ~(~(_BYTE)v16 & (v16 | 0x15) & (~(_BYTE)v16 | 0xEA))) | (~(_BYTE)v16 ^
      v18 = ~(~(~(_BYTE)v16 & v17) & ~(v16 & ~v17));
      enc[i] = ~(~(~(~(_BYTE)v16 & 0xA8) & ~(v16 & 0x57) & v18) & ~(~(~(~(_BYTE)v16 & 0xA8) & ~(v16 & 0x57)) & ~v18));
      putchar(enc[i]);
    }
    putchar('\n');
  }
```

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：

```
root@ubuntu:~# cd re/
root@ubuntu:~/re# ls
10  11  12  access.log  capstone  chacha20  dbgsrv  example-1_recovered  exp.py  gghdl  xiangyunbei
root@ubuntu:~/re# cd dbgsrv/
root@ubuntu:~/re/dbgsrv# ls
21111961978038e80fb  admin_panel  babyre  babyre.orig  csb  linux_server  linux_server64  ps  sob
root@ubuntu:~/re/dbgsrv# ./linux_server64
IDA Linux 64-bit remote debug server(ST) v7.5.26. Hex-Rays (c) 2004-2020
Listening on 0.0.0.0:23946...
```

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：

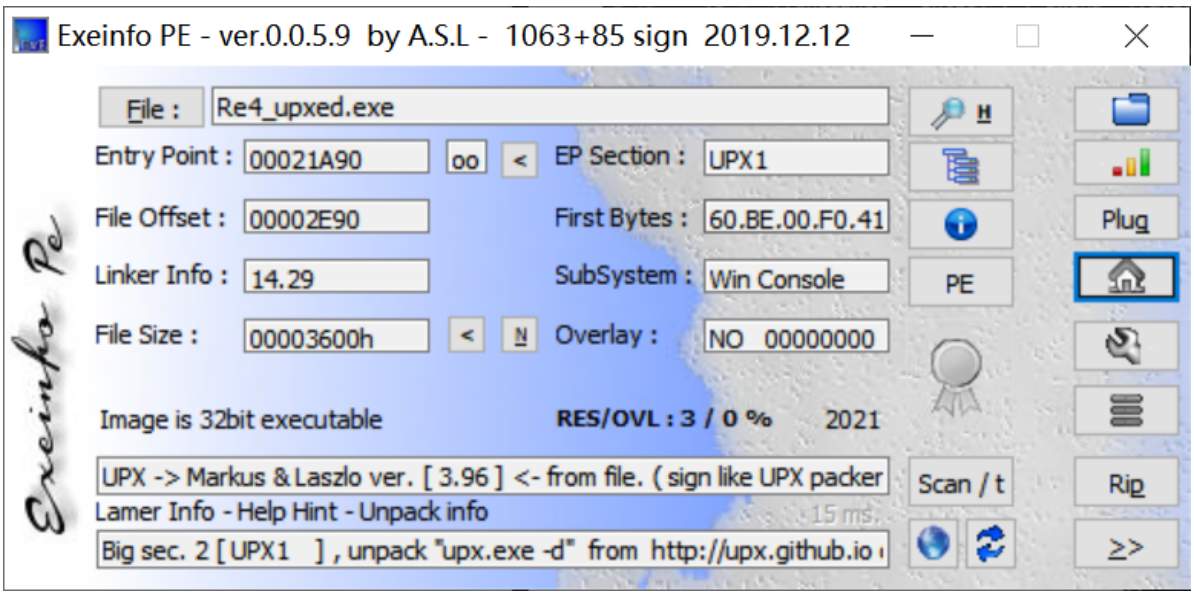
```
  35  }
  36  else if ( v27 == 2 )
  37  {
  38    nptr = (char *)v26[1];
  39    v24 = strtol(nptr, 0LL, 10);
  40    v3 = ~((unsigned __int16)~(*(_DWORD *)(&v20 - 2) & ~((*((_DWORD *)&v20 - 4) | 0xE15) & (~(unsigned __int16)*((_DWORD *)&v20 - 4) | 0xF1EA))) & (unsign
  41    v4 = *((_DWORD *)&v20 - 4);
  42    v5 = ~(~(~(~v4 & ~((~v23 | 0x2ABDF1EA) & (v23 | 0xD5420E15))) & ~(v4 & (~v23 | 0x2ABDF1EA) & (v23 | 0xD5420E15)) & ~(~(~v4 & v3) & ~(v4 & ~v3))) & ~(~
  43    v6 = v5 & (v5 ^ ~(v4 & ~((v4 | 0xD5420E15) & (~v4 | 0x2ABDF1EA))) & ~(~v4 & (v4 | 0xD5420E15) & (~v4 | 0x2ABDF1EA))) | (~v4 ^ (~v4 & v5 | v4 & ~(~v4
  44    v7 = ~((v4 | 0xD5420DCA) & (~v4 | 0x2ABDF235) & ~(~(~v4 & v6) & ~(v4 & ~v6))) & ~(~((v4 | 0xD5420DCA) & (~v4 | 0x2ABDF235)) & ~(~v4 & v6) & ~(v4 & ~v6
  45    v8 = ~v7 & (~v7 ^ ~(v4 & ~((v4 | 0xD5420E15) & (~v4 | 0x2ABDF1EA))) & ~(~v4 & (v4 | 0xD5420E15) & (~v4 | 0x2ABDF1EA))) | (~v4 ^ (~v4 & ~v7 | v4 & v7))
  46    v9 = ~(~(~v4 & v8) & ~(v4 & ~v8));
  47    v22 = ~(~(~(~v4 & ~((~v23 | 0x2ABDF1EA) & (v23 | 0xD5420E15))) & ~(v4 & (~v23 | 0x2ABDF1EA) & (v23 | 0xD5420E15)) & v9) & ~(~(~v4 & ~((~v23 | 0x2ABD
  48    if ( v24 == v22 )
  49    {
  50      puts("Correct! Here is your flag:");
  51      for ( i = 0; i < 34; ++i )
  52      {
  53        v10 = *((_DWORD *)&v20 - 4);
  54        v11 = enc[i] & (enc[i] ^ ~(v10 & ~((v10 | 0x15) & (~(_BYTE)v10 | 0xEA))) & ~(~(_BYTE)v10 & (v10 | 0x15) & (~(_BYTE)v10 | 0xEA))) | (~(_BYTE)v10 ^
  55        v12 = ~(~(_BYTE)v10 & v11) & ~(v10 & ~v11);
  56        enc[i] = ~(~((v10 | 0x57) & (~(_BYTE)v10 | 0xA8) & ~v12) & ~(~((v10 | 0x57) & (~(_BYTE)v10 | 0xA8)) & v12));
  57        v13 = enc[i];
000008B9 main:48 (4008B9)
```

将输入改为1453即可得到正确的flag：

```
Correct! Here is your flag:
scuctf{just_ask_you_to_have_Linux}
```

[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.
```

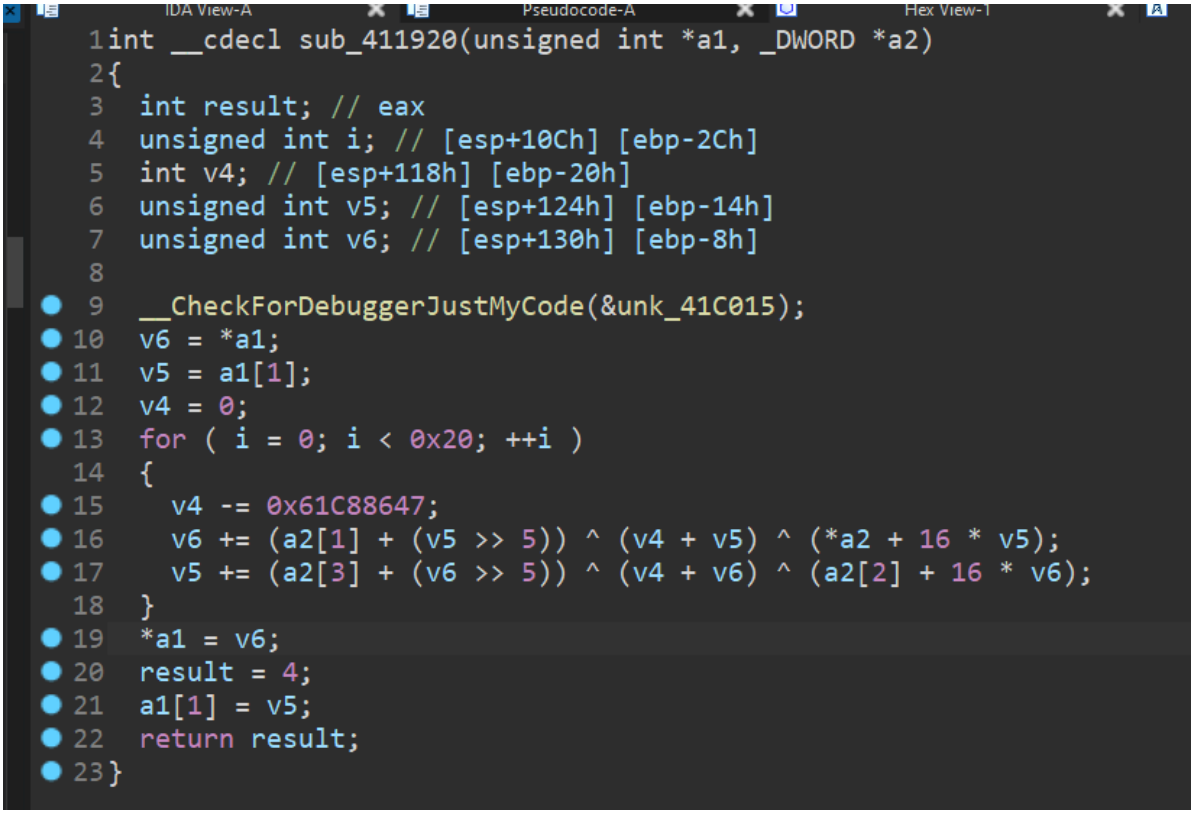
找到关键代码，应该是把输入加密之后与密文进行比较：

```
__CheckForDebuggerJustMyCode(&unk_41C015);
qmemcpy(v8, "THIS_IS_THE_KEYa", 16);
v6 = 0x5618435D;
v7[0] = -1608884855;
v7[1] = 1372647817;
v7[2] = 1872257453;
v7[3] = 835943406;
v7[4] = -1137280198;
puts(Buffer);
puts(off_41A004);
puts(off_41A008);
puts(off_41A00C);
puts(off_41A010);
j_memset(Str, 0, 0x64u);
puts("Input your flag:");
sub_41127B("%100s", (char)Str);
if ( j_strlen((const char *)Str) != 24 )
{
  puts("Wrong!");
  exit(0);
}
for ( i = 0; i < 3; ++i )
{
  sub_41100A(&Str[i], v8);
  if ( LODWORD(Str[i]) != *(&v6 + 2 * i) || HIDWORD(Str[i]) != v7[2 * i] )
  {
    puts("Wrong!");
    exit(0);
  }
}
puts("Correct! flag is your input!");
return 0;
}
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：

```
QWORD v6[4]; // [esp+154h] [ebp-38h]
int v7[5]; // [esp+174h] [ebp-18h] BYREF

__CheckForDebuggerJustMyCode(&unk_41C015);
qmemcpy(v7, "THIS_IS_THE_KEYa", 16);
LODWORD(v6[0]) = 0x5618435D;
HIDWORD(v6[0]) = 0xA01A5D89;
LODWORD(v6[1]) = 0x51D0F189;
HIDWORD(v6[1]) = 0x6F9861AD;
LODWORD(v6[2]) = 0x31D37BEE;
HIDWORD(v6[2]) = 0xBC367B3A;
puts(Buffer);
puts(off_41A004);
puts(off_41A008);
puts(off_41A00C);
puts(off_41A010);
j_memset(Str, 0, 0x64u);
puts("Input your flag:");
sub_41127B("%100s", (char)Str);
if ( j_strlen((const char *)Str) != 24 )
{
  puts("Wrong!");
  exit(0);
}
for ( i = 0; i < 3; ++i )
{
  sub_41100A((int)&Str[i], (int)v7);
  if ( LODWORD(Str[i]) != LODWORD(v6[i]) || HIDWORD(Str[i]) != HIDWORD(v6[i]) )
  {
```

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



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 <cstdio>

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}
```

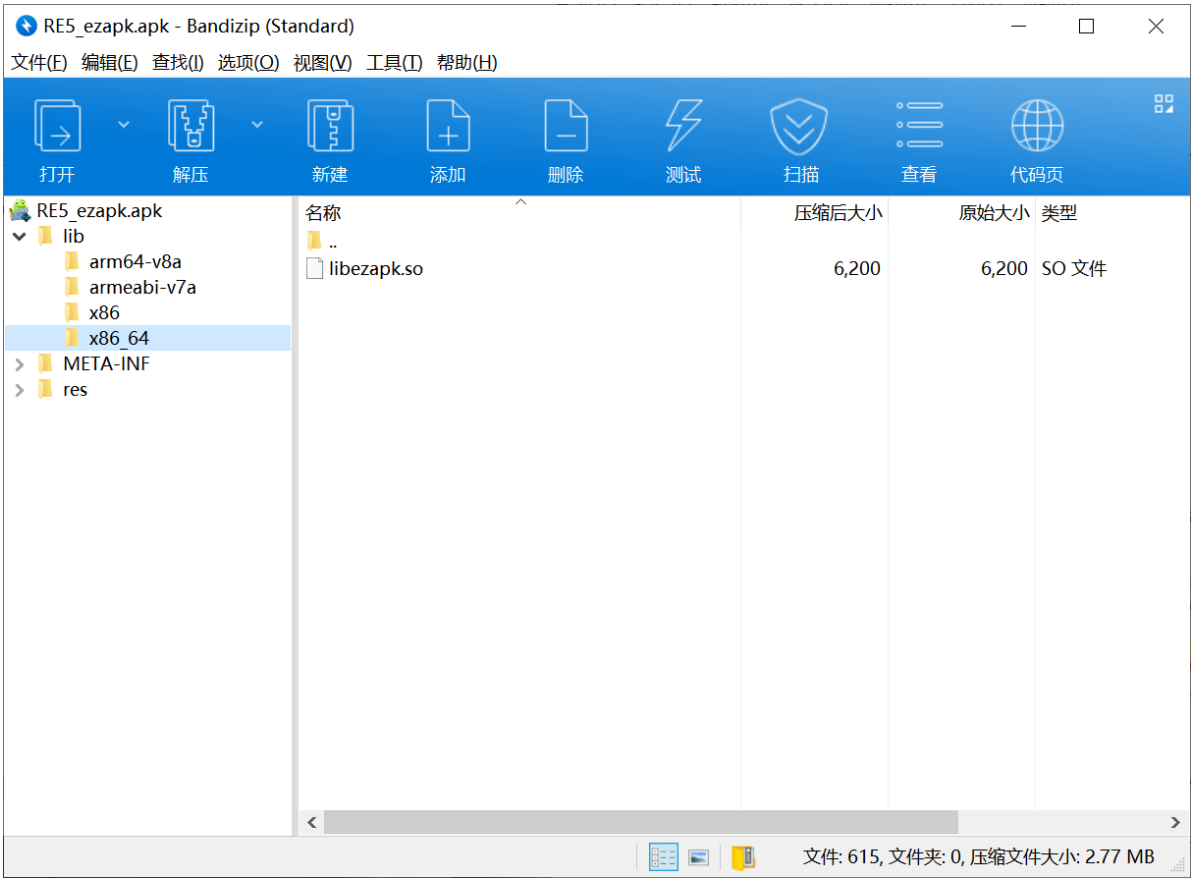
当然也可以手动脱壳，CTF-Wiki的脱壳教程：https://ctf-wiki.org/reverse/windows/unpack/esp/

[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一个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：

```
    static {
        System.loadLibrary("ezapk");
    }

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

    /* renamed from: lambda$onCreate$0$cn-bluesadi-ezapk-MainActivity  reason: not valid java name */
    public /* synthetic */ void m2lambda$onCreate$0$cnbluesadiezapkMainActivity(View view) {
        String obj = this.password.getText().toString();
        if (check(obj)) {
            this.result.setText(String.format("Success!\nscuctf{%s}", new Object[]{MD5(obj)}));
            this.result.setTextColor(getColor(R.color.green));
            return;
        }
        this.result.setText("Wrong password.");
        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）：

```
1bool __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, a3, &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 < 2;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 <cstdio>

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\x9
b\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 && !strncmp(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：

```
1int __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：

```
1void __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
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
0000002C
```

还原结构体后的代码如图所示，还是比较清晰的：

```
    case 0xFu:
      rsh((int)&a1->regs[a1->PC[1]], (int)&a1->regs[a1->PC[2]]);
      a1->PC += 3;
      break;
    case 0x10u:
      rol((int)&a1->regs[a1->PC[1]], (int)&a1->regs[a1->PC[2]]);
      a1->PC += 3;
      break;
    case 0x11u:
      ror((int)&a1->regs[a1->PC[1]], (int)&a1->regs[a1->PC[2]]);
      a1->PC += 3;
      break;
    case 0x12u:                                 // mov_r2r
      a1->regs[a1->PC[1]] = a1->regs[a1->PC[2]];
      a1->PC += 3;
      break;
    case 0x13u:
      a1->regs[a1->PC[1]] = _byteswap_ulong(*(_DWORD *)(a1->PC + 2));// mov_i2r
      a1->PC += 6;
      break;
    case 0x14u:                                 // push reg
      *a1->sp_ = a1->regs[a1->PC[1]];
      a1->sp_ += 4;
      a1->PC += 2;
      break;
    case 0x15u:                                 // push imm
      *a1->sp_ = _byteswap_ulong(*(_DWORD *)(a1->PC + 1));
      a1->sp_ += 4;
      a1->PC += 5;
      break;
    case 0x16u:                                 // pop reg
      v5 = a1->PC[1];
```

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 <cstdio>
#include <cstdint>
#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,
```

```
    0x57, 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, 0x63, 0x1A,
    0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x6F, 0x1A, 0x00, 0x1C,
0x01, 0x13, 0x00, 0x00,
    0x00, 0x00, 0x6D, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00,
0x65, 0x1A, 0x00, 0x1C,
    0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x20, 0x1A, 0x00, 0x1C, 0x01, 0x13,
0x00, 0x00, 0x00, 0x00,
    0x74, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x6F, 0x1A,
0x00, 0x1C, 0x01, 0x13,
    0x00, 0x00, 0x00, 0x00, 0x20, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00,
0x00, 0x00, 0x73, 0x1A,
    0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x63, 0x1A, 0x00, 0x1C,
0x01, 0x13, 0x00, 0x00,
    0x00, 0x00, 0x75, 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,
    0x66, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x20, 0x1A,
0x00, 0x1C, 0x01, 0x13,
    0x00, 0x00, 0x00, 0x00, 0x63, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00,
0x00, 0x00, 0x68, 0x1A,
    0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x61, 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, 0x65, 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, 0x65, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00,
0x00, 0x00, 0x21, 0x1A,
    0x00, 0x1C, 0x01, 0x1D, 0x23, 0x18, 0x23, 0x13, 0x00, 0x00, 0x00, 0x00,
0x0A, 0x1A, 0x00, 0x1C,
    0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x49, 0x1A, 0x00, 0x1C, 0x01, 0x13,
0x00, 0x00, 0x00, 0x00,
    0x6E, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x70, 0x1A,
0x00, 0x1C, 0x01, 0x13,
    0x00, 0x00, 0x00, 0x00, 0x75, 0x1A, 0x00, 0x1C, 0x01, 0x13, 0x00, 0x00,
0x00, 0x00, 0x74, 0x1A,
    0x00, 0x1C, 0x01, 0x13, 0x00, 0x00, 0x00, 0x00, 0x3A, 0x1A, 0x00, 0x1C,
0x01, 0x1D, 0x07, 0x18,
    0x07, 0x1B, 0x17, 0x1E, 0x00, 0x1F, 0x13, 0x01, 0x00, 0x00, 0x00, 0x1A,
0x20, 0x00, 0x01, 0x22,
    0x00, 0x03, 0x19, 0x00, 0x1C, 0x01, 0x19, 0x01, 0x1C, 0x01, 0x13, 0x02,
0x00, 0x00, 0x00, 0x73,
    0x20, 0x00, 0x02, 0x22, 0x00, 0x03, 0x13, 0x03, 0x00, 0x00, 0x00, 0x63,
0x20, 0x01, 0x03, 0x22,
    0x00, 0x03, 0x19, 0x00, 0x1C, 0x01, 0x19, 0x01, 0x1C, 0x01, 0x13, 0x02,
0x00, 0x00, 0x00, 0x75,
    0x20, 0x00, 0x02, 0x22, 0x00, 0x03, 0x13, 0x03, 0x00, 0x00, 0x00, 0x63,
0x20, 0x01, 0x03, 0x22,
    0x00, 0x03, 0x19, 0x00, 0x1C, 0x01, 0x19, 0x01, 0x1C, 0x01, 0x13, 0x02,
0x00, 0x00, 0x00, 0x74,
    0x20, 0x00, 0x02, 0x22, 0x00, 0x03, 0x13, 0x03, 0x00, 0x00, 0x00, 0x66,
0x20, 0x01, 0x03, 0x22,
```

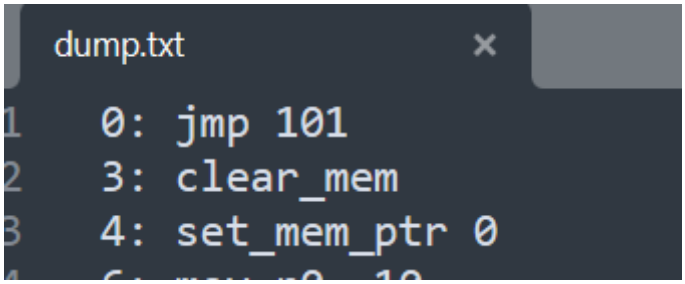
```
    0x00, 0x03, 0x19, 0x00, 0x1C, 0x01, 0x13, 0x02, 0x00, 0x00, 0x00, 0x7B,
0x20, 0x00, 0x02, 0x22,
    0x00, 0x03, 0x1E, 0x00, 0x1C, 0x19, 0x19, 0x00, 0x13, 0x02, 0x00, 0x00,
0x00, 0x7D, 0x20, 0x00,
    0x02, 0x22, 0x00, 0x03, 0x1E, 0x00, 0x1C, 0x07, 0x15, 0x00, 0x00, 0x00,
0x12, 0x15, 0x00, 0x00,
    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);
```

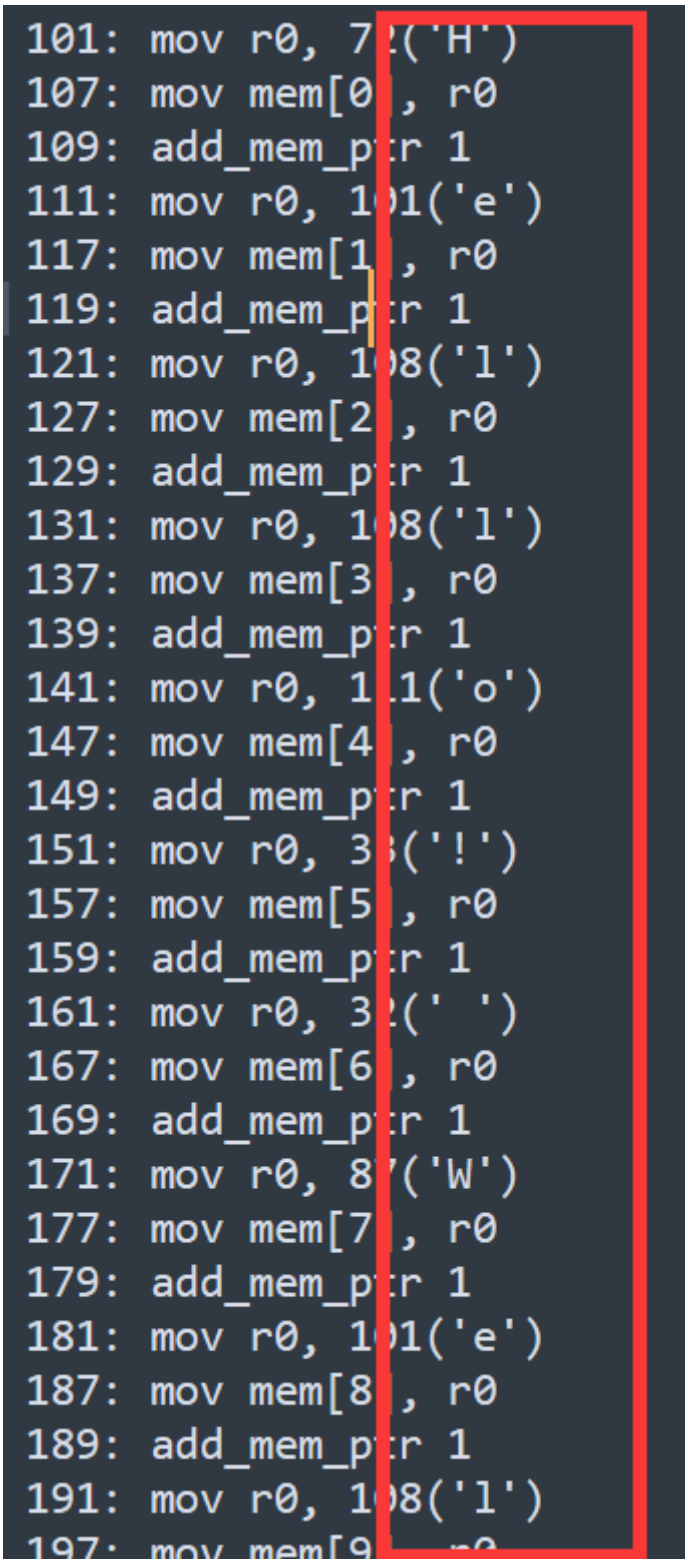
```
            if(imm >= 32 && imm <= 126){
                printf("mov r%d, %d(\'%c\')\n", code[PC + 1], imm, imm);
            }else{
                printf("mov r%d, %d\n", code[PC + 1], imm);
            }
            PC += 6;
            break;
        case 0x14:
            printf("push r%d\n", 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;
        }
    }
}
```

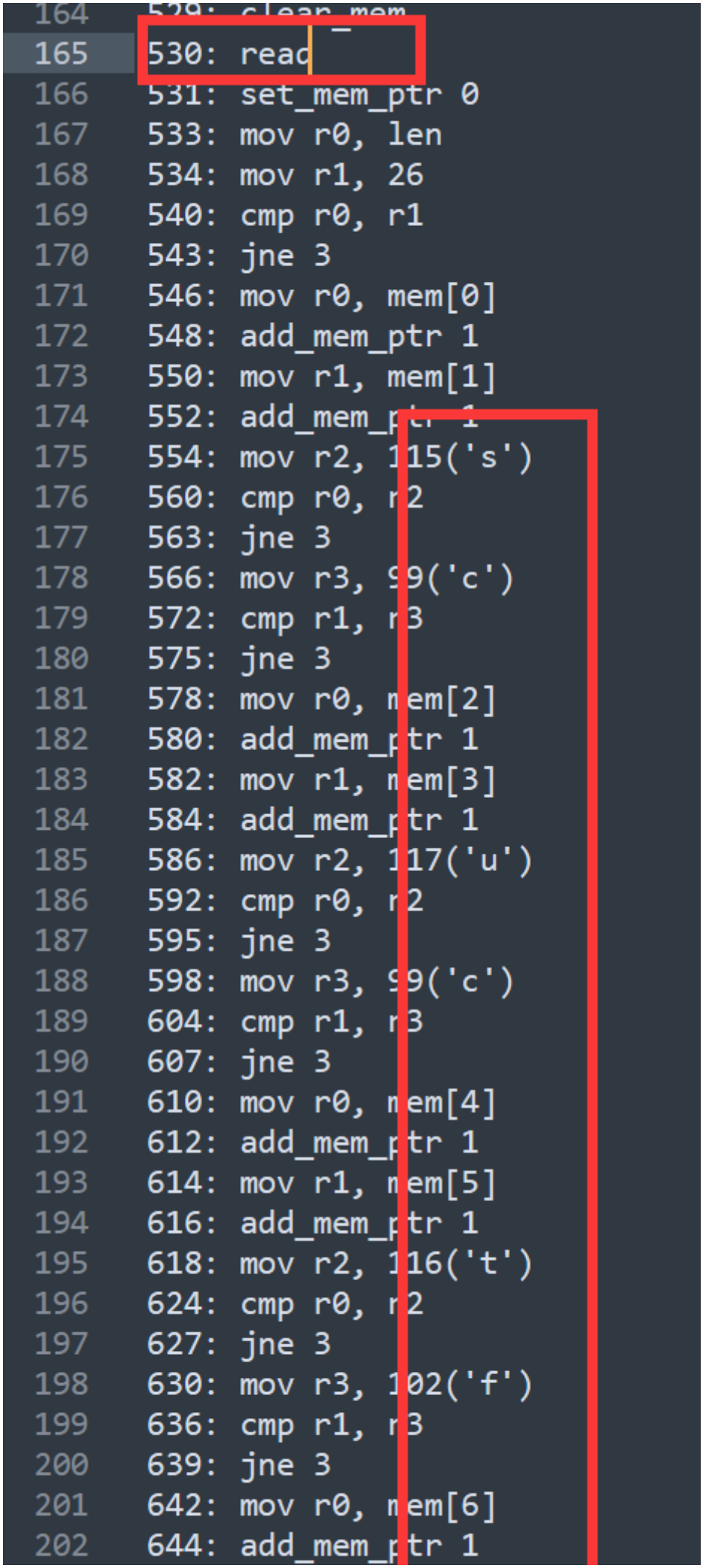
分析dump下来的VM指令，首先直接跳转到101位置处的指令：



看样子应该是一些输出信息，直接略过继续往下分析：

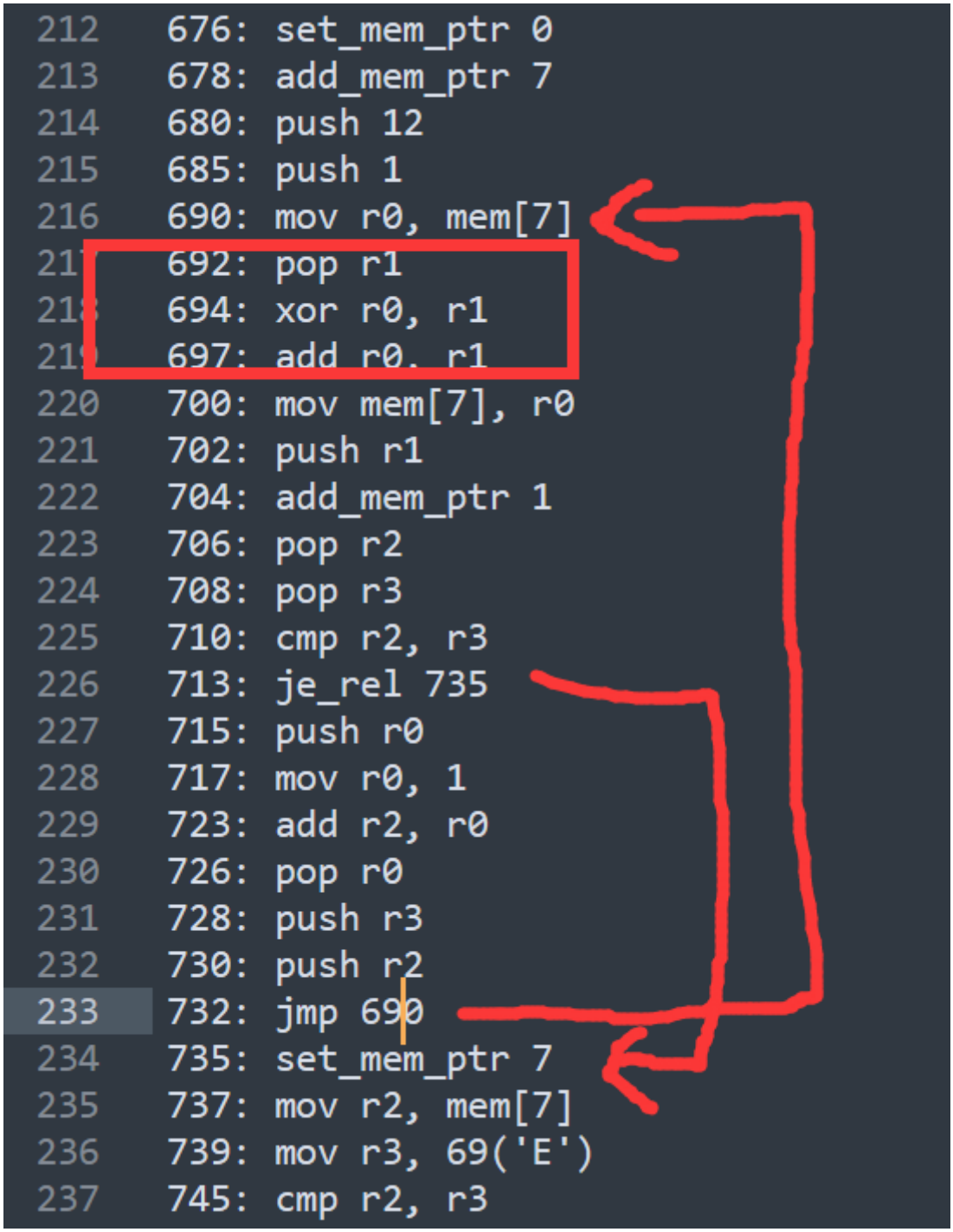


读取输入，判断输入的长度是不是26，以及格式是不是`scuctf{xxx}`：

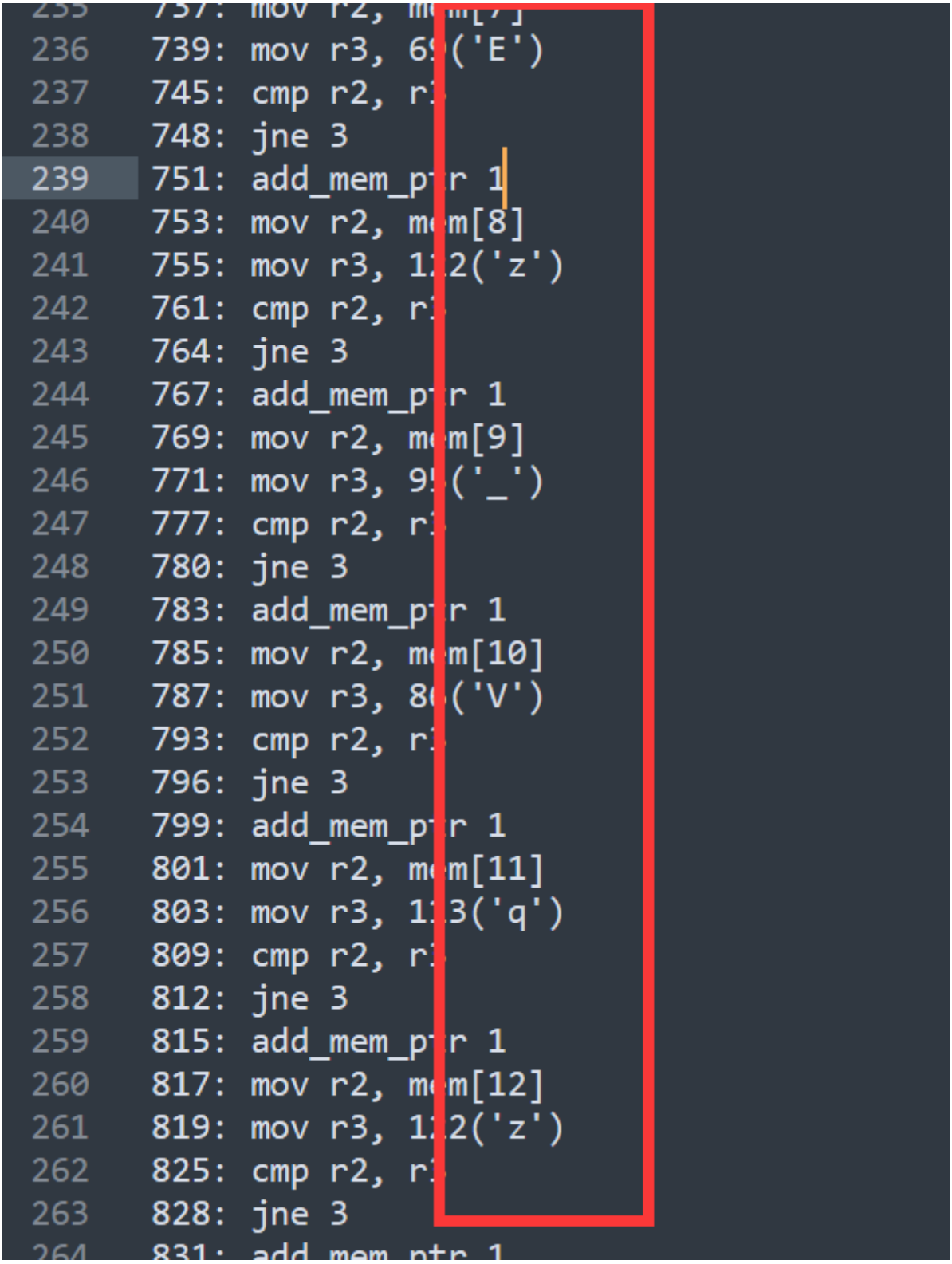

530: read
531: set_mem_ptr 0
533: mov r0, len
534: mov r1, 26
540: cmp r0, r1
543: jne 3
546: mov r0, mem[0]
548: add_mem_ptr 1
550: mov r1, mem[1]
554: mov r2, 115('s')
560: cmp r0, r2
563: jne 3
566: mov r3, 99('c')
572: cmp r1, r3
575: jne 3
578: mov r0, mem[2]
580: add_mem_ptr 1
582: mov r1, mem[3]
584: add_mem_ptr 1
586: mov r2, 117('u')
592: cmp r0, r2
595: jne 3
598: mov r3, 99('c')
604: cmp r1, r3
607: jne 3
610: mov r0, mem[4]
612: add_mem_ptr 1
614: mov r1, mem[5]
616: add_mem_ptr 1
618: mov r2, 116('t')
624: cmp r0, r2
627: jne 3
630: mov r3, 102('f')
636: cmp r1, r3
639: jne 3
642: mov r0, mem[6]
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：



最后与密文做比较：



写出exp：

```
#include <cstdio>

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']())
```

第二点是需要设置 `用0填充未约束内存` 选项：

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

因为程序在读取输入之前没有初始化VM结构体的`len`变量（一个小疏忽），所以在程序的一开始打印"Hello..."语句时`len`是处于未约束状态的，这样会导致angr在下面的for循环中产生路径爆炸。为了避免这种情况，我们需要用0来填充未约束的内存：

```
  break;
case 0x18u:                                             // write
  v6 = a1->PC[1];
  if ( (unsigned int)v6 > a1->len )
    v6 = a1->len;
  for ( i = 0; i < v6; ++i )
    putchar(*(unsigned __int8 *)(a1->mem_ptr + i));
  a1->PC += 2;
  break;
```

[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：

```
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
```

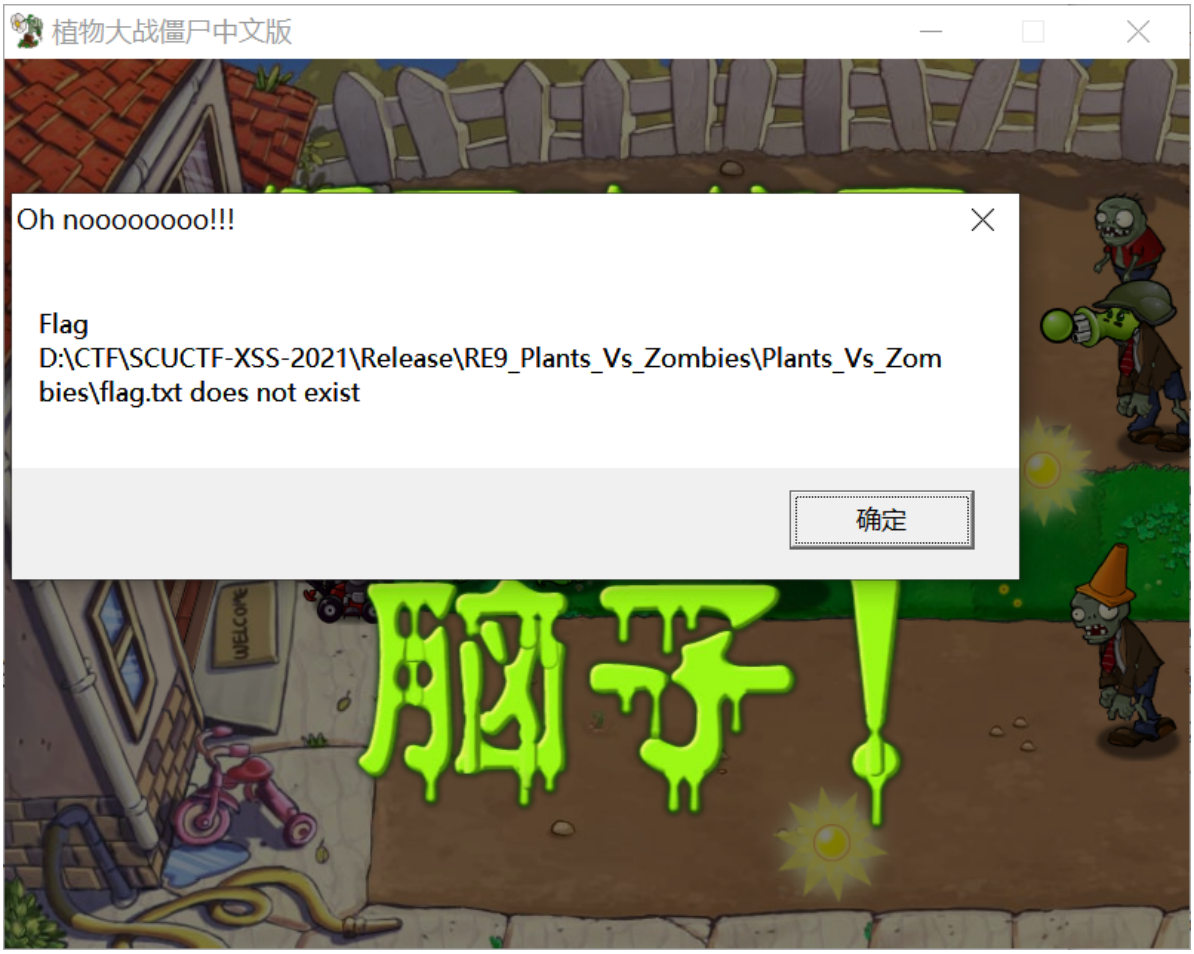
根据加密流程写出exp：

```python
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

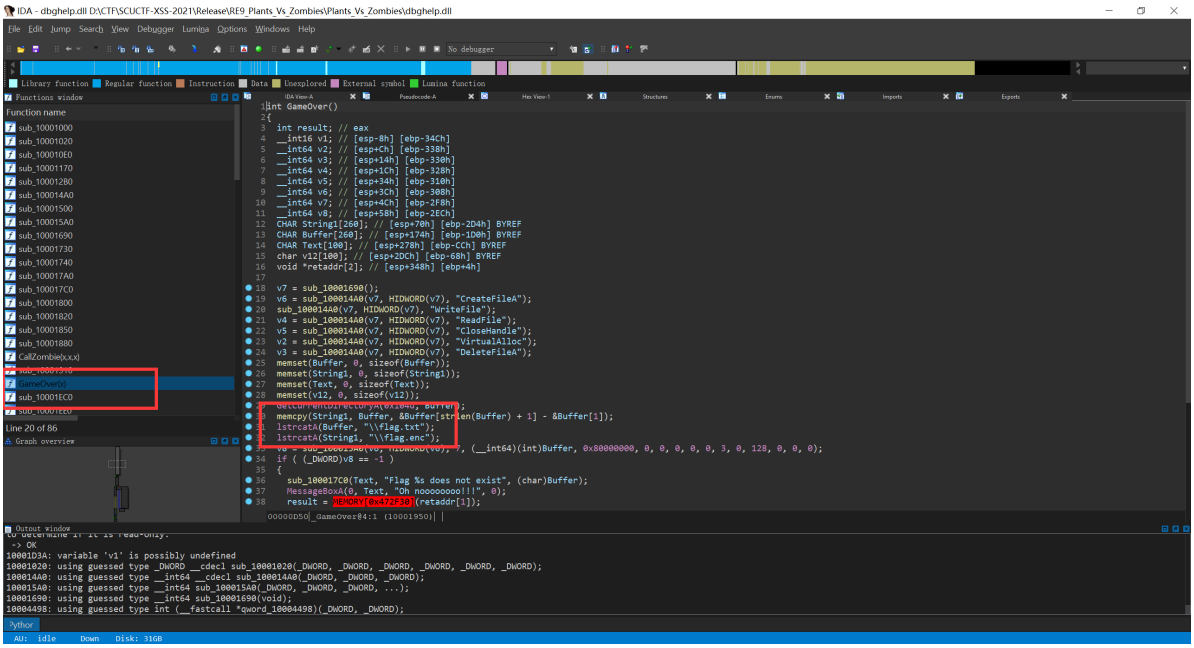
这题会在游戏失败加密PVZ目录下的flag.txt文件，加密后的文件保存为flag.enc，如果没有flag.txt文件也会提示。所以此题的关键是找出加密代码，并对flag.enc文件解密得到flag：



首先将PlantsVsZombies.exe拖入IDA，可以发现在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：

```
start           proc near               ; CODE XREF: start-18F↑j
                                        ; start-171↑j

; FUNCTION CHUNK AT .text:0061EA12 SIZE 0000018D BYTES

                push    offset aDbghelpDll ; "DBGHELP.DLL"

loc_61EBEC:                             ; CODE XREF: start-10↑j
                call    ds:LoadLibraryA
                call    sub_62E8F1
                jmp     loc_61EA12
start           endp
```

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：

```
v7 = sub_10001690();
v6 = sub_100014A0(v7, HIDWORD(v7), "CreateFileA");
sub_100014A0(v7, HIDWORD(v7), "WriteFile");
v4 = sub_100014A0(v7, HIDWORD(v7), "ReadFile");
v5 = sub_100014A0(v7, HIDWORD(v7), "CloseHandle");
v2 = sub_100014A0(v7, HIDWORD(v7), "VirtualAlloc");
v3 = sub_100014A0(v7, HIDWORD(v7), "DeleteFileA");
```

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：

```
else
{
  sub_100015A0(ReadFile_, HIDWORD(ReadFile_), 5, v8, (__int64)(int)v12, 100, 0, 0, 0, 0, 0);
  sub_100015A0(CloseHandle, HIDWORD(CloseHandle), 1, v8);
  sub_100015A0(DeleteFile, HIDWORD(DeleteFile), 1, (__int64)(int)Buffer);
  if ( !*(_QWORD *)&shellcode )
  {
    *(_QWORD *)&shellcode = sub_100015A0(VirtualAlloca, HIDWORD(VirtualAlloca), 4, 0, 0, 32, 0, 12288, 0, 64, 0);
    sub_10001020(shellcode, *(&shellcode + 1), &unk_100040E8, (int)&unk_100040E8 >> 31, 32, 0);
  }
  result = MK_FP(v1, shellcode)(&v12[strlen(v12) + 1] - &v12[1], &loc_10001D3B);
}
```

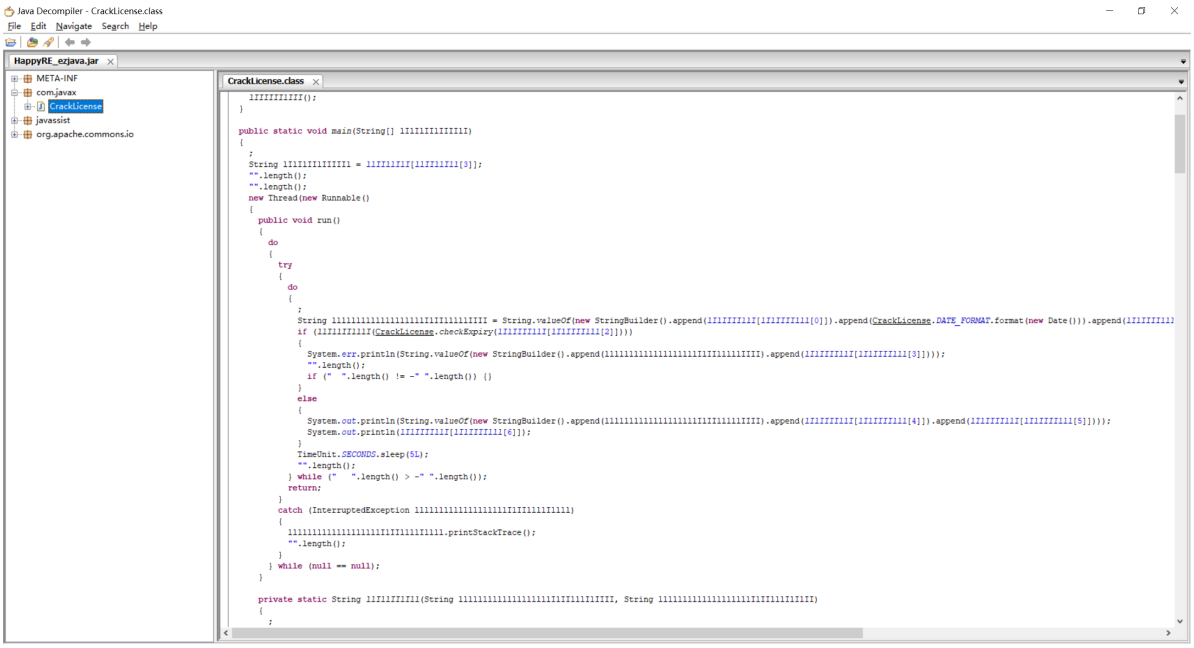
查看shellcode的内容，这段shellcode非常简单，本来是为了降低题目难度，没想到有同学直接用密码学的手段推出了加密流程（非预期解）：

根据加密流程和密文写出exp：

```
#include <cstdio>
#include <cstdint>

// scuctf{knockin_on_heavens_gate}
//
\x6e\x6d\xb0\x6f\x92\xd1\x11\x74\xd5\xf6\x76\x78\x39\xda\xf9\xfc\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\xf9\xfc\xdd\xfc\x1f\xb
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：



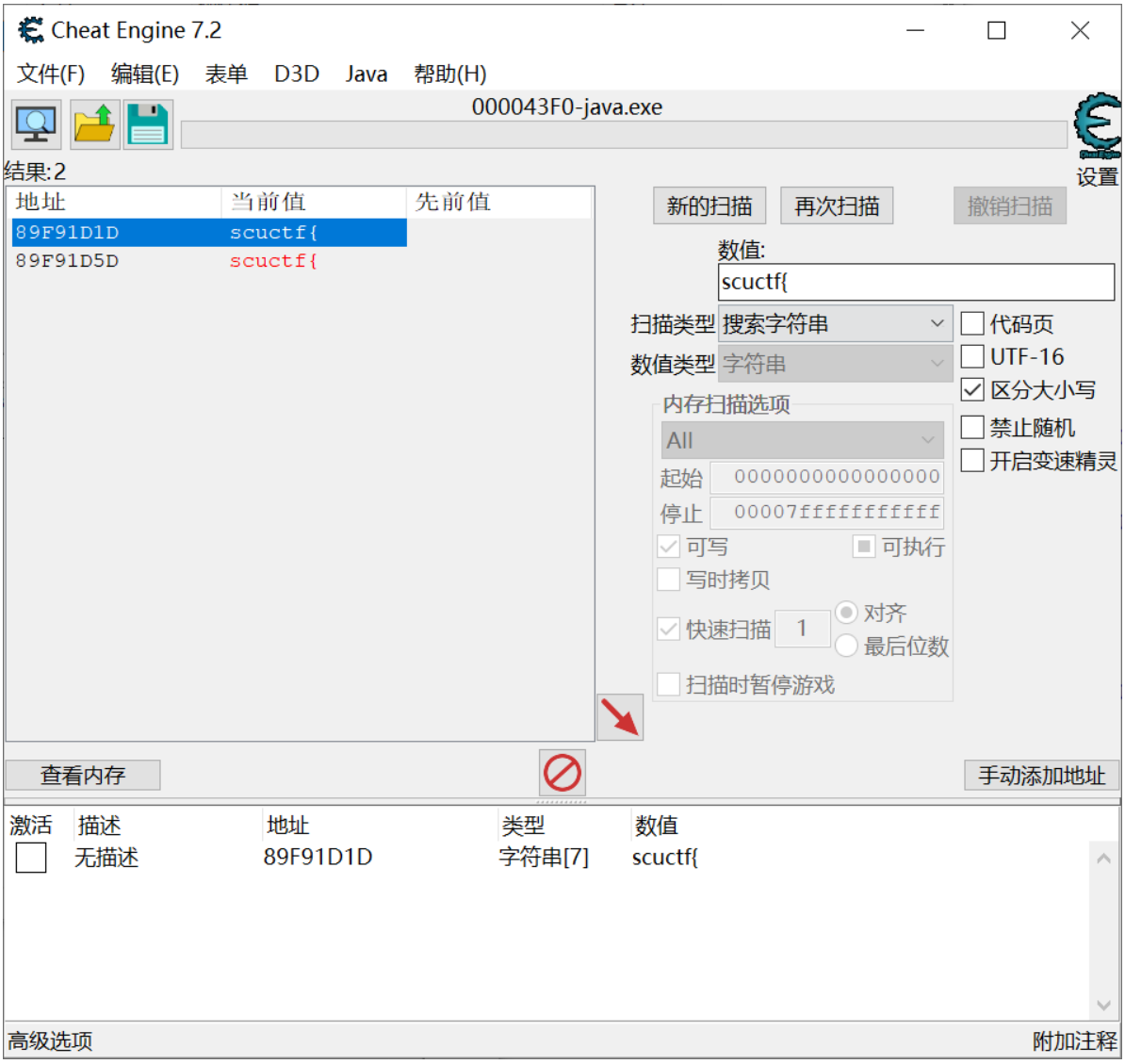
这题通过常规手段逆向有一定困难，因此可以考虑另辟蹊径。首先运行一下这个jar文件，发现会一直输出这一段话：

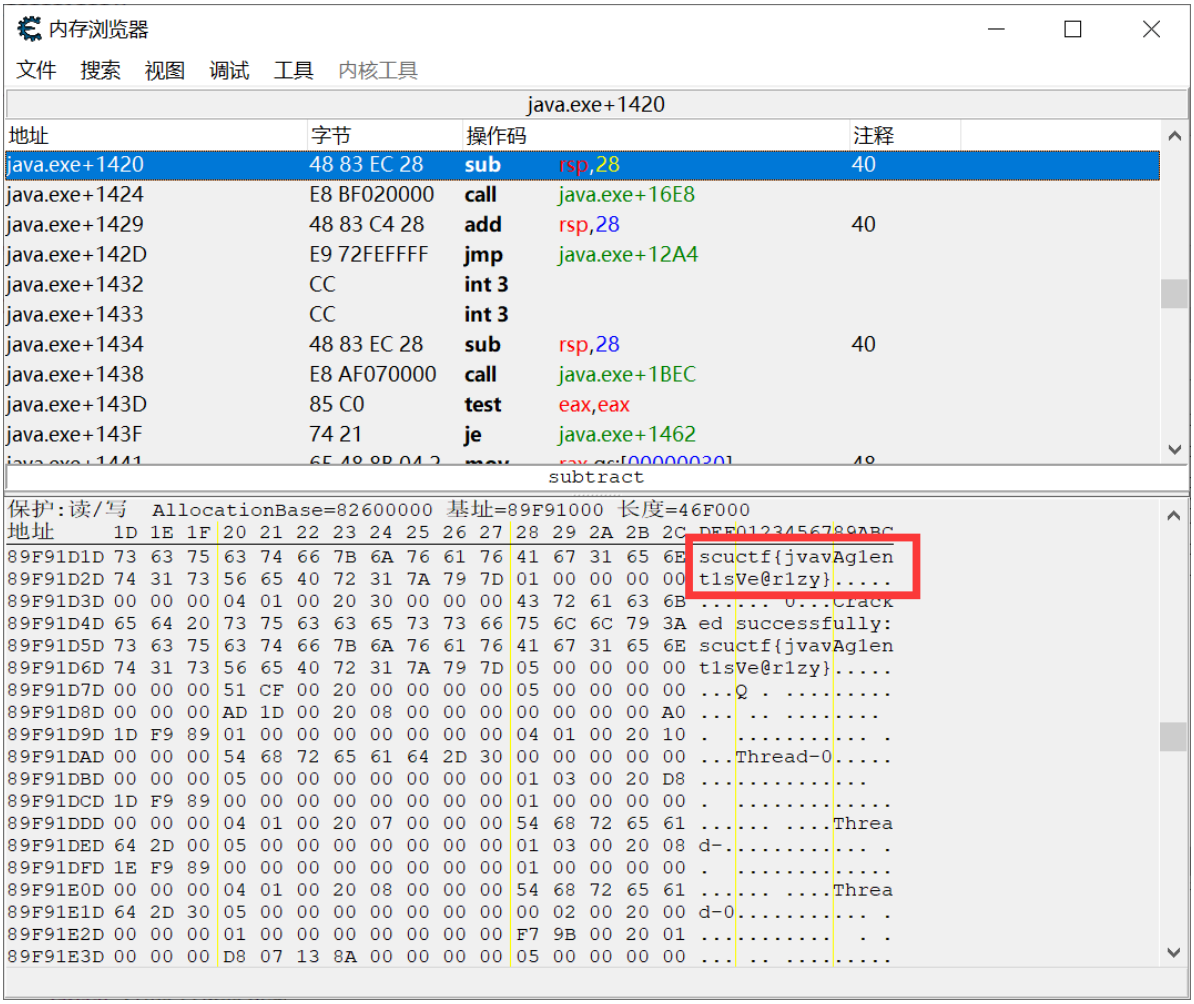
```
D:\CTF\SCUCTF-XSS-2021\Release\HappyRE_ezjava>java -jar HappyRE_ezjava.jar
[2021-12-06 21:12:01] Your license has expired, please purchase a new license from y4tacker!
[2021-12-06 21:12:06] Your license has expired, please purchase a new license from y4tacker!
```

对应的应该是这一部分的内容：

```
{
  do
  {
    ;
    String llllllllllllllllllIIIIIIIIII = String.valueOf(new StringBuilder().append(lIIIIIIIII[lIIIIIIIII[0]]).append(CrackLicense.DATE_FORMAT.format(new Date
    if (llIIIIIIIII(CrackLicense.checkExpiry(lIIIIIIIII[lIIIIIIIII[2]])))
    {
      System.err.println(String.valueOf(new StringBuilder().append(lllllllllllllllllllIIIIIIIII).append(lIIIIIIIII[lIIIIIIIII[3]])));
      "".length();
      if ("  ".length() != -" ".length()) {}
    }
    else
    {
      System.out.println(String.valueOf(new StringBuilder().append(lllllllllllllllllllIIIIIIIII).append(lIIIIIIIII[lIIIIIIIII[4]]).append(lIIIIIIIII[lIIIIIIII
      System.out.println(lIIIIIIIII[lIIIIIIIII[6]]);
    }
    TimeUnit.SECONDS.sleep(5L);
    "".length();
  } while ("  ".length() > -" ".length());
```

我们可以发现，如果校验失败会输出刚刚截屏中的那一段话，如果校验成功则应该输出正确的flag。可上面的if语句中并没有进行任何的加解密操作，而是直接输出一个字符串，所以可以认为其实flag已经提前解密出来在内存中的某个位置了，因此可以考虑用CE扫描内存，果然找到了了flag：


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
数值类型 字符串
代码页
UTF-16
区分大小写
禁止随机
开启变速精灵
内存扫描选项
All
起始 0000000000000000
停止 00007fffffffffff
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
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
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
89F91D1D 73 63 75 63 74 66 7B 6A 76 61 76 41 67 31 65 6E scuctf{jvavAg1en
89F91D2D 74 31 73 56 65 40 72 31 7A 79 7D 01 00 00 00 00 t1sVe@r1zy}.....
89F91D3D 00 00 00 04 01 00 20 30 00 00 00 43 72 61 63 6B
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{jvavAg1en
89F91D6D 74 31 73 56 65 40 72 31 7A 79 7D 05 00 00 00 00 t1sVe@r1zy}.....
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 61 ....Threa
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 61 ....Threa
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 ... .