buuctf刷题记录（3）

原创

7earn 于 2021-07-19 20:57:09 发布 274 收藏

分类专栏： 笔记 学习

本文链接：https://blog.csdn.net/weixin_53409153/article/details/118907216

笔记 同时被 2 个专栏收录

10 篇文章 0 订阅

订阅专栏

学习

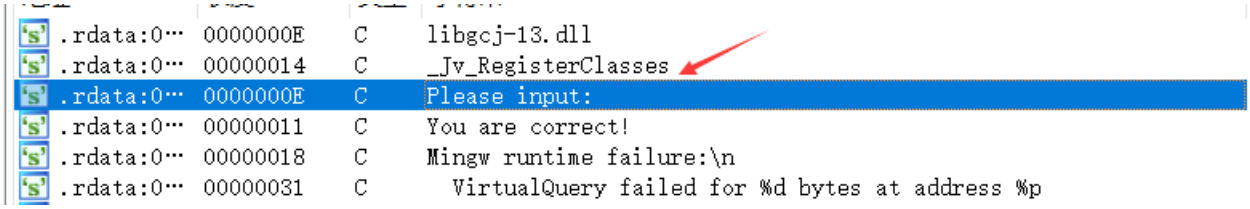
10 篇文章 0 订阅

订阅专栏

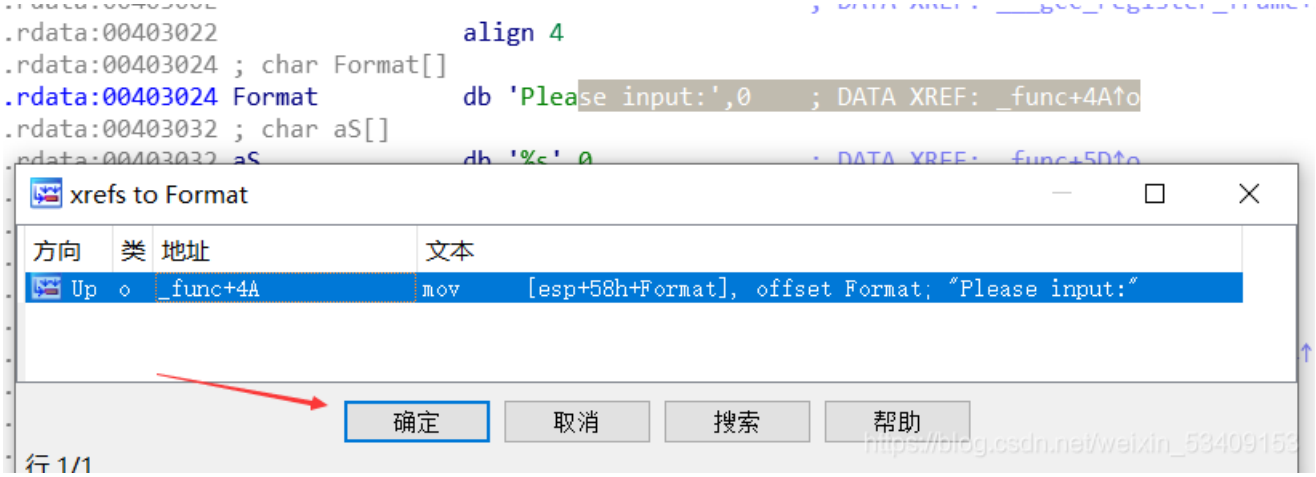
[ACTF新生赛2020]rome

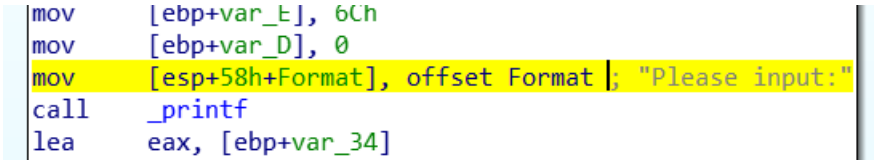
首先，还是查壳：

无壳，为32位：



跟进，跳转到交叉应用列表：





然后，在F5查看伪代码：

```
int func()
{
  int result; // eax
  int v1; // [esp+14h] [ebp-44h]
  int v2; // [esp+18h] [ebp-40h]
  int v3; // [esp+1Ch] [ebp-3Ch]
  int v4; // [esp+20h] [ebp-38h]
  unsigned __int8 v5; // [esp+24h] [ebp-34h]
  unsigned __int8 v6; // [esp+25h] [ebp-33h]
  unsigned __int8 v7; // [esp+26h] [ebp-32h]
  unsigned __int8 v8; // [esp+27h] [ebp-31h]
  unsigned __int8 v9; // [esp+28h] [ebp-30h]
  int v10; // [esp+29h] [ebp-2Fh]
  int v11; // [esp+2Dh] [ebp-2Bh]
```

```
int v12; // [esp+34h] [ebp-24h]
int v13; // [esp+35h] [ebp-23h]
unsigned __int8 v14; // [esp+39h] [ebp-1Fh]
char v15; // [esp+3Bh] [ebp-1Dh]
char v16; // [esp+3Ch] [ebp-1Ch]
char v17; // [esp+3Dh] [ebp-1Bh]
char v18; // [esp+3Eh] [ebp-1Ah]
char v19; // [esp+3Fh] [ebp-19h]
char v20; // [esp+40h] [ebp-18h]
char v21; // [esp+41h] [ebp-17h]
char v22; // [esp+42h] [ebp-16h]
char v23; // [esp+43h] [ebp-15h]
char v24; // [esp+44h] [ebp-14h]
char v25; // [esp+45h] [ebp-13h]
char v26; // [esp+46h] [ebp-12h]
char v27; // [esp+47h] [ebp-11h]
char v28; // [esp+48h] [ebp-10h]
char v29; // [esp+49h] [ebp-Fh]
char v30; // [esp+4Ah] [ebp-Eh]
char v31; // [esp+4Bh] [ebp-Dh]
int i; // [esp+4Ch] [ebp-Ch]

v15 = 81;
v16 = 115;
v17 = 119;
v18 = 51;
v19 = 115;
v20 = 106;
v21 = 95;
v22 = 108;
v23 = 122;
v24 = 52;
v25 = 95;
v26 = 85;
v27 = 106;
v28 = 119;
v29 = 64;
v30 = 108;
v31 = 0;
printf("Please input:");
scanf("%s", &v5);
result = v5;
if ( v5 == 65 )
{
  result = v6;
  if ( v6 == 67 )
  {
    result = v7;
    if ( v7 == 84 )
    {
      result = v8;
      if ( v8 == 70 )
      {
        result = v9;
        if ( v9 == 123 )
        {
          result = v14;
          if ( v14 == 125 )
          {
            v1 = v10;
```

```
              v2 = v11;
              v3 = v12;
              v4 = v13;
              for ( i = 0; i <= 15; ++i )
              {
                if ( *((_BYTE *)&v1 + i) > 64 && *((_BYTE *)&v1 + i) <= 90 )
                  *((_BYTE *)&v1 + i) = (*((char *)&v1 + i) - 51) % 26 + 65;
                if ( *((_BYTE *)&v1 + i) > 96 && *((_BYTE *)&v1 + i) <= 122 )
                  *((_BYTE *)&v1 + i) = (*((char *)&v1 + i) - 79) % 26 + 97;
              }
              for ( i = 0; i <= 15; ++i )
              {
                result = (unsigned __int8)*(&v15 + i);
                if ( *((_BYTE *)&v1 + i) != (_BYTE)result )
                  return result;
              }
              result = printf("You are correct!");
            }
          }
        }
      }
    }
  }
  return result;
}
```

v15到v30所显示的值是经过下面的算法加密后的值，所以我们要逆回去。
直接上脚本：

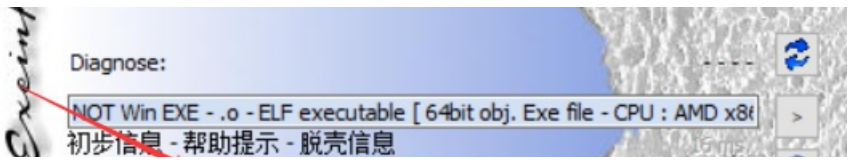
```
a=[81,115,119,51,115,106,95,108,122,52,95,85,106,119,64,108,0]
flag=''
for i in range(0,16):
    for k in range(0,127):
        z=k
        if k>64 and k<=90:
            k=(k-51)%26+65
        if k>96 and k<=122:
            k=(k-79)%26+97
        if k==a[i]:
            flag+=chr(z)
print(flag)
```

运行得到：

```
Connected to pydev debugger (build 211.7442.45)
Cae3ar_th4_Gre@t
```

加上`flag{Cae3ar_th4_Gre@t}`

[GUET-CTF2019]re

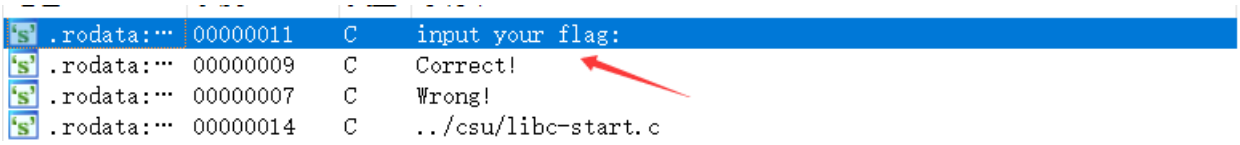


查壳：

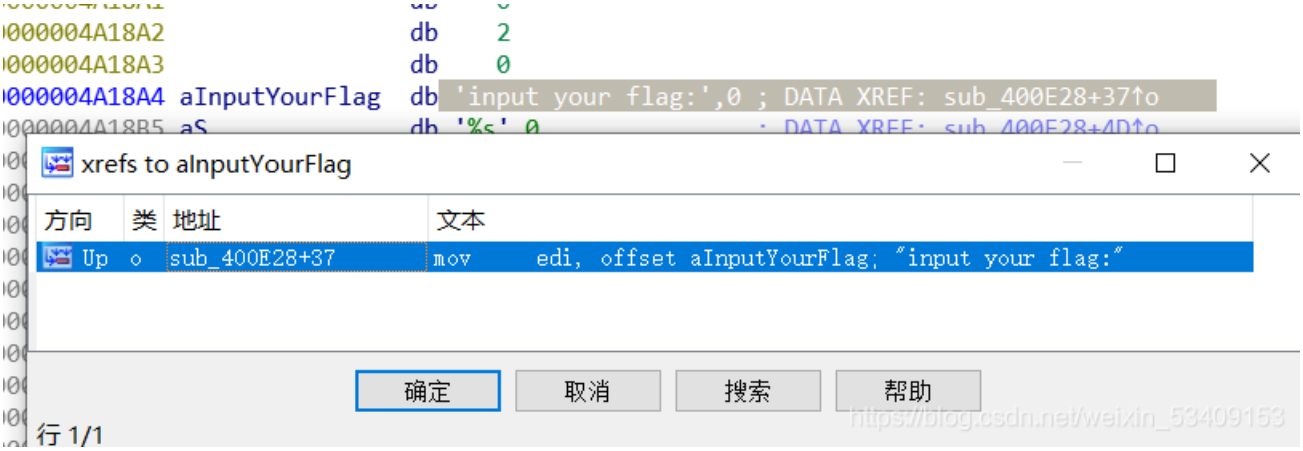
查壳后发现是64位，但是有UPX加壳，然后脱壳后:

```
UPX\upx-3.96-win64>upx -d re
                       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
   --------------------   ------   -----------   -----------
    840640 <-    304524   36.23%   linux/amd64   re

Unpacked 1 file.
```


在拖入IDA中：



跟进，跳转到交叉引用列表：



然后F5查看伪代码：

```
v5 = 0LL;
v6 = 0LL;
v7 = 0LL;
sub_40F950((unsigned __int64)"input your flag:");
sub_40FA80((unsigned __int64)"%s");
if ( (unsigned int)sub_4009AE((char *)&v4) )
{
  v0 = "Correct!";
  sub_410350("Correct!");
}
```

可以很简单的知道关键函数`sub_4009AE`

```
_BOOL8 __fastcall sub_4009AE(char *a1)
{
  if ( 1629056 * *a1 != 166163712 )
    return 0LL;
```

```
if ( 6771600 * a1[1] != 731332800 )
  return 0LL;
if ( 3682944 * a1[2] != 357245568 )
  return 0LL;
if ( 10431000 * a1[3] != 1074393000 )
  return 0LL;
if ( 3977328 * a1[4] != 489211344 )
  return 0LL;
if ( 5138336 * a1[5] != 518971936 )
  return 0LL;
if ( 7532250 * a1[7] != 406741500 )
  return 0LL;
if ( 5551632 * a1[8] != 294236496 )
  return 0LL;
if ( 3409728 * a1[9] != 177305856 )
  return 0LL;
if ( 13013670 * a1[10] != 650683500 )
  return 0LL;
if ( 6088797 * a1[11] != 298351053 )
  return 0LL;
if ( 7884663 * a1[12] != 386348487 )
  return 0LL;
if ( 8944053 * a1[13] != 438258597 )
  return 0LL;
if ( 5198490 * a1[14] != 249527520 )
  return 0LL;
if ( 4544518 * a1[15] != 445362764 )
  return 0LL;
if ( 3645600 * a1[17] != 174988800 )
  return 0LL;
if ( 10115280 * a1[16] != 981182160 )
  return 0LL;
if ( 9667504 * a1[18] != 493042704 )
  return 0LL;
if ( 5364450 * a1[19] != 257493600 )
  return 0LL;
if ( 13464540 * a1[20] != 767478780 )
  return 0LL;
if ( 5488432 * a1[21] != 312840624 )
  return 0LL;
if ( 14479500 * a1[22] != 1404511500 )
  return 0LL;
if ( 6451830 * a1[23] != 316139670 )
  return 0LL;
if ( 6252576 * a1[24] != 619005024 )
  return 0LL;
if ( 7763364 * a1[25] != 372641472 )
  return 0LL;
if ( 7327320 * a1[26] != 373693320 )
  return 0LL;
if ( 8741520 * a1[27] != 498266640 )
  return 0LL;
if ( 8871876 * a1[28] != 452465676 )
  return 0LL;
if ( 4086720 * a1[29] != 208422720 )
  return 0LL;
if ( 9374400 * a1[30] == 515592000 )
  return 5759124 * a1[31] == 719890500;
return 0LL;
```

```
}
```

在这里a1就是我们输入的v4，反向除一下就可以了，而且其中没有a[6]的运算：

```
flag{e65421110ba03099a1c039337}
```

这上面缺了一位，只能爆破得到a7=1
得到`flag{e165421110ba03099a1c039337}`

[ACTF新生赛2020]usualCrypt

查壳：

发现无壳，然后就拖入IDA中：

跟进，F5查看伪代码：

```
v7 = 0;
v8 = 0;
v9 = 0;
sub_401080((int)&v10, strlen(&v10), (int)&v5);
v3 = 0;
while ( *((_BYTE *)&v5 + v3) == byte_40E0E4[v3] )
```

进入主函数：

```
int __cdecl sub_401080(int a1, int a2, int a3)
{
  int v3; // edi
  int v4; // esi
  int v5; // edx
  int v6; // eax
  int v7; // ecx
  int v8; // esi
  int v9; // esi
  int v10; // esi
  int v11; // esi
  _BYTE *v12; // ecx
  int v13; // esi
  int v15; // [esp+18h] [ebp+8h]

  v3 = 0;
  v4 = 0;
  sub_401000();
```

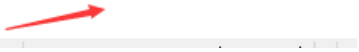
```
  v5 = a2 % 3;
  v6 = a1;
  v7 = a2 - a2 % 3;
  v15 = a2 % 3;
  if ( v7 > 0 )
  {
    do
    {
      LOBYTE(v5) = *(_BYTE *)(a1 + v3);
      v3 += 3;
      v8 = v4 + 1;
      *(_BYTE *)(v8++ + a3 - 1) = byte_40E0A0[(v5 >> 2) & 0x3F];
      *(_BYTE *)(v8++ + a3 - 1) = byte_40E0A0[16 * (*(_BYTE *)(a1 + v3 - 3) & 3)
                                           + (((signed int)*(unsigned __int8 *)(a1 + v3 - 2) >> 4) & 0xF)];
      *(_BYTE *)(v8 + a3 - 1) = byte_40E0A0[4 * (*(_BYTE *)(a1 + v3 - 2) & 0xF)
                                          + (((signed int)*(unsigned __int8 *)(a1 + v3 - 1) >> 6) & 3)];
      v5 = *(_BYTE *)(a1 + v3 - 1) & 0x3F;
      v4 = v8 + 1;
      *(_BYTE *)(v4 + a3 - 1) = byte_40E0A0[v5];
    }
    while ( v3 < v7 );
    v5 = v15;
  }
  if ( v5 == 1 )
  {
    LOBYTE(v7) = *(_BYTE *)(v3 + a1);
    v9 = v4 + 1;
    *(_BYTE *)(v9 + a3 - 1) = byte_40E0A0[(v7 >> 2) & 0x3F];
    v10 = v9 + 1;
    *(_BYTE *)(v10 + a3 - 1) = byte_40E0A0[16 * (*(_BYTE *)(v3 + a1) & 3)];
    *(_BYTE *)(v10 + a3) = 61;
LABEL_8:
    v13 = v10 + 1;
    *(_BYTE *)(v13 + a3) = 61;
    v4 = v13 + 1;
    goto LABEL_9;
  }
  if ( v5 == 2 )
  {
    v11 = v4 + 1;
    *(_BYTE *)(v11 + a3 - 1) = byte_40E0A0[((signed int)*(unsigned __int8 *)(v3 + a1) >> 2) & 0x3F];
    v12 = (_BYTE *)(v3 + a1 + 1);
    LOBYTE(v6) = *v12;
    v10 = v11 + 1;
    *(_BYTE *)(v10 + a3 - 1) = byte_40E0A0[16 * (*(_BYTE *)(v3 + a1) & 3) + ((v6 >> 4) & 0xF)];
    *(_BYTE *)(v10 + a3) = byte_40E0A0[4 * (*v12 & 0xF)];
    goto LABEL_8;
  }
LABEL_9:
  *(_BYTE *)(v4 + a3) = 0;
  return sub_401030(a3);
}
```

经过分析，看到有个base64转换表，大概这是一个base64的加密。然后看一下sub_401000()函数：

```
signed int sub_401000()
{
  signed int result; // eax
  char v1; // cl

  result = 6;
  do
  {
    v1 = byte_40E0AA[result];
    byte_40E0AA[result] = byte_40E0A0[result];
    byte_40E0A0[result++] = v1;
  }
  while ( result < 15 );
  return result;
}
```

这里是将base64转换表个别顺序进行交换。函数中两个数组分别是转换表的不同位置。
最后再看一下sub_401030()函数：

```
  *(_BYTE *)(v4 + a3) = 0;
  return sub_401030(a3);
}
```

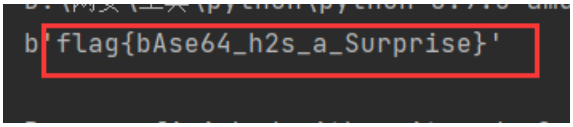


```
int __cdecl sub_401030(const char *a1)
{
  __int64 v1; // rax
  char v2; // al

  v1 = 0i64;
  if ( strlen(a1) != 0 )
  {
    do
    {
      v2 = a1[HIDWORD(v1)];
      if ( v2 < 97 || v2 > 122 )
      {
        if ( v2 < 65 || v2 > 90 )
          goto LABEL_9;
        LOBYTE(v1) = v2 + 32;
      }
      else
      {
        LOBYTE(v1) = v2 - 32;
      }
      a1[HIDWORD(v1)] = v1;
LABEL_9:
      LODWORD(v1) = 0;
      ++HIDWORD(v1);
    }
    while ( HIDWORD(v1) < strlen(a1) );
  }
  return v1;
}
```

这个函数就是将加密后的结果大小写互换。
解题的思路：
1.结果大小写互换
2.修改base64转换表
3.加密结果通过转换表得到正常的加密后的结果
4.base64解密
脚本：

```
import base64

secret = 'zMXHz3TIgnxLxJhFAdtZn2fFk3lYCrtPC2l9'.swapcase()
a = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
dict = {}
offset = 10
flag = ''
for i in range(len(a)):
    dict[a[i]] = a[i]
for i in range(6,15):
    b = dict[a[i]]
    dict[a[i]] = dict[a[i+offset]]
    dict[a[i+offset]] = b
for i in range(len(secret)):
    flag += dict[secret[i]]
flag = base64.b64decode(flag)
print(flag)
```

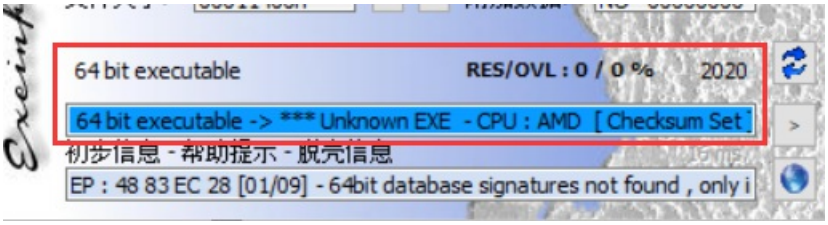
最后得到：



答案`flag{bAse64_h2s_a_Surprise}`

[MRCTF2020]Transform

查壳：



发现无壳，64位，拖入IDA：

```
__int64 v3; // rdx
__int64 v4; // rdx
char v6[104]; // [rsp+20h] [rbp-70h]
int j; // [rsp+88h] [rbp-8h]
int i; // [rsp+8Ch] [rbp-4h]

sub_402230(argc, argv, envp);
sub_40E640(argc, (__int64)argv, v3, (__int64)"Give me your code:\n");
sub_40E5F0(argc, (__int64)argv, (__int64)v6, (__int64)"%s");
if ( strlen(*(const char **)&argc) != 33 )
```

```
  {
    sub_40E640(argc, (__int64)argv, v4, (__int64)"Wrong!\n");
    system(*(const char **)&argc);
    exit(argc);
  }
  for ( i = 0; i <= 32; ++i )
  {
    byte_414040[i] = v6[dword_40F040[i]];
    v4 = i;
    byte_414040[i] ^= LOBYTE(dword_40F040[i]);
  }
  for ( j = 0; j <= 32; ++j )
  {
    v4 = j;
    if ( aGyURsywBFbLwya[j] != byte_414040[j] )
    {
      sub_40E640(argc, (__int64)argv, j, (__int64)"Wrong!\n");
      system(*(const char **)&argc);
      exit(argc);
    }
  }
  sub_40E640(argc, (__int64)argv, v4, (__int64)"Right!Good Job!\n");
  sub_40E640(argc, (__int64)argv, (__int64)v6, (__int64)"Here is your flag: %s\n");
  system(*(const char **)&argc);
  return 0;
}
```


然后看一下 byte_414040

```
}
for ( i = 0; i <= 32; ++i )
{
  byte_414040[i] = v6[dword_40F040[i]];
  v4 = i;
  byte_414040[i] ^= LOBYTE(dword_40F040[i]);
}
```

然后发现：

```
.data:000000000040F004                  align 40h
.data:000000000040F040 ; signed int dword_40F040[40]
.data:000000000040F040 dword_40F040     dd 9, 0Ah, 0Fh, 17h, 7, 18h, 0Ch, 6, 1, 10h, 3, 11h, 20h
.data:000000000040F040                                          ; DATA XREF: main+79↑o
.data:000000000040F040                                          ; main+B8↑o
.data:000000000040F040                  dd 1Dh, 0Bh, 1Eh, 1Bh, 16h, 4, 0Dh, 13h, 14h, 15h, 2, 19h
.data:000000000040F040                  dd 5, 1Fh, 8, 12h, 1Ah, 1Ch, 0Eh, 8 dup(0)
.data:000000000040F0E0 aGyURsywBFbLwya db 'gy{',7Fh,'u+<RSyW^]B{-*fB~LWyAk~e<\EobM',0
.data:000000000040F0E0                                          ; DATA XREF: main+EF↑o
.data:000000000040F102                  align 40h
.data:000000000040F140 off_40F140       dq offset qword_40E6A8  ; DATA XREF: sub_402190+4↑r
```


思路是先将dword_40F040和byte_40F0E0异或一下得到打乱后的输入的字符串之后将打乱的字符串还原回去即可得到输入的字符串

然后脚本：

```
#include <stdio.h>
int main()
{
int i;
char c[33];
char a[]={0x09,0x0a,0x0f,0x17,0x07,0x18,0x0c,0x06,0x01,0x10,0x03,0x11,0x20,0x1d,0x0b,0x1e,0x1b,0x16,0x04,0x0d,0x
13,0x14,0x15,0x02,0x19,0x05,0x1f,0x08,0x12,0x1a,0x1c,0x0e,0} ;
char b[]={0x67,0x79,0x7b,0x7f,0x75,0x2b,0x3c,0x52,0x53,0x79,0x57,0x5e,0x5d,0x42,0x7b,0x2d,0x2a,0x66,0x42,0x7e,0x
4c,0x57,0x79,0x41,0x6b,0x7e,0x65,0x3c,0x5c,0x45,0x6f,0x62,0x4d};
for(i=0;i<=32;i++)
{
 b[i]=b[i]^a[i];
}
for(i=0;i<=32;i++)
{
 c[a[i]]=b[i];
}
for(i=0;i<=32;i++)
{
 printf("%c",c[i]);
}
}
```

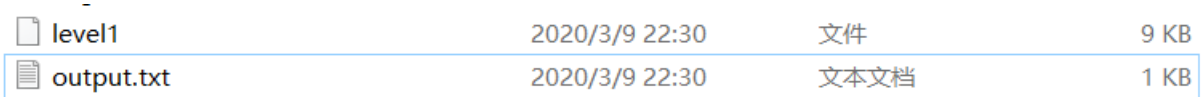
运行：

```
MRCTF{Tr4nsp0sltiON_Clph3r_1s_3z}
--------------------------------
```

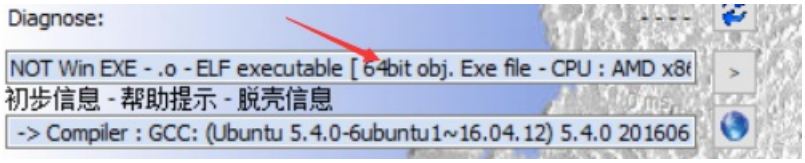
得到：`MRCTF{Tr4nsp0sltiON_Clph3r_1s_3z}`

[WUSTCTF2020]level1

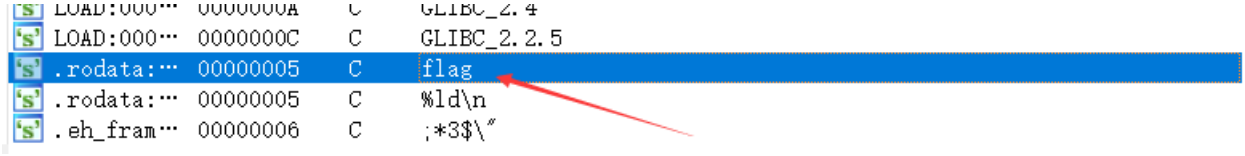
先是解压得到：



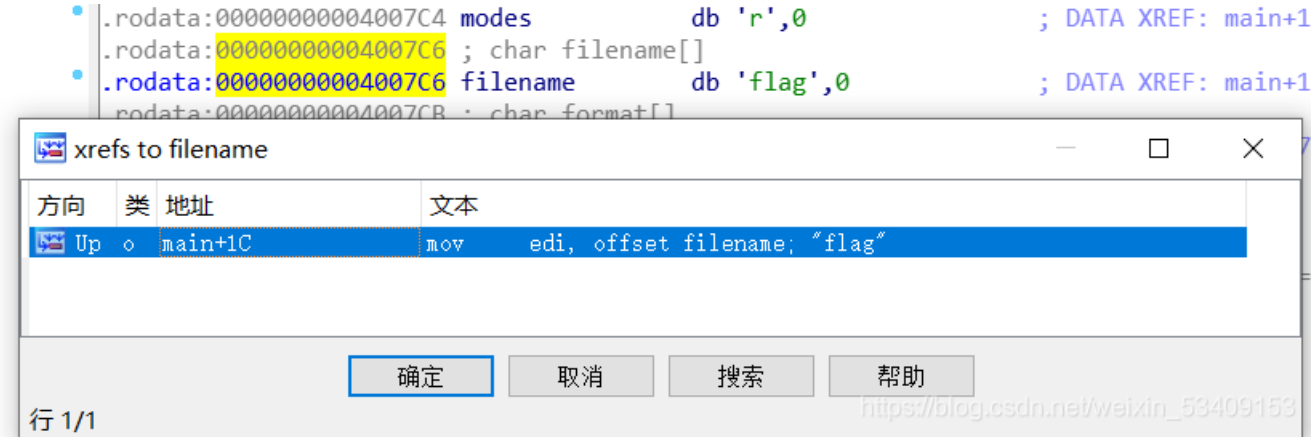
然后对第一个文件查壳：



发现无壳，拖入IDA：



跟进，跳转到交叉引用列表：



然后F5查看伪代码：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  FILE *stream; // ST08_8
  signed int i; // [rsp+4h] [rbp-2Ch]
  char ptr[24]; // [rsp+10h] [rbp-20h]
  unsigned __int64 v7; // [rsp+28h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  stream = fopen("flag", "r");
  fread(ptr, 1uLL, 0x14uLL, stream);
  fclose(stream);
  for ( i = 1; i <= 19; ++i )
  {
    if ( i & 1 )
      printf("%ld\n", (unsigned int)(ptr[i] << i));
    else
      printf("%ld\n", (unsigned int)(i * ptr[i]));
  }
  return 0;
}
```

程序很简单，一开始打开flag文件读出了flag文件里的内容，之后将里面的内容按照12行到18行的语句进行处理后输出。

4.附件里还有一个文件，output.txt，猜测就是输出后的内容，将里面的数据根据这个算法还原一下flag里的内容
然后脚本：

```
a = [198,232,816,200,1536,300,6144,984,51200,570,92160,1200,565248,756,1474560,800,6291456,1782,65536000]

for i in range(19):
    if ((i+1) & 1):
        print(chr(a[i] >> (i+1)), end="")
    else:
        print (chr(a[i] // (i+1)),end="")
```

运行得到：

```
ctf2020{d9-dE6-20c}
```

flag：`ctf2020{d9-dE6-20c}`