【CSICTF】pydis2ctf WriteUp

原创

古月浪子 于 2020-07-23 16:05:22 发布 326 收藏 1

文章标签： CTF

本文链接：https://blog.csdn.net/tqydyqt/article/details/107474498

这题我感觉我做的有问题，最后虽然答案是勉强凑出来了

题目给了3个附件

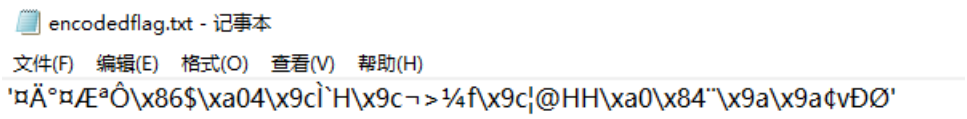
cipher1：

```
2           0 LOAD_CONST               1 ('')
            2 STORE_FAST               1 (ret_text)

3           4 LOAD_GLOBAL              0 (list)
            6 LOAD_FAST                0 (text)
            8 CALL_FUNCTION            1
           10 GET_ITER
      >>   12 FOR_ITER                42 (to 56)
           14 STORE_FAST               2 (i)

4          16 LOAD_FAST                0 (text)
           18 LOAD_METHOD              1 (count)
           20 LOAD_FAST                2 (i)
           22 CALL_METHOD              1
           24 STORE_FAST               3 (counter)

5          26 LOAD_FAST                1 (ret_text)
           28 LOAD_GLOBAL              2 (chr)
           30 LOAD_CONST               2 (2)
           32 LOAD_GLOBAL              3 (ord)
           34 LOAD_FAST                2 (i)
           36 CALL_FUNCTION            1
           38 BINARY_MULTIPLY
           40 LOAD_GLOBAL              4 (len)
           42 LOAD_FAST                0 (text)
           44 CALL_FUNCTION            1
           46 BINARY_SUBTRACT
           48 CALL_FUNCTION            1
           50 INPLACE_ADD
           52 STORE_FAST               1 (ret_text)
           54 JUMP_ABSOLUTE           12

6     >>   56 LOAD_FAST                1 (ret_text)
           58 RETURN_VALUE
```

cipher2：

```
3           0 LOAD_CONST               1 ('S')
            2 STORE_FAST               1 (xorKey)

 4          4 LOAD_GLOBAL              0 (len)
            6 LOAD_FAST                0 (inpString)
            8 CALL_FUNCTION            1
           10 STORE_FAST               2 (length)

 5          12 LOAD_GLOBAL             1 (range)
           14 LOAD_FAST                2 (length)
           16 CALL_FUNCTION            1
           18 GET_ITER
      >>   20 FOR_ITER                56 (to 78)
           22 STORE_FAST               3 (i)

 6          24 LOAD_FAST                0 (inpString)
           26 LOAD_CONST               0 (None)
           28 LOAD_FAST                3 (i)
           30 BUILD_SLICE              2
           32 BINARY_SUBSCR
           34 LOAD_GLOBAL              2 (chr)
           36 LOAD_GLOBAL              3 (ord)
           38 LOAD_FAST                0 (inpString)
           40 LOAD_FAST                3 (i)
           42 BINARY_SUBSCR
           44 CALL_FUNCTION            1
           46 LOAD_GLOBAL              3 (ord)
           48 LOAD_FAST                1 (xorKey)
           50 CALL_FUNCTION            1
           52 BINARY_XOR
           54 CALL_FUNCTION            1
           56 BINARY_ADD
           58 LOAD_FAST                0 (inpString)
           60 LOAD_FAST                3 (i)
           62 LOAD_CONST               2 (1)
           64 BINARY_ADD
           66 LOAD_CONST               0 (None)
           68 BUILD_SLICE              2
           70 BINARY_SUBSCR
           72 BINARY_ADD
           74 STORE_FAST               0 (inpString)
           76 JUMP_ABSOLUTE           20

 9     >>   78 LOAD_FAST                0 (inpString)
           80 RETURN_VALUE
```

encodedflag：

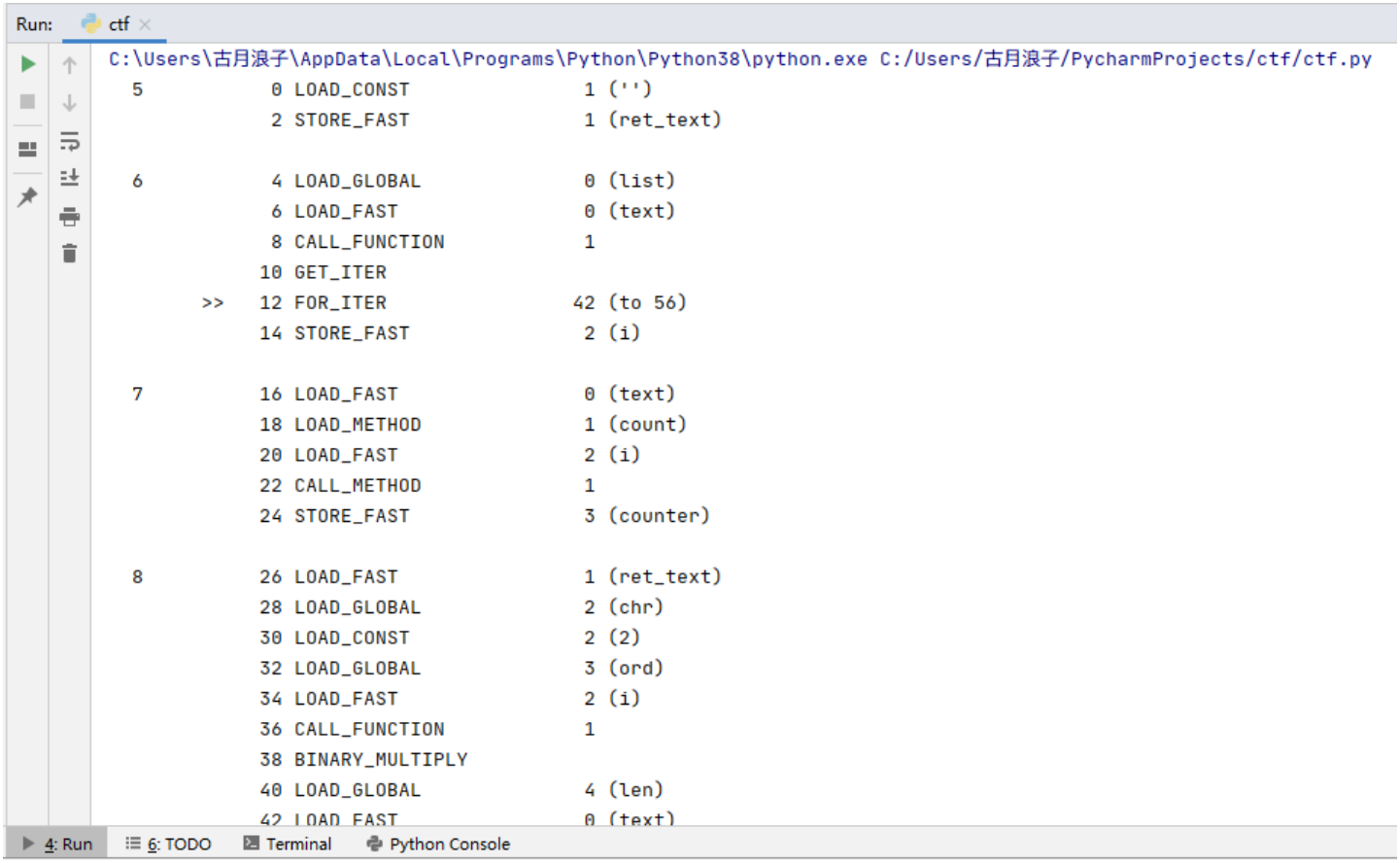


根据题目很容易想到python的dis库
具体的分析方法可以直接使用搜索引擎一键传送到学习区，这里不细讲了

随着我一阵摸索+乱凑，终于把python代码完整的还原出来了

```python
import dis


def cipher1(text):
    ret_text = ''
    for i in list(text):
        counter = text.count(i)
        ret_text += chr(2 * ord(i) - len(text))
    return ret_text


print(dis.dis(cipher1))


def cipher2(inpString):
    xorKey = 'S'
    length = len(inpString)
    for i in range(length):
        inpString = inpString[:i] + chr(ord(inpString[i]) ^ ord(xorKey)) + inpString[i + 1:]
    return inpString


print(dis.dis(cipher2))
```

可以看到运行出来打印的和附件给的几乎一模一样，因此我们有把握认为加密代码就是我还原出来的



然后就是写逆向代码了，结果。。。

```cpp
#include <iostream>

using namespace std;

int main()
{
 unsigned char flag[58] = {
 0xC2, 0xA4, 0xC3, 0x84, 0xC2, 0xB0, 0xC2, 0xA4, 0xC3, 0x86, 0xC2, 0xAA, 0xC3, 0x94, 0xC2, 0x86,
 0x24, 0xC2, 0xA0, 0x34, 0xC2, 0x9C, 0xC3, 0x8C, 0x60, 0x48, 0xC2, 0x9C, 0xC2, 0xAC, 0x3E, 0xC2,
 0xBC, 0x66, 0xC2, 0x9C, 0xC2, 0xA6, 0x40, 0x48, 0x48, 0xC2, 0xA0, 0xC2, 0x84, 0xC2, 0xA8, 0xC2,
 0x9A, 0xC2, 0x9A, 0xC2, 0xA2, 0x76, 0xC3, 0x90, 0xC3, 0x98
 };
 for (size_t i = 0; i < 58; i++)
 {
  flag[i] += 34;
  flag[i] /= 2;
 }
 cout << (char*)flag << endl;
}
```

不要问我为什么长度是58而加的是34，不要问我为什么没有异或，不要问我为什么这答案看起来这么怪... **因为是错的**

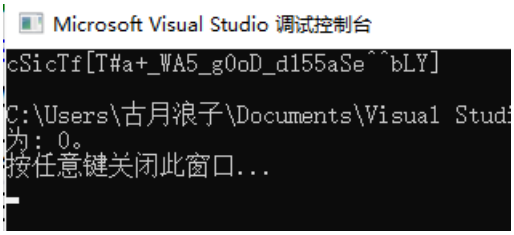
但是如果老老实实按照代码去还原，输出的是乱码 =_=

初步怀疑可能是WinHex的问题或者是python编码的问题？因为按理来说加密后的每个字节要么全是奇数要么全是偶数，我在WinHex里看到的又有奇数又有偶数...

因此，如果你用其他的二进制编辑工具能够正确提取密文的话，这道题应该就很简单了

在这里我记录一下我自己当时做题的过程

观察到0xc2和0xc3反复出现，出现的位置往往是上面图中的r字符，因此全部去掉，去掉之后长度就是34了。去掉之后，看起来有点像flag了，但是还是不对



然后我根据上面的 cSicTf[xxx] 与其他题目的 csictf{xxx} 发现了一个规律，对应字符的原位置的前面如果是0xc3的，要变成小写/大括号
改完以后是这样

> csictf{T#a+_wA5_g0oD_d155aSe^^bLy}

嗯，这个终于对了 (Ⅲ￣ω￣)