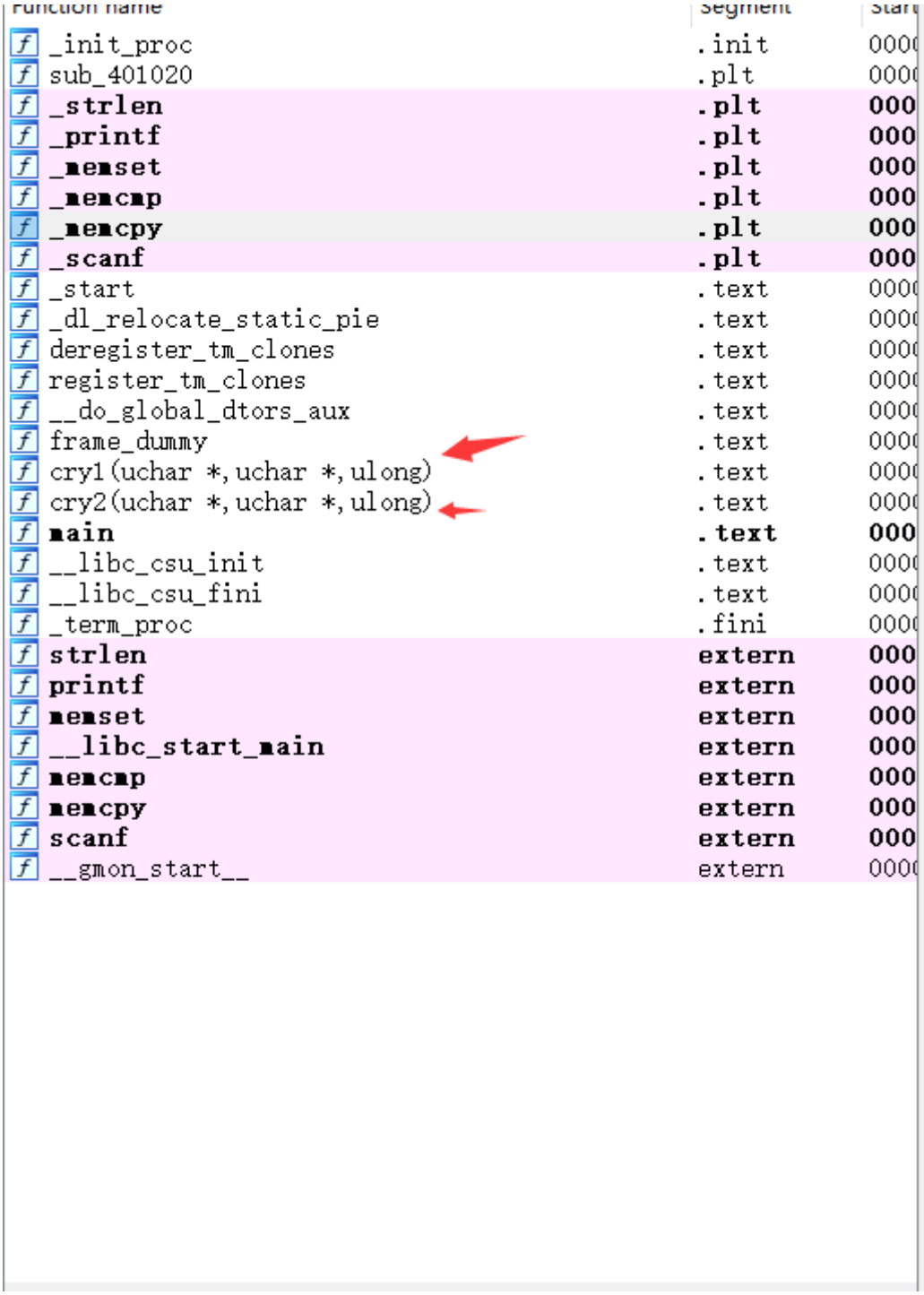
RE3

题目是用ollvm来混淆的 所以得用IDA去混淆插件D-801去除混淆

先用D-801去除混淆

可以看到程序主要有两个加密函数 一个是cry1

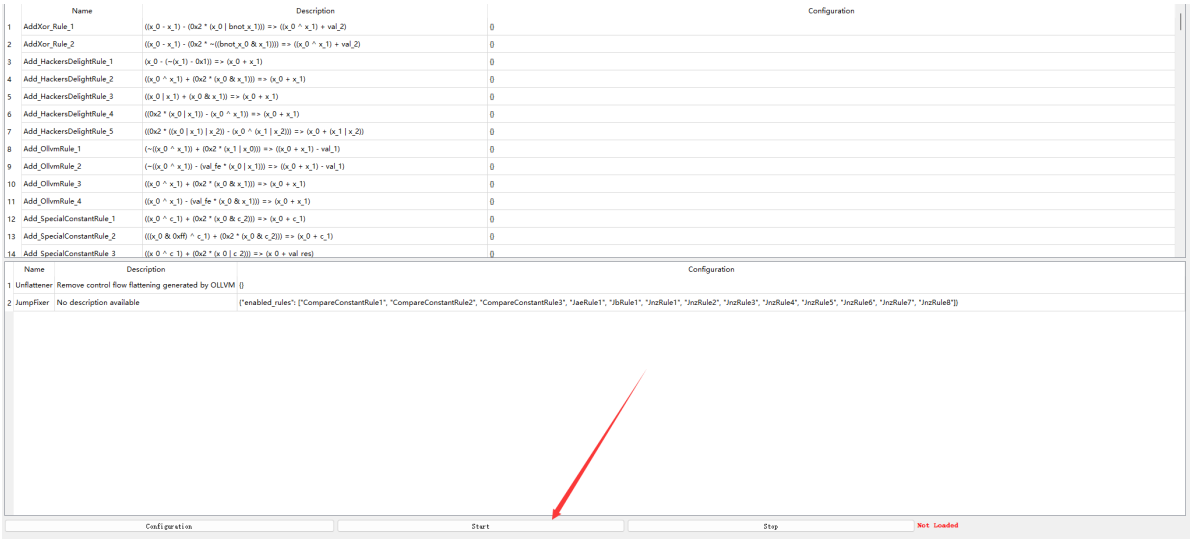
一个是cry2



但是都带有混淆

```
  v17 = a1;
  v16 = a2;
  v15 = a3;
  v14 = 0;
  v13 = 0;
  memset(s, 0, 0x100uLL);
  v14 = 0;
  v10 = 1135530334;
  while ( 1 )
  {
    while ( 1 )
    {
      while ( 1 )
      {
        while ( 1 )
        {
          while ( 1 )
          {
            while ( 1 )
            {
              while ( 1 )
              {
                while ( 1 )
                {
                  while ( 1 )
                  {
                    while ( 1 )
                    {
                      while ( v10 == -2026262691 )
                      {
                        ++v14;
                        v10 = 219113649;
                      }
                      if ( v10 != -1800435896 )
                        break;
                      v10 = -1154024163;
                    }
                    if ( v10 != -1154024163 )
                      break;
                    v5 = -1800435896;
                    v18 = v14 < 256;
                    if ( (y < 10) ^ (((((_BYTE)x - 1) * (_BYTE)x) & 1) == 0) | (y < 10
                                                                              && ((((_BYTE)x - 1) * (_BYTE)x) & 1) == 0) )
                      v5 = 741148234;
                    v10 = v5;
                  }
                  if ( v10 != -872009407 )
                    break;
                  v13 = ((unsigned __int8)s[v14] + v13 + v17[v14]) % 256;
                  v11 = v17[v14];
                  v17[v14] = v17[v13];
                  v17[v13] = v11;
                  v10 = -2026262691;
                }
                if ( v10 != 219113649 )
                  break;
                v7 = 1468176970;
                if ( v14 < 256 )
                  v7 = -872009407;
```

我们运行D-801插件



	Name	Description	Configuration
1	AddXor_Rule_1	((x_0 - x_1) - (0x2 * (x_0 | bnot_x_1))) => ((x_0 ^ x_1) + val_2)	{}
2	AddXor_Rule_2	((x_0 - x_1) - (0x2 * ~((bnot_x_0 & x_1)))) => ((x_0 ^ x_1) + val_2)	{}
3	Add_HackersDelightRule_1	(x_0 - (~(x_1) - 0x1)) => (x_0 + x_1)	{}
4	Add_HackersDelightRule_2	((x_0 ^ x_1) + (0x2 * (x_0 & x_1))) => (x_0 + x_1)	{}
5	Add_HackersDelightRule_3	((x_0 | x_1) + (x_0 & x_1)) => (x_0 + x_1)	{}
6	Add_HackersDelightRule_4	((0x2 * (x_0 | x_1)) - (x_0 ^ x_1)) => (x_0 + x_1)	{}
7	Add_HackersDelightRule_5	((0x2 * ((x_0 | x_1) | x_2)) - (x_0 ^ (x_1 | x_2))) => (x_0 + (x_1 | x_2))	{}
8	Add_OllvmRule_1	(~((x_0 ^ x_1)) + (0x2 * (x_1 | x_0))) => ((x_0 + x_1) - val_1)	{}
9	Add_OllvmRule_2	(~((x_0 ^ x_1)) - (val_fe * (x_0 | x_1))) => ((x_0 + x_1) - val_1)	{}
10	Add_OllvmRule_3	((x_0 ^ x_1) + (0x2 * (x_0 & x_1))) => (x_0 + x_1)	{}
11	Add_OllvmRule_4	((x_0 ^ x_1) - (val_fe * (x_0 & x_1))) => (x_0 + x_1)	{}
12	Add_SpecialConstantRule_1	((x_0 ^ c_1) + (0x2 * (x_0 & c_2))) => (x_0 + c_1)	{}
13	Add_SpecialConstantRule_2	(((x_0 & 0xff) ^ c_1) + (0x2 * (x_0 & c_2))) => (x_0 + c_1)	{}
14	Add_SpecialConstantRule_3	((x_0 ^ c_1) + (0x2 * (x_0 | c_2))) => (x_0 + val_res)	{}

	Name	Description	Configuration
1	Unflattener	Remove control flow flattening generated by OLLVM	{}
2	JumpFixer	No description available	{"enabled_rules": ["CompareConstantRule1", "CompareConstantRule2", "CompareConstantRule3", "JaeRule1", "JbRule1", "JnzRule1", "JnzRule2", "JnzRule3", "JnzRule4", "JnzRule5", "JnzRule6", "JnzRule7", "JnzRule8"]}

然后混淆基本去除完毕了

发现cry1和cry2只是RC4加密的初始化密钥和加密操作

这里key = 'keys_LLVM'

```
  v10 = 0;
  printf("Do you know LLVM?\n");
  v9 = 0;
  scanf("%s", input);
  memset(s, 0, 0x100uLL);
  memcpy(v7, "keys_LLVM", sizeof(v7));
  v3 = strlen((const char *)v7);
  cry1(s, v7, v3);
  v4 = strlen((const char *)(unsigned int)input);
  cry2(s, (unsigned __int8 *)input, v4);
  v11 = strlen((const char *)(unsigned int)input);
  if ( v11 == 23 && memcmp(input, &enc, 0x17uLL) == 0 )
  {
    if ( y_7 >= 10 && y_7 < 10 )
      goto LABEL_13;
    while ( 1 )
    {
      printf("Congratulations~\n");
      if ( y_7 < 10 || y_7 >= 10 )
        break;
LABEL_13:
      printf("Congratulations~\n");
    }
  }
  else
  {
    if ( y_7 < 10 || y_7 >= 10 )
      goto LABEL_10;
    do
    {
      printf("Sorry try again.\n");
LABEL_10:
      printf("Sorry try again.\n");
    }
    while ( y_7 >= 10 && y_7 < 10 );
  }
  return v10;
}
```

然后提取出enc的数据就直接解密就好了

exp

```
import binascii
from Crypto.Util.number import *
def rc4_crypt(text,key):
    textlen=len(text)
    keylen=len(key)
    ciper=[]
    count=0
    s=list(range(256))
    for i in range(256):
        count=(count+s[i]+key[i%keylen])%256
        s[i],s[count]=s[count],s[i]
    i=0
    j=0
    for m in range(textlen):
        i=(i+1)%256
        j=(j+s[i]+1)%256
        s[i],s[j]=s[j],s[i]
        k=s[(s[i]+s[j])%256]
        ciper.append(k^text[m])
    ciper_text=''.join("%02x"%i for i in ciper)
    return ciper_text.upper()
```

```
if __name__ == "__main__":
    s =
[0x54,0x89,0x61,0x10,0xd7,0xdf,0x33,0x11,0xc0,0x43,0xdf,0x76,0xdf,0x28,0xda,0xe6
,0x13,0x45,0x81,0x6d,0x79,0x18,0xc7]
    data = 'BE3924244CD030697D071E65BEBE5BF050EE84D2C94611'
    key = '6b6579735f4c4c564d'
    print("rc4 result:", rc4_crypt(binascii.a2b_hex(data),
binascii.a2b_hex(key.upper())))
    print(long_to_bytes(0x4D4F43534354467B6D315F4C4C564D5F7730726C64217D))
```