BUUCTF-RE-0x04

原创

AngelSnow1129 于 2021-05-10 20:09:43 发布 224 收藏

分类专栏： Reserve CTF 刷题

本文链接：https://blog.csdn.net/qq_56336850/article/details/116608397


Reserve 同时被 3 个专栏收录

3 篇文章 0 订阅

订阅专栏


CTF

4 篇文章 0 订阅

订阅专栏


刷题

6 篇文章 0 订阅

订阅专栏

---

title: BUUCTF-RE-0x04
date: 2021-05-10 19:58:34
tags:

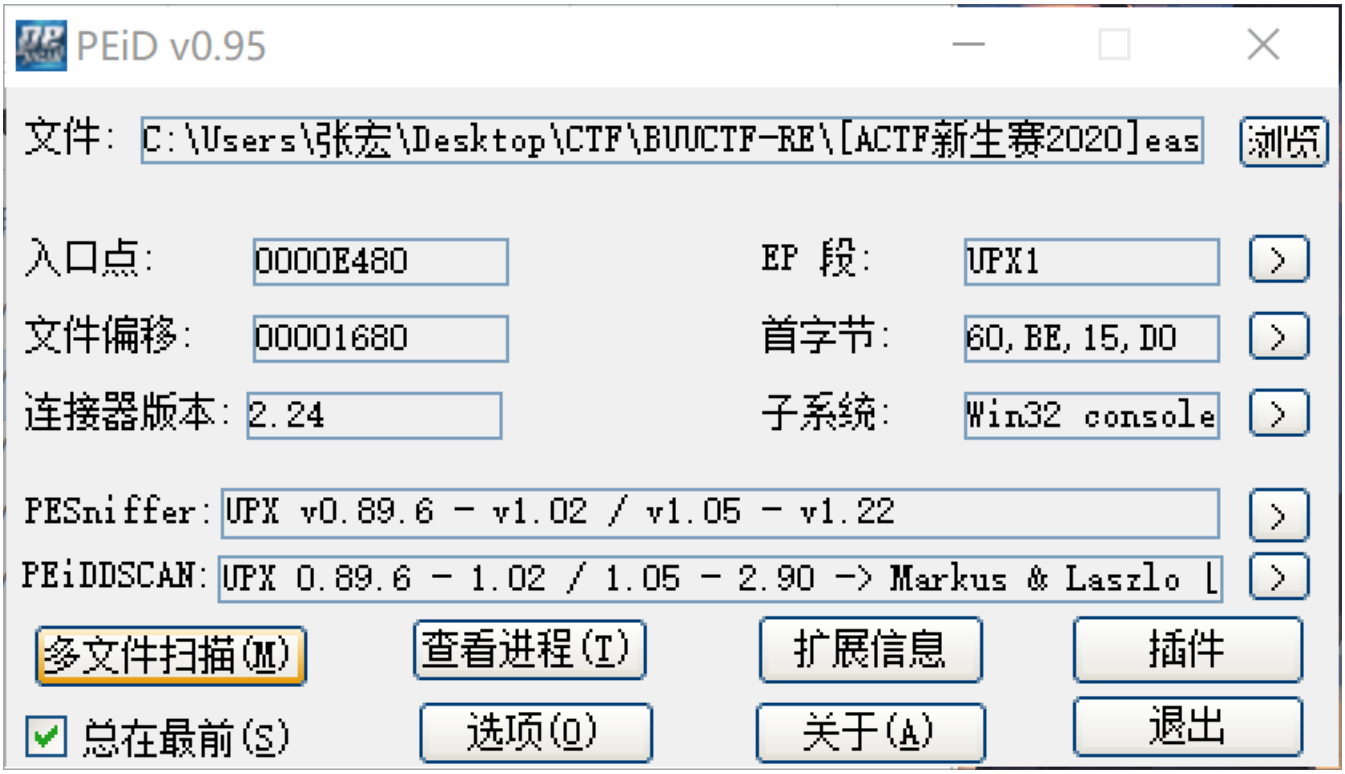
刷题记录。狗头~~

[ACTF新生赛2020]easyre

首先看题

啥都没有，下载下来看见有两个运行文件，有一个是可以使用的。

查壳

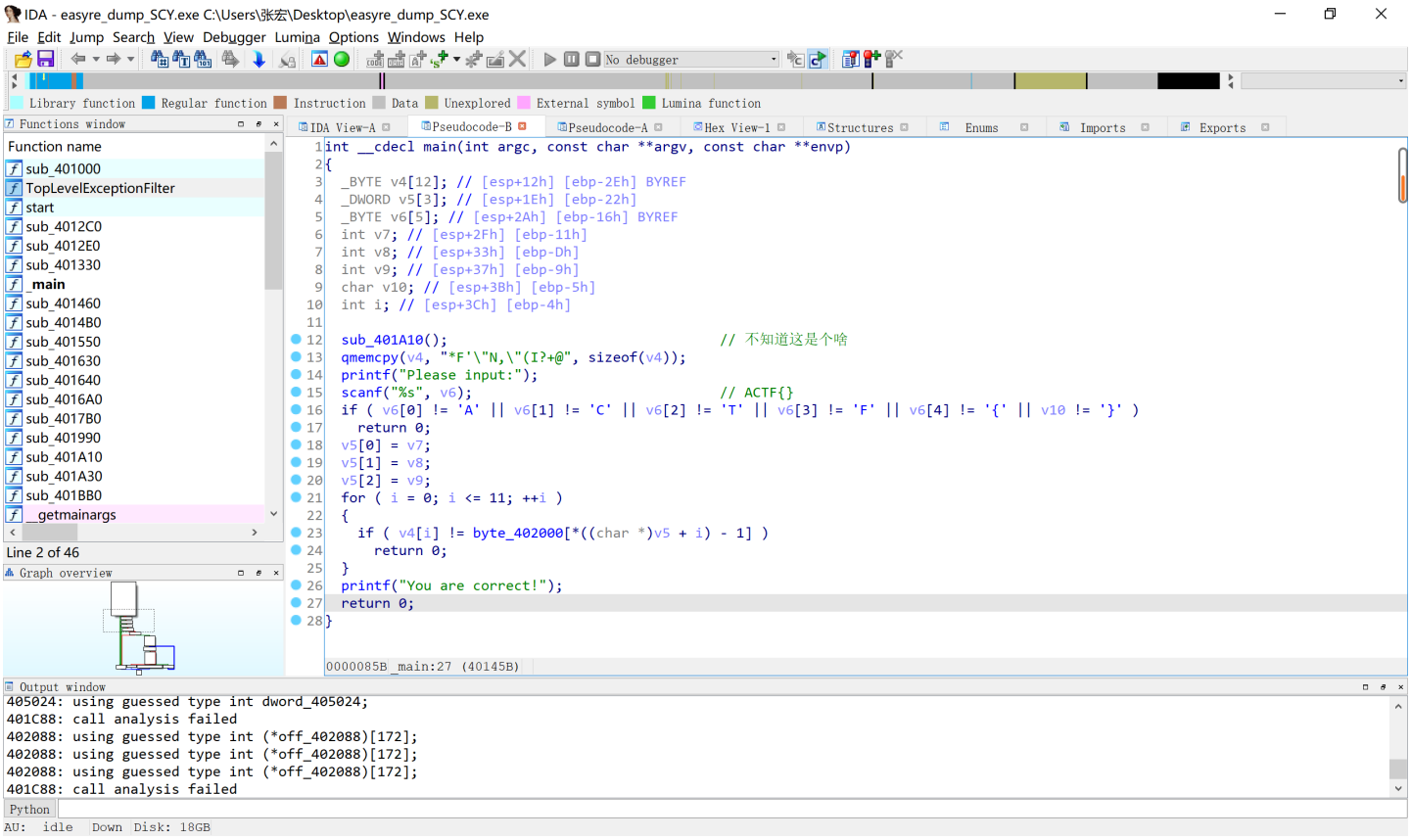


UPX壳，因为会脱压缩壳了。所以就脱吧，没有用脱壳机。

放入OllyDbg脱壳失败了，地址直接是乱的。然后放入的是X64dbg脱得壳，这个就很明显。

使用ESP定律把壳脱了。然后修复之后，可以运行了。

放入IDA然后简单分析之后就做不来了。发现这些数据可能有问题。



然后查看思考，可能是我脱壳没有脱干净。可是直接使用脱壳机脱得壳反汇编的结果也是一样的。所以可能就是这样的。前面显示的是一个读不懂的函数，然后后面的比较应该是进行比较。可是跟flag好像没啥关系，也跟输入的内容没啥关系。所以这里就有点不是很懂了。

```
  char v10; // [esp+3Bh] [ebp-5h]
  int i; // [esp+3Ch] [ebp-4h]

  sub_401A10();                                 // 不知道这是个啥
  qmemcpy(v4, "*F'\"N,\"(I?+@", sizeof(v4));
  printf("Please input:");
  scanf("%s", v6);                              // ACTF{}
  if ( v6[0] != 'A' || v6[1] != 'C' || v6[2] != 'T' || v6[3] != 'F' || v6[4] != '{' || v10 != '}' )
    return 0;
  v5[0] = v7;
  v5[1] = v8;
  v5[2] = v9;
  for ( i = 0; i <= 11; ++i )
  {
    if ( v4[i] != byte_402000[*((char *)v5 + i) - 1] )
      return 0;
  }
  printf("You are correct!");
  return 0;
}
```

char[15]

应该是12啊，这里是15，迷糊

```
# 主要的一个函数的解析
int __cdecl main(int argc, const char **argv, const char **envp)
{
  _BYTE v4[12]; // [esp+12h] [ebp-2Eh] BYREF
  _DWORD v5[3]; // [esp+1Eh] [ebp-22h]
  _BYTE v6[5]; // [esp+2Ah] [ebp-16h] BYREF
  int v7; // [esp+2Fh] [ebp-11h]
  int v8; // [esp+33h] [ebp-Dh]
  int v9; // [esp+37h] [ebp-9h]
  char v10; // [esp+3Bh] [ebp-5h]
  int i; // [esp+3Ch] [ebp-4h]

  sub_401A10();                                 // 不知道这是个啥
  qmemcpy(v4, "*F'\"N,\"(I?+@", sizeof(v4));
  printf("Please input:");
  scanf("%s", v6);                              // ACTF{}
  if ( v6[0] != 'A' || v6[1] != 'C' || v6[2] != 'T' || v6[3] != 'F' || v6[4] != '{' || v10 != '}' )
        //这里的字符是手动改变过的
    return 0;
  v5[0] = v7;
  v5[1] = v8;
  v5[2] = v9;
  for ( i = 0; i <= 11; ++i )
  {
    if ( v4[i] != byte_402000[*((char *)v5 + i) - 1] )// v4 = 数组[flag[i] -1]

      return 0;
  }
  printf("You are correct!");
  return 0;
}
```

后来查看文档，有搜索python字符串里面需要转义字符，就是需要斜杠来加入到字符串里面。所以将V4写成这样才对

```
# 原来的v4
v4  = "*F'\"N,\"(I?+@"
# 改一下
v4  = "*F\'\"N,\"(I?+@"
```

然后就是12个字符了，不过有点奇怪的是为什么IDA显示说是char【15】，而只有14个呀，所以可能数据末尾可能是有换行符的吧。暂时这么猜想的。

接下来，就开始写逆向的算法。

可能是因为python不熟的缘故，所以算法最后的结果如下。然后中间还有一个就是在使用Shift+E将数据到处的时候，好像不知道怎么就把其中一个空格给删不见了。这个就有点奇特了。

flag

```
# easyre
v4  = "*F\'\"N,\"(I?+@"

# list = "~}|{zyxwvutsrqponmlkjihgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>=<;:9876543210/.-,+*)(\x27&%$# ! \""
list = "~}|{zyxwvutsrqponmlkjihgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>=<;:9876543210/.-,+*)(\x27&%$# !\""


flag =""

for i in v4:
    temp = list.find(i)+1
    flag += chr(temp)

print("flag{"+flag+"}")   # flag{U9X_1S_W6@T?}
print("ACTF{"+flag+"}")   # ACTF{U9X_1S_W6@T?}
```

这个之间踩得坑还是挺多的。

OK，解决了。

[GWCTF 2019]pyre

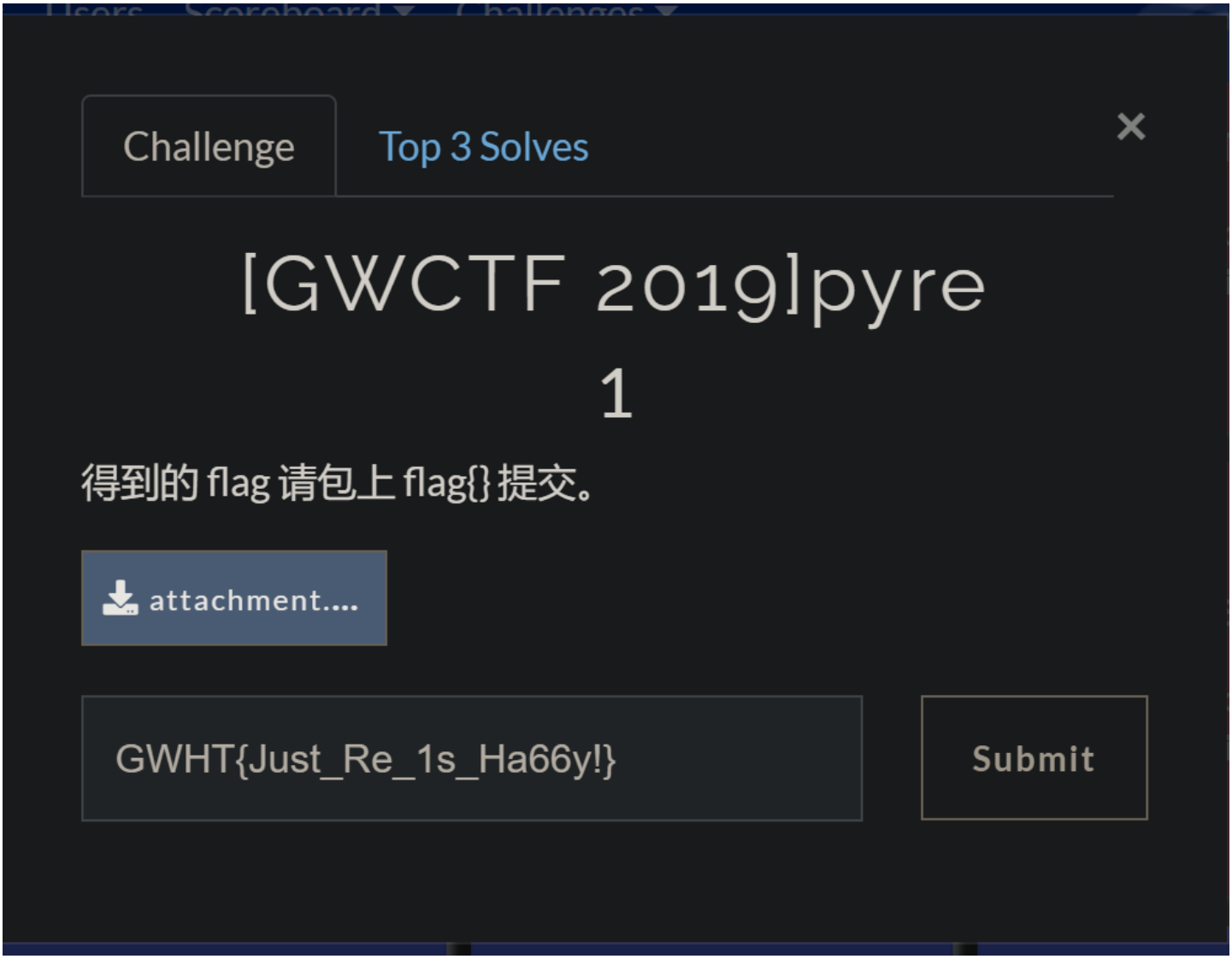
工具和环境

```
uncompyle6      #一个将pyc文件还原成py文件的工具
python3         #看着代码，把他写出来
```

小知识点

做完之后，看见的知识点应该考察的是算法逆向过程中的**异或运算**。即一个数两次异或得到自己本身。

题目




```
[GWCTF 2019]pyre
```

得到的 `flag` 请包上 `flag{}` 提交。

下载

下载得到的是一个pyc的文件，所以就使用uncompyle6 将这个编译的文件，反汇编得到**py文件**。

源码如下

```
# uncompyle6 version 3.7.4
# Python bytecode 2.7 (62211)
# Decompiled from: Python 3.7.8 (tags/v3.7.8:4b47a5b6ba, Jun 28 2020, 08:53:46) [MSC v.1916 64 bit (AMD64)]
# Embedded file name: encode.py
# Compiled at: 2019-08-19 21:01:57
print 'Welcome to Re World!'
print 'Your input1 is your flag~'
l = len(input1)
for i in range(l):
    num = ((input1[i] + i) % 128 + 128) % 128
    code += num

for i in range(l - 1):
    code[i] = code[i] ^ code[(i + 1)]

print code
code = ['\x1f', '\x12', '\x1d', '(', '0', '4', '\x01', '\x06', '\x14', '4', ',', '\x1b', 'U', '?', 'o', '6', '*'
, ':', '\x01', 'D', ';', '%', '\x13']
```

算法逆向（python3）

```
code = ['\x1f', '\x12', '\x1d', '(', '0', '4', '\x01', '\x06', '\x14', '4', ',', '\x1b', 'U', '?', 'o', '6', '*'
, ':', '\x01', 'D', ';', '%', '\x13']

l = len(code)
# print(l)
for i in range(l-2,-1,-1):
    code[i] = chr(ord(code[i]) ^ ord(code[(i + 1)]))

# print(code)
flag = ""

for i in range(l):
    num = ord(code[i]) - i
    if num <0:
        num+=128;
    flag +=chr(num)

print(flag)
# GWHT{Just_Re_1s_Ha66y!}
```

得到flag

```
GWHT{Just_Re_1s_Ha66y!}
```

因为题目要求提交flag形式的。所以还要改一下

```
flag{Just_Re_1s_Ha66y!}
```

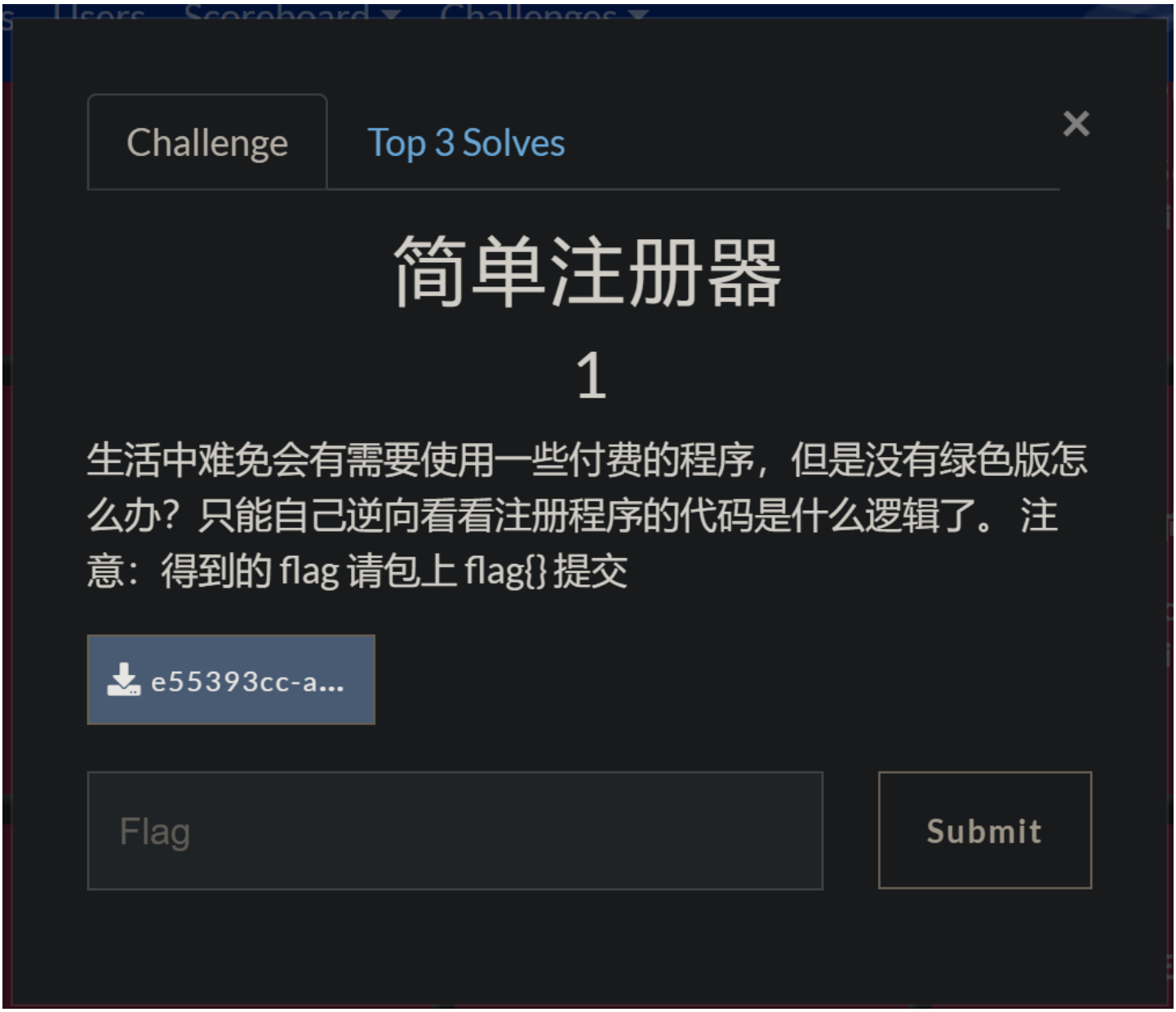
这下对了。

简单的注册器

工具及环境

```
jadk_v1.2.0   #用来反汇编apk文件
python       #计算出flag
ApkScan-PKID.jar      #查壳工具
```

题目



要求

```
简单注册器
生活中难免会有需要使用一些付费的程序，但是没有绿色版怎么办？只能自己逆向看看注册程序的代码是什么逻辑了。 注意：得到的 flag 请包上 flag{} 提交
```

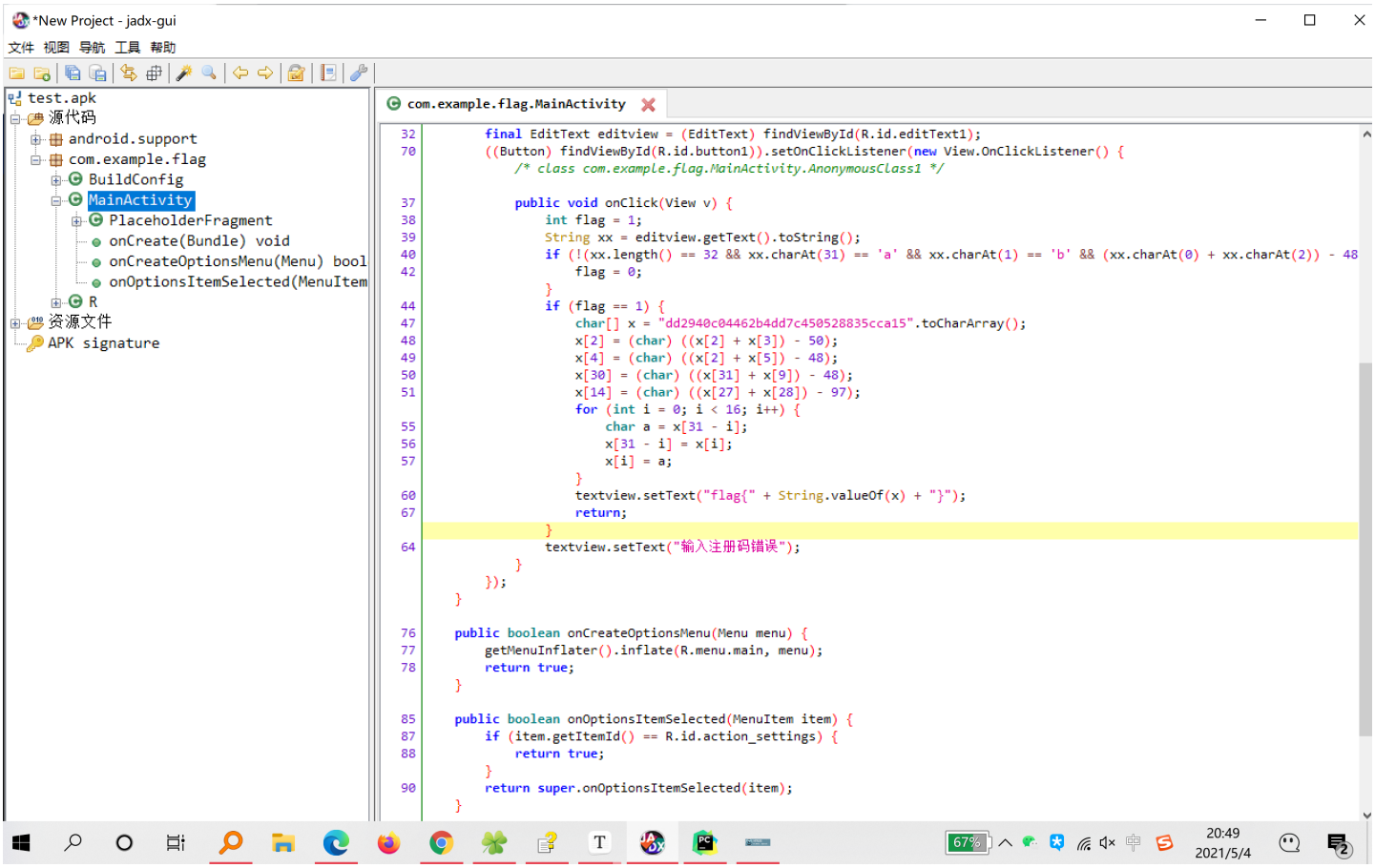
下载

下载后看见是一个APK，所以尝试着查下壳。

查壳

用了查壳的软件没有查到壳。

使用jadx反汇编查看（这个工具感觉是非常友好的软件），在这个上面的安卓应用反汇编的更强吧。



进入主函数就看见了我们需要的东西。所以接下来。就照着这个写就好了。

python3写的

```
str = "dd2940c04462b4dd7c450528835cca15"
# 将str转化为list可以使用下标来处理。
list = []
for j in str:
    list.append(j)
x = list   # 处理了之后，就用原来的函数来处理
x[2] =  chr(ord(x[2])+ ord(x[3]) - 50 )
x[4] =  chr(ord(x[2])+ ord(x[5]) - 48);
x[30] = chr(ord(x[31])+ ord(x[9]) - 48);
x[14] = chr(ord(x[27])+ ord(x[28]) - 97);

for i in range(0,16):
    s = x[31-i]
    x[31-i] = x[i]
    x[i] = s

str2 = "".join(list)   # 处理完之后，将原来的改成字符串打印输出
flag = ''
flag = flag+"flag{"+str2+"}"
print(flag)
# print(x)
```

然后运行就出来了。

flag

```
flag{59acc538825054c7de4b26440c0999dd}
```

[BJDCTF2020]JustRE

工具以及环境

```
Exeinfo PE        # 查壳
```

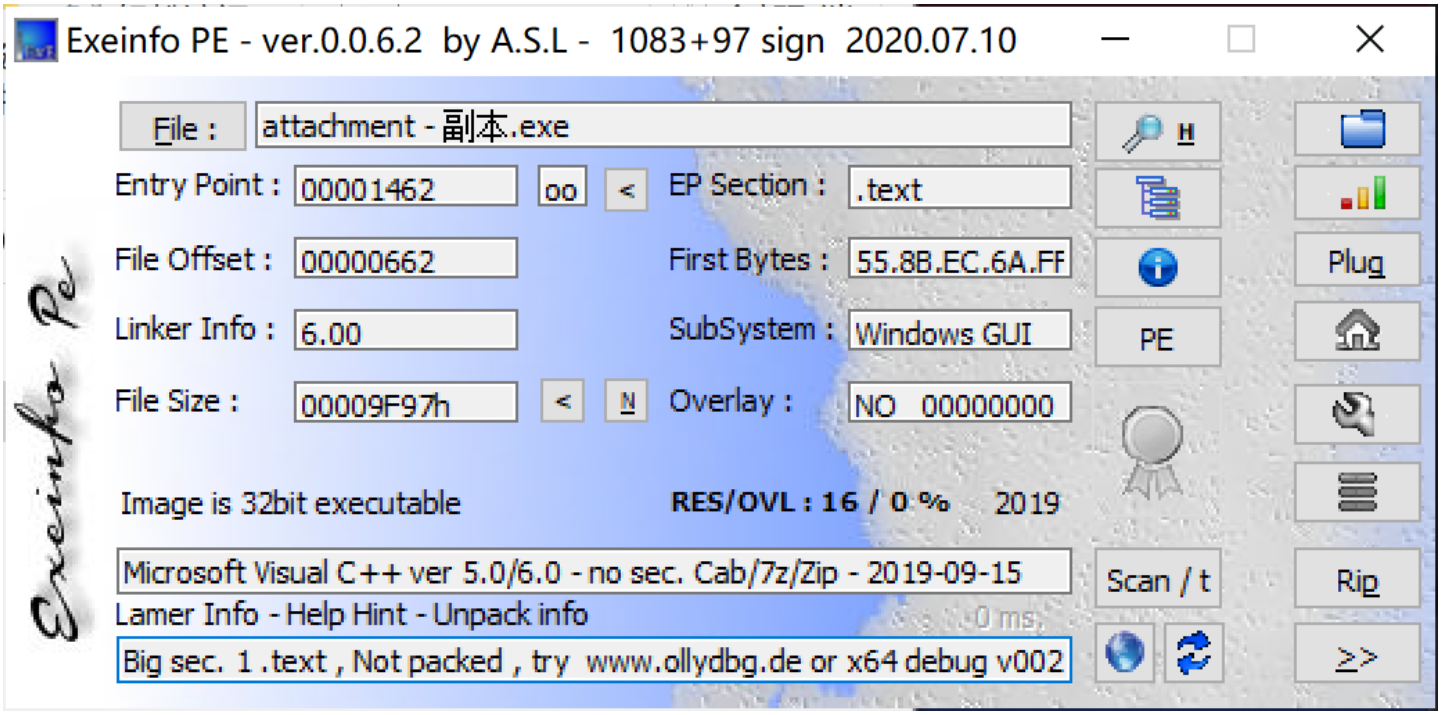
题目及描述

```
[BJDCTF2020]JustRE

得到的 flag 请包上 flag{} 提交。来源：
https://github.com/BjdsecCA/BJDCTF2020
```

下载 attachment.exe

查壳

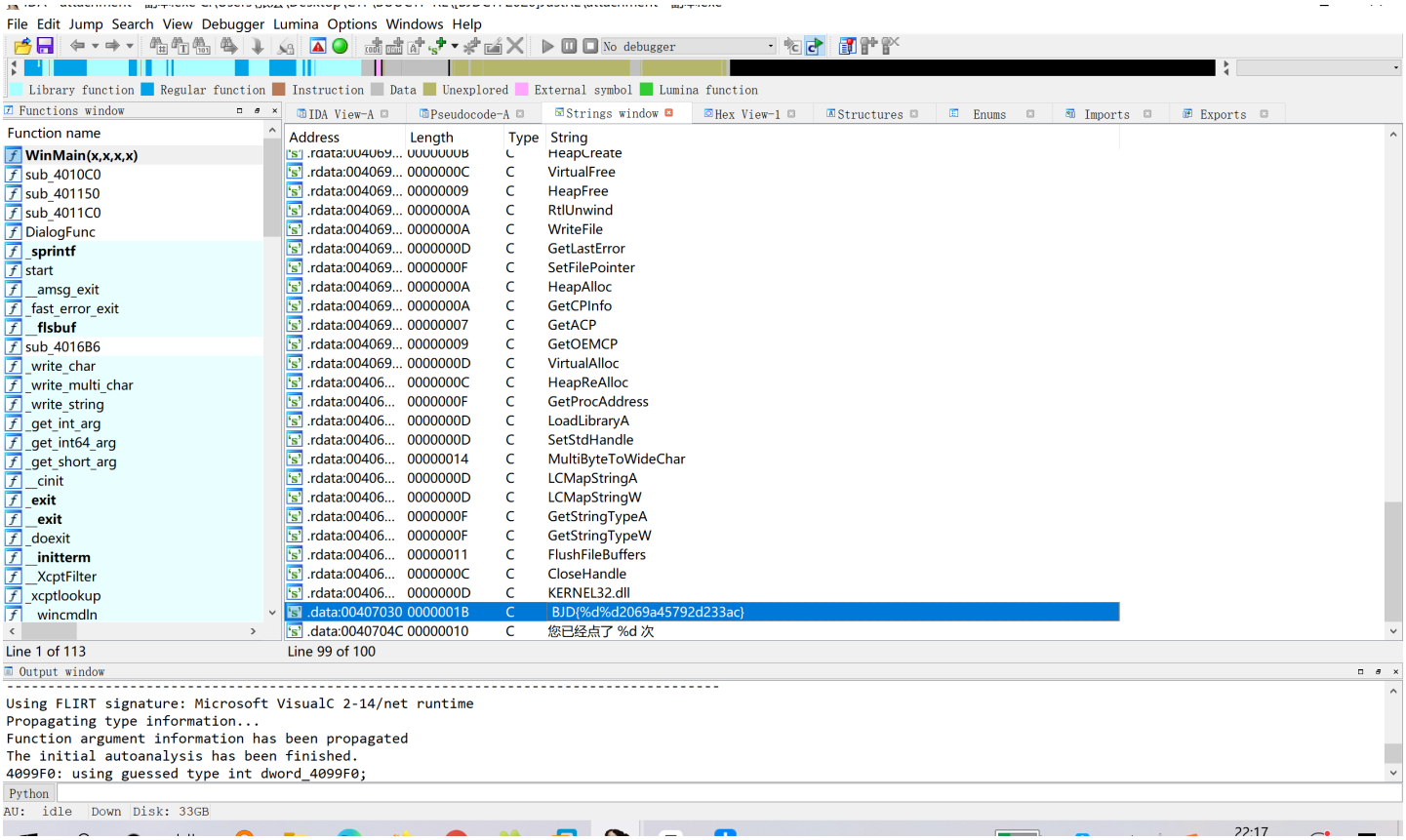


没有壳那么我们就考虑使用IDA和OD

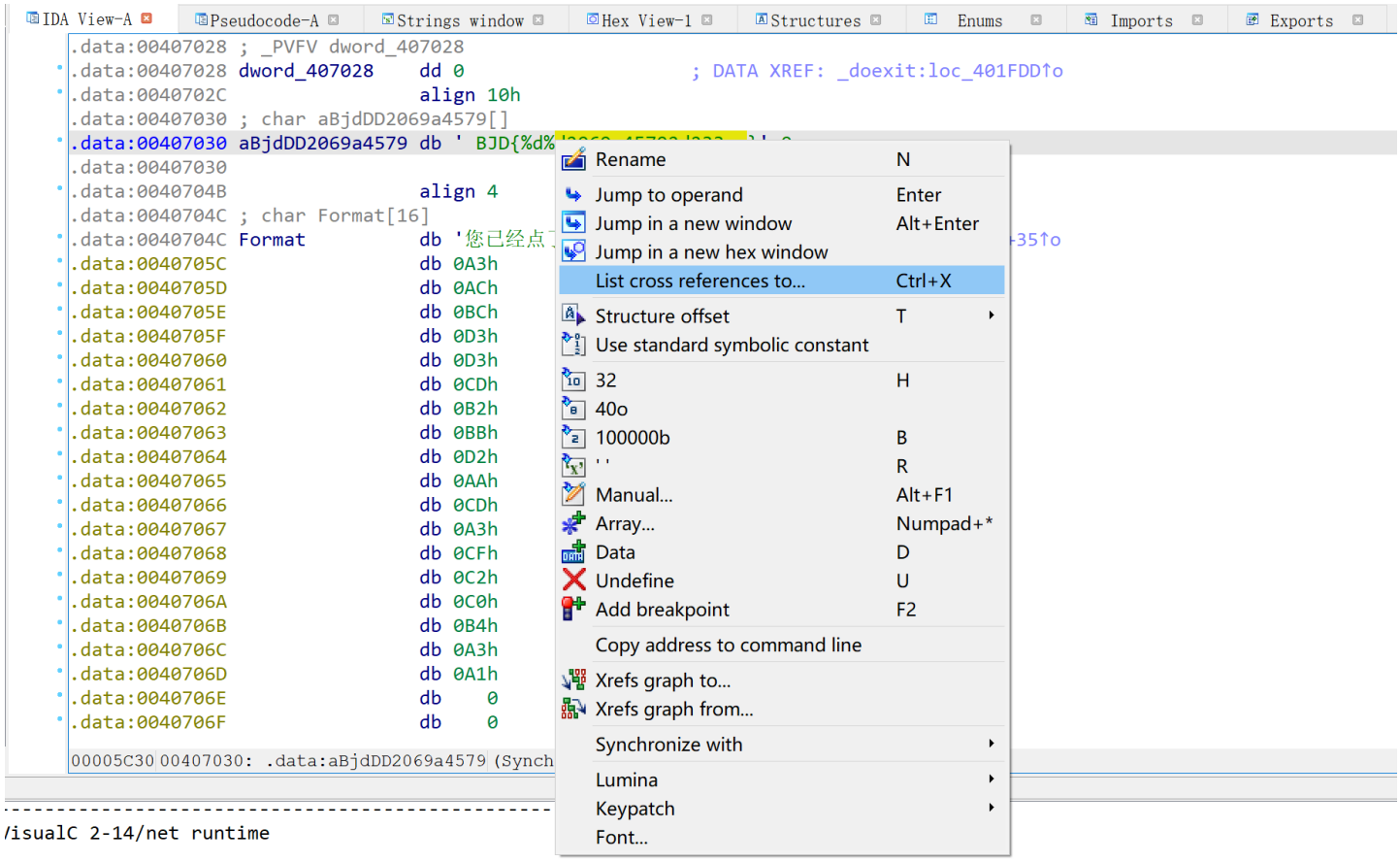
因为这里是32位的程序，先使用IDA32.

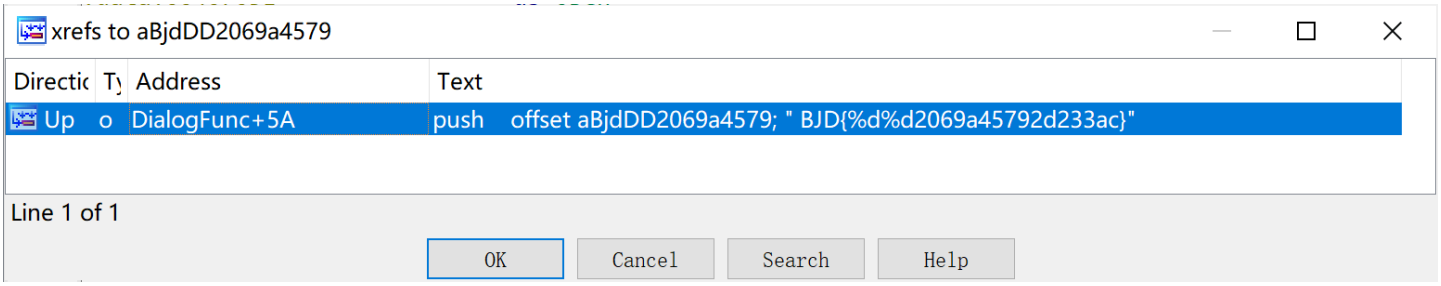
逆向过程

通过搜索字符串，然后发现字符串。



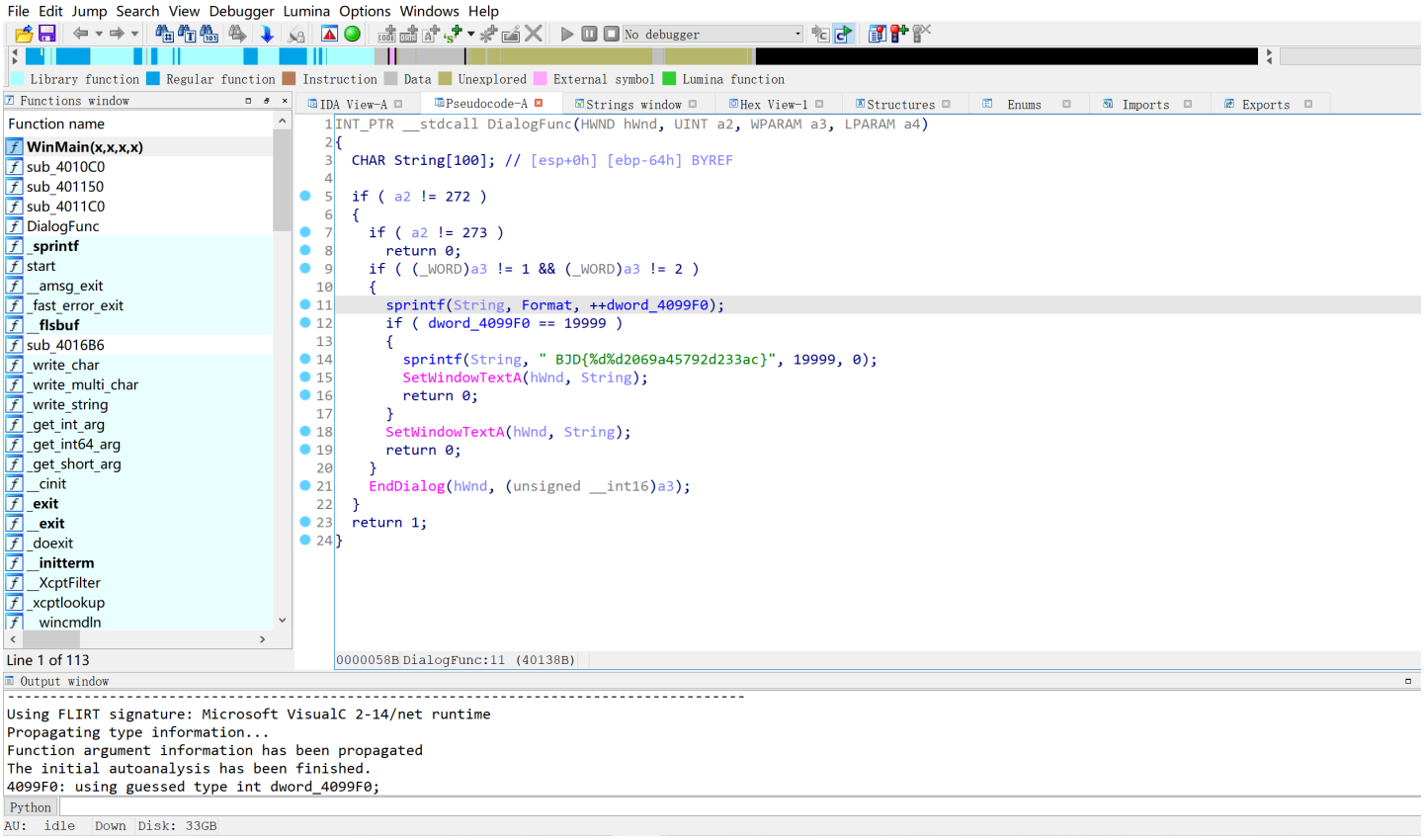
双击到达那里，然后选中后右键选择。





确定后点击确定。

可以猜出大概是这个意思。



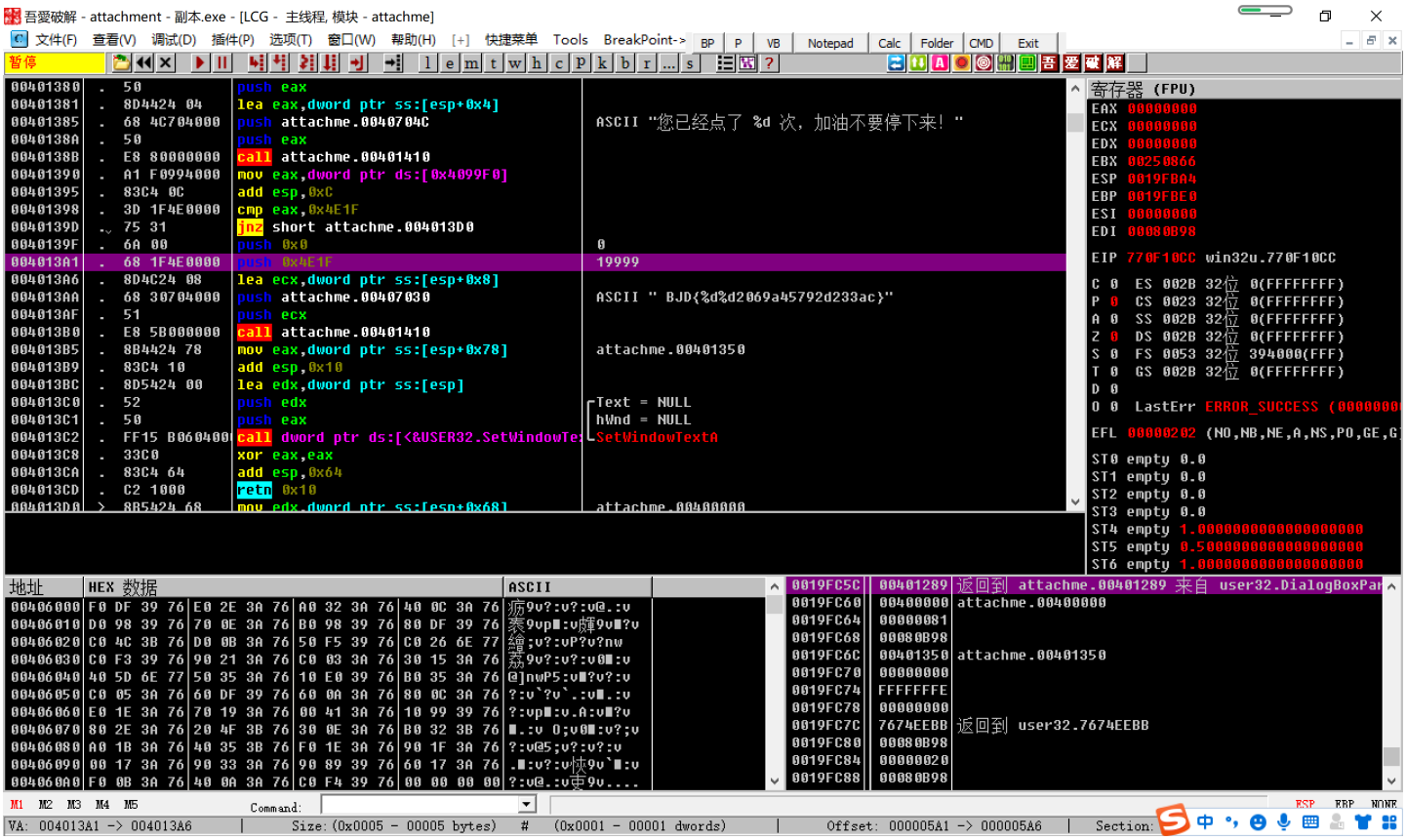
Flag

BJD{1999902069a45792d233ac}

然后在BUUCTF里面需要将格式修改一下变成
flag{1999902069a45792d233ac}

知识点

这个使用动态调试的时候电脑死机了。所以就静态调试搞出来了。大概需要我们点击19999次。



总结

BJD{1999902069a45792d233ac}
所以
flag{1999902069a45792d233ac}