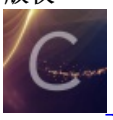
【jarvisoj刷题之旅】逆向题目DDCTF - Android Normal的writeup

原创

iqiqiya 于 2018-09-27 15:34:16 发布
分类专栏： 我的逆向之路 我的CTF之路 ------jarvisojCTF 我的CTF进阶之路 文章标签： 【jarvisoj刷题之旅】逆向题目DDCTF - Andr 逆向题目DDCTF - Android Normal的wri DDCTF - Android Normal的writeup Android Normal



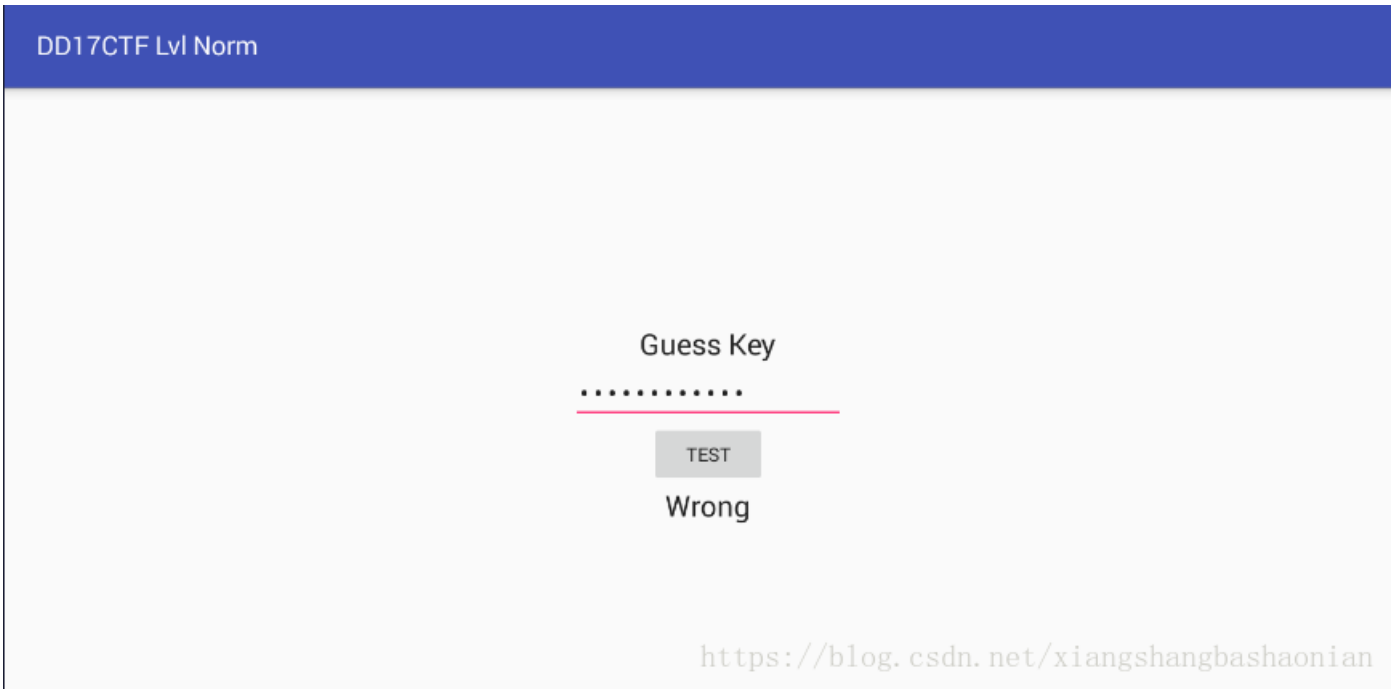
Android Normal

下载后输入解压密码进行解压　得到Readme.txt与DDCTF-Normal.apk

Readme.txt

```
1  赛题背景：本挑战结合了Android，Java，
   C/C++，加密算法等知识点，考察了挑战者的binary逆向技术和加密算法能力。
2
3  赛题描述：本题是一个app，请试分析app中隐藏的key，逆向加密算法并得到对应的秘钥
   。可以在app中尝试输入key，如果正确会显示"correct"，如果错误会显示"Wrong"。
4  提　　示：阅读assembly code，理解xor的加密逻辑和参数，解出答案。
5  评分标准：key正确则可进入下一题。
```


将apk载入模拟器运行（顺便吐槽下 蓝叠咋不能竖屏。。。）



输入123456789　出现Wrong

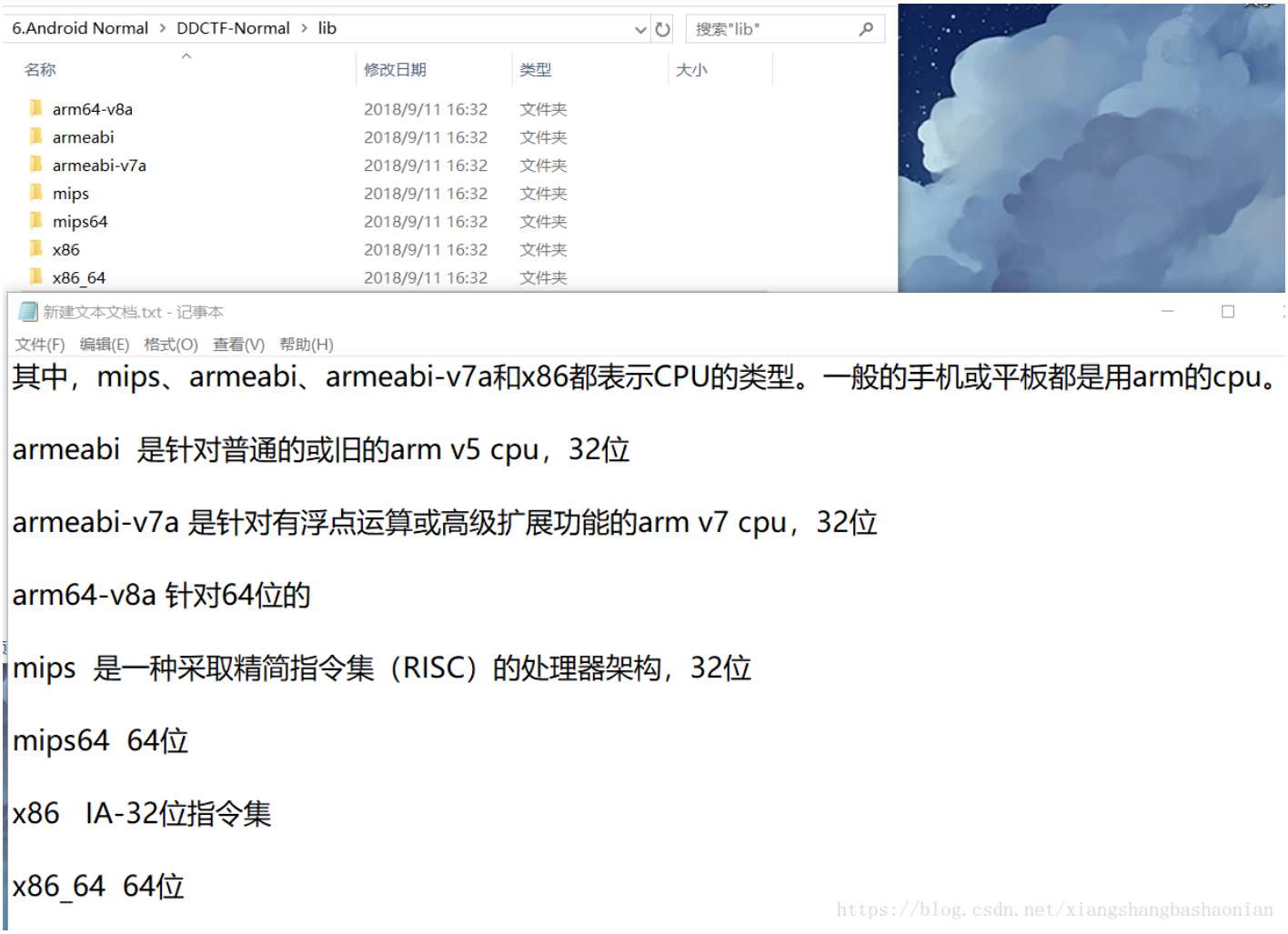
载入jeb

反编译成java代码分析

程序流程很简单　就是一个简单的字符串比较 只不过要对比的字符串放进了native层

这个stringFromJNI()函数就是要在Java代码中调用的Native函数

那么就要IDA载入分析so文件



因为前面有调用hello-libs.so

所以想找和这个名字一样的 结果发现每个文件夹里都有

这时候就蒙了

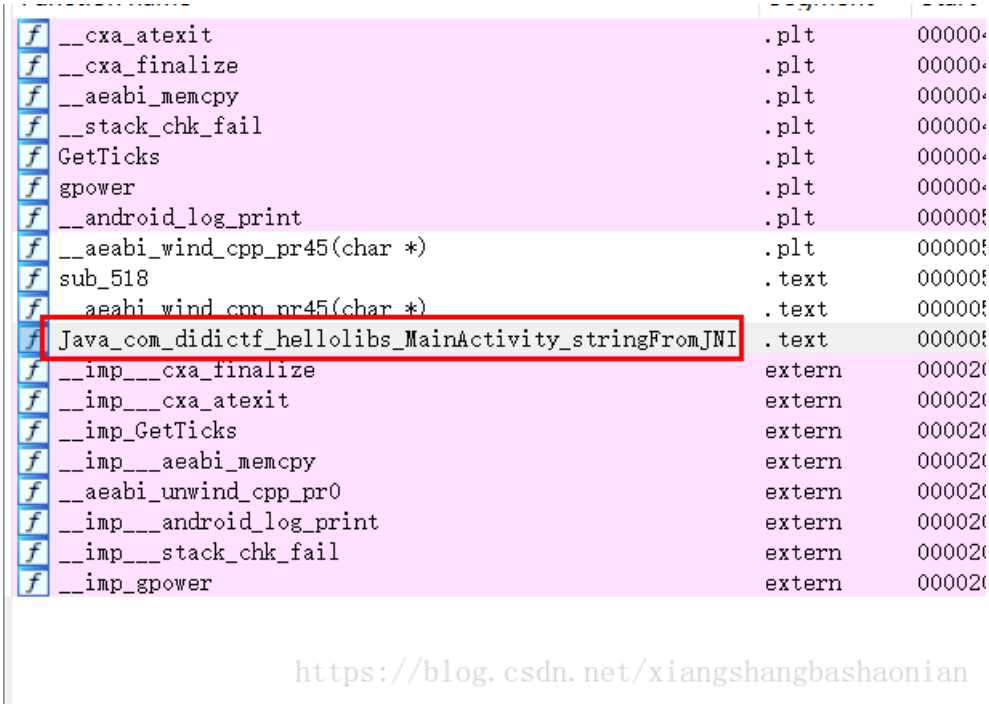
看了大佬的writeup

https://blog.csdn.net/wannafly1995/article/details/80954371

大佬说通常采用armeabi-v7a的so文件

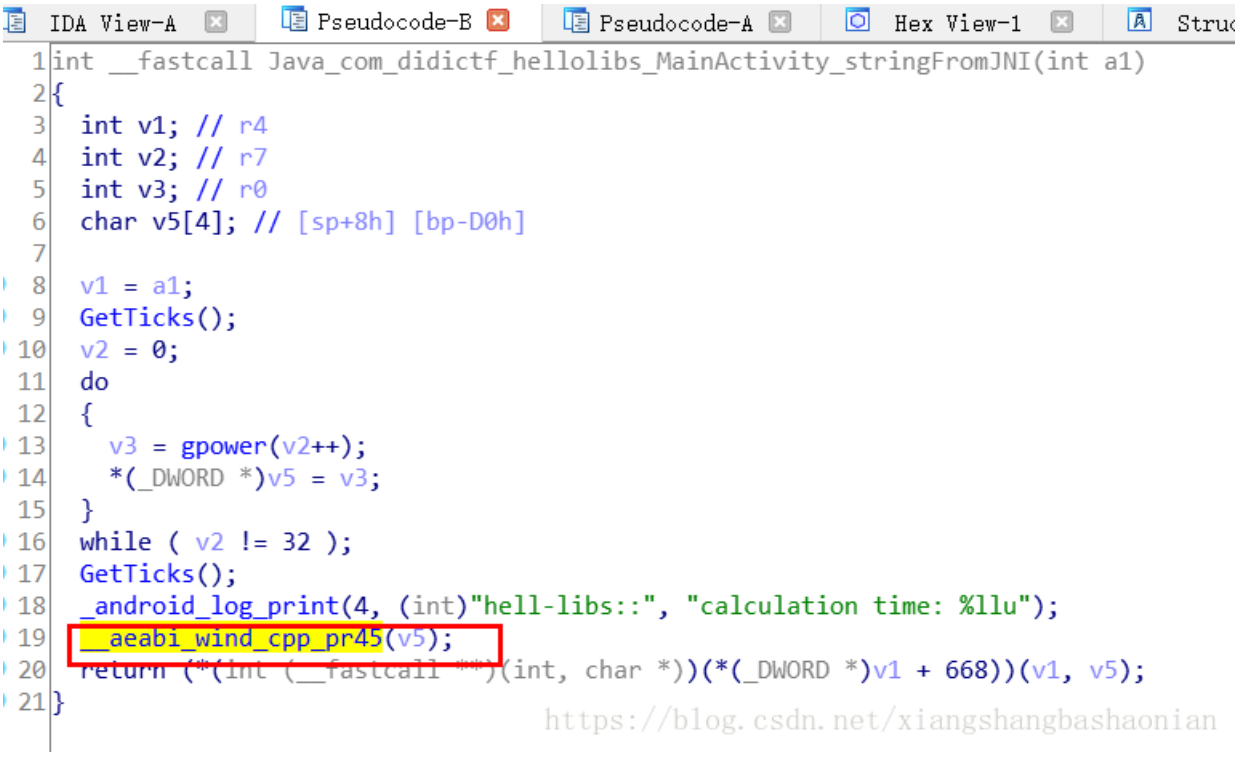
那就用它咯

可以发现



F5变成c代码

分析可知有用的函数就是他了__aeabi_wind_cpp_pr45()



双击再双击

IDA View-A | Pseudocode-B | Pseudocode-A | Hex View-1 | Structures | Enums | Imports

```
int __fastcall __aeabi_wind_cpp_pr45(char *a1)
{
  char *v1; // r4
  int i; // r1
  int j; // r3
  int v4; // r6
  int v5; // r5
  int v6; // r2
  char c[182]; // [sp+6h] [bp-CAh]
  int v9; // [sp+BCh] [bp-14h]

  v1 = a1;
  i = 0;
  do
  {
    c[i] = a[i] ^ b[i];
    ++i;
  }
  while ( i != 182 );
  j = 0;
  do
    v4 = c[(c[0] >> 1) + j++];                  // c[0] = a[0] ^ b[0]  所以c[0]=57  c[0] >> 1 =28
                                                // 因为当v4等于0时才退出循环  所以就是要找值是0 且下表在28以后的c数组中的元素
                                                // 那么我们将他们全部打印出来 就可以看到flag
  while ( v4 );
  v5 = j - 1;
  v6 = 0;
  if ( j - 1 >= 1 )
  {
    _aeabi_memcpy(a1, &c[c[0] >> 1], j - 1);
    v6 = v5;
  }
  v1[v6] = 0;
  return _stack_chk_guard - v9;
}
```

00000580 _Z21__aeabi_wind_cpp_pr45Pc:31 (580)

双击a[i]与b[i]就能得到内容 dump即可

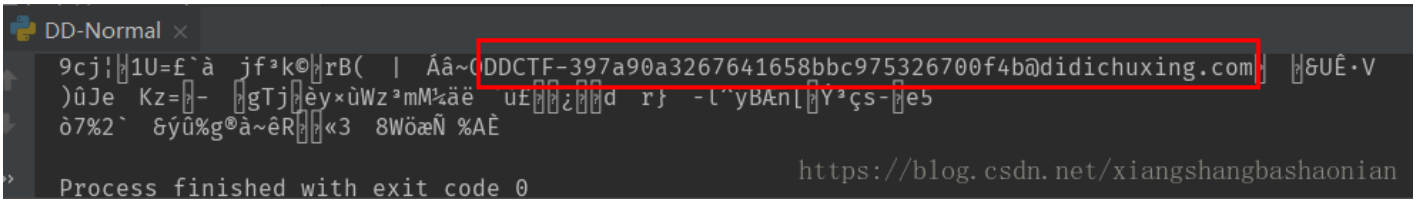
Py大法好：

```
a = [0xD8, 0xC2, 0x6B, 0x42, 0x82, 0x67, 0xC8, 0x4D, 0x7A, 0x95,

  0xE8, 0x81, 0x48, 0xC1, 0x9E, 0x40, 0xE8, 0xFB, 0xCF, 0xE6,

  0x4F, 0xBA, 0xE6, 0xAF, 0x78, 0x19, 0x6F, 0x9C, 0xE9, 0xF7,

  0x7A, 0xDD, 0x42, 0xCE, 0x8C, 0x03, 0xB8, 0x66, 0xD3, 0xAB,

  0x00, 0x7E, 0xDE, 0x3E, 0x53, 0xDE, 0x30, 0x91, 0x3D, 0xF7,

  0xCD, 0x72, 0x14, 0x51, 0x82, 0xEE, 0x1B, 0x8D, 0xB4, 0x8C,

  0xD0, 0x8A, 0xF6, 0x9A, 0x96, 0x71, 0x98, 0x62, 0x93, 0x4A,

  0x30, 0x2F, 0x9C, 0xA8, 0x79, 0x16, 0xC1, 0xE0, 0xEC, 0xD7,

  0xE5, 0xEC, 0x8A, 0x64, 0xB4, 0x46, 0xCF, 0xD9, 0xE5, 0x96,

  0xF3, 0x94, 0x73, 0xA9, 0xFF, 0xEA, 0xCB, 0x15, 0x9C, 0x7C,

  0xA1, 0xD8, 0x3E, 0xBB, 0x1D, 0x38, 0xCB, 0x55, 0xD0, 0x19,

  0x25, 0xB2, 0x0B, 0x92, 0xE8, 0x88, 0xAE, 0x06, 0xA2, 0x9B,

  0x93, 0x64, 0x5E, 0xFB, 0x09, 0x05, 0xF6, 0x2F, 0x1F, 0x35,

  0xCC, 0xEF, 0x05, 0x6C, 0x19, 0x42, 0x38, 0xA5, 0x59, 0x2E,

  0x80, 0x0A, 0x19, 0xEC, 0x33, 0x5B, 0xBB, 0xD6, 0xEB, 0x2B
```

```
  0x80, 0x0A, 0x19, 0xFC, 0x55, 0x5B, 0xDD, 0xD6, 0xED, 0x2B,
  0xAC, 0xF7, 0x0E, 0xAD, 0xD8, 0x57, 0x40, 0x98, 0x71, 0x2C,
  0x78, 0x68, 0x91, 0x82, 0x4F, 0x5B, 0xD6, 0x40, 0x8F, 0x03,
  0xBD, 0x55, 0x0B, 0x47, 0x3D, 0xF4, 0x5A, 0x49, 0x5B, 0xF2,
  0xA2, 0x9E]
b = [0xE1, 0xA1, 0x01, 0xE4, 0x82, 0x56, 0x9D, 0x70, 0xD9, 0xF5,
  0x08, 0x10, 0x22, 0xA7, 0x2D, 0x2B, 0x41, 0xF0, 0xBD, 0xA4,
  0x67, 0x3D, 0x9A, 0x20, 0xB9, 0xFB, 0x11, 0xD3, 0xAD, 0xB3,
  0x39, 0x89, 0x04, 0xE3, 0xBF, 0x3A, 0x8F, 0x07, 0xEA, 0x9B,
  0x61, 0x4D, 0xEC, 0x08, 0x64, 0xE8, 0x04, 0xA0, 0x0B, 0xC2,
  0xF5, 0x10, 0x76, 0x32, 0xBB, 0xD9, 0x2E, 0xBE, 0x86, 0xBA,
  0xE7, 0xBA, 0xC6, 0xFC, 0xA2, 0x13, 0xD8, 0x06, 0xFA, 0x2E,
  0x59, 0x4C, 0xF4, 0xDD, 0x01, 0x7F, 0xAF, 0x87, 0xC2, 0xB4,
  0x8A, 0x81, 0x8A, 0xF2, 0xB6, 0x60, 0x9A, 0x13, 0x52, 0xC0,
  0x6D, 0x9E, 0x5A, 0x52, 0xB5, 0x8F, 0x47, 0x5E, 0xE6, 0x41,
  0xAD, 0xF5, 0xBB, 0xA9, 0x7A, 0x6C, 0xA1, 0x4C, 0x38, 0x60,
  0xF2, 0x4B, 0x5C, 0xE8, 0x5B, 0xE5, 0xE3, 0xBA, 0x46, 0x70,
  0x33, 0x04, 0xA7, 0x58, 0x19, 0x10, 0x49, 0x20, 0x1D, 0x51,
  0x48, 0x9D, 0x78, 0xF9, 0xB4, 0x2E, 0x66, 0x58, 0x1B, 0xE8,
  0xEE, 0x51, 0x09, 0x21, 0x80, 0xBC, 0xC8, 0x7B, 0xF5, 0x4E,
  0x99, 0xFD, 0xFC, 0x9A, 0xFD, 0x65, 0x20, 0x13, 0x57, 0xD1,
  0x83, 0x4D, 0xF6, 0x2C, 0xAF, 0x25, 0x3C, 0x12, 0xF0, 0x7C,
  0x16, 0x66, 0x97, 0x7F, 0x6A, 0x02, 0xBC, 0x98, 0x52, 0xD7,
  0xE3, 0x56]
c = []
j = 0
flag = ''
for i in range(0,len(a)):
    c.append(a[i] ^ b[i])
    if c[i] == 0:
```

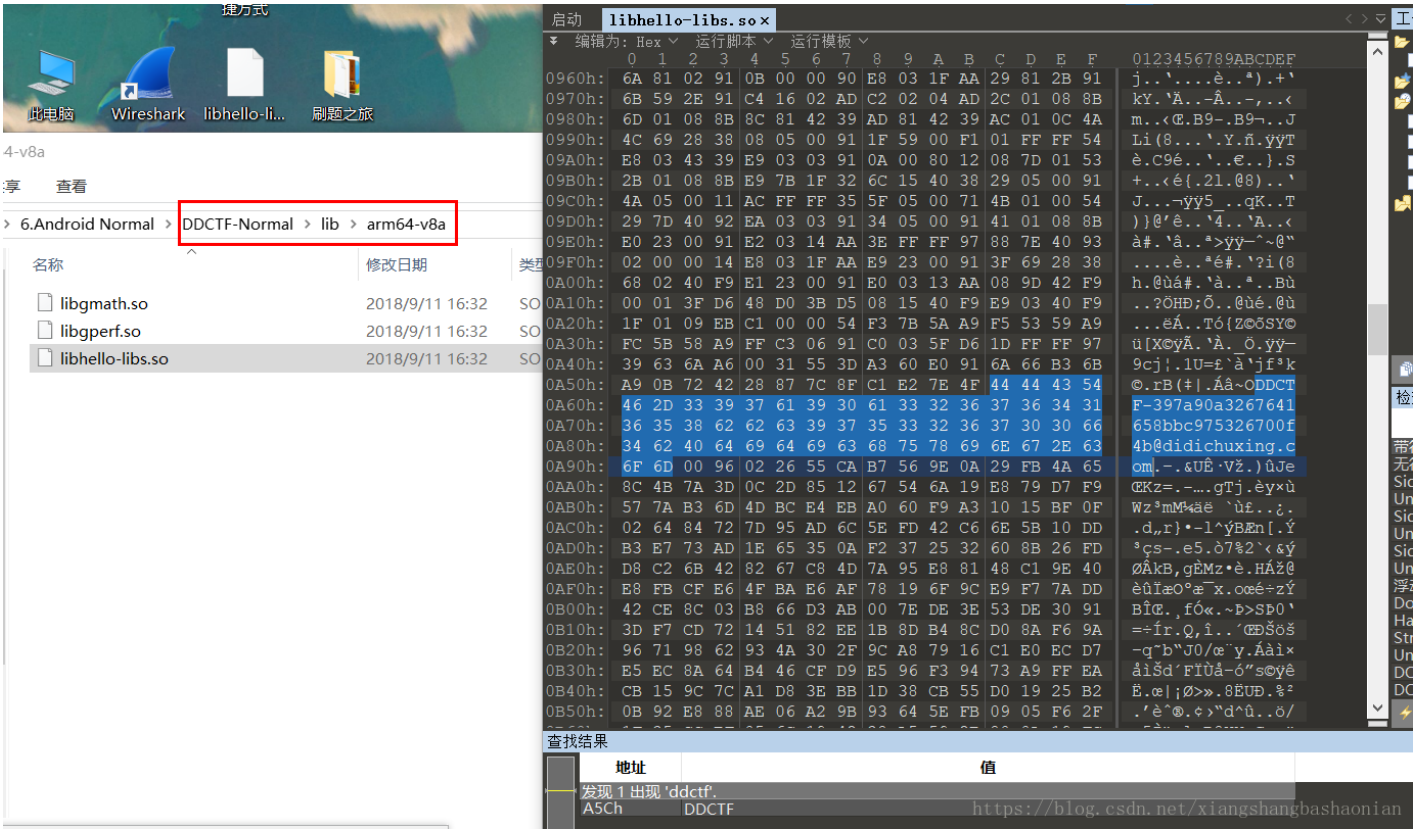
```
        print(i)

for i in c:

    flag += chr(i)

print(flag)
```

最后找到flag:



后来像再看看其他大佬思路：

十六进制编辑器打开\lib\arm64-v8a\libhello-libs.so居然直接可以看到flag。。。



\lib\x86\libhello-libs.so与

\lib\x86_64\libhello-libs.so中都可以直接发现flag

可怕。。。

更多的是疑惑 如果有哪位大佬 看到 可不可以麻烦给小弟讲解一下

还有就是可以改smile代码进行动态调试