Android加密 看雪,Android加密与解密入门两题

转载

胡俊琪 于 2021-06-02 16:08:31 发布 42 收藏
文章标签： Android加密 看雪

写在最前面

本次题目来自看雪2w班9月题。密码学一直是安全的基础，Android安全也不例外，这次9月份的题分别从java层和C层考察了密码学中常用的对称加密、hash函数以及一些基础的编码，但是不是单纯的算法分析题，可以说是很好的练习题了。

9月第一题

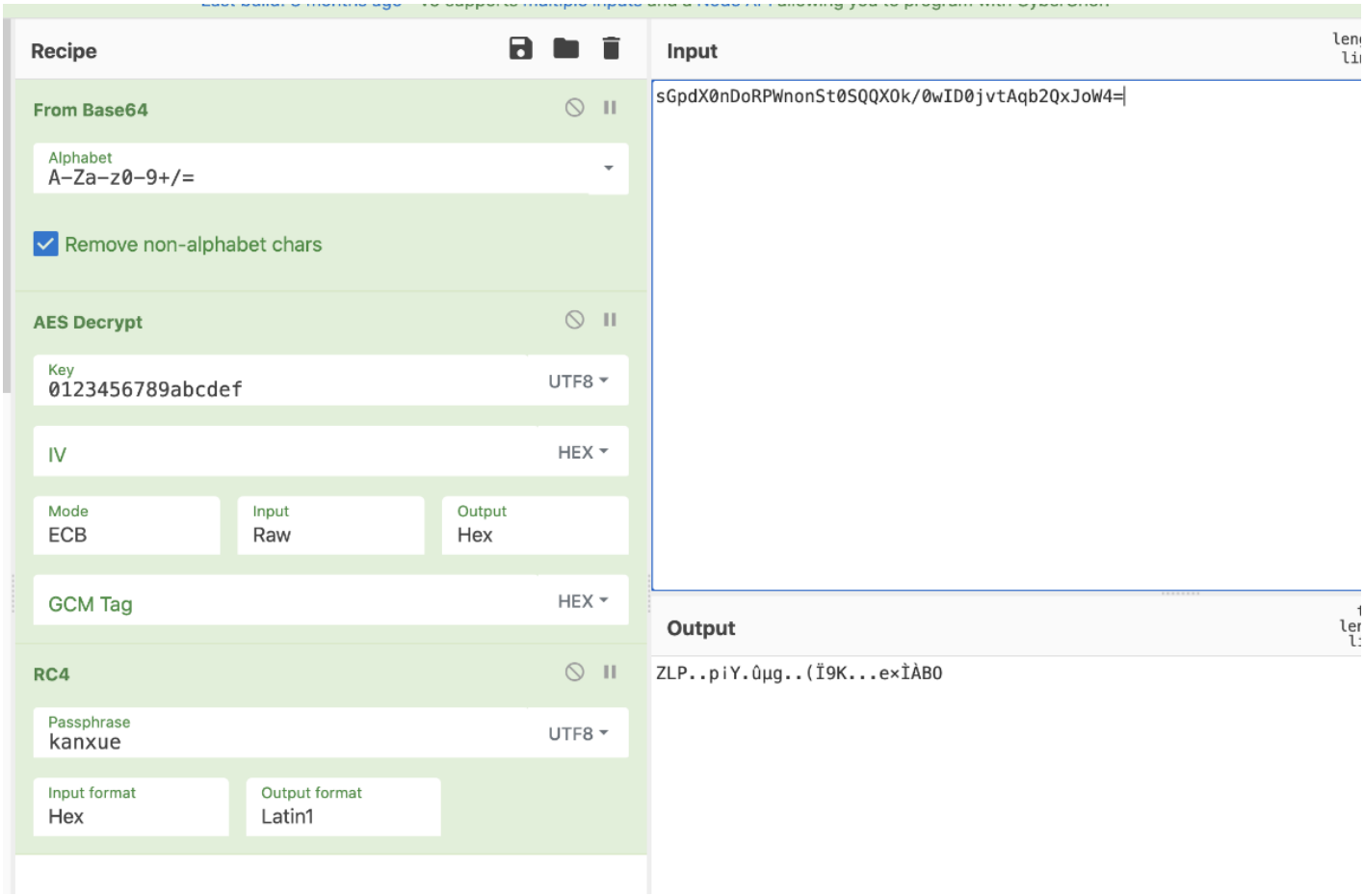
脱壳，脱壳后进行逆向，

```
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import com.stub.StubApp;

public class MainActivity extends AppCompatActivity {
    static {
        StubApp.interface11(0x562);
        System.loadLibrary("native-lib");
    }

    public MainActivity() {
        super();
    }

    protected native void onCreate(Bundle arg1) {
    }

    public native String stringFromJNI() {
    }
}
```

一开始感觉so文件完全没啥用，反而有一个Utils的类十分可疑

很明显的test函数是入口，然后调用bbbbb函数进行加密得到的返回值作为aaaaa函数的参数进行加密，最后确认是否等于Utils.Cipher

一开始一看 不是很明显嘛直接CyberChef，然而。。



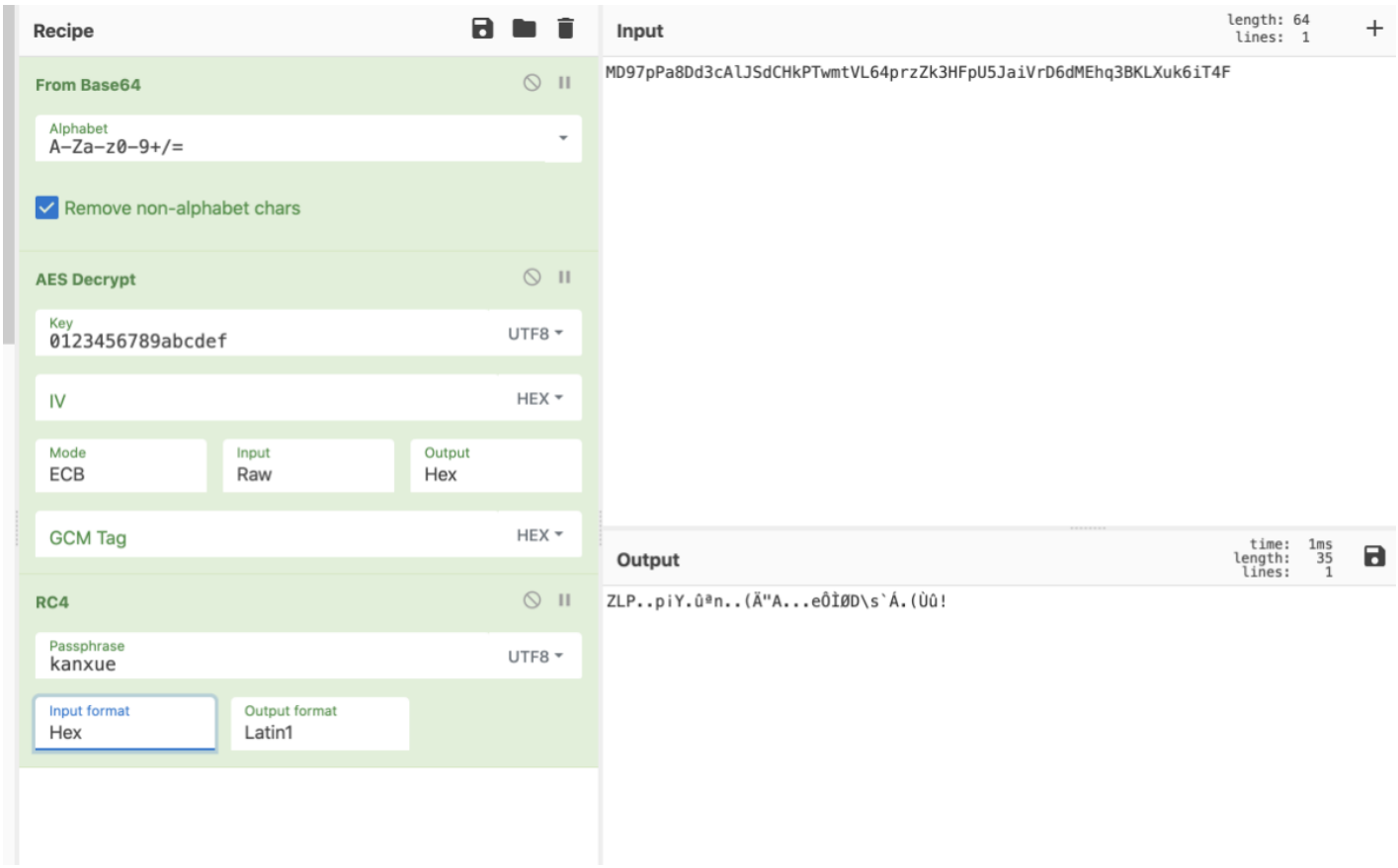
很明显不对。。想到之前寒冰师傅出的题，一定是动态修改，静态看的肯定不准

于是直接用Objection打印吧

```
com.kanxue.test on (Android: 8.1.0) [usb] # android heap print fields 0x2492
Handle 0x2492 is to class com.kanxue.test.Utils
Name    Value
------  ------------------------------------------------------------------
cipher  MD97pPa8Dd3cAlJSdCHkPTwmtVL64przZk3HFpU5JaiVrD6dMEhq3BKLXuk6iT4F
com.kanxue.test on (Android: 8.1.0) [usb] # android hooking watch class_method com.kanxue.test.U
```

果然。。。

再来CyberChef

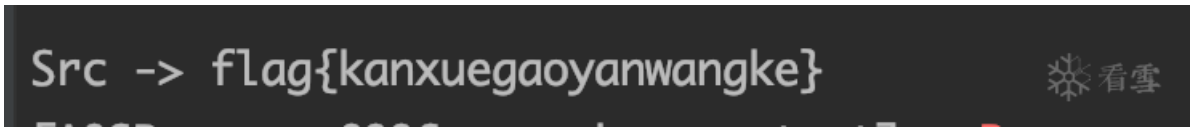


还是不对，难道不是也不是AES嘛 。。

后来经过主动调用发现AES是对的，那么RC4魔改了？？？？直接抠出来用Java工程跑一遍，确实和标准的RC4不一致。。

不过由于RC4这种是一个对称密码，那么我直接拿AES解密后的去再调用一次这个函数就行了。。

最终frida跑出来原来的正确的flag

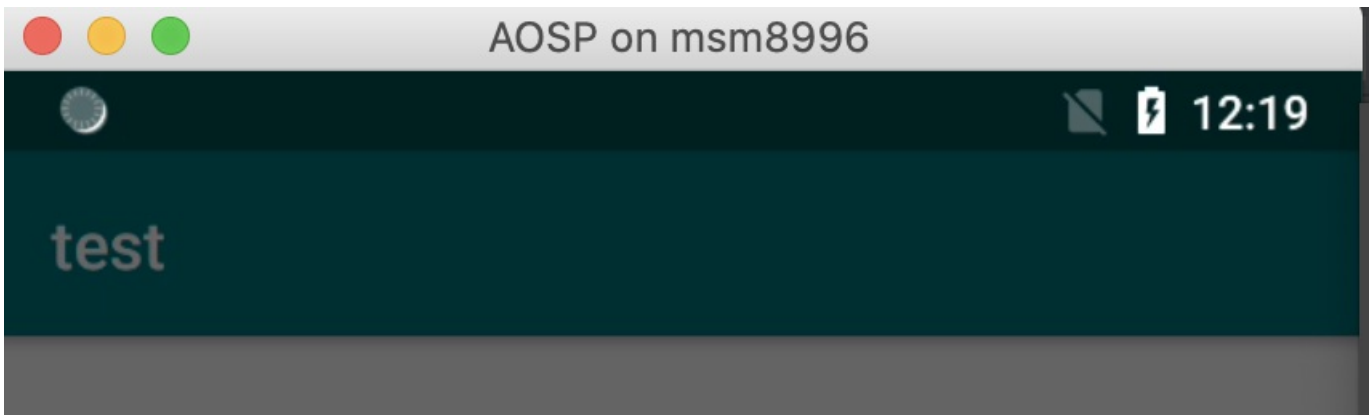


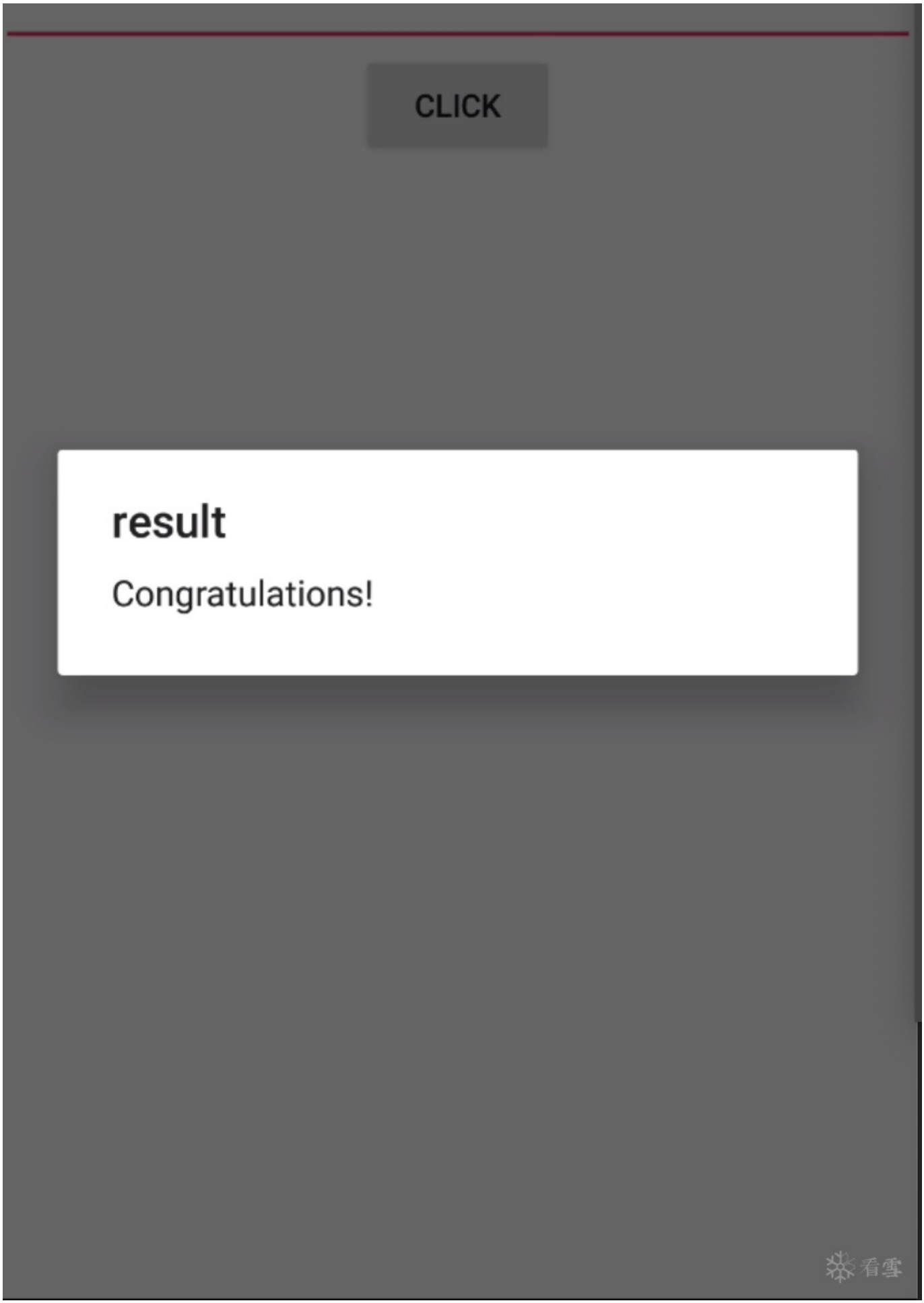
脚本关键函数如下

这里的c3bfc...是我用CyberChef逆出来的。

或者自己写一个java工程，把这个类的所有代码拷出来。。。写个反向的工程就行了。这里我贴出我为了印证RC4的Java工程的代码吧

最后验证索然无味



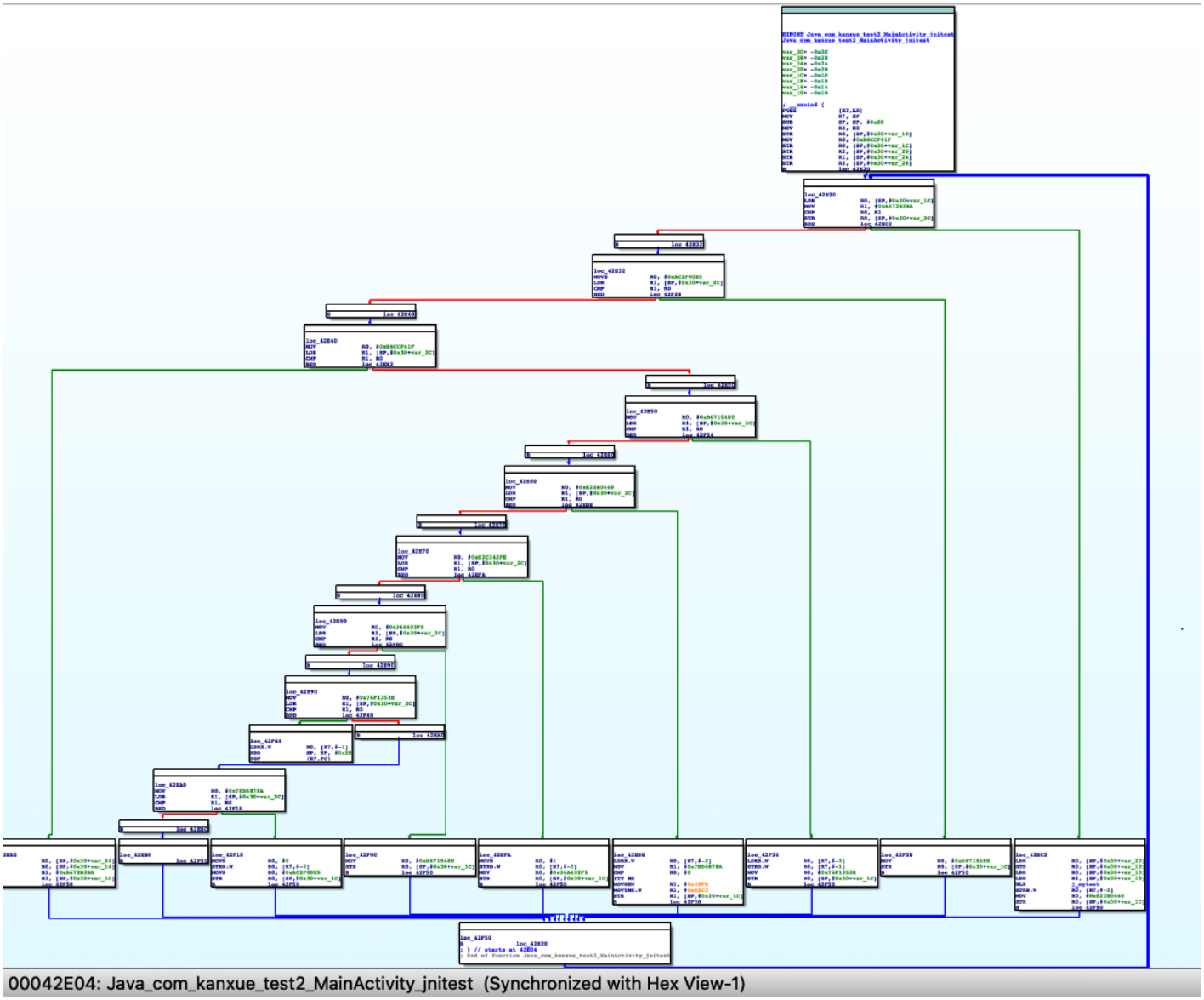


九月第二题

```
public class MainActivity extends AppCompatActivity {
    static {
        StubApp.interface11(0x562);
        System.loadLibrary("native-lib");
    }

    public MainActivity() {
        super();
    }

    public native boolean jnitest(String arg1) {
    }

    protected native void onCreate(Bundle arg1) {
    }

    public native String stringFromJNI() {
    }
}
```

脱壳后查看代码。。猜测onCreate函数应该是360给native化了，暂时不管，从jnitest函数入手。

先静态看看



emmmmm这个函数ollvm了，差点就准备放弃。。。F5看看，静态看了看发现实际真实块只有一个

```
11  v7 = a1;
12  v6 = -1228082145;
13  v5 = a3;
14  while ( 1 )
15  {
16    while ( 1 )
17    {
18      while ( 1 )
19      {
20        while ( 1 )
21        {
22          while ( 1 )
23          {
24            while ( 1 )
25            {
26              while ( 1 )
27              {
28                while ( v6 == -1502432326 )
29                {
30                  v9 = j_mytest(v7, v5);
31                  v6 = -500497336;
32                }
33                if ( v6 != -1406205723 )
34                  break;
35                v6 = -697215872;
36              }
37              if ( v6 != -1228082145 )
38                break;
39              v6 = -1502432326;
40            }
41            if ( v6 != -697215872 )
42              break;
43            v10 = v8;
44            v6 = 1995519294;
45          }
46          if ( v6 != -500497336 )
47            break;
48          v3 = 2127988618;
49          if ( v9 )
50            v3 = -490519810;
51          v6 = v3;
52        }
53        if ( v6 != -490519810 )
54          break;
55        v8 = 1;
56        v6 = 916755189;
57      }
58      if ( v6 != 916755189 )
59        break;
60      v6 = -697215872;
61    }
62    if ( v6 == 1995519294 )
63      break;
64    if ( v6 == 2127988618 )
65    {
66      v8 = 0;
67      v6 = -1406205723;
```

跟进看看，最后跟进到mytest这个函数

```
58            }
59            if ( v18 != -1409559632 )
60              break;
61            *v28 = v31;
62      2     v5 = sub_D7EC((char *)&unk_540C5, *v26);
63            *v29 = (unsigned __int8 *)v5;
64            v6 = (const char *)sub_8748(*v29, 16);
65            *v30 = v6;
66            v7 = strcmp(*base64_encoded, *v30);
67            v8 = -1248156704;
68            if ( !v7 )
69              v8 = -1757769800;
70            v21 = v8;
71          }
72          if ( v18 != -1308050006 )
73            break;
74          v21 = 1325086481;
75        }
76        if ( v18 != -1248156704 )
77          break;
78        v24 = 0;
79        v21 = -1308050006;
80      }
81      if ( v18 != -701396411 )
82        break;
83      md5_encrypted = (int)&v15;
84      v26 = &v14;
85      base64_encoded = (const char **)&v13;
86  1   v28 = &v12;
87      v29 = (unsigned __int8 **)&v11;
88      v30 = (const char **)&v10;
89      a1a = v19;
90      a2a = v20;
91      v17 = &v16;
92      input = GetStringUTFChars(v19, v20, 0);
93      v3 = (int)v17;
94      *(v17 - 2) = input;
95      sub_428BC(*(char **)(v3 - 8), md5_encrypted);
96      *v26 = (int)sub_42888();
97      v4 = (const char *)sub_8748((unsigned __int8 *)md5_encrypted, 16);
98      *base64_encoded = v4;
99      v21 = 1675127349;
100   }
101   if ( v18 != 207755500 )
102     break;
103   v21 = 1325086481;
104 }
105 if ( v18 == 557042104 )
106   break;
107 if ( v18 == 1325086481 )
```

这个混淆的不是很严重，基本块都在，稍微看了看执行顺序，会发现先执行了1基本块，然后执行2号基本块

稍微跟进里面的几个函数一看就能恢复出来，

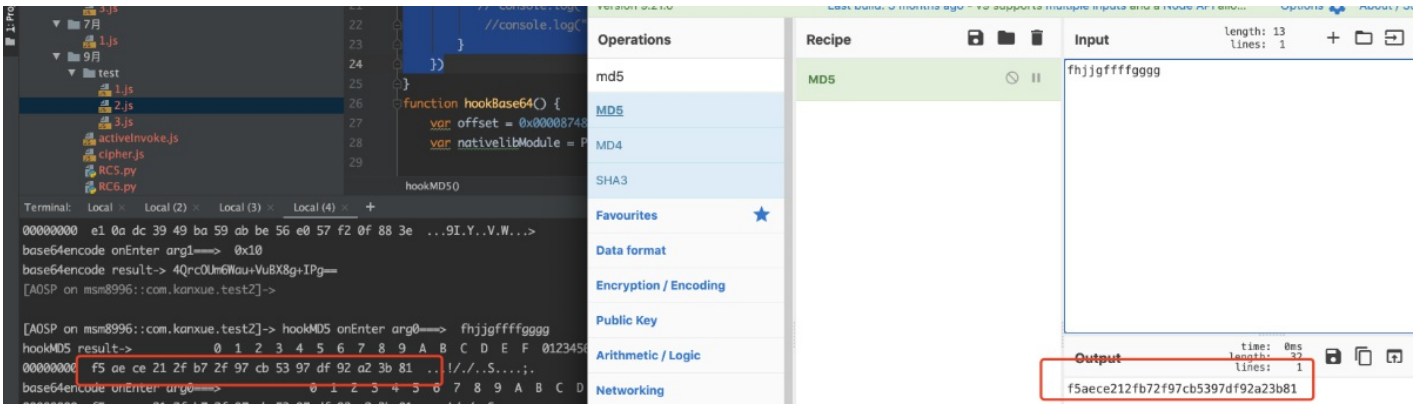
先看sub_428bc

```
{
  while ( 1 )
  {
    while ( 1 )
    {
      while ( 1 )
      {
        v6 = v9;
        if ( v9 != -2139782514 )
          break;
        v11 = &v5;
        v9 = -618147614;
      }
      if ( v6 != -2106017840 )
        break;
      sub_410B8((int)v12, (int)input_copy, len);// 2. second
      v9 = -27426038;
    }
    if ( v6 != -618147614 )
      break;
    v12 = &v4;
    s = input_ptr;
    *v11 = v8;
    v9 = 2005492363;
  }
  if ( v6 == -166479348 )
    break;
  switch ( v6 )
  {
    case -27426038:
      sub_414FC((int)v12, *v11);                // third
      v9 = 1530220377;
      break;
    case 1530220377:
      v9 = -166479348;
      break;
    case 1949234848:
      sub_4100C(v12);                           // first
      input_copy = s;
      len = strlen(s);
      v9 = -2106017840;
      break;
    case 2005492363:
      v2 = -166479348;
      if ( *v11 )
        v2 = 1949234848;
      v9 = v2;
```

跟进第一个执行的函数

```
  a1[16] = 0;
  a1[19] = 0;
  a1[18] = 0;
  v2 = -1512846780;
  do
  {
    while ( 1 )
    {
      while ( v2 == -1709893753 )
      {
        v3[20] = 0x67452301;
        a1[21] = 0xEFCDAB89;
        a1[22] = 0x98BADCFE;
        v4 = a1 + 23;
        v2 = 3365083583;
      }
      if ( v2 != -1512846780 )
        break;
      v3 = a1;
      v2 = -1709893753;
    }
  }
  while ( v2 != -929883713 );
  result = v4;
  *v4 = 271733878;
  return result;
}
```

会发现有几个特别明显的hex值，猜想sub_428bc是md5,emmmm不想看了，猜想这个jnitest的函数是处理我们的输入的，先直接hook吧,最终关键代码如下



图片左边是hook的结果，右边是CyberChef的加密结果，hook多次后发现，sub_428bc函数确实是md5 hash函数，第一个参数是输入，第二个参数是用于存储md5加密后的byte数组的地址。

```
        if ( v18 != -701396411 )
          break;
        md5_encrypted = (int)&v15;
        v26 = &v14;
        base64_encoded = (const char **)&v13;
        v28 = &v12;
        v29 = (unsigned __int8 **)&v11;
        v30 = (const char **)&v10;
        a1a = v19;
        a2a = v20;
        v17 = &v16;
        input = GetStringUTFChars(v19, v20, 0);
        v3 = (int)v17;
        *(v17 - 2) = input;
        sub_428BC(*(char **)(v3 - 8), md5_encrypted);
        *v26 = (int)sub_42888();
        v4 = (const char *)sub_8748((unsigned __int8 *)md5_encrypted, 16);
        *base64_encoded = v4;
        v21 = 1675127349;
      }
      if ( v18 != 207755590 )
```

而mytest函数中sub_8748使用FindCrypt插件会发现是一个base64加密函数，hook再次确认，是base64加密，函数的第一个参数是我们md5加密后的值，第二个参数是固定的16.

```
base64encode onEnter arg0===>            0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  f5 ae ce 21 2f b7 2f 97 cb 53 97 df 92 a2 3b 81  ...!/./..S....;.
base64encode onEnter arg1===>  0x10
base64encode result-> 9a7OIS+3L5fLU5ffkqI7gQ==
```

在hook的过程中会发现，mytest函数第二个基本块，也就是下面这张图中。也调用了sub_8748函数且生成的结果唯一。其值为4QrcOUm6Wau+VuBX8g+IPg==

```
        break;
      *v28 = v31;
      v5 = sub_D7EC((char *)&unk_540C5, *v26);
      *v29 = (unsigned __int8 *)v5;
      v6 = (const char *)sub_8748(*v29, 16);
      *v30 = v6;
      v7 = strcmp(*base64_encoded, *v30);
      v8 = -1248156704;
      if ( !v7 )
        v8 = -1757769800;
      v21 = v8;
    }
```

直接逆推对应的md5 哈希值为e10adc3949ba59abbe56e057f20f883e

一解emmmm,flag是123456

验证发现是对的。。。

小结

在我做这两个练习题的过程中，主要使用静态的代码逆向去进行大概的逻辑分析，使用frida的hook和主动调用去进行动态验证，压根没有什么IDA进行动态调试，所以最后还是喊一句frida牛逼！

另外我在做第一题时，想研究一下如何做到静态jeb看的字符串和动态使用Objection查看的字符串不同的这个技术，可是我竟然在so文件中没有找到操作对应位置的stringID的地方，懵了，希望知道的大佬不吝赐教2333，最后一句，寒冰师傅牛逼！

附件附上

上传的附件：

1.zip

(8.32MB，25次下载)