Kanxue看雪KCTF2019-Q1第二题【变形金钢】Writeup

原创

iqiqiya 于 2019-03-27 21:09:41 发布 652 收藏
分类专栏： 我的逆向之路 我的CTF之路 ------看雪KCTF2019-Q1 我的CTF进阶之路 文章标签： Kanxue看雪KCTF2019-Q1第二题【变形金钢】Wr Kanxue看雪KCTF2019-Q1第二题Writeup KCTF2019-Q1 Writeup

我的逆向之路 同时被 3 个专栏收录
108 篇文章 10 订阅
订阅专栏

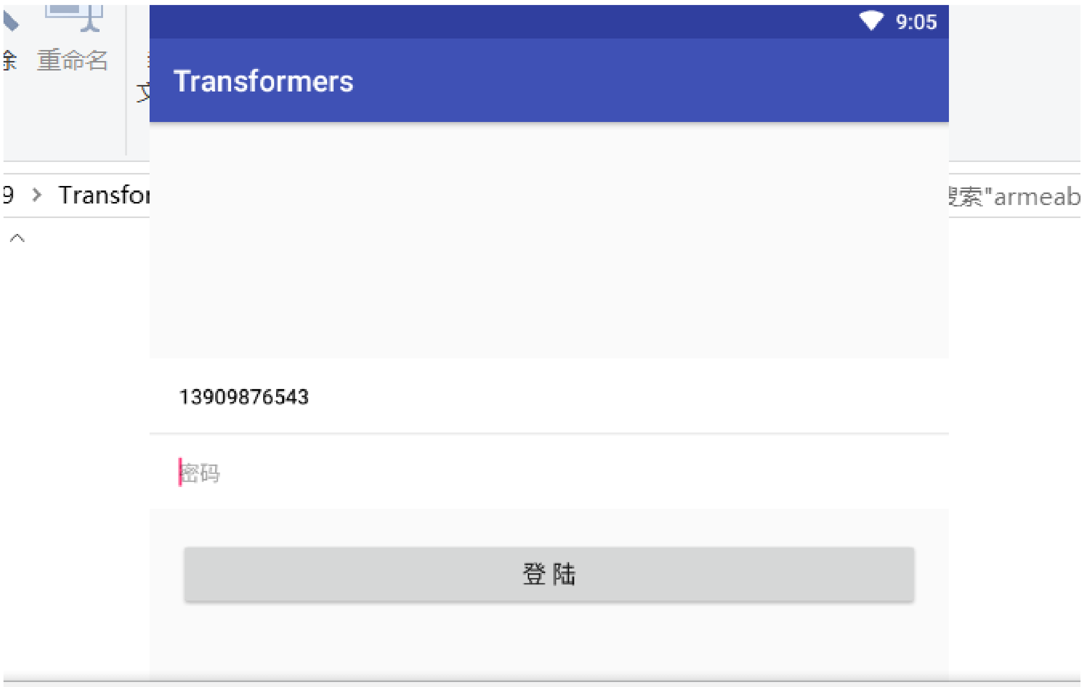
我的CTF之路
92 篇文章 5 订阅
订阅专栏

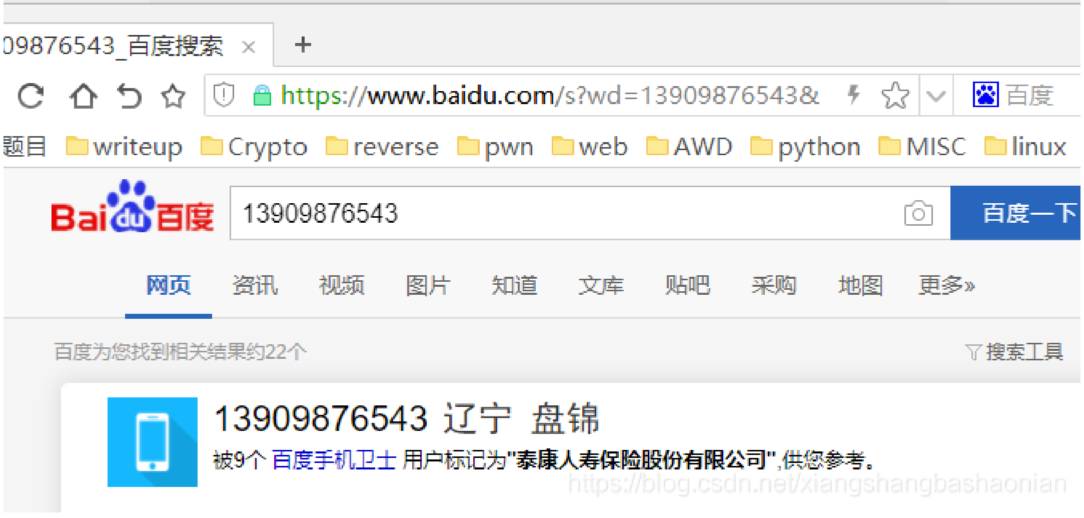
------看雪KCTF2019-Q1
4 篇文章 0 订阅
订阅专栏

> 第二题：变形金钢

解压得到一个apk

模拟器运行　是一个登陆注册　账号直接就有了　还是一个手机号（好奇就百度了一下）



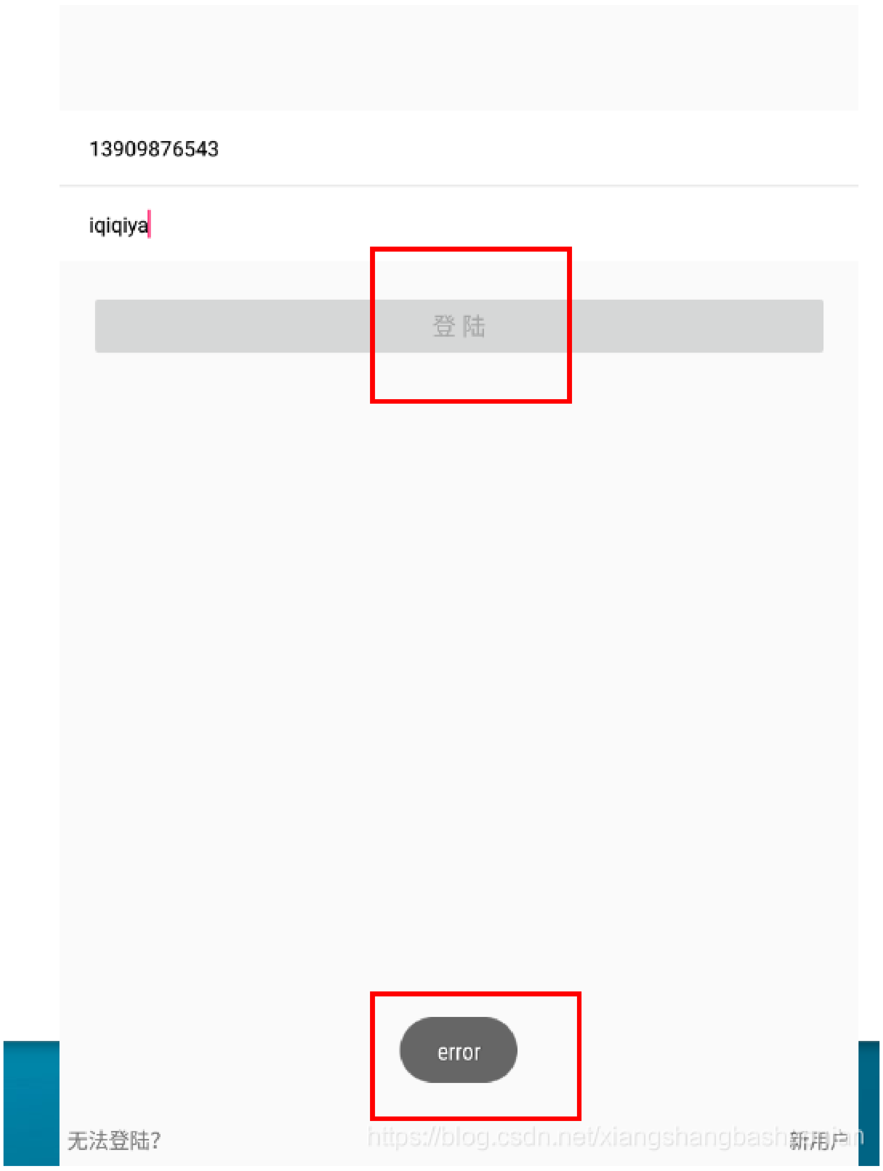

09876543_百度搜索
https://www.baidu.com/s?wd=13909876543&
题目 writeup Crypto reverse pwn web AWD python MISC linux
13909876543
百度一下
网页 资讯 视频 图片 知道 文库 贴吧 采购 地图 更多»
搜索工具

百度为您找到相关结果约22个

13909876543 辽宁 盘锦

被9个 百度手机卫士 用户标记为"**泰康人寿保险股份有限公司**",供您参考。

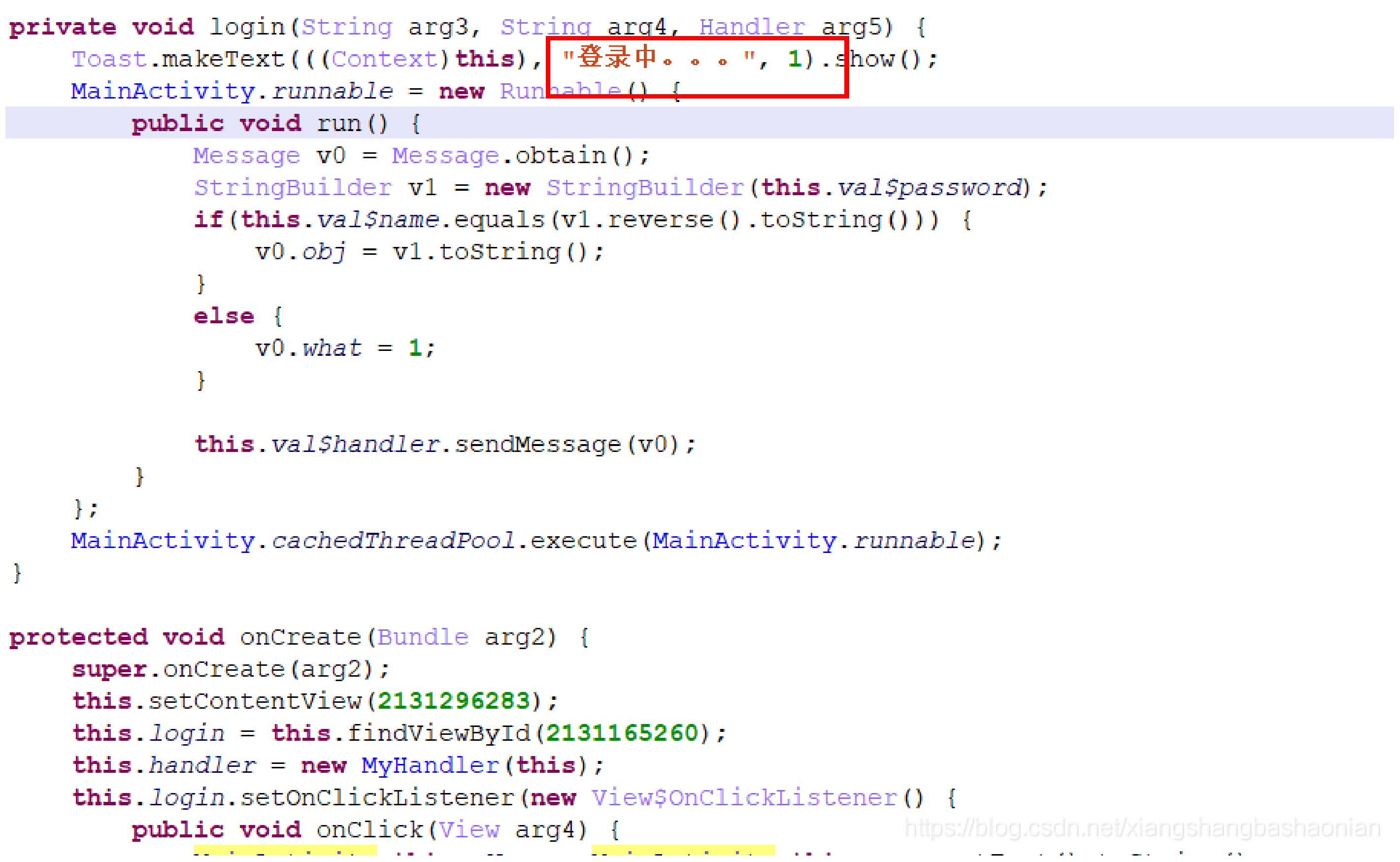
随意输入iqiqiya　会弹出消息框 并且登录按钮不可用 必须重启应用



好了 载入解压下apk 并没有发现什么奇怪的 比如jar文件之类的

直接载入jeb进行查看   奔向MainActivity中的onCreate方法   右键反编译成java代码 发现理论上会调用login方法

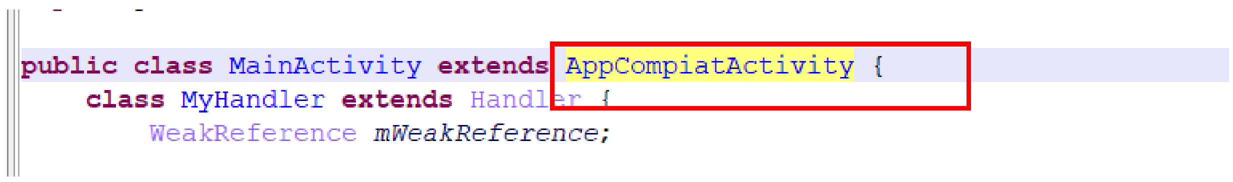
但是实际上并没有 "登陆中。。。"这个字符串运行中并未出现



```
private void login(String arg3, String arg4, Handler arg5) {
    Toast.makeText(((Context)this), "登录中。。。", 1).show();
    MainActivity.runnable = new Runnable() {
        public void run() {
            Message v0 = Message.obtain();
            StringBuilder v1 = new StringBuilder(this.val$password);
            if(this.val$name.equals(v1.reverse().toString())) {
                v0.obj = v1.toString();
            }
            else {
                v0.what = 1;
            }

            this.val$handler.sendMessage(v0);
        }
    };
    MainActivity.cachedThreadPool.execute(MainActivity.runnable);
}

protected void onCreate(Bundle arg2) {
    super.onCreate(arg2);
    this.setContentView(2131296283);
    this.login = this.findViewById(2131165260);
    this.handler = new MyHandler(this);
    this.login.setOnClickListener(new View$OnClickListener() {
        public void onClick(View arg4) {
```

而且也没有字符串"error" 并且没有什么运算

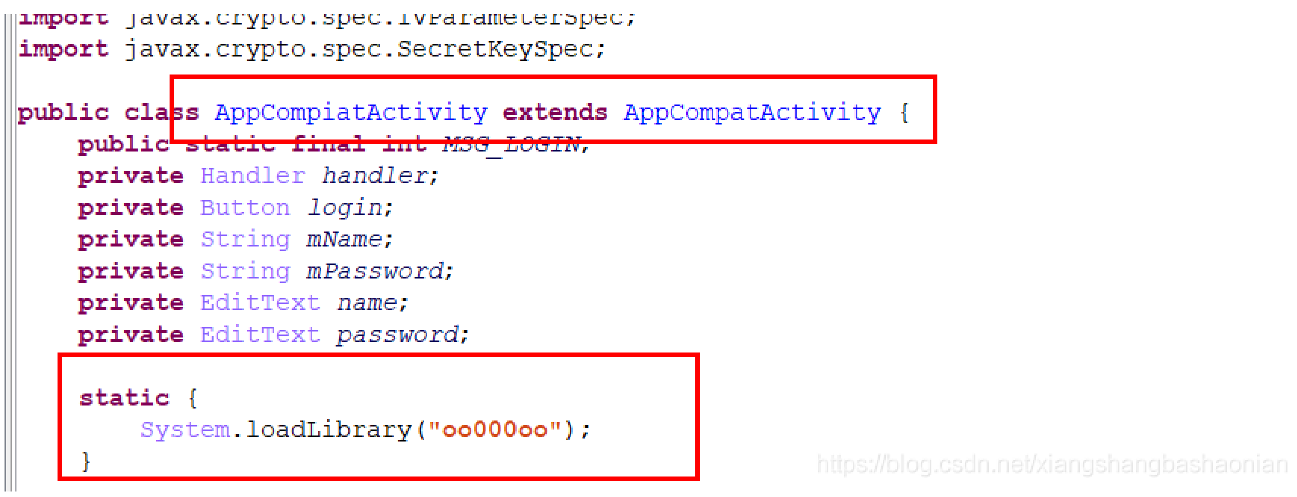
赛后看WP 了解到这里改变了Activity的方法和调用顺序。因为onStart在onCreate之后执行，所以重写了父类的onStart方法，并在onStart方法中覆盖了onCreate的逻辑，进而最后生效的是onStart内的逻辑。



```
public class MainActivity extends AppCompiatActivity {
    class MyHandler extends Handler {
        WeakReference mWeakReference;
```

MainActivity 继承的 AppCompiatActivity 比原生的 AppCompatActivity 中间多了个 i 并且有加载名为oo000oo的so文件

双击进去可以发现



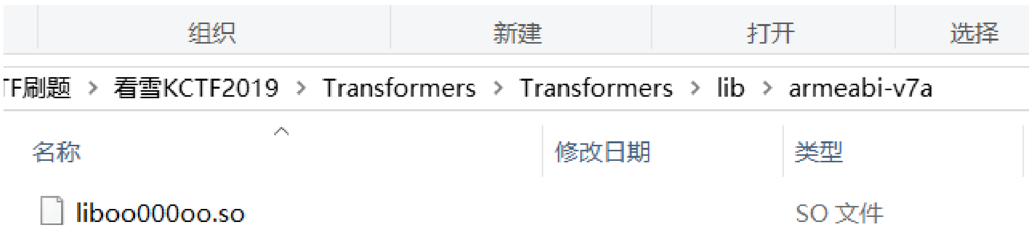
```
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public class AppCompiatActivity extends AppCompatActivity {
    public static final int MSG_LOGIN;
    private Handler handler;
    private Button login;
    private String mName;
    private String mPassword;
    private EditText name;
    private EditText password;

    static {
        System.loadLibrary("oo000oo");
    }
```

OnStart()的实现如下

```
protected native boolean eq(String arg1) {
    }
protected void onStart() {
        super.onStart();
        this.login = this.findViewById(2131165260);
        this.login.setOnClickListener(new View$OnClickListener() {
            public void onClick(View arg5) {
                AppCompiatActivity.this.mName = AppCompiatActivity.this.name.getText().toString();
                AppCompiatActivity.this.mPassword = AppCompiatActivity.this.password.getText().toString();
                if(!TextUtils.isEmpty(AppCompiatActivity.this.mName) && !TextUtils.isEmpty(AppCompiatActivi
                        .this.mPassword)) {
                    int v1 = 0;
                    AppCompiatActivity.this.login.setEnabled(false);
                    if(AppCompiatActivity.this.eq(AppCompiatActivity.this.mPassword)) {//eq方法
                        byte[] v5 = AppCompiatActivity.this.mPassword.getBytes();
                        int v3 = 24;
                        if(v5.length != v3) {
                            byte[] v2 = new byte[v3];
                        label_38:
                            if(v1 < v2.length) {
                                byte v3_1 = v1 < v5.length ? v5[v1] : ((byte)v1);
                                v2[v1] = v3_1;
                                ++v1;
                                goto label_38;
                            }

                            v5 = v2;
                        }

                        v5 = AppCompiatActivity.dec(v5, "2ggdrsLgM7iPNYPQrD58Rg==".getBytes());
                        AppCompiatActivity v1_1 = AppCompiatActivity.this;
                        StringBuilder v2_1 = new StringBuilder();
                        v2_1.append("flag{");
                        v2_1.append(new String(v5));
                        v2_1.append("}");
                        Toast.makeText(((Context)v1_1), v2_1.toString(), 1).show();
                    }
                    else {
                        Toast.makeText(AppCompiatActivity.this, "error", 1).show();
                    }

                    return;
                }

                Toast.makeText(AppCompiatActivity.this, "用户名或密码为空", 1).show();
            }
        });
```

可以看到有一个native的方法

我们接着分析oo000oo.so



组织 新建 打开 选择

TF刷题 › 看雪KCTF2019 › Transformers › Transformers › lib › armeabi-v7a

名称	修改日期	类型
liboo000oo.so		SO 文件

载入IDA

查看函数列表 可以看到datadiv_decode5009363700628197108()中是几个字符串的初始化操作 且都是异或运算

```
char *datadiv_decode5009363700628197108()
{
  int v0; // r0
  int v1; // r0
  int v2; // r0
  char *result; // r0

  v0 = 0;
  do
  {
    byte_4020[v0] ^= 0xA5u;    //记作s1
    ++v0;
  }
  while ( v0 != 37 );
  v1 = 0;
  do
    byte_4050[v1++] ^= 0xA5u;   //记作s2
  while ( v1 != 66 );
  v2 = 0;
  do
    byte_40A0[v2++] ^= 0x84u;   //记作s3
  while ( v2 != 42 );
  result = &byte_40D0;
  byte_40CA ^= 0xFCu;             //记作s4
  byte_40CB ^= 0xFCu;
  byte_40CC ^= 0xFCu;
  byte_40D0 ^= 0x62u;
  byte_40D1 ^= 0x62u;
  byte_40D2 ^= 0x62u;
  byte_40D3 ^= 0x62u;
  byte_40D4 ^= 0x62u;
  byte_40D5 ^= 0x62u;
  byte_40D6 = __PAIR__(HIBYTE(byte_40D6), (unsigned __int8)(byte_40D6 ^ 0x62)) ^ 0x6200;
  byte_40D8 ^= 0x62u;
  byte_40D9 ^= 0x62u;
  byte_40DA ^= 0x62u;
  byte_40DB ^= 0x62u;
  byte_40DC ^= 0x62u;
  byte_40DD ^= 0x62u;
  byte_40DE ^= 0x62u;
  byte_40DF ^= 0x62u;
  byte_40E0 ^= 0x62u;
  byte_40E1 ^= 0x62u;
  byte_40E2 ^= 0x62u;
  byte_40E3 ^= 0x62u;
  byte_40E4 ^= 0x62u;
  byte_40E5 ^= 0x62u;
  return result;
}
```

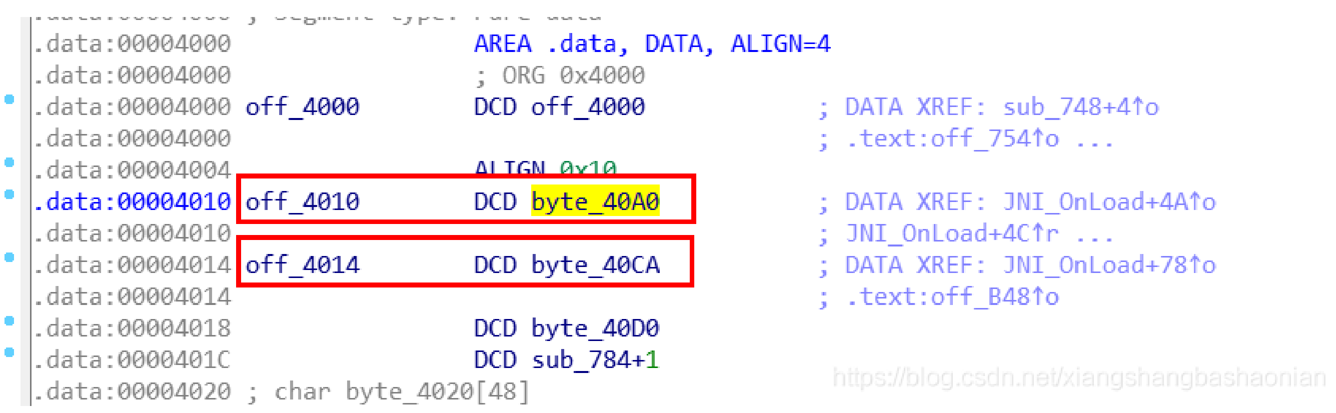
python解密可以得到

```
# -*- coding: utf-8 -*-
s1 =[
  0x93, 0x90, 0x95, 0xC3, 0x9C, 0x95, 0x9C, 0xC6, 0x88, 0x92,
  0x97, 0x94, 0x92, 0x88, 0x96, 0x93, 0x91, 0x92, 0x88, 0x9C,
  0x96, 0x96, 0x94, 0x88, 0xC6, 0x9D, 0x97, 0xC1, 0xC3, 0x9D,
  0xC7, 0x9C, 0x9D, 0xC0, 0x9C, 0x9D, 0xA5]#xor 0xa5
s2 = [
  0x84, 0x9F, 0x86, 0x81, 0x80, 0x83, 0x8D, 0x8C, 0x8E, 0x88,
  0x8F, 0x8A, 0xC5, 0xDB, 0xFA, 0xFE, 0xF8, 0xDE, 0xD8, 0x9A,
  0x99, 0x9B, 0x89, 0x8B, 0xE5, 0xFB, 0xC4, 0xC7, 0xC6, 0xC1,
  0xC0, 0xC3, 0xC2, 0xCD, 0xCC, 0xCF, 0xCE, 0xC9, 0xC8, 0xCB,
  0xCA, 0xD5, 0xD4, 0xD7, 0xD6, 0xD1, 0xD0, 0xD3, 0xD2, 0xDD,
  0xDC, 0xDF, 0x95, 0x94, 0x97, 0x96, 0x91, 0x90, 0x93, 0x92,
  0x9D, 0x9C, 0xF9, 0x82, 0x9E, 0xA5]#xor 0xa5
s3 = [
  0xE5, 0xEA, 0xE0, 0xF6, 0xEB, 0xED, 0xE0, 0xAB, 0xF7, 0xF1,
  0xF4, 0xF4, 0xEB, 0xF6, 0xF0, 0xAB, 0xF2, 0xB3, 0xAB, 0xE5,
  0xF4, 0xF4, 0xAB, 0xC5, 0xF4, 0xF4, 0xC7, 0xEB, 0xE9, 0xF4,
  0xED, 0xE5, 0xF0, 0xC5, 0xE7, 0xF0, 0xED, 0xF2, 0xED, 0xF0,
  0xFD, 0x84]#xor 0x84
s4 = [
  0x99, 0x8D, 0xFC, 0x4A, 0x2E, 0x08, 0x03, 0x14, 0x03, 0x4D,
  0x0E, 0x03, 0x0C, 0x05, 0x4D, 0x31, 0x16, 0x10, 0x0B, 0x0C,
  0x05, 0x59, 0x4B, 0x38, 0x62]#xor 0xfc   0x62
ds1 = ''
ds2 = ''
ds3 = ''
ds4 = ''
for i in range(len(s1)):
  ds1+=chr(s1[i]^0xa5)
print ds1
for i in range(len(s2)):
  ds2+=chr(s2[i]^0xa5)
print ds2
for i in range(len(s3)):
  ds3+=chr(s3[i]^0x84)
print ds3
for i in range(len(s4)):
  if i<2:
    ds4+=chr(s4[i]^0xfc)
  else:
    ds4+=chr(s4[i]^0x62)
print ds4
#ds1 = 650f909c-7217-3647-9331-c82df8b98e98
#ds2 = !:#$%&()+-*/`~_[]{}?<>,.@^abcdefghijklmnopqrstuvwxyz0123456789\';
#ds3 = android/support/v7/app/AppCompiatActivity
#ds4 = eq`(Ljava/lang/String;)Z
```

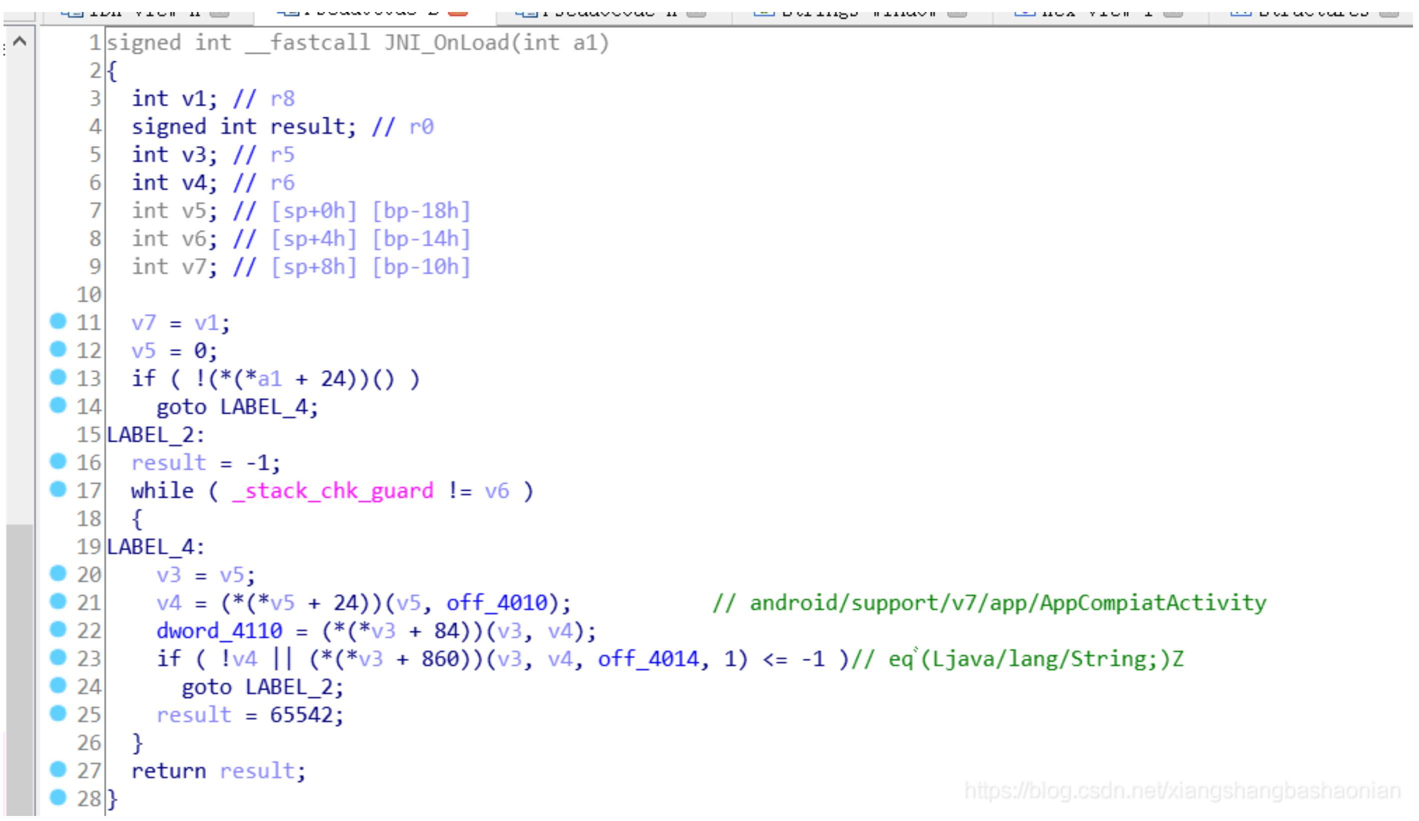
可以看到ds1像是一个序列号　ds2是base64变化的编码表　而ds3，ds4指的是我们的eq()方法

那么现在就要找eq()在哪里定义

我们接着看JNI_Onload()　里边有两个off_4010 off_4014　分别对应前面我们得到的ds3,ds4

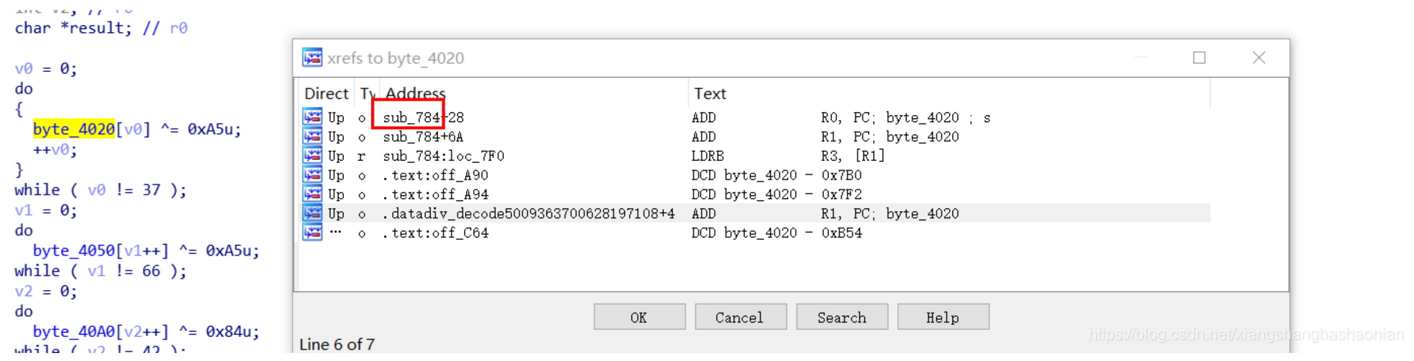


```
.data:00004000                 AREA .data, DATA, ALIGN=4
.data:00004000                 ; ORG 0x4000
.data:00004000 off_4000        DCD off_4000            ; DATA XREF: sub_748+4↑o
.data:00004000                                         ; .text:off_754↑o ...
.data:00004004                 ALIGN 0x10
.data:00004010 off_4010        DCD byte_40A0           ; DATA XREF: JNI_OnLoad+4A↑o
.data:00004010                                         ; JNI_OnLoad+4C↑r ...
.data:00004014 off_4014        DCD byte_40CA           ; DATA XREF: JNI_OnLoad+78↑o
.data:00004014                                         ; .text:off_B48↑o
.data:00004018                 DCD byte_40D0
.data:0000401C                 DCD sub_784+1
.data:00004020 ; char byte_4020[48]
```



```
signed int __fastcall JNI_OnLoad(int a1)
{
  int v1; // r8
  signed int result; // r0
  int v3; // r5
  int v4; // r6
  int v5; // [sp+0h] [bp-18h]
  int v6; // [sp+4h] [bp-14h]
  int v7; // [sp+8h] [bp-10h]

  v7 = v1;
  v5 = 0;
  if ( !(*(*a1 + 24))() )
    goto LABEL_4;
LABEL_2:
  result = -1;
  while ( _stack_chk_guard != v6 )
  {
LABEL_4:
    v3 = v5;
    v4 = (*(*v5 + 24))(v5, off_4010);           // android/support/v7/app/AppCompiatActivity
    dword_4110 = (*(*v3 + 84))(v3, v4);
    if ( !v4 || (*(*v3 + 860))(v3, v4, off_4014, 1) <= -1 )// eq(Ljava/lang/String;)Z
      goto LABEL_2;
    result = 65542;
  }
  return result;
}
```

也就是说这里调用了eq()　我们查看字符串数组引用　可以发现除了那个初始化函数 还有一个sub_784()



```
char *result; // r0

v0 = 0;
do
{
  byte_4020[v0] ^= 0xA5u;
  ++v0;
}
while ( v0 != 37 );
v1 = 0;
do
  byte_4050[v1++] ^= 0xA5u;
while ( v1 != 66 );
v2 = 0;
do
  byte_40A0[v2++] ^= 0x84u;
```

```
Direct Ty Address                                  Text
Up     o  sub_784+28                               ADD             R0, PC; byte_4020 ; s
Up     o  sub_784+6A                               ADD             R1, PC; byte_4020
Up     r  sub_784:loc_7F0                          LDRB            R3, [R1]
Up     o  .text:off_A90                            DCD byte_4020 - 0x7B0
Up     o  .text:off_A94                            DCD byte_4020 - 0x7F2
Up     o  .datadiv_decode5009363700628197108+4     ADD             R1, PC; byte_4020
...    o  .text:off_C64                            DCD byte_4020 - 0xB54
```

双击进入查看

```
int __fastcall sub_784(int a1)
{
  size_t v1; // r10
  unsigned __int8 *v2; // r6
  _BYTE *v3; // r8
  _BYTE *v4; // r11
  int v5; // r0
  size_t v6; // r2
  char *v7; // r1
  int v8; // r3
  int v9; // r1
  unsigned int v10; // r2
  int v11; // r3
  int v12; // r0
  int v13; // r4
  unsigned __int8 v14; // r0
  _BYTE *v15; // r3
  _BYTE *v16; // r5
  char *v17; // r4
  int v18; // r5
  int v19; // r1
  int v20; // r0
  signed int v21; // r1
  int v22; // r2
  size_t v23; // r0
  unsigned int v24; // r8
  unsigned int v25; // r5
  _BYTE *v26; // r0
  int v27; // r3
  int v28; // r10
  unsigned int v29; // r2
  int v30; // r12
  bool v31; // zf
  _BYTE *v32; // r1
  bool v33; // zf
  int v34; // r3
  int v35; // r1
  unsigned __int8 v36; // r11
  unsigned int v37; // lr
  char v38; // r1
  char *v39; // r2
  int v40; // t1
  unsigned int v42; // [sp+4h] [bp-234h]
  unsigned int v43; // [sp+8h] [bp-230h]
  unsigned int v44; // [sp+10h] [bp-228h]
  char *s; // [sp+14h] [bp-224h]
  char v46[256]; // [sp+18h] [bp-220h]
  char v47[256]; // [sp+118h] [bp-120h]
  int v48; // [sp+218h] [bp-20h]

  s = (*(*a1 + 0x2A4))();                              // 取password
  v1 = strlen(byte_4020);                              // 获取长度"650f909c-7217-3647-9331-c82df8b98e98"  也就是v1=3
  v2 = malloc(v1);
  v3 = malloc(v1);
  v4 = malloc(v1);
  _aeabi_memclr(v2, v1);
  _aeabi_memclr(v3, v1);
  _aeabi_memclr(v4, v1);
  if ( v1 )
  {
    v5 = 0;
    v6 = v1;
    v7 = byte_4020;
    do
    {
      v8 = *v7++;
      if ( v8 != '-' )
        v3[v5++] = v8;
      v3[v5++] = v8;
      --v6;
    }
    while ( v6 );                               // 这个do while循环用于将"_"去掉并且倒序 给v3[]  v5=32
    if ( v5 >= 1 )
    {
      v9 = v5 - 1;                              // v9=31
      v10 = -8;
      v11 = 0;
      v12 = 0;
      do
      {
        if ( (v11 | (v10 >> 2)) > 3 )           // 右移2相当于除以4
        {
          v13 = v12;
        }
        else
        {
          v13 = v12 + 1;
          v2[v12] = 45;
        }
        v14 = v3[v9--];
        v11 += 0x40000000;
        v2[v13] = v14;
        ++v10;
        v12 = v13 + 1;
      }
      while ( v9 != -1 );
      if ( v13 >= 0 )
      {
        v15 = v4;
        while ( 1 )
        {
          v16 = *v2;
          if ( (v16 - 97) <= 5u )
            break;
          if ( (v16 - 48) <= 9u )
          {
            v16 = &unk_23DE + v16 - 48;
            goto LABEL_18;
          }
LABEL_19:
          *v15++ = v16;
          --v12;
          ++v2;
          if ( !v12 )
            goto LABEL_20;
        }
        v16 = &unk_23D8 + v16 - 97;
LABEL_18:
        LOBYTE(v16) = *v16;
        goto LABEL_19;
      }
    }
  }
LABEL_20:
  _aeabi_memcpy8(v46, &unk_23E8, 256);          // 自定义的RC4中的box
                                                // D7DF02D4FE6F533C256C999706568FDE40116407361570CA18177D6A
  v17 = v47;
  v18 = 0;
  do
  {
    sub_D20(v18, v1);
    v47[v18++] = v4[v19];
  }
  while ( v18 != 256 );
  v20 = (v47[0] - 41);
  v46[0] = v46[v20];
  v46[v20] = -41;
  v21 = 1;
  do
  {
    v22 = v46[v21];
    v20 = (v20 + v47[v21] + v22) % 256;
    v46[v21++] = v46[v20];
    v46[v20] = v22;
  }
  while ( v21 != 256 );
  v23 = strlen(s);                              // s其实就是password
  v24 = v23;                                    // v23,v24都是password的长度
  v25 = v4[3];
  v43 = 8 * (3 - -3 * (v23 / 3));
  v42 = v25 + v43 / 6;
  v26 = malloc(v42 + 1);
  if ( v24 )
  {
    v28 = 0;
    v29 = 0;
    v30 = 0;
    v44 = v25;
    do
    {
      v28 = (v28 + 1) % 256;
      v35 = v46[v28];
      v30 = (v30 + v35) % 256;
      v46[v28] = v46[v30];
      v46[v30] = v35;
      v17 = v46[v28];
      v36 = v46[(v35 + v17)] ^ s[v29];          // 用一串常数与password进行xor再进行base64编码(用的是下面那个变形
      if ( v29 && (v27 = 0xAAAAAAAB * v29 >> 32, v37 = 3 * (v29 / 3), v37 != v29) )
      {
        v31 = v29 == 1;
        if ( v29 != 1 )
          v31 = v37 + 1 == v29;
        if ( v31 )
        {
          v32 = byte_4050;                      // 这里是变形后的base64码表!!:#$%&()+-*/`~_[]{}?<>,.@^abcdefg
          v26[v44 + v29] = byte_4050[v26[v44 + v29] | (v36 >> 4)];
          v17 = &v26[v44 + v29];
          v27 = 4 * v36 & 0x3C;
          v17[1] = v27;
          if ( v29 + 1 >= v24 )
            goto LABEL_53;
        }
        else
        {
          v33 = v29 == 2;
          if ( v29 != 2 )
            v33 = v37 + 2 == v29;
          if ( v33 )
          if ( v35 )
          {
            v17 = (v36 & 0xC0);
            v34 = v44++ + v29;
            v26[v34] = byte_4050[v26[v34] | (v17 >> 6)] ^ 0xF;// 接着对上一步的输出4位一组分割  每组第二个与0xf
            v27 = &v26[v34];
            *(v27 + 1) = byte_4050[v36 & 0x3F];
          }
        }
      }
      else
      {
        v26[v44 + v29] = byte_4050[v36 >> 2] ^ 7;// 每组第0个与0x7异或
        v17 = &v26[v44 + v29];
        v27 = 16 * v36 & 0x30;
        v17[1] = v27;
        if ( v29 + 1 >= v24 )
        {
          v38 = byte_4050[v27];
          *(v17 + 1) = 15163;
          goto LABEL_43;
        }
      }
      ++v29;
    }
    while ( v29 < v24 );
  }
  while ( 1 )
  {
    if ( v43 )
    {
      v32 = (&dword_0 + 1);
      v17 = v42;
      v39 = &byte_24E8;                         // 取出最后要比较的字符串" {9*8ga*l!Tn?@#fj'j$\g;;"
      do
      {
        v27 = v26[v25++];
        v40 = *v39++;
        if ( v40 != v27 )                       // 这里就是比较的地方
          v32 = 0;
      }
      while ( v25 < v42 );
    }
    else
    {
      v32 = (&dword_0 + 1);
    }
    v26 = (_stack_chk_guard - v48);
    if ( _stack_chk_guard == v48 )
      break;
LABEL_53:
    v38 = v32[v27];
    v17[2] = 52;
LABEL_43:
    v17[1] = v38;
  }
  return v32;
}
```

分析可知：

> 1. 内置字符串 650f909c-7217-3647-9331-c82df8b98e98 去掉 "_"再逆序加上"_"
> 2. 用了个字符替换算法得到 RC4加密中的key
> 3. 使用自定义box 的 rc4 算法实现加密输入
> 4. 使用改了base64编码表的进行base64编码
> 5. 最后和内置的字符串"{9*8ga*l!Tn?@#fj'j",0x24,"\g;;"对比

直接上代码：

```
import string

s = " {9*8ga*l!Tn?@#fj'j$\g;;"
cc = ''
for i in xrange(len(s)):
     if (i % 4 == 0):
             cc += chr(ord(s[i]) ^ 7)
     elif (i % 4 == 2):
             cc += chr(ord(s[i]) ^ 0xF)
     else:
             cc += s[i]
my_base64table = "!:#$%&()+-*/`~_[]{}?<>,.@^abcdefghijklmnopqrstuvwxyz0123456789\\';"
std_base64table ="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/="
t = string.maketrans(my_base64table, std_base64table)
print map(ord, cc.translate(t).decode('base64'))
#[253, 30, 138, 78, 9, 202, 144, 3, 231, 241, 133, 159, 155, 247, 131, 62, 14]
```

RC4解密实现：

```
代码来自https://bbs.pediy.com/thread-249973.htm
#include <cstdio>
#include <cstring>

unsigned char sbox[] =
{
  0xD7, 0xDF, 0x02, 0xD4, 0xFE, 0x6F, 0x53, 0x3C, 0x25, 0x6C,
  0x99, 0x97, 0x06, 0x56, 0x8F, 0xDE, 0x40, 0x11, 0x64, 0x07,
  0x36, 0x15, 0x70, 0xCA, 0x18, 0x17, 0x7D, 0x6A, 0xDB, 0x13,
  0x30, 0x37, 0x29, 0x60, 0xE1, 0x23, 0x28, 0x8A, 0x50, 0x8C,
  0xAC, 0x2F, 0x88, 0x20, 0x27, 0x0F, 0x7C, 0x52, 0xA2, 0xAB,
  0xFC, 0xA1, 0xCC, 0x21, 0x14, 0x1F, 0xC2, 0xB2, 0x8B, 0x2C,
  0xB0, 0x3A, 0x66, 0x46, 0x3D, 0xBB, 0x42, 0xA5, 0x0C, 0x75,
  0x22, 0xD8, 0xC3, 0x76, 0x1E, 0x83, 0x74, 0xF0, 0xF6, 0x1C,
  0x26, 0xD1, 0x4F, 0x0B, 0xFF, 0x4C, 0x4D, 0xC1, 0x87, 0x03,
  0x5A, 0xEE, 0xA4, 0x5D, 0x9E, 0xF4, 0xC8, 0x0D, 0x62, 0x63,
  0x3E, 0x44, 0x7B, 0xA3, 0x68, 0x32, 0x1B, 0xAA, 0x2D, 0x05,
  0xF3, 0xF7, 0x16, 0x61, 0x94, 0xE0, 0xD0, 0xD3, 0x98, 0x69,
  0x78, 0xE9, 0x0A, 0x65, 0x91, 0x8E, 0x35, 0x85, 0x7A, 0x51,
  0x86, 0x10, 0x3F, 0x7F, 0x82, 0xDD, 0xB5, 0x1A, 0x95, 0xE7,
  0x43, 0xFD, 0x9B, 0x24, 0x45, 0xEF, 0x92, 0x5C, 0xE4, 0x96,
  0xA9, 0x9C, 0x55, 0x89, 0x9A, 0xEA, 0xF9, 0x90, 0x5F, 0xB8,
  0x04, 0x84, 0xCF, 0x67, 0x93, 0x00, 0xA6, 0x39, 0xA8, 0x4E,
  0x59, 0x31, 0x6B, 0xAD, 0x5E, 0x5B, 0x77, 0xB1, 0x54, 0xDC,
  0x38, 0x41, 0xB6, 0x47, 0x9F, 0x73, 0xBA, 0xF8, 0xAE, 0xC4,
  0xBE, 0x34, 0x01, 0x4B, 0x2A, 0x8D, 0xBD, 0xC5, 0xC6, 0xE8,
  0xAF, 0xC9, 0xF5, 0xCB, 0xFB, 0xCD, 0x79, 0xCE, 0x12, 0x71,
  0xD2, 0xFA, 0x09, 0xD5, 0xBC, 0x58, 0x19, 0x80, 0xDA, 0x49,
  0x1D, 0xE6, 0x2E, 0xE3, 0x7E, 0xB7, 0x3B, 0xB3, 0xA0, 0xB9,
  0xE5, 0x57, 0x6E, 0xD9, 0x08, 0xEB, 0xC7, 0xED, 0x81, 0xF1,
  0xF2, 0xBF, 0xC0, 0xA7, 0x4A, 0xD6, 0x2B, 0xB4, 0x72, 0x9D,
  0x0E, 0x6D, 0xEC, 0x48, 0xE2, 0x33
};

int main() {
    unsigned char key[256] = {0};
    char k[] = "36f36b3c-a03e-4996-8759-8408e626c215" ;
    unsigned char plain[] = {253, 30, 138, 78, 9, 202, 144, 3, 231, 241, 133, 159, 155, 247, 131, 62, 14};
//    unsigned char plain[] = {253, 31, 74, 242, 6, 138, 148, 4, 231, 77, 128, 159, 143, 248, 195, 250};
    for (int i = 0; i < 256; i++)
    {
        key[i] = k[i % strlen(k)];
    }
int j = key[0] - 0x29;
    sbox[0] = sbox[j];
    sbox[j] = 0xd7u;
//    int j;
    unsigned char temp;
    for(int i=1;i<256;i++) {
        j=(j+sbox[i]+key[i])%256;
        temp=sbox[i];
        sbox[i]=sbox[j];
        sbox[j]=temp;
    }
    int i = 0; j = 0; int t = 0, kk = 0;
    for(kk=0;kk<strlen((char *)plain);kk++)
    {
        i=(i+1)%256;
        j=(j+sbox[i])%256;
        temp=sbox[i];
        sbox[i]=sbox[j];
        sbox[j]=temp;
        t=(sbox[i]+sbox[j])%256;
        plain[kk]^=sbox[t];
    }
    puts((char *)plain);
}
```

实际注册是APP调用so时,虚拟机会自动首先寻找JNI_OnLoad(JNIEnv*...)函数的,自动从这个函数进行执行,所以注册或者初始化一些信息可以在这里进行

参考链接：

https://bbs.pediy.com/thread-250376.htm

https://bbs.pediy.com/thread-250395.htm

https://bbs.pediy.com/thread-249973.htm