

“摸仙堡”解题报告

杨舒晗 20307130386 王楠欣 20307130377 李丹琦 20307130353 李昊霖 20307130260

SwYxSzOt:

Attacker: 王楠欣

Flag: FLAG{YtUareASuccessOr}

控制流混淆，需要一步步找到真正可以返回 1 的方法，分析该方法的 check 逻辑。

(找入口时，可以先在 app 中任意输入字符串，看 toast 是什么，根据显示的“WRONG!”可在 jeb 中直接搜索文本内容找到对应的 toast 代码，再根据需要满足的条件里出现的各种方法的调用，分析输入字符串后程序执行的逻辑，也可以查 button、edittext 的 id 等等。) 本 apk 的 flag 被分为了两段，在 switch-case 语句执行中分别进行 replace 等一系列操作，最后与两个常量字符串比较，当均满足相等条件时，方法 return1，toast 显示 RIGHT。分析中发现，因为多次的 replace 操作，导致这个 apk 的 flag 存在多解。

过程图：

(1) 根据方法 invoke 多次点击方法，找到的 return 口，分析四个方法调用，最后锁定第三个传参为 a (即完整 flag) 的方法，追踪 Pmdacmxweufv 方法，寻找 return 1 的逻辑。

```
public static boolean kxynekxvmbcguj(String a) {
    if(a.length() != 22) {
        return false;
    }

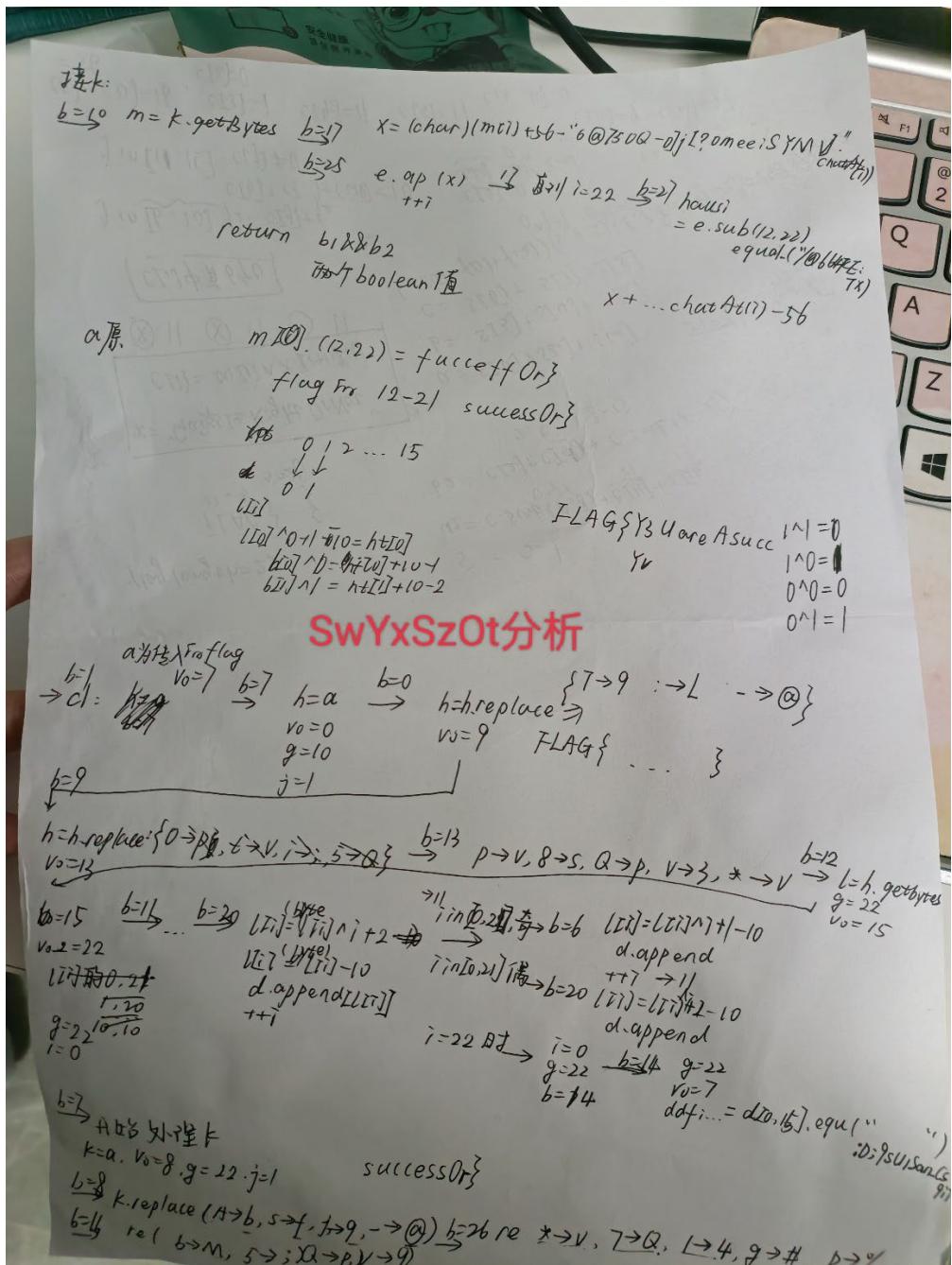
    if(a.charAt(0) == 70 && a.charAt(1) == 76 && a.charAt(2) == 65 && a.charAt(3) == 71 && a.charAt(4) == 0x7B && a.charAt(21) == 0x7D) {
        String flag = a.substring(5, 21);
        return (11jtjt1lltjiljiltjfif.brfdqpbnaus(flag)) || (11jtjt1lltjiljiltjfif.uhgyddbpdyawu(flag)) || (11jtjt1lltjiljiltjfif.pmdacmxweufv(a)) || (11jtjt1lltjiljiltjfif.pmdacmxweufv(b));
    }

    return false;
}
```

(2) 通过追踪 Pmdacmxweufv 方法，找到 klwxamgrhwanyu 方法。

```
public static boolean klwxamgrhwanyu(String a) {
    int v4_1;
    int v18;
    int v8;
    int b = 1;
    char x = ' ';
    int i = 0;
    int j = 0;
    boolean ddfivagxddfuih = false;
    boolean hausiddfafvjafv = false;
    StringBuilder d = new StringBuilder();
    StringBuilder e = new StringBuilder();
    int g = 10;
    String h = null;
    String k = null;
    byte[] l = null;
    byte[] m = null;
    while(j > -1) {
        int j = j;
        byte[] m = m;
        int g = g;
        switch(b) {
            case 0: {
                h.replace("T", "9").replace(":", "L").replace("-", "@");
                v8 = 9;
                m = m;
                g = g;
                j = j;
                break;
            }
            case 1: {
                v8 = 7;
                g = g;
                j = j;
                m = m;
                break;
            }
            case 2: {
                return true;
            }
        }
    }
}
```

(3) 分析 switch-case 语句



(4) 根据分析逻辑锁定关键判定代码

0-16

```

case 14: {
    g = g;
    v0 = 7;
    ddfivagxddfuih = d.substring(0, 16).equals(ljiilljiliftttftitfjf.c);
    j = j;
    m = m;
    break;
}

```

12-22:

```

        case 27: {
            hausiddfafvjafv = e.substring(12, 22).equals("/@664KE:TX");
            v0 = 24;
            j = j;
            m = m;
            g = g;
            break;
        }
    }

```

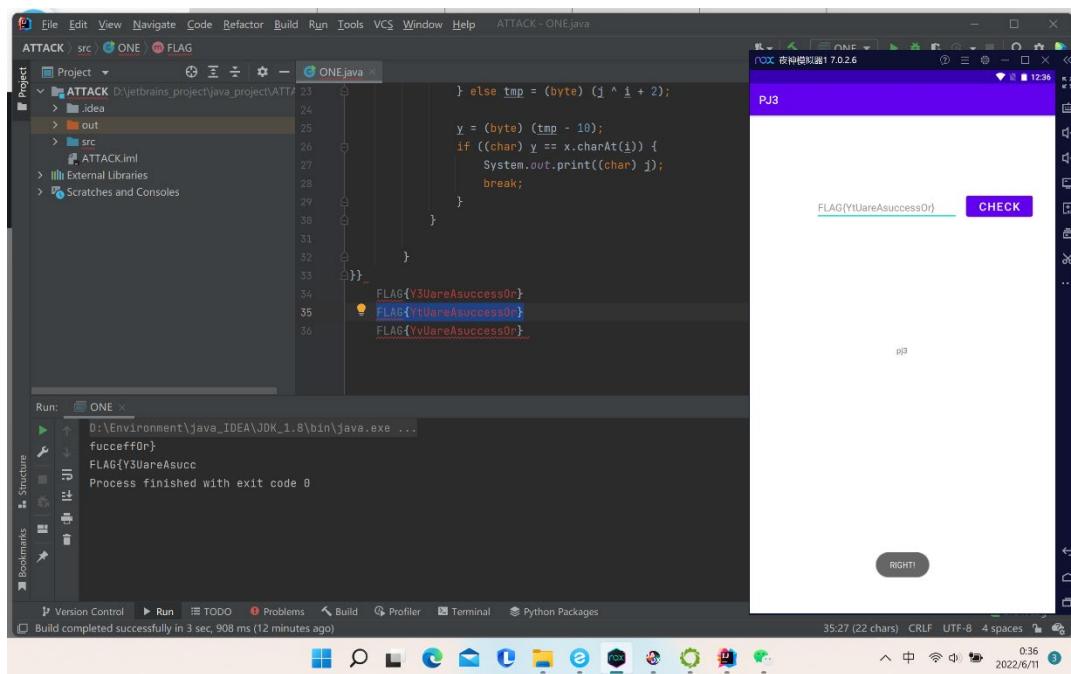
最终 return:

```

case 24: {
    return (ddfivagxddfuih) && (hausiddfafvjafv);
}

```

(5) 根据以上分析写 decode 代码并运行出结果，根据 replace 代码确定一些满足条件的 flag



Unauthorized:

Attacker: 王楠欣

Flag: FLAG{humankindcannotbearverymuchreality}

jeb 分析中，发现 toast 的内容与 edithint 有关，于是 andorid studio 打开 apk 发现 hint 字符串为"RIGHT or WRONG"，根据 toast 内容，if 条件需满足 a==0 且 check (k) 返回值为 1。

对 a 的值，根据逻辑分析，输入的 f 经 base64 加密后满足 if 条件，c 被重置为一个包含该 apk 所有类的字符串数组 ps 中索引为 2 的字符串，以该字符串命名的类中 invoke 方法，返回值为 0，即 a=0（分析见后续草稿纸图）。这同时满足了索引不为 3，不进入后续 if (i==3) 后的代码块执行。a=0 后不被重置。

接下来是 nativecode 的分析，nativecode 的逻辑分析如后图 ida 截图，主要逻辑为：在 switch-case 语句中，需要仅遍历 dest 数组中所有值为 1 的位，并执行—操作，最后 dest 数组为全 0 数组时，才不会 return 0。其中 s[] 数组的值与 flag 内容相关，而

`s[]`的值又与遍历的 `dest` 数组的索引有关。根据 `dest[]`最终全 0，可以推出 `s[]`数组的值（见草稿纸推演），再反推 `flag` 的值（这需要自己写解密函数）。

过程图：

(1) 锁定 toast 语句对应的 flag check 入口，寻找 `h` 字符串的值，希望根据 toast 内容来确定 `a` 与 `MainActivity.this.check()` 应该满足当的条件。(这决定了 `substring` 的截取段) 同时寻找对 `a` 进行赋值的代码。

```

    ...
    if(v4_4 == 0 || !MainActivity.this.check(k) ? 8 : 0;
    if(!((boolean)((int)v1_1))) {
        Toast.makeText(MainActivity.this, h.substring(v4_4, v4_4 + 6), 0).show();
        return;
    }
    String sr = MainActivity.this.getResources().getString(0x7f0e0077); // string:root_nightroot_check\n\t\ntext="a";
    Toast.makeText(MainActivity.this, sr, 0).show();
}

```

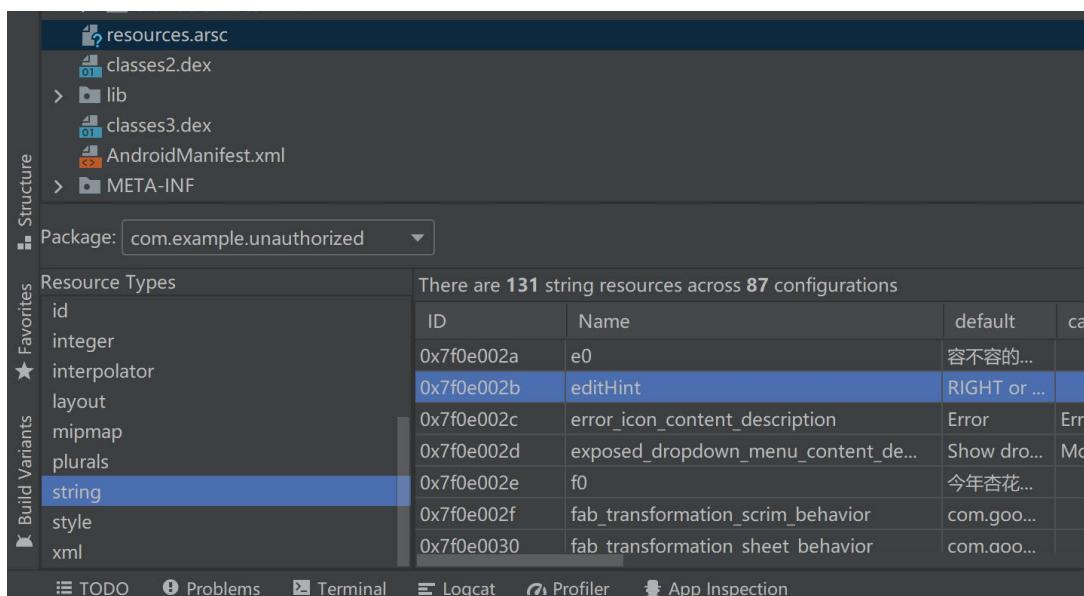
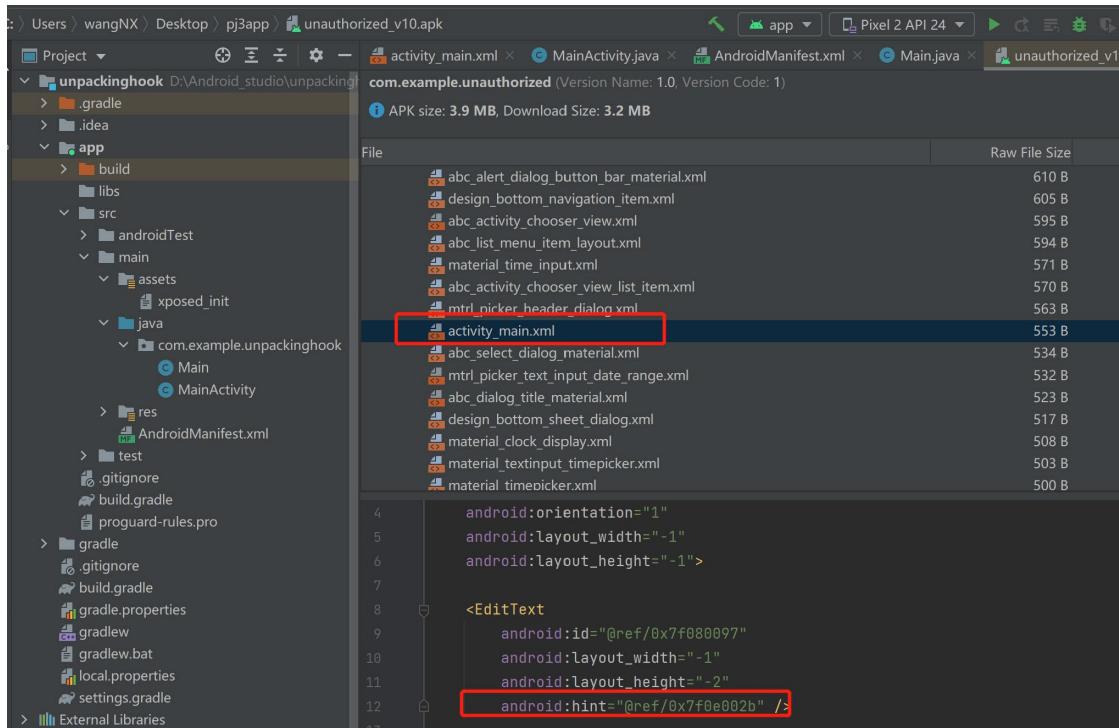
```

    ...
    if(ef.length() == 56 && ef.charAt(ef.length() - 2) == 61 && ef.charAt(ef.length() - 1) == 61) {
        c = myutil.encode(f.substring(0, 5) + ((char)f.charAt(39)));
        k = f.substring(5, 39);
    }

    int i;
    for(i = 0; i < MainActivity.this.ps.length(); ++i) {
        if(MainActivity.this.ps[i].equals(c)) {
            try {
                Class c = Class.forName("com.example.unauthorized." + c);
                Method[] M = c.getDeclaredMethods();
                if(M.length != 0) {
                    Object o = c.newInstance();
                    if(o != null) {
                        int v13 = (int)((Integer)o));
                        a = v13;
                        break;
                    }
                }
            }
        }
    }
}

```

(2) 用 andorid studio 打开 apk 寻找 hint 值，跟据 hint 字符串内容，确定 if 条件中需要满足 `a==0`，`check` 返回值为 1，使 toast 内容为“RIGHT”而非“WRONG”



(3) ida 打开.so 分析 check 函数

IDA - libunauthorized.so C:\Users\wangNX\Desktop\pj3app\unauthorized_v10\lib\x86\libunauthorized.so

```

Function name
sub_540
_cxa_finalize
_cxa_atexit
register_atfork
_stack_chk_fail
memmove
JNINvn::GetStringUTFLength(jstring *)
JNINvn::GetStringUTFChars(jstring *,juc)
memset
sub_5F0
nullsub_1
_jnullsub_1
(char,char,int)
(_char,_char,int)
Java_com_example_unauthorized_MainActivity_check:jstring
JNINvn::GetStringUTFLength(jstring *)
JNINvn::GetStringUTFChars(jstring *,juc)
_cxa_atexit
_cxa_finalize
_register_atfork
_stack_chk_fail
memcpy
memset

Line 15 of 23
AU: idle Down Disk: 14GB
00000A57 Java_com_example_unauthorized_MainActivity_check:32 (A57)

```

字符串数组的值（修改类型以及 array）

IDA - libunauthorized.so C:\Users\wangNX\Desktop\pj3app\unauthorized_v10\lib\x86\libunauthorized.so

```

Function name
sub_540
_cxa_finalize
_cxa_atexit
register_atfork
_stack_chk_fail
memmove
JNINvn::GetStringUTFLength(jstring *)
JNINvn::GetStringUTFChars(jstring *,juc)
memset
sub_5F0
nullsub_1
_jnullsub_1
(char,char,int)
(_char,_char,int)
Java_com_example_unauthorized_MainActivity_check:jstring
JNINvn::GetStringUTFLength(jstring *)
JNINvn::GetStringUTFChars(jstring *,juc)
_cxa_atexit
_cxa_finalize
_register_atfork
_stack_chk_fail
memcpy
memset

Line 15 of 23
AU: idle Down Disk: 14GB
00000A57 Java_com_example_unauthorized_MainActivity_check:32 (A57)

```

(4) ida 中 check 函数的逻辑分析（主要逻辑见前方陈述，草稿逻辑推演如下）

~~main check~~
~~of next 7 = {0xA000 | 00}~~

E7-12B

6

if

SZJ为1的偏数j位置，对应Vb=0

为0的*i*位置 对应 $u_6 = s[i+1]$

switch-case $\frac{1}{n}$

偏應: $SIN = 1$: default

$$\text{Pr}(\text{odd}) = \frac{1}{2} = 0.5 : \text{case } \underline{\text{S}(I+1)}.$$

517

$$\begin{aligned} \text{ind} = 31 \rightarrow V_0 = 21 \rightarrow V_0 = 11 \rightarrow V_0 = 19 \rightarrow V_0 = 28 \rightarrow V_0 = 37 \rightarrow V_0 = 46 \rightarrow V_0 = 56 \\ \text{set } = 14 + 16 = \rightarrow V_0 = 66 \rightarrow V_0 = 76 \rightarrow V_0 = 68 \rightarrow V_0 = 60 \rightarrow V_0 = 52 \rightarrow V_0 = 43 \\ \rightarrow V_0 = 34 \rightarrow V_0 = 25 \rightarrow V_0 = 15 \end{aligned}$$

ST17	偶数为0	奇数	1	3	5	7	9	11	13	15	17	19	21	23
			7	7	5	4	4	4	3	3	3	1	1	1

25	27	29	31	33
8	8	8	7	5

unauthorized分析2

(4) 写解密函数拿 flag

balcakbox:

Attacker: 王楠欣

Flag: BACAB

对该 apk 的逆向，一开始因为提示以及 MainActivity 中的各种解密，我写了很多的解密函数来拿到 tool（主要是 base64 图片解码）、hint 等。后来发现可能这是误导，因为即使拿到 tool 和 hint，对于不了解魔方规则的人并不能很好地结合 ida 中的 check 函数找到 flag。于是我只专注于 flagcheck 的入口，用 ida 分析 native code。分析前需要在 ida 中删去一些冗余代码才能 F5（这与其他比如 lab10、“无能狂怒”等 apk 的操作相同）。分析 c 代码逻辑，可以确定 flag 由 A,B,C,D 四种字符组成，长度为 5，于是想到利用 xposed hook 该 apk 的，调用 Check 方法，利用枚举法，return 1 的即为需要找的 flag。

过程图：

- (1) 各种解密 (对最终拿到 flag 帮助不大)

Tool:

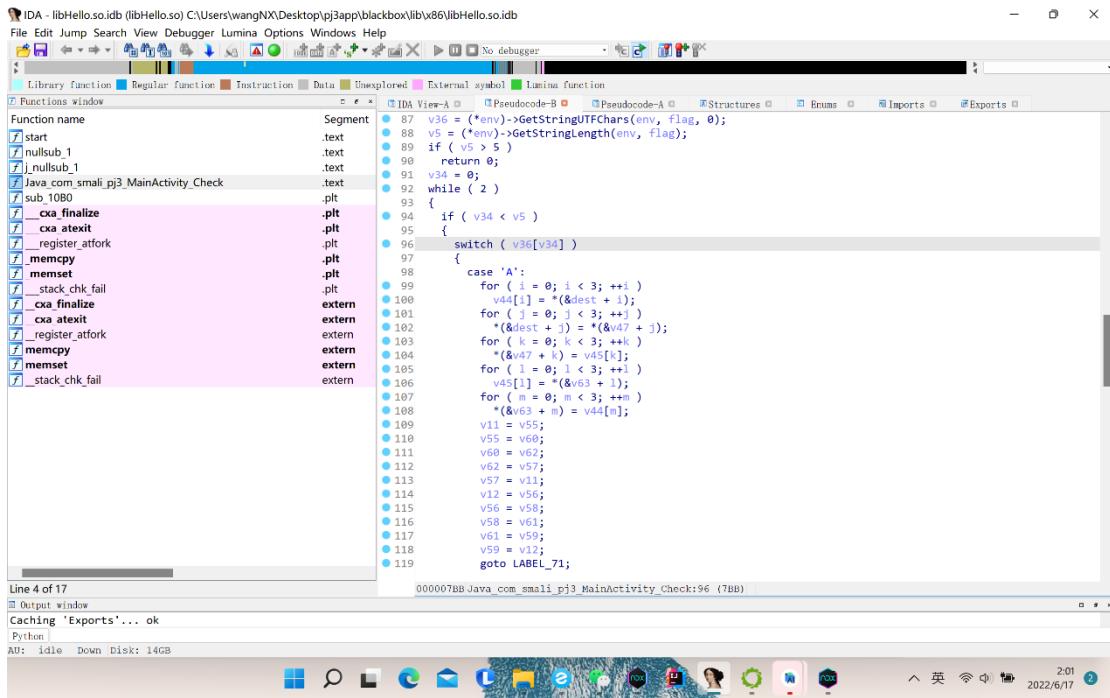


Hint:

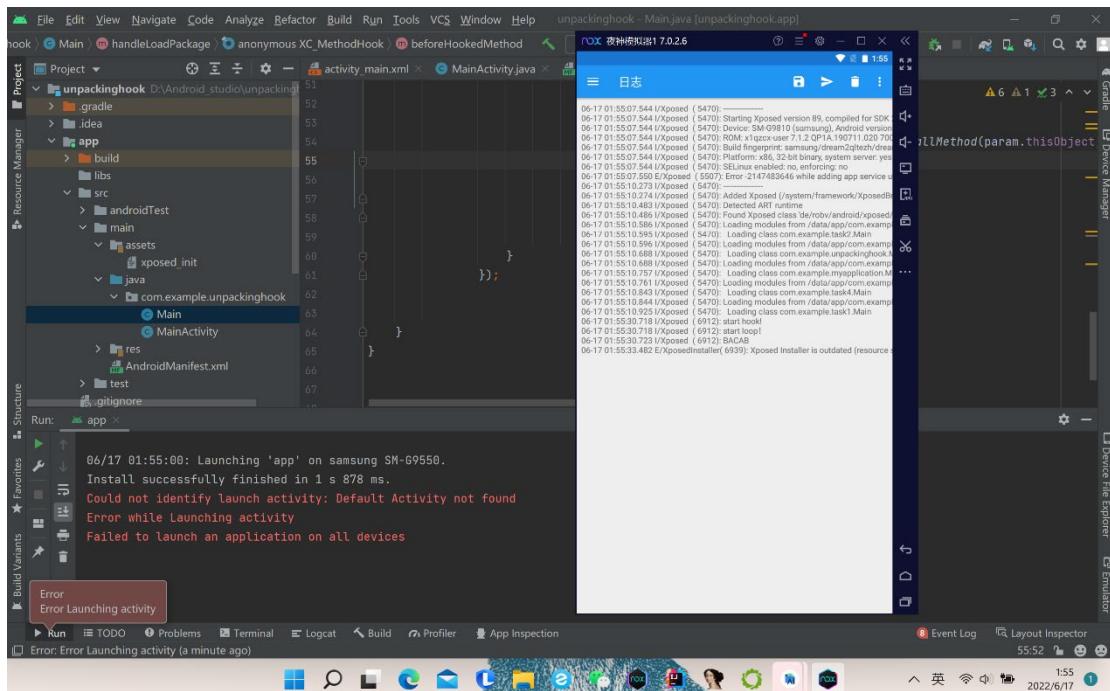
```
}💡  
//gethint(): {Hint:1=white,2=green,3=orange,4=red,5=yellow,6=blue}
```

(2) 专注于 flagcheck 入口，ida 分析 check 函数逻辑

```
String v2 = flag.getText().toString();
if(MainActivity.this.Check(v2)) {
    Toast.makeText(MainActivity.this.getApplicationContext(), "RIGHT!", 1).show();
    return;
}
```



(3) 分析到对 flag 的限制后（即长度为 5，由 A,B,C,D 四种字符组成），5 个 for 循环枚举满足该类限制的字符串，并调用 check 方法查看 return 值，当 return 值为 1 时，将该字符串输出到 xposed 日志中，可见 flag 是“BACAB”



四方客栈

Attacker: 王楠欣

Flag: flag{c0m_n2bb_we11d0ne}

本 apk 通过验证输入序列显示回复，当输入正确的序列时，回复显示 flag。用 jeb 进行分析发现需要调用某个方法对输入序列进行验证，验证成功时，HTTP 联网回复 flag。分析发现该方法与 apk 中的一个解密函数有关，但因为该函数非常复杂，所有我在反汇编代码寻找类似于验证的方法，发现 MainActivity 中的 native code check 函数，猜测该函数即为验证序列是否正确的函数，于是用 ida 对 check 函数进行逻辑分析，最终写解密函数拿正确的输入序列。输入序列后得到 flag。

过程图：

(1) Jeb 分析发送操作，mejfk 为输入的序列。

因为当 v7==1 时才能进入回复信息包括多余字符串的代码部分，所以需要在此之前进入 v7 赋值为 1 的代码段。V7=1 的同时，对 v5_2 赋值为解密字符串+输入序列。

```
if(((Boolean)Class.forName(new DecryptManager().decrypt(MainActivity.rev("db6f3606f3fbabb91a8b3dfe24d551f061fd855
    v5_2 = new DecryptManager().decrypt(MainActivity.rev("c67a74bed123171d")) + MainActivity.mejfk;
    v7 = 1;
})
else {
    v5_2 = new DecryptManager().decrypt(MainActivity.rev("06ea65a0fae72f9d"));
}

new Thread(() -> try {
    new Http(MainActivity.demkt, v5_2).run();
}
catch(Exception v2) {

    if(v7 == 1) {
        v5_3.setText("回复: " + new DecryptManager().decrypt(MainActivity.rev("1cc7b8e182e0c667")) + Http.status + new DecryptManager().decryp
        return;
    }
}

v5_3.setText("回复: " + new DecryptManager().decrypt(MainActivity.rev("25f21e61c48d868595ecd04972d7f9ff")) + Http.status);
```

(2) 查看 decrypt 方法，结合后续 new Http(MainActivity.demkt, v5_2).run() 代码。

decrypt 解密字符串后与输入序列一起，与 Http 联网操作，可以猜测这里调用了某个方法，进行了对输入序列的验证，这个方法与解密字符串相关。但因为所见 decrypt 方法中运用了大量的控制流混淆和字符串混淆，暂且不在解密函数上操作，而是根据以上分析在 jeb 反汇编出的代码中寻找可能的 check 函数。

```
public String decrypt(String arg5) {
    String v0 = "\u06E6\u06DF\u06EB\u06D9\u06D7\u06E7\u06D7\u06D8\u06D6\u06E8\u06DF\u06D7\u06D8\u06DA\u06E8\u06E4\u06DC\u06E7\label_1:
    switch(v0.hashCode() ^ 0x8F ^ 0x7D ^ 708 ^ 0x17F ^ 296 ^ 903 ^ 0x20F ^ 1940060661) {
        case -6956556: {
            v0 = "\u06D8\u06EB\u06E8\u06D8\u06E5\u06E8\u06D7\u06DA\u06DC\u06D8\u06D7\u06E8\u06E2\u06DB\u06DC\u06E7";
            goto label_1;
        }
        case 269042929: {
            return new String(this.decrypt(DecryptManager.hexStr2ByteArr(arg5)));
        }
        case 867718209: {
            v0 = "\u06DA\u06E5\u06DF\u06EC\u06E6\u06EB\u06DC\u06E2\u06E8\u06D8\u06EB\u06E8\u06E4\u06E1\u06D8\u06DB\u06D9";
        }
        default: {
            goto label_1;
        }
    }
}

public byte[] decrypt(byte[] arg2) {
    try {
        return this.decryptCipher.doFinal(arg2);
    }
    catch(Exception v0) {
        v0.printStackTrace();
        return new byte[10];
    }
},
```

```

public static byte[] hexStr2ByteArr(String arg12) {
    String v0 = "\u06E7\u06DF\u06E7\u06EC\u06E8\u06E7\u06DF\u06D7\u06D8\u06D9\u06E6\u06E4\u06E0\u06D9";
    int v1 = 0;
    String v3 = null;
    int v5 = 0;
    byte[] v6 = null;
    int v7 = 0;
    byte[] v8 = null;
label_9:
    switch(v0.hashCode() ^ 612 ^ 300 ^ 322 ^ 306 ^ 23 ^ 344 ^ 0xF2 ^ 0xCB5F2AE9) {
        case -1712056063: {
            return v6;
        }
        case -1648540208: {
            v0 = "\u06D0\u06D7\u06E0\u06DA\u06E2\u06E8\u06D8\u06DB\u06E0\u06E8\u06D8\u06E5\u06DC\u06EC\u06DB\u06D9\u06E1";
            v6 = new byte[v7 / 2];
            goto label_9;
        }
        case -1626485811: {
            v0 = "\u06E7\u06DC\u06D6\u06D8\u06E5\u06E8\u06D8\u06DA\u06D8\u06D9\u06E7\u06EB\u06E0\u06D8\u06E8\u06D9\u06E5";
            goto label_9;
        }
        case -1305243953: {
            v0 = "\u06DA\u06E1\u06DF\u06EB\u06E0\u06E5\u06E2\u06E0\u06D6\u06DC\u06E8\u06E1\u06D8\u06D8\u06E2";
            v7 = v8.Length;
            goto label_9;
        }
        case -1204587318: {
            v0 = "\u06E7\u06E4\u06EB\u06D7\u06E2\u06E0\u06D7\u06E7\u06E5\u06E7\u06D8\u06D9\u06EB\u06E4\u06E1";
            goto label_9;
        }
        case -1163418442: {
            v1 = v5 + 2;
            v0 = "\u06D8\u06E0\u06DA\u06DB\u06E2\u06E0\u06D8\u06E7\u06D6\u06DF\u06DC\u06E8\u06D8\u06E6\u06DA";
            goto label_9;
        }
    }
}

```

- (2) 发现 MainActivity 中存在的 native code check 方法，认为极大可能就是验证输入序列是否正确的函数，开始用 ida 分析函数逻辑，如下为了 return 1，需要使 v0==15。在 for 循环中 v10 的值依次++。在 v10<=15 时 break，跳过 Label_14 进入 while 循环，当++v14==0 时，跳到 Label_14 执行。循环中涉及到 v13 的赋值，v13=v17[v12]。因为最后 v10 会=15，故 v12=16，而 v17 数组 type 为 int[2]，有异常，因此将 v17 类型转变为 int[16]，发现了对 v17 的赋值操作，对输入的字符串进行处理。对输入的字符串，对应的将每个字符串转换成数字（if 条件限制和 mfta 函数，每个字符对应传参中的 v7+v6）char-87 或 char-48（最终范围为 0-15）。此外，因为对 v17 类型的更改，check 函数中涉及到 v17 数组运算的部分代码也发生了改变（图 4）。

```

61     }
62     while ( v3 );
63     v8 = v15 != 0;
64   }
65     result = 0;
66     if ( v8 && !HIDWORD(v24) )
67   {
68       v10 = 0;
69       for ( i = -14; ; ++i )
70     {
71       v12 = v10++;
72       if ( v10 <= 0xE )
73         break;
74     LABEL_14:
75       if ( v10 == 15 )
76         return 1;
77     }
78     v13 = v17[v12];
79     v14 = i;
80     while ( v13 <= *((_DWORD *)&v24 + v14 + 1) )
81   {
82     if ( !++v14 )
83       goto LABEL_14;
84   }
85     LABEL_20:
86     result = 0;
87   }
88   return result;
89 }

```

The screenshot shows the IDA Pro interface with the assembly pseudocode. Two specific regions are highlighted with red boxes: one around the `LABEL_14:` label and another around the `LABEL_20:` label. The code is annotated with line numbers from 61 to 89.

```

16     int v10, // [esp+4h] [ebp-50h] BYREF
17     int v17[16]; // [esp+8h] [ebp-54h] BYREF
18     unsigned int v18; // [esp+48h] [ebp-14h]
19
20     v18 = __readgsdword(0x14u);
21     *(_QWORD *)&v17[14] = 14LL;
22     *(_QWORD *)&v17[12] = 0x7000000000000000;
23     *(_QWORD *)&v17[10] = 0x6000000000000000;
24     *(_QWORD *)&v17[8] = 0xA000000000000000;
25     *(_QWORD *)&v17[6] = 0xF000000000000000;
26     *(_QWORD *)&v17[4] = 0x4000000000000000;
27     *(_QWORD *)&v17[2] = 0x3000000000000000;
28     *(_QWORD *)&v17 = 0x1000000000000005LL;
29     v16 = 15;
30
31     v15 = 1;
32     v3 = (*env)->GetStringLength(env, input);
33     v4 = (*env)->GetStringUTFChars(env, input, 0);
34     if ( v3 <= 0 )
35     {
36         v8 = 1;
37     }
38     else
39     {
40         v5 = v4;
41         do
42         {
43             v7 = *v5;
44             if ( (unsigned __int8)(*v5 - 48) >= 0xAu && (unsigned __int8)(v7 - 97) >= 6u )
45             {
46                 v15 = 0;
47                 goto LABEL_20;
48             }
49             v6 = -87;
50             if ( (char)v7 < 58 )
51                 v6 = -48;
52             mfta(&v16, v7 + v6, v17, &v15);
53             ++v3;
54             --v3;
55         }
56         while ( v3 );
57         v8 = v15 != 0;
58
59         if ( v8 && !v17[15] )
60         {
61             v10 = 0;
62             for ( i = -14; ; ++i )
63             {
64                 v12 = v10++;
65                 if ( v10 <= 0xE )
66                     break;
67             LABEL_14:
68                 if ( v10 == 15 )
69                     return 1;
70             }
71             v13 = v17[v12];
72             v14 = i;
73             while ( v13 <= v17[v14 + 15] )
74             {
75                 if ( !++v14 )
76                     goto LABEL_14;
77             }
78             LABEL_20:
79             result = 0;
80         }
81     }
82     return result;
83 }
```

(图 4)

(3) 分析 mfta 并写解密函数。mfta 函数主体为 switch-case 代码段，同样分析 case 语句执行，可以发现，程序的目的是将 v17 划分为行列数为 4 的矩阵，等价于拼图游戏，每个拼图上带有数值，可以和相邻拼图块交换位置（矩阵中的数值因此交换）。由图 4 中 while 循环条件可知，最终 v17 应该从小到大排序。

V17={5, 1, 8, 3, 9, 4, 12, 15, 2, 10, 11, 6, 13, 7, 14, 0}

$\rightarrow V17 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$

解数字华容道，得到 a2 序列，根据 a2 还原输入字符串。在 app 中输入正确的序列，回复中即为 flag

The screenshot shows the assembly code for a Java application in the IDA Pro interface. The assembly code is as follows:

```
49     result = a1;
50     switch ( *a1 )
51     {
52         case 0:
53             if ( a2 != 4 && a2 != 1 )
54                 goto LABEL_50;
55             v5 = *a3 ^ a3[a2];
56             *a3 = v5;
57             v6 = a3[a2] ^ v5;
58             a3[a2] = v6;
59             *a3 ^= v6;
60             *a1 = a2;
61             break;
62         case 1:
63             if ( (unsigned int)a2 > 5 )
64                 goto LABEL_50;
65             v7 = 37;
66             if ( _bittest(&v7, a2) )
67                 goto LABEL_50;
68             v8 = a3[1] ^ a3[a2];
69             a3[1] = v8;
70             v9 = a3[a2] ^ v8;
71             a3[a2] = v9;
72             a3[1] ^= v9;
73             *a1 = a2;
74             break;
75         case 2:
76             if ( (unsigned int)a2 > 6 )
77                 goto LABEL_50;
```

The corresponding Java code is:

```
public static void main(String[] args){
    int[] a2=cala2();
    //System.out.println(a2.length);
    StringBuffer input=new StringBuffer();
    for(int i=0;i<a2.length;i++)
        if (a2[i] >= 10) {
            input.append((char) (a2[i] + 87));
        } else {
            input.append((char) (a2[i] + 48));
        }
    System.out.println(input);
}
```

```
four ×
D:\jdk1.8\JDK\bin\java.exe ...
ed9840159ab76aefb7321567326a9def

Process finished with exit code 0
```

Please input the answer here:

0159ab76aefb7321567326a9def

Hint: 本CrackMe需要联网验证，因为联网有一定延迟，
请核对下方信息。请提交回复的flag，而非提交的序列

提交的序列: ed9840159ab76aefb7321567326a9def

回复: flag{c0m_n2bb_we11d0ne}

tryonetry.apk

attacker: 李丹琦

flag: FLAG{381654729210386}

Apk 没有办法正确安装，在 OnCreate 类中看到调用了 finish 函数，发现正是这里导致的。

```
protected void onCreate(Bundle arg4) {
    super.onCreate(arg4);
    ActivityMainBinding v0 = ActivityMainBinding.inflate(this.getLayoutInflater());
    this.binding = v0;
    this.setContentView(v0.getRoot());
    this.binding.button.setOnClickListener(new MyOnClickListener2(this));
    if(!CheckSimulator.getSimulatorInfo(this).isEmpty()) {
        this.finish();
    }
}
```

修改 smali 使得条件判断由 !CheckSimulator.getSimulatorInfo(this).isEmpty() 变成 CheckSimulator.getSimulatorInfo(this).isEmpty(), 再重新生成 apk。

```
165     .line 37
166     .local v1, "simulatorInfo":Ljava/util/List;
167     invoke-interface {v1}, Ljava/util/List;->isEmpty()Z
168
169     move-result v2
170
171     if-eqz v2, :cond_0
172
173     .line 38
174     return-void
175
176     .line 40
177     :cond_0
178     return-void
```

再看 MainActivity，发现主要函数是 check，同时 chock 函数限制了输入的参数应该是 FLAG{} 的形式。接下来我查看 check 的 native 代码：

```
int __cdecl Java_com_example_myapplication_MainActivity_check(_JNIEnv *a1, int a2, int a3)
{
    unsigned __int8 v4; // [esp+16h] [ebp-52h]
    int v5; // [esp+34h] [ebp-34h]
    char v6[16]; // [esp+38h] [ebp-30h] BYREF
    char v7[16]; // [esp+48h] [ebp-20h] BYREF
    unsigned int v8; // [esp+58h] [ebp-10h]

    v8 = __readgsdword(0x14u);
    v5 = a01(a1, a3);
    std::string::basic_string<decltype(nullptr)>(v7, v5);
    std::string::basic_string(v6, v7);
    v4 = a07(v6) & 1;
    std::string::~string((int)v6);
    std::string::~string((int)v7);
    return v4;
}
```

主要调用 a01 和 a07 两个函数。首先查看 a01，发现其应该是将 string 转化成了 char 数组。

```

1 void * __cdecl a01(_JNIEnv *a1, int a2)
2 {
3     int v2; // eax
4     const void *v4; // [esp+28h] [ebp-40h]
5     signed int v5; // [esp+2Ch] [ebp-3Ch]
6     int v6; // [esp+30h] [ebp-38h]
7     char v7; // [esp+38h] [ebp-30h]
8     int v8; // [esp+3Ch] [ebp-2Ch]
9     void *v9; // [esp+40h] [ebp-28h]
10
11    v9 = 0;
12    v8 = _JNIEnv::FindClass(a1, "java/lang/String");
13    v7 = _JNIEnv::NewStringUTF(a1, "UTF-8");
14    v2 = _JNIEnv::GetMethodID(a1, v8, "getBytes", "(Ljava/lang/String;)[B");
15    v6 = _JNIEnv::CallObjectMethod(a1, a2, v2, v7);
16    v5 = _JNIEnv::GetArrayLength(a1, v6);
17    v4 = (const void *)_JNIEnv::GetByteArrayElements(a1, v6, 0);
18    if ( v5 > 0 )
19    {
20        v9 = malloc(v5 + 1);
21        memcpy(v9, v4, v5);
22        *((_BYTE *)v9 + v5) = 0;
23    }
24    _JNIEnv::ReleaseByteArrayElements(a1, v6, v4, 0);
25    return v9;
26}

```

再查看 a07，一波分析下来会发现这就是主函数。首先将 char 转换成 string 类型，然后判断输入是否为数字，并将字符串的前九位和后面部分分别转换成 long long int

```

for ( i = 0; i < (unsigned int)(sub_32B50(a1) - 1); ++i )
{
    if ( i > 4 && (*(char *)sub_32B80(a1, i) < 48 || *(char *)sub_32B80(a1, i) > 57) )
    {
        v9 = rand() % 10 + 48;
        *((_BYTE *)sub_32B80(a1, i)) = v9;
    }
}
sub_32BC0((int)v18, a1, 5, 9);
v8 = std::stoll((int)v18, 0, 10);
std::string::~string((int)v18);
v1 = sub_32B50(a1);
sub_32BC0((int)v17, a1, 14, v1 - 1);
v7 = std::stoll((int)v17, 0, 10);
std::string::~string((int)v17);

```

然后调用 a06，发现这个函数的功能应该是根据 flag 的前九位依据一些算法将 map 数组中对应的位置变成 0。

```

v7 = a1;
for ( i = 0; i < 20; ++i )
{
    map[100 * (v7 % 0x4D) + v7 % 0x4F] = 0;
    v7 = v7 * v7 % 0x4F;
}
v5 = a1;
for ( j = 0; j < 20; ++j )
{
    map[100 * (v5 % 0x1F) + v5 % 0x39] = 0;
    v5 = 2 * v5 % 0x4F;
}
result = a1;
v3 = a1;
for ( k = 0; k < 25; ++k )
{
    a03(v3 % 7, v3 % 9, 0);
    v3 = (k + 7 * v3) % 0x4F;
    result = k + 1;
}
return result;

```

接着调用 a04，将压扁的控制流恢复，发现该函数主要是判断输入的九位数其前 i 位能否被 i 整除，即第一位数字可以被 1 整除，第一二位数字组成的整数可以被 2 整除，在网上查找相关算法时发现这个数应该是 381654729。

(参考: <https://blog.csdn.net/ziyuzhao123/article/details/20803535>)

```

for ( i = 1; i <= 9; ++i )
    cou[2 * i] = i;
for ( j = 1; j <= 9; ++j )
{
    for ( k = 1; k <= 9; ++k )
    {
        a[10 * j + k] = a08(j - 1, k - 1);
        if ( (int)a[10 * j + k] <= 0 )
        {
            ++cou[2 * j + 1];
        }
        else
        {
            gong[10 * a09(j, k) + a[10 * j + k]] = 1;
            lie[10 * k + a[10 * j + k]] = 1;
            hang[10 * j + a[10 * j + k]] = 1;
            v3 = a[10 * j + k];
            have += a10(j, k) * v3;
        }
    }
}
sort(&cou[2], &cou[20], a12);
for ( l = 1; l <= 9; ++l )
{
    for ( m = 1; m <= 9; ++m )
    {
        if ( !a[10 * cou[2 * l] + m] )
        {
            s[4 * u] = cou[2 * l];
            s[4 * u + 1] = m;
            s[4 * u + 2] = a10(cou[2 * l], m);
            v0 = a09(cou[2 * l], m);
            v1 = u++;
            s[4 * v1 + 3] = v0;
        }
    }
}
a11(0, have);
return 73 * most;

```

然后调用 a13，发现该函数应该是将一个 9*9 的数组，数组中每个元素都是 0~9，结合 a06 中将数组的一些位置变为 0，猜测本题与数独有关。并且在第一个 for 循环中，该算法对数组中的每个数字赋予一个权值，最后计算出每个元素和它的权值相乘的和。由于该算法调用了多个函数且很长，我怀疑出题的同学可能是在网上找到算法进行改写，于是我在网上搜索和数独、权值相关的内容，发现了靶形数独和金字塔数独，分别查找相关的代码并比较后发现本题应该是金字塔数独。

(参考 https://blog.csdn.net/qq_42580577/article/details/88920234)

借助网上找到的代码理解后，我用 C++ 根据题目的逻辑解出数独后，计算得到各个元素和它的权值之和，经过一点简单的运算得到最后的 result 应为 210386，也就是 flag 的后半部分。

```
1 // #include<algorithm>
2 // #include <iostream>
3 using namespace std;
4
5 int map[] = { 7, 9, 9, 9, 9, 9, 2, 6, 0, 9, 3, 6, 5, 2, 5, 7, 5, 8, 9, 9, 8, 5, 1, 5, 2,
6 | 7, 8, 3, 6, 5, 5, 1, 3, 9, 9, 0, 6, 2, 5, 0, 9, 2, 2, 5, 7, 2, 7, 0, 7, 1,
7 | 9, 1, 6, 3, 0, 3, 4, 2, 7, 0, 6, 9, 7, 6, 3, 5, 2, 3, 0, 3, 8, 7, 8, 0, 3,
8 | 3, 4, 3, 7, 1, 1, 2, 6, 2, 2, 3, 0, 8, 9, 9, 8, 3, 2, 9, 9, 5, 9, 3, 1, 9,
9 | 7, 5, 2, 5, 8, 7, 6, 0, 1, 1, 5, 0, 3, 0, 0, 0, 1, 1, 6, 4, 6, 3, 7, 9, 0, 3,
10 | 4, 1, 3, 7, 8, 8, 2, 2, 3, 7, 2, 3, 0, 3, 6, 8, 6, 3, 5, 3, 2, 5, 2, 9, 6,
11 | 3, 6, 2, 3, 5, 7, 5, 5, 1, 0, 9, 0, 2, 6, 3, 8, 5, 5, 9, 9, 1, 3, 8, 8,
12 | 2, 0, 8, 9, 9, 6, 0, 5, 5, 0, 2, 3, 2, 6, 7, 7, 7, 2, 8, 3, 5, 4, 7, 5, 1,
13 | 1, 3, 1, 3, 8, 6, 7, 9, 8, 2, 4, 4, 0, 2, 5, 7, 5, 7, 9, 9, 9, 2, 5, 2,
14 | 5, 7, 3, 2, 5, 1, 8, 9, 2, 0 };
15
16 int u, ok, most = -1, have;
17 int cou[20];
18 int gong[100];
19 int lie[100];
20 int hang[100];
21 int s[400];
22
23 int a08(int a1, int a2)
24 {
25     int v3;
26
27     v3 = a2 + 9 * a1;
28     if (v3 <= 40)
29         return map[100 * v3 + 307 + v3];
30     if (v3 >= 90)
31         return -1;
32     return map[100 * (43 - (v3 - 40)) + 7 + v3];
33 }
```

```
35 | int a09(int a1, int a2)
36 | {
37 |     int v3;
38 |
39 |     if (a1 > 3)
40 |     {
41 |         if (a1 > 6)
42 |         {
43 |             if (a2 > 3)
44 |             {
45 |                 if (a2 > 6)
46 |                     v3 = 9;
47 |                 else
48 |                     v3 = 8;
49 |
50 |             else
51 |             {
52 |                 v3 = 7;
53 |             }
54 |         }
55 |         else if (a2 > 3)
56 |         {
57 |             if (a2 > 6)
58 |                 v3 = 6;
59 |             else
60 |                 v3 = 5;
61 |         }
62 |         else
63 |         {
64 |             v3 = 4;
65 |         }
66 |
67 |     else if (a2 > 3)
68 |     {
69 |         if (a2 > 6)
70 |             v3 = 3;
71 |         else
72 |             v3 = 2;
73 |     }
74 |
75 |     else
76 |     {
77 |         v3 = 1;
78 |     }
79 |     return v3;
80 |
81 | int a10(int a1, int a2)
82 | {
83 |     if (a1 == 1 || a2 == 1 || a1 == 9 || a2 == 9)
84 |         return 6;
85 |     if (a1 == 2 || a2 == 2 || a1 == 8 || a2 == 8)
86 |         return 7;
87 |     if (a1 == 3 || a2 == 3 || a1 == 7 || a2 == 7)
88 |         return 8;
89 |     if (a1 == 4 || a2 == 4 || a1 == 6 || a2 == 6)
90 |         return 9;
91 |     return 10;
92 | }
93 |
94 | bool a12(int a1, int a2, int a3, int a4)
95 | {
96 |     return a2 < a4;
97 | }
```

```

99 void dfs(int a1, int a2)
100 {
101     int i;
102
103     if (a1 == u)
104     {
105         if (a2 > most)
106         {
107             most = a2;
108             return;
109         }
110     }
111     else
112     {
113         for (i = 1; i <= 9; ++i)
114         {
115             if (!hang[10 * s[4 * a1] + i] && !lie[10 * s[4 * a1 + 1] + i] && !gong[10 * s[4 * a1 + 3] + i])
116             {
117                 gong[10 * s[4 * a1 + 3] + i] = 1;
118                 lie[10 * s[4 * a1 + 1] + i] = 1;
119                 hang[10 * s[4 * a1] + i] = 1;
120                 dfs(a1 + 1, i * s[4 * a1 + 2] + a2);
121                 gong[10 * s[4 * a1 + 3] + i] = 0;
122                 lie[10 * s[4 * a1 + 1] + i] = 0;
123                 hang[10 * s[4 * a1] + i] = 0;
124             }
125         }
126     }
127     return;
128 }

```

```

int main()
{
    int v0, v1, v3;
    int m, k, l, j, i;
    int a[100];
    int have;
    for (i = 1; i <= 9; ++i)
        cou[2 * i] = i;
    for (j = 1; j <= 9; ++j)
    {
        for (k = 1; k <= 9; ++k)
        {
            a[10 * j + k] = a08(j - 1, k - 1);
            if ((int)a[10 * j + k] <= 0)
            {
                ++cou[2 * j + 1];
            }
            else
            {
                gong[10 * a09(j, k) + a[10 * j + k]] = 1;
                lie[10 * k + a[10 * j + k]] = 1;
                hang[10 * j + a[10 * j + k]] = 1;
                v3 = a[10 * j + k];
                have += a10(j, k) * v3;
            }
        }
    }
    sort(&cou[2], &cou[20], a12);

```

```

158 for (l = 1; l <= 9; ++l)
159 {
160     for (m = 1; m <= 9; ++m)
161     {
162         if (!a[10 * cou[2 * l] + m])
163         {
164             s[4 * u] = cou[2 * l];
165             s[4 * u + 1] = m;
166             s[4 * u + 2] = a10(cou[2 * l], m);
167             v0 = a09(cou[2 * l], m);
168             v1 = u++;
169             s[4 * v1 + 3] = v0;
170         }
171     }
172 }
173 dfs(0, have);
174 printf("%d", 73 * most);
175 }

```

逃离 FDU.apk

attacker: 李丹琦 王楠欣

FLAG{1A3B5C7D9E1F3G5H7I9N1O3P5Q7R9S1T3U5V7W9Y}

(王楠欣)

(1) Jeb 分析可知, BigCheck 的返回值为 true 时, toast 为 right。

分析 BigCheck 函数可知 flag 格式为 FLAG{...}, 虽然代码中设置 BigCheck 传入参数字符串为“haha”, 以及对 return 值的条件做了常量 return false 处理, 使得输入 flag 后不能得到 toast right 的出现效果, 但是因为只需要提交正确的 flag, 可以直接对 JniUtil.check() 进行分析。这需要用到 ida 分析 nativecode 逻辑。

```
boolean v2 = MainActivity.this.BigCheck("haha");
if(user.isEmpty()) {
    Toast.makeText(MainActivity.this, "同学你的用户名是?", 0).show();
    return;
}

if(v2) {
    Toast.makeText(MainActivity.this, "right", 0).show();
    return;
}

public boolean BigCheck(String arg1) {
    byte[] hint = "FLAG{usaprevrwngbzkdyaabqec)aewagsekk)".getBytes(StandardCharsets.UTF_8);
    if(arg1.length() != 46) {
        return false;
    }

    byte[] old = arg1.getBytes(StandardCharsets.UTF_8);
    byte[] newbyte = new byte[46];
    newbyte[0] = 76;
    newbyte[1] = 76;
    newbyte[2] = 65;
    newbyte[3] = 71;
    newbyte[4] = 0x78;
    newbyte[45] = 0x7D;
    int i;
    for(i = 5; i < 45; ++i) {
        newbyte[i] = (byte)(old[i] ^ hint[i]);
    }

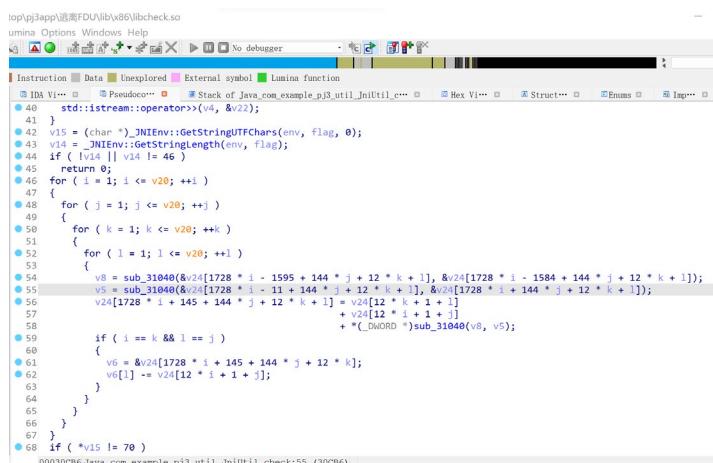
    String v11 = String.valueOf(MainActivity.byteToChar(newbyte));
    return "Let's play a game".length() / 2 - 6 == v11.length() / 2 - 6 ? JniUtil.check(v11) : false;
}

public static char[] byteToChar(byte[] arg4) {
    Charset charset = Charset.forName("ISO-8859-1");
    ByteBuffer byteBuffer = ByteBuffer.allocate(arg4.length);
    byteBuffer.put(arg4);
    byteBuffer.flip();
    return charset.decode(byteBuffer).array();
}

public class JniUtil {
    static {
        System.loadLibrary("check");
    }

    public static native boolean check(String arg0);
}
}
```

(2) ida 分析部分, 需要满足传入 flag 前 5 个字符为 FLAG{, 然后将 v20 传入 check2, 使返回值为 1, 分析 check2



```

if ( *v15 != 70 )
    return 0;
v16 = v15 + 1;
if ( *v16 != 76 )
    return 0;
v17 = v16 + 1;
if ( *v17 != 65 )
    return 0;
v18 = v17 + 1;
if ( *v18 != 71 )
    return 0;
v19 = v18 + 1;
if ( *v19 == 123 )
    v21 = check2(v19 + 1) & 1;
else
    v21 = 0;
return v21;

```

- (3) Check2, 使 v20 值为 1, 分析代码可知不能通过 goto 到 LABEL_69 执行 return, 因为出现 goto 语句的部分均将 v20 赋值为 0, 于是不能满足那些 if 条件,

```

if ( v14 < 0 || v14 > 9 || v13 < 0 || v13 > 9 )
{
    v20 = 0;
    goto LABEL_69;
}
if ( *((_DWORD *)&s[5 * v14] + v13) == 1 )
{
    v20 = 0;
    goto LABEL_69;
}

LABEL_69:
    sub_32DE0(v21);
    return v20;
}

```

(李丹琦)

我直接看到 native 代码中的 check2。首先看到 v20 值应为 1, 且不能通过 goto 来执行 return。即应该使 $v14 == 9 \&& v13 == 9$, 从而执行以下代码:

```

if ( v14 == 9 && v13 == 9 )
    break;
++a1;
}
v27 = a1 + 1;
if ( *v27 == '}' )
    v20 = v27[1] == 0;

```

然后观察与 v14 和 v13 有关的代码。首先从 switch-case 中可以知道 flag 应该是由 U/D/L/R 以及数字组成的, 而且这几个字符显然对应 up/down/left/right。结合下面调用 goto 的条件可知, 要满足 $v14 == 9 \&& v13 == 9$, 需要在循环中 v14、v13 加减时保持在 0~9 中间, 且 s 数组对应元素不为 1。

```

switch (*a1)
{
    case 'U':
        ++a1;
        for ( jj = 0; jj < *a1 - 48; ++jj )
            --v14;
        break;
    case 'D':
        ++a1;
        for ( kk = 0; kk < *a1 - 48; ++kk )
            ++v14;
        break;
    case 'I':
        ++a1;
        for ( ll = 0; ll < *a1 - 48; ++ll )
            --v13;
        break;
    case 'R':
        ++a1;
        for ( mm = 0; mm < *a1 - 48; ++mm )
            ++v13;
        break;
}
if ( v14 < 0 || v14 > 9 || v13 < 0 || v13 > 9 )
{
    v20 = 0;
    goto LABEL_69;
}
if ( *((_DWORD *)&s[5 * v14] + v13) == 1 )
{
    v20 = 0;
    goto LABEL_69;
}

```

查看 s 数组，发现主要调用的地方有三个，且第二个地方与 s 本身无关。S 的值要么是 1 要么是 0，此时我猜测这应该是道类似迷宫的题。

```

memset(s, 0, sizeof(s));
memcpy(dest, &unk_AAAD0, sizeof(dest));
for ( i = 0; i < 33; ++i )
    *((_DWORD *)s[5 * (dest[i] / 10)] + dest[i] % 10) = 1;
sub_32BF0(v21);
for ( j = 0; j < 33; ++j )
{
    if ( *((_DWORD *)s[5 * (dest[j] % 10)] + dest[j] / 10) == 1 )
        sub_32C20(v21, &dest[j]);
}

memcpy(v23, &unk_AAB54, 0x84u);
for ( m = 0; m < 33; ++m )
    *((_DWORD *)s[5 * (v23[m] / 10)] + v23[m] % 10) = 1;

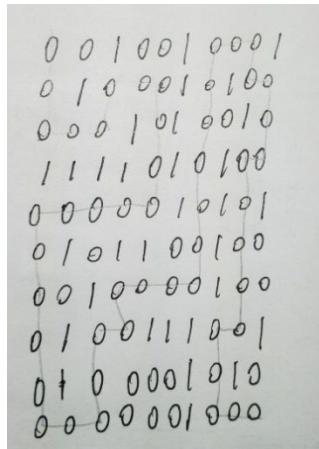
```

查看两个给 s 赋值的数组并且计算得到 s，考虑 v14 和 v13 的取值范围，将 s 组织为 10*10 的数组。

```

; int dword_AAAD0[33]
dword_AAAD0    dd 4Ah, 33h, 2Dh, 0Bh, 3Eh, 47h, 0Fh, 19h, 36h, 1Fh, 39h
                ; DATA XREF: check2(char const*)+181o
                dd 4Ch, 23h, 9, 43h, 56h, 5, 59h, 20h, 4Fh, 2Fh, 25h, 1Eh
                dd 17h, 35h, 1Ch, 60h, 51h, 31h, 11h, 21h, 4Bh
; int dword_AAB54[33]
dword_AAB54    dd 43h, 1Fh, 0Bh, 2, 11h, 4Ch, 31h, 17h, 21h, 4Ah, 36h
                ; DATA XREF: check2(char const*)+3441o
                dd 2Dh, 56h, 4Bh, 1Ch, 5, 19h, 33h, 25h, 3Eh, 59h, 20h
                dd 47h, 51h, 4Fh, 39h, 35h, 23h, 0Fh, 1Eh, 9, 2Fh, 60h

```



得到字符串应为 D2R2U1R2D3L4D5R3U3R3U6R2D1R1D2L1D4L1D2R2，再与 BlackBox 中的字符串异或，得到 flag 是 FLAG{1A3B5C7D9E1F3G5H7I9N1O3P5Q7R9S1T3U5V7W9Y}

Loopy

attacker: 杨舒晗

FLAG{PiIgkciDGTXjn@IRmQDPqmhNmmsxy{{20220613180102030000000000000000}}

Flag 长度为 70，从 native code 可以看出格式为 FLAG{xxx}，先对字符串分段进行处理，将字符串转换为 byte 数组，flag[5:20] 与 "YP\\W[SYtwdhZ^p\\" 异或，flag[21:36] 与 "b]at'A]X~]]CHIKK" 异或，用 calendar.get 获取当前年月日以及小时拼接成字符串并在末位加"010203"得到一个字符串转换为字节数组记为 when，flag[37:53]中每一个字符为 when[i] - f_3[i] + '0' 得到，对 54 至 68 位字符没有处理。然后用 native lib 中的 check 函数检查处理过的字符串，需要使 flag[5] 为字符 0,1,4,5 中的任意一个进入第一层 while 循环，然后 flag[5:68] 要通过 iamafunctiOn 检查。iamafunctiOn 将长为 64 的字符串分为八个字符一组，要求最后一组不能有字符'1'或'3'；若一组中的第 8+j 个元素为'1'或'3'，则最后一个元素必须为偶数，且满足：

- 1) a[j-7]==2 or 3; a[j+1]<='0'
- 2) a[j-7] !=2 or 3; a[j+1]>'0' && !='3'
- 3) 若 j=-8, a[line+1][j] >'0' && !='3'
- 4) 若 j!= -8, ① a[j+7] == 1 or 3; a[line+1][j]<='0'
 ② a[j+7] != 1 or 3; a[line+1][j]>'0' && !='3'

发现构造 64 个字符全为 0 的字符串可以通过该函数检查，此外，这可以满足 check 函数中的第二层 while 循环 break 的条件通过检查。经过 apk 处理后得到所有字符均为 0 的字符串按顺序拼接放到给定格式中即为 flag

听说你也无能狂怒那我就放心了

attacker: 杨舒晗

FLAG{ABaijt&zjEybCbtAsdxtXiaDo'Bsd0AgSawamura}

首页 LOGIN 无法进入，用 monitor 检测点击事件对应函数

```
Name
· 19 com.example.pj3.MainActivity$1.onClick (Landroid/view/View;)V
  Parents
```

jeb 逆向查看对应函数，发现首页设置有输入框但 apk 中无法看到，注意到 yourself 函数，返回 4，4 对应的 visibility 为 INVISIBLE，需要修改为 0 才是 VISIBLE 用 Apktool_M_v2.4.0-220522.apk 反编译 apk 修改 smali 文件中 yourself 的返回值为 0 后重打包签名安装可以得到输入框。

```
this.name_EditText.setVisibility(this.yourself());
this.pass_EditText.setVisibility(this.yourself());
```

```
.method private yourself()I
  .locals 1
  .line 28
  const/4 v0, 0x0
  return v0
.end method
```

你必须看清你自己才能开始登录！

姓名

密码

LOGIN

姓名为“admin”，密码通过反射调用函数检查，密码的前三个字符为函数所在的类名。观察 MainActivity 所在包发现，单字符命名的自定义类只有 A,B,C,D 四个类，推测会调用其中的函数检查。tmp 初始为 6，接收函数返回值，通过三次反射调用函数并将 tmp 和 a[i]作为参数，更新 tmp，值最后需要为 24。A,B,C,D 中都只有一个 get 函数，进行了控制流混淆，其中 A 只有四个 case，一个 default，根据控制流可以判断返回对传入的参数之和。考虑到首页可能与 flag 无关，分析 C,D 的控制流困难，修改 smali 使最后的条件判断改为 tmp 等于 21 (0x15)，即将 tmp=6 与 a={4,3,8}中依次累加可以得到 21，此时输入 AAA 登录，进入 flag 检查页面。

```
MainActivity.this.tmp = (int)((Integer)method.invoke(clazz.newInstance(), ((int)MainActivity.this.tmp), ((int)MainActivity.this.a[i])));  
  
if(MainActivity.this.tmp == 24) {  
    Toast.makeText(MainActivity.this, "登录成功！", 0).show();  
  
    igit v4, v4, Lcom/example/pj3/MainActivity;->tmp:I  
    const/16 v5, 0x15  
    if-ne v4, v5, :cond_2
```

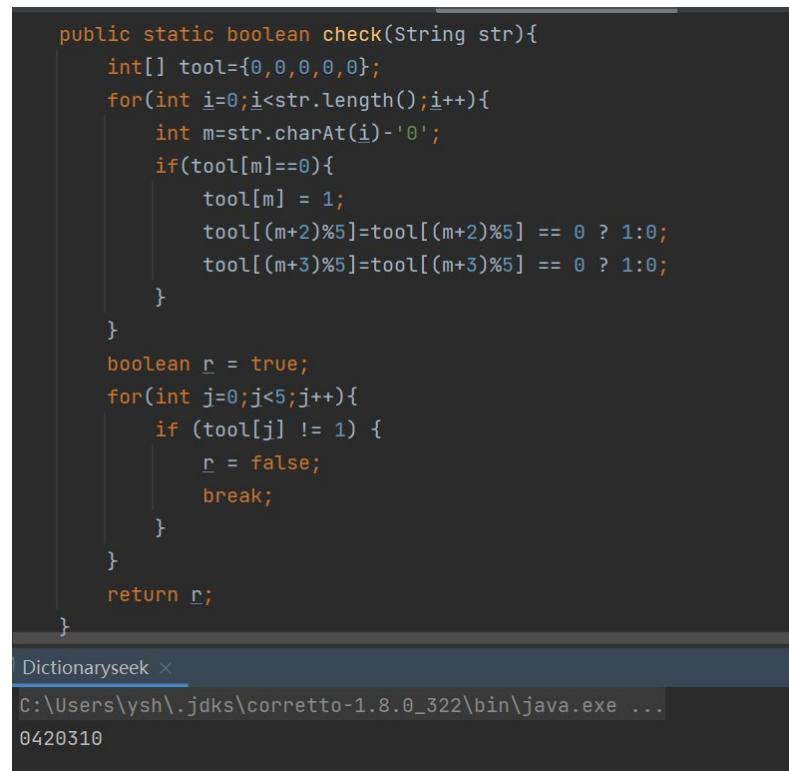
之后 flag 检查在 FlagActivity 中。

```
19 com.example.pj3.FlagActivity$1.onClick (Landroid/view/View;
```

flag 长度 46，格式 FLAG{xxx}，检查核心逻辑在 native code 中。无法 f5，修改垃圾代码后得到函数伪代码。首先将 flag 中下标为 5,14,17,20,28,31,35 的七个字符进行处理，函数 pj 进行检查。

函数 pj：记传入的字符数组为 a，要求字符为 0-4 中的数字，且 a[0]为 0，a[1]为 3 or 4 a[3]为 0 or 1，a[5]为 2, 3 or 4，a 中元素指示从 tool 开始的偏移量。指针 tool 在函数中没有初始化，发现为.bss，全局变量，应该默认初始化为 0，最后要求 tool 前五个元素为 1。当 tool[i]为 0 时置 tool[i]为 1，且 tool[(i+2)%5]和 tool[(i+3)%5]取反，反之 tool[i]清零。由于长度限制为 7，直接利用字典{'0','1','2','3','4'}根据条件穷举七位数得到唯一结果：0420310

```
if(str.length()==7&&check(str)&&str.charAt(0)=='0'&&str.charAt(1)>='2'&&str.charAt(3)<='2'&&str.charAt(5)>='1')  
    System.out.println(str);
```



```
public static boolean check(String str){  
    int[] tool={0,0,0,0,0};  
    for(int i=0;i<str.length();i++){  
        int m=str.charAt(i)-'0';  
        if(tool[m]==0){  
            tool[m] = 1;  
            tool[(m+2)%5]=tool[(m+2)%5] == 0 ? 1:0;  
            tool[(m+3)%5]=tool[(m+3)%5] == 0 ? 1:0;  
        }  
    }  
    boolean r = true;  
    for(int j=0;j<5;j++){  
        if (tool[j] != 1) {  
            r = false;  
            break;  
        }  
    }  
    return r;  
}  
  
Dictionaryseek x  
C:\Users\ysh\.jdks\corretto-1.8.0_322\bin\java.exe ...  
0420310
```

根据对 flag 中七个字符的处理可以推断, flag[5] = 'A', flag[14] = 'E', flag[17] = 'C', flag[20] = 'A', flag[28] = 'D', flag[31] = 'B', flag[35] = 'A'

```
*v3 = flag_[5] - flag_[2];
v3[1] = flag_[14] - flag_[5];
v3[2] = flag_[17] - flag_[5];
v3[3] = flag_[20] - flag_[5];
v3[4] = flag_[28] - flag_[20];
v3[5] = flag_[31] - flag_[20];
v3[6] = flag_[35] - flag_[20];
v7 = v3;
```

之后, 函数 is 设置了一个 5*5 的 01 矩阵

1	0	0	0	0
0	0	0 (终点)	0	0
0	1	0	0	1
0	1	0	0	0
0 (起点)	1	0	0	0

之后, 将 flag[6]至 flag[13]的字符串与 flag[15]到 flag[44]的字符串拼接得到 v18, 长度为 38, 然后分别将 v18 从 0 开始长度为 19 的字符串 v13 以及从 19 开始的字符串后半部分 v12 分别传入 difficult 和 right 进行检查。

```
if ( (pj(flag7_) & 1) != 0 )
{
    is();
    sub_9630((int)v17, (int)ptr, 6, 8);
    v4 = sub_96C0((int)ptr);
    sub_9630((int)v16, (int)ptr, 15, v4 - 16);
    sub_95C0(v18);
    std::string::~string(v16);
    std::string::~string(v17);
    sub_9630((int)v15, (int)v18, 0, 19);
    v5 = sub_96C0((int)v18);
    sub_9630((int)v14, (int)v18, 19, v5 - 19);
    std::string::basic_string(v13, v15);
    v8 = (char *)difficult((int)v13, trans1);
    std::string::~string(v13);
    std::string::basic_string(v12, v14);
    v7 = (char *)difficult((int)v12, trans2);
    std::string::~string(v12);
    re = (right(v8) & 1) != 0 && (right(v7) & 1) != 0;
```

函数 difficult 中, 若字符串中下标为 8 的字符为'y'则替换为's', 否则与 trans 中对应字符异或。通过分析函数 right 得知字符串异或的结果必须由 'w', 'a', 's', 'd' 组成, 用 'w', 'a', 's', 'd' 分别控制方向上左下右, 在 is 设置的矩阵中移动, 从坐标 (0,4), 即第五行第一列开始, 要求不能超过 5*5 的范围, 并且不能走到值为 1 的位置上, 并且要经过所有非 1 的格子, 到终点 (2,1), 容易得到满足要求的路线为 wwwdwdddssassdasaawww

```

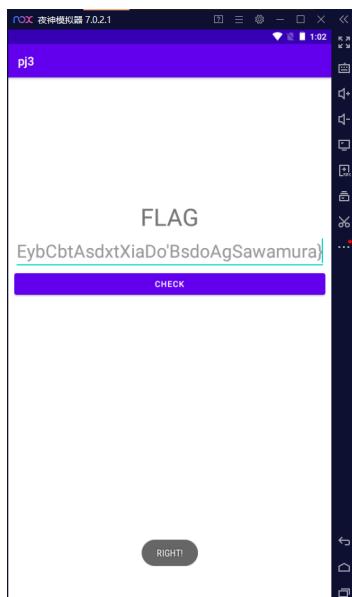
● 10 x = 0;
● 11 y = 4;
● 12 for ( i = 0; i < 19; ++i )
● 13 {
● 14     if ( a1[i] == 'w' )
● 15         --y;
● 16     if ( a1[i] == 'a' )
● 17         --x;
● 18     if ( a1[i] == 's' )
● 19         ++y;
● 20     if ( a1[i] == 'd' )
● 21         ++x;
● 22     if ( x < 0 || x > 4 || y < 0 || y > 4 || a[5 * x + y] == 1 )
● 23         goto LABEL_27;
● 24     a[5 * x + y] = 1;
● 25 }
● 26 if ( x != 2 || y != 1 )
● 27 {
● 28 LABEL_27:
● 29     is();
● 30     v7 = 0;
● 31     return v7 & 1;
● 32 }
● 33 for ( j = 0; j < 5; ++j )
● 34 {
● 35     for ( k = 0; k < 5; ++k )
● 36     {
● 37         if ( !a[5 * j + k] )
● 38             goto LABEL_27;
● 39 }

```

1	0	0	0	0
0	0	0 (终点)	0	0
0	1	0	0	1
0	1	0	0	0
0 (起点)	1	0	0	0

将 wwwdwdddssassdsaawww 分别与给定的字符串进行异或，需要注意的是第一个's'在字符串中的下标刚好为 8，因此 flag 的对应位置字符为'y'。

进行字符串拼接，“FLAG{A” + “Baijt&zj” + “E” + “ybCbtAsdxtXiaDo'BsdoAgSawamura” + “}”得到 flag



“PJ3”

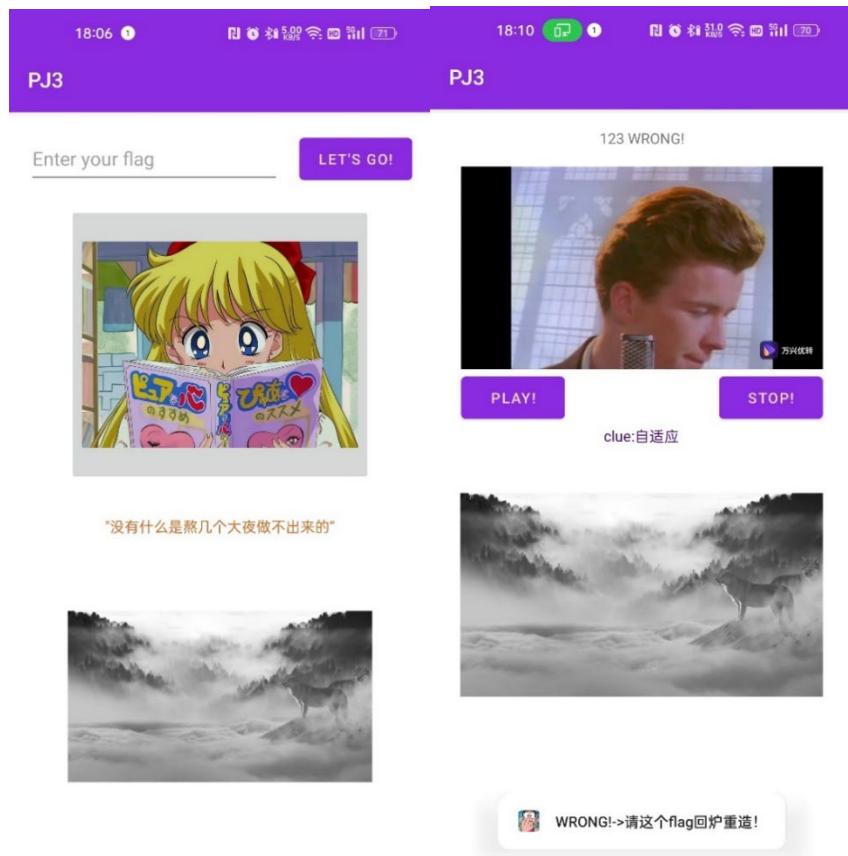
attacker: 李昊霖 (完成对 apk 的全部逆向并破解)

杨舒晗 (前期尝试分析但未破解成功, 对部分细节加以补充)

Flag: little potatoes dream big

(李昊霖的完整分析)

在模拟器里尝试打开 app, 老是疯狂闪退, 数次尝试以后直接发到手机上安装了, 没想到居然成功打开, 推测可能是加入了检测 frida/xposed 等反逆向模块导致, 从 app 界面可以看到需要输入 flag, 点击按钮后, 噢…



使用 jeb 打开查看代码的逻辑, 发现确实加入了 xposed 检测的部分:

```
installedPackages = packageManager.getInstalledPackages();
if(installedPackages != null) {
    linkedList.add("xposed");
    linkedList.add("supersu");
    linkedList.add("genymotion.super");
    linkedList.add("superuser");
    linkedList.add("io.v.a.exposed");
    goto label_19;
}
```

检查 mainactivity, 找到 flag 的检查部分, 可以看到分为 right 和 true_right 两部分, 而 true_right 还要求 state 值非 0, 想要修改 state 值, 需要先进入 right, 由此, 先进行 right 条件的破解

```

public void validate(View arg9) {
    Intent intent = new Intent(this, xxxxXxxAxaaxxa.class);
    String flag = ((EditText)this.findViewById(0x7F080098)).getText().toString(); // id:editTextTextPerso
    StringBuider sb = new StringBuider(flag);
    if((new xxxxXxxxxXaaaAa().check(flag).booleanValue()) && this.state != 0) {
        Toast.makeText(this.getApplicationContext(), "RIGHT!->你可以歇一会儿了!", 0).show();
        sb.append(" ").append("True_Right");
        this.state = 2;
    }
    else if(new xxxxXxxAaaaxa().check(flag)) {
        Toast.makeText(this.getApplicationContext(), "RIGHT!->请继续你的表演!", 0).show();
        sb.append(" ").append("RIGHT!");
        this.state = 1;
    }
    else {
        Toast.makeText(this.getApplicationContext(), "WRONG!->请这个flag回炉重造!", 0).show();
        sb.append(" ").append("WRONG!");
        this.state = 0;
    }
    intent.putExtra("com.example.PJ3.extramessage", String.valueOf(sb));
    this.startActivity(intent);
}
}

```

打开 check 函数，发现为 native 代码：IDA 启动！

```

public class xxXxxXxxAaaaxa {
    static {
        System.loadLibrary("native-lib");
    }

    public native boolean check(String arg1);
}
}

```

在 IDA 中找到 check 函数：

```

if ( (sub_1C4E0(v39, "FLAG{ ") & 1) == 0 )
{
    sub_1C520(v38);
    v21 = 1;
    v10 = sub_1C4E0(v38, "[");
}

```

从这里可以推断 flag 格式：FLAG{XXXX}

```

for ( i = 0; i < v24; ++i )
{
    if ( *(_BYTE *)sub_1C4B0(v40, i) == 32 )
        ++v23;
}

```

从这里 ASCII 码 32 为空格（blankspace），根据 v23=12 才能进入接下来的代码推断 flag 中含有 12 个空格

其后的代码逻辑比较混乱，经过修改变量名和变量类型后，可以大概看出：对划分后的字符串块分第一个后剩下的进行两个 check 方法的检查

```

firt_string = (void *)string_at(split_input, 1);
first_string_ = sub_1C810(firt_string);
String_class = _JNIEnv::FindClass(env, "java/lang/String");
String_ini = _JNIEnv::GetMethodID(env, String_class, "<init>", "([BLjava/lang/String;)V");
v9 = _strlen_chk(first_string_, -1);
v16 = _JNIEnv::NewByteArray(env, v9);
v8 = _strlen_chk(first_string_, -1);
(JNIEnv::SetByteArrayRegion(env, v16, 0, v8, first_string_));
v15 = _JNIEnv::NewStringUTF(env, "GB2312");
first_string__ = _JNIEnv::NewObject(env, String_class, String_ini, v16, v15);
if ( _JNIEnv::CallObjectMethod(env, object1, method_check1, first_string_) )

```

```

{
    for ( j = 2; j <= 11; ++j )
    {
        j_th_string = (void *)string_at(split_input, j);
        v33 = sub_1C810(j_th_string);
        v7 = __strlen_chk(v33, -1);
        bytearry = _JNIEnv::NewByteArray(env, v7);
        stringlength = __strlen_chk(v33, -1);
        _JNIEnv::SetByteArrayRegion(env, v16, 0, stringlength, v33);
        v11 = _JNIEnv::NewObject(env, String_class, String_ini, bytearry, v15);
        if ( !_JNIEnv::CallObjectMethod(env, object2, method_check2, v11, j - 2) )
        {
            result = 0;
            goto LABEL_28;
        }
    }
}

```

只有两个 check 都通过，才能返回 1:

```

116     if ( !_JNIEnv::CallObjectMethod(env, object2, method_check2, v11, j - 2) )
117     {
118         result = 0;
119         goto LABEL_28;
120     }
121     result = 1;
122 }
123 else
124 {
125     result = 0;
126 }
127
128 LABEL_28:
129 std::string::~string(v36);
130 sub_1CA40(split_input);
131
132 }
133 else
134 {
135     result = 0;
136 }
137 std::string::~string(input__);
138 return result;
139

```

而两个 check 方法来自 java 对象，回到 jeb 根据路径寻找

```

v39 = __readgsdword(0x14u);
v31 = _JNIEnv::FindClass(env, "com/example/PJ3/xxxxxxAaaaxaa");
object1 = _JNIEnv::AllocObject(env, v31);
method_check1 = _JNIEnv::GetMethodID(env, v31, "check", "(Ljava/lang/String;)Ljava/lang/Boolean;");
v28 = _JNIEnv::FindClass(env, "com/example/PJ3/xXXXXXAAaaxaaAa");
object2 = _JNIEnv::AllocObject(env, v28);
method_check2 = _JNIEnv::GetMethodID(env, v28, "check", "(Ljava/lang/String;I)Ljava/lang/Boolean;");


```

Check 函数 1:

```

static {
    HashMap v0 = new HashMap();
    xxxxXXxaAaaaxaa.hashmap = v0;
    v0.put(Character.valueOf('1'), Integer.valueOf(1));
    xxxxXXxaAaaaxaa.hashmap.put(Character.valueOf('2'), Integer.valueOf(2));
    xxxxXXxaAaaaxaa.hashmap.put(Character.valueOf('3'), Integer.valueOf(3));
    xxxxXXxaAaaaxaa.hashmap.put(Character.valueOf('4'), Integer.valueOf(4));
    xxxxXXxaAaaaxaa.hashmap.put(Character.valueOf('5'), Integer.valueOf(5));
    xxxxXXxaAaaaxaa.hashmap.put(Character.valueOf('6'), Integer.valueOf(6));
    xxxxXXxaAaaaxaa.hashmap.put(Character.valueOf('7'), Integer.valueOf(7));
    xxxxXXxaAaaaxaa.hashmap.put(Character.valueOf('8'), Integer.valueOf(8));
    xxxxXXxaAaaaxaa.hashmap.put(Character.valueOf('9'), Integer.valueOf(9));
    xxxxXXxaAaaaxaa.hashmap.put(Character.valueOf('X'), Integer.valueOf(10));
    xxxxXXxaAaaaxaa.hashmap.put(Character.valueOf('J'), Integer.valueOf(11));
    xxxxXXxaAaaaxaa.hashmap.put(Character.valueOf('Q'), Integer.valueOf(12));
    xxxxXXxaAaaaxaa.hashmap.put(Character.valueOf('K'), Integer.valueOf(13));
}

public Boolean check(String aaaxa) {
    int xxaa = aaaxa.length();
    Boolean v2 = Boolean.valueOf(false);
    if(xxaa != xxxxXXxaAaaaxaa.hashmap.size()) {
        return v2;
    }
}

```

JQK…看得出来是扑克牌爱好者… (栓 Q)

```

if(ax >= class1.hashmap.size() / 2) {
    return Boolean.valueOf(true);
}

```

Return true 的条件： ax 值大于 hashmap 大小的一半； ax 自增的条件：顺利执行其上的代码而不 return false：

```
try {
    while(true) {
        label_33:
        if(ax >= class1.hashmap.size() / 2) {
            return Boolean.valueOf(true);
        }

        if(((int)((Integer)class1.hashmap.get(Character.valueOf(saass.charAt(ax))))) >= ((int)((Integer)class1.h
            return false;
        }

        int v5 = (int)((Integer)class1.hashmap.get(Character.valueOf(saass.charAt(ax))));
        int v6 = (int)((Integer)class1.hashmap.get(Character.valueOf(saass.charAt(ax + 1 + class1.hashmap.size() /
            break;
        }

    }
    catch(Exception e) {
        return false;
    }

    if(v5 >= v6) {
        return false;
    }

    ++ax;
    goto label_33;
}
```

Saass 为 string， aaxa 为链表，重命名一下方便阅读

```
StringBuilder saass = new StringBuilder();
LinkedList aaxa = new LinkedList();
...
```

String 的构成：从输入字符串中隔位取入，直到字符串取完（字符串长度应为 13）

```
int i;
for(i = 0; i < input_length; ++i) {
    if(!class1.hashmap.containsKey(Character.valueOf(input.charAt(i)))) {
        return false;
    }

    List.offer(Character.valueOf(input.charAt(i)));
}

while(input_length != 0) {
    List.offer((Character)List.poll());
    string.append(List.poll());
    --input_length;
}
```

要求逻辑：1、前 7 位递增，后 7 位递减；2、第 i 位小于第 i+7 位

容易组合得到符合要求的序列：123456KQJX987

再按照 string 的构造逻辑写出逆构造函数，得到结果：K182Q394J576X（居然只是第一步的第一个 check 函数。。。写这么难良心不痛吗）

```

public static void main(String []args){
    String y ="123456KQJX987";
    System.out.println(reverse(y));
}

public static String reverse(String str){
    LinkedList<Character> x = new LinkedList<Character>();
    for(int i=0;i!=str.length();i++){
        x.addFirst(str.charAt(str.length()-i-1));
        x.addFirst(x.removeLast());
    }
    return x.toString();
}

```

C:\Users\Ham\.jdks\corretto-1.8.0_322\bin\java.exe ...
[K, 1, 8, 2, Q, 3, 9, 4, J, 5, 7, 6, X]

进入下一个 check 函数：

```

public class class2 {
    public static int[] aaaaAa;

    static {
        class2.aaaaAa = new int[]{1, 5, 5, 5, 5, 9, 10, 11
    }

    public Boolean check(String input, int aa) {
        String v1 = input;

```

先简单的重命名一下方便阅读，函数逻辑如下：

```

static {
    class2.data = new int[]{1, 5, 5, 5, 5, 9, 10, 11, 2, 7, 7, 11, 2, 7, 10, 12, 3, 8, 8, 9, 7, 8, 10, 13, 4, 8, 1
}

public Boolean check(String input, int input_number) {
    String v1 = input;
    XXXXXXXXaAaaax cal = new XXXXXXXXaAaaax();
    HashMap hashmap1 = new HashMap();
    HashMap hashmap2 = new HashMap();
    int i;
    for(i = input_number * 4; i < input_number * 4 + 4; ++i) {
        int count = 1;
        if(hashmap1.containsKey(Integer.valueOf(class2.data[i]))) {
            count = ((int)((Integer)hashmap1.get(Integer.valueOf(class2.data[i])))) + 1;
        }
        hashmap1.put(Integer.valueOf(class2.data[i]), Integer.valueOf(count));
    }
}

```

根据输入键值依次取连续四个 data 里的值生成 hashmap1， hashmap1 记录各值的出现次数

```

    for(i = 0; i < input.length(); ++i) {
        if(v1.charAt(i) >= 0x30 && v1.charAt(i) <= 57) {
            list.offer(Integer.valueOf(v1.charAt(i) - 0x30));
            mark = 1;
        }
        else if(mark == 1) {
            int value = 0;
            if(!list.isEmpty()) {
                while(!list.isEmpty()) {
                    value = value * 10 + ((int)((Integer)list.poll()));
                }
            }
            int count = 1;
            if(hashmap2.containsKey(Integer.valueOf(value))) {
                count = ((int)((Integer)hashmap2.get(Integer.valueOf(value)))) + 1;
            }
            hashmap2.put(Integer.valueOf(value), Integer.valueOf(count));
        }
    }
    else {
        mark = 0;
    }
}

```

从输入的字符串中取数字生成 hashmap2，逻辑同 hashmap1

```

for(i = 0; i < string.length(); ++i) {
    char x = string.charAt(i);
    if(x == 43) {
        x = '-';
    }
    else if(x == 45) {
        x = '/';
    }
    else if(x == 42) {
        x = '+';
    }
    else if(x == 0x2F) {
        x = '*';
    }
    string.append(x);
}

```

对输入字符串作字符替换，都是运算符号替换

```

if(hashmap1.equals(hashmap2)) {
    try {
        if(cal.calculate(new String(string)) == 24) {
            return Boolean.valueOf(true);
        }
    }
    catch(Exception e) {
        return Boolean.valueOf(false);
    }
}

```

若两个 map 值相同，进入 calculate 函数，若返回值为 24，得到 true

```

public double calculate(String expression) {
    Stack resultStack = new Stack();
    this.prepare(expression);
    Collections.reverse(this.aaaaAa);
    while(!this.aaaaAa.isEmpty()) {
        String currentValue = (String)this.aaaaAa.pop();
        if(!this.isOperator(currentValue.charAt(0))) {
            resultStack.push(currentValue);
            continue;
        }

        String secondValue = (String)resultStack.pop();
        resultStack.push(this.calculate(((String)resultStack.pop()), secondValue, currentValue.charAt(0)));
    }

    return Double.parseDouble(((String)resultStack.pop()));
}

```

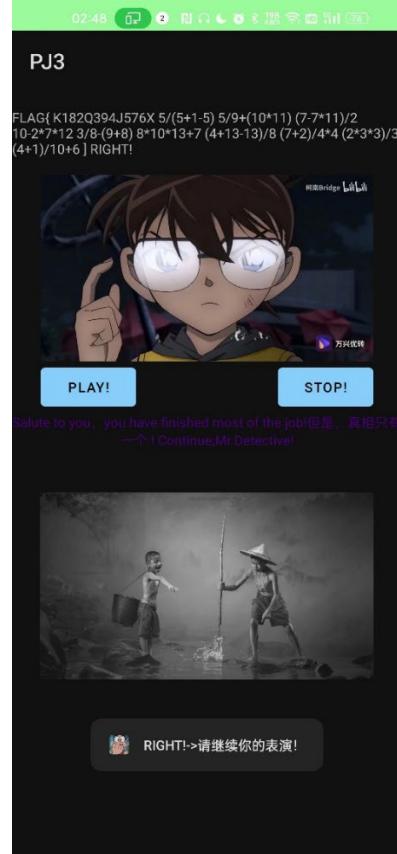
Calculate 函数：能看出来是一个中缀表达式计算函数。综合整个函数逻辑，推测是需要用

data 的值来组成表达式，在经过符号替换后，计算值等于 24，即得到结果（用 hashmap 这种校验方法真的很丑。。yysy，不知道写代码的怎么想的）

由于 data 刚好给出了 40 个现成的数字，只需要给它们加上符合要求的符号即可。在添加符号满足条件后，还需要进行符号转化，相关函数如下：

```
public static String changesign(String str){  
    StringBuilder x = new StringBuilder();  
    for(int i=0;i<str.length();i++){  
        char c=str.charAt(i);  
        if(c=='|') c='+';  
        else if(c=='/') c='-';  
        else if(c=='+') c='*';  
        else if(c=='*') c='/';  
        x.append(c);  
    }  
    return x.toString();  
}  
  
public static void main(String [] args){  
    String[] formula ={ "5*(5-1/5)", "5*9-(10+11)", "(7/7+11)*2", "10/2+7*12", "3*8/(9-8)", "8+10+13-7"  
    , "(4-13/13)*8", "(7-2)*4+4", "(2+3+3)*3", "(4-1)*10-6" };  
    String[] changed = new String[10];  
    for(int i=0;i!=formula.length;i++)  
        changed[i] = changesign(formula[i]);  
    //String str2="5/(5+1-5)";  
    //QuickCalculator qc =new QuickCalculator();  
    for(int i=0;i!=formula.length;i++)  
        System.out.print(changed[i]+ ' ');  
}  
}
```

组装起来，得到了第一个 flag： FLAG{ K182Q394J576X 5/(5+1-5) 5/9+(10*11) (7-7*11)/2
10-2*7*12 3/8-(9+8) 8*10*13+7 (4+13-13)/8 (7+2)/4*4 (2*3*3)/3 (4+1)/10+6]



路漫漫其修远兮…然后进入对应 true-right 的 check 函数：

```
public Boolean check(String str) {  
    return Boolean.valueOf(this.AaaaaAaaa(str).equals(this.aaa));  
}
```

简单的修改变量名后：

```
public Boolean check(String str) {
    return Boolean.valueOf(this.str_process1(str).equals(this.encrypted_key));
}

--- ----- \v
this.encrypted_key = "6173160544155644631716002355038768913851716257602";
```

进入 str_process1

```
public String str_pprocess1(String str) {
    String str1 = this.str_process2(str);
    StringBuilder str3 = new StringBuilder();
    int i;
    for(i = 0; i < str1.length(); ++i) {
        int judge_result = this.judge(0, str1.charAt(i));
        if(str1.charAt(i) >= 97) {
            str3.append(judge_result + 106);
        }
        else {
            str3.append(judge_result + 57);
        }
    }
    int i;
    for(i = str3.length() % 5; i > 0; --i) {
        int x = str3.length() - i;
        int y = "0301".length() - i;
        if(str3.charAt(x) != "0301".charAt(y)) {
            return "6173160544155644631716012355038768913851716257602";
        }
    }
}
```

可以看到，这里的 `return` 是错误的，与最终比较的字符串错开了一位，不能让函数执行到这里

```
private String str_process2(String str) {
    char[] str2char = str.toCharArray();
    StringBuilder result = new StringBuilder();
    int i;
    for(i = 0; i < str2char.Length; ++i) {
        result.append(Integer.toHexString(((char)(str2char[i] ^ 255))));
    }
    return result.toString();
}
```

Str process2. 简单的异或处理:

```
public native int judge(int arg1, int arg2) {
```

Judge, 来自 native 库 (怎么又来…):

利用 IDA 进入到 judge 实体，简单修改后得到：

```
v8[v7] = 0;
v8[v7 + 1] = input1;
v8[v7 + 2] = input2;
v6 = (unsigned __int8 *)&off_49524 - 241265;
while ( 1 )
{
    i = *v6;
    if ( i == 15 )
        break;
    switch ( i )
    {
        case 32:
            v8[v6[1]] = v8[v6[3]] + v8[v6[2]];
            v6 += 4;
            break;
        case 33:
            v8[v6[1]] = v8[v6[2]] - v8[v6[3]];
            v6 += 4;
            break;
        case 34:
            v8[v6[1]] = v8[v6[3]] * v8[v6[2]];
            v6 += 4;
            break;
        case 35:
            v8[v6[1]] = v8[v6[2]] / v8[v6[3]];
            v6 += 4;
            break;
        case 176:
            v8[v6[1] & 0xF] += v8[(int)v6[1] >> 4];
            v6 += 2;
            break;
    }
}
return v8[v6[1]];
```

V6 的位置：

-00000038 var_38	dd ?	; offset
-00000034 var_34	dd ?	
-00000030 var_30	dd ?	
-0000002C var_2C	dd ?	
-00000028 var_28	dd ?	; offset
-00000024 var_24	dd ?	
-00000020 var_20	dd ?	
-0000001C var_1C	dd ?	
-00000018 var_18	dd ?	
-00000014 var_14	dd ?	
-00000010 var_10	dd ?	

V8 的位置：

-00000028 var_28	dd ?
-00000024 var_24	dd ?
-00000020 var_20	dd ?
-0000001C var_1C	dd ?
-00000018 var_18	dd ?
-00000014 var_14	dd ?
-00000010 var_10	dd ?

则可以推断 case 中用到的 v6[3] 和 v6[2] 分别对应放入 v8 的 input1 和 input2。整个 switch 的逻辑是随着 i 的递增分别对两个输入参数执行求和、差、积、商，并依次累加的操作，而原函数中 judge 第一个参数始终为 0，则可以根据参数 2 来推 judge 的返回值为参数 ASCII 码的负数。

```

for(i = 0; i < str1.length(); ++i) {
    int judge_result = this.judge(0, str1.charAt(i));
    if(str1.charAt(i) >= 97) {
        str3.append(judge_result + 106);
    }
    else {
        str3.append(judge_result + 57);
    }
}

```

这里的添加逻辑是：若大于 96（为字母），则将 ASCII 反数加上 106，否则加上 57。
特别的，注意到 48-57 对应 ASCII 中的 0-9，对比上一个，97-106 同样在取反加 106 后得到 0-9 中的数字，由此可以大致归纳出这里的转化逻辑。

```

for(i = str3.length() % 5; i > 0; --i) {
    int x = str3.length() - i;
    int y = "0301".length() - i;
    if(str3.charAt(x) != "0301".charAt(y)) {
        return "6173160544155644631716012355038768913851716257602";
    }
}

```

下面的循环由于要避开 return，忽略就好

```

StringBuilder str*5 = new StringBuilder(str3.substring(0, str3.length() - str3.length() % 5));
StringBuilder str*5_processed = new StringBuilder();
int i;
for(i = 0; i < str*5.length() / 5; ++i) {
    StringBuilder str-tmp = new StringBuilder();
    int j;
    for(j = i; j < str*5.length(); j += str*5.length() / 5) {
        str-tmp.append(str*5.charAt(j));
    }
    str*5_processed.append(this.execute(str-tmp.toString()));
}
return str*5_processed.toString();
}

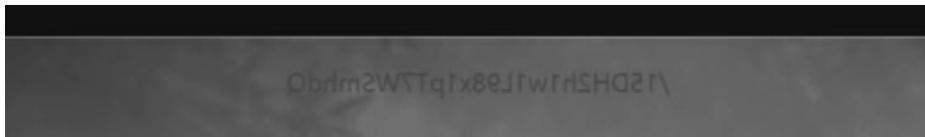
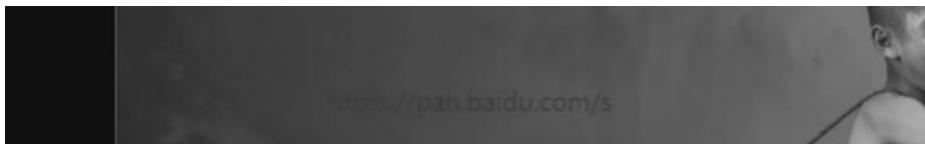
public String execute(String input) {
    BigInteger aaAa = new BigInteger("151");
    BigInteger aAaa = new BigInteger("100889");
    return new BigInteger(input).modPow(aaAa, aAaa).toString();
}

```

首先对 str3 截取 5 的倍数长度，然后两个 for 循环对 str 的内容重排，将内层重排的子字符串进行 execute 运算操作，最终返回值。

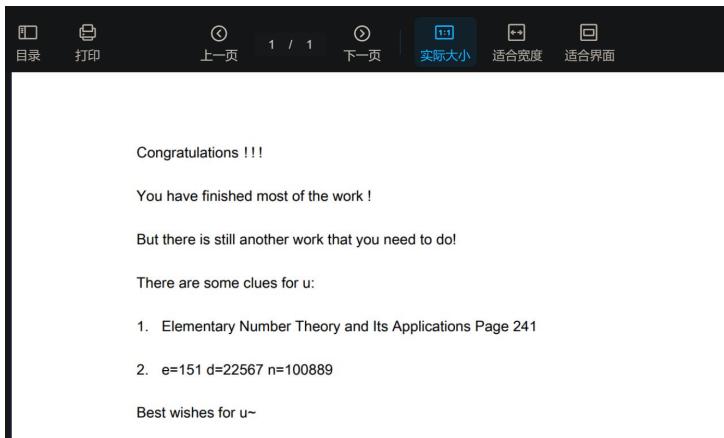
目标明确：逆构造该函数，得到能够在变换后与 encrypted_key 相同的字符串。但还有一个问题无法结局：execute 步骤的指数运算好像……不像是能够逆构造的亚子。

求教了作者之后得到一点提示：翻查之前截图的时候，发现：





尝试把它们拼凑在一起，是一个网址：RSA 加密（栓 Q，真的栓 Q）



则最终得到的逆构造函数思路：反 RSA 加密→重新排布位置→逆 judge 函数+for 循环重新赋值→逆 string_process2 函数→得到原码

构造 RSA 解密函数，还需要得到分组情况，再次回到原 class，发现 len 长得很像

```
static {
    class3.Len = "5555555545";
    System.loadLibrary("native-lib");
}

public class3() {
    this.encrypted_key = "6173160544155644631716002355038768913851716257602";
```

尝试写一下看看：

```
public class test {
    public static void main(String[] args){
        String key = "6173160544155644631716002355038768913851716257602";
        int[] leng = {5,5,5,5,5,5,5,4,5};
        int pointer = 0;
        StringBuilder result = new StringBuilder();
        for(int i=0;i!=leng.length;i++){
            String tmp = key.substring(pointer,pointer+leng[i]);
            StringBuilder tmp2 = new StringBuilder(RSAdecode(tmp));
            while(tmp2.length()!=5)
                tmp2.insert(0, '0');
            System.out.println(tmp2.toString()+":"+pointer);
            result.append(tmp2.toString());
            pointer += leng[i];
        }
        System.out.print(result.toString());
    }

    public static String RSAdecode(String str){
        BigInteger d = new BigInteger("22567");
        BigInteger n = new BigInteger("100889");
        BigInteger result = new BigInteger(str).modPow(d,n);
        return result.toString();
    }
}
```

```
test x
01100:40
68751:44
00060695470610634884110108496610000899930110068751
```

第一步还原出 RSA 加密前的字符串：

```
00060695470610634884110108496610000899930110068751
```

```
public class test {
    public static void main(String[] args){
        String key = "00060695470610634884110108496610000899930110068751";
        String result = ReRank(key);
        System.out.print(result);
    }
    public static String ReRank(String str){
        StringBuilder result = new StringBuilder();
        for(int i=0;i!=5;i++)
            for(int j = i ;j<str.length();j+=5)
                result.append(str.charAt(j));
        return result.toString();
    }
}
```

test ×
C:\Users\Ham\.jdks\corretto-1.8.0_322\bin\java.exe ...
06031818060964140918051809091764081609050764060301
进程已结束,退出代码0

第二步进行重新排列。

第三步需要还原 judge 替换前的值，这步有个关键的问题是不知道哪一些数字对应原本的字母，而哪一些数字又对应原本的数字，卡了很久以后又再次求教作者，得到提示：class 里的 pos 数组（这谁想得到。。。）

```
this.pos = new int[]{5, 7, 11, 12, 13, 15, 19, 21, 23, 27, 29, 30, 0x1F, 33, 35, 37, 39, 42, 43, 45};
```

```
public class test {
    public static void main(String[] args){
        String key = "06031818060964140918051809091764081609050764060301";
        String result = ReValue(key);
        System.out.print(result);
    }
    static int[] pos = new int[]{5,7,11,12,13,15,19,21,23,27,29,30,31,33,35,37,39,42,43,45};
    public static String ReValue(String str) {
        StringBuilder n = new StringBuilder(str);
        int tag=0;
        StringBuilder a = new StringBuilder();
        for(int i = 0; i < n.length(); i++){ //还原十六进制字符串
            if(tag!= pos.length && i == pos[tag]){
                a.append((char)(106-n.charAt(i)+48));
                tag++;
            } else
                a.append((char)(57-n.charAt(i)+48));
        }
        return a.toString();
    }
}
```

test ×
C:\Users\Ham\.jdks\corretto-1.8.0_322\bin\java.exe ...
93968b8b939adf8f908b9e8b909a8cdf9b8d9a9e92df9d9698

最终得到：93968b8b939adf8f908b9e8b909a8cdf9b8d9a9e92df9d9698；

来到最后一步：

```

private String str_process2(String str) {
    char[] str2char = str.toCharArray();
    StringBuilder result = new StringBuilder();
    int i;
    for(i = 0; i < str2char.Length; ++i) {
        result.append(Integer.toHexString(((char)(str2char[i] ^ 255))));
    }

    return result.toString();
}

```

可以看到函数将原输入语句转换成为了 16 进制的 string 语句，需要将其对应的变换回去：

```

public class test {
    public static void main(String[] args){
        String key = "93968b8b939adf8f908b9e8b909a8cdf9b8d9a9e92df9d9698";
        String result = Hex2String(key);
        System.out.print(result);
    }
}

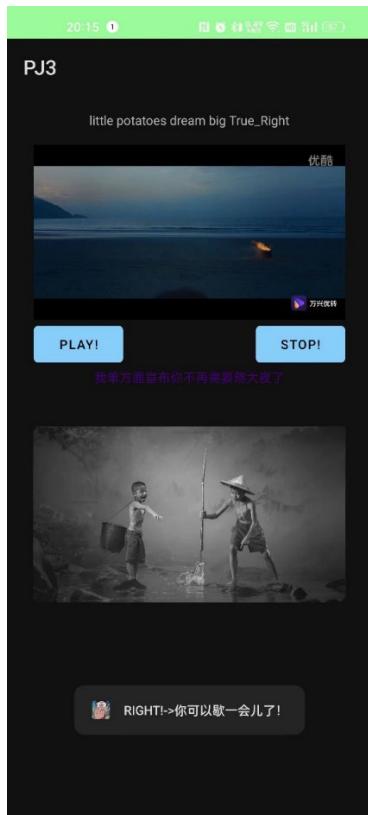
public static String Hex2String(String str){
    StringBuilder s = new StringBuilder();
    for(int i=0;i<str.length()-1;i+=2)
        s.append((char)(Integer.parseInt(str.substring(i,i+2), radix: 16)^255));
    return s.toString();
}

```

test ×
C:\Users\Ham\.jdks\corretto-1.8.0_322\bin\java.exe ...
little potatoes dream big

得到最终的答案： little potatoes dream big

下班！



(杨舒晗：对于 apk 闪退以及对 flag True_Right 检查的分析补充)

Apk 闪退，由于 MainActivity 中总会执行到下图中的 finish() 将进程的活动推向后台，然后执行 System.exit(0) 杀死整个进程。在模拟器上安装 Apktool_M_v2.4.0-220522.apk 用来反编译 PJ3.apk 修改 smali 使 if 判断为假，然后重打包签名安装，可以正确进入 apk 页面

```
.line 53
.local v3, "e4":Ljava/lang/Exception;
invoke-virtual {v3}, Ljava/lang/Exception:->printStackTrace()V

.line 55
.end local v3    # "e4":Ljava/lang/Exception;
:goto_4
if-nez v2, :cond_4

.line 56
sget-object v3, Ljava/lang/System;->out:Ljava/io/PrintStream;
const-string v4, "error!"

invoke-virtual {v3, v4}, Ljava/io/PrintStream:->println(Ljava/lang/String;)V

.line 57
invoke-virtual {p0}, Lcom/example/PJ3/MainActivity:->finish()V

.line 58
const/4 v3, 0x0
invoke-static {v3}, Ljava/lang/System;->exit(I)V

if (z2) {
    System.out.println("error!");
    finish();
    System.exit(0);
}
```

观察发现 MainActivity 中有一个变量 state，初始为 0，提示“WRONG”时设置 state=0，“RIGHT”时设置 state=1，“True_Right”设置 state=2。为绕开第一重“RIGHT”修改 smali 设置 state 初始为 1，判断错误后也为 1，重新打包

```
if (new xxxXXXXXXaaAa().check(flag).booleanValue() && this.state != 0) {
    Toast.makeText(getApplicationContext(), "RIGHT!->你可以歇一会儿了！", 0).show();
    sb.append(" ").append("True_Right");
    this.state = 2;
} else if (new xxXxxXxaAaaaxa().check(flag)) {
    Toast.makeText(getApplicationContext(), "RIGHT!->请继续你的表演！", 0).show();
    sb.append(" ").append(right);
    this.state = 1;
} else {
    Toast.makeText(getApplicationContext(), "WRONG!->请这个flag回炉重造！", 0).show();
    sb.append(" ").append(wrong);
    this.state = 0;
}
```

要得到“True_Right”，需要通过 xxxXXXXXXaaAa().check 对字符串的检验
xxxXXXXXXaaAa() 中的 judge 函数实际调用 native code hahahaha 函数，返回作为参数传入的字符的负值。对输入的 flag 的总体处理是将 flag 的字符每个字节取反得到字节数组 s，然后调用函数 judge 得到 s[i] 的负值，

```
if(s[i] >= 'a')    sb+=106- s[i]
else      sb+=57- s[i]
```

由此得到由字符 0-9 组成的字符串 sb.toString()

然后若字符串长度 len 不是 5 的倍数，则末尾的 len%5 的个字符需对应“0301”中的后 len%5 个（即若 len%5==1，则末尾为‘1’；若 len%5==2，则末尾为‘01’）

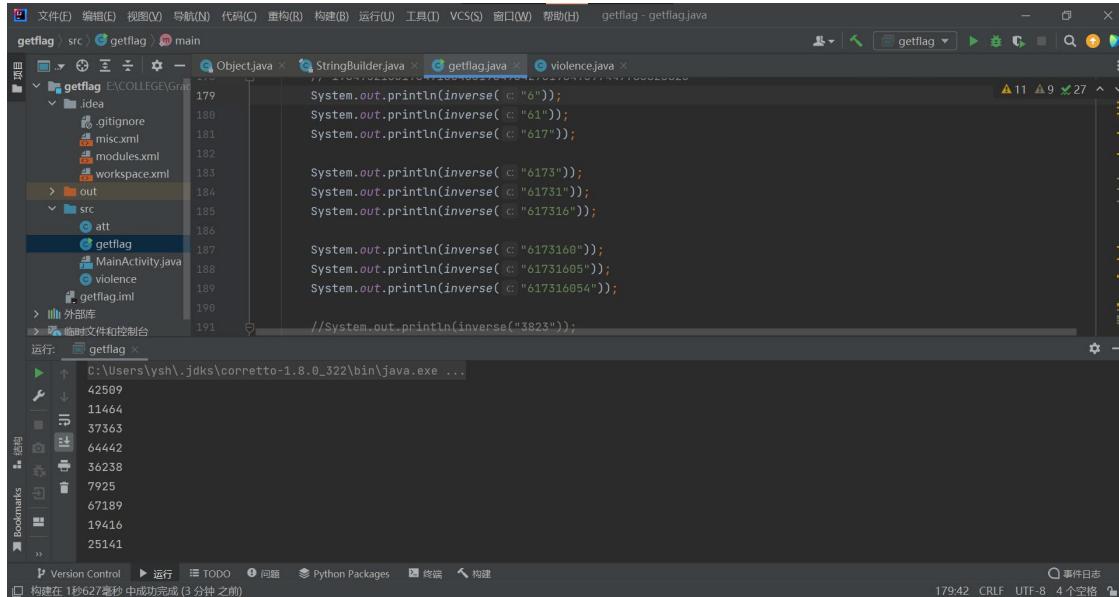
然后舍去字符串中长度超出 5 的倍数的部分，记为 str，长度为 length，记 length/5=k，将字符串分为 k 组，进行两重循环，外层循环遍历 str 中下标为 0 到 k-1 的字符；对其中的每个字符 str[i](i=0,1,..,k-1)，进行内层循环，取迭代的 m 初始为 i，步长为 k，每取到一个附加到 StringBuilder 的末尾，取遍 str 后得到将得到字符串用 exponentEncode 进行指数加密 (e=151,m=10089)

```
int i;
for(i = 0; i < str.length() / 5; ++i) {
    StringBuilder seg = new StringBuilder();
    int m;
    for(m = i; m < str.length(); m += str.length() / 5) {
        seg.append((char)str.charAt(m));
    }
    sb.append(this.exponentEncode(seg.toString()));
}

public String exponentEncode(String Aaaa) {
    BigInteger aaAa = new BigInteger("151");
    BigInteger aAaa = new BigInteger("10089");
    return new BigInteger(Aaaa).modPow(aaAa, aAaa).toString();
}
```

指数加密后得到的字符串需要与给定的字符串

aaaa="6173160544155644631716002355038768913851716257602"相等，我之前尝试对aaaa分组进行指数加密的逆运算，考虑到可显示字符以及ascii码只有7位，可以排除指数加密的逆运算得到的字符串中下标为偶数的字符为'2', '3', '8', '9'的可能性，但后来情况比较多，实在难以得到正确答案



```
getflag.java
179 System.out.println(inverse("6"));
180 System.out.println(inverse("61"));
181 System.out.println(inverse("617"));
182 System.out.println(inverse("6173"));
183 System.out.println(inverse("61731"));
184 System.out.println(inverse("617316"));
185 System.out.println(inverse("6173160"));
186 System.out.println(inverse("61731605"));
187 System.out.println(inverse("617316054"));
188 //System.out.println(inverse("3823"));

violence.java
179:27
```

加之即便进行逆运算筛选出唯一的答案后还需要进一步译码，由于得到的字符串中0-9的数字也有如下并非完全一一对应的关系（如'4'可能对应flag转换后的字节数组中的'a'或'0'），译码复杂，最终没有成功

0	1	2	3	4	5	6	7	8	9
6	7	8	9	a	b	c	d	e	f
				0	1	2	3	4	5

【注】通过李昊霖的解题思路了解到需要通过第一重'RIGHT'检查后从跳转的新页面的下方图片中获取提示，其实这可以通过修改smali，将输入错误时传入新页面的intent的extramessage由“WRONG!”改为“RIGHT！”，这样就能在输入错误时直接跳转到需要的页面，获得提示，而避免第一重'RIGHT'检查（不过关键在于其实很难想到从图片中获取提示）

```
.line 81
:cond_1
invoke-virtual {p0}, Lcom/example/PJ3/MainActivity:->getApplicationContext()Landroid/content/Context;
move-result-object v4

const-string v7, "WRONG!->\u8bf7\u8fd9\u4e2aflag\u56de\u7089\u91cd\u9020\uff01"

invoke-static {v4, v7, v6}, Landroid/widget/Toast:->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I
move-result-object v4

invoke-virtual {v4}, Landroid/widget/Toast;->show()V

.line 82
invoke-virtual {v3, v5}, Ljava/lang/StringBuilder:->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v4

const-string v5, "RIGHT!" RIGHT!

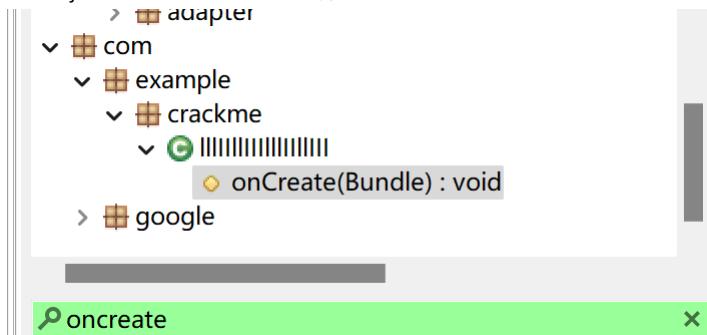
invoke-virtual {v4, v5}, Ljava/lang/StringBuilder:->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

“奥特曼”

attacker: 李昊霖

FLAG{oarrumjb11235813B0gJ1pJ0cM3fJ0gB}

先进 jeb 找到 oncreate 函数:



可以同一个 class 下找到输入判断逻辑，简单重命名后得到:

```
public void onClick(View v) {
    if(v.getId() == 0x7F080058) { // id:button
        String inputText = this.editText.getText().toString();
        Boolean result = class1.check(inputText);
        if(!Boolean.valueOf(class2.format(inputText)).booleanValue()) {
            Toast.makeText(this.getApplicationContext(), "WRONG FORMAT!", 0).show();
            return;
        }
        if(!result.booleanValue()) {
            result = Boolean.valueOf(Boolean.valueOf(class2.format(inputText)).booleanValue() | result.bool
        }
        if((result.booleanValue()) && (class1.divide(inputText).booleanValue())) {
            Toast.makeText(this.getApplicationContext(), "RIGHT!", 0).show();
            return;
        }
        Toast.makeText(this.getApplicationContext(), "WRONG!", 0).show();
    }
}
```

首先要通过 format 检验，然后在 result 和 divide 同时返回 true 的情况下，取得 flag：

值得注意的是，中间插入的 if 语句使得 result 的值不再重要，实际上由 format 决定：

```
if(!result.booleanValue()) {
    result = Boolean.valueOf(Boolean.valueOf(class2.format(inputText)).booleanValue() | result.booleanValue())
}
```

Format 是一个 native 函数：

```
public static native boolean format(String arg0) {
```

在 IDA 中找到 format 函数：

```

1 v5 = (_BYTE *)_JNIEnv::GetStringUTFChars(env, input, 0);
2 v4 = _JNIEnv::GetStringLength(env, input);
3 if ( *v5 != 70 )
4     return 0;
5 if ( v5[1] != 76 )
6     return 0;
7 if ( v5[2] != 65 )
8     return 0;
9 if ( v5[3] != 71 )
10    return 0;
11 if ( v5[4] == 123 )
12    return v5[v4 - 1] == 125;
13 return 0;
14
15 }
```

可以看到它要求输入字符串的格式为 FLAG{XXXX}

接下来进入 divide 函数：

```

public static Boolean divide(String input) {
    if(input.length() != 38) {
        return Boolean.valueOf(false);
    }

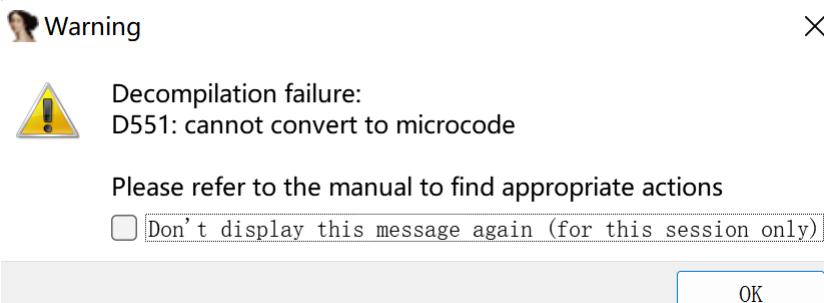
    String part1 = input.substring(5, 13);
    String part2 = input.substring(13, 21);
    String part3 = input.substring(21, 29);
    String part4 = input.substring(29, 37);
    if(!class3.w(part1).equals(class1.x(1))) {
        return Boolean.valueOf(false);
    }

    if(!class4.e(part2).equals(class1.x(2))) {
        return Boolean.valueOf(false);
    }

    return class5.e(part3).equals(class1.x(3)) ? Boolean.valueOf(class6.b(part4).equals(class1.x(4))) : Boolean.valueOf(class7.c(part4));
}

public static native String x(int arg0);
```

简单重命名后，可以看到函数要求输入字符串长度为 28，且将 flag 内容拆分为了 4 个长度为 8 的字符串，且每一个都要与 x 函数相应的返回值相等，才能返回 true，而 x 函数位于 native code 中（怎么大家都这么喜欢 native code...），再次回到 IDA



IDA 表示：转不了，那先看二进制文件：

• .text:0000D549	db 74h ; t
• .text:0000D54A	db 72h ; r
• .text:0000D54B	db 75h ; u
• .text:0000D54C	db 65h ; e

Patch 过程中发现了字符串：true

```
• .text:0000D55C      db 62h ; b |  
• .text:0000D55D      db 62h ; b  
• .text:0000D55E      db 63h ; c  
• .text:0000D55F      db 64h ; d  
• .text:0000D560      db 66h ; f  
• .text:0000D561      db 69h ; i  
• .text:0000D562      db 62h ; b  
• .text:0000D563      db 64h ; d  
• .text:0000D564      "  
    ^
```

字符串 2: bbcdfibd

```
• .text:0000D573      db 31h ; 1  
• .text:0000D574      db 36h ; 6  
• .text:0000D575      db 38h ; 8  
• .text:0000D576      db 31h ; 1  
• .text:0000D577      db 31h ; 1  
• .text:0000D578      db 36h ; 6  
• .text:0000D579      db 37h ; 7  
• .text:0000D57A      db 33h ; 3  
• .text:0000D57B      db 32h ; 2  
• .text:0000D57C      db 31h ; 1  
• .text:0000D57D      db 39h ; 9  
• .text:0000D57E      db 33h ; 3  
• .text:0000D57F      db 31h ; 1  
• .text:0000D580      db 31h ; 1  
• .text:0000D581      db 37h ; 7  
• .text:0000D582      db 37h ; 7  
• .text:0000D583      db 35h ; 5  
• .text:0000D584      db 38h ; 8  
• .text:0000D585      db 31h ; 1  
• .text:0000D586      db 39h ; 9  
• .text:0000D587      db 33h ; 3  
• .text:0000D588      db 31h ; 1  
• .text:0000D589      db 31h ; 1  
• .text:0000D58A      db 36h ; 6  
• .text:0000D58B      4h  ^
```

发现字符串 3: 168116732193117758193116

```
• .text:0000D59A      db 2Eh ; .  
• .text:0000D59B      db 7Eh ; ~  
• .text:0000D59C      db 55h ; U  
• .text:0000D59D      db 36h ; 6  
• .text:0000D59E      db 29h ; )  
• .text:0000D59F      db 7Dh ; }  
• .text:0000D5A0      db 54h ; T  
• .text:0000D5A1      db 24h ; $  
• .text:0000D5A2      db 0
```

字符串 4: .~U6})T\$

```

1 // positive sp value has been detected, the output may be wrong!
2 int __cdecl Java_com_example_crackme_111111111111111_x(_JNIEnv *a1, int a2, int a3)
3 {
4     char *v4; // [esp-4h] [ebp-2Ch]
5     char *v5; // [esp+18h] [ebp-10h]
6
7     switch ( a3 )
8     {
9         case 1:
10            JUMPOUT(0xD54E);
11        case 2:
12            v5 = v4;
13            break;
14        case 3:
15            v5 = v4;
16            break;
17        case 4:
18            v5 = v4;
19            break;
20        default:
21            return _JNIEnv::NewStringUTF(a1, v5);
22    }
23    return _JNIEnv::NewStringUTF(a1, v5);
24 }

```

全部 patch 完成后，虽然还是没得到完全正确的函数，但基本逻辑已经能推断出来了，四个字符串对应四个返回值，到这里就可以去看等号另一边的加密函数了：

先看针对 part1 的 w:

```

public static native String a(String arg0) {
}

public static String w(String str) {
    String s;
    String d = class3.a(str);
}

```

首先调用 a, 又是 native 库:

```

1 int __cdecl Java_com_example_crackme_IIIIIIlllllIIIIlll_a(_JNIEnv *a1, jobject a2, jstring input)
2 {
3     const char *v3; // eax
4     int v5; // [esp+10h] [ebp-48h]
5     int v6; // [esp+34h] [ebp-24h]
6     char v7[16]; // [esp+38h] [ebp-20h] BYREF
7     unsigned int v8; // [esp+48h] [ebp-10h]
8
9     v8 = __readgsword(0x14u);
10    v6 = _JNIEnv::GetStringUTFChars(a1, input, 0);
11    std::string::basic_string<decltype(nullptr)>(v7, v6);
12    std::string::insert(v7, 0, &aA0b1c2d3e4f5g6[21]);
13    std::string::insert(v7, 2, aChjk);
14    std::string::insert(v7, 7, &aChjk[1]);
15    std::string::insert(v7, 9, &aChjk[2]);
16    std::string::insert(v7, 10, &aChjk[3]);
17    v3 = (const char *)sub_DB40(v7);
18    v5 = _JNIEnv::NewStringUTF(a1, v3);
19    std::string::~string(v7);
20    return v5;
21 }

```

在字符串位置 0 插入 A

```
.rodata:00035288 | text "UTF-16LE", 'a0b1c2d3e4f5g6h7i8j9xA'
```

在位置 2 插入 C, 位置 7 插入 H, 位置 9 插入 J, 位置 10 插入 K

```
.rodata:000352B4 | text "UTF-16LE", 'CHJK'
```

最终返回一个长度为 20 的字符串

```

if(i % 3 == 0) {
    str1 = class5.a[i / 3].replace(c, 'X');
}
else {
    str1 = i % 3 == 1 ? class8.a[(i - 1) / 3].replace(c, 'X') : class7.getFlag()[(i - 2) / 3].replace(c, 'X');
}

if(!class7.d(str1).equals(class3.b[i])) {
    return "false";
}

```

接下来的代码，对字符串中角标为 3 倍数的字符，将其在 class5，字符串数组 a 对应项中的出现替换为‘X’，并将替换后的值赋给 str1

```

class5.a = new String[]{"I", "lean", "sunsets", "of", "suburbs"};
class5.count = 1;

```

对于角标模三余 1 的字符，将其在 class8，字符串数组 a 对应项中的出现替换为‘X’，并将替换后的值赋给 str1

```

class8.a = new String[]{"offer", "streets", "the", "the"};

```

对于角标模三余 2 的字符，将其在 class7，getflag 函数返回项对应项中的出现替换为‘X’，并将替换后的值赋给 str1

```

public static native String[] getFlag() {
}

```

Getflag 为 native 函数：

在 IDA 中可以找到其返回取值的字符串组：

```

.rodata:000352BC aYou          db 'you',0
.rodata:000352C0 aDesperate    db 'desperate',0
.rodata:000352CA aMoon         db 'moon',0
.rodata:000352CF aJagged       db 'jagged',0

```

到这里，替换方法已经出来了，最后是验证部分：

```

if(!class7.d(str1).equals(class3.b[i])) {
    return "false";
}

```

D 方法最终调用 a 函数：

```

public static String a(byte[] original) {
    byte[] bytes = class7.c(original);
    StringBuffer stringBuffer = new StringBuffer("");
    int i;
    for(i = 0; i < bytes.length; ++i) {
        short a = (short)(bytes[i] & 15);
        short b = (short)((bytes[i] & 0xF0) >>> 4);
        if(b < 10) {
            stringBuffer.append(b);
        } else {
            stringBuffer.append(class7.MMP.get(Short.valueOf(b)));
        }
        if(a < 10) {
            stringBuffer.append(a);
        } else {
            stringBuffer.append(class7.MMP.get(Short.valueOf(a)));
        }
    }
    return stringBuffer.toString();
}

```

而其中输入的字符串经过 c 的处理: c 通过调用位于 class7 中的数组 b 来生成 keybyte 字符组, 并最终与输入字符异或, 得到返回值

```

class7.b = new int[]{23, 22, 24, 4, 51, 26, 37, 27, 24, 6, 26, 38, 29, 35, 18, 21, 14, 3, 12, 4, 41, 39, 18, 44, 54, 21}

public static byte[] c(byte[] original) {
    byte[] keyByte = new byte[class7.b.length];
    int i;
    for(i = 0; true; ++i) {
        int[] v2 = class7.b;
        if(i >= v2.length) {
            break;
        }
        keyByte[i] = (byte)"dfsad@%$@TDGDF%$#%@#%WFRGFDHJKcvxznmfdsgdfgs2432534fgdf46t".charAt(v2[i]);
    }

    int k = 0;
    byte[] encryptByte = new byte[original.length];
    int i = 0;
    while(i < original.length) {
        encryptByte[i] = (byte)(keyByte[k % keyByte.length] ^ original[i]);
        ++i;
        ++k;
    }
    return encryptByte;
}

```

到这里, 函数 d 的算法就清楚了, 来看另一边 class3 的 b 字符串组:

```
class3.b = new String[]{"0F", "1E31340146", "3F3827", "2A320A0A", "35230A01513215", "22322114511E073037", "350F3C175"}
```

得到构造思路: 根据 class3.b 值→解出 class7.d 的返回值, 根据返回值→逆推 class7.d 的输入值, 得到 str1→通过 str1, 得到各个字符串中最终被替换的字符→得到 str2 的字符组成, 也即 class3.a 的返回值→得到 class3.a 的输入值, 也即函数 w 的 input 字符串→得到答案

具体细节由于这里用到的加密算法都比较简单, 主要是繁琐。最终得到字符串替换后值:

I xffer you lexn steets despeXate sXnsets the Xoon of the Xagged suXurXs

从而得出字符串中被替换的字符, 得到 part1 答案: oarrumjb

下面进入加密 part2 的函数 e:

```

public static String e(String i) {
    return class4.q(i, class4.Map2String(class4.b()));
}

public static void f() {
    class4.b();
}

public static native String q(String arg0, String[] arg1) {
}

```

又是 native 库? ? ? ?

函数 q 的传入值由 map2string 得到

```

public static String[] Map2String(Map arg6) {
    if(arg6 != null && arg6.size() != 0) {
        String[] mParams = new String[arg6.size() * 2];
        int i = 0;
        for(Object v3: arg6.keySet()) {
            String key = (String)v3;
            mParams[i * 2] = key;
            mParams[i * 2 + 1] = (String)arg6.get(key);
            ++i;
        }
        return mParams;
    }
    return new String[0];
}

```

Map2string 的传入参数为 b:

```

public static Map b() {
    HashMap map = new HashMap();
    map.put("4", "P");
    map.put("8", "o");
    map.put("6", "R");
    map.put("7", "E");
    map.put("5", "I");
    map.put("9", "l");
    map.put("2", "o");
    map.put("0", "v");
    map.put("1", "e");
    map.put("3", "U");
    return map;
}

```

其思路是将 map b 转化为一个按照{键名, 键值}排布的字符串数组, 其实就是扁平化处理
而 q 函数: 则 jia 能够数字替换为字母, 字母替换为 x

```

for ( j = 0; j < v25; ++j )
{
    sub_E590(v29, 1, *(_BYTE *)(&v26 + j));
    sub_E640(v28);
    sub_E6E0(v27);
    if ( (sub_E5F0(v28, v27) & 1) != 0 )
    {
        v9 = std::map<std::string, std::string>::operator[](v41, v29);
        sub_E860(s, v9);
    }
    else
    {
        sub_E8A0(s, "null");
        for ( k = 0; (unsigned int)(2 * k) < sub_E8E0(v43); ++k )
        {
            v4 = sub_EA40(v43, 2 * k);
            if ( (sub_E910(v4, v29) & 1) != 0 )
            {
                v5 = sub_EA40(v43, 2 * k + 1);
                sub_E860(s, v5);
            }
        }
        std::string::~string(v29);
    }
}
v6 = (const char *)sub_D3B0((int)s);
v8 = _JNIEnv::NewStringUTF((JNIEnv *)a1, v6);
std::string::~string(s);
sub_EA70(v41);
sub_EAA0(v43);
return v8;

```

而由于 x 函数中，已经得到了 q 的运算结果应为 bbcdfibd，将其对应 q 的逻辑进行逆变换，可以得到结果：11235813

下一个进入到 part3 的加密函数，其调用函数 c：而 c 通过函数 d 对字符串中的每一个字符进行相应的加密

```

public static String c(String ssr) {
    StringBuilder s = new StringBuilder();
    int i;
    for(i = 0; i < ssr.length(); ++i) {
        s.append(class5.d(ssr.charAt(i)));
    }

    return s.toString();
}

```

而函数 d，不出所料的，又是 native code：

```

public static native int d(char arg0) {
}

```

在 IDA 中还原后得到：

```

v9 = 1;
for ( i = 10; a3 / i; i *= 10 )
    ++v9;
v7 = 0;
for ( j = 0; j < v9; ++j )
{
    v5 = a3 / (int)sub_D5C0(10, j) % 10;
    v7 += (int)sub_D5C0(27, j) * v5;
}
return v7;

```

其逻辑为对输入的字符进行简单的指数运算，还原较为简单
而最终 c 的返回值同样也已经得到：168116732193117758193116
最终可以得到 part3 答案：B0gJ1pJ0
最后是 part4 的加密函数：

```

public static String b(String str) {
    return class6.m(class6.c(str), class6.key);
}

```

其调用了，再一次的，native code m：

```

public static native String m(String arg0, char[] arg1) {
}

```

而输入的参数，一个来自函数 c：

```

public static String c(String str) {
    int v1 = 0;
    String s1 = str.substring(0, str.length() / 2);
    String s2 = str.substring(str.length() / 2, str.length());
    int[] k1 = class6.d(s1);
    int[] k2 = class6.a(s1, s2);
    StringBuilder result = new StringBuilder();
    int v7;
    for(v7 = 0; v7 < k1.Length; ++v7) {
        result.append(((char)k1[v7]));
    }

    while(v1 < k2.Length) {
        result.append(((char)k2[v1]));
        ++v1;
    }

    return result.toString();
}

```

一个来自 key：（套娃是吧 😅）

```

public static final char[] key;
public static String keyword;
public static String p;
public static final String s;
public static String translateKeyword;
private static final char[] wJ;

static {
    String v0 = IIIIIIIlllIIIllllll.c();
    class6.s = v0;
    class6.key = class6.a(v0);
}

```

Key 由 a 生成

```

public static char[] a(String s1) {
    StringBuilder v = new StringBuilder();
    int i;
    for(i = 0; i < s1.length(); ++i) {
        v.append(s1.charAt(i));
    }

    v.append("\\\"");
    v.append("-");
    return v.toString().toCharArray();
}

```

A 的参数来自 c, 是 native code:

```

public static native String c() {
}

```

通过 patch 在 IDA 中找到 c 的值

.text:0000CE60	db 38h ; 8
.text:0000CE61	db 4Eh ; N
.text:0000CE62	db 27h ; '
.text:0000CE63	db 50h ; P
.text:0000CE64	db 5Bh ; [
.text:0000CE65	db 52h ; R
.text:0000CE66	db 43h ; C
.text:0000CE67	db 29h ;)
.text:0000CE68	db 58h ; X
.text:0000CE69	db 42h ; B
.text:0000CE6A	db 4Dh ; M
.text:0000CE6B	db 5Ah ; Z
.text:0000CE6C	db 64h ; d
.text:0000CE6D	db 7Ah ; z

到这里只剩最后一块拼图了：函数 m

```

v12 = __readgsdword(0x14u);
v10 = _JNIEnv::GetStringUTFChars(a1, a3, 0);
v9 = _JNIEnv::GetStringLength(a1, a3);
v8 = _JNIEnv::GetCharArrayElements(a1, a4, 0);
sub_D120(v11);
for ( i = 0; i < v9; ++i )
{
    if ( *(char *)(&v10 + i) >= 33 )
        std::string::push_back(v11, *(_BYTE *)(&v10 + i));
    else
        std::string::push_back(v11, *(_BYTE *)(&v8 + 2 * i));
}
(JNIEnv::ReleaseCharArrayElements(a1, a4, v8, 0);
v4 = (const char *)sub_D3B0((int)v11);
v6 = _JNIEnv::NewStringUTF((JNIEnv *)a1, v4);
std::string::~string(v11);
return v6;

```

M 的输出值为之前已经得到的.<~U6}T\$, 于是再一次层层反推, 最终得到 part4: cM3fJ0gB 四个 part 组合, 得到最终的 flag:

FLAG{oarrumjb11235813B0gJ1pJ0cM3fJ0gB}

