工具

Jeb，ida

知识点

1. .So
2. Base64 替换密码表

流程分析

1. MainActivity

```
protected void onCreate(Bundle arg3) {
    super.onCreate(arg3);
    this.setContentView(2130968603);
    this.findViewById(2131427446).setOnClickListener(new View$OnClickListener(((Context)this)) {
        public void onClick(View arg4) {
            if(MainActivity.a(this.b, this.a.findViewById(2131427445).getText().toString())) {
                Toast.makeText(this.a, "You are right!", 1).show();
            }
            else {
                Toast.makeText(this.a, "You are wrong! Bye~", 1).show();
            }
        }
    });
}
}
```

用 MainActivity 类中的 a 函数去验证

```
static boolean a(MainActivity arg1, String arg2) {
    return arg1.a(arg2);
}

private boolean a(String arg3) {
    boolean v0_1;
    try {
        v0_1 = this.ncheck(new a().a(arg3.getBytes()));
    }
    catch(Exception v0) {
        v0_1 = false;
    }

    return v0_1;
}
```

a类中的a函数

这里的 ncheck（）函数是一个 native 函数，等下去找 so 库中对应的函数

2. a 类是一个 base64 的替换表加密。

几个明显的特征如下：

① 定义了一个密码表

```
static {
    a.a = new char[]{'i', '5', 'j', 'L', 'W', '7', 'S', '0', 'G', 'X', '6', 'u', 'f', '1', 'c', 'v', '3', 'n', 'y', '4', 'q', '8', 'e', 's', '2', 'Q',
}
```

② 每次循环的长度为 3，定义了长度为 4 的数组

```
int v0;
for(v0 = 0; v0 <= arg10.length - 1; v0 += 3) {
    byte[] v5 = new byte[4];
    int v3 = 0;
    byte v2 = 0;
```

③ 长度不够补充“=”

```
    for(v2_1 = 0; v2_1 <= v8; ++v2_1) {
        if(v5[v2_1] <= 63) {
            v4.append(a.a[v5[v2_1]]);
        }
        else {
            v4.append('=');
        }
```

3. 将 apk 解压拿到 so 库函数，将 libnative.so 用 ida 打开

```
6  const char *v6; // r6
7  int v7; // r0
8  char *v8; // r2
9  char v9; // r1
10 int v10; // r0
11 bool v11; // nf
12 unsigned __int8 v12; // vf
13 int v13; // r1
14 signed int result; // r0
15 char v15; // [sp-35h] [bp-35h]
16 int v16; // [sp-18h] [bp-18h]
17
18 v16 = v3;
19 v4 = a1;
20 v5 = a3;
21 v6 = (const char *)(*(int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)a1 + 676))(a1, a3, 0);
22 if ( strlen_0(v6) == 32 )
23 {
24   v7 = 0;
25   do
26   {
27     v8 = &v15 + v7;
28     *(&v15 + v7) = v6[v7 + 16];
29     v9 = v6[v7++];
30     v8[16] = v9;
31   }
32   while ( v7 != 16 );
33   (*(void (__fastcall **)(int, int, const char *))(*(_DWORD *)v4 + 680))(v4, v5, v6);
34   v10 = 0;
35   do
36   {
37     v12 = __OFSUB__(v10, 30);
38     v11 = v10 - 30 < 0;
39     HIBYTE(v16) = *(&v15 + v10);
40     *(&v15 + v10) = *(&v15 + v10 + 1);
41     *(&v15 + v10 + 1) = HIBYTE(v16);
42     v10 += 2;
43   }
44   while ( v11 ^ v12 );
45   v13 = memcmp_0(&v15, "MbT3sQgX039i3g==AQOoMQFPskB1Bsc7", 0x20u);
46   result = 0;
47   if ( !v13 )
48     result = 1;
49 }
50 else
51 {
52   (*(void (__fastcall **)(int, int, const char *))(*(_DWORD *)v4 + 680))(v4, v5, v6);
53   result = 0;
54 }
55 return result;
56}
```

前16位与后16位互换位置

每两位互换一下位

cmp

解题思路

1. 从字符串首位开始每两位交换一次位置
2. 前 16 位和后 16 位交换位置
3. 变异的 base64 解密

脚本

```python
import base64

# 每两位交换一次位置
str1 = list("MbT3sQgX039i3g==AQOoMQFPskB1Bsc7")
```

```
str_twoChange = ""
for i in range(0,len(str1),2):
    str_twoChange += str1[i+1] + str1[i]
# 前16位与后16位交换位置
str_F16_B16_Change = ""
str2 = list(str_twoChange)
str_F16_B16_Change = ''.join(str2[i] for i in range(16,32))+''
.join(str2[i] for i in range(0,16))
# 变异base64解密
base_new = ['i', '5', 'j', 'L', 'W', '7', 'S', '0', 'G', 'X',
 '6', 'u', 'f', '1', 'c', 'v', '3', 'n', 'y', '4', 'q', '8',
 'e', 's', '2', 'Q', '+', 'b', 'd', 'k', 'Y', 'g', 'K', 'O',
 'I', 'T', '/', 't', 'A', 'x', 'U', 'r', 'F', 'l', 'V', 'P',
 'z', 'h', 'm', 'o', 'w', '9', 'B', 'H', 'C', 'M', 'D', 'p',
  'E', 'a', 'J', 'R', 'Z', 'N']
base_new_str = ''.join(i for i in base_new)
base_original_str = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmno
pqrstuvwxyz0123456789+/'
flag = base64.b64decode(str_F16_B16_Change.translate(str.maket
rans(base_new_str,base_original_str)))
print(flag)
```

Base64 编码的补充介绍

base64 编码的作用和诞生

为什么会有 Base64 编码呢？因为有些网络传送渠道并不支持所有的字节，例如传统的邮件只支持可见字符的传送，像 ASCII 码的控制字符就 不能通过邮件传送。这样用途就受到了很大的限制，比如图片二进制流的每个字节不可能全部是可见字符，所以就传送不了。最好的方法就是在不改变传统协议的情 况下，做一种扩展方案来支持二进制文件的传送。把不可打印的字符也能用可打印字符来表示，问题就解决了。Base64 编码应运而生，Base64 就是一种 基于 64 个可打印字符来表示二进制数据的表示方法。

Base64 编码原理

看一下 Base64 的索引表，字符选用了"A-Z、a-z、0-9、+、/" 64 个可打印字符。数值代表字符的索引，这个是标准 Base64 协议规定的，不能更改。64 个字符用 6 个 bit 位就可以全部表示，一个字节有 8 个 bit 位，剩下两个 bit 就浪费掉了，这样就不得不牺牲一部分空间了。这里需要弄明白的就是一个 Base64 字符是 8 个 bit，但是有效部分只有右边的 6 个 bit，左边两个永远是 0。

数值	字符	数值	字符	数值	字符	数值	字符
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

那么怎么用 6 个有效 bit 来表示传统字符的 8 个 bit 呢？8 和 6 的最小公倍数 是 24，也就是说 3 个传统字节可以由 4 个 Base64 字符来表示，保证有效位数是一样的，这样就多了 1/3 的字节数来弥补 Base64 只有 6 个有效 bit 的不足。你也可以说用两个 Base64 字符也能表示一个传统字符，但是采用最小公倍数的方案其实是最减少浪费的。结合下边的图比较容易理解。Man 是三个 字符，一共 24 个有效 bit，只好用 4 个 Base64 字符来凑齐 24 个有效位。红框表示的是对应的 Base64，6 个有效位转化成相应的索引值再对应 Base64 字符表，查出"Man"对应的 Base64 字符是"TWFU"。说到这里有个原则不知道你发现了没有，要转换成 Base64 的最小单位就是三个字节，对一个字符串来说每次都是三个字节三个字节的转换，对应的是 Base64 的四个字节。这个搞清楚了其实就差不多了。

文本	M								a								n							
ASCII编码	77								97								110							
二进制位	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
索引	19								22								5							
Base64编码	T								W								F							

举一个完整的例子，例如："Iamashabi."

可以分成三个字节一组的形式，下方就是对应的Base64编码，四个字符一组。

Iam ash abi .

SWFt YXNo YWJp Lg==

但是转换到最后你发现不够三个字节了怎么办呢？愿望终于实现了，我们可以用两 个 Base64 来表示一个字符或用三个 Base64 表示两个字符，像下图的 A 对应的第二个 Base64 的二进制位只有两个，把后边的四个补 0 就是了。所以 A 对应的 Base64 字符就是 QQ。上边已经说过了，原则是 Base64 字符的最小单位是四个字符一组，那这才两个字 符，后边补两个"="吧。其实不用"="也不耽误解码，之所以用"="，可能是考虑到多段编码后的 Base64 字符串拼起来也不会引起混淆。由此可见 Base64 字符串只可能最后出现一个或两个"="，中间是不可能出现"="的。下图中字符"BC"的编码过程也是一样的。

文本（1 Byte）	A																							
二进制位	0	1	0	0	0	0	0	1																
二进制位（补0）	0	1	0	0	0	0	0	1	0	0	0	0												
Base64编码	Q						Q						=				=							
文本（2 Byte）	B								C															
二进制位	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1								
二进制位（补0）	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1	0	0						
Base64编码	Q						k						M				=							