

xxxx签名算法逆向&&python脚本实现

前言

有一段时间没看安卓了，找几个软件练练手。

这是一个考驾照用的 app.

官方网址: http://www.*****baodian.com/

本文就分析一下在重置密码时对数据包进行签名来防篡改的方案。

正文

burp抓取https数据

首先导入 burp 证书到手机，装上 xposed，然后安装

<https://github.com/WooyunDota/DroidSSLUnpinning>

这个插件应该就可以抓到 https 数据了。

然后触发找回密码逻辑，抓包

Type	Name	Value
URL	_manufacturer	samsung
URL	_systemVersion	5.1.1
URL	_device	SM-G9350
URL	_imei	862514326681779
URL	_productCategory	jiakaobaodian
URL	_operator	T
URL	_androidId	0086049272635872
URL	_mac	02:00:00:00:00:00
URL	_appUser	ffd830178df6460e8b401c1421b1e148
URL	_pkgName	com.handsgo.jiakao.android
URL	_screenDpi	1.2
URL	_screenWidth	720
URL	_screenHeight	1280
URL	_network	wifi
URL	_launch	4
URL	_firstTime	2018-03-09 09:28:53
URL	_apiLevel	22
URL	_userCity	320100
URL	_p	464D555059511A4751565E5F475A
URL	_ipCity	320100
URL	_j	1.0
URL	schoolName	æœ®æœ¥èœ©éœ%æ ¡
URL	schoolCode	-1
URL	_webviewVersion	4.7
URL	_mcProtocol	4.0
URL	_r	66ae8eb0a00b4d6d9451eed3ee169a1
URL	sign	fa0424349a82e84a3789929a93831c7701
Body	phoneNumber	13333333333
Body	captchaId	e5410186e8afa433e4a4e8019901ff40
Body	captchaCode	5659

这里有一个 sign 的参数，可以推测这个是用来保存签名的参数。

如果正常发送的数据包得到的响应类似于下面（这里是验证码输入错误的情况）

Target: <https://au>

Request

- [Raw](#)
- [Params](#)
- [Headers](#)
- [Hex](#)

```
POST /api/open/v2/forgot-password/check.htm?_platform=android&_srv=t&_appName=jiaakao
baodian&_product=%E9%A9%BE%E8%80%83%E5%AE%9D%E5%85%B8&_vendor=xiaomi&_renyuan=X
YX&_version=6.9.8&_system=LMY48Z&_manufacturer=samsung&_systemVersion=5.1.&_de
vice=SM-G9350&_imei=862514326681779&_productCategory=jiaakaoBaodian&_operator=T&
_androidId=0086049272635873&_mac=02%3A00%3A00%3A00%3A00&_appUser=ffd830178
df6460e8b401c1421b1e148e_pkName=com.handsgo.jiaakao.android&_screenDpi=1.28_
screenWidth=720&_screenHeight=1280&_network=wifi&_launch=4&_firstTime=2018-03-09%2
009%3A28%3A53&_apiLevel=22&_userCity=320100&_p=464D555059511A4751565E5F475A&_ip
City=320100&_j=1.0&schoolName=%E6%9C%AA%E6%8A%A5%E8%80%83%E9%A9%BE%E6%AO%A1&sch
oolCode=-1&_webViewVersion=4.7&_mcProtocol=4.0&_r=78ae8eb0a00b4d6d9451eeecd3ee16
9a&sign=8b5c95d8e839cce7588db0c3ee315c0501 HTTP/1.1
User-Agent: Mozilla/5.0 (Linux; Android 5.1.1; SM-G9350 Build/LMY48Z)
AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/39.0.0.0
Safari/537.36
Accept-Encoding: gzip, deflate
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 83
Host: auth.mucang.cn
Connection: close
phoneNumber=13333333333&captchaId=e5410186e8afa433e4a4e8019901ff40&captchaCode=
5659
```

Response

- [Raw](#)
- [Headers](#)
- [Hex](#)
- [JSON Beautifier](#)

```
{
  "data": null,
  "errorCode": 20011,
  "message": "验证码错误，请重新输入",
  "success": false
}
```

如果重放数据包

Response

- [Raw](#)
- [Headers](#)
- [Hex](#)
- [JSON Beautifier](#)

```
{
  "data": null,
  "errorCode": -1002,
  "message": "非法请求，重复的URL。",
  "success": false
}
```

经过测试发现 `_r` 参数用于标识数据包的唯一性，如果修改其他的字段会提示 重复的 URL，但是如果修改了 `_r`，则会提示 URL 签名错误

request

- [Raw](#)
- [Params](#)
- [Headers](#)
- [Hex](#)

DST request to /api/open/v2/forgot-password/check.htm

Type	Name	Value
RL	_screenWidth	720
RL	_screenHeight	1280
RL	_network	wifi
RL	_launch	4
RL	_firstTime	2018-03-09 09:28:53
RL	_apiLevel	22
RL	_userCity	320100
RL	_p	464D555059511A4751565E5F475A1
RL	_ipCity	320100
RL	_j	1.0
RL	schoolName	æ²»æ²»¥è²»é©%æ¡
RL	schoolCode	-12
RL	_webViewVersion	4.7
RL	_mcProtocol	4.0
RL	_r	66ae8eb0a00b4d6d9451eeecd3ee169a21
RL	sign	eb47700ab5b621fb9014100607a8346301
ody	phoneNumber	13333333333
ody	captchaId	e5410186e8afa433e4a4e8019901ff40

response

- [Raw](#)
- [Headers](#)
- [Hex](#)
- [JSON Beautifier](#)

```
{
  "data": null,
  "errorCode": 506,
  "message": "URL签名错误",
  "success": false
}
```

于是大概可以推测服务端校验请求的流程

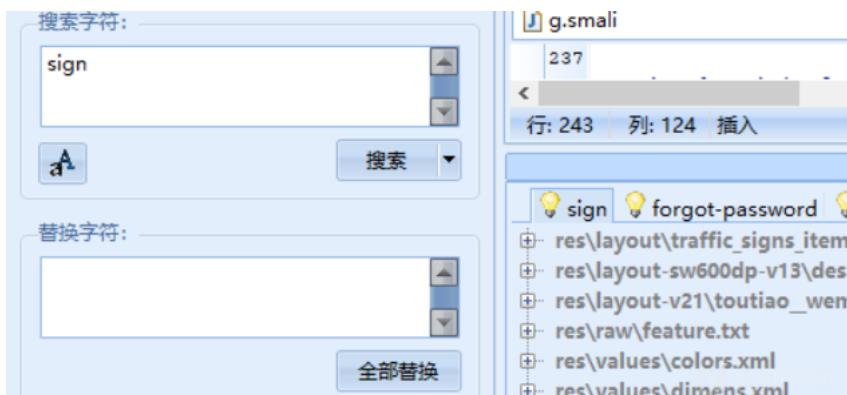
- 拿到 `_r` 判断是否是重复的请求
- 计算请求的 签名，与签名字段进行校验
- 如果前面两步均通过，服务端则认为数据包没有被篡改，于是开始校验 验证码。

定位关键代码

我列举一下我用了的方法

搜参数名

程序中可能会用到参数的名称来设置参数的值，因为我的目标是分析签名算法，`sign` 这个参数名这么的明显，就搜他了。



发现太多的文件里面有这个关键字, 不可用

搜 url 路径

观察抓到的数据包, 发现它是向 /api/open/v2/forgot-password/check.htm 发起了 POST 请求。

这个貌似靠谱, 可以追踪到进行请求的代码, 于是从第一个结果开始分析, 一路往下, 追踪, 大概搞清楚了发送请求的流程, 以及一些参数的得到方式, 但是貌似是看花眼了, 而且程序的混淆强度比较大, 没用找到 _r 和 sign 设置位置。于是开始了其他的定位尝试

监控系统层的加密类和方法

进行签名常用的方案就是对待签名数据, 算 hash 或者用一些加密算法。于是想着是不是可以通过监控 java 层常用的加密 api 来定位算法。

<https://github.com/Chenyuxin/CryptoFucker>

找到了这个插件, 通过 hook 的方式来监控, 同时会在 /sdcard/ydsec/包名.txt 留下日志。

```

1
2 MD5 data:
3
4 0x00000000 2F 61 70 69 2F 6F 70 65 |6E 2F 76 34 2F 63 6F 6E /api/open/v4/con
5 0x00000010 66 69 67 2F 67 65 74 2E 68 74 6D 3F 5F 75 73 65 fig/get.htm?_use
6 0x00000020 72 43 69 74 79 3D 33 32 30 31 30 30 26 5F 69 70 rCity=320100&_ip
7 0x00000030 43 69 74 79 3D 33 32 30 31 30 30 City=320100
8
9
10 MD5 result:
11
12 0x00000000 31 AB CB 7E A6 3E BE A1 7F 56 E1 4B 9E 33 0C A9
13
14
15 SHA data:
16

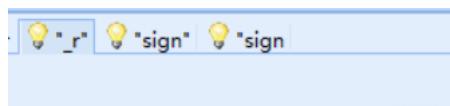
```

试了试, 发现也没有找到相关数据, 通过这样的方式被加密, 于是推测应该是自己实现了相关的加密算法。

再次搜参数名

忽然想起，参数名如果在程序中被使用的话应该是直接存在于程序中的，然后 small 代码引用字符串会加上“来包裹字符串

开始了另一种搜索尝试



发现搜 "sign" 和 "sign" 得到的结果不多，但是貌似都不太相关，倒是 "_r"，就两个结果。

想起既然 _r 用于标识请求的唯一性，那么签名肯定是要用到它的，而且 _r 应该每次都是随机生成的，那么用到它的位置，应该离生成 sign 的位置不远了。

进入第一个结果的代码（程序太大了，jetty打不开之前的我分析时的数据，只能重新看没有命名函数名的了）

```
public static String a(String arg6, String arg7, String arg8, Map arg9) {
    StringBuilder v1 = new StringBuilder(arg7);
    HashMap v2 = new HashMap();
    ((Map)v2).put("_r", UUID.randomUUID().toString().replace("-", ""));
    if(!d.v(arg9)) {
        Iterator v3 = arg9.entrySet().iterator();
        while(v3.hasNext()) {
            Object v0 = v3.next();
            ((Map)v2).put(((Map$Entry)v0).getKey(), ((Map$Entry)v0).getValue());
        }
    }
    ax.a.a(v1, "4.7", ((Map)v2), true, null);
    return arg6 + a.N(v1.toString(), arg8);
}
```

先知社区

可以知道 _r 通过

UUID.randomUUID().toString().replace("-", "")

生成随机字符串，来保证数据包不被重放

```
private static String N(String arg1, String arg2) {
    if(!ad.isEmpty(arg2)) {
        arg1 = aa.ao(arg1, arg2);
    }
    return arg1;
}

public static String a(String arg6, String arg7, String arg8, Map arg9) {
    StringBuilder v1 = new StringBuilder(arg7);
    HashMap v2 = new HashMap();
    ((Map)v2).put("_r", UUID.randomUUID().toString().replace("-", ""));
    if(!d.v(arg9)) {
        Iterator v3 = arg9.entrySet().iterator();
        while(v3.hasNext()) {
            Object v0 = v3.next();
            ((Map)v2).put(((Map$Entry)v0).getKey(), ((Map$Entry)v0).getValue());
        }
    }
    ax.a.a(v1, "4.7", ((Map)v2), true, null);
    return arg6 + a.N(v1.toString(), arg8);
}
```

先知社区

生成 _r 后，会调用 a.N，其中会调用 aa.ao，跟进去看看

```

public static String ao(String url, String arg9) {
    int v0_2;
    int v4;
    String v2;
    String v0_1;
    URI v3;
    int v7 = 63;
    int v6 = 38;
    int v1 = -1;
    try {
        v3 = new URI(url); ①
        v0_1 = v3.getRawPath();
        if(v3.getRawQuery() != null) {
            v2 = v0_1 + "?" + v3.getRawQuery().replace("+", "%20");
        }
    } else {
        goto label_99;
    }

    goto label_20;
}
catch(Exception v0) {
    goto label_91;
}

label_99:
v2 = v0_1;

```

根据 URL(url) 可以确定，第一个参数是一个 url，接着对 url 上的参数进行了处理。

然后定位 sign 参数的起始位置

```

label_99:
v2 = v0_1;
try {
label_20:
sign_start = v2.indexOf("sign="); // 定位 sign 参数的起始位置
if(sign_start != v1) {
    sign_end = sign_start + 5;
    while(true) {
        if(sign_end >= v2.length()) {
            goto label_97;
        }
        else if(v2.charAt(sign_end) != v6) {
            ++sign_end;
            continue;
        }

        goto label_28;
    }
}
else {
    goto label_97;
}

goto label_28;
}
catch(Exception v0) {
    goto label_91;
}

```

接着删除了 sign 参数

```

label_97:
sign_end = v1;
try {
label_28:
StringBuilder v5 = new StringBuilder(v2);
if(sign_start != v1) {
    if(sign_end != v1) {
        v5.delete(sign_start, sign_end); ②
    }
    else {
        v5.delete(sign_start, v2.length()); ③
    }
}

if(v5.charAt(sign_start - 1) != v6 && v5.charAt(sign_start - 1) != v7) {
    goto label_41;
}

v5.deleteCharAt(sign_start - 1);
}

```

然后重新生成 sign 参数

```

label_97:
    sign_end = v1;
    try {
        label_28:
        String Builder v5 = new String Builder(v2);
        if(sign_start != v1) {
            if(sign_end != v1) {
                v5.delete(sign_start, sign_end);
            }
        } else {
            v5.delete(sign_start, v2.length());
        }
        if(v5.charAt(sign_start - 1) != v6 && v5.charAt(sign_start - 1) != v7) {
            goto label_41;
        }
        v5.deleteCharAt(sign_start - 1);
    }
}

```



调用了 Riddle.s 生成了 sign 参数，传入的参数是处理后的 url_path 和签名用的 key。

Riddle.s 是一个 native 方法，在 libtnpn.so 里面，去 lib\armeabi 目录下找就是了。

```

public class Riddle {
    static {
        System.loadLibrary("tnpn");
    }

    public Riddle() {
        super();
    }

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

    public static native byte[] e(byte[] arg0) {
    }

    public static native boolean isAdExpired(Object arg0) {
    }

    public static native void nativeClear(String arg0) {
    }

    public static native void nativeFasterClear(int arg0, Object arg1) {
    }

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

```



添加 脱压缩 测试 常看 删减 常找 回复 信息 | 扫描病毒 汉化 自脱压缩式

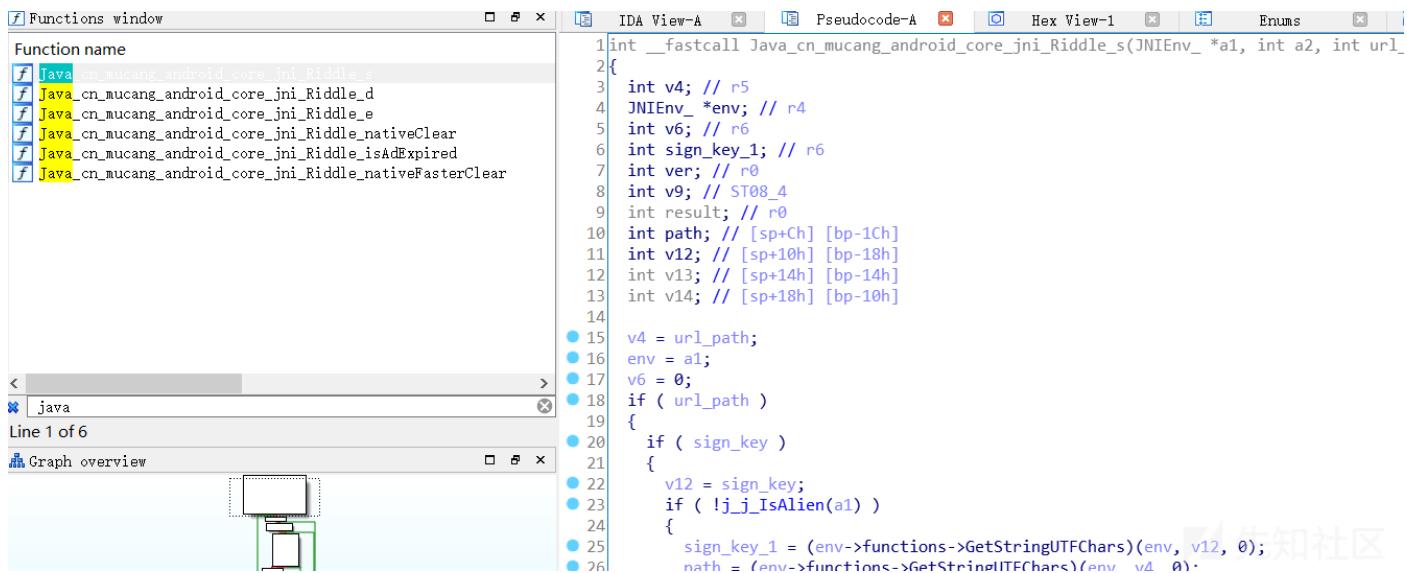
com.handsgo.jiakao.android.apk\lib\armeabi - ZIP 压缩文件, 解包大小为 119,936,154 字节

名称
libBaiduMapSDK_search_v4_1_1.so
libBaiduMapSDK_util_v4_1_1.so
libDiffDecoder.so
liblocSDK7.so
libpl_droidsonroids_gif.so
libpl_droidsonroids_gif_surface.so
libRongIMLib.so
libtnpn.so
libuninstallwatcher.so
libweibosdkcore.so

< |



拖进 ida 发现 so 倒是挺友好的，没有混淆。



进入 Java_cn_mucang_android_core_jni_Riddle_s，这就是 Riddle.s 方法在 so 中的命名，jni 函数的第一个参数类型为 `JNIEnv_*`，首先需要导入 `jni.h`，然后设置一下类型便于分析。

```

` v12 = sign_key;
if ( !j_j_IsAlien(a1) )
{
    sign_key_1 = (env->functions->GetStringUTFChars)(env, v12, 0);
    path = (env->functions->GetStringUTFChars)(env, v4, 0);
    ver = j_j_GetSigningVersion(sign_key_1); // 如果 sign_key 以 *#06# 开头，则返回1
                                                //
    if ( ver == 1 )
    {
        ver = j_j_SignUrl1(path, sign_key_1 + 5, &v13); |
    }
    else if ( !ver )
    {
        ver = j_j_SignUrl0(path, sign_key_1, &v13);
    }
}

```

首先就是把传入的参数变成 c 语言中的字符串类型，然后根据 key 的格式判断，使用哪种签名方案，经过调试重置密码用的 key 为 `*#06#i31rRYudcZZ2fIx9f16VqJV8` 所以会调用

`ver = j_j_SignUrl1(path, sign_key_1 + 5, &v13);`

跟进到 `SignUrl1`

```

v18 = buf;
key_ = key;
path_1 = path;
key_len = j_strlen(key);
decode_key = j_j_base64decode(key_, key_len); // 对 key 进行 base64 解码
decoded_key = decode_key;
raw_len = j_strlen(decoded_key);
if ( raw_len >= 1 )
{
    ptr = decoded_key;                                // 对解码后的 key，进行异或解密
    do
    {
        *ptr = (*ptr - 42) ^ 0x2A;
        --raw_len;
        ++ptr;
    }
    while ( raw_len );
}
v19 = decoded_key;
j_j_SignUrl0(path_1, decoded_key, &md5);
v20 = md5;
j_j_PrintCharArray("signed0");
```

```

首先对 key 进行解密，然后进入 `j_j_SignUrl0`

```

signed int __fastcall SignUrl0(int path, int key, int *a3)
{
 int *v3; // ST00_4
 char *v4; // r6
 char *v5; // r4
 int path_len; // r5
 int key_len; // r0
 char *v8; // r5
 void *v9; // r0
 int v10; // r4
 int v11; // r0

 v3 = a3;
 v4 = key;
 v5 = path;
 path_len = j_strlen(path);
 key_len = j_strlen(v4);
 v8 = j_malloc(path_len + key_len + 1);
 j strcpy(v8, v5); // 拼接 path + key
 j strcat(v8, v4);
 v9 = j_malloc(16);
 v10 = v9;
 *v3 = v9;
 v11 = j_strlen(v8);
 j_j_MD5(v8, v11, v10);
 j_free(v8);
 return 16;
}

```



流程很清晰，就是拼接 path 和解密后的 key，然后计算 md5，存在 a3 地址，回到 SignUrl1

```

v20 = md5;
j_j_PrintCharArray("signed0");
key_sum = *key_;
v11 = 0;
if (*key_)
{
 key_len_1 = j_strlen(key_);
 if (key_len_1 >= 2)
 {
 i = 1;
 do
 key_sum += key_[i++];
 while (i < key_len_1);
 }
 calc_sum(key_sum, 19);
 v11 = v14;
}

```



接着又对传入的 没有被解密的key 来了一个求和，然后进入

`calc_sum(key_sum, 19);`

这个函数有点复杂，我是直接抠了出来，用 python 重写了。先继续看后面。

```

7 }
3 calc_sum(key_sum, 19);
9 v11 = v14;
0 } _____
1 i_1 = 17;
2 if (!v11)
3 i_1 = 16;
4 sign_buf = j_malloc(i_1 + 1);
5 *v18 = sign_buf;
6 j_aeabi_memcpy(sign_buf, v20, 16);
7 if (v11 > 0)
8 sign_buf[16] = v11;
9 sign_buf[i_1] = 0;

```



把 v11 的值附加到之前刚刚计算好的 md5 的数值后面，v11 来自于 v14，而 v14 并没有设置的地方，这里是 IDA F5 识别错了。直接看汇编吧。

```

loc_787B160E ; r1 = 0x13
 MOVS R1, #0x13
 MOVS R0, R5 ; r0--> key_sum
 BL calc_sum
 MOVS R4, R1

```

实际上 v14 就是调用 calc\_sum 后的 r1 寄存器。

跟进到 sub\_788640D4

```

sub_788640D4
; FUNCTION CHUNK AT 788640C4 SIZE 00000010 BYTES
 CMP R1, #0
 BEQ loc_788640C4

```

```

; START OF FUNCTION CHUNK FOR sub_78863FF8
; ADDITIONAL PARENT FUNCTION sub_788640D4
loc_788640C4
 CMP R0, #0
 MOVGT R0, #0x7FFFFFFF
 MOVLT R0, #0x80000000
 B loc_78864B24
; END OF FUNCTION CHUNK FOR sub_78863FF8

```

首先对传入的参数和 lr 进行了保存，然后调用 sub\_78864000，然后用之前保存的参数值与 sub\_78864000 的返回值进行运算，结果保存到 r1（专门用来迷惑 ida ^\_^）。

伪代码：

```

a1 = sub_78864000(a1, a2)
t = save_a2 * a1
ret = save_a1 - t
return ret

```

sub\_78864000 看起来比较复杂，我就直接用 python，照着重写了一遍。

```

else
{
 v4 = a1;
 if (a1 < 0)
 v4 = -a1;
 if (v4 <= a2)
 {
 if (v4 < a2)
 a1 = 0;
 if (v4 == a2)
 a1 = (v3 >> 31) | 1;
 }
 else if (a2 & (a2 - 1))
 {
 v5 = __clz(a2) - __clz(v4);
 v6 = a2 << v5;
 v7 = 1 << v5;
 a1 = 0;
 while (1)
 {
 if (v4 >= v6)
 {
 v4 -= v6;
 a1 |= v7;
 }
 }
 }
}

```

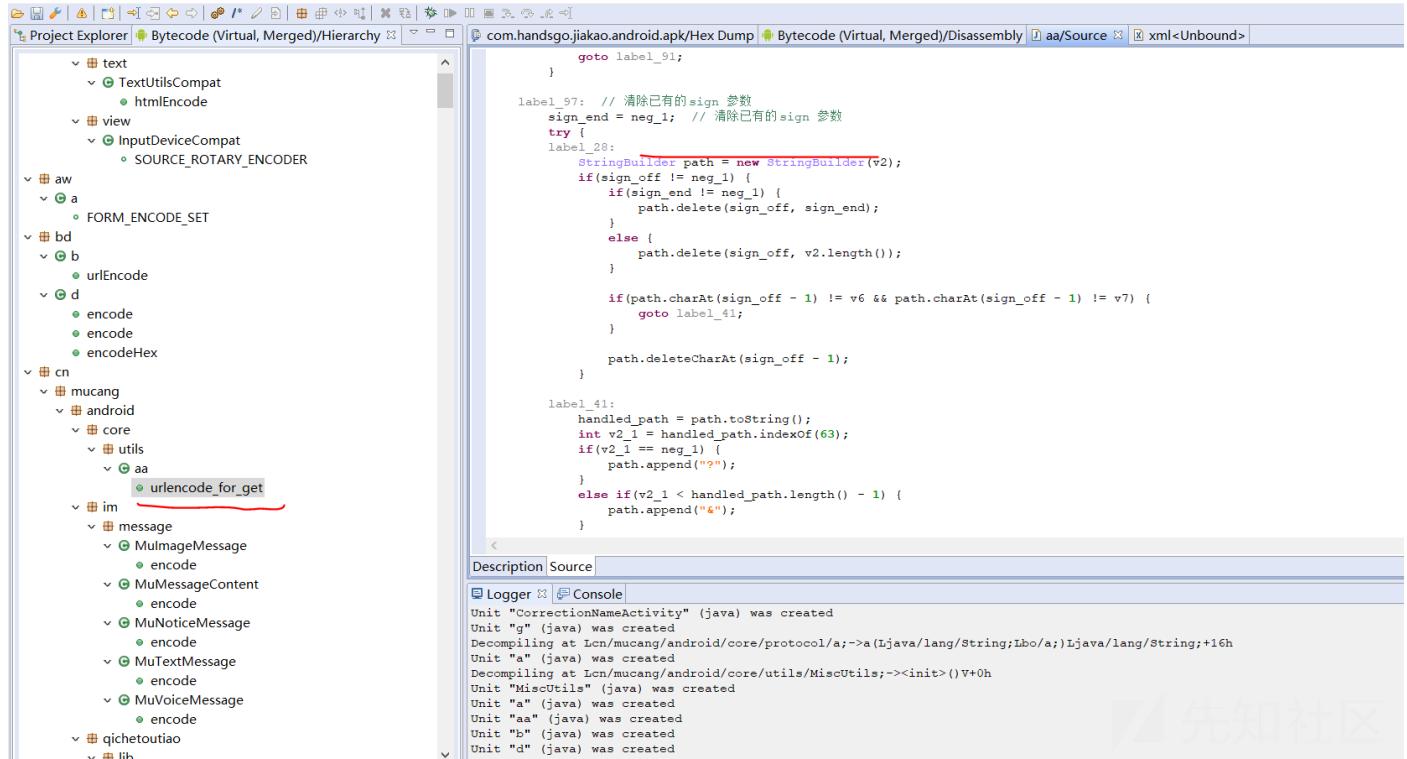
其中有一个有意思的地方：ida 无法对 `clz` 指令进行转换，所以用

```
v5 = __clz(a2) - __clz(v4);
```

表示

```
CLZ R2, R1
CLZ R0, R3
SUB R0, R2, R0
```

这使得重写造成了困扰，我的解决办法是，根据 `clz` 的作用，自己实现。



实现的 python 代码如下：

```
def calc_clz(reg):
 return 32 - len(str(bin(reg))[2:])
```

最后的脚本：

```
from urllib.parse import urlparse
from pprint import pprint
import base64
import hashlib

示例数据包
POST /api/open/v2/forgot-password/check.htm?_platform=android&_srv=t&_appName=jiakaoabodian&_product=%E9%A9%BE%E8%80%83%E5%AE%9D%E5%85%B8&_vendor=xiaomi&_renyuan=XYX&_version=
User-Agent: Mozilla/5.0 (Linux Android 5.1.1 SM-G9350 Build/LMY48Z) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/39.0.0.0 Safari/537.36
Accept-Encoding: gzip, deflate
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 83
Host: auth.mucang.cn
Connection: close

phoneNumber=13333333333&captchaId=e5410186e8afa433e4a4e8019901ff40&captchaCode=5659
```

```
def calc_sum(a1, a2):
 save_a1 = a1
 save_a2 = a2
 v3 = a1 ^ a2
 if a2 < 0:
 a2 = -a2
 if a2 == 1:
 if (v3 ^ a1) < 0:
 a1 = -a1
```

```

else:
 v4 = al
 if al < 0:
 v4 = -al
 if v4 <= a2:
 if v4 < a2:
 a1 = 0
 if v4 == a2:
 a1 = (v3 >> 31) | 1
 elif (a2 & (a2 - 1)):
 v5 = calc_clz(a2) - calc_clz(v4)
 v6 = a2 << v5
 v7 = 1 << v5
 a1 = 0
 while True:
 if v4 >= v6:
 v4 -= v6
 a1 |= v7
 if v4 >= v6 >> 1:
 v4 -= v6 >> 1
 a1 |= v7 >> 1
 if v4 >= v6 >> 2:
 v4 -= v6 >> 2
 a1 |= v7 >> 2
 if v4 >= v6 >> 3:
 v4 -= v6 >> 3
 a1 |= v7 >> 3
 v8 = v4 == 0
 if v4:
 v7 >>= 4
 v8 = v7 == 0
 if v8:
 break
 v6 >>= 4
 if v3 < 0:
 a1 = -a1
else:
 a1 = v4 >> (31 - calc_clz(a2))
 if v3 < 0:
 a1 = -a1
t = save_a2 * a1
ret = save_a1 - t
return ret

def decode_key(key):
 key = base64.b64decode(key)
 de_key = ''
 for c in key:
 de_key += chr((c - 42) ^ 0x2a)
 return de_key

def get_md5(src):
 m = hashlib.md5()
 m.update(src.encode('UTF-8'))
 return m.hexdigest()

def calc_clz(reg):
 return 32 - len(str(bin(reg))[2:])

def sign_url(url, key):
 o = urlparse(url)
 key = key[5:]
 # target = o.path + "?" + o.query

 query_list = o.query.split("&")
 for q in query_list:

```

```
if "sign=" in q:
 query_list.remove(q)
target = o.path + "?" + "&".join(query_list)
target = "/api/open/v2/forgot-password/check.htm?_platform=android&_srv=t&_appName=jiakaobaodian&_product=%E9%A9%BE%E8%80%83%E5%AE%9D%E5%85%B8&_vendor=xiaomi&_renyuan=XYX&_
decoded_key = decode_key(key)
md5 = get_md5(target + decoded_key)
key_sum = 0
for k in key:
 key_sum += ord(k)

sum = hex(calc_sum(key_sum, 0x13))[2:]
if len(sum) % 2:
 sum = "0" + sum
sign = md5 + sum
print(sum)
print(sign)

if __name__ == '__main__':
 sign_key = "*#06#131rRYudcZZ2fIx9fI6VqJV8"
 url = "https://auth.mucang.cn/api/open/v2/forgot-password/check.htm?_platform=android&_srv=t&_appName=jiakaobaodian&_product=%E9%A9%BE%E8%80%83%E5%AE%9D%E5%85%B8&_vendor=xiac

sign_url(url, sign_key)
print(calc_sum(0x82b, 0x13))
```

来源: <https://www.cnblogs.com/hac425/p/9416910.html>