

# Crack App | 某新闻 app 参数 sn 加密逻辑分析

原创 煌金的咸鱼 咸鱼学Python 2022-05-11 20:02

收录于合集

#Crack App 11 #app逆向 6 #Frida 4



咸鱼学Python

高级爬虫工程师，专注逆向、爬虫与反爬虫技术

270篇原创内容

公众号

点击上方“咸鱼学Python”，选择“**加为星标**”

第一时间关注Python技术干货！



图源：网络

## 今日目标

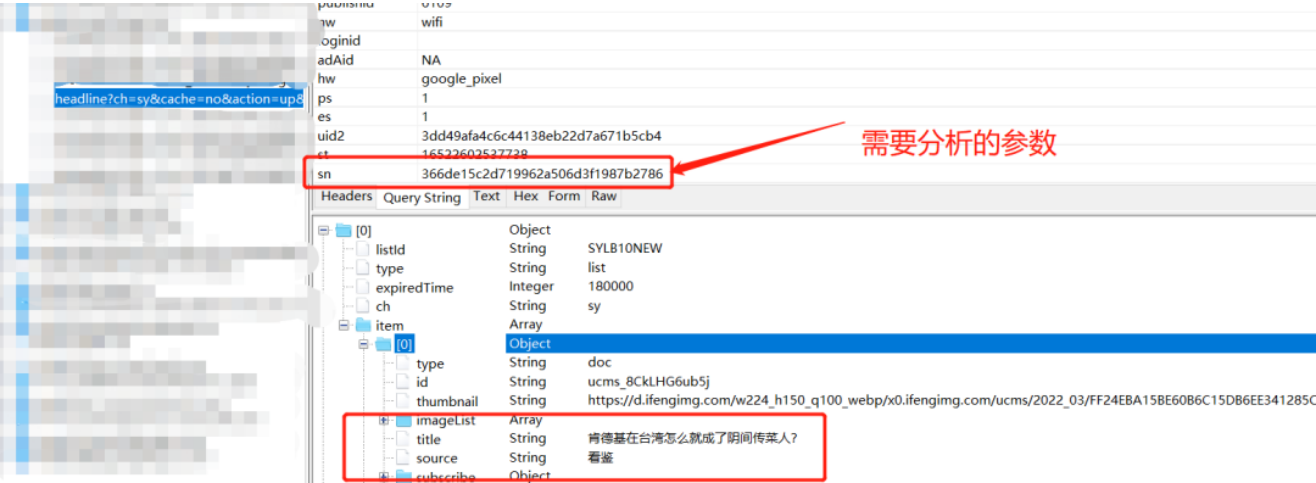
aHR0cHM6Ly93d3cud2FuZG91amlhLmNvbS9hcHBzLzQwMzYz

爬虫相关的 Js 逆向越来越卷了，吓得我赶紧开始学 App 🤔

抓包分析

今天分析的是这个 app 的首页信息流

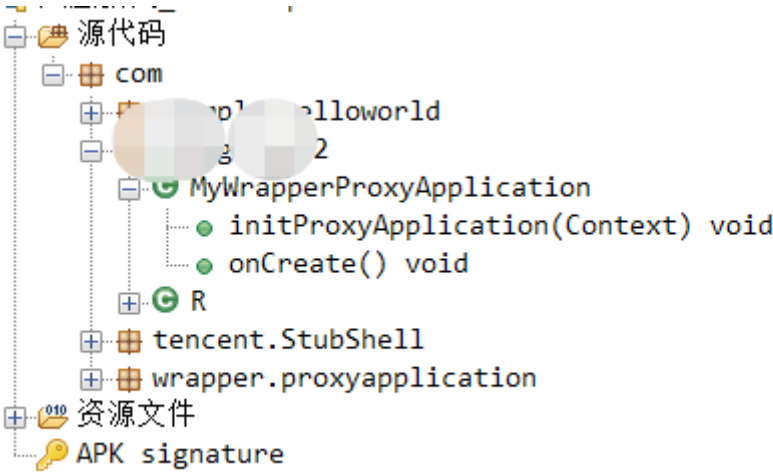
使用的抓包组合是 charles + postern 抓包如下



通过这个包可以看到返回了 title 还有新闻相关来源，参数中带有 sn

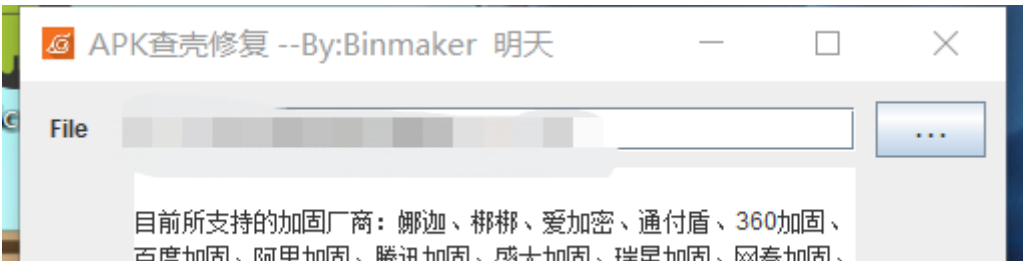
这个参数的长度让人不得不怀疑是不是 md5 的加密，不过还是要看看加密的逻辑是不是

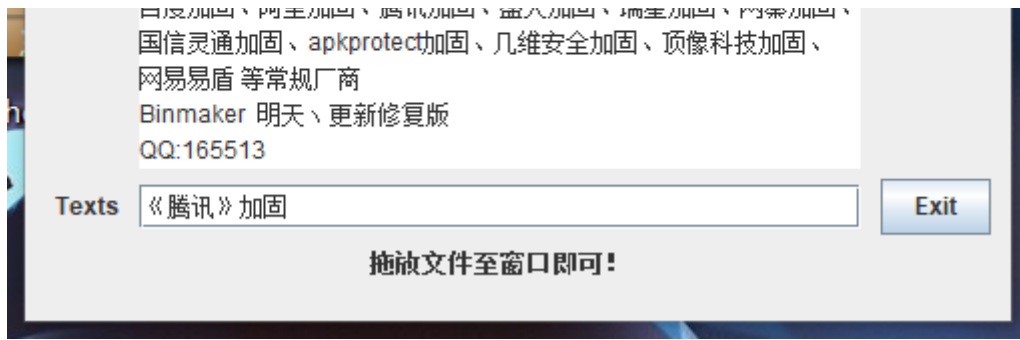
把 apk 拖入 jadx 看看



70 + M 一下就反编译完了，一看就不对劲，jadx 中啥都没有，应该是加壳了

查个壳看看





用的是腾讯加固，所以分析之前还是要脱壳看看

## App 脱壳

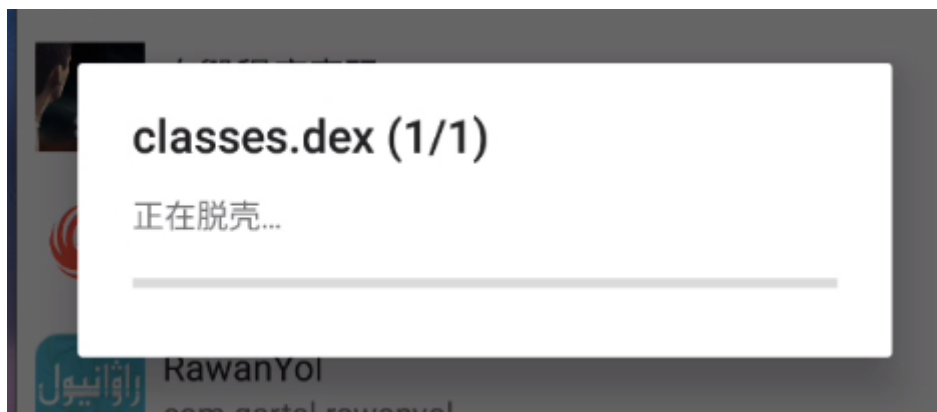
app 脱壳的工具有很多，比如：

yang 神 frida\_dump : [https://github.com/lasting-yang/frida\\_dump](https://github.com/lasting-yang/frida_dump)

寒冰大大的 FART : <https://github.com/hanbinglengyue/FART>

目前世面上的脱壳工具还是很多的，今天我是用的是 BlackDex ，一款脱壳 App，先试试看能不能脱下来我们需要的内容

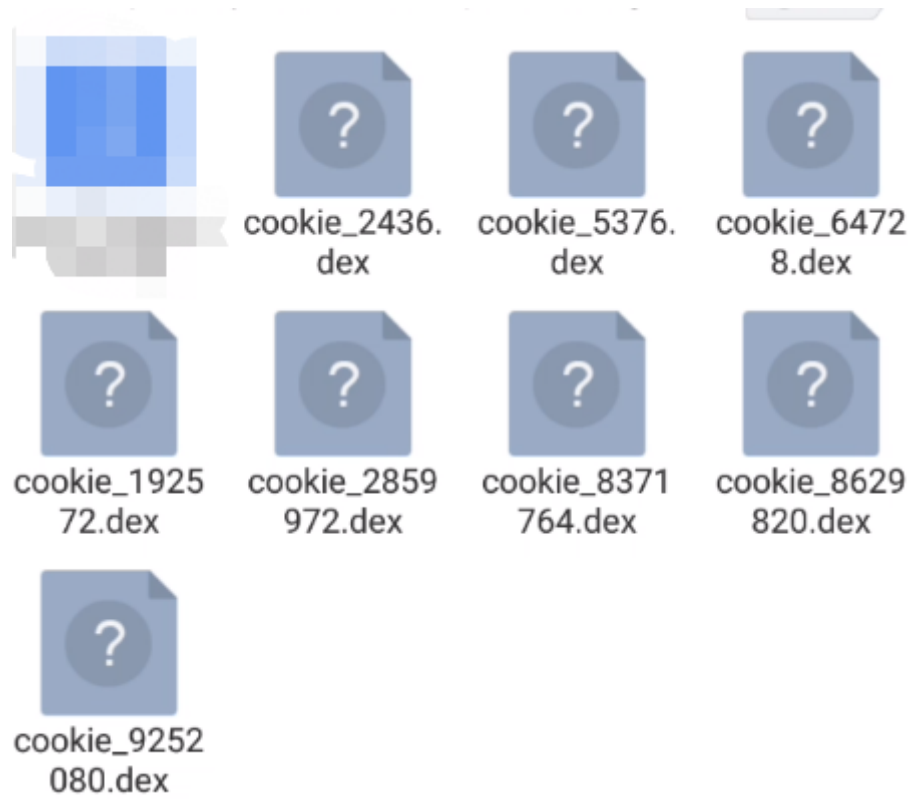
打开 BlackDex 然后再打开目标 App 就可以了



脱壳完成就会保存到指定的目录下



用 ES文件浏览器 打开对应的路径



就看到脱好的 dex 咯

然后将这些个 dex 压缩好，剪切到的 /sdcard 中，然后用命令拉到 pc 上

```
adb pull /sdcard/xxx.zip [pc path]
```

名称	修改日期	类型	大
com.iew2.zip	2022/5/11 18:03	压缩(zipped)文件...	
3.apk	2022/4/14 22:51	APK 文件	

然后再用 jadx 分析就好了

ps: 压缩的 zip 直接拖到 jadx 里面就行了

加密参数定位与分析

拖到 jadx 里面编译之后检索 sn= 可以找到下面的逻辑

```
public String a(boolean z) {  
    ...  
}
```

```

Long valueOf = Long.valueOf(System.currentTimeMillis() / 1000);
String str = String.valueOf(valueOf) + ((int) ((Math.random() * 8999.0d) + 1000.0d));
StringBuilder sb = new StringBuilder();
sb.append("gv=" + this.f2360a);
sb.append("&av=" + this.b);
sb.append("&uid=" + this.c);
sb.append("&deviceid=" + this.d);
sb.append("&proid=" + this.e);
sb.append("&os=" + this.f);
sb.append("&df=" + this.g);
sb.append("&vt=" + this.h);
sb.append("&screen=" + this.i);
sb.append("&publishid=" + this.j);
sb.append("&nw=" + this.k);
sb.append("&loginid=" + this.l);
if (z) {
    sb.append("&token=" + this.m);
}
sb.append("&adAid=" + this.n);
sb.append("&hw=" + this.o);
sb.append("&ps=" + this.p);
sb.append("&es=" + this.q);
sb.append("&uid2=" + this.r);
sb.append("&st=" + str);
sb.append("&sn=" + a(str, z));
if ("1".equals(amd.dD)) {
    sb.append(ContainerUtils.FIELD_DELIMITER);
    sb.append("clear");
    sb.append(ContainerUtils.KEY_VALUE_DELIMITER);
    sb.append(amd.dD);
}
return sb.toString();
}

```



通过上面的逻辑大概可以知道 `valueOf` 是时间戳然后 `str` 是根据时间戳计算出来的结果

现在看 `a`，`a` 的方法就在下面

```

public String a(String str, boolean z) {
    String str2;
    String str3 = z ? this.m : "";
    try {
        str2 = this.f2360a + this.e + this.j + this.c + this.l + str3 + str + NativeSecureparam.readMD5Key();
    } catch (Throwable unused) {
        str2 = this.f2360a + this.e + this.j + this.c + this.l + str3 + str;
    }
    return c.jy.b(str2).toLowerCase();
}

```

这里对 `z` 做了一个判断，所以需要返回看下我们抓包里面有没有 `z` 代表的 `token`，如果有的话这里就要带上 `str3` 也就是 `token` 一块计算，如果没有的话就不用

传入了 `str` 和 `z` 之后就计算出一个 `str2`，这个 `str2` 会进入 `b` 方法

跟进去看看 `b`，这里的 `b` 的逻辑是一个 md5，逻辑很清晰

```

public static String b(String str) {
    StringBuffer stringBuffer;
    NoSuchAlgorithmException e;
    StringBuffer stringBuffer2 = new StringBuffer();
    byte[] bytes = str.getBytes();
    try {
        MessageDigest instance = MessageDigest.getInstance("MD5");
        instance.reset();
        instance.update(bytes);
    }
}

```

```

byte[] digest = instance.digest();
stringBuffer = new StringBuffer();
for (int i = 0; i < digest.length; i++) {
    try {
        if (Integer.toHexString(digest[i] & UByte.MAX_VALUE).length() == 1) {
            stringBuffer.append(0);
        }
        stringBuffer.append(Integer.toHexString(digest[i] & UByte.MAX_VALUE));
    } catch (NoSuchAlgorithmException e2) {
        e = e2;
        e.printStackTrace();
        return stringBuffer.toString().toUpperCase();
    }
}
digest.toString();
} catch (NoSuchAlgorithmException e3) {
    e = e3;
    stringBuffer = stringBuffer2;
    e.printStackTrace();
    return stringBuffer.toString().toUpperCase();
}
return stringBuffer.toString().toUpperCase();
}

```

所以真就是我们文章开头猜的那样，最后经过了一次 md5 计算

知道最后的逻辑是 md5 之后，就要看看 md5 的入参是什么，就要用到 frida 动态调试了

这个时候就有人问了，这些参数明明在上面的逻辑里面都能看到，为什么还要动态调试？

因为静态分析的结果不可信，动态调试输出的结果更加可信，所以要动态调试和静态调试相互印证

## Frida hook 动态调试

用下面的 hook 代码

```

Java.perform(function () {
    var aaa = Java.use('类名');
    aaa.b.overload('java.lang.String').implementation = function (a) {
        console.log("参数==>:" + a);
        var result = this.b(a);
        console.log("结果==>:" + result);
        return result;
    };
});

```

得到的结果如下

```

参数==>: ifengn 5109v001yE2Y0ATN3gjNwAjNyYmM40A0gfr3r340gf16522659139022acF%#*{_blmQt@..ifvy
结果==>: 51520B E73D9C138D69DE642
参数==>: ifengn 5109v001yE2Y0ATN3gjNwAjNyYmM40A0gfr3r340gf16522659137499acF%#*{_blmQt@..ifvy
结果==>: 3F6DE8 3C5FA7544C6043C7E

```



```

参数==>: .fengr 6109v001yE2Y0ATN3gjNwAjNyYmM40A0gfr3r340gf16522659138189acF%#*{_blmQt@..ifvy
结果==>: 5B2BA 373D04332D79320F1
参数==>: .fengr 6109v001yE2Y0ATN3gjNwAjNyYmM40A0gfr3r340gf16522659139150acF%#*{_blmQt@..ifvy
结果==>: 08ED98 F3874773953AD7363
参数==>: .fengr 6109v001yE2Y0ATN3gjNwAjNyYmM40A0gfr3r340gf16522659138459acF%#*{_blmQt@..ifvy
结果==>: 6F5168 911E80ABF5DCD327E
参数==>: .fengr 6109v001yE2Y0ATN3gjNwAjNyYmM40A0gfr3r340gf16522659132489acF%#*{_blmQt@..ifvy
结果==>: 17F018 640300DEDCD65C2F7
参数==>: .ifengr 6109v001yE2Y0ATN3gjNwAjNyYmM40A0gfr3r340gf16522659138574acF%#*{_blmQt@..ifvy
结果==>: B3CC78 B6AF2105FF23C41D9

```

发现除了 `jadx` 里面能找到的参数之外，后面跟了一串乱码，这一串乱码是怎么来的？

回到 `jadx` 里面，可以看到就是 `NativeSecureparam.readMD5Key()`；返回的

```

    public String a(String str, boolean z) {
        String str2;
        String str3 = z ? this.m : "";
        try {
            str2 = this.f2360a + this.e + this.j + this.c + this.l + str3 + str + NativeSecureparam.readMD5Key();
        } catch (Throwable unused) {
            str2 = this.f2360a + this.e + this.j + this.c + this.l + str3 + str;
        }
        return c.jy.b(str2).toLowerCase();
    }

```

现在需要分析 `NativeSecureparam.readMD5Key()`；是怎么计算出来的，通过这个方法的名字可以知道，这个字符串就是 `md5` 的盐值

## so 分析

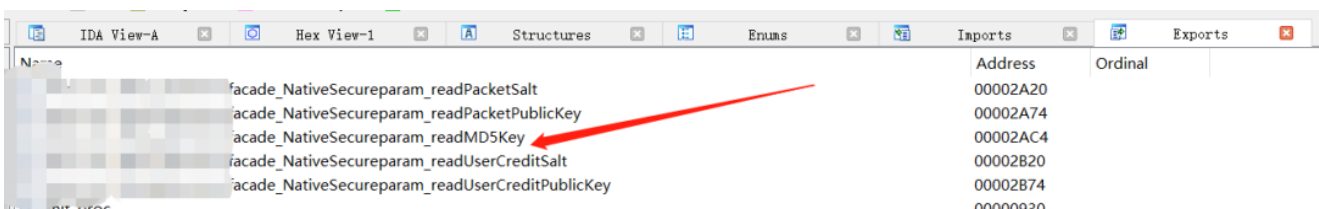
通过 `jadx` 可以知道这个方法是一个 `Native` 方法

```

2
3 public class NativeSecureparam {
4     public static native String readMD5Key() throws Throwable;
5
6     public static native String readPacketPublicKey() throws Throwable;
7
8     public static native String readPacketSalt() throws Throwable;
9
10    public static native String readUserCreditPublicKey() throws Throwable;
11
12    public static native String readUserCreditSalt() throws Throwable;
13
14    static {
15        try {
16            System.loadLibrary("ifeng_secure");
17        } catch (Throwable unused) {
18        }
19    }
20 }

```

来自 `libifeng_secure.so`，用 `IDA` 打开对应的 `so` 文件



32 位的 so 报错 JUMPOUT，所以直接偷懒用 64 的

找到对应的函数，点进去，可以看到下面的逻辑

```
__int64 __fastcall Java_com_android_nativeSecureparam_readMD5Key(__int64 a1)
{
    char v2[24]; // [xsp+0h] [xbp-30h] BYREF
    __int64 v3; // [xsp+18h] [xbp-18h]

    v3 = *(_QWORD *)(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
    strcpy(v2, "acF5b..ifvy");
    return (*(__int64 (__fastcall **)(__int64, char *))(*(_QWORD *)a1 + 1336LL))(a1, v2);
}
```

这里要修改一下参数类型，可以看到下面的逻辑

```
1 jstring __fastcall Java_com_android_nativeSecureparam_readMD5Key(JNIEnv *a1)
2 {
3     char v2[24]; // [xsp+0h] [xbp-30h] BYREF
4     __int64 v3; // [xsp+18h] [xbp-18h]
5
6     v3 = *(_QWORD *)(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
7     strcpy(v2, "acF5b..ifvy");
8     return (*a1->NewStringUTF(a1, v2));
9 }
```

其实就是将一串字符赋值 v2 并返回，没有其他的操作，回到 jadx 再看下逻辑就知道其实 sn 的逻辑就是

```
md5(版本号 + proid + publishid + uid + loginid(未登录是空) + "" + st + so中的字符串)
```

md5 就不用算法还原了吧，大家应该都会。

以上就是今天的全部内容了~

公众号配套技术交流群，备注【咸鱼666】，入群交流





我是没有更新就在摸鱼的咸鱼

收到请回复~

我们下次再见。




咸鱼学Python

高级爬虫工程师，专注逆向、爬虫与反爬虫技术

270篇原创内容

公众号



对了，看完记得一键三连，这个对我真的很重要。

收录于合集 #Crack App 11

下一篇 · Crack App | 一键刷机脚本，逆向环境搭建一步到位

发表于上海

喜欢此内容的人还喜欢

揭秘.NET Core剪裁器背后的技术

微软中国MSDN

---

Java on Visual Studio Code 4月更新

微软中国MSDN

---

学习在云上部署Java应用程序

微软中国MSDN