你上线的APP安全吗？

自我介绍

黄帅

博客地址:http://blog.csdn.net/mynameishuangshuai

曾在一家安全公司任职，从事过十多款安卓应用的研发，工作期间热衷于研究移动应用的安全问题；希望通过今天的分享把安卓研发的安全知识普及给安卓开发者，帮助大家避免不必要的漏洞，让上线的APP更加安全。

分享主题

1. 当前常见的APP安全问题
2. 程序员如何做到安全研发
3. Android M 和 N 的安全更新

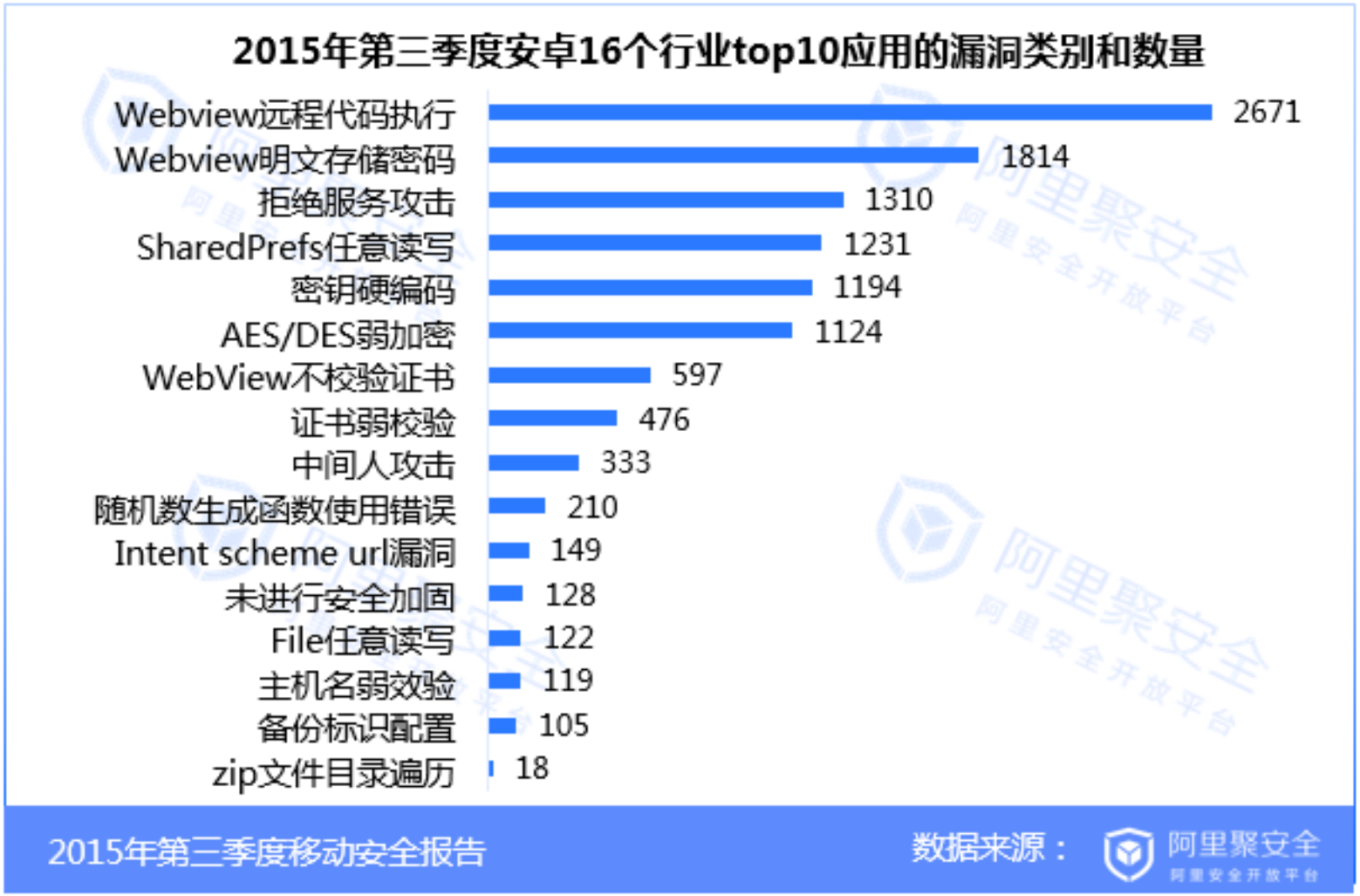
分享主题

1 当前常见的APP安全问题

当前常见的APP安全问题

调查：大家平时开发中都遇到过哪些安全问题？

当前常见的APP安全问题



在电商、社交、金融、游戏、政务等等16个行业的top10应用中100%都有漏洞

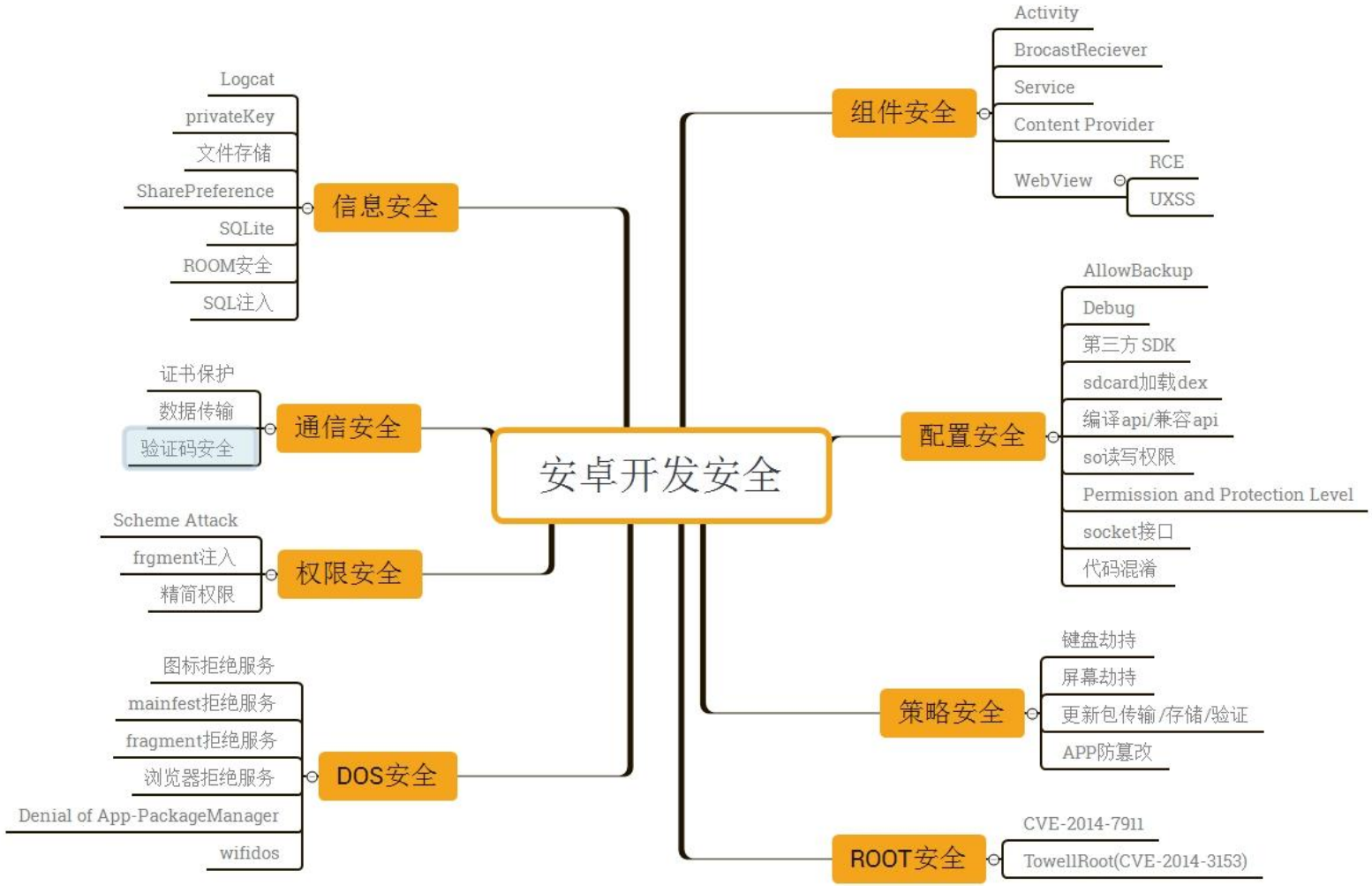
分享主题

2

程序员如何做到安全研发 ？

一、常见安全问题分析总结

总结Android研发安全要点



（一）组件安全-Activity

-Activity访问权限的控制（可能会导致恶意调用页面，接收恶意数据）

1.私有Activity不应被其他应用启动且应该确保相对是安全的
2.关于Intent的使用要谨慎处理接收的Intent以及其携带的信息，尽量不发送敏感信息，并进行数据校验
3.设置android:exported属性，不需要被外部程序调用的组件应该添加android:exported=”false”属性

-Activity劫持（正常的Activity界面被恶意攻击者替换上仿冒的恶意Activity界面进行攻击和非法用途。）

在登录窗口或者用户隐私输入等关键Activity的onPause方法中检测最前端Activity应用是不是自身或者是系统应用，如果发现恶意风险，则给用户一些警示信息，提示用户其登陆界面以被覆盖，并给出覆盖正常Activity的类名。

设置Activity的属性：可防止系统截屏
this.getWindow().addFlags(WindowManager.LayoutParams.FLAG_SECURE);

（一）组件安全-Service

Service劫持、拒绝服务

以拒绝服务为例，常见的两种漏洞：
1.Java.lang.NullPointerException 空指针异常导致的拒绝服务
2.类型转换异常导致的拒绝服务ClassCastException

1.应用内部使用的Service应设置为私有
2.针对Service接收到的数据应该验证并谨慎处理
3.内部Service需要使用签名级别的protectionLevel来判断是否未内部应用调用

（一）组件安全-BrocastReciver

发送虚假、恶意广播

1.设置AndroidManifest.xml，添加下面的属性

```
<receiver android:name="MyBroadCastReceiver" android:exported="true">
<intent-filter>
   <action android:name="MyBroadcast">
   </action>
</intent-filter>
</receiver>
```

false

2.控制权限来防止类似事件发生

（一）组件安全-Content Provider

Content Provider扮演着应用间数据共享桥梁的角色，其主要的安全风险为SQL注入和本地文件目录遍历

1.设置AndroidManifest.xml，添加下面的属性

```
<receiver android:name="MyBroadCastReceiver" android:exported="true">
<intent-filter>
   <action android:name="MyBroadcast">
   </action>
</intent-filter>
</receiver>
```

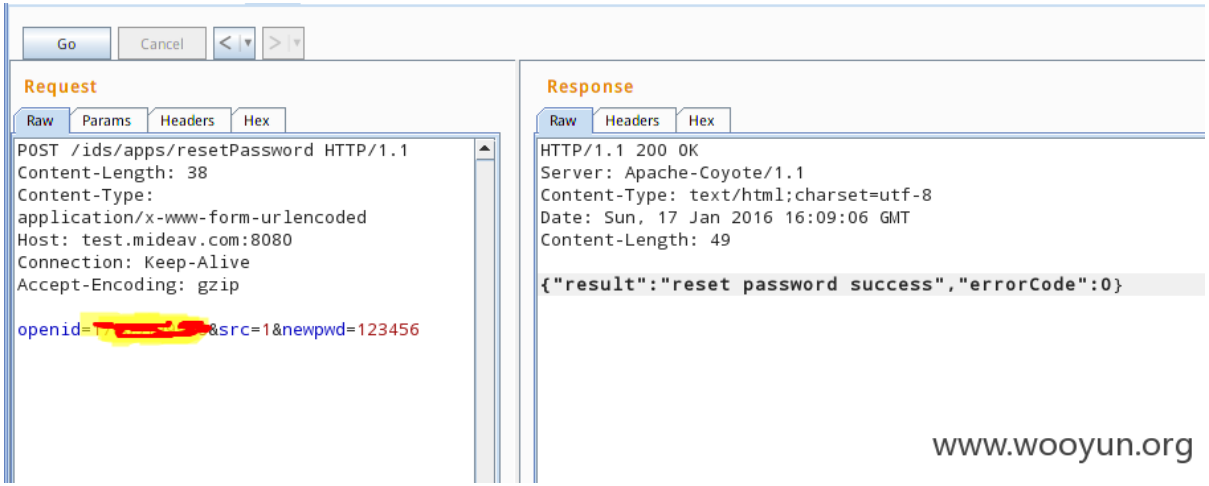
false

2.通过控制权限来限制对Content Provider的访问

（二）传输安全-本地校验用户验证码

code 区域

```
HTTP/1.1 200 OK

Server: Apache-Coyote/1.1

Content-Type: text/html;charset=utf-8

Date: Sun, 17 Jan 2016 15:30:34 GMT

Content-Length: 68


{"result":{"captcha":"905697","mobile":"13888888888"},"errorCode":0}
```



（二）传输与安全-数据加密

常用加密算法有：MD5加密、DES加密、加密，最大的问题是密钥的存储

研发人员应做到：

1.在代码层一定要把密钥放的更隐蔽一些。

2.所有APP的客户端不要用同一个密钥，对于密钥生成的方法，尽可能的根据里的用户ID或者手机的一些信息来生成一个，避免随机生成随机。

（三）第三方组件安全-DDOS

之前做APP项目，在研发后期对待上线的APP进行了好多方面的安全测试，自己觉得APP不会有什么安全问题了，但是APP上线后却被细心的白帽子提交了漏洞，究其原因是我们APP项目中使用的第三方推送平台，该平台SDK存在一个漏洞，导致攻击者可以伪造广播，利用其进行恶意代码执行和获取私有文件。

（四）WebView远程代码执行漏洞

Android API level 16以及之前的版本存在远程代码执行安全漏洞，该漏洞源于程序没有正确限制使用WebView.addJavascriptInterface方法，远程攻击者可通过使用Java Reflection API利用该漏洞执行任意Java对象的方法，简单的说就是通过addJavascriptInterface给WebView加入一个JavaScript桥接接口，JavaScript通过调用这个接口可以直接操作本地的JAVA接口。

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。攻击者利用该漏洞可以根据客户端能力为所欲为。

（四）WebView远程代码执行漏洞

1. API Level等于或高于17的Android系统
Google在4.2版本之后，规定允许被调用的函数必须以@JavascriptInterface进行注解，所以如果某应用依赖的API Level为17或者以上，就不会受该问题的影响（注：Android 4.2中API Level小于17的应用也会受影响）。

2. API Level小于17的Android系统
建议不要使用addJavascriptInterface接口，如果一定要使用:
1) 如果使用HTTPS协议加载URL，应进行证书校验防止访问的页面被篡改挂马；
2) 如果使用HTTP协议加载URL，应进行白名单过滤、完整性校验等防止访问的页面被篡改；
3) 如果加载本地Html，应将html文件内置在APK中，以及进行对html页面完整性的校验；

（四）XSS

XSS，这个问题其实在手机上并没有像PC那么严重。目前这种攻击肯定是存在的，但是基于攻击可能会导致的危害，其实并没有像PC上那么广泛。但是大家也需要注意一下，尤其这个东西可能更多的是说去盗取你的cookies信息，通过这个cookies信息可以做免密码的登录的东西。

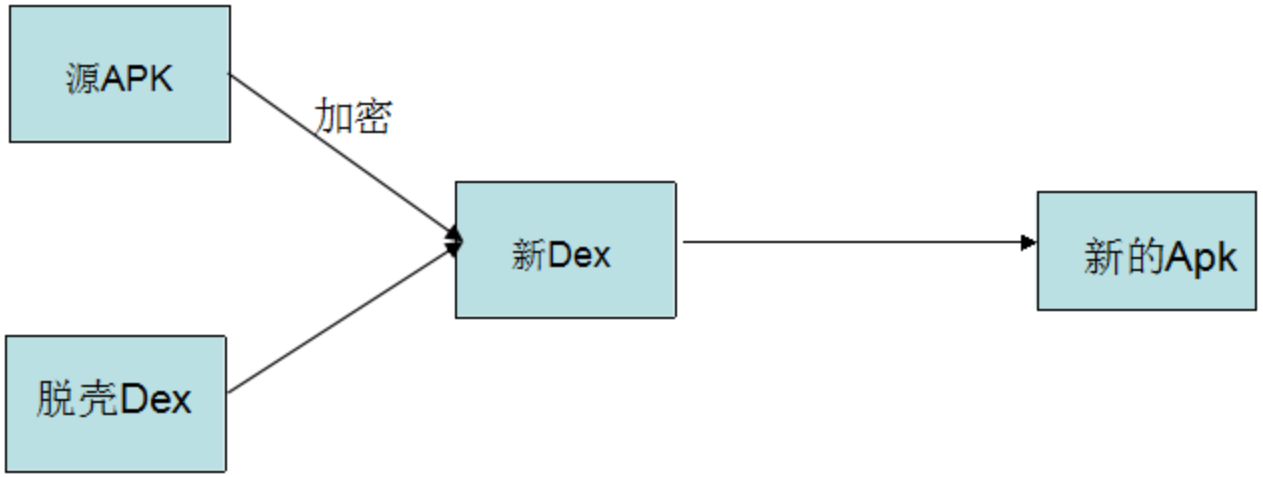
（五）应用安全-代码混淆

混淆代码能有效防止apk文件被反编译，进而查看源代码。
在android项目中找到module的gradle配置文件，添加proguard配置

```
-libraryjars class_path 应用的依赖包，如android-support-v4
-keep [,modifier,...] class_specification 这里的keep就是保持的意思，意味着不混淆某些类
-keepclassmembers [,modifier,...] class_specification 同样的保持，不混淆类的成员
-keepclasseswithmembers [,modifier,...] class_specification 不混淆类及其成员
-keepnames class_specification 不混淆类及其成员名
-keepclassmembernames class_specification 不混淆类的成员名
-keepclasseswithmembernames class_specification 不混淆类及其成员名
-assumenosideeffects class_specification 假设调用不产生任何影响，在proguard代码优化时会将该调用remove掉。
-dontwarn [class_filter] 不提示warnning
```

也可以使用第三方的GexGuard进行高级混淆，以获得更多的安全性。

（五）应用安全-APP加固



梆梆安全 BANGCLE

NAGA·IN

360加固保 jiagu.360.cn

（五）应用安全-二次打包

攻击不一定是要偷走些什么，有时候也会加入猛料！

Java 代码中加入签名校验
NDK 中加入签名校验
热修复dex文件校验
利用二次打包工具本身的缺陷阻止打包

（五）应用安全-键盘攻击

输入数据监听攻击
键盘截屏攻击
输入数据篡改攻击
未加密前篡改
来自系统底层的内存dump攻击

（五）应用安全-键盘攻击

自己绘制软键盘

对键盘的数据输入过程、数据存储过程、内存数据换算过程进行加密

使用第三方加固软键盘

（五）应用安全-外部调用APP

Android有一个特性，可以通过点击网页内的某个链接打开APP，或者在其他APP中通过点击某个链接打开另外一个APP（AppLink），一些用户量比较大的APP，已经通过发布其AppLink SDK，开发者需要申请相应的资格，配置相关内容才能使用。
这些都是通过用户自定义的URI scheme实现的，不过背后还是Android的Intent机制。

第一种用户自定义的URI scheme形式如下：

```
scheme://host/path?parameters
```

第二种的Intent Scheme URL形式如下：

```
intent://host#Intent;参数;end
```

（五）应用安全-外部调用APP

开发中我们常在APP中获取到来自网页的数据后，并生成一个intent，然后发送给别的组件使用这些数据。比如使用Webview相关的Activity来加载一个来自网页的url，如果此url来自url scheme中的参数，如：temp://temp.gdg.com?load_url=http://www.gdg.net。

如果在APP中，没有检查获取到的load_url的值，攻击者可以构造钓鱼网站，诱导用户点击加载，就可以盗取用户信息。

（五）应用安全-外部调用APP

对于开发者来讲，还是不要信任来自网页端的任何intent，为了安全起见，在使用网页传过来的intent时，还是要进行过滤和检查。具体做法可以参考两篇博文《Android Intent Scheme URLs攻击》和《Intent Scheme URL attack》。作者提供了如下的安全Intent Filter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);
```

二、程序员的安全编码规范

程序员安全意识修养

程序员的编码规范很重要，很重要，很重要！

代码规范

对于重要的类、方法、变量我们在定义时一定要明白一下几个问题：

它们返回什么

它们获取什么作为参数

它们是否绕过安全性检查

它们是否是 public、 private等等

它们是否含有绕过包边界从而绕过包保护的方法调用

代码规范

对于高安全性的代码为了防止暴露，可以从以下几个方面考虑：

限制对变量的访问

让每个类和方法都成为 final，除非有足够的理由不这样做

不要依赖包作用域

使类不可克隆（克隆允许绕过构造器而轻易地复制类实例）

使类不可序列化（避免外部源获取对象的内部状态的控制）

避免硬编码敏感数据

查找恶意代码

allowBackup

allowBackup，有的人对它很陌生，它其实是Google提供了一个备份的功能，如果用户要换手机了，可以用它把APP导到另一个手机上，用户本身使用的一些数据（聊天记录）会跟着一起备份过去。之前在乌云漏洞报告平台上，发现很多公司去把这个属性给加上，除了备份使用外，它还带来了两个问题：

1. 恶意程序也可以利用这个备份，并把你备份的数据传到网上，在他的手机上打开。如果我们在做换手机操作过程中没有做全面的校验，那将会造成直接性的用户隐私信息泄露，用户的个人信息和隐私操作将会被暴露出来。

2.这个备份的时候，会生成一个加密的文件。这个加密文件是可以被篡改的，而且你再利用这个备份文件导到另一个手机的时候，校验的时候可以直接过。就是你把备份导出来，病毒可以在导入文件加入恶意的代码或者其他的数据。你去进行备份的时候，相当于重新安装了，你在另外一个手机重新安装的时候，你重装的的APP上就携带着的病毒，后果非常危险。

程序员安全意识修养

统一添加全部权限，为了省事 

```
<uses-permission android:name="android.permission.RECEIVE_USER_PRESENT" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS" />
<uses-permission android:name="com.android.launcher.permission.READ_SETTINGS" />

……
```

一些细节问题

关闭调试模式

上线前关闭调试模式android:debuggable=”false“

混淆日志

平时开发中大家最常使用的就是Log日志，喜欢把一些重要数据通过日志打印出来方便查看，比如把用户的账号密码打印出来，如果不关闭日志打印，这在上线后就非常容易导致敏感信息泄露。不法人员通过监控Log，就能知道你这个APP使用的一些关键信息。

有人说，我统一设置日志开关，上线时把日志关闭不就行了么，其实这样做也不保险，因为逆向APP第一步动作就是先找Log的开关，找到开关之后，把它打开，然后跑一遍程序，就很容易找到隐私信息。我的建议是对于安全性要求较高的APP，尽量还是通过混淆把Log的所有的信息去掉。

已有APP如何发现安全问题？

如何检测发现APP漏洞

静态分析

在不运行源码的情况下，通过IDA、apktool、dex2jar、jd-gui、smali2dex等静态分析工具对应用进行反编译，并对反编译后的java文件、xml文件等文件静态扫描分析，通过关键词搜索等静态方式将具有安全隐患的代码进行摘录并存入到检测平台后台，为后续的安全检测报告提供数据依据。

动态调试

对应用软件安装、运行过程的行为监测和分析。检测的方式包括沙箱模型和虚拟机方式。虚拟机方式通过建立与Android手机终端软件运行环境几乎一样的虚拟执行环境，手机应用软件在其中独立运行，从外界观察应用程序的执行过程和动态，进而记录应用程序可能表现出来的恶意行为。

人工众测

1.企业招聘相关安全人员，对APP应用软件安装、运行过程的行为监测和分析，记录APP程序可能表现出来的风险行为。
2.使用第三方众测平台，帮助企业发现APP应用漏洞及风险。

免费的云测平台

360显微镜平台 http://appscan.360.cn/

腾讯金刚平台 http://service.security.tencent.com/kingkong

阿里聚安全平台 https://jaq.alibaba.com/

免费的云测平台

关键配置

1	程序数据任意备份	有风险	收起

风险描述

AndroidManifest.xml配置文件中没有设置allowBackup标志(默认为true)或将allowBackup标志设置为true。当这个标志被设置为true时应用程序数据可以备份和恢复，adb调试备份允许恶意攻击者复制应用程序数据。

修复建议

AndroidManifest.xml 配置文件中中设置为android:allowBackup="false"。

2	程序可被任意调试	安全	查看
3	Activity组件暴露	有风险	查看
4	BroadcastReceiver组件暴露	安全	查看
5	ContentProvider组件暴露	安全	查看
6	Service组件暴露	安全	查看

Google怎么应对Android安全问题

Android有这么多厉害的漏洞，Google家里人知道吗？

分享主题

3 Android M 和 N 的安全更新

Android6.0（M）的权限改进

谷歌在2015年8月份时候，发布了Android 6.0版本，其中的很大的一部分变化，是在用户权限授权上，谷歌意识到之前默认授权的不合理也不安全，所以6.0之后改进了用户权限授权模式，即只有在用户需要使用权限的时候，才去授权请求，提高了应用权限的安全等级。

Android6.0（M）的权限改进

右边是Android6.0以后规定的危险权限列表，它们包括身体传感器、日历、摄像头、通讯录、地理位置、麦克风、电话、短信以及存储空间等。

对于开发者来讲Android 6.0在我们原有的AndroidManifest.xml声明权限的基础上，又新增了运行时权限动态检测，要求我们需要更重视权限的安全管理，并设计友好的权限申请机制。

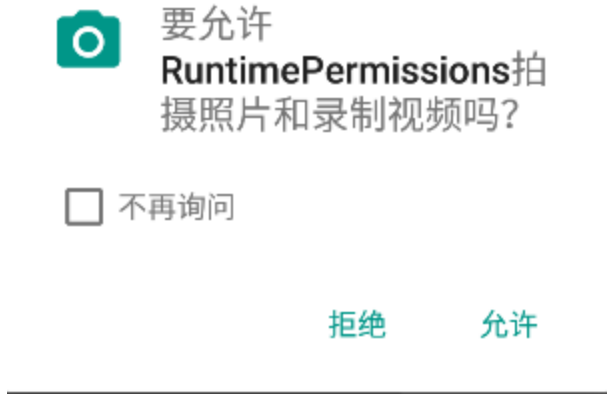
Permission Group	Permissions
CALENDAR	• READ_CALENDAR • WRITE_CALENDAR
CAMERA	• CAMERA
CONTACTS	• READ_CONTACTS • WRITE_CONTACTS • GET_ACCOUNTS
LOCATION	• ACCESS_FINE_LOCATION • ACCESS_COARSE_LOCATION
MICROPHONE	• RECORD_AUDIO
PHONE	• READ_PHONE_STATE • CALL_PHONE • READ_CALL_LOG • WRITE_CALL_LOG • ADD_VOICEMAIL • USE_SIP • PROCESS_OUTGOING_CALLS
SENSORS	• BODY_SENSORS
SMS	• SEND_SMS • RECEIVE_SMS • READ_SMS • RECEIVE_WAP_PUSH • RECEIVE_MMS
STORAGE	• READ_EXTERNAL_STORAGE • WRITE_EXTERNAL_STORAGE

Android6.0（M）的权限改进

检查并申请权限

```
if (ContextCompat.checkSelfPermission(this, Manifest.permission.WRITE_EXTERNAL_STORAGE)
        != PackageManager.PERMISSION_GRANTED) {
    //申请WRITE_EXTERNAL_STORAGE权限
    ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
            WRITE_EXTERNAL_STORAGE_REQUEST_CODE);
}
```

请求权限后，系统会弹出请求权限的Dialog



Android6.0（M）的权限改进

用户选择允许或需要后，会回调onRequestPermissionsResult方法, 该方法类似于onActivityResult

```
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    doNext(requestCode,grantResults);
}
```

我们接着需要根据requestCode和grantResults(授权结果)做相应的后续处理

```
private void doNext(int requestCode, int[] grantResults) {
      if (requestCode == WRITE_EXTERNAL_STORAGE_REQUEST_CODE) {
          if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
              // Permission Granted
          } else {
              // Permission Denied
          }
      }
  }
```

Android7.0（N）的安全更新

谷歌在2016年8月份时候，发布了Android 7.0版本，该版本Google对Android的安全做了大幅度的提升，并在9月6日将安全更新信息公布。

Android7.0（N）的安全更新

系统权限变更

1.在Android N之前，开发者必须申请GET_ACCOUNTS权限，才能获取设备账户上的信息，而Android N则弃用了该权限，开发者可以在不申请GET_ACCOUNTS的情况下访问设备上的账户信息。

2.新增OPEN_EXTERNAL_DIRECTORY行为。与原先申请WRITE_EXTERNAL_STORAGE权限后即可写外部存储不同的是，在Android N中，开发者首先需要通过该Action在外部存储中申请创建一个可写目录，方可对外部存储进行写操作，这种方式进一步加强外部存储的安全性能。

3. 为了防止利用点击来恶意劫持APP的行为，权限对话框将不会显示用户接口层的内容。

Android7.0（N）的安全更新

系统安全改进

1.引入了新的开机验证机制，在设备开机时首先会验证系统文件完整性，一旦检测到系统文件被篡改（或者是引导镜像存在被修改的痕迹），则禁止系统启动或限定部分功能。

2. 通过限制对私有文件的访问，强化了应用间的隔离效果，进一步保护了应用程序私有数据的安全。

Android7.0（N）的安全更新

应用间文件共享安全

1.应用私有目录被限制访问，文件所有者将无法通过设置私有文件访问模式为MODE_WORLD_READABLE与MODE_WORLD_WRITEABLE来开放私有文件的读写权限；

2.通过file:///URI也将无法访问到应用包之外的路径，若要在应用间共享文件，开发者可以发送一项content://URI，并授权URI的临时访问权限。进行此授权的最简单方式是使用FileProvider类，有效确保了共享文件的安全问题。

3. DownloadManager将不能通过文件名来共享私有文件的信息。旧版应用在访问 COLUMN_LOCAL_FILENAME 时可能出现无法访问的路径。 面向 Android N 或更高版本的应用在尝试访问 COLUMN_LOCAL_FILENAME 时会触发 SecurityException。这里建议大家访问由DownloadManager公开的文件的首选方式是使用ContentResolver.openFileDescriptor()

Android7.0（N）的安全更新

网络安全的改进

1.提升了安卓设备的隐私保护，移除了某些可以持久访问目标设备的标识，例如设备的MAC地址。

2.为了让app更加轻松地控制网络通信数据的访问权限，SDK24以上的系统将不再支持用户使用自签名的证书了。同时所有安装了新版Android系统的设备必须使用相同的证书颁发机构。

3.完善了网络安全配置，开发人员可以更加方便地配置网络安全策略。

Android7.0（N）的安全更新

设备加密机制

1.从Android N开始，所有的新设备必须要提供对密钥存储的硬件支持，且还要在用户使用密钥解锁设备时，提供防止暴力破解的安全保护。这样用户的所有数据只能够在特定的设备上由用户来进行解密。

2.基全新的文件加密机制，系统存储空间和用户配置存储空间将会分开进行加密。这与全盘加密不同，因为全盘加密会将设备中所有的数据一次加密完成。新文件的加密机制可以通过更细粒度地加密来保护每一位用户的数据安全，而且还可以提升被加密文件的独立性。设备中的每一份文件都会使用一个唯一的密钥来加密，而能够解密这些文件的只有你的设备密码。

送给大家一句话

安全不是独行侠而是系统性的运维过程！

谢谢大家，更多精彩经请关注博客或微信公众号