移动互联网安全

第八章移动终端应用攻防与代码安全最佳实践

黄 玮

内容提纲

- 应用组件安全
- 数据安全
- Webview安全
- 其他Android安全编码最佳实践

组件安全

	攻击手段
Activity	* 构造*Intent*直接调用，实现非授权访问 * 后台守护进程通过进程枚举，直接启动新*Activity*覆盖到当前*Activity*进行『点击』劫持，实现钓鱼攻击（零权限要求）
Service	* 构造*Intent*直接调用，实现非授权访问
Broadcast Receiver	* 构造*Intent*直接发送虚假广播消息，实现非授权访问 * 注册同名*IntentFilter*，实现广播消息监听和劫持
Content Provider	* 直接访问暴露的*URI*，实现非授权访问

组件安全——危害

- 恶意调用Activity
- 恶意接收数据（监听/截获敏感数据，例如无序广播）
- 仿冒应用，例如（恶意钓鱼，启动登陆界面）
- 调用组件并接受组件返回数据
- 拦截（有序）广播
- 非授权访问或恶意篡改数据（针对Content Provider接口的SQL注入）

组件安全加固

- 最小化组件暴露
- 设置组件访问权限
- 暴露组件对应代码的内部运行时检查

组件安全加固——最小化组件暴露

- 不参与跨应用调用的组件添加android:exported="false"属性

```
9084        <receiver
9085            android:name="com.alipay.android.app.LiveConnectReceiver"
9086            android:exported="false"
9087            >
9088            <intent-filter
9089                >
9090                <action
9091                    android:name="com.alipay.android.app.pay.ACTION_CREATE_LIVE_CONNECT"
9092                    >
9093                </action>
9094            </intent-filter>
9095        </receiver>
```

```
40        <service
41            android:name="com.taobao.tao.pay.PayService"
42            android:exported="false"
43            >
44        </service>
45        <service
46            android:name="com.taobao.cache.service.ChocolateCacheService"
47            android:exported="false"
48            >
49            <intent-filter
50                >
51                <action
52                    android:name="com.taobao.cache.IMultiCacheService"
53                    >
54                </action>
55            </intent-filter>
56        </service>
```

```
<activity
    android:theme="@7F0F0005"
    android:name="com.taobao.open.oauth.OauthActivity"
    android:exported="false"
    android:launchMode="1"
    android:screenOrientation="1"
    android:configChanges="0x000000A0"
    >
</activity>
```

组件安全加固——设置组件访问权限

参与跨应用调用的组件或者公开的广播、服务设置权限

(1) 组件添加android:permission属性

(2) 声明<permission>属性

(3) 调用组件者声明<uses-permission>

```
<permission
    android:name="com.tencent.qqhead.permission.getheadresp"
    android:protectionLevel="0x00000002"
    >
</permission>
```

```
<uses-permission
    android:name="com.tencent.qqhead.permission.getheadresp"
    >
</uses-permission>
```

public static final int FLAG_COSTS_MONEY

Flag for flags, corresponding to costsMoney value of permissionFlags.

Constant Value: 1 (0x00000001)

public static final int PROTECTION_DANGEROUS

Dangerous value for protectionLevel, corresponding to the dangerous value of protectionLevel.

Constant Value: 1 (0x00000001)

public static final int PROTECTION_NORMAL

A normal application value for protectionLevel, corresponding to the normal value of protectionLevel.

Constant Value: 0 (0x00000000)

public static final int PROTECTION_SIGNATURE

System-level value for protectionLevel, corresponding to the signature value of protectionLevel.

Constant Value: 2 (0x00000002)

public static final int PROTECTION_SIGNATURE_OR_SYSTEM

System-level value for protectionLevel, corresponding to the signatureOrSystem value of protectionLevel.

Constant Value: 3 (0x00000003)

```
<service
    android:name="com.taobao.android.sso.internal.AlipayAuthenticationService"
    android:permission="android.permission.ACCOUNT_MANAGER"
    android:enabled="false"
    android:exported="true"
    >
    <intent-filter
        >
        <action
            android:name="android.accounts.AccountAuthenticator"
            >
        </action>
    </intent-filter>
    <meta-data
        android:name="android.accounts.AccountAuthenticator"
        android:resource="@7F060001"
        >
    </meta-data>
    <meta-data
        android:name="android.accounts.AccountAuthenticator.customTokens"
        android:value="true"
        >
    </meta-data>
    <meta-data
        android:name="com.taobao.android.sso.Version"
        android:value="@7F0D0000"
        >
    </meta-data>
</service>
```

暴露组件对应代码的内部运行时检查

Android 提供各种 API 来在运行时检查、执行、授予和撤销权限。这些 API 是 android.content.Context 类的一部分，这个类提供有关应用程序环境的全局信息。

```
3 if (context.checkCallingOrSelfPermission("com.test.custempermission")
4         != PackageManager.PERMISSION_GRANTED) {
5             // The Application requires permission to access the
6             // Internet");
7 } else {
8     // OK to access the Internet
9 }
```

内容提纲

- 应用组件安全
- 数据安全
- Webview安全
- 其他Android安全编码最佳实践

数据安全

- 存储
  - 内部存储
  - 外部存储
- 密钥认证
- 传输（通信）
- 边信道信息泄露

外部存储安全

- AndroidManifest.xml中声明以下权限就可以读写外部存储设备上的任意数据

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE">
```

- 相关漏洞实例
  - 小米MIUI系统造成用户大量敏感数据泄露
  - 手机QQ 2012（Android）3.0可导致用户聊天信息泄露
  - 印象笔记客户端设计缺陷可导致用户信息泄露
  - 人人网Android客户端可能导致劫持或恶意软件安装
    - 从不可信位置安装软件未做文件真实性和完整性检查

内部存储安全

- 应用使用内部存储时，数据默认是只能被当前应用访问（基于文件访问权限设置）
- 一旦系统被root或代码编写错误，受保护数据将被其他程序访问到

```
try {
    FileOutputStream fos = openFileOutput("config.txt", MODE_PRIVATE);
    fos.write("hello world".getBytes());
    fos.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

```
root@generic_x86:/data/data/edu.cuc.cs/files # ll
-rw-rw---- u0_a65   u0_a65          11 2014-12-07 23:42 config.txt
root@generic_x86:/data/data/edu.cuc.cs/files # cat config.txt
hello worldroot@generic_x86:/data/data/edu.cuc.cs/files #
```


ref: http://developer.android.com/guide/topics/data/data-storage.html#filesInternal

密钥认证

- 将应用中使用的密钥存储在设备的硬件支持的密钥库
  - 仅少部分运行 Android 7.0（API 级别 24）的设备支持硬件级密钥认证；其他所有运行 Android 7.0 的设备则使用软件级密钥认证
  - 如果设备支持硬件级密钥认证，将使用认证根密钥签署此链中的根证书，设备制造商已在出厂时将根密钥注入到设备的硬件支持的密钥库中

传输（通信）安全

- 软件与软件
  - Intent通信机制安全
    - Broadcast Receiver
- 软件与网络（服务器）
  - SSL加密通信
  - 验证服务器证书
    - （基于系统内置授权CA证书，）检查服务端证书合法性
  - 验证主机名
    - 检查服务器端证书配置（有效时间、CN等）

ref: https://developer.android.com/training/articles/security-ssl.html

验证服务器证书的常见问题

- 颁发服务器证书的 CA 未知
  - 从 Android 4.2 (Jelly Bean) 开始，Android 目前包含在每个版本中更新的 100 多个 CA
- 服务器证书不是由 CA 签署的，而是自签署
  - 可以创建自己的 TrustManager，这次直接信任服务器证书
- 服务器配置缺少中间 CA
  - Android 等操作系统通常仅直接信任根 CA，这会在服务器证书（由中间 CA 签署）与证书验证程序（了解根 CA）之间留下一个小的信任缺口

验证主机名的常见问题

- 证书签发或服务器端配置错误
  - java.io.IOException: Hostname 'example.com' was not verified
- 虚拟托管
  - 当多个使用 HTTP 的主机名共享服务器时，网络服务器可以通过HTTP/1.1 请求识别客户端正在寻找哪个目标主机名。遗憾的是，使用 HTTPS 会使情况变得复杂，因为服务器必须在看到 HTTP 请求前知道返回哪个证书。为了解决此问题，较新的 SSL 版本（特别是 TLSv.1.0 及更高版本）支持服务器名称指示 (SNI)，后者允许 SSL 客户端（主动发起）向服务器指定预期的主机名，以便可以返回正确的证书。
  - 自 Android 2.3 开始，HttpsURLConnection 就支持 SNI
    - openssl s_client -servername fendou.us -connect huangwei.me:443

其他注意事项

- 直接使用SSLSocket不会执行主机名验证，需要执行自己的主机名验证
- 把某些证书甚至整个CA列入黑名单，处理可能的CA被攻陷导致的信息泄漏和证书滥签
- 使用客户端证书进行双向身份认证

网络通信安全进一步加固

- 自定义信任锚：针对应用的安全连接自定义哪些证书颁发机构(CA)值得信任。例如，信任特定的自签署证书或限制应用信任的公共CA集
- 仅调试重写：在应用中以安全方式调试安全连接，而不会增加已安装用户的风险。
- 证书固定（白名单），将应用的安全连接限制为特定的证书（不再依赖于系统内置CA的验证机制）
- 明文通信选择退出：防止应用意外使用明文通信

https://developer.android.com/training/articles/security-config.html

使用 Nogotofail 主动测试自己的应用程序

- 查找 TLS/SSL 配置错误和漏洞
- 验证修复并监测回归
- 了解哪些应用和设备正在生成哪些流量

边信道信息泄露

- 日志
- 系统剪贴板信息
- URL缓存
- 浏览器Cookie对象
- 第三方统计数据

内容提纲

- 应用组件安全
- 数据安全
- Webview安全
- 其他Android安全编码最佳实践

WebView安全

- addJavascriptInterface API相关漏洞利用
  - CVE-2012-6636/CVE-2013-4710/CVE-2014-1939/ CVE-2014-7224
  - 远程代码执行漏洞实例索引
  - 远程执行任意Java对象方法，可被用于网页挂马，进而实现系统级别的入侵事件
- Webkit内核漏洞利用UXSS
  - 绕过系统内置浏览器（Webkit内核）的同源性访问本地文件或非当前域的任意cookie等

ref: http://developer.android.com/guide/webapps/webview.html

WebView安全

- 漏洞实例
  - 58同城app远程代码执行
  - 迅雷APP远程代码执行漏洞
- 漏洞检测
  - Android Webview挂马漏洞在线检测
  - addjsif漏洞在线检测
  - UXSS漏洞在线检测

WebView安全加固

- addJavascriptInterface API相关漏洞修补
  - 如果无需与JS交互，禁止调用addJavascriptInterface方法
  - 在载入页面时对URL进行白名单判定，只有存在白名单中的域才允许导出或调用相关的Java类或方法
  - Android 4.2及以后版本，使用新增的@JavascriptInterface显式声明需要暴露给JS访问的Java对象方法

ref: http://drops.wooyun.org/papers/548

ref: http://blog.csdn.net/leehong2005/article/details/11808557

WebView安全加固

- Webkit内核漏洞利用UXSS修补
  - 此问题属于android webkit的漏洞，请尽量使用最新版的android系统
  - 服务端禁止iframe嵌套X-FRAME-OPTIONS:DENY。详见：http://drops.wooyun.org/papers/104
  - 客户端使用setAllowFileAccess(false)方法禁止webview访问本地域。详见：setAllowFileAccess(boolean)
  - 客户端使用onPageStarted (WebView view, String url, Bitmap favicon)方法在跳转前进行跨域判断
  - 客户端对iframe object标签属性进行过滤

ref: http://drops.wooyun.org/tools/3186

内容提纲

- 应用组件安全
- 数据安全
- Webview安全
- 其他Android安全编码最佳实践

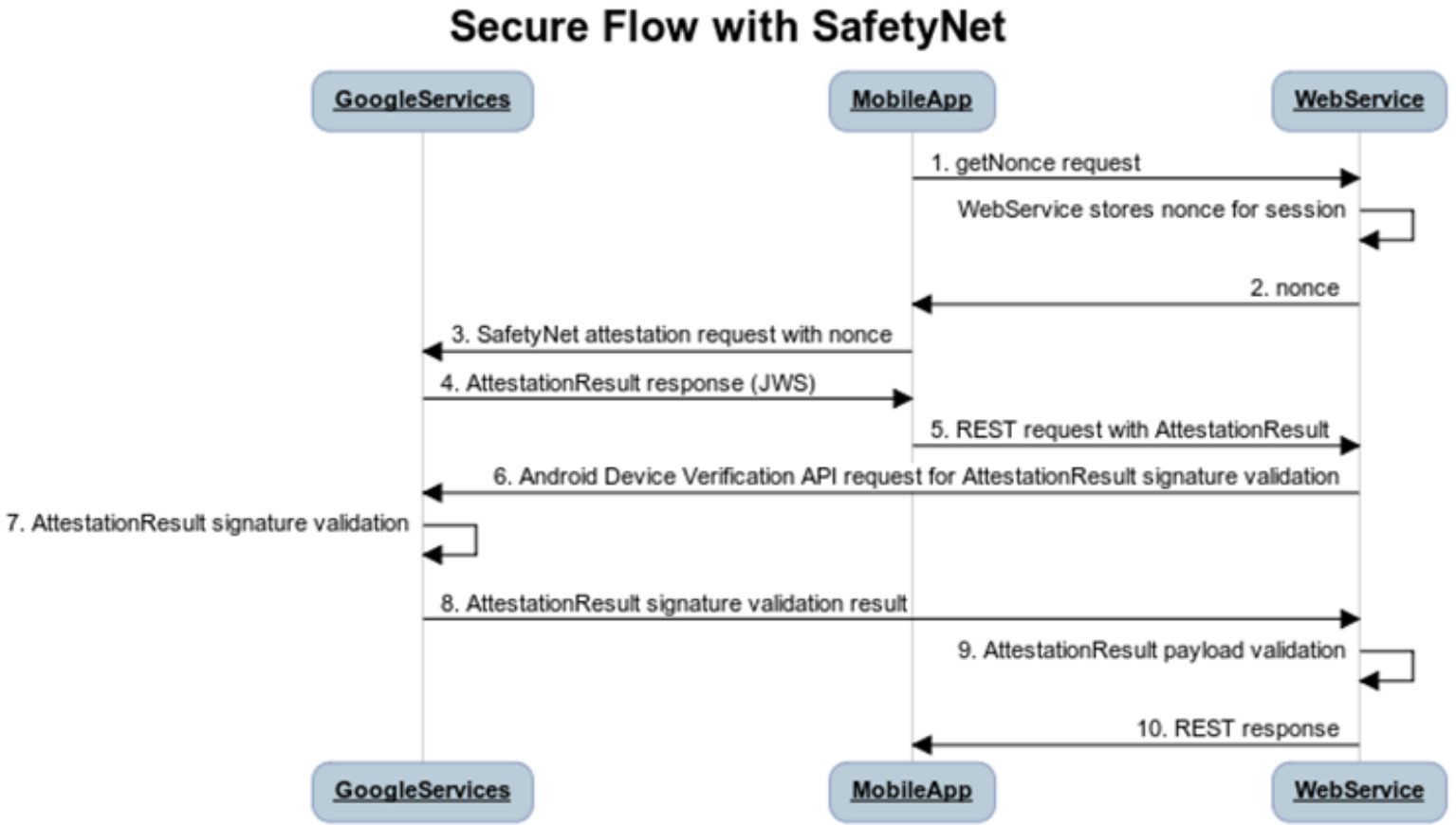
其他Android安全编码最佳实践

- 使用SafetyNet系列API
- 动态加载代码
- 避免使用监听localhost的IP通信来交换隐私和重要数据
  - 设备上所有应用都可以访问localhost
    - 本地暴力破解、服务枚举、数据遍历等风险
  - 优先使用内置认证机制的Android IPC，例如Service
  - 绑定到 INADDR_ANY 比监听localhost还要糟糕，因为这样一来，您的应用可能会收到任何位置发来的请求

https://developer.android.com/training/best-security.html

SafetyNet

- 云端基于Google API，手机端依赖Google Play Service
  - —Google Play Service 2.3+



SafetyNet Attestation API

- 检查Android设备是否可以通过自定义的Android兼容性测试套装（CTS, Compatibility Test Suite）
  - SafetyNet 通过检查设备上的软件和硬件信息来评估其完整性
  - 评估结果是一份使用密码签署的声明，对设备的基本属性——例如总体完整性和与 Android 的兼容性 (CTS)——以及应用的相关元数据（例如其软件包名称和签名）进行证实
  - 可以帮助服务器区分哪些流量来自真实的兼容Android 设备，哪些流量来自不太可信的来源（包括非 Android 设备、已root设备）
  - 如果SafetyNet检测到了系统文件或者用户权限被篡改过，那么它将会阻止系统访问特定的API

使用Safe Browsing API检查URL安全性

- 依赖于Google的Safe Browsing服务
  - 目前，SafeyNet 实现的是Safe Browsing Network Protocol v4
    - 保护用户隐私
    - 保持电池电量
    - 降低网络带宽消耗量

https://developer.android.com/training/safetynet/safebrowsing.html

SafetyNet Verify Apps API

- 检查系统中是否存在潜在恶意软件
- 通常在调用Verify Apps API之前先调用 Attestation API 检测确认系统完整性
  - 通过这个API检查目标系统是否开启了 Verify Apps功能
  - 识别并提示用户：目标系统已安装了某已知恶意软件

使用作用域目录（Scoped Directory）访问

- 常见做法
  - 在manifest文件中请求READ_EXTERNAL_STORAGE或WRITE_EXTERNAL_STORAGE将允许访问外部存储上的所有公共目录，这可能导致访问的内容超出应用需要的内容（不安全）
  - 使用存储访问框架通常会让您的用户通过一个系统UI选取目录，如果应用始终访问同一个外部目录，则该操作没有任何必要（不易用）

Android 7.0提供简化的API来访问常见的外部存储目录

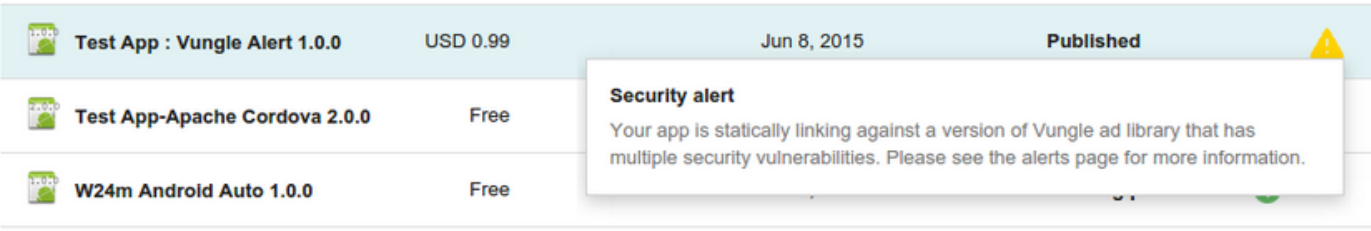
https://developer.android.com/training/articles/scoped-directory-access.html

App Security Improvement Program

- 面向Google Play上的app开发者的安全改进计划
  - —app在被审核通过之前，会接受自动化安全扫描
  - —修复截止日期之后的新提交app如果依然存在已知漏洞则会被禁止上架

Table 1: Warning campaigns with associated deadline for remediation.

Campaign	Started	Remediation Deadline	Support Page
Path Traversal	9/22/2017	1/17/2018	Support page
Insecure Hostname Verification	11/29/2016	3/01/2017	Support page
Fragment Injection	11/29/2016	3/01/2017	Support page
Supersonic Ad SDK	9/28/2016	1/26/2017	Support page
Libpng	6/16/2016	9/17/2016	Support page
Libjpeg-turbo	6/16/2016	9/17/2016	Support page
Vpon Ad SDK	6/16/2016	9/17/2016	Support page
Airpush Ad SDK			
MoPub Ad SDK			
OpenSSL ("logjam" and CVE-2015-3194, CVE-2014-0224)			
TrustManager			



ref: https://developer.android.google.cn/google/play/asi.html