Android 应用安全开发规范

序号	版本号	修订日期	修订概述	修订人	审核人	备注
1	V1.0	2012-04-20	初稿	白开心		
2	V1.1	2013-03-20	添加 webview 安全	白开心		
3	V1.2	2013-06-24	补充 https 服务端证书验证	白开心		

1. Android 安全模型概述

Android 系统主要通过 Linux 系统的内核安全、安全的进程间通信、托管的应用沙盒环境、应用签名、应用定义－用户授权等安全机制来保护用户数据、系统资源，并实现应用之间的隔离。

1.1 系统层安全模型

Android 系统是一个权限分离的系统，它在底层依赖 Linux 系统内核进行权限控制。Linux 内核为 Android 主要提供了基于用户的权限模型、进程隔离、可扩展的安全的进程间通信、移除内核中无用或者潜在危险模块等功能。

Android 系统中的每个应用在安装时都被分配一个唯一的 user ID（Linux user ID），应用以该 user ID 对应的用户身份运行在独立的进程空间中。这个基于用户的权限控制模型在内核层建立了的应用的沙盒环境，在默认情况下，应用运行时拥有自己的虚拟机、文件、数据库等资源，并且只能访问自己的资源，没有权限执行不利于系统、用户和其他应用的相关操作（比如读写联系人、其他应用的文件，访问网络等）。

在 Android 整体框架及软件栈的结构中，内核层之上的应用框架、android 运行时及系统库函数都运行在托管的沙盒环境，如图 1.1 所示。

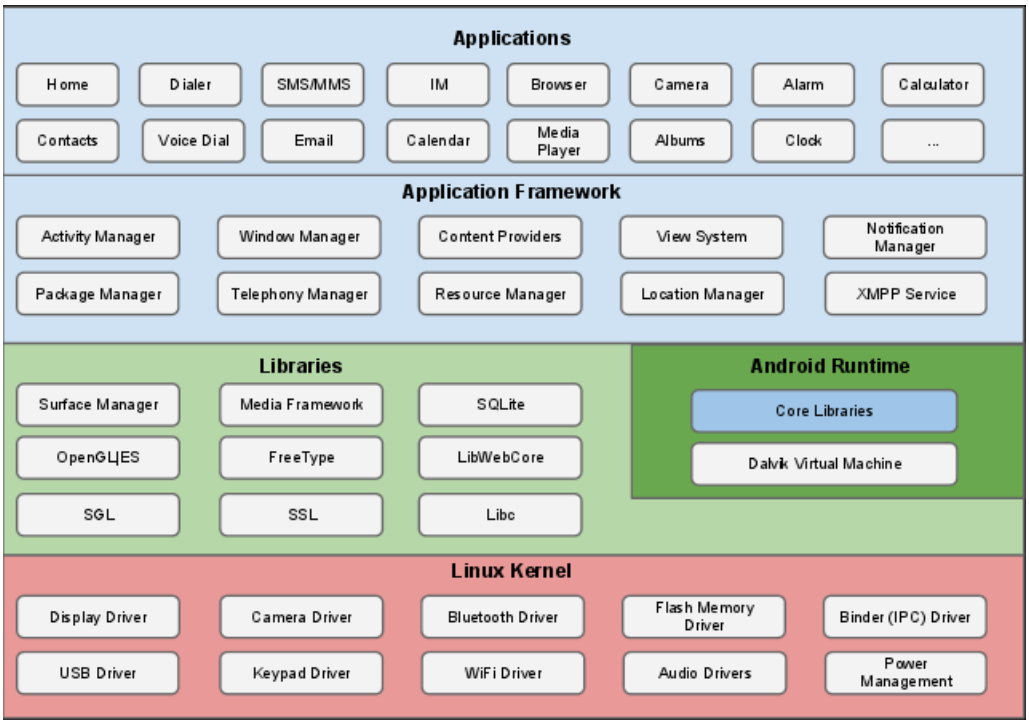

图 1.1 android 软件栈

Android 系统本身还具备文件系统加密、手机设备访问的密码保护、更安全的内存管理（ASLR、基于硬件的堆、栈代码执行阻断）等特性功能。

1.2 应用层安全模型

由于 Android 通过沙盒机制、受限的 API 调用来限制应用对系统资源和其他应用的访问权限，所以每个应用必须在应用的清单文件（AndroidManifest.xml）中提前定义好运行时所需要的权限。当用户安装应用时，系统通过弹出一个列举了该应用运行所需要的权限清单的对话框来向用户请求授权或拒绝安装，这就是 android 的“开发定义－用户授权”的应用层权限控制模型。另外，应用也可以在 AndroidManifest.xml 中对自己的开放组件进行访问权限控制，限制其它应用的非授权访问。

应用层的另外一个安全机制是应用的数字签名，Android 系统要求安装到系统的每一个应用程序都是经过数字证书签名的。数字签名用来标识应用程序的作者，并在应用程序之间建立信任关系。签名用的数字证书并不一定需要权威的 CA 机构认证，它可以是自签名的证书。Android 应用的签名主要有以下作用。

1) 用于程序升级。只有当新版程序和旧版程序具有相同的数字签名时，Android 系统才对该应用进行更新安装，并且不再弹出“应用的权限列表对话框”来进行授权申请。Android 系统禁止更新安装签名不一致的 APK。
2) 通过“signature”或“signatureOrSystem”级别的权限控制，在具有相同签名的应用程序之间安全的共享数据和代码。
3) 用于程序的模块化设计和开发。Android 系统允许拥有相同数字签名的程序运行在同一个进程中。

2. 应用组件安全

Android 应用主要由 Activity、BroadcastReceiver、Service 和 Content Provider 四类基本组件构成，前三类组件间的通信（Inter-Component Communication，组件间的通信，简称 ICC）通过 Intent 消息完成。

Intent 是一个抽象封装了接收组件名、要执行的动作、数据的消息，intent 根据是否指定了接收组件的名称分为显式的 intent 和隐式的 intent。

1) 显式的 intent 中明确指出了消息的接收组件。
2) 隐式的 intent 则是指发送该消息的组件只定义了消息的类型、动作、标记和携带的数据等信息，而对消息的接收组件名称并没有明确指定。由于隐式的 intent 中不包含接收组件的名称，因此开放的接收组件需要向系统注册 intent filter，向系统声明它们所能够接收、处理的 intent 的类型。

导出组件（或者叫公共组件）是指应用组件的“EXPORTED”标记为 true，或者该组件包含至少一个 intent filter。

如果在应用开发中没有对导出的公共组件做好访问权限控制、没有对组件间的通信 intent 做合法性检查或者在 intent 中携带有敏感数据，那么 android 的消息通信本身也可能成为受攻击平面。

2.1 ICC 安全风险

2.1.1 发送隐式 intent 的风险

当一个正常的应用组件发送隐式的 intent 时，如果没有对接收组件的权限进行控制（安全的权限级别为 signature 和 signatureOrSystem），那么恶意应用的导出组件也能定义相应的 intent filter（除了系统 Action 外，可以包含 Intent 中所有的 Action、Data、Category）来接收，甚至截获、篡改该 intent。

(一) 广播窃听、截获

当正常应用发送一个公共的广播时（指广播是隐式的 intent 且没有 signature 或 signatureOrSystem 级别的权限保护），恶意应用可以注册更高优先级的 intent filter 来接收广播，如果广播是由 sendOrderedBroadcast 发出，则还可能会造成广播被截获、篡改。因此广播消息可能遭受的攻击有两种：（1）被动的监听，窃取广播消息；（2）主动的拦截，甚至篡改广播消息，导致拒绝服务攻击和伪造攻击。如图 2.1 所示。

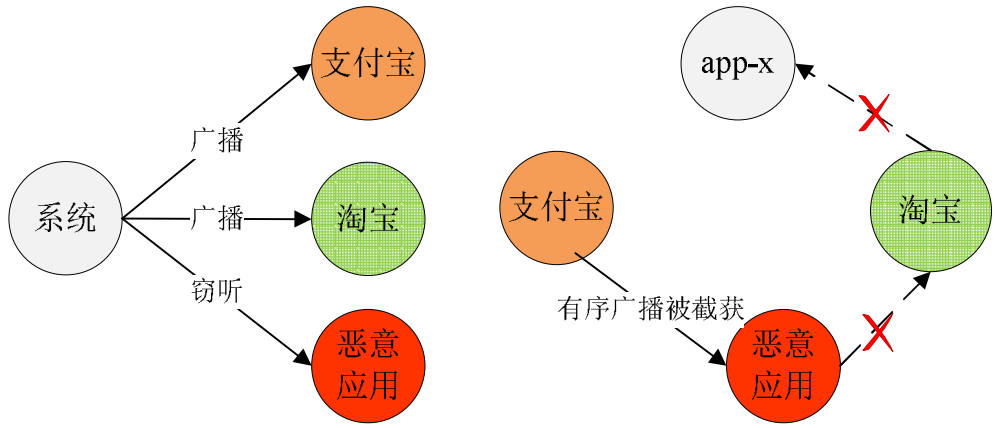

图 2.1 广播监听和截获

(二) Activity 劫持

如果本该由正常的 activity 接收处理的隐式的 intent 消息被一个恶意应用的 activity 接收到，除了会造成 intent 消息泄露外，还可能导致正常用户被恶意 activity 钓鱼攻击，或者返回恶意数据给发送 intent 的组件。

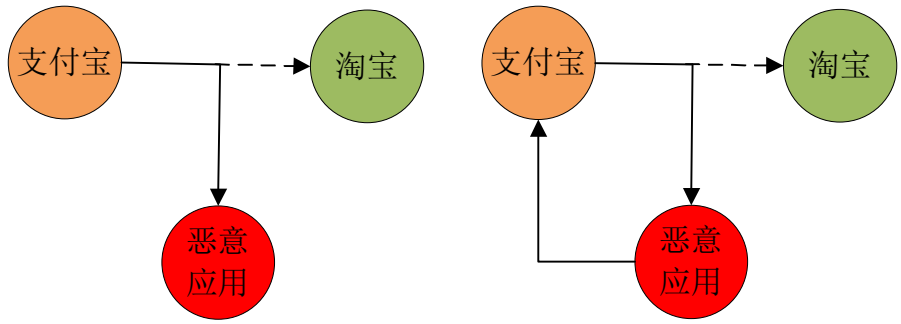

图 2.2 Activity 劫持示例

图 2.2 以支付宝的应用组件发送隐式 intent 通过 android 系统来调用淘宝应用的开放的 Activity 组件为例，展示了恶意应用进行 activity 劫持的方式。图 2.2 左边，恶意应用的 activity 向系统声明具有接收支付宝发出的隐式 intent 的能力，

被系统启动后进行钓鱼攻击。图 2.2 右边，支付宝的应用组件发送隐式 intent 来调用淘宝应用的开放的 Activity 组件，并且需要返回结果时，这时恶意应用不仅可以劫持淘宝的 activity，而且能够向返回的结果中注入恶意数据来欺骗支付宝的应用组件。

(三) Service 劫持

Service 劫持同 activity 劫持类似，是指恶意应用的服务截获了一个发送给正常服务的隐式 intent，这样恶意应用的服务就可以作恶意操作，并可能向调用组件返回欺骗的响应或者恶意数据。Service 劫持比 activity 劫持更隐蔽的一点是：由于 service 没有界面，当 intent 劫持发生时，系统不会弹出对话框给用户进行选择目的 service 接收者。

(四) 特殊的 intent 的使用风险

Android 应用进程之间的数据共享和访问由 Content Provider 和 Content Resolver 来完成，数据在 Content Provider 中的存储地址用 Content URI 来标识。

Intent 自身可以携带指向数据地址的 URI，如果消息发送者在 intent 中设置了 FLAG_GRANT_READ_URI_PERMISSION 或 FLAG_GRANT_WRITE_URI_PERMISSION 标志，并且拥有该 URI 指向数据的 Content Provider 在 manifest 文件里定义了该 URI 的访问权限，那么截获到这种 intent 的恶意程序最终也拥有访问该 URI 指向的数据的权限，这样便导致了数据的泄露或篡改。

另外一类特殊的 intent 是 Pending Intent，它是由一个应用创建并传递给其它应用使用。Pending Intent 保留了创建该 intent 的组件的所有权限和特权，而接收到该 pending intent 的应用还能把它发送给其它应用，这时该 intent 仍携带创建者的权限。如果恶意程序获取到了这个 pending intent，那么这个 intent 创建者的权限就会被滥用。

2.1.2 接收 intent 的风险

如果应用的导出组件（exported component）没有对调用它的组件作权限的控制，或者没有对接收到的 intent 消息进行合法性检查，则可能会接收到恶意的 intent，造成应用崩溃或导致其他的恶意利用。图 2.3 展示了恶意应用发送一个恶意消息给支付宝的某个导出组件进行 intent spoofing，但是支付宝的这个导出组件并没期望去接收这样的一个 Intent。

图 2.3 支付宝的导出组件接收到恶意的 intent

(一) 恶意广播注入

如果导出的 Broadcast Receiver 组件盲目地相信进入的广播 intent，可能会执行不恰当的响应动作或者对 intent 中的恶意数据进行操作。由于广播接收器通常会传递命令或数据给 Services 和 Activities，如果 receiver 遭到恶意数据注入，那么恶意的 intent 就能传播到整个应用程序。

如果应用中注册了一个用来接收系统广播的 Broadcast Receiver，该广播接收

器将变成可公开访问。虽然包含系统动作（如android.intent.action.BOOT_COMPLETED）的intent只能由android系统广播发出，但是在这种情况下，恶意程序可以通过直接发送显式的intent来调用该receiver，所以receiver中需要检查下intent的数据、发送者、action等，对于重要的receiver添加权限保护。

(二) 恶意启动 Activity

如果导出的activity组件没有对调用者身份校验、或者没有对自身进行访问权限控制（主要指signature或signatureOrSystem权限），可能会被恶意应用发出的显式或隐式intent启动，这会带来以下风险：

1) 影响应用的正常状态，如果activity使用了intent中携带的恶意数据，还可能使数据存储受破坏；
2) 如果activity会返回数据给调用者，可能会造成数据泄露。

(三) 恶意启动 Service

类似于恶意启动activity，如果导出的service组件没有做好身份校验和权限控制，也会被其他任意应用启动或绑定，造成数据泄露或执行未授权的任务。

2.2 ICC 基本规范

1) 应用组件之间的通信（ICC）尽量使用显式intent；intent中不要携带敏感数据（如密码、token、卡号等）。
2) 从intent 接收者的角度看，intent Filter并不是一个安全边界。开发者应该意识到：声明一个intent filter时会将该组件导出，并暴露在攻击之下。因此为了组件更安全，应该避免导出组件，除非这个组件是专门设计用来处理外部程序请求或者接收系统消息的，同时关键的逻辑或状态转换动作不要放在导出组件里。
3) 如果一个组件必须要导出(如接收系统广播)，只有指定发送 intent 给它的组件所需要的权限才能保证安全（signature或signatureOrSystem才被认为是安全的权限级别），并且在处理接收到的消息之前，需要对接收到的 intent（包括发送者身份、动作、数据等）做合法性检查，看是否是期望接收到的消息。
4) 内部使用的组件可设置为私有，<android:exported="false">。
5) 对于关键广播的发送组件，需要明确指定广播接收者应该具有的接收权限（signature或signatureOrSystem）。对于广播接收者，可以定义发送消息给它的发送者必须具有的发送权限。避免使用sticky broadcast。
6) 为防止重要数据泄露，应用的Content Provider必须在manifest文件中指定read权限。

2.3 Webview 安全

http://developer.android.com/guide/webapps/webview.html#AddingWebView

Android应用可以使用webview组件来展示web页面。Android应用可以在应用代码中利用 addJavascriptInterface 接口给javascript代码调用。webview中可以启用javascript，调用本地暴露的接口。

```
WebView myWebView = (WebView) findViewById(R.id.webview);
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
webView.addJavascriptInterface(new WebAppInterface(this), "JsObject");

class JsObject {
    @JavascriptInterface
    public String toString() { return "injectedObject"; }
}
```

Html 页面调用接口的示例如下。

```
<input type="button" value="Say hello" onClick="showString()" />
<script type="text/javascript">
      function showString() {
            JsObject.toString();
      }
</script>
```

2.3.1 webview 安全风险

使用 addJavascriptInterface 接口允许 js 脚本控制 android 应用，如果嵌入的 html 页面的部分内容可由用户控制，恶意攻击者就会在 html 页面嵌入能够执行本地应用的恶意代码

2.3.2 安全要求

除非 Webview 中嵌入的 html 页面和 javascript 脚本完全由开发者提供和控制，否则不要使用 addJavascriptInterface 接口。

针对 Android 4.2 系统及以上版本的系统，使用@JavascriptInterface 标注来代替 addjavascriptInterface 方法

不要允许用户跳转到其他域的页面。

3. 防敏感数据泄露

本章提到的敏感数据，包括定义、录入、传输、展示等，请参考《SofaMVC 安全编码规范》第 5 章。

http://doc.alipay.net/pages/viewpage.action?pageId=11255676。

3.1 数据的本地存储

Android 应用开发中可以利用 Shared Preferences、File、SQLite 提供的 API

在设备内部、外部存储（SD 卡或其他应用共享的存储）或应用目录下创建并访问文件、数据库。

3.1.1 文件访问控制

GetSharedPreferences、openFileOutput 、openOrCreateDatabase 等方法的参数中通常包含一个 int 型的文件创建模式－mode，它的取值主要有以下几种

1) MODE_PRIVATE：应用程序私有。
2) MODE_WORLD_READABLE：所有应用可读。
3) MODE_WORLD_WRITEABLE：所有应用可写。

建议采用 MODE_PRIVATE，避免其他应用能够访问。

如果文件或数据库表中需要保存敏感数据，必须进行强加密存储或者存储 hash 值，加密密钥不能同时存储在移动设备中。应用卸载时，必须删除这些机密数据。

3.1.2 防 SQL 注入

(一) SQLite sql 注入风险

Android 使用轻量级的 SQLite 作为关系型数据库，如果采用字符串拼接用户输入参数的方式构造 SQL 语句进行数据库查询，，就会存在 sql 注入风险，可能造成数据泄露、篡改或删除。

例如根据某个表的主键 id 用拼接参数的方式来查询数据，

″SELECT * FROM table_name WHERE id = ’″ + userId +″’″。

如果 userId 是获取的用户输入，那么它可能是非法数据，在查询时导致数据泄露。比如用户输入：1’ or id <>‘（或者 1’ or ‘1’ =’ 1），则整个语句就会变成：″SELECT * FROM table_name WHERE id = ‘1’ or id <> ‘’ ″，该 sql 语句执行后就会把整个表的数据读取出来，如图 3.1 所示。

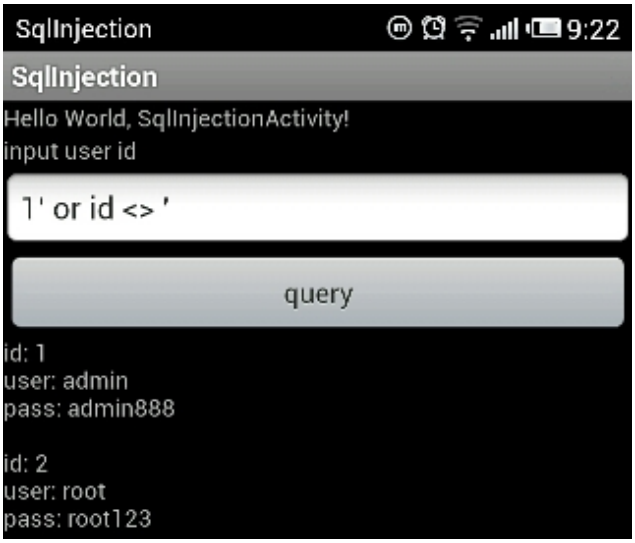

图 3.1 SQLite sql 注入示例

(二) SQL 注入防范

防御 sql 注入的方法是采用参数化的查询方式，而不是拼接字符串，以图

3.1 中的示例来说，正确的代码写法应该为：

```
String userId = userId_EditText.getText().toString();
DatabaseHelper dbHelper = new DatabaseHelper(this,DB_NAME,null,VERSION);
SQLiteDatabase db = dbHelper.getWritableDatabase();
String args[] = {userId};
Cursor cursor = db.rawQuery("SELECT * FROM usertable WHERE id = ?", args);
```

或者采用 public Cursor query (SQLiteDatabase db, String[] projectionIn, String selection, String[] selectionArgs, String groupBy, String having, String sortOrder)方法查询，代码如下。

```
String columns [] = { "id", "username", "password" };
String selectionClause = "id = ?";
String [] selectionArgs = { "" };
selectionArgs[0] = userId;
Cursor cursor = db.query("usertable", columns, selectionClause, selectionArgs, null, null, null);
```

此外，还可以对用户输入的数据进行类型（是否是期望的数据类型）和格式检查（对不合法字符进行过虑，比如“’、<、>、--”等）。

3.1.3 webview 缓存密码的设置

应用使用 webview 的默认配置在加载其他应用的登录、认证页面时，系统会提示用户对输入的账号、密码是否保存，如果选择了“是”，密码就会保存在这个应用私有目录的/data/data/app_name/databases/webview.db 中（比如 http://www.wooyun.org/bugs/wooyun-2013-020246）。

为了避免 webview 默认保存用户的密码，开发者在初始化 webview 实例后，请加上一行代码：theWebView.getSettings().setSavePassword(false)，将其设置为不保存密码，就不会提示用户选择保存密码。

3.2 网络传输

为保证数据传输过程中的安全，包括数据的机密性和完整性，客户端与服务端之间的通信采用 https 协议。

SSL/TLS 通信中，客户端通过数字证书判断服务器是否可信，并采用与服务器协商的会话密钥进行加密通信。在有些情况下，开发者使用了自己生成的证书后，客户端发现证书无法与系统可信根 CA 形成信任链，出现了 CertificateException 等异常，这时就会中断检查服务器证书的有效性，或选择接受所有的证书。这种实现可能导致中间人攻击。

比如在钓鱼 WiFi 网络中，攻击者可以通过设置 DNS 欺骗使客户端与指定的服务器进行通信。攻击者在服务器上部署另一个证书，在会话建立阶段，客户端会收到这张证书。如果客户端忽略这个证书的异常，或者接受这个证书，就会成功建立会话、开始加密通信。但攻击者拥有私钥，因此可以解密得到客户端发来

数据的明文。攻击者还可以模拟客户端，与真正的服务器联系，充当中间人做监听。因此在代码实现时，开发者不要自己实现一个 X509TrustManager 接口，同时将其中的 checkServerTrusted()方法实现为空（即不检查服务器是否可信），或者在 SSLSocketFactory 的实例中，通过 setHostnameVerifier(SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER)，接受所有证书。

解决问题的一种方法是从可信 CA 申请一个证书。但在移动软件开发中，不推荐这种方法。除了申请证书的时间成本和经济成本外，这种验证只判断了证书是否 CA 可信的，并没有验证服务器本身是否可信。例如，攻击者可以盗用其他可信证书，或者盗取 CA 私钥为自己颁发虚假证书，这样的攻击事件在过去两年已有多次出现。

事实上，移动软件大多只和固定的服务器通信，因此可以在代码中更精确地直接验证服务端的证书是否是某张特定的证书，这种方法称为“证书锁定”（certificate pinning）。实现证书锁定的方法有两种：一种是实现 X509TrustManager 接口（http://developer.android.com/training/articles/security-ssl.html），另一种则是使用 KeyStore（http://www.thoughtcrime.org/blog/authenticity-is-broken-in-ssl-but-your-app-ha/）。具体可参考 Android 开发文档中 HttpsURLConnection 类的概览说明。

除 https 协议本身提供的身份认证、数据加密、签名的保护机制外，建议对一些重要数据（如用户的密码、交易订单等）进行业务层的数据加密和签名。因为 https 协议本身在各个 OS 实现过程中也可能出现漏洞，比如 iOS4.3.5 曾经存在 SSL 中间人攻击的漏洞，具体可参考下述网址。

https://www.trustwave.com/spiderlabs/advisories/TWSL2011-007.txt

3.3 日志输出

如果应用的安装包在上线时没有关闭日志输出开关，则上线后可能会在日志中输出一些敏感的数据信息，比如与服务端的通信数据（包括用户账号、手机号、订单号、交易金额等）。而任意低权限的应用均可以调用 logcat 来记录系统的 log 信息，很容易造成重要数据泄露。

为防止线上应用在日志中输出敏感数据，可以对 android.util.Log 类进行封装，添加一个日志输出开关，在测试环境下把日志开关设置为 true 进行输出调试用，而在上线时将开关关闭。

关于日志输出可参考《支付宝敏感信息及日志打印规范》。

3.4 数据输入、展示

客户端需要对用户输入的数据进行合法性检查（包括长度、类型、是否包含非法字符等）。

敏感信息录入时使用安全的方式录入（如手机软键盘等），防止信息泄露。对敏感信息的展示采用部分截断展示的方式，比如信用卡号、身份证号等只展示前 6 后 4 位。

关于敏感数据的输入、展示可参考《支付宝敏感信息及日志打印规范》。

4. 防止逆向破解

4.1 安装包逆向破解风险

如果 android 应用在编译时没有做反编译的混淆处理，应用的 apk 包在被恶意攻击者反编译后，可能会暴露数据传输、加密、存储等关键代码或业务逻辑。

apk 包还可能被注入恶意代码后重新编译、打包、签名，如果服务端未对应用的签名进行校验的话，可能会存在恶意的假冒客户端与服务端交互，并对用户进行钓鱼欺诈或资金窃取。

4.2 安装包逆向破解过程

4.2.1 应用编译过程

Android 应用从 java 源码到 apk 安装包，主要经过以下过程，如图 4.1 所示。具体编译过程可参考安全中心撰写的《android 应用安全测试》文档，或者参考 http://developer.android.com/guide/developing/building/index.html。

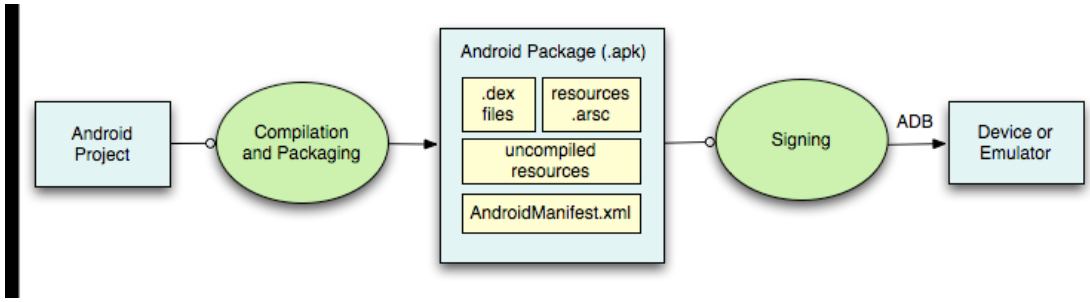

图 4.1 android 应用编译、打包、签名过程

1) java 编译器将所有的.java 源码文件、R.java、.aidl 文件编译成 java 虚拟机能够执行的.class 字节码文件。
2) Dex 工具将上个步骤输出的.class 文件、第 3 方的库文件和.class 文件转化为 Dalvik 虚拟机能够执行的.dex 字节码。
3) Apkbuilder 工具将.dex 字节码、已经编译好的资源文件和其它的资源文件打包成.apk 的安装包。
4) .apk 安装包签名。签名密钥可以是调试密钥或者发布用的签名（eclipse 中调试运行应用时会自动进行签名，应用发布时需要自己生成签名的密钥、keystore 等），签名工具可以用 jarsigner 或 signapk。
5) 如果是对要发布的应用签名后，通常还需要用 zipalign 工具将 apk 包进行地址对齐，减少加载时的内存占用。

4.2.2 安装包的反编译过程

反编译过程刚好与编译过程相反，目的是在缺少源代码的情况下，从安装包查看原来的代码逻辑。反编译过程如图 4.2。

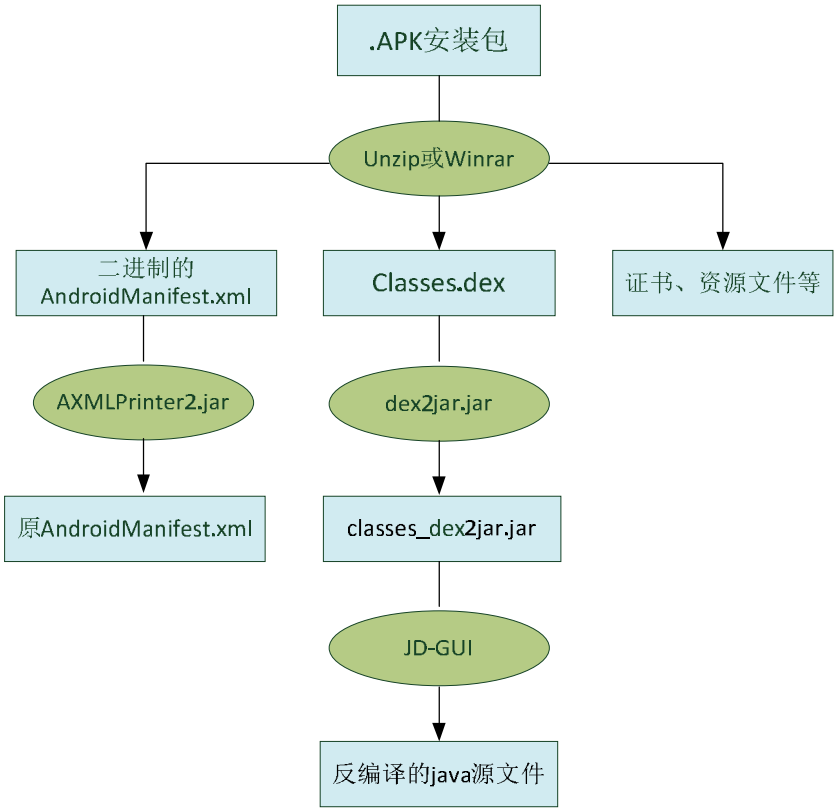

图 4.2 反编译 APK 安装包过程

1) 利用 winrar、winzip 或 unzip 将.APK 的安装包解压，得到解压后的资源文件、二进制的 AndroidManifest.xml、classes.dex 及其它目录。
2) 利用 dex2jar 工具将解压出来的 classes.dex 放到下载 http://code.google.com/p/dex2jar/后的解压目录中，然后在命令行运行 dex2jar classes.dex，就会在目录下生成 classes_dex2jar.jar 文件（包含.class 字节码的包）。
3) 利用 JD-GUI（反编译.class 字节码到源 java 的软件）http://java.decompiler.free.fr/?q=jdgui，用它打开第 2 步生成的 classes_dex2jar.jar 文件，将 jar 包文件转化为反编译后的 java 源代码，如图 4.3 所示。

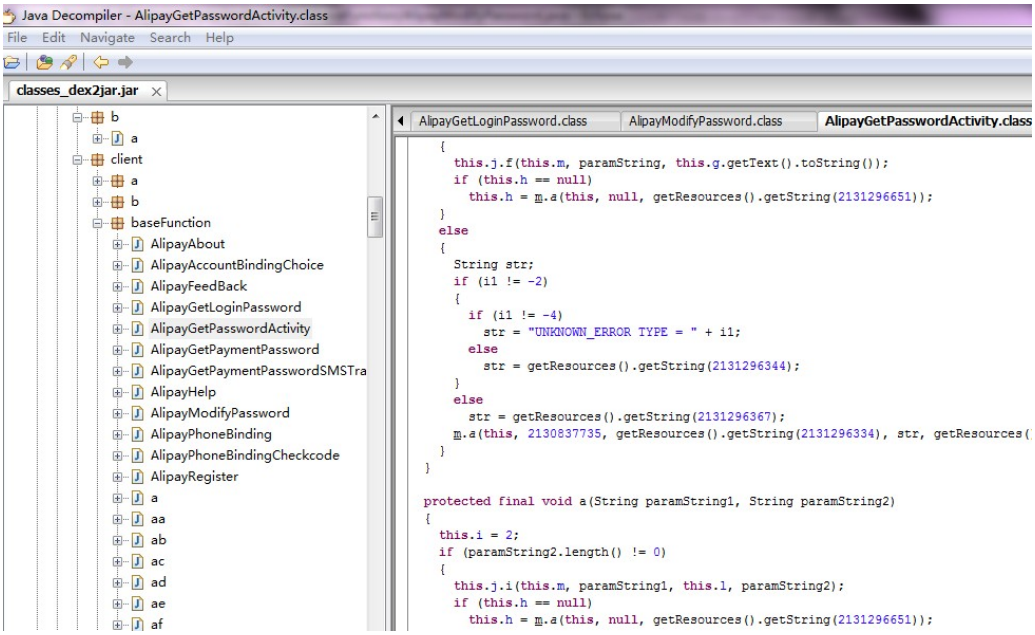

图 4.3 反编译得到的 java 代码

4) 利用 AXMLPrinter2.jar 解码二进制的 AndroidManifest.xml 文件，命令行进入第 1 步的解压目录，输入 java -jar AXMLPrinter2.jar AndroidManifest.xml > mainfest.xml，得到源代码中的 AndroidManifest.xml 文件，解码前后文件对比如图 4.4 所示。

图 4.4 解码前后的 AndroidManifest.xml 文件对比

4.3 代码混淆保护

1) 应用发布前，需要对 Java 开发的应用进行代码混淆后再编译，提高逆向破解的难度。目前可利用 android sdk 在 eclipse 中集成的 proguard 插件，在应用编译前，在 eclipse 工程目录下的 proguard.cfg 文件中添加对核心

代码类的混淆规则（请测试代码混淆后应用的原功能是否正常）。关于proguard 的具体使用方法可参考 http://proguard.sourceforge.net/#manual/introduction.html。

2) 建议对关键业务逻辑的代码用 C 和 JNI 来实现，可以对 so 文件进行加壳，如 UPX；JNI 的 so 库文件必须对调用它的应用进行签名校验，避免核心流程被人直接调用。
3) 对于应用自身的 apk 文件和 so 库文件在执行之前需要校验签名是否正确。

5.参考资料

http://www.claudxiao.net/2013/03/android-webview-cache/