移动应用APP安全开发基线

2020年10月15日

# 目录

[目录 2](#_Toc53674864)

[安全编码规范及基线 4](#_Toc53674865)

[1 概述 4](#_Toc53674866)

[2 安全基线 4](#_Toc53674867)

[3 APP安全现状分析 4](#_Toc53674868)

[4 APP安全开发基线\_Android 4](#_Toc53674869)

[4.1 组件安全 4](#_Toc53674870)

[4.2 WebView安全 7](#_Toc53674871)

[4.3 代码安全 7](#_Toc53674872)

[4.4 程序保护 9](#_Toc53674873)

[4.5 输出安全 12](#_Toc53674874)

[4.6 传输安全 12](#_Toc53674875)

[4.7 存储安全 12](#_Toc53674876)

[4.8 日志安全 13](#_Toc53674877)

[4.9 应用发布安全 14](#_Toc53674878)

[5 APP安全开发基线\_iOS 14](#_Toc53674879)

[5.1 代码安全 14](#_Toc53674880)

[5.2 输出安全 16](#_Toc53674881)

[5.3 程序保护 18](#_Toc53674882)

[5.4 传输安全 18](#_Toc53674883)

[5.5 存储安全 18](#_Toc53674884)

[5.6 日志安全 18](#_Toc53674885)

[5.7 应用发布安全 18](#_Toc53674886)

[6 业务通用安全开发基线 18](#_Toc53674887)

[6.1 用户安全 18](#_Toc53674888)

[6.2 身份与访问控制 19](#_Toc53674889)

[6.3 密码算法安全 21](#_Toc53674890)

[6.4 会话安全 22](#_Toc53674891)

[6.5 会话管理 23](#_Toc53674892)

[6.6 支付安全 23](#_Toc53674893)

[6.7 运行环境安全 24](#_Toc53674894)

[7 移动行业安全现状 25](#_Toc53674895)

# 安全编码规范及基线

## 概述

由于移动应用开发团队通常将精力集中在产品设计、功能实现、用户体验和系统效率等方面，而很少考虑安全问题。为提高移动信息化系统移动应用客户端及服务端的整体安全能力水平，特制定本安全开发指南用于指导开发团队进行安全编码，从安全开发生命周期的早期竖起移动应用的安全保障。

## 安全基线

依据安全编码的要求，并提供实际的修复代码、修复建议及测试方案形成安全基线列表文档。有了安全基线就必须有对应的安全测试人员，依照基线检查内容定期进行安全检查。

## APP安全现状分析

通过对现有多款移动应用进行渗透分析，发现主要风险集中在程序源码保护不到位、APP开发配置不当，用户敏感信息明文存储、协议数据明文传输、开发设计缺陷导致的高危业务逻辑漏洞等。反应了APP开发人员对移动端应用安全性认识薄弱，缺乏安全基本规范的问题。

## 参考规范

l  《信息安全技术移动智能终端个人信息保护技术要求》

l  《YD/T 1438-2006 数字移动台应用层软件功能要求和测试方法》

l  《YD/T 2307-2011 数字移动通信终端通用功能技术要求和测试方法》

l  《YD/T3039-2015移动智能终端应用软件安全技术要求》

l  《YD/T3082-2016移动智能终端上的个人信息保护技术要求》

l  《YD/T3079-2016移动互联网长在线应用系统总体技术要求》

l  《YD/T2846-2015移动互联网安全监测体系架构》

l  《YD/T2848.2-2015移动互联网恶意程序检测方法第2部分:终端侧》

l  《YD/T2848.1-2015移动互联网恶意程序检测方法第1部分:网络侧》

l  《YD/T2847-2015移动互联网恶意程序监测与处置管理平台数据接口规范》

l  《YD/T2694-2014移动互联网联网应用安全防护要求》

l  《YD/T2695-2014移动互联网联网应用安全防护检测要求》

l  《YD/T2743-2014移动终端设备应用程序开放接口技术要求》

## APP安全开发基线\_Android

### 组件安全

1) 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，这个属性说明它是私有的，只有同一个应用程序的组件或带有相同用户ID的应用程序才能启动或绑定该服务。

<Activity

android:name=".LoginActivity"

android:label="@string/app\_name"

android:screenOrientation="portrait"

android:exported="false">

2) Activity导出安全。通常声明的Activity默认是不可导出的，但是在<Activity>标签中包含<intent-filter>时，Activity默认导出。一般应用只在其主页面（程序入口）的Activity标签中包含<intent-filter>，而如果在一些需要登录验证后才能进入的页面的标签中包含了<intent-filter>，则会被其他应用调用，跳过验证直接进入页面

3) 公开的Activities必须声明权限标签

<permission

android:name="权限名"

android:description="详细描述"

android:label="简介"

android:protectionLevel="signature"/>

android:permission="权限名"

android:protectionLevel=”dangerous”

4) Broadcast关键的广播信息只能由本单位私钥签名的应用接收

5) Broadcast公开的广播信息的接收必须声明接收权限

voidsendBroadcast(Intentintent,StringreceiverPermission);可公开给第三方应用程序访访问，可声明为dangerous，只允许本单位的应用程序访问指定signature.

6) ContentProvider

android:grantUriPermission=”false”

android:readPermission=”权限名”

android:writePermission=”权限名”

7) ContentProvider子Uri

grant-uri-permission

android:path=”string”

android:pathPrefix=”string”

android:pathPattern=”string”

8) Activity，Service等组件有些是对外开放的（manifest.xml中export为true或定义的intent-filter的组件是可导出组件），当这些组件传递的参数未做校验时，这些参数可能导致APP拒绝服务。需严格校验输入参数，注意空值判定和类型转换判断，防止由于异常输入导致的应用崩溃

9) 设置组件访问权限。Intent进行权限设置，包括Activity、Content provider、Broadcast receiver、Service等。尽量避免隐式调用，如需隐式调用，需进行权限限制，反界面劫持

10) 限制不同应用件组件访问许可。应用与应用之间的通讯，Intent应明确指定目的组件的名称，防止第三方程序进行中间人攻击

11) 针对contentprovider，可细粒度地为读取数据和写入数据设置不同的权限，对应的manifest标签分别为android:readPermission和android:writePermission

12) 反Activity劫持。Activity劫持程序运行枚举当前运行的进程，发现目标进程，弹出伪装程序，可进行钓鱼攻击、盗号等。解决这个问题需要系统安装能够保护支付安全的软件。常见方案有两种：一是“支付清场，即在支付程序启动时对手机环境进行清场，将未知的恶意软件杀掉，那么恶意程序也就不再能够进行Activity劫持。二是可以利用安全软件高权限的特点，在支付过程中监控Activity劫持行为，实时拦截这种攻击。

13) 如果这个组件只打算给自己开发的其他软件使用，而不希望暴露给第三方软件，在定义权限时，protection Level字段应该选择signature。这种设置要求权限使用者（即调用者）与权限定义者（即被调用者）必须由相同的证书进行签名，因此第三方无法使用该权限，也就无法调用该组件。默认情况下，service、broadcastreceiver和contentprovider是暴露的，申明了Intent-filter的actvity也是暴露的。如果它们只被同一个软件中的代码调用，应该设置为不暴露，在AndroidManifest.xml中为这个组件加上属性android:exported=”false”

14) 尽量避免隐式调用。除了基于Intent类中已有ACTIONs的隐式调用，绝大部分隐式调用都属于这两种情况：同一软件中不同组件的调用；同一开发者不同软件间的调用。这两种情况下，事实上，开发时都已可以确定要调用的组件是哪个。因此可以避免隐式调用，改为基于组件名的显式调用

15) 对于必须隐式调用的，sendBroadcast(Intent,String)，它的第二个参数可以指定接受者需要的权限

16) 为避免Webview组件存在安全漏洞，AndroidManifest.xml中的targetSdkVersion设置大于等于17

17) 导出Activity组件的安全风险其原因在于导出Activity，任何软件都可以调用它，包括攻击者编写的软件，可能产生恶意调用，应用会产生拒绝服务等问题。遇到这样的问题，如果它们只被同一个软件中的代码调用，将Activity属性改为android:exported=“false”，如果组件需要对外暴露，应该通过自定义权限限制对它的调用。

如果组件需要对外暴露，exported值设置为true，则protection-Level应使用normal或dangerous。应该通过自定义权限限制对它的调用。首先，在实现了被调用组件的软件的Android-Manifest.xml中自定义一个权限：

Android：protectionLevel=“signature”

接下来，为被调用组件添加这个权限限制，即在AndroidManifest.xml中为这个组件添加android:permission属性。

另一种方法是在组件的实现代码中使用Context.checkCallingPermission()检查调用者是否拥有这个权限。

最后，要调用这个暴露的组件，调用者所在的软件应该申明使用这个权限，即在AndroidManifest.xml中添加相应的use-permission申明。

18) 检查AndroidManifest.xml文件中各组件定义标签的安全属性是否设置恰当。如果组件无须跨进程交互，则不应设置exported值为true。例如，如下图所示，当MyService的exported属性为true时，将可以被其他应用调用（当有设置权限(permissions)时，需要再考察权限属性。如android:protectionLevel为signature或signatureOrSystem时，只有相同签名的APK才能获取权限。）

关于AndroidSDK中对exported属性的默认设置说明：对service、Activity、receiver，当没有指定exported属性时，没有过滤器（IntentFilter）则该服务只能在应用程序内部使用，相当于exported设置为false。如果至少包含了一个过滤器，则意味着该服务可以给外部的其他应用提供服务，相当于exported为true。对provider，SDK小于等于16时，默认exported为true，大于16时，默认为false。（某些广播如android.intent.action.BOOT\_COMPLETED是例外）

### WebView安全

1. 使用Webview时需要关闭Webview的自动保存密码功能，防止用户密码被Webview明文存储。设置Webview.getSettings().setSavePassword(false)即可
2. 禁止Webview明文存储
3. 禁止Webview远程代码执行，防止被远程控制。在特定版本的安卓系统上，中间人可以利用Webview的漏洞执行任意代码

### 代码安全

1. 关键业务逻辑代码c+jni
2. 对so文件加壳UPX
3. so对调用它的应用进行签名校验
4. 进行异常处理
5. 禁止使用硬编码定义敏感内容
6. 对应用程序调用的所有第三方代码和库进行验证（确保来源是否可靠、是否有支持维护、是否不含后门等）
7. SharedPreference禁止使用MODE\_WORLD\_READABLE和MODE\_WORLD\_WRITABLE，以防存在内容被替换的风险
8. File模式安全配置1.避免openFileOutput使用MODE\_WORLD\_WRITEABLE和MODE\_WORLD\_READABLE模式创建进程间通信的内部存储(InternalStorage)文件，以防存在内容被替换的风险。如果需要与其他进程应用进行数据共享，请考虑使用contentprovider，详情参照Google官方安全指导。2.避免滥用“android:sharedUserId”属性；3.避免将密码等敏感数据信息明文存储在内部存储(InternalStorage)文件中
9. 使用Webview时需要关闭Webview的自动保存密码功能，防止用户密码被Webview明文存储设置

Webview.getSettings().setSavePassword(false)

1. 使用SecureRandom时不要使用SecureRandom(byte[]seed)这个构造函数，会造成生成的随机数不随机。建议通过/dev/urandom或者/dev/random获取的熵值来初始化伪随机数生成器PRNG
2. 避免代码逻辑缺陷导致的跨界，如Activity劫持
3. 采用NDK开发核心模块
4. 使用多种技术进行代码保护，防重放、防篡改、防劫持
5. 开发安全的接口，如通过避免语句的完全解释或采用参数化接口等方式实现
6. 数据库应使用存储过程或参数化查询，并严格定义数据库用户的角色和权限
7. 实现serializable接口时需定制序列化协议。应实现自己的wirteObject和readObject函数，提供特定逻辑对敏感数据进行保护，通过加密算法或哈希算法，避免可能的伪造或重放攻击。或通过将敏感字段，标记为transient来回避序列化操作
8. 执行数据库查询前设置查询类型。设置查询类型为只读，避免潜在的危险读写查询操作
9. APP插件调用第三方代码或库文件来源的可靠性，验证第三方接口或组件的合法性，确保随时跟进第三方库的更新
10. 避免在public方法中返回private内部数据。在public方法中，如果直接返回数组等内部数据，会导致其private修饰符失效，刻意使用者刻意通过此内容修改内部数据状态
11. 避免危险的publicstaticfinal数组声明。publicstaticfinal不能确保数组的内容不会被修改，此类数组应该被声明为private避免访问
12. 定义public和static字段为final
13. 去掉开发时的调试代码，关闭调试开关，删除多余的Log代码
14. 确保锁在发生异常时候时被正确释放。如果在try代码块中获取了一个锁，必须在finaliy块中对其进行释放，避免死锁
15. 避免在获取锁后调用sleep方法。如果获取锁后调用sleep方法，可能导致系统响应速度变慢，或造成死锁。应尽可能通过设计来避免在等待前获取锁
16. 移动应用客户端程序在开发阶段启用安全编译选项，安全编译选项包括但不限于NX、Shared、ASLR、GS、SSEH
17. 移动应用客户端程序在开发过程中应限定对BannedAPI的使用，BannedAPI是指：如果使用不当会引发安全漏洞的API的统称
18. 在运行环境中对代码进行错误检查

* 最小化执行解释器的权限
* 定义全面的转义语法
* 对执行解释器进行Fuzz测试
* 用沙箱保护执行解释器

1. 采用NDK开发核心模块
2. 使用多种技术进行代码保护，包括：

* ProGuard，进行符号信息混淆，增加分析难度，轻度对抗反汇编
* DexClassLoader，动态加载DEX文件，通过反射调用其中代码执行，实现功能的透明扩展，实现代码加密
* DexGuard，实现字段混淆，可有效防止二次打包发布
* HoseDex2Jar，实现DEX头隐藏代码
* 伪加密等异常ZIP格式的反反汇编
* 花指令和无效指令的反反汇编

1. 当ManifestallowBackup被设置为true或不设置该标志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。在AndroidManifest.xml中设置android:allowBackup=false
2. 在manifest.xml中定义Debuggable项，如果该项被打开，app存在被恶意程序调试的风险，可能导致泄漏敏感信息泄漏等问题。显示的设置manifest的debuggable标志为false
3. SQL防注入，对输入进行类型和格式检查，并采用参数化查询方式

CursorrawQuery(Stringsql,String[]selectionArgs);

Cursorquery(Stringtable,String[]columns,Stringselection,String[]

selectionArgs,StringgroupBy,Stringhaving,StringorderBy);

1. 关键控件的安全Taphijacking

voidsetFilterTouchesWhenObscured(booleanenabled);

android:filterTouchesWhenObscured=”false”

### 程序保护

1) 代码保护技术手段包括：APK伪加密、花指令、SMC、信息隐藏等

2) 针对ptrace注入目标进程攻击，客户端可以通过ptrace占坑的方式进行防护

3) 在对客户端软件进行加固处理之前，应先将自生的签名校验功能关闭或移除，否则会被识别为盗版应用

4) 安装包进行签名，通过签名做自校验

5) 使用代码混淆工具，例如SDK带的ProGuard以及其他Java混淆器

6) 采用NDK开发核心模块。APK的java层代码很容易被反编译，而C/C++库反汇难度较大。客户端程序可以把关键代码以JNI方式放在so库里。so库中是经过编译的arm汇编代码，可以对其进行加壳保护，以防止逆向分析

7) 使用官方或第三方的软件保护方案，例如SDK的android.drm包、GooglePlay的软件许可（ApplicationLicensing）支持

8) 对自身文件进行完整性校验。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。一可通过程序自检测，在程序运行时，自我进行签名比对，比对样本可以存放在APK包内，也可存放于云端。缺点是程序被破解时，自检测功能同样可能遭到破坏，使其失效；二可通过可信赖的第三方进行检测。由可信赖的第三方程序负责APK的软件安全问题。对比样本由第三方收集，放在云端。缺点是需要联网检测，在无网络情况下无法实现功能。用crc32对classes.dex文件的完整性进行校验，zipfile.getEntry("classes.dex").getCrc()。或用哈希值对整个APK完整性进行校验MessageDigest.getInstance("SHA-1").update(bytes,0,byteCount)，用linux下的sha1sum命令计算我们的APK的哈希值（sha1sumverification.apk）并存入服务器，由于我们要对整个APK的完整性进行校验，所以我们的算出哈希值就不能存在资源文件中了因为APK中任何的改动都会引起最终APK生成的哈希值的不同

9) 可采用成熟的第三方公司提供的加固方案进行整体加固，包括混淆、加密、加壳、动态加载等技术为移动应用进行全面的安全加固，对抗代码注入、反编译、逆向工程等恶意行为

10) DEX加壳、加花指令，整体进行加解密

11) so文件保护。使用自定义加载器加载so文件，so文件的关键函数实现动态加解密。对加密so文件进行优化压缩，对加密so文件源码进行加密隐私，加密后so文件能有效防止IDA等工具逆向分析

12) 使用系统机制防止注入攻击

13) 使用公钥算法验证APK包的完整性，私钥不存储。在验证时生成，并且私钥与程序代码完整性相关

14) Android的DEX分段进行加解密

15) Android的APK通过文件的相互关联做为密钥对要加密的部分进行加解密，原生进行自校验

16) 使用判断敏感文件以及内存状态检测调试器

17) 可以以加密代码的方式防止反编译

18) 通过对app的完整性保护，防止app被篡改或二次打包盗版

19) 防止应用运行时被动态注入，防止被外挂、木马偷窃账号密码，修改支付金额等

20) 对资源文件指纹签名进行验证保护，防止资源文件被篡改

21) 签名验证，可有效遏制盗版行为。可通过DEX加密自身加入对原始应用和加固后应用签名的校验，实现防止二次打包

22) DEX保护，可采取多重保护技术实现。通过静态先抽取DEX中code段，并且加密抽取出的code段，形成被抽空保护的DEX文件；动态内存解密、加载、动态回填、运行加密后DEX文件的保护方式，这种保护方式避免了系统生成临时的DEX缓存文件，并且有效的屏蔽逆向破解者通过静态反汇编阅读代码指令的可能，并且也在一定程度上防止了内存dump

23) 防止进程/线程附加。主要目的是防止动态调试，如果检测到被保护APK被附加则程序退出。在反调试程序中，全面检测运行APK中的各项进程或线程的状态，根据状态判断是否被附加，若被附加即程序及时退出

24) 防止进程注入。主要目的是防止进程注入，搜索内存数据，如果检测到被保护APK的内存空间非法访问则程序退出。在反调试程序中，特定模块监测被保护APK的内存空间，如果被保护的内存空间被其他进程读取、写入，则反调试程序将会监测到，并作出相应处理

25) 对市面上常用的调试器进行检测。主要目的是防止恶意调试器对被保护APK进行恶意访问，如果检测到调试器启动，被保护的APK程序自行退出。反调试程序中有特定的线程扫描整个系统进程中的PID，正常情况下系统不会启动调试器程序或系统不会包含有调试器程序，所以在系统内存中发现有启动的调试器程序，反调试程序就会检测到，并作出相应的处理

26) 反调试与DEX壳进行相互校验。主要目的是确保整个DEX壳和反调试的完整性，如果检测到被保护的APK中DEX壳和反调试部分被篡改，则被保护APK程序自行退出。在反调试程序中利用特定导出的用于校验的接口和DEX壳进行相互校验，当被保护APK程序启动时，DEX壳先做自身校验，并且主动通过反调试程序导出的接口传入校验文件与反调试程序通讯，反调试程序根据和DEX壳制定的协议，对校验文件进行二次校验，并算出校验值，算出的校验值与DEX传入的校验值进行比对，如果比对结果不一致，则说明DEX壳和反调试被篡改，做出相应的处理

27) 防止壳启动调试。利用自主的安全编译器把壳代码乱序、拆分方法，并插入垃圾代码、反调试代码，增加调试难度

28) 防ZjDroid插件内存dump DEX。实时监测ZDJORID插件的攻击行为，当发现被保护APK受到此类攻击时，采取相应的保护措施

29) 防内存篡改。主要目的是防止恶意程序动态篡改受保护APK的进程内存空间。在反调试程序中利用特定的线程去实时检测当前的APK进程内存空间是否被恶意程序攻击篡改，如有发现，则采取自我保护的机制

30) 防系统核心库被HOOK（劫持）攻击。主要目的是防止恶意程序通过HOOK android系统库（例如：libc.so、libdvm.so、libart.so）获取APK中的敏感信息或某些违法操作。通过监测hook行为，去识别是否有劫持系统库的操作

### 输出安全

1. 移动应用源代码中的冗余或注释代码须彻底删除。比如涉及开发人员信息、调试信息等等
2. 身份证号等敏感信息使用相同位数的同一特殊字符，例如\*和#代替
3. 屏蔽系统技术错误信息，不将系统产生的错误信息直接反馈给客户端

### 传输安全

1. 禁止不加密明文传输敏感数据。对敏感数据采用基于SSL/TLS的HTTPS进行传输。禁止使用HTTP协议登录账户或交换数据
2. 禁止SSL通信不检查证书有效性。在SSL/TLS通信中，客户端通过数字证书判断服务器是否可信，并采用证书中的公钥与服务器进行加密通信。禁止在代码中不检查服务器证书的有效性（checkServerTrusted()方法实现为空），或选择接受所有的证书（setHostnameVerifier(SSLSocketFactory.ALLOW\_ALL\_HOSTNAME\_VERIFIER)），因为这样会存在中间人攻击的风险。必须使用STRICT\_HOSTNAME\_VERIFIER并校验证书，详细安全编码方案请参照Google官方关于SSL的安全建议：
3. https://developer.android.com/training/articles/security-ssl.html
4. 支付过程必须数据必须全程加密
5. 移动办公应用，采用VPN链路加密技术，强化网络传输通道安全
6. 涉及支付，客户端和服务端通信强制采用https加密
7. 移动应用和服务器端SSL通信必须严格检查服务器端证书有效性，避免手机用户访问钓鱼网站泄露敏感信息。代码中需要有SSL验证。

### 存储安全

1. Sdcard存储的信息可以被其他应用任意读取，如果有敏感信息会造成信息被恶意应用获取风险。通过内部存储存文件openFileOutput(filename,Context.MODE\_PRIVATE)或者对于敏感的数据加密后存储
2. 本地存储密钥存在被攻击者利用并通过密钥构造伪数据的风险。可以通过网上的第三方加固平台来实现，同时建议在进行加解密等敏感操作的时候，将算法写到so扩展库中，这样处理会更安全
3. 敏感数据或文件，必须加密存储
4. 禁止将隐私数据明文保存在外部存储。建议对隐私数据进行加密，密码保存在内部存储，由系统托管或者由用户使用时输入
5. 禁止将系统数据明文保存在外部存储。目的是为避免被恶意软件读取
6. 禁止将软件运行时依赖的数据保存在外部存储。针对配置文件，建议保存到内部存储；如果必须存储到SD卡，则应该在每次使用前判断它是否被篡改，例如，与预先保存在内部的文件哈希值进行比较
7. 禁止将软件安装包或者二进制代码保存在外部存储。建议在安装或加载位于SD卡的任何文件（APK、DEX、JAR）之前，对其完整性做验证，判断其与实现保存在内部存储中的（或从服务器下载来的）哈希值是否一致
8. 禁止创建全局可读或全局可写的内部文件。在使用openFileOutput(Stringname,intmode)方法创建内部文件时，禁止将第二个参数设置为Context.MODE\_WORLD\_READABLE或Context.MODE\_WORLD\_WRITEABLE，这个文件变为全局可读或全局可写的。如果要跨应用共享数据，建议实现一个ContentProvider组件，提供数据的读写接口，并为读写操作分别设置一个自定义权限
9. 使用数据方式存放敏感信息，并在不再需要时显式覆盖清除它。避免存储在字符串对象中，因字符串对象在下一次垃圾回收机制GC才有可能被覆盖掉
10. 对私有目录（通常位于“／data／data／应用名称／”）下的文件权限进行限制，正常的文件权限最后三位应为空（类似“rw-rw----”），即除应用自己以外任何人无法读写，目录文件则允许多一个执行位（类似“rwxrwx—x”）
11. 由于可以使用ptrace监视其他进程的方式进行内存窥视，关闭android:debugable="false"，反ptrace防止dump内存。想完全防止ptrace并不可能，可以增加dump出的内容的分析破解难度，因此需对敏感信息进行加密
12. 禁止密码被Webview明文存储。使用Webview时需要关闭Webview的自动保存密码功能,防止用户密码被Webview明文存储。设置Webview.getSettings().setSavePassword(false)

### 日志安全

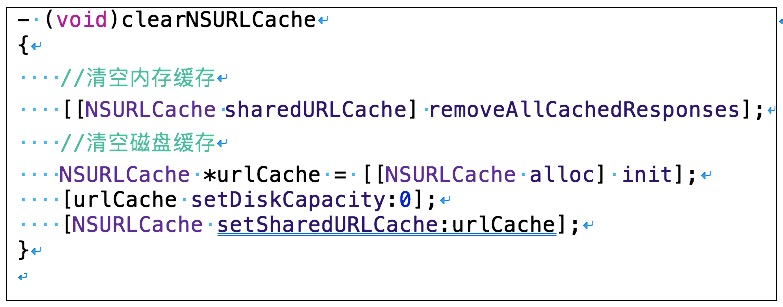
1. 对源码中的开发日志信息Log进行清除，提高程序运行效率，防止日志暴露敏感信息
2. 禁止输出调试日志信息，应用Release版本发布时应关闭Log输出。可以通过proguard来删除各种日志输出代码。然后导出APK时，将会过滤掉日志代码。通过配置proguard，将类android.util.Log的方法给置为为无效代码

### 应用发布安全

* 1. 发布应用中禁止包含测试代码
  2. 发布应用中禁止输出日志信息
  3. 打开服务器校验功能
  4. 应用程序应能正确安装到设备上
  5. 包含数字签名信息

## APP安全开发基线\_iOS

### 代码安全

1. 开发过程中对书写的类、方法的名字进行混淆，以到达直接从名字难以辨认其功能的作用
2. 使用iOS源码混淆方案，对类方法的逻辑流程进行高强度的混淆，这样即使源码头被Dump出来，逻辑也是难以分析的
3. 确保在生成应用程序时调试符号被剥离。剥离调试符号将减少二进制的大小，并增加逆向工程的难度。要删除调试符号，请在项目的构建设置中将“Strip Debug SymbolsDuring Copy”设置为YES
4. 清除NSUrl缓存以防止存在泄漏敏感数据的风险：
5. 防止缓冲区溢出，缓冲区溢出造成的危害比较多样，轻微的就是程序直接崩溃，比较严重的是错误的写入覆盖了其他敏感数据，造成数据的丢失；最严重的莫过于执行恶意代码，因为数据写入越界，恶意代码可以将原先正常的函数返回地址修改为自己的代码，从而获得整个软件的执行权。



1. UIWebView安全，在webView:shouldStartLoadWithRequest:方法中加载指定域名的请求，过滤非HTTPS请求和未知域名：



### 输出安全

1. 禁止不加密明文传输敏感数据。对敏感数据采用基于SSL/TLS的HTTPS进行传输。禁止使用HTTP协议登录账户或交换数据
2. 禁止SSL通信不检查证书有效性。在SSL/TLS通信中，客户端通过数字证书判断服务器是否可信，并采用证书中的公钥与服务器进行加密通信。禁止在代码中不检查服务器证书的有效性。
3. 支付过程必须数据必须全程加密
4. 禁止UIWebView远程代码执行
5. 使用更安全的WKWebView控件
6. 最小化攻击面，在webView:shouldStartLoadWithRequest:方法中加载指定域名的请求，过滤非HTTPS请求和未知域名
7. 移动办公应用，采用VPN链路加密技术，强化网络传输通道安全
8. 涉及支付，客户端和服务端通信强制采用https加密
9. 移动应用和服务器端SSL通信必须严格检查服务器端证书与域名的有效性，避免用户访问钓鱼网站泄露敏感信息。代码中需要有SSL验证：



### 程序保护

1. 对自身文件进行完整性校验
2. 安装包进行签名，通过签名做自校验
3. 避免使用通用的加密解密库、网络请求库等，防止被黑客通过hook的手段在通用的地方埋点

### 传输安全

1. 禁止不加密明文传输敏感数据。对敏感数据加密后采用基于SSL/TLS的HTTPS进行传输
2. 禁止SSL通信不检查证书有效性。在SSL/TLS通信中，客户端通过数字证书判断服务器是否可信，并采用证书中的公钥与服务器进行加密通信。
3. 移动办公应用，采用VPN链路加密技术，强化网络传输通道安全。

### 存储安全

1. 对保存在本地的用户隐私信息进行加密处理
2. 禁止UIWeb中实现加密算法以及秘钥的明文存储
3. 防止iOS7以上的系统进入后台后自动截图

### 日志安全

在APP发布前一定要把程序内所有可能泄露敏感信息的提示日志都删除

### 应用发布安全

1. 发布应用中禁止包含测试代码
2. 打开服务器校验功能
3. 应用程序应能正确安装到设备上
4. 包含数字签名信息

## 业务通用安全开发基线

### 用户安全

1. 为每一个具有访问权限的用户分配唯一的ID，以保证对于关键数据和支付软件的操作能够被追溯到已知的、被授权的用户。
2. 除了分配唯一的ID，至少采样下列方式的一种方法以鉴别所有合法用户：

* 口令
* 短信认证
* 令牌设备
* 生物特征

1. 针对高风险业务，采用双因子身份认证机制；
2. 加密所有密码，无论是在传输过程中或存储在任何应用组件中；
3. 采用适当的密码管理策略：

* 在执行口令重置前认证用户身份；
* 口令最小长度不低于六个字符；
* 如果一个会话空闲的时间超过一定时长，要求用户再次输入口令以重新激活终端应用；
* 采用一种或几种有效的方法防止密码的暴力猜解，合适的方法可以包括但不限于：

a)设置密码验证次数的限制，指对登录失败次数设置合理上限，超限可以采取账号锁定等措施；

b）提升密码复杂度，例如在设置密码过程中要求用户将密码设置为包含两类以上不同字符等。

1. 登录过程中的加密机制，建议采用体系比较完善的HTTPS机制
2. 设置连续失败登录次数，超过限定次数应在短时间内锁定系统登录权限
3. 设置登录验证码防止暴力破解。设置连续失败登录次数，超过限定次数应在短时间内锁定系统登录权限
4. 登录输入框记住/不记住用户名、密码
5. 提示登录失败的原因，或统一提示，模糊失败原因
6. 比较新旧密码，不允许相同
7. 判定密码长度，不少于6位
8. 采用\*代替密码显示
9. 输入密码、二维等敏感信息时，禁用屏幕录像功能

### 身份与访问控制

1. 除了那些特定设为“公开”的内容以外，对所有的网页和资源要求身份验证。
2. 所有的身份验证过程必须在可信系统（比如：服务器）上执行。
3. 在任何可能的情况下，建立并使用标准的、已通过测试的身份验证服务。
4. 为所有身份验证控制使用一个集中实现的方法，其中包括利用库文件请求外部身份验证服务。
5. 将身份验证逻辑从被请求的资源中隔离开，并使用重定向到或来自集中的身份验证控制。
6. 所有的身份验证控制应当安全的处理未成功的身份验证。
7. 所有的管理和账户管理功能至少应当具有和主要身份验证机制一样的安全性。
8. 如果您的应用程序管理着凭证的存储，那么应当保证只保存了通过使用强加密单向salted哈希算法得到的密码，并且只有应用程序具有对保存密码和密钥的表/文件的写权限（如果可以避免的话，不要使用MD5算法）。
9. 密码哈希必须在可信系统（比如：服务器）上执行。
10. 只有当所有的数据输入以后，才进行身份验证数据的验证，特别是对连续身份验证机制。
11. 身份验证的失败提示信息应当避免过于明确。比如：可以使用“用户名和/或密码错误”，而不要使用“用户名错误”或者“密码错误”。错误提示信息在显示和源代码中应保持一致。
12. 为涉及敏感信息或功能的外部系统连接使用身份验证。
13. 用于访问应用程序以外服务的身份验证凭据信息应当加密，并存储在一个可信系统（比如：服务器）中受到保护的地方。源代码不是一个安全的地方。
14. 只使用HTTPPost请求传输身份验证的凭据信息。
15. 非临时密码只在加密连接中发送或作为加密的数据（比如，一封加密的邮件）。通过邮件重设临时密码可以是一个例外。
16. 通过政策或规则加强密码复杂度的要求（比如：要求使用字母、数字和/或特殊符号）。身份验证的凭据信息应当足够复杂以对抗在其所部署环境中的各种威胁攻击。
17. 通过政策和规则加强密码长度要求。常用的是8个字符长度，但是16个字符长度更好，或者考虑使用多单词密码短语。
18. 输入的密码应当在用户的屏幕上模糊显示（比如：在Web表单中使用“password”输入类型）。
19. 当连续多次登录失败后（比如：通常情况下是5次），应强制锁定账户。账户锁定的时间必须足够长，以阻止暴力攻击猜测登录信息，但是不能长到允许执行一次拒绝服务攻击。
20. 密码重设和更改操作需要类似于账户创建和身份验证的同样控制等级。
21. 密码重设问题应当支持尽可能随机的提问（比如：“最喜爱的书”是一个坏的问题，因为《圣经》是最常见的答案）。
22. 如果使用基于邮件的重设，只将临时链接或密码发送到预先注册的邮件地址。
23. 临时密码和链接应当有一个短暂的有效期。
24. 当再次使用临时密码时，强制修改临时密码。
25. 当密码重新设置时，通知用户。
26. 阻止密码重复使用。
27. 密码在被更改前应当至少使用了一天，以阻止密码重用攻击。
28. 根据政策或规则的要求，强制定期更改密码。关键系统可能会要求更频繁的更改。更改时间周期必须进行明确。
29. 为密码填写框禁用“记住密码”功能。
30. 用户账号的上一次使用信息（成功或失败）应当在下一次成功登录时向用户报告。
31. 执行监控以确定针对使用相同密码的多用户帐户攻击。当用户ID可以被得到或被猜到时，该攻击模式用来绕开标准的锁死功能。
32. 更改所有厂商提供的默认用户ID和密码，或者禁用相关帐号。
33. 在执行关键操作以前，对用户再次进行身份验证。
34. 为高度敏感或重要的交易账户使用多因子身份验证机制。
35. 如果使用了第三方身份验证的代码，仔细检查代码以保证其不会受到任何恶意代码的影响。
36. 对下载资源进行访问授权控制
37. 限制单位时间内事务请求数量
38. 用户权限最小化
39. 不允许同一个账号同时在多个设备上成功登录客户端
40. 限制客户端登录尝试次数
41. 登录失败提示内容统一

### 密码算法安全

1. 避免使用过时的加密算法。如DES、RC2。可使用3DES、AES，并配合高强度密钥来加密数据
2. 确保密码学加密算法密钥强度。NIST标准SP800-57推荐AES加密密钥至少128bit，以确保数据在获得2030年计算能力之前的完整性和机密性。RSA算法密钥则确保在1024bit以上
3. 使用SecureRandom时不要使用SecureRandom(byte[]seed)这个构造函数，会造成生成的随机数不随机。建议通过/dev/urandom或者/dev/random获取的熵值来初始化伪随机数生成器PRNG
4. 如果有初始密码，首次登陆时强制客户修改初始密码
5. 有自动检验弱口令的能力，当密码输入不符合复杂度要求时，不允许注册成功或修改密码，，应提示客户不使用简单密码
6. 及时加密登录密码/支付密码等
7. 图形认证码应满足
   * + 由数字和字母等字符混合组成
     + 随机产生
     + 采取图片底纹干扰、颜色变换、设置非连续性及旋转图片字体、变异字体显示样式等有效方式，防范恶意代码自动识别图片上的信息
     + 具有使用时间限制并仅能使用一次
     + 图形认证码应由服务器生成，客户端源文件中不应包含图形验证码文本内容
8. 定期更换密钥
9. 登录密码等其它密码不能明文保存在移动终端本地
10. 支付密码不能保存在移动终端本地
11. 禁止硬编码形式存储密钥
12. 不直接存储用户的密码信息，而是存储用户的密码的摘要（HASH）信息，每次用户登录时进行摘要匹配
13. 采用一种或几种有效的方法防止密码的暴力猜解，合适的方法可以包括但不限于：

* 设置密码验证次数的限制，指对登录失败次数设置合理上限，超限可以采取账号锁定等措施
* 提升密码复杂度，例如在设置密码过程中要求用户将密码设置为包含两类以上不同字符等

### 会话安全

1. 会话标识唯一、随机、不可猜测
2. 会话过程中应维持认证状态
3. 退出登录或移动应用移动应用关闭后，应立即终止会话
4. 会话应设置超时时间，当空闲时间超过设定时间应自动终止会话
5. 会话结束后，应及时清除会话信息
6. 对系统的最大并发会话连接数进行限制
7. 当应用系统通信双方中的一方在指定时间内未作任何响应，另一方能够自动结束会话
8. 访问令牌机制。Access\_token是一个访问令牌，包含了登录会话的安全信息。Access\_token是用户登录后由服务器端生成的唯一的、标示用户登录信息的令牌，用户在访问受限（如登录访问）信息时需传递该参数
9. Token应动态变化

### 会话管理

1. 使用服务器或者框架的会话管理控制。应用程序应当只识别有效的会话标识符。
2. 会话标识符必须总是在一个可信系统（比如：服务器）上创建。
3. 会话管理控制应当使用通过审查的算法以保证足够的随机会话标识符。
4. 为包含已验证的会话标识符的cookie设置域和路径，以为站点设置一个恰当的限制值。
5. 注销功能应当完全终止相关的会话或连接。
6. 注销功能应当可用于所有受身份验证保护的网页。
7. 在平衡的风险和业务功能需求的基础上，设置一个尽量短的会话超时时间。通常情况下，应当不超过几个小时。
8. 禁止连续的登录并强制执行周期性的会话终止，即使是活动的会话。特别是对于支持富网络连接或连接到关键系统的应用程序。终止时机应当可以根据业务需求调整，并且用户应当收到足够的通知已减少带来的负面影响。
9. 如果一个会话在登录以前就建立，在成功登录以后，关闭该会话并创建一个新的会话。
10. 在任何重新身份验证过程中建立一个新的会话标识符。
11. 不允许同一用户ID的并发登录。
12. 不要在URL、错误信息或日志中暴露会话标识符。会话标识符应当只出现在HTTPcookie头信息中。比如，不要将会话标识符以GET参数进行传递。
13. 通过在服务器上使用恰当的访问控制，保护服务器端会话数据免受来自服务器其他用户的未授权访问。
14. 生成一个新的会话标识符并周期性地使旧会话标识符失效（这可以缓解那些原标识符被获得的特定会话劫持情况）。
15. 在身份验证的时候，如果连接从HTTP变为HTTPS，则生成一个新的会话标识符。在应用程序中，推荐持续使用HTTPS，而非在HTTP和HTTPS之间转换。
16. 为服务器端的操作执行标准的会话管理，比如，通过在每个会话中使用强随机令牌或参数来管理账户。该方法可以用来防止跨站点请求伪造攻击。
17. 通过在每个请求或每个会话中使用强随机令牌或参数，为高度敏感或关键的操作提供标准的会话管理。
18. 为在TLS连接上传输的cookie设置“安全”属性。
19. 将cookie设置为HttpOnly属性，除非在应用程序中明确要求了客户端脚本程序读取或者设置cookie的值

### 支付安全

* 1. 传输过程全程加密，保障数据传输过程，防止敏感数据篡改。尽量采用国家密码管理局通过的加密算法
  2. 会话空闲的时间超过一定时长，要求用户再次输入口令以重新激活终端应用
  3. 客户端支付信息显示最少的用户敏感数据，限制数据存储和保留时间，达到恰好能满足法律和管理规定需要的程度。显示卡号等敏感信息时进行部分遮挡
  4. 双向认证，例如使用SSL/TLS或IPSEC等协议
  5. 支付环节进行多因子验证
  6. 保障移动APP插件与支付网关系统间交互的安全，如通过数字签名认证系统身份、支付网关信息交互加密、移动APP与支付网关APP插件互相认证
  7. 确保APP插件调用第三方代码或库文件来源的可靠性，验证第三方接口或组件的合法性，确保随时跟进第三方库的更新
  8. 遵循移动应用与支付网关在外网上关键数据最小化传递原则，采用“后台推送与前台跳转配对”方式。采取内外网数据分离传输方式增强移动支付安全性。后台应用系统通过内网传输信息，前台软件通过外网进行支付操作。前台软件通过订单号与支付网关进行交互，支付网关根据订单号在后台系统通过内网专线查询出对应信息并实现支付扣款
  9. 应对移动应用进行签名，标识移动应用的来源和发布者，保证客户所下载的移动应用来源于所信任的机构
  10. 在启动和更新时应进行真实性和完整性校验，防范移动应用被篡改或替换。
  11. 临时文件中不应出现敏感信息，临时文件包括但不限于Cookies。移动应用应禁止在身份认证结束后存储敏感信息，防止敏感信息的泄露
  12. 保护在客户端启动的用于访问WEB的进程，防止非法程序获取该进程的访问权限

1. 输入密码时，需禁用屏幕截屏功能
2. 采取代码混淆等技术手段
3. 验证短信宜采进行加密

### 运行环境安全

1. 对移动办公终端进行应用控制。通过使用应用控制策略，定义哪些应用是必装的，哪些是允讲安装的，哪些是不允讲安装的。若终端不满足准入检查策略，则移动终端弹出提示消息，提醒用户作出相应修改，否则不允许接入

## 移动行业安全现状

近年来，智能移动终端设备用户群体涵盖面极广，其中较大比重的用户安全意识较为淡薄、且缺乏对相关恶意应用的甄别能力，移动应用安全形势不容乐观，一旦爆发影响版本较多的安全漏洞，会造成极广的用户影响面。对于移动应用市场安全，国家网信办、工信部等多部门曾出台相关政策规范行业监管。政策也有对移动应用安全做了明确规定，应建立事前检测、事中加固、事后感知的动态防御体系，真实有效保障移动应用安全，同时该防御体系也更符合国家监管部门要求。