计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(11):233−237 [doi: 10.15888/j.cnki.csa.006107]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

Android 移动应用的三种安全解决方法探究①

顾学海[1], 王聪颖[2], 胡　牧[1], 蒋厚明[1], 曹海涛[1]

[1](南瑞集团公司 (国网电力科学研究院), 南京 210000)
[2](江苏经贸职业技术学院, 南京 210000)

摘　要: 随着移动终端智能设备的迅速更新换代, android 操作系统的移动终端应用的种类也越来越多, 并且更新速度快. Android 智能手机的迅猛发展, 使得 android 安全机制成为开发者和用户都关注的热点问题. 在对 android 应用安全现状和安全机制简要分析下, 针对三种安全问题, 提出了相应的解决方案, 并以实际例子, 实现了相关安全机制的解决方法, 实践证明, 提高了应用的安全性.

关键词: android; 安全现状; 安全机制

引用格式: 顾学海,王聪颖,胡牧,蒋厚明,曹海涛.Android 移动应用的三种安全解决方法探究.计算机系统应用,2017,26(11):233–237. http://www.c-s-a.org.cn/1003-3254/6107.html

Three Security Solutions of Android Mobile Application

GU Xue-Hai[1], WANG Cong-Ying[2], HU Mu[1], JIANG Hou-Ming[1], CAO Hai-Tao[1]

[1](Nari Group Corporation (National Electricity Science Research Institute), Nanjing 210000, China)
[2](Jiangsu Vocational Institute of Commerce, Nanjing 210000, China)

Abstract: With the rapid upgrading of intelligent mobile terminal equipment, the mobile terminal applications of android operating system also have more species, and their update is faster. The rapid development of android smart phones makes android security mechanism a hot issue for developers and users. After a brief analysis of present situation and the security mechanism for the android security, aiming at the three security problems, the corresponding solutions are proposed. With practical examples, the method is realized a relevant security mechanism solution. Practice has proved that the method improves the security of the application.

Key words: android; safety situation; security mechanism

Android 操作系统是 Google 最早于 2007 年 11 月推出的一种基于 Linux 2.6 核心的开源智能手机操作系统, 至今已经发布了多个版本, 目前已发展至 android 7.0(android N) 版本. 在现阶段的发展过程中, android 主要具有操作系统、用户界面以及应用程序三个组成部分[1]. Android 终端设备具有相对独立的操作系统, 具有响应快、应用多、操作方便、实用性高等特点, 给人们的办公和生活等方面都带来了方便和快捷. 兼顾系统的性能和开发方便等特点让开发者对 android 开发也情有独钟. 在享受便捷的同时, 由于 android 系统中具有开放源码的特征, 使得开发者或用户对其安全隐患予以高度的重视, android 系统的安全机制成为开发者研究一个重要课题.

1　Android 安全机制

Android 的系统架构[2]从下至上可以简单分为四层: linux 内核层、硬件抽象层、系统运行时库层、应用程序框架层和应用程序层, 如图 1 所示. 在方便用户

① 基金项目: 国家电网公司科技项目 (524606160205)
收稿时间: 2016-02-03; 修改时间: 2016-02-23; 采用时间: 2017-04-05

使用, 方便开发者开发的过程中, 人们对安全性的考虑也越来越多, 特别是涉及到个人的金钱交易和隐私信息, 单位的涉密信息的相关应用. Android 系统开源的特点, 能让系统不断丰富发展, 但是市场上根据开源代码所定制的 ROM 各有不同, 在系统层对应的安全防范层次也有所差别. 另外, android 应用市场对 android 应用的审核比较宽松, 许多漏洞有了生成的机会, 降低了安全性. 目前移动应用涉及的单位涉密信息、个人金钱交易信息越来越多, 需要对 app 安全防范不断加强. 本文结合国网员工信息统计 app, 对该 app 涉及到的三种安全问题进行了说明, 列举了病毒、app 重打包、数据传输安全这三个方面进行了安全机制的说明.

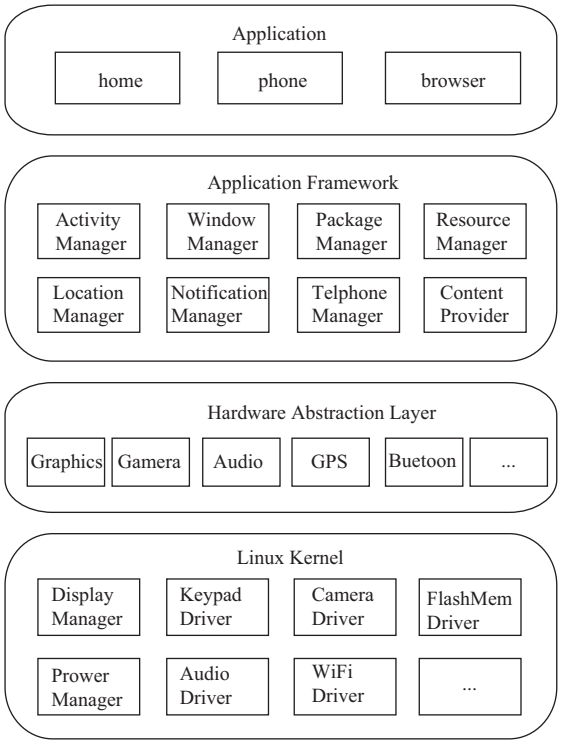


图 1 android 系统架构图

(1) 病毒

手机病毒主要是为了窃取隐私、获取敏感交易密码、恶意消耗手机流量等. Android 的恶意软件层出不穷, 除了熟知的木马、病毒、蠕虫、DDOS、劫持、僵尸等还有如间谍软件 (spyware)、恐吓软件 (scareware)、勒索软件 (ransomware)、广告软件 (adware) 等[3]. 根据危害结果将 android 病毒分 5 大类: 移动支付、资费消耗、隐私窃取、远程控制、黑客工具[6,7]. 手机木马是 android 病毒主要内容, 主要是一些恶意的应用程序. “微信鬼面”、“手银鬼手”、“银联间谍”是比较热门的木马病毒, 这些病毒都是通过钓鱼的方式窃取网银账号密码及验证码信息, 从而盗刷银行卡余额. 手机木马有的独立存在, 有的则伪装成图片文件的方式附在正版 app 上, 隐蔽性极强, 部分病毒还会出现变种, 并且一代比一代更强大[4]. 这些病毒有一些通用的特点[4]:

1) 母包和恶意子包的运行方式.

2) 使用技术手段阻止用户通过正常方式卸载相关应用.

3) 主要目的是窃取用户账户密码, 从而窃取用户内部信息或银行卡余额.

4) 通过短信作为指令通道.

(2) app 二次打包

通过正版 apk 包进行解析反编译后重新加入恶意的代码逻辑, 进行代码注入[5], 生成一个新的带有恶意功能的 apk 文件. 这种方式插入的恶意功能一般都与病毒相关, 而且用户基本无法识别, 具有较强的伪装性. 防止 app 二次打包也可以防止病毒的入侵和传播[4].

(3) 数据传输安全

数据在传输过程被黑客攻击, 劫持传输的数据, 从而进行篡改等操作. 目前许多企业级业务应用或者与金钱交易相关的应用都会通过 https 方式进行请求数据, 试图保证数据传输安全, 但是目前对 https 的攻击日趋增长, ROM 的参差不齐使得 https 机制也不能完全保证数据的传输安全.

2 解决方案探究及实现

在互联网技术的发展趋势下, 国家电网公司对移动应用的开发也进行了深入的研究. 针对国网公司内部的移动应用开发, 用户数据、业务数据等内容都是公司的重要信息, 所以更加需要 android 安全机制的实现, 从而保证数据的安全性. 下面以国网公司员工信息统计 app 为例, 验证了上述的安全机制的实现.

(1) app 病毒扫描模块: 杀毒引擎属于静态的检测方法, 该方法可以对系统中的应用程序进行鉴定[9]. 病毒扫描主要是通过扫描的方式获取 android 智能设备已安装的 app 的包名、so、dex、签名等信息, 把获取的信息上传到云端, 通过对病毒库中的签名与 apk 应用程序签名的比较判断是否属于病毒范畴.

国网员工统计 app 对应的病毒扫描模块包括: 下载病毒库信息, 杀毒引擎实现, 获取程序的签名. 通过

移动智能设备连接服务器端, 把病毒的版本库信息下载到设备本地, 将解析的数据存放到 List 集合中, 即从服务器解析病毒库并获取到病毒库的集合, 代码实现包资源管理器的实例化, 获取到当前智能终端中的所有包名, 杀毒引擎根据病毒库比对当前系统里面的程序包名签名进行杀毒, 检查当前的 packname 和对应签名是否与病毒库里面的信息相同, 如果一致, 则添加为病毒, 并记录病毒数. 其中病毒检测代码如下:

```
for(VirusBean virusbean : virusbeans){
  if(packname.equals(virusbean.getPackname())
  && taskInfo.getAppSignature(packname).equals
(virusbean.getSignature())) {
  virusResult.add(packname);
    }
  }
    count++;
  }
  Message msg = new Message();
  msg.what = SCANNING_FINISH;
  msg.obj = count;
  handler.sendMessage(msg);
} catch (Exception e) {
  e.printStackTrace();
  }
}
}.start();
return super.onTouchEvent(event);
}
```

获取程序签名的代码如下:

```
public String getAppSignature(String packname){
  try {
      PackageInfo packinfo =pm.getPackageInfo
(packname, PackageManager.GET_SIGNATURES);
return packinfo.signatures[0].toCharsString();
  } catch (NameNotFoundException e) {
    e.printStackTrace();
    return null;
  }
}
```

(2) 防止二次打包: apk 文件是 android 平台下的可执行文件, 它本质上是一种 zip 压缩文件格式, 其文件结构及作用大致如表 1 所示[10,11]. apk 重打包有利于防止注入恶意功能与病毒. 防止重打包原理是: 在 java 层、服务端或者 so 层, 通过比较安装 apk 与正版 apk 签名信息 MD5 加密后的 MD5 码, 来判断是否被重打包. 防止二次打包的一种重要方式就是对 apk 进行加固处理, 加固后的代码文件都已经被隐藏起来, 只保留了加固后的保护程序, 从而使得破解者无法通过反编译找到源程序所对应真正的源码, 阻止了二次打包操作.

表 1 app 文件目录

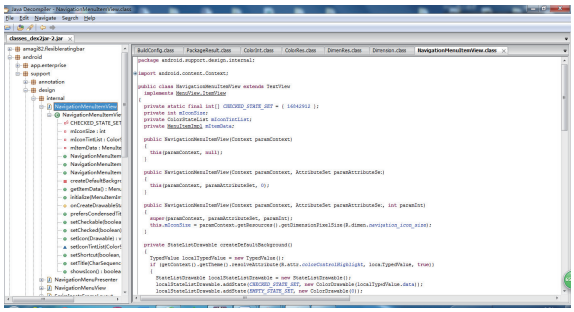
文件	作用
lib目录	存放程序所用的动态库文件
META-INF目录	存放应用程序签名信息
assert目录	资源文件夹, 该目录下的资源不会在R.java中自动生成ID
res目录	资源文件夹, 该目录下的资源文件会在R. java中被自动生成ID值
AndroidManifest	项目清单文件, 用于配置组件, 声明权限等
resources.arsc	编译后的二进制资源文件
classes.dex	Java源码编译后的Dalvik虚拟机字节码文件

理解 apk 的机器识别原理对防止重打包有一定的帮助, 因为签名和包名是 apk 的唯一识, 所以可以通过包名来确定对应的 apk 并利用签名来判断是否是正版 apk[8]. 国网公司员工信息统计 app 内部在启动的时候通过获取 apk 本身的签名然后和正确的签名做对比来识别自己是否被二次打包. 通过 PackageManager 对象可以获取 apk 自身的签名, 代码如下:

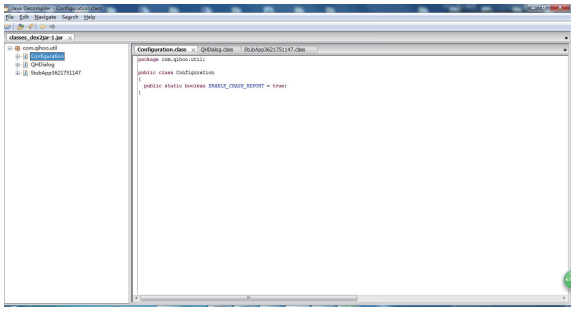
```
  public void getSignInfo(){
  try{
      PackageInfo packageInfo = getPackageManager().
getPackageInfo(
    getPackageName(), PackageManager.GET_
SIGNATURES);
      Signature[] signs =packageInfo.signatures;
      parseSignature(sign.toByteArray());
    } catch(Exception e){
      e.printStackTrace();
    }
  }
```

通过分析签名码可以得到一串 apk 签名的 MD5 值的字符串, 通过获取的签名 MD5 值与正确的 MD5 值进行对比, 就可以识别其 apk 是否被重打包[8].

app 未签名加固前与签名加固后的反编译效果如图 2 所示, 其中对未加固 app 反编译后, 能够看到原来开发 app 时的相关代码, 通过防止二次打包加固处理后的 app, 对其进行反编译后, 无法复现 app 的相关代码, 从而可以避免黑客进行代码的修改, 并防止二次打包的操作.



(a) 未签名加固 app 反编译



(b) 已签名加固app反编译

图 2 反编译效果图

(3) 数据传输安全: android 移动应用不会只有客户端 app, 往往与后台服务之间有着数据的相互交换等操作. 在数据传输的过程中, 会存在部分敏感数据, 所以需要使用正确安全的传输方式, 从而避免敏感信息被截取. 可以采取如下措施[6]:

1) 使用 SSL 协议 (或者基于 SSL 的协议) 传输敏感数据.

2) 验证服务器证书的有效性, 如果证书的有效性验证失败, 则直接断开连接, 从而保证安全.

3) 尽量减少使用短信进行敏感信息的发送.

4) app 应用程序对敏感数据加密后再通过 SSL 进行传输.

目前虽然很多 app 使用了 https 通信方式, 但是并没有对 SSL 证书有效性做验证. 攻击者可以轻而易举地获取手机用户的明文通信内容. 针对该问题, 可以通过服务器证书认证锁定的方法解决. 其原理是在代码中精确的验证当前服务器是否持有某张指定的证书. 证书锁定的方法可以通过 X509TrustManager 接口来实现, 它通过在 SSL 回调函数中读取服务器证书密钥并和程序预埋的证书密钥进行对比, 如果两者不一致则强行断开链接, 如图 3 所示.

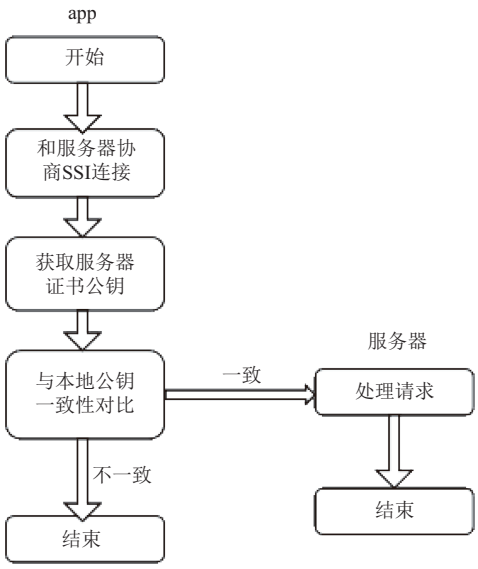


图 3 数据传输安全流程

使用 https 需要定义实现 X509TrustManager 接口的类, 并重写里面的方法, 这些方法会在建立 SSL 链接的过程中被自动调用.

https 协议中获取的服务器证书链 (chain) 将自动传入方法代码如下:

```
try{
    TrustManagerFactory tmfac =
TrustManagerFactory.getInstance("X509");
    tmfac.init((KeyStore) null);
  for (TrustManager trustManager:
tmfac.getTrustManagers()) {
    ((X509TrustManager)
trustManager).checkServerTrusted(chain, authType);
    }
    } catch (Exception e) {
      throw new CertificateException(e);
    }
```

其中, authType 为建立 SSL 链接时使用的非对称加密算法, try 中执行 SSL/TLS 常规性检查, 把编码后的密钥转换成 16 进制形式的大整数, 比较预埋证书密钥和

服务器证书密钥是否一致, 如果比较一致则处理请求, 如果不一致, 停止请求. 用 fiddler 工具分别拦截了数据传输安全未处理前和处理后的数据信息, 如图 4 所示, 没有进行数据安全处理前返回值内容可以看到相关信息, 通过 SSL 协议传输, 并验证服务器证书的有效性处理后的返回值内容为一串密文, 从而无法截获相关敏感信息, 并且也无法进行篡改操作, 从传输层面保证了数据的安全性.

(a) 未处理前

(b) 已处理后

图 4 数据传输安全处理前后比对图

(4) 三种实现方式的比较

上述三种安全防御方案, 是从三个不同的角度分析并提出了 android 的安全性提高的处理方案, 并得以实现. 病毒扫描和识别层面是通过病毒扫描机制, 实现病毒的查杀处理, 对于一般国网公司员工外网手机用户如果不装其他杀毒软件, 也可以通过国网信息统计 app 进行病毒的相关检测和查杀, 保护公司 app 的正常运行. 防止 app 二次打包是从黑客从代码篡改的层面, 进行的安全防护. 通过 app 签名和加固相结合的方法, 实现代码无法反编译, 从而防止二次打包. 数据传输安全是从传输的层面来保证安全和防止数据泄露和篡改, 通过使用 SSL 进行传输, 验证服务器证书的有效性和对敏感信息加密相结合的方法, 保证传输过程的安全.

3 结语

Android 系统将近十年的发展, 让用户体会到了 android 应用的便利性, 同时 android 的安全机制和架构也不断的受到挑战和考验. 随着移动应用的进一步普及与发展, android 的安全防御问题也需要不断的更新与发展. 本文从病毒、app 二次打包、数据传输安全三个方面对 android 的安全机制进行了深入分析, 并已公司员工信息统计 app 为例, 从病毒、app 二次打包和数据传输安全三个方面提出了解决方案, 并实现了相关的安全防御, 大大提高了公司移动应用 app 的安全性.

参考文献

1 钱锋. Android 安全机制与解决方案分析. 信息与电脑, 2006, (17): 61, 94.
2 Android system framework. http://source.androidcom/devices/index.html.
3 卿斯汉. Android 安全研究进展. 软件学报, 2016, 27(1): 45–71. [doi: 10.13328/j.cnki.jos.004914]
4 Android 应用安全现状与解决方案 (学习资料). http://blog.csdn.net/yzzst/article/details/46471277. [2015-06-15].
5 Android 防注入研究. http://blog.csdn.net/asmcvc/article/details/17436683. [2013-12-20].
6 许艳萍, 马兆丰, 王中华, 等. Android 智能终端安全综述. 通信学报, 2016, 37(6): 169–184. [doi: 10.11959/j.issn.1000-436x.2016127]
7 腾讯移动安全实验室. 腾讯移动安全实验室 2015 年上半年手机安全报告. 腾讯移动安全实验室, 2015.
8 Android APP 如何防止二次打包揭秘. https://my.oschina.net/1590538xiaokai/blog/309504.
9 丁竹青, 张维君. 基于对 Android 安全体系的分析. 信息通信, 2016, (9): 159–160.
10 梅瑞, 武学礼, 文伟平. 基于 Android 平台的代码保护技术研究. 信息网络安全, 2013, (7): 10–15.
11 伍景珠. 基于 Android 平台的软件保护方案的研究与实现 [硕士学位论文]. 北京: 北京邮电大学, 2013.