



专题:基于 Android 系统的终端安全

Android 广播机制安全性研究

卿斯汉^{1, 2, 3}

(1. 中国科学院软件研究所, 北京 100190;

2. 北京大学软件与微电子学院, 北京 102600;

3. 中国科学院信息工程研究所信息安全国家重点实验室, 北京 100093)

摘要:近年来, Android 组件的安全性是研究热点, 但作为 Android 四大组件之一的广播接收器, 则鲜见有关其安全性的研究成果。在研究 Android 广播机制的基础上, 对其安全结构与特点进行了深入分析。基于 Android 广播机制的安全挑战, 针对性地给出相应的安全对策。最后, 总结并展望今后的研究方向。

关键词:Android 系统; 广播机制; 广播接收器; 恶意软件

中图分类号:TP301

文献标识码:A

doi: 10.11959/j.issn.1000-0801.2016255

Research on Android security of broadcasting mechanism

QING Sihan^{1, 2, 3}

1. Institute of Software, Chinese Academy of Sciences, Beijing 100190, China

2. School of Software and Microelectronics, Peking University, Beijing 102600, China

3. State Key Laboratory of Information Security, Institute of Information Engineering,
Chinese Academy of Sciences, Beijing 100093, China

Abstract: In recent years, there has been increased interest in the research of Android component security. Broadcast receiver is one of the 4 components in Android, strangely enough, few papers have been published regarding to its security. Android broadcasting mechanism, and its security framework and features based on this were analyzed. According to the security challenges of Android broadcasting mechanism, the corresponding countermeasures and solutions were proposed. Finally, a summary of the results and the future research directions were given.

Key words: Android system, broadcasting mechanism, broadcast receiver, malware

1 引言

在 Android 中, 广播接收器(broadcast receiver)是接收广播消息的邮箱, 应用程序可以拥有多个广播接收器。以广播接收器为核心, Android 有自己独特的广播机制。

广播接收器是 Android 的四大组件之一, 在应用中几乎无处不在。Android 系统和任何应用程序都可以发送广播消息(通过发送 Intent 对象), 广播接收器的功能是对广播消息进行过滤并进行相应的响应。很多广播来自系统代码, 例如: 通知开机完成、电池电量低、收到电话或短信、时

收稿日期: 2016-07-20; 修回日期: 2016-09-28

基金项目: 国家自然科学基金资助项目(No.61170282)

Foundation Item: The National Natural Science Foundation of China(No.61170282)



区变更、网络状态改变等,使用户可以及时做出反应。应用程序也可以发送广播,如通知其他应用程序某个文件已经下载成功。

Android 中的广播机制是非常有用的,许多原本需要开发者实现的操作只需等待广播通知即可,有效地减少了开发者的工作负担。Android 的广播机制使用观察者模式,基于消息发布和事件订阅模型,将广播的发送者和接收器极大程度地解耦,使系统能够更加方便地集成和扩展。

广播接收器不提供用户界面,因此不能与用户直接交互。如果发生了广播事件,广播接收器可以启动一个 activity(活动)响应它们收到的信息;或者通过 notification manager(通知管理)通知用户。通知有多种方式:背景灯闪烁、设备震动、发出声响等。通常,在状态栏放置一个持久图标,用户可以打开从而获取广播消息。

广播事件通过 Intent 动作(action)指明和激活。表 1 是一些系统的广播事件举例。

表 1 一些系统的广播事件举例

动作	说明
action_battery_charged	充电状态或电池电量改变
action_battery_low	电池电量低
action_boot_completed	系统完成引导过程,只广播一次
action_package_added	设备上安装了一个新的应用程序包
action_package_changed	已有的应用程序包发生变化
action_package_removed	已有的应用程序包已从设备中清除
action_timezone_changed	时区已经改变
action_time_changed	时间已经设置
action_time_tick	当前时间已经改变
action_wallpaper_changed	当前壁纸已经改变

近年来,Android 安全方面的文献数量^[1-5]大幅增长,包括 Android 组件安全。但是,唯独广播接收器安全方面的文献数目很少,仅散见于博客和技术论坛,如参考文献[6]等,少有严谨和系统的研究。某些相关的文献涉及短信安全^[7,8]。事实上,Android 广播机制的构造与特定逻辑形成了不少安全盲点,可能被黑客利用。本文将对此进行全面而深入的分析。

2 Android 的广播机制

2.1 Android 广播机制的参与角色和 AMS

Android 的广播机制有两个参与角色:广播发送者和

广播接收器。Android 的广播机制通常涉及跨进程通信,由 AMS(activity manager service)进行统一管理。其中,AMS 是系统启动时加载的一个服务线程,运行于 system_progress 进程中,主要负责管理系统中的组件运行。广播的发送流程是从广播发送者到 AMS,然后从 AMS 到广播接收器,以上两步都是通过 Android 的 Binder 进程间的通信机制实现的。具体地说,广播发送者将广播发送到 AMS,收到特定的广播后,AMS 在自身的注册中心中查询哪些广播接收器订阅了该广播。AMS 查找符合相应过滤条件(intent filter/permission 等)的广播接收器,然后将广播分别发送给它们。注意,广播的发送、接收和处理是异步实现的,即 AMS 不等待广播接收器接收和处理完广播后就会返回。

2.2 Android 系统的广播接收器类型

根据广播接收器的接收方式,Android 系统的广播接收器主要分为以下 3 类,如图 1 所示。

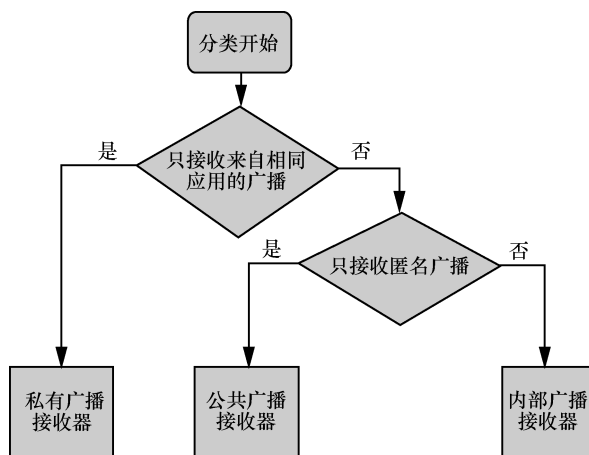


图 1 Android 系统的 3 类广播接收器

(1)私有广播接收器

只能接收来自相同应用程序发送的广播,因此是最安全的广播接收器。

(2)公共广播接收器

可以接收来自所有应用程序发送的广播。

(3)内部(in-house)广播接收器

只能接收内部应用程序发送的广播。

根据定义方式,广播接收器可以分为 2 种类型:静态广播接收器和动态广播接收器。静态广播接收器,不能接收某些系统发送的广播,如 action_battery_charged;应用程序从初始引导直到卸载为止都可以接收广播。动态广播接

收器,可以接收静态广播接收器不能接收的广播;应用程序可以控制接收广播的周期,仅当 activity 在前台时才能接收广播;不能建立私有广播接收器。

2.3 创建广播接收器对象

创建广播接收器对象时,需要继承 android.content.BroadcastReceiver 类,并重写其 onReceive(Context, Intent)回调方法。例如:创建一个名为 NewReceiver 的广播接收器的方法如下所示。

```
package com.ABC.receiver;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.util.Log;
public class NewReceiver extends BroadcastReceiver {
    private static final String TAG = "NewReceiver";
    @Override
    public void onReceive(Context context, Intent intent){
        String msg = intent.getStringExtra("msg");
        Log.i(TAG, msg);
    }
}
```

在 onReceive(Context, Intent)方法中,可以获取广播中 Intent 包含的数据,取得有用的信息。创建了该 BroadcastReceiver 之后,还需要为它注册。只有向 Android 系统注册了广播接收器组件,才能接收特定的广播消息。

2.4 广播接收器的注册方式

广播接收器的注册方式有两种:静态注册和动态注册。其中,静态注册由 PMS(package manage system)管理,动态注册由 AMS 管理。其中,PMS 是开机时由 systemServer 启动的分组管理服务,包括分组的安装、卸载和管理等。参与注册的两个角色是 BroadcastReceiver 和 IntentFilter,IntentFilter 用来决定需要接收什么广播。

2.4.1 静态注册模式(默认注册方式)

在 AndroidManifest.xml 清单文件中通过<receiver>标签声明和注册。例如:闹钟(alarm)应用程序的广播接收器 NewReceiver 希望能够接收 action_time_changed 和 action_time_zone_changed 系统广播事件,就必须在 Android Manifest.xml 清单文件中注册 NewReceiver,如下所示。

```
<receiver android:name = ".NewReceiver">
    <intent-filter>
```

```
<actionandroid:name="android.intent.action.TIME_ CHAN
GED"/>
```

```
<actionandroid:name="android.intent.action.TIME_ZONE_
CHANGED"/>
```

```
</intent-filter>
```

```
</receiver>
```

静态注册的广播接收器是常驻型的,在 Android 3.1 以前,当应用程序关闭或者停止后,如果有广播事件发生,广播接收器仍然会启动并接收广播消息。

从 Android 3.1 开始,上述情况发生了变化,即 Intent 增加了两个新的 flag 参数:flag_include_stoped_packages 和 flag_exclude_stoped_packages,用来追踪应用程序当前是否处于运行状态。在发送广播时,系统默认直接增加了值为 flag_exclude_stoped_packages 的 flag,防止广播无意或不必要地开启未启动 App 的后台服务。

2.4.2 动态注册模式

动态注册是非常驻型的,如果广播接收器的宿主程序的生命周期结束,则该接收器也会相应地结束。此外,动态注册的广播接收器还会受其宿主组件状态的影响,如果宿主组件(activity 或 service)被关闭,则该接收器也会随之失效。

动态注册时,无需在 AndroidManifest.xml 清单文件中注册广播接收器组件,而是直接用在代码中调用 Context.registerReceiver()方法进行注册。例如:

```
NewReceiver receiver = new NewReceiver();
IntentFilter filter = new IntentFilter();
filter.addAction("android.intent.action.Battery_changed");
registerReceiver(receiver, filter);
```

当实例化 NewReceiver 的宿主组件时,NewReceiver 会被动态地注册到系统中。当该宿主组件销毁时,动态注册的 NewReceiver 就不能再接收到相应的广播消息。

注意,如果在 activity 或 service 中注册了一个广播接收器,但该 activity 或 service 被销毁时未解除广播接收器的注册,系统会报异常。因此,需要进行下述操作。

```
@Override
public void onDestroy() {
    super.onDestroy();
    unregisterReceiver(receiver);
}
```



2.5 广播的发送/接收和广播接收器的生命周期

将 Intent 对象传递给下述方法:Context.sendBroadcast(), Context.sendOrderedBroadcast()或 Context.sendStickyBroadcast(),可以发送广播。

当广播接收器监听到发送的广播时,Android 调用 onReceive(Context, Intent) 回调方法,将包含广播消息的 Intent 对象传递给该方法。但 onReceive() 的执行时间不能过长,否则当超出阈值(通常为 5 s)后会弹出超时对话框。当广播接收器执行 onReceive() 方法时,广播接收器处于活跃状态;当 onReceive() 返回时,广播接收器处于不活跃状态。因此,从调用 onReceive(Context, Intent) 开始到结束构成了广播接收器的生命周期。亦即,仅在调用 onReceive() 方法期间,BroadcastReceiver 对象是有效的。一旦代码从该函数返回,系统就视广播接收器对象终止,不再活跃。

由于 BroadcastReceiver 仅在响应广播消息时才是活跃的,所以无需显式地关闭这类组件。因此,BroadcastReceiver 只有 void onReceive(Context curContext, Intent broadcastMsg) 这种回调方法。

运行活跃 BroadcastReceiver 的进程是受保护的,防止被杀死,但 Android 在任何时刻都可以杀死运行不活跃 BroadcastReceiver 的进程,释放内存空间给其他进程。这时,如果响应广播消息的时间过长就会出现问題,因此需要另一个线程进行必要的工作。如果回调方法 onReceive() 孕育一个线程然后返回,则包括新线程在内的整个进程都被视为不活跃进程(除非该进程中存在其他活跃组件),使其处于随时可能被终止的境地。解决这个问题的方法是回调方法 onReceive() 启动一个执行该任务的 service,使 Android 系统知道该进程还有需要完成的活跃工作。

2.6 Android 系统的广播类型

根据接收广播是否有序,广播可以分为普通广播(normal broadcast)和有序广播(ordered broadcast)两种类型。

(1) 普通广播

对于多个广播接收器来说是完全异步的,亦称为无序广播。无序广播并不是真的没有顺序,普通广播的接收器也要排队等待广播,只不过在广播传递的过程中,每一个广播接收器都必须遵守规则,将消息传递给下一个接收器,直到最后一个接收器收到广播。但是,因为执行 onReceive() 的过程是异步的,所以从执行结果无法得

到接收顺序的规律。对于这种广播,接收者无法终止广播,即无法阻止其他接收者的接收动作。普通广播是通过调用 Context.sendBroadcast() 方法发送的,其优点是效率很高;缺点是缺乏灵活性,即 BroadcastReceiver 无法使用 setResult() 系列函数、getResult() 系列函数及 abort() 函数的 API。典型的普通广播是开机启动广播 boot_completed。

(2) 有序广播

通过调用方法 Context.sendOrderedBroadcast 发送,BroadcastReceiver 按照队伍中的先后顺序依次接收,接收顺序由相应 Intent 过滤器中的 android:priority 属性决定。广播接收器可以通过调用 setResult() 系列函数将自定义的结果传递给下一个接收器;通过 getResult() 系列函数取得上一个接收器返回的结果;还可以通过 abort() 系列函数让系统丢弃该广播,使其他接收器不能再接收该广播,达到拦截广播的效果,甚至可以对广播进行恶意修改,使后面的接收器得到错误的参数值。典型的有序广播是短信广播。

根据广播是否在系统中长期保留,广播可以分为普通广播和黏性广播(sticky broadcast)两种类型。

当广播发送给允许接收的广播接收器之后,普通广播就自动消失。所有的广播接收器都同时接收广播。

黏性广播不消失,并在系统中保留。所以,此后调用 registerReceiver() 的应用程序可以接收黏性广播。因为黏性广播不会自动消失,所以必须显式地调用 removeStickyBroadcast() 才能删除不必要的黏性广播。此外,需要权限的受限广播接收器不能接收黏性广播。典型的黏性广播是电池状态变化通知(action_battery_changed)。

黏性有序广播同时具有有序广播和黏性广播的特点。类似于黏性广播,它不允许要求特殊权限的广播接收器接收广播。

基于不同的组合:是否有序和是否黏性,Android 系统共有 4 种不同类型的广播,见表 2。

表 2 Android 系统中的广播类型

广播类型	发送方法	是否有序	是否黏性
普通广播	sendBroadcast()	否	否
有序广播	sendOrderedBroadcast()	是	否
黏性广播	sendStickyBroadcast()	否	是
黏性有序广播	sendStickyOrderedBroadcast()	是	是

3 Android 广播机制的安全结构与特点

3.1 权限

为了保障广播机制的安全性,Android 系统为广播的发送和接收设置了系统级的权限。

3.1.1 系统权限

Android 系统和应用程序都可以对外广播 Intent 对象,但只有具备所需权限且已经注册了该 Intent 对象指定动作的广播接收器才能被激活,接收该 Intent 对象。访问权限既可以通过发送 Intent 也可以通过接收 Intent 设置。

(1) 发送 Intent 时设置权限

在 `sendBroadcast(Intent, String)` 或 `sendOrderedBroadcast(Intent, String, BroadcastReceiver, android.os.Handler, int, String, Bundle)` 中设置非空权限参数。这时,只有具有这种权限的广播接收器(在 `AndroidManifest.xml` 清单文件中通过 `<uses-permission>` 标签申请该权限)才能接收到该广播消息。例如:要接收 `boot_completed`(系统完成引导过程)广播 Intent 动作,应用程序需要具有 `receive_boot_completed` 权限。

(2) 接收 Intent 时设置权限

调用 `registerReceiver(BroadcastReceiver, IntentFilter, String, android.os.Handler)` 或静态地在 `AndroidManifest.xml` 清单文件中通过 `<receiver>` 标签,在注册时设置非空权限参数。这时,只有具有这种权限的广播(在 `AndroidManifest.xml` 清单文件中通过 `<uses-permission>` 标签申请该权限)才能发送 Intent 给广播接收器。

3.1.2 内部权限

除 Android 操作系统定义的系统级权限外,应用程序也可以定义自己的权限。通过使用内部定义的签名权限,可以建立内部应用程序的安全通信信道。这时,需要注意的关键点是:

- 定义“签名级”权限: `android:protectionLevel="signature"`;
- 两个待通信的内部程序用相同的开发者密钥签名。

3.2 Google 版本更新中的若干安全性增强措施

一般而言,应用程序内的广播比公共广播更为安全。基于此,Android v4 兼容包中给出了封装好的 `LocalBroadcastManager` 类,用于统一处理应用程序内的广播,使用方式与通常的公共广播几乎相同,只是“注册/取消注册”广播接收器和发送广播时将 `context` 变成了 `LocalBroadcastManager`。

Android 4.4 之后,新增加了一个“默认短信服务”机制——`default SMS`。亦即,如果要实现短信拦截功能,就必须成为 `default SMS`,把所有短信相关的功能全部囊括,然后才能进行短信拦截。但要成为 `default SMS`,必须进行大量适配性和兼容性工作,包括短信、彩信、单双卡等。这种做法相当于要求短信拦截类软件成为一个功能非常完善的通讯录类应用,大大增加恶意应用开发者的成本和负担。

但是,这种机制也存在安全漏洞,恶意软件仍然可以在不成为 `default SMS` 的条件下对短信进行“写”操作。攻击方法是通过 Android 4.2 后增加的 App Ops 权限管理功能,在短信的 TAB 中找到自己的应用程序,并进入相应的权限管理界面。

3.3 广播接收器的优先级

对同一个广播,不同的应用程序可以注册多个 `BroadcastReceiver` 进行接收。如果恶意程序先于正常程序抢先收到广播,就可以窃取通信数据并中断广播,造成很大的安全隐患。例如:短信广播被拦截后,被拦截的短信就不会在收件箱中出现,也不会有来信提醒。更严重的是,用户对此毫无所知。

因此,无论从安全或者功能的角度,还是从系统层或者应用层的角度,了解广播优先级的形成规律,降低广播拦截风险是一个重要课题。

广播的分发过程是异步的,由 AMS 进行管理。AMS 收到广播后,根据不同类型的广播接收器,按照广播发送者指定的发送模式进行分发。具体地说,AMS 向自己的消息队列发送消息 `broadcast_intent_MSG`,向所有的接收器进行分发操作,所有的分发操作都由 `processNextBroadcast()` 方法实现。

Google 的官方文档没有说明广播接收器的接收顺序,研究表明,Android 原生系统中广播接收器的接收顺序如图 2 所示。

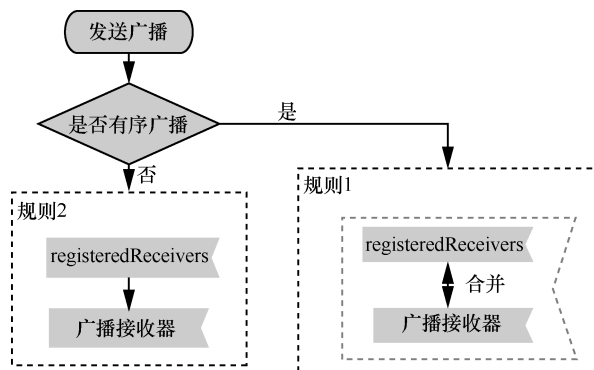


图 2 广播接收器的接收顺序



其中,规则 1 是:拥有最高 priority 属性值的接收器最先收到广播;priority 属性值相同时,动态广播先于静态广播接收;属性值相同的动态广播,接收顺序等同于注册顺序;属性值相同的静态广播,接收顺序同样等同于注册顺序,即文件扫描顺序。

规则 2 是:动态广播优先于静态广播;对于同种类型的广播,拥有较高 priority 属性值的优先级较高;属性值相同的动态广播,接收顺序等同于注册顺序;属性值相同的静态广播,接收顺序同样等同于注册顺序,即文件扫描顺序。

注意,不同版本的 Android 系统,文件扫描顺序不同。

这里需要指出,普通广播的分发过程大多是异步的,很难从实验结果验证广播优先级的形成规律。此外,前一个接收器对下一个接收器接收广播的影响不大,既不能拦截也不能篡改结果,所以研究聚焦于有序广播的接收器优先级问题。

3.4 其他相关的安全结构与安全相关问题

3.4.1 android:exported 属性的设置

在 Android 系统中,android:exported 属性决定广播接收器能否接收其他应用程序发出的广播,这个属性的默认值是由接收器中是否有 intent-filter 决定的:如果有 intent-filter,默认值为 true,否则默认值为 false。

3.4.2 AMS 与 Binder 机制安全

前文提到过,广播的发送是从广播发送者到 AMS,然后从 AMS 到广播接收器,以上两步都是通过 Android 的 Binder 进程间的通信机制实现的。可见,AMS 与 Binder 机制的安全是十分重要的。

首先,AMS 作为广播发送的路由器,保存了每个广播接收器的信息和次序,如果恶意软件能够渗透到 AMS 内部,就有可能在发送广播之前确定广播接收器的接收顺序,从而从源头获得对广播的优先抢占。

其次,如果黑客可以攻陷 Binder 进程间通信机制,也可以收到类似的效果。

Binder 架构通信是 C/S(客户端/服务端)模型。客户端发起通信并等待服务端的响应。Binder 架构使用客户端代理(proxy)进行通信。在服务器端,通过线程池响应客户端的通信请求。客户端进程的代理对象实现与 Binder 内核驱动的通信。另一个进程是具有多 Binder 线程的服务器进程。Binder 框架生成大量新线程处理通信请求,直至达到预定义的最大线程数。代理对象与 Binder 驱动打交道,后者将信息交付给目的对象。

Binder 框架具有 3 层结构:高层是 Android 应用程序的接口层;中间层是 Binder 框架在用户空间实现的中间层;底层是内核驱动层。

关于 Binder 机制更详细的描述及其安全性,已经超出本文的范围,不再赘述。

3.4.3 不同类型广播接收器的特点

(1)私有广播只有应用程序内部才能够接收,具有下述特点。

- 在 3 种类型的广播中,安全性最高。
- 动态广播接收器不能注册为私有接收器,因此私有广播接收器只能是静态广播接收器。
- 不定义 Intent 过滤器,显式地设置 exported 属性为 false。
- 即使 Intent 是相同应用程序发送的,仍需安全地处理收到的 Intent。

(2)公共广播可以由任意应用程序发送和接收,具有下述特点。

- 可同时用于静态广播接收器和动态广播接收器。
- 在 3 种类型的广播中,安全性最低。
- 显式地设置 exported 属性为 true。
- 因为广播可被恶意软件接收,所以必须安全地处理收到的 Intent。并且,返回结果时不要包含敏感信息。

(3)内部广播只能由内部应用程序发送和接收,其他应用程序都不能发送和接收。内部广播接收器具有下述特点。

- 可同时用于静态广播接收器和动态广播接收器。
- 安全性低于私有广播,但高于公共广播。
- 通过定义内部签名权限发送和接收广播。
- 显式地设置 exported 属性为 true。
- 静态或动态注册内部广播接收器时,必须声明签名权限。
- 验证内部签名权限是由内部应用程序定义的。

4 Android 广播机制的安全挑战与对策

4.1 Android 广播机制的安全挑战

在分析 Android 广播机制的安全挑战时,应当考虑以下几个方面。

4.1.1 Android 广播机制面临的安全挑战

概括地说,Android 面临下述 4 个方面的安全挑战。

- 敏感广播消息的泄露。
- 恶意软件拦截广播,终止广播的传送。
- 恶意软件修改广播消息,发送虚假广播。
- DoS 攻击:不断收到广播并处理,消耗手机资源。

4.1.2 Android 广播机制的攻击面

概括地说,Android 广播机制的攻击面包括如下几个方面。

- 针对系统内核层的攻击,包括利用操作系统的安全缺陷进行漏洞利用。
- 针对 AMS 和 Binder 机制的攻击。例如:通过注入技术入侵 AMS 的宿主进程 system_server,控制广播接收顺序。
- 针对 Intent 和 Intent 过滤器的攻击。例如:其他应用程序针对性地发出与当前应用的 Intent 过滤器相匹配的广播,致使当前应用不断地接收广播并处理,形成 DoS 攻击。又如,其他应用程序注册与当前应用一致的 Intent 过滤器,用于接收相同的广播,致使广播消息泄露。
- 针对应用程序级的攻击。例如:没有指定 receiverPermission 参数的有序广播面临异常终止的威胁。这时,包括恶意软件在内的大量应用程序都可以收到广播。因为有序广播根据优先级次序发送,所以具有高优先级的恶意软件可以先收到广播并执行 abortBroadcast()方法,使其他接收器收不到广播。

4.1.3 不同类型的广播与广播接收器面临不同的安全风险

普通广播、有序广播和黏性广播面临不同的安全风险。例如:黏性广播具有敏感信息泄露的风险;有序广播具有被拦截的风险等。

不同类型的广播接收器安全性不同,因此具有不同程度的安全风险。其中,私有广播接收器的安全性最高;公共广播接收器的安全性最低;二者之间是内部广播接收器。

4.2 Android 广播机制的安全对策

针对上述 Android 广播机制的安全挑战分析,可以得出下述安全对策。

(1)当通过广播发送敏感信息时,尽可能地限定(通过 Intent 或内部定义的签名权限)可接收的广播接收器,仅使可信的广播接收器才能接收。

(2)黏性广播(包括黏性有序广播)中不能包含敏感信

息。Android 系统中,黏性广播是通过隐式 Intent 使用的,指定 receiverPermission 参数的广播无法发送。所以,包括恶意软件在内的大量应用程序都可以收到黏性广播,因此黏性广播中不能包含敏感信息。

(3)内部广播接收器只接收内部应用程序发送的广播,仅靠 Android 系统级权限保护是不够的,应当根据第 3.1.2 节所述,进一步由内部定义的签名权限进行保护。

(4)仅用于某个特定应用程序的广播接收器必须设置为私有广播接收器,避免收到来自其他应用程序的广播。这时,不能在 AndroidManifest.xml 清单文件中设置 Intent 过滤器,如下例所示。如果这样设置,则其他应用程序的公共广播接收器仍然有可能收到广播。

```
<!-- 不能如下配置私有广播接收器 -->
<!-- 设置 intent-filter 是不可取的 -->
<receiver
    android:name=".PrivateReceiver"
    android:exported="false">
    <intent-filter>
        <action android:name="com.ABC.android.broadcast.
MY_ACTION" />
    </intent-filter>
</receiver>
```

私有广播是 Android 广播机制中最安全的,因此可以发送敏感信息。但是,仍存在安全挑战。如下例所示,需要注意如下几方面。

- 显式地设置 exported 属性为 false。
- 即使 Intent 是相同应用程序发送的,仍需安全地处理收到的 Intent。
- 使用指定类名的显式 Intent 调用相同应用程序中的广播接收器。

```
public class PrivateSenderActivity extends Activity {
    // 发送普通广播
    public void onSendNormalClick(View view) {
        //使用指定类名的显式 Intent 调用相同应用程序中的广播接收器
        Intent intent = new Intent(this, PrivateReceiver.class);
        // 因为广播接收器在相同应用程序中,所以可以发送敏感信息
        intent.putExtra("PARAM", "来自广播发送者的敏感信息");
    }
}
```



```
sendBroadcast(intent);
}
// 发送有序广播
public void onSendOrderedClick(View view) {
    // 使用指定类名的显式 Intent 调用相同应用程序中的广播接收器
    Intent intent = new Intent(this, PrivateReceiver.class);
    // 因为广播接收器在相同应用程序中,所以可以发送敏感信息
    intent.putExtra("PARAM", "来自广播发送者的敏感信息");
    sendOrderedBroadcast(intent, null, mResultReceiver, null, 0, null, null);
}
```

(5)安全地处理收到的 Intent。尽管安全威胁的程度取决于广播接收器的类型,首要的是在处理收到的 Intent 数据之前先验证 Intent 的安全性。

公共广播接收器可能收到恶意软件发送的 Intent。私有广播接收器不可能直接收到来自其他应用程序的 Intent,但公共组件收到的来自其他应用程序的 Intent 数据可能会转发到私有广播接收器。因此,不能认为私有广播接收器收到的 Intent 是无条件安全的。内部广播接收器具有一定程度的安全风险,因此必须验证收到的 Intent 的安全性。

(6)当返回结果信息时,防止目标应用程序泄露结果信息。通过 setResult()返回结果信息的应用程序的可靠性取决于广播接收器的类型。在公共广播接收器情形,目标应用程序可能是恶意软件,所以结果信息可能会被恶意使用。在私有广播接收器和内部广播接收器情形,目标应用程序是内部应用程序,所以无需特别处理。

(7)安全地处理来自广播接收器的结果数据。尽管安全威胁的程度取决于返回结果数据的广播接收器类型,由于收到的结果可能是攻击数据,因此必须安全地处理结果数据。

当发送(源)广播接收器是公共广播接收器时,收到的返回数据可能是恶意软件的攻击数据。当发送(源)广播接收器是私有广播接收器时,似乎没有风险。但是,其他应用程序收到的数据可能作为结果数据间接地被转发。因此,不能认为结果数据是无条件安全的。当发送(源)广播接

收器是内部广播接收器时,具有一定程度的安全风险。由于结果数据可能是攻击数据,所以必须安全地处理结果数据。

(8)提高广播接收器的优先级,降低自身广播被拦截的风险。可以采用的方法如下。

- 提高 BroadcastReceiver 的 priority 属性值。priority 的默认值为 0,最大值为 Integer.max_value。
- 尽可能采用动态方式注册 BroadcastReceiver。
- 注册时间越早越好。

基于以上考虑,可以进行如下设置。

- 监听 Android 的开机广播事件 boot_completed,提早注册时间。
- 启动后开启一个 service,并在其中动态注册 BroadcastReceiver。
- 将 BroadcastReceiver 的 priority 属性值设为最大,即 Integer.max_value。

此外,关于“注册时间越早越好”,还可以增加几个额外的触发点,例如:wifi_state_changed、connectivity_change、access_network_state 和 access_wifi_state。上述系统事件,都比 boot_completed 触发得早。

(9)防止利用注入技术,注入 AMS 的宿主进程 system_server,从中截获 AMS 的 broadcastIntent()方法,插入黑客的攻击代码之中,调整广播接收器的队列顺序,取得对广播的绝对控制。

5 结束语

近年来,针对 Android 组件安全的研究大幅增长。但令人吃惊的是,尽管广播接收器是 Android 的四大组件之一,在 Android 应用程序之间的交互中扮演重要角色,却鲜少针对 Android 广播机制安全性的研究,更缺乏系统性的深入研究。

有鉴于此,本文对 Android 的广播机制及其安全性进行了全面的分析,研究结果表明 Android 广播机制的安全隐患是实实在在存在的。在实践中,针对 Android 广播机制的攻击也无处不在。例如:“10086”诈骗的流程是:黑客通过伪基站批量发送伪造的 10086 短信,诱使用户点击钓鱼链接。上当的用户会在 Android 手机的钓鱼页面输入网银账号和密码,并下载安装“伪中国移动客户端”手机病毒。中招后,该病毒会在后台监控用户短信内容,拦截短信,并获取网银验证码,最后窃取用户的网银资金。

可以预见,今后针对 Android 广播机制的攻击不会减少,而且会日益增加,并且攻击的手法会更加先进和复杂。与此同时,应当对 Android 广播机制及其安全性进行深层次的挖掘。例如:分析 PMS 的构造函数及其详细的工作流程,包括:建立 AppDirObserver;监听/system/framework、/system/app、/vendor/app、/data/app、/data/app-private 等目录的增加(add)和删除(delete)等操作;调用 scanDirLI()函数扫描上述目录的过程和结果以及 parseApplication()和 parsePackage()等关键函数;分析 AMS 及其关键函数 broadIntentLocked()等,在此基础上提出更加有针对性的安全防御措施。

参考文献:

- [1] ENCK W, ONGTANG M, MCDANIEL P. Understanding Android security[J]. IEEE Security & Privacy, 2009, 7(1): 50-57.
- [2] ZHANG M, YIN H. AppSealer: automatic generation of vulnerability-specific patches for preventing component hijacking attacks in Android applications [C]//The 21st Network and Distributed System Security Symp (NDSS 2014), February 23-26, 2014, San Diego, California, USA. [S.l.:s.n.], 2014.
- [3] OCTEAU D, MCDANIEL P, JHA S, et al. Effective inter-component communication mapping in Android with epicc: an essential step towards holistic security analysis[C]//The 22nd USENIX Security Symp (USENIX 2013), August 14-16, 2013, Washington, DC, USA. New York: ACM Press, 2013.
- [4] CUI X M, YU D, CHAN P, et al. CoChecker: detecting capability and sensitive data leaks from component chains in Android[C]//The 19th Australasian Conf. (ACISP 2014), July 7-9, 2014, Wollongong, Australia. [S.l.:s.n.], 2014: 446-453.
- [5] 卿斯汉. Android 安全研究进展 [J]. 软件学报, 2016, 27(1): 45-71.
QING S H. Research progress on Android security[J]. Journal of Software, 2016, 27(1): 45-71.
- [6] 杜文涛. Android 组件研究—广播接收器[EB/OL]. [2016-08-05]. <http://blog.csdn.net/windskier/article/details/7251742>.
DU W T. Research on Android component-broadcast receiver [EB/OL]. [2016-08-05]. <http://blog.csdn.net/windskier/article/details/7251742>.
- [7] HAMANDI K, CHEHAB A, ELHAJJ I H, et al. Android SMS malware: vulnerability and mitigation[C]//The 27th International Conference on Advanced Information Networking and Applications Workshops, March 25-28, 2013, Barcelona, Spain. New Jersey: IEEE Press, 2013.
- [8] KIM D, RYOU J. SecureSMS: prevention of SMS interception on android platform [C]//The 8th International Conference on Ubiquitous Information Management and Communication (ICUIMC '14), Jan 9, 2014, Seam Reab, Cambodia. New York: ACM Press, 2014: 485-487.

[作者简介]



卿斯汉(1939-),男,中国科学院软件研究所首席研究员,北京大学软件与微电子学院教授,主要研究方向为移动安全、可信计算、云安全、操作系统安全等。