文章编号: 2095-6134(2014)06-827-09

基于 Fuzzing 的 Android 应用通信过程漏洞挖掘技术*

王 凯, 刘奇旭, 张玉清

(中国科学院大学 国家计算机网络入侵防范中心, 北京 101408)

(2013 年 9 月 27 日收稿; 2014 年 1 月 3 日收修改稿)

Wang K, Liu Q X, Zhang Y Q. Android inter-application communication vulnerability mining technique based on Fuzzing [J]. Journal of University of Chinese Academy of Sciences, 2014, 31(6): 827-835.

摘 要 在通信过程中,如果 Android 应用对其私有组件保护不充分,会导致组件暴露漏洞的存在. 以往针对 Android 应用通信过程的漏洞挖掘方法不能准确发现这种安全威胁. 为解决上述问题,提出一种结合 Fuzzing 技术和逆向分析的漏洞挖掘方法,设计并实现了漏洞挖掘工具 KMDroid. 实验结果表明,KMDroid 可以有效挖掘应用通信过程中存在的安全漏洞.

关键词 Android; Fuzzing; 逆向分析; 应用通信; 安全漏洞

中图分类号: TP393.08 **文献标志码:** A **doi:** 10.7523/j.issn.2095-6134.2014.06.015

Android inter-application communication vulnerability mining technique based on Fuzzing

WANG Kai, LIU Qixu, ZHANG Yuqing

(*National Computer Network Intrusion Protection Center, University of Chinese Academy of Science, Beijing* 101408, *China*)

Abstract If an Android application could not protect its private components well in the process of inter-application communication, there would exist exposed component vulnerabilities. The current vulnerability mining technique cannot identify such vulnerabilities accurately. To solve this problem, we propose a new vulnerability mining method which combines Fuzzing with reverse analysis, and design a vulnerability mining tool named KMDroid. Experimental results show that KMDroid can discover the vulnerability of inter-application communication effectively.

Key words Android; Fuzzing; reverse analysis; inter-application communication; vulnerabilities

Android 是一款由 Google 公司发布的智能移动操作系统. 2013 年第一季度数据显示,Android 设备占据了智能移动设备总销量的近 75%,成为全球最受欢迎的智能移动操作系统[1]. 截止到 2013 年 7 月中旬,Android 官方应用市场"Google Play"的应用已超过 75 万款[2]. 在 Android 拥有巨大数量的用户和应用的同时,由于应用开发入门门槛较低、从 Google Play 上获取官方开发者签名较为容易、以及 Google Play 没有采取严格的安全审核等一系列原因,导致 Android 应用面临较为

* 国家自然科学基金(61272481,61303239)、北京市自然科学基金(4122089)、国家发改委信息安全专项(发改办高技[2012]1424 号)和中国科学院大学校长基金资助

† 通信作者, E-mail: wangk@nipc.org.cn

严峻的安全问题[3-4].

在Android诸多安全研究领域中,关于应用通信过程的安全研究受到研究者的广泛关注. Android应用通常由多个构件组成[5],不论发生在应用内还是应用间的组件通信都主要以Intent作为通信媒介. 应用功能组件化实现了手机内程序功能的复用,例如,音乐播放器不需要实现音频文件的解码,只需调用系统提供的音乐播放服务模块即可. 这些组件间的通信机制,简称ICC(inter-component communication)[6],在没有安全防护的情况下,允许任意应用发送消息给目标组件,这极有可能导致应用功能泄露. 在Android 2.1到2.3.1版本中,攻击者只需发送一条消息给系统应用"设置"中的一个组件,就能在没有授权的情况下开启蓝牙、Wi-Fi、GPS等[7]. 2012年至今,就有5个此类应用漏洞被CVE收录(CVE-2012-4005、CVE-2012-4905、CVE-2012-5182、CVE-2013-0122、CVE-2013-3579),这些漏洞严重威胁着用户的信息安全. 但是ICC机制中潜在的安全问题并没有得到开发者的足够重视,大量漏洞仍隐藏在应用通信过程中. 因此,有针对性地研究应用通信过程中的漏洞挖掘技术是必要的.

针对以上问题,基于Fuzzing漏洞挖掘技术[8-9],以ICC机制为消息测试渠道,本文设计了一种新的测试方法,并开发了KMDroid工具. 关键思路在于在测试数据生成过程中,利用逆向工程和源码分析[10],得到目标应用处理消息时使用的自定义名称,从而可以构造出有效的测试数据[11],注入到应用运行过程中. 本研究方法不仅能发现目标组件在处理畸形消息的过程中抛出异常的情况,还能研究在测试数据注入之后应用所暴露的界面和功能. 本文通过研究分析这些安全漏洞,展示应用通信过程中,可能存在的安全漏洞及易于遭受的攻击类型,为应用开发和Android系统改进提供参考和建议.

1 应用通信过程安全性分析

1.1 应用组件化及组件间的通信媒介

Android应用组件分为Activity、Service、Broadcast Receiver和Content Provider共4种类型. 每个应用可由不同类型的多个组件组成.

- Activity: 应用界面,负责与用户进行交互.
- Service: 后台处理程序,也可作为应用的守护程序在开机时启动,用来执行一些需要持续运行的操作.
- Broadcast Receiver: 监听符合特定条件的消息广播,是事件驱动程序的理想手段.
- Content Provider: 使用相应的数据接口存储和共享数据,往往通过Uri来实现对该组件的定位.

应用及其组件的特性描述存储在名为AndroidManifest[12]的XML文件中,以便系统在应用安装和运行的过程中获取一些必要的信息. 该文件也为KMDroid测试应用组件提供了有用信息.

除了Content Provider外,其他3种组件之间的通信都需要Intent消息来实现. Intent是ICC过程的消息传递媒介,其具体内容如表1所示. Intent消息既可通过设定Component name来显示指定目标组件的名称; 也可设定action、category等描述目标组件特性的标签,交给系统判断由哪个组件处理这个消息. 如果在AndroidManifest文件中应用组件声明的子元素<intent-filter>中的action、category标签与Intent消息所携带的相匹配,那么这个组件便可以接收该Intent消息. 通过第2种方式,在调用其他组件完成功能时,实现组件间的解耦.

表1 Intent中的标签

Table 1 Attributes in Intent

标签	内容描述
action	操作动作
category	动作分类
data	动作涉及数据
extra	附加数据
component name	目标组件名称

1.2 应用通信过程的安全机制

1.2.1 组件的私有化

通过设置AndroidManifest文件中组件的android: exported标签为false,可以使组件只能接收来自本应用内的消息,而其他应用发来的消息则不被接收,实现组件私有化. 如果应用组件在AndroidManifest中设置了<intent-filter>子元素,表示这个组件希望能被其他组件调用,因此android: exported的缺省值为true; 如果不包含<intent-filter>子元素,则android: exported的缺

省值为 false.

1.2.2　Permission 机制

Android 应用在申请使用网络数据、蓝牙、电话等功能时,需要在 AndroidManifest 文件中包含一个或更多的 < uses-permission > 标签来声明所要使用的权限[13]. 安装过程中,程序安装器会显示待安装应用所申请的高安全级别的权限,在获得用户的许可后,才能拥有对应权限. 应用组件也可以自定义权限,如果要调用该组件,则需要申请权限; 否则不能开启目标组件.

1.2.3　应用通信过程的安全机制潜在问题

在开发者安全意识良好的情况下,开发过程中实现以上 2 种安全机制可以较好地保障组件间通信时的安全. 但这 2 种安全机制是复杂而脆弱的. 开发者错误使用 < intent-filter > 子标签可能导致组件的暴露; 用户在安装应用时疏忽权限列表,可能会导致应用组件自定义权限的泄露.

引言中提到的系统原生应用“设置”中的“com. android. settings. widget. SettingsAppWidgetProvider”组件,就暴露了功能组件,导致攻击者通过图 1 所示的小段代码就可发送 Intent 消息来打开 Wi-Fi 而不需要申请使用任何权限.

```
Intent intent = new Intent("test");
intent.setClassName("com.android.settings",
        "com.android.settings.widget.SettingsAppWidgetProvider");
intent.addCategory(Intent.CATEGORY_ALTERNATIVE);
intent.setData(Uri.fromParts("0", "0", "0")); // 0 is for WiFi
sendBroadcast(intent);
```

图 1　发送 Intent 消息开启 WiFi 的代码

Fig. 1　Codes that open WiFi by sending intent

如果没有恰当的保护,应用组件使用的功能越强,可能泄露的功能也越多,造成的危害也就越大. 像图 1 中设置 Intent 消息的 data 项,或设置 Intent 消息的 Extra 项,都可向目标组件中注入数据. Data 项通常以 Uri 的形式设定,而 Extra 项则是用户自定义的数据,由表 2 中的 3 个要素组成,可以携带的数据类型更为丰富. 这 3 个要素可以通过分析目标组件在接收过程中的源码来获取,这导致目标组件很容易受到 Extra 项的消息注入攻击.

表 2　Extra 项 3 要素

Table 2　Three elements in Extra

要素	描述
Type	数据类型
Name	数据名称
Value	数据值

2　KMDroid 工具的设计与实现

根据对应用通信过程的安全性分析,本文设计了基于 Fuzzing 的漏洞挖掘工具 KMDroid. 图 2 中展示了 KMDroid 的架构图. KMDroid 以 Java 为开发语言,运行于计算机端,由两大功能组件组成,分别实现了: 逆向分析和测试命令生成器. KMDroid 生成命令之后,通过 Android 的 ADB Tools[14] 发送给被测设备,由系统自带的 am 命令解释并发送 Intent 消息给目标组件,目标组件的运行结果通过 Logcat[15] 工具捕获并存储到计算机端. 这种设计使测试数据的生成、测试结果的统计等操作都可以在计算机端完成,减轻了 Android 设备在测试过程中的负载.

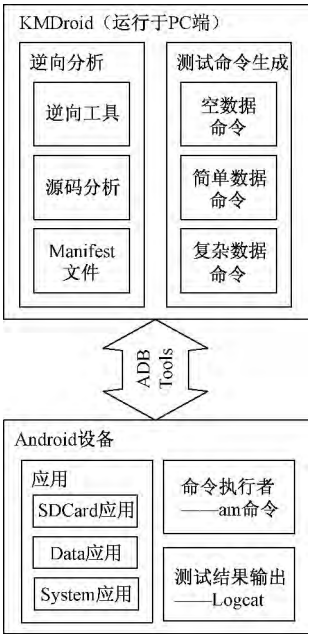

图 2　KMDroid 系统架构

Fig. 2　System architecture of KMDroid

利用 KMDroid 针对一个目标应用的测试流程如图 3 所示. 在获取应用 APK 文件之后,首先通过逆向工程获得源码,然后分析 AndroidManifest 文件即可得出所有组件; 取其中

一个待测组件，分析源码，根据其处理 Intent 消息的过程来帮助构造测试消息；在消息发出后通过 Logcat 获取应用日志输出，分析其行为，并做记录、截图等处理以方便更加细致的人工分析. 测试流程的关键在于逆向分析和 Fuzzing 数据的产生. 下面就这两部分功能的实现进行详细介绍.

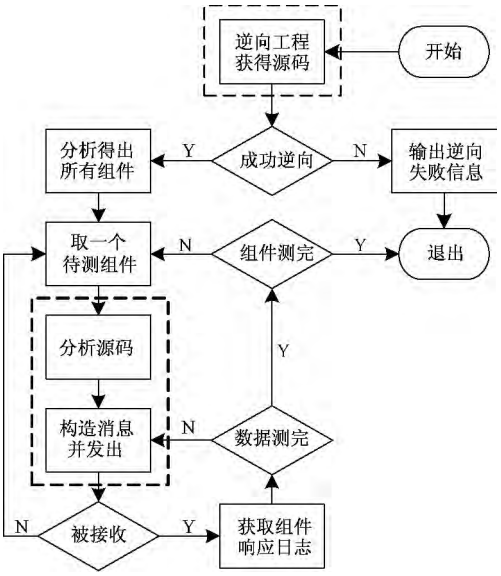

图3 应用测试流程

Fig. 3 Process of testing application

2.1 逆向分析

本文利用多种工具配合完成了对源码的分析工作. 其具体分析步骤如下.

1) 解压 APK 文件(ZIP 文件格式)，得到编码过的 AndroidManifest. xml 及二进制文件 classes. dex;

2) 利用 axmlprinter2 解码 AndroidManifest. xml，解析其中组件信息;

3) 利用 Dex2jar 工具处理 classes. dex 文件，得到 jar 文件，解压得到. class 文件;

4) 利用 jad 反编译. class 文件，得到相应的 Java 源码文件;

5) 配合 AndroidManifest 分析所得组件名称，从对应文件夹找到其源码.

Android 的 Intent 消息中所包含的 Extra 项的名称和类型都可以通过源码分析直接获得，源码分析的伪代码如图 4.

2.2 测试命令生成

经过逆向分析，可知被测目标组件是否会从 Intent 消息中获取 Extra 项数据，及 extra 的 type

```
While(Code_Not_End)
    code = get_code_at_line(line_count);
    if code_match_style(code, "*.get*Extra(*)");
        extra_type = get_string_between(code, "get", "Extra")
        extra_name = get_string_between(code, "(", ")")
        store_extra_details(extra_type, extra_name)
    end if
    line_cout++
```

图4 解析 Extra 信息的伪代码

Fig. 4 Pseudocode for parsing Extras´ information

和 name 是什么. KMDroid 使用 ADB Tools 中的 am 命令发送 intent 并支持 Int 型、String 型、Boolean 型 3 类 Extra 项的测试，在进行的初步统计中发现这 3 类涵盖了大部分情况(见 3.1.1 中的数据统计). 针对这 3 类数据，本文设计 3 种测试数据产生模式，如表 3 所示.

表3 测试数据类型

Table 3 Types of testing data

数据模式	测试数据内容	
空数据	不添加 Component name 外的任何信息	
简单数据	Int	0
	Boolean	True, False
	String	"/sdcard" "127.0.0.1", "http://www.test.com",
复杂数据	简单数据 + 从源码中解析出的该类型数据	

空数据可以用来测试目标应用是否接收外部消息，以及收到空数据时是否得到正确处理；简单数据可以简易而快速地测试大量应用；而复杂数据则是简单数据加上从利用源码中提取的相应类型数据，能够大量而细致地测试目标组件，因为被测应用可能在源码中检测消息中的数据是否和某些特定值匹配，所以源码中提取出的数据很可能构造出有效的测试数据. KMDroid 并没有使用针对长字符串、临界数据等易于产生缓冲区溢出、边界溢出的测试数据，这是由于 Android 应用的主要开发语言是 Java，而 Java 并不存在缓冲区溢出等漏洞.

在准备好测试数据之后，按照 am 命令格式设置传入参数即可(图5).

3 KMDroid 工具测试及结果分析

本次测试的平台是 Android 4.0.3 版本的华

```
start an Activity: am start [-D] [ -W] <INTENT >
        -D: enable debugging
        -W: wait for launch to complete
start a Service: am startservice <INTENT>
send a broadcast Intent: am broadcast <INTENT>
```

图 5　am 命令格式

Fig. 5　Form of am command

为 Media Pad,具体参数见表 4,测试应用通过“Google Play”官方渠道下载.

表 4　测试设备详细参数

Table 4　Details of testing device

设备名称	系统版本	内核版本	处理器	RAM
华为 Media Pad	4. 0. 3	3. 0. 8	ARM Cortex-A8	1G

测试分为 2 部分进行. 第 1 部分是利用 KMDroid 工具自动测试了 140 个应用程序,并对所得数据统计分析. 进行此部分测试的目的是: 统计 Extra 项各类数据数量,验证测试 String、Int 和 Boolean3 种数据类型的合理性; 统计各类组件被成功开启的数量,证明应用组件暴露的广泛性; 统计抛出的异常次数,初步表明组件暴露可能带来的危害. 第 2 部分则有针对性地对 27 个流行应用进行细致的测试分析,统计结果并阐述了漏洞的 7 种分类,更加深入地分析了组件暴露可能导致的安全问题.

3. 1　自动测试及数据分析

本实验对从 Google Play 中下载排行榜前 100 的 76 个应用及系统自带的 64 个应用进行了测试,共 140 个. 其中,利用逆向工程工具,成功对 131 个应用进行逆向,并根据源码获得 Intent 的消息结构,成功率约 93. 6%.

3. 1. 1　Extra 数据类型

KMDroid 在测试应用时对 Intent 消息的 Extra 类型进行了统计,以验证利用 am 命令测试 Int、String、Boolean3 类数据的合理性. 在 140 个应用中,共发现 6 315 个被处理的 Extra 项. 各种类型的具体数量见表 5.

表 5　Extra 项数据类型数量统计

Table 5　Data statistics on Extra

数据类型	Int	String	Boolean	其他(共 25 种)
数量	1 204	3 075	791	1 425

可以看出,选择这 3 类 Extra 项进行测试,在实现简单的基础上,覆盖了 Extra 数据项总数的 77. 4%,在数据类型的覆盖上较为广泛.

3. 1. 2　组件的成功开启与拒绝开启

本实验共测试 Activity 组件 3 314 个,Server 组件 374 个,Receiver 组件 282 个. 根据组件的运行结果可以分为 2 类,第 1 类是 Intent 消息顺利发出,并被接收; 第 2 类则是组件使用一定的安全策略,拒绝非授权消息来源. 本文将这 2 类分别用“Started”和“Denied”标注,如图 6 所示.

Activity 的总体组件数目最多,被开启的和被拒绝的组件大约各占一半. 此外,1 728 个被成功开启的 Activity 之中,有 339 个被展示到屏幕上; Service 总数相对较少,被成功开启的比例较多,占 62. 3%; 而 Receiver 的总数最少,但是被拒绝开启的组件仅占其总数的 2. 5%. 可以看出,有较大比例的应用组件被暴露在外,可能受到第 3 方应用的攻击.

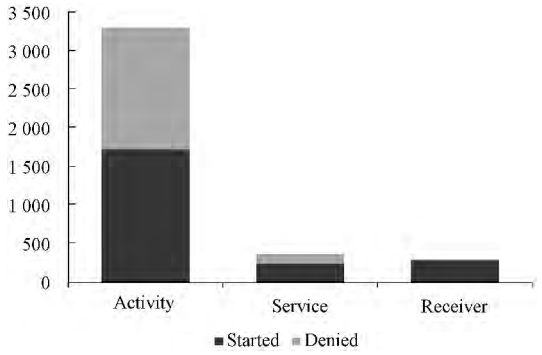

图 6　成功开启和被拒绝开启的组件数

Fig. 6　Data statistics on started and denied components

3. 1. 3　异常抛出与应用崩溃

通过畸形数据的 Intent 消息注入,KMDroid 共捕获了 8 种异常 99 次,具体分布如图 7.

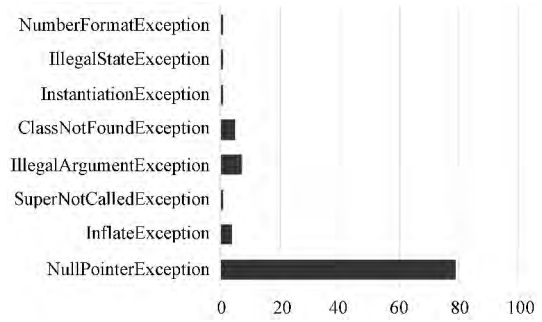

图 7　抛出异常分布情况

Fig. 7　Types and distributions of exceptions

这些异常都是被测组件运行时处理数据发生错误抛出的，被系统捕获，并导致整个应用进程的强制关闭. 攻击者可以通过发送恶意的 Intent 消息造成异常抛出、进程关闭，实现拒绝服务攻击.

可以看出，NullPointerException 占据了检测到异常的 79.8%. 为了更好地研究异常发生的原因，本文对逆向工程得到的应用源码进行分析，并发现了较有代表性的一段引发异常的代码(图 8). 可以看出，这段代码从 Intent 中获取信息后，没有检查包含该信息的对象(通常为字符串类型)是否为空，而直接调用该对象的功能函数，导致空指针异常的抛出.

```
String action = intent.getAction();
if (action.equals(
Mms.Intents.CONTENT_CHANGED_ACTION)) {
    ..... // 应用代码
}
```

图 8 导致 NullPointerException 的语句

Fig. 8 Codes that lead to NullPointerExcepiton

为更好地分析 NullPointerException 的分布，本文统计了 Activity、Service、Receiver3 类应用组件发生该异常的数量. Activity 和 Receiver 中都有较多的异常发生，分别是 49 个和 26 个，而 Service 发生异常的数量很少，仅有 1 个. 在捕获的 79 个 NullPointerException 中，有 21 个来自同一应用. 这说明，由于开发者对外来数据处理的不严谨，异常可能会集中在特定的应用上. 可以看出，异常的抛出既有一定广泛性，又有一定的集中性.

3.2 针对性测试及漏洞分析

为更加深入地分析组件暴露可能导致的安全问题，本文测试了 27 个热门应用，其中，安全管理类应用、社交类应用以及移动购物类应用各 9 个. 安全管理类应用拥有大量的高级权限，社交类应用中存在大量的隐私信息，而移动购物类与应用于用户的财产安全息息相关.

假设登录账户、同意使用协议等应用首次被使用时，需要进行的操作都已执行，并允许设备记住这些信息. 如果测试中应用界面显示隐私信息或注入的数据，则认为目标应用存在安全漏洞. 因为包含隐私信息的应用界面，可以被恶意应用通过截屏的方式截获，而包含注入数据的应用界面，则可能会被用以实现社会工程类的欺骗行为.

在 27 个被测应用中，有 11 个应用表现良好，不存在安全漏洞，其中安全管理类 6 个，社交类 2 个，移动购物类 3 个. 在其他 16 个应用中发现的安全漏洞有交易及社交隐私泄露、账户名信息泄露、文本框注入、输入框注入、开启网页、地理位置信息泄露、特殊功能界面暴露等类型. 具体统计数据见表 6.

表 6 漏洞分类及数量统计

Table 6 Statistics on different vulnerabilities

漏洞类型	存在漏洞应用数量	存在漏洞组件数量
交易及社交隐私泄露	5	12
账户名信息泄露	6	8
文本框注入	4	8
输入框注入	4	5
开启网页	3	4
地理位置泄露	2	2
其他	4	6

·交易及社交隐私泄露

主要存在于社交类和移动购物类应用中. 攻击者可以向目标应用发送 Intent 消息，使应用界面组件显示到前台; 而且这些交易或社交信息往往是应用通过网络服务器实时获得的.

·账户名信息泄露

18 个社交和移动购物类的应用中，有 1/3 的应用会在接收到消息后再次开启登录界面，并暴露出用户填写过的账号名称信息，这些信息往往是自定义昵称、电子邮箱地址、手机号码等. 攻击者通过收集这些信息，可以获得可观的个人账户数据.

·文本框注入

有些 Activity 组件的文本框的显示字符串是从 Intent 中携带的 Extra 数据中提取的，而有些组件可能会接受来自第 3 方应用程序的 Intent 消息. 文本框注入是指攻击者通过发送精心构造的 Intent 消息，向目标组件的文本框中注入特定的字符串，以达到欺骗用户等攻击手段.

·输入框注入

与文本框注入类似，只是攻击者构造的 Intent 消息将会被注入到输入框信息中. 在用户看来，这种注入的消息类似于输入框中的初始文字，例如输入格式提示、系统通知或输入记录等.

·开启网页

攻击者可以通过发送 Intent,使用应用自带的 Webview 组件,或系统中的浏览器打开网页. 更严重的情况是,有 1 个组件会开启 Intent 消息中指定的网页地址,这可能会造成对用户的误导,发生钓鱼攻击等欺诈行为.

·地理位置泄露

泄露了用户的地址信息. 这些地址信息可能是被攻击应用通过全球定位或者网络地址位置获取的,前者较为准确而后者较易获取. 攻击者通过开启这些组件,可以轻易地让用户的地理位置显示在前台界面上,然后通过截屏等方式窃取这些信息.

·其他特定功能组件暴露

KMDroid 还发现 2 个组件会暴露出卸载界面,2 个组件会开启错误通知界面,1 个组件会暴露应用列表界面, 1 个组件会以 Intent 中包含的 Int 型 Extra 数据作为端口号开启 Http 服务. 这些包含特定功能的组件原本不该暴露在外,如果被攻击者在适当的时机恶意开启,就可能发生诱导用户误操作的可能.

3.3　测试结果讨论

通过自动测试和统计数据的分析可以看出,应用会暴露大量组件,并且有些组件在接收到其他应用传来的消息时,没有对消息来源和消息内容做足够的检查,导致空指针引用等异常的抛出. 此外,Android 应用进程在异常发生时,整个进程都会关闭. 这虽然可以很好地保证程序的正常运行,但对于用户来说,在应用中开启一个新的组件发生异常时,关闭新组件、回到上一个界面并显示错误提示消息,比关闭整个进程更加合理.

通过针对性测试可以看出,应用暴露在外的组件不仅存在抛出异常导致系统崩溃的风险,而且通过构造有效的测试消息,可能会引发功能组件暴露、界面被篡改和隐私泄露等问题. 应用组件使用的功能越强大,遭到攻击时产生的危害也就越大.

很多 Android 应用在通信过程中存在着较为明显的安全隐患. 这与其复杂的 ICC 机制,以及部分应用开发者薄弱的安全意识有关. Android 系统中的 ICC 机制既适用于应用内消息的传递,也适用于应用之间的消息传递,并且对于来自应用外的消息,只要目标组件暴露在外且没有严格的权限控制,消息就会被接收处理. 这就对应用开发者在接收外部消息时的安全性意识,提出了较高的要求. 如果应用组件包含敏感的隐私信息,或重要的系统功能,那么应用开发者应当通过设置组件对应用外程序不可见,或设置调用组件需要特定权限等方式,以保障组件的安全性. 如果不得不暴露组件,应当在收到消息时,做好足够的安全检查.

4　相关工作

4.1　相关研究介绍

在 KMDroid 工具之前,其他安全研究者已针对 Android 的 ICC 机制设计出一些漏洞挖掘方法,并开发了具体工具,下面对几种工具进行简要介绍.

Intent Fuzzer 工具[16]　来自 iSEC Partners 安全顾问公司. 该应用安装在 Android 设备上,通过调用 startActivity(Intent) 、startService(Intent) 、sendBroadcast(Intent) ,分别发送 intent 消息到 Activity、Service 和 Broadcast Receiver 中. 系统中所有暴露在外的这 3 类组件,可通过 PackageManager 获取,但仅支持对于全部 Service、Broadcast Receiver 的自动化测试. 不能同时测量大量 Activity 的原因是该工具在大量开启 Activity 组件后,会越来越多地占用系统资源而无法回收,最终导致系统崩溃. 该工具是较早的一款针对 Android 应用通信过程的 Fuzzing 工具,但存在不能完全实现测试的自动化、没有日志输出、仅发送空消息等问题.

JarJarBinks 工具[17]　受 Intent Fuzzer 工具的启发,并进行了改进. 首先改用 startActivityForResult(Intent, int) 来开启 Actvity 组件,从而可以在 Activity 开启一段时间后,调用 finishActivity( ) 以关闭组件,回收资源. 此外,在产生 Intent 测试消息时,设计了 4 类测试数据: 半有效的 Action 和 Data、空的 Action 和 Data、随机的 Action 和 Data、随机的 Extras. 可以看出,该工具能有效配置 Intent 中各项数据,以实现广泛测试. 但是,这种测试数据不能针对应用,进行深入有效的数据测试. 例如,填充 Extra 项的测试中,并无法准确获得 Extra 中的 type 和 name 信息,只能设置为 Android 提供的推荐 name 列表来进行测试. 此外,在测试过程中仍需要人工介入,关闭应用崩溃时的系统弹窗,不能完全实现自动化测试.

ComDroid 工具[18] 利用逆向工程与源码分析，对源码进行静态的审计，来研究应用组件间通信过程中可能存在的安全漏洞. 该团队将基于 Intent 的攻击分为 2 种: 未授权的 Intent 接收者和恶意的 Intent 消息注入. 通过 Dedexer 工具将应用程序进行逆向处理，并利用静态分析方法，分析可能存在的安全漏洞及其分类. 研究者系统地分类研究了应用组件通信时可能存在的安全漏洞，但对于特定应用的实际攻击测试功能却不够深入，不能直接给出被攻击时的运行结果.

总的看来，上述工具都提供了 Android 组件间通信机制的漏洞挖掘功能，并能提供一定程度上的自动测试功能; 但 KMDroid 工具，在测试数据生成及自动化测试等方面的表现更令人满意.

4.2 和相关工作的比较

从工具功能和测试效果 2 个角度，对 KMDroid 工具与 Intent Fuzzer、JarJarBinks、ComDroid 等 3 个工具进行比较，形成表 7.

表 7 KMDroid 与其他工具的比较

Table 7 Comparison between KMDroid and other tools

比较项目 \ 工具名称		Intent Fuzzer	JarJar Binks	ComDroid	KMDroid
功能比较	基于 Fuzzing	√	√	X	√
	源码分析	X	X	√	√
	Extra 填充	X	○	X	√
	自动测试	○	○	√	√
测试效果比较	监测异常	○	√	X	√
	数据注入	X	○	X	√
	发现暴露组件	○	○	√	√

注: 表格中√表示具备该属性; ○表示具备该属性但相关功能并不完善; X 表示没有该属性.

通过功能比较，可以看出，KMDroid 在功能实现方面，既使用 Fuzzing 技术作为漏洞挖掘的主要方法，又利用逆向工程和源码分析，获取构造消息所需的必要信息. 与其他 2 个基于 Fuzzing 的工具相比，可以构造更加准确的测试数据; 与基于逆向工程和源码分析的 ComDroid 相比，可以更好地研究目标应用在收到恶意消息后的行为.

通过实验测试结果的比较可以看出，KMDroid 在测试结果方面，不仅可以像其他 2 个 Fuzzing 工具那样监测异常抛出，而且能将数据注入到目标组件中，并发现可能暴露的组件; 而静态分析工具 ComDroid 则主要用于发现功能组件暴露情况，不能检测可能发生异常或者暴露组件的具体行为.

5 结束语

本文介绍了一种基于 Fuzzing 的 Android 应用通信过程漏洞挖掘的新研究思路，并实现了漏洞挖掘工具 KMDroid. 得益于在 Fuzzing 工具中添加了逆向工程和源码分析模块，本文的漏洞挖掘技术可以准确地构造出包含 Extra 项的 Intent 消息，加深了测试数据的深度，挖掘出 7 类潜在的安全漏洞. 下一步工作中，可以继续实现除 Int 型、String 型和 Boolean 型之外的 Extra 项的测试; 在源码分析的过程中，测试静态分析已经发现的安全漏洞，并在后续工作中补充新的安全漏洞分类; 或者根据现有的安全防范方案[19]，进行进一步的攻击测试.

参考文献

[1] Lunden I. Nearly 75% of all smartphones sold In Q1 were android, with samsung at 30%; mobile sales overall nearly flat: gartner [EB/OL]. [2013-06-26]. http://techcrunch.com/2013/05/14/android-nearly-75-of-all-smartphones-shipped-in-q1-samsung-tops-30-mobile-sales-overall-nearly-flat-says-gartner.

[2] AppBrain. Number of available Android applications [EB/OL]. [2013-07-19]. http://www.appbrain.com/stats/number-of-android-apps.

[3] Enck W, Ongtang M, McDaniel P. Understanding android security [J]. Security & Privacy, IEEE, 2009, 7(1): 50-57.

[4] Ding L P. Analysis the security of android [J]. Netinfo Security, 2012(3): 28-31 (in Chinese).
丁丽萍. Android 操作系统的安全性分析 [J]. 信息网络安全, 2012(3): 28-31.

[5] Android Developers. Components [EB/OL]. [2013-06-23]. http://developer.android.com/guide/components/fundamentals.html#Components.

[6] Android Developers. Intents and intent filters [EB/OL]. [2013-06-26]. http://developer.android.com/guide/components/intents-filters.html. Application.

[7] Android Open Source Project. Settings app-security bug [EB/OL]. [2013-06-26]. http://code.google.com/p/android/issues/detail?id=14602.

[8] Miller B P, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities [J]. Communications of the ACM, 1990, 33(12): 32-44.

[9] Liu Q X, Zhang Y Q. TFTP vulnerability exploiting technique based on fuzzing [J]. Computer Engineering, 2007, 33(20): 142-147 (in Chinese).

刘奇旭，张玉清. 基于 Fuzzing 的 TFTP 漏洞挖掘技术[J]. 计算机工程，2007，33(20)：142-147.

[10] Yu P Y，Huang J F，Gong Y Z. Static code analysis of Android application information leakage [J]. Software，2012，33(10)：1-5 (in Chinese).
于鹏洋，黄俊飞，宫云战. Android 应用隐私泄露静态代码分析[J]. 软件，2012，33(10)：1-5.

[11] Luo C，Zhang Y Q，Wang L，et al. Automatic network protocol analysis and vulnerability discovery based on symbolic expression [J]. Journal of Graduate University of Chinese Academy of Sciences，2013，30(2)：278-284 (in Chinese).
罗 成，张玉清，王 龙，等. 基于符号表达式的未知协议格式分析及漏洞挖掘[J]. 中国科学院研究生院学报，2013，30(2)：278-284.

[12] Android Developers. Manifest [EB/OL]. [2013-06-26]. http: // developer. android. com/guide/topics/manifest/manifest-element. html#.

[13] Android Developers. Permissions [EB/OL]. [2013-06-26]. http: // developer. android. com/guide/topics/security/permissions. html.

[14] Android Developers. Android debug bridge [EB/OL]. [2013-06-26]. http: // developer. android. com/tools/help/adb. html.

[15] Android Developers. Logcat [EB/OL]. [2013-06-26]. http: // developer. android. com/tools/help/logcat. html.

[16] iSEC Partners. Intent Fuzzer [EB/OL]. [2013-06-26]. https: // www. isecpartners. com/tools/mobile-security/intent-fuzzer. aspx.

[17] Maji A K，Arshad F A，Bagchi S，et al. An empirical study of the robustness of inter-component communication in Android [C] // Dependable Systems and Networks (DSN)，2012 42nd Annual IEEE/IFIP International Conference on. IEEE，2012：1-12.

[18] Chin E，Felt A P，Greenwood K，et al. Analyzing inter-application communication in Android [C] // Proceedings of the 9th International Conference on Mobile Systems，Applications，and services. ACM，2011：239-252.

[19] Kantola D，Chin E，He W，et al. Reducing attack surfaces for intra-application communication in android [C] // Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. ACM，2012：69-80.

(上接第 759 页)

李文君. 唐山市钢铁工业发展特征及动力机制分析[J]. 地理科学进展，2003，22(2)：203-210.

[21] 王海壮，栾维新，马新华. 我国钢铁工业沿海布局战略研究[J]. 世界地理研究，2011(2)：140-147.

[22] Tian S C，Zhang W Z. Evolution of spatial pattern of Chinese iron and steel industry and the influencing factors [J]. Progress in Geography，2009，28(4)：537-545 (in Chinese).
田山川，张文忠. 中国钢铁工业空间格局的演化及影响机制[J]. 地理科学进展，2009，28(4)：537-545.

[23] Chen X，Chen W，Zhang L，et al. The scope judgment of "Pan-Yangtze River Delta" based on inter-regional links [J]. Progress in Geography，2010，29(3)：370-376 (in Chinese).
陈晓，陈雯，张蕾，等. 基于区际联系的"泛长三角"范围判定[J]. 地理科学进展，2010，29(3)：370-376.

[24] 范剑勇. 长三角一体化、地区专业化与制造业空间转移[J]. 管理世界，2004(11)：77-84.

[25] 李其世，顾德骥，尹灏，等. 上海钢铁工业志[M]. 上海：上海社会科学院出版社，2001.

[26] Jiang F T，Chen W G，Huang J B，et al. The limitations and defective results of the policy of regulating investment [J]. China Industrial Economics，2007，(6)：53-61 (in Chinese).
江飞涛，陈伟刚，黄健柏，等. 投资规制政策的缺陷与不良效应: 基于中国钢铁工业的考察[J]. 中国工业经济，2007，(6)：53-61.