基于静态污点分析的 Android 应用 Intent 注入漏洞检测方法

王允超 魏 强 武泽慧

(解放军信息工程大学 郑州 450001) (数学工程与先进计算国家重点实验室 郑州 450001)

摘 要 针对 Android 应用程序组件间通信过程中的消息载体 Intent 有可能被攻击者构造进而引发组件被恶意注入的安全风险问题,提出了一种基于静态污点分析的检测方法。在构建 Android 应用的函数调用图和控制流图的基础上,通过跟踪应用组件内和组件间不可信 Intent 消息的污点传播过程,检测应用中潜在的 Intent 注入漏洞。用该方法对 4 类标准测试应用和 50 款第三方应用进行测试,实验结果表明了该方法的可行性和有效性。

关键词 Android,静态污点分析,函数调用图,控制流图,Intent 注入漏洞

中图法分类号 TP309 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.9.038

Approach of Android Applications Intent Injection Vulnerability Detection Based on Static Taint Analysis

WANG Yun-chao WEI Qiang WU Ze-hui

(PLA Information Engineering University, Zhengzhou 450001, China)

(State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China)

Abstract As a message carrier in the process of component communication of Android application, Intent can be malformed by an attacker, leading to security risk of malicious component injection. A detection approach based on static taint analysis was presented. On the basis of building call graph and control flow graph of Android application, by tracking the taint propagation with in and between components, the potential Intent injection vulnerability can be detected. This method is used to test four types of benchmark and fifty third-party applications, and the experimental results show the feasibility and effectiveness of the proposed approach.

Keywords Android, Static taint analysis, Call graph, Control flow graph, Intent injection vulnerability

1 引言

随着 Android 智能手机的流行以及市场占有率的日益增长,越来越多的应用涌入到 Android 官方市场和第三方应用市场中。由于许多应用开发者的安全编程意识较差以及应用市场并没有采取全面的安全检测措施,导致很多应用存在安全漏洞,因此很容易遭受黑客攻击从而给用户带来安全风险。

为了保护应用安全,Android 为每个应用分配了一个用户 ID,运行在自己独立的虚拟机内。然而在一个协作的环境中,应用程序彼此之间需要共享数据,为此 Android 提供了一个灵活的通信机制,即进程间通信(Inter-Process Communication,IPC)。在 Android 系统中,IPC 以 Binder 和 Intent 两种方式存在,其中 Intent 作为消息载体(包含 action,data,type,extras 等属性),可以在同一应用或不同应用的组件之间传递消息,即通过 Intent 可以实现组件间通信(Inter-Component Communication,ICC)。大量研究表明,这种 ICC 机制的引入将应用程序暴露在外界的各种压力条件下,应用程序很可能会收到大量不受信任的通信数据从而触发新的安全漏洞。Intent 注入漏洞就是一种比较常见的 ICC 漏洞,具有该类型漏洞的应用通常包含一个公开组件,其接收到一个恶意构造的 Intent 消息后,没有进行有效的安全验证而直接将其解析出来的参数用于一些安全敏感操作,从而导致权限提升、信息泄露甚至远程代码执行等恶意行为。

目前,针对 Intent 注入漏洞的检测方法主要包含动态分析和静态分析两种。动态分析通常采用 Fuzzing 方法,将随机输入发送给目标应用,通过监测软件异常来判断潜在的安全漏洞。JarJarBinks[1] 通过 Fuzzing 方法对 Android 应用的 ICC 进行经验评估;根据自定义的变异策略将 ICC 的参数修改成任意不正确的值,然后发送给目标应用组件。Intent Fuzzer[2] 则采用了一种全新的静态分析和测试用例生成相结合的方法。针对应用的所有公开组件,通过静态分析获得 Intent 的数据结构,然后对相应的字段进行随机变异。与 JarJarBinks 相比,Intent Fuzzer 能够发现代码更深层逻辑的异常。然而这种针对 Intent 消息进行 Fuzzing 的方法得到了大量的空指针异常,很难发现可以利用的漏洞,因此效果并不好。

与动态分析相比,静态分析不需要实际运行应用,而是直接分析应用的 Dalvik 字节码,提取潜在的恶意行为。ComDroid[3] 采用静态分析的方法检测 ICC 问题,但是作者认为只要组件暴露并且没有权限保护就存在 Intent 注入攻击,这种粗粒度的分析方法会导致大量误报。CHEX[4] 通过连接所有从入口点可达的代码段发现应用组件间的数据流以检测组件劫持漏洞,但是由于没有实现组件间的数据流分析从而无法

到稿日期:2015-08-12 返修日期:2015-10-30

王允超(1992-),男,硕士生,主要研究方向为网络信息安全,E-mail:w_yunchao@sina.com;魏 强(1979-),男,副教授,硕士生导师,主要研究方向为网络信息安全;武泽慧(1988-),男,博士生,主要研究方向为网络信息安全。

检测私有组件注入情况，因此存在漏报。Epicc[5]将 ICC 问题转换为一个 IDE[6]问题进行解决，但是与 ComDroid 一样存在大量误报。在 2014 年 Blackhat 大会上，Daniele 提出了一种针对 Android 应用中 Intent 消息漏洞的静态检测和自动利用方法[7]，但是该方法只能够检测单个 Activity 组件中的漏洞，与 CHEX 一样存在漏报。

由于组件暴露并不意味着就存在 Intent 注入，而且 Intent 注入不仅仅存在于公开组件中，因此上述针对 ICC 问题的研究普遍存在误报和漏报数量。为了有效检测 Android 应用中的 Intent 注入漏洞，本文基于静态污点分析技术设计了一个原型工具 IntentTracer，通过定义 Intent 消息为污染源并同时跟踪组件内和组件间的污点数据传播的方法，可以有效减少误报和漏报数量。

2 问题定义

2.1 Android ICC

Android 应用由 4 类基本组件构成，分别是 Activity，Service，Broadcast Receiver 和 Content Provider。Intent 是 Android 应用 ICC 机制的主要媒介，通过 Intent 可以启动 Activity 和 Service 并可以向 Broadcast Receiver 发送广播消息。

Intent 包含显式和隐式两种。显式 Intent 通过名字指定了接收组件，因此 Intent 会发送到某个特定的应用组件；而隐式 Intent 的接收方可以是所有满足条件的组件，由 Android 系统确定哪个应用可以接收该 Intent。在 manifest 文件中为组件添加 intent-filter 标签，通过定义 action，category 和 data 可以接收不同的 Intent。另外，所有设置了 intent-filter 标签并且没有标记 exported 属性为 false 的组件默认是公开的，其它应用均可以向公开组件发送 Intent 消息。

2.2 威胁模型

文献[8]第一次提出了 Intent 注入漏洞，具有该漏洞的应用通常存在一个能够接收外部 Intent 消息的公开组件，而应用的内部处理逻辑直接将该 Intent 消息中解析出来的参数信息用于一些安全敏感操作，并没有经过有效安全验证，如果该 Intent 消息是攻击者精心构造的，那么就会造成 Intent 注入攻击行为。在 Intent 注入的威胁模型中包含攻击者和受害者两个角色，如图 1 所示。

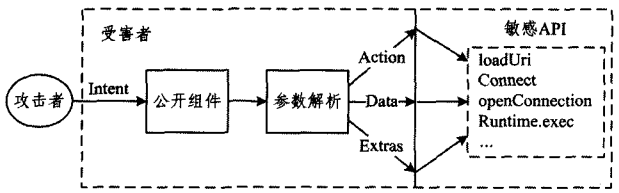

图 1 Intent 注入攻击威胁模型

攻击者是 Intent 的发送方，可能是如下 3 种情况。(1)来自 Android 市场或者任意的第三方应用市场的恶意应用。通常这些应用除了实现自身正常的功能外还可能会携带恶意代码，当用户执行某些特定操作后就会触发恶意行为，比如向目标 APP 发送一个恶意构造的 Intent。(2)恶意的链接，这个链接被重定向到一个恶意构造的 URI，当用户点击链接后，就会向某个存在漏洞的应用发送一个 Intent。(3)存在漏洞的正常应用。比如通过一个支持 Intent scheme URL[9]并且对 Intent 过滤不严的浏览器间接向系统中已安装的 APP 发送 Intent。

受害者是 Intent 的接收方，指的是那些包含公开组件且权限保护较弱甚至没有权限保护的应用。当这些公开组件接收到恶意 Intent 时，会执行一些与其预期不相符的安全敏感操作，导致攻击行为发生。

2.3 一个 Intent 注入示例

图 2 是一个存在 Intent 注入漏洞的 Android 应用，该应用包含 Activity01，BroadcastReceiver01 和 Activity02 3 个组件。其中 Activity01 是公开组件，BroadcastReceiver01 和 Activity02 是私有组件。

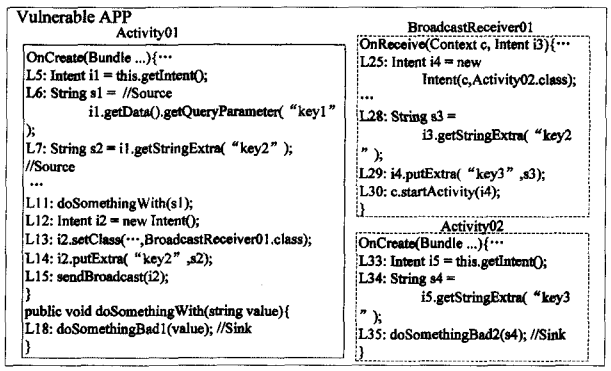

图 2 Intent 注入示例应用

假设 doSomethingBad1()和 doSomethingBad2()是两个安全敏感的 API 函数，那么这个例子中存在两个 Intent 注入漏洞，第一个在公开组件 Activity01 中，从 Intent 提取的参数 s1 没有经过任何安全检查直接传入方法 doSomethingBad1()。第二个在私有组件 Activity02 中，组件 Activity01 提取的 Intent 参数 s2 通过 ICC 先后赋值给 s3 和 s4，最终传入 doSomethingBad2()方法，同样没有进行任何安全检查。这两个漏洞表明了不仅公开组件存在 Intent 注入，借助 ICC 私有组件也可能存在 Intent 注入，如文献[10]所述的 Next_Intent 攻击就是利用 Dropbox 应用的私有组件 VideoPlayerActivity 中存在的 Intent 注入漏洞导致 Dropbox 账户 token 信息泄露。而之前的研究[3-5,7]均无法检测私有组件的 Intent 注入漏洞。因此本文的目的就是实现一个静态分析工具，通过静态污点分析的方法检测 Android 应用中公开组件以及私有组件潜在的 Intent 注入漏洞。

3 IntentTracer 基本架构

本文基于静态污点分析技术，针对 Android 应用程序设计了一个原型工具 IntentTracer。如图 3 所示，IntenTracer 对 Android 应用程序的分析过程主要包括预处理、中间过程处理和污点分析 3 个阶段。

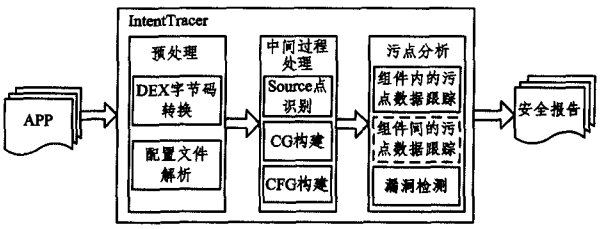

图 3 IntentTracer 系统框架图

3.1 预处理阶段

预处理阶段主要包括 DEX 字节码转换和配置文件解析 2 个模块。DEX 文件是 Dalvik 虚拟机的可执行文件，由于

DEX 字节码指令相对复杂从而增加了分析的复杂度，因此 IntentTracer 首先会将 DEX 字节码转换成中间表示语言，以便于后续阶段的分析。配置文件解析主要负责对 AndroidManifest. xml 文件进行解析，按照 2.1 节的规则提取应用中包含的公开组件，并将其添加到待分析的目标组件列表中。

3.2 中间过程处理阶段

中间过程处理阶段主要包括 Source 点识别、函数调用图(CG)构建和控制流图(CFG)构建 3 个模块。Source 点识别是根据预先定义的 Source 方法在预处理阶段分析得到的目标组件中进行查找并将返回值标记为污染源，只有存在 Source 方法才会对该组件进行分析。CFG 和 CG 是之后进行污点分析的基础，CFG 用于辅助函数内的数据流分析，CG 用于辅助函数间的调用分析。

3.3 污点分析阶段

污点分析阶段包括组件内的污点数据跟踪、组件间的污点数据跟踪和漏洞检测 3 个模块。组件内和组件间的污点数据跟踪主要是在 CFG 和 CG 基础上进行分析。图 3 中组件间的污点数据跟踪用虚线框标出是因为只有当组件内的污点数据跟踪过程中存在 ICC 方法调用并且方法参数为污点数据时才开始进行组件间的污点数据跟踪，这种按需分析的策略不再盲目地对应用中的所有组件进行分析，可以有效提高分析效率。漏洞检测指在污点分析的过程中如果存在某个 Sink 方法的相关参数信息为污点数据，IntentTracer 就会记录从 Source 到 Sink 点的执行路径并输出安全报告。

4 IntentTracer 设计与实现

IntentTracer 基于开源的 Java 静态分析框架 Soot[11] 实现，利用 Soot 提供的插件 Dexpler[12] 将 Dalvik 字节码转换成三地址码的中间表示 Jimple，在此基础上构建控制流图和函数调用图，然后进行静态污点分析，检测应用中潜在的 Intent 注入漏洞。

4.1 Source 和 Sink 定义

IntentTracer 采用静态污点分析方法，通过对不信任的输入数据(即 Source)做标记，静态跟踪程序运行过程中污点数据的传播路径，检测使用污点数据的安全敏感操作(即 Sink)。本文的 Source 和 Sink 定义如下。

(1) Source 方法：Intent 对象中用于提取 Data 和 Extras 属性信息的 GET 方法。Intent 对象包含很多 GET 方法，但是因为 Intent 注入漏洞主要是由于从 Intent 对象中提取的 Data 和 Extras 属性信息没有经过有效的安全验证导致的，所以本文筛选出 Intent 对象中获取 Data 和 Extras 参数有关的 GET 方法并将其定义为 Source 方法，例如 getStringExtra()，getData()等。

(2) Sink 方法：一些与安全敏感操作相关的 API，例如加载网页 loadUrl()、执行命令 Runtime. exec()以及数据库查询 query()等。

4.2 构建 CG 和 CFG

CG 是一个有向图，图中的结点为一个函数，图中的边表示一个调用点。构建函数调用图，首先需要确定程序的入口函数，然而与传统的 Java 应用程序不同，Android 应用并没有统一的程序入口 main 函数。Android 应用由组件构成，在应用运行期间，操作系统通过调用组件的生命周期方法(OnCreate，OnStart，OnResume 等)改变应用的运行状态，因此组件的每个生命周期方法都有可能作为程序的入口函数。

由于 Intent 注入漏洞检测主要关注来自外部的 Intent 消息的传递过程，当一个公开组件接收到一个 Intent 消息时，对 Intent 对象属性信息的提取通常在某个特定的生命周期方法中，例如 Activity 组件的 OnCreate()方法，根据该特性，定义不同组件类型与组件入口方法对应关系，如表 1 所列。IntentTracer 根据组件的类型选取相应的组件入口方法作为 CG 的入口结点，之后从 CG 入口开始解析所有函数调用语句，进行 CG 构建。

表 1 组件入口方法

组件类型	组件入口方法
Activity	void OnCreat(Bundle savedInstanceState)；
Service	void OnBind()；void OnStartCommand()；
Broadcast Receiver	void OnReceive()；

CFG 也是一个有向图，图中的结点为基本块，图中的边表示一个基本块到另一个基本块的条件信息。CFG 生成是在构建的 CG 的基础上，通过深度优先遍历其中的所有结点，最后为 CG 中的每一个函数生成一个 CFG。

4.3 静态污点分析

静态污点分析基于构建的 CG 和 CFG，包括组件内的污点数据跟踪和组件间的污点数据跟踪。

4.3.1 组件内的污点数据跟踪

在进行组件内的污点数据跟踪时，IntentTracer 根据 Source 定义将目标组件从外部接收到的 Intent 对象的相关属性信息标记为污染源，并在程序分析的过程中进行污点数据的跟踪，检测污点数据是否流入 Sink 方法中。污点分析从 CG 的入口点开始分析，遍历每个函数的 CFG，当遇到方法调用时，通过实参与形参的绑定关系分析污点数据在方法调用时的污点传播。

污染状态的传播在 Jimple 中间表示上进行。根据不同指令的语义，对污点信息进行传播。污点传播策略中定义了对每种指令的污染状态传播逻辑，如表 2 所列，其中定义 $T(v)$表示变量 v 的污点状态，$T(v)$=true 表示变量 v 为污点数据。假设 $T(a)$=true 并且 doSomethingWith(v)是任意以变量 v 为形参的函数。

表 2 污点传播策略

语句类型	语句形式	污点传播策略
赋值语句	b=a；	T(b)=T(a)
	b. f=a；	T(b. f)=T(a)，T(b)=T(a)
	b=a. f；	T(b)=T(a. f)
	b=doSomethingWith(a)；	T(v)=T(a)，T(b)=T(returnVal)
返回语句	return a；	T(returnVal)=T(a)
函数调用语句	doSomethingWith(a)；	T(v)=T(a)

4.3.2 组件间的污点数据跟踪

在进行组件内的污点数据跟踪过程中，如果遇到方法调用是一个 ICC 方法并且其参数 Intent 为污点数据，那么此时需要跟踪组件间的污点信息传播。如图 2 所示的示例应用，当程序分析到组件 Activity01 中的 sendBroadcast 方法时，由于参数 i2 为污点数据，因此需要跟踪组件间的污点数据传播，然后继续进行 BroadcastReceiver01 组件内的污点分析，同样也需要跟踪 BroadcastReceiver01 组件和 Activity02 组件间

的污点信息传播。然而，由于组件间通信是由操作系统参与完成的，这造成了组件间控制流图的不连续，如图 4 所示。Activity01 组件的 sendBroadcast 方法与 BroadcastReceiver01 组件的 OnReceive 方法之间并没有显示的调用过程，因此静态分析无法跟踪组件间的数据流信息。

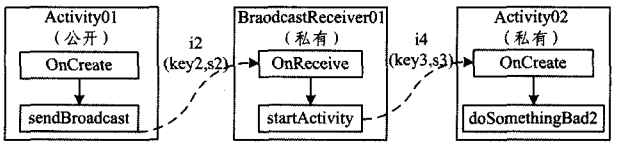

图 4　示例应用 ICC 控制流

实现组件间的污点数据跟踪，首先需要确定目标组件。当分析到 ICC 方法时，首先解析方法参数 Intent 信息，对于显式 Intent，通过获取其 Component Name 属性中的信息确定目标组件。对于隐式 Intent，需要分析其 action，data 和 category 3 个属性信息并在 AndroidManifest. xml 文件中遍历相应组件类型的 intent-filter 标签，通过信息匹配确定目标组件。当目标组件确定后，IntentTracer 通过构建一张 ICC 方法和目标组件生命周期方法的映射表来实现组件间控制流和数据流的连续性，如表 3 所列。某个 ICC 方法被调用时，会自动替换成对应的目标组件的生命周期方法同时将 Intent 信息传递到目标组件对应的方法中进行组件内的污点数据跟踪。

表 3　ICC 方法与目标组件生命周期方法映射表

目标组件类型	ICC 方法	生命周期方法	Intent 信息传递
Activity	startActivity(Intent i) startActivityForResult(Intent i,…)	OnCreate(Bundle …)	i→getIntent()
Service	startService(Intent i)	OnStartCommand(Intent intent,…)	i→Intent
	bindService(Intent i,…)	OnBind(Intent intent)	i→Intent
Broadcast Receiver	sendBroadcast(Intent i) sendBroadcast(Intent i,…) sendOrderedBroadcast(Intent i,…) sendOrderedBroadcast(Intent i,…,…) sendStickyBroadcast(Intent i) sendStickyOrderedBroadcast(Intent i,…)	OnReceive(…,Intent intent)	i→Intent

4.3.3　漏洞检测

Intent 注入漏洞的检测发生在污点跟踪的过程中。当 IntentTracer 分析 Jimple 中间表示语句时，如果当前语句包含函数调用，则首先判断该方法是否位于定义的 Sink 方法列表中，如果是，则进一步判断该方法所使用的相关参数是否为污点数据；如果是，则意味着存在一个潜在的 Intent 注入漏洞，然后根据程序的数据流信息，生成该方法相关参数的数据依赖图，根据数据依赖图的结点信息记录从 Source 到该 Sink 方法的污点跟踪路径，即生成 trace 记录以便进一步分析。

5　实验结果

5.1　实验环境

实验平台为 64 位 Intel Xeon 机器，其处理器核数为 32，CPU 频率为 2. 13GHz，物理内存大小为 32GB，操作系统为 Ubuntu12. 04 版本。分别选取标准测试应用和第三方应用两组样本集进行测试。

5.2　对比测试

为验证和评估 IntentTracer 的有效性，本文设计了 4 类标准测试应用，均包含两个组件，分别是公开组件 C_1 和私有组件 C_2，如表 4 所列。分别用 IntentTracer 与 Epicc 对这 4 类应用进行分析，分析结果如表 5 所列。可以发现，针对这 4 类应用，对在检测能力方面，IntentTracer 均可以正确检测，而 Epicc 只能正确检测出 A 类型的应用，对另外 3 类均存在漏报或误报，这是由于 Epicc 认为所有公开组件均存在 Intent 注入漏洞并没有进行细粒度的分析。另外，在性能方面，IntentTracer 的分析所用平均时间明显少于 Epicc，主要原因是 IntentTracer 关注的是外部接收的 Intent 消息的传递过程，只分析应用中的公开组件，按需分析私有组件而且在对单个组件的处理过程中并不处理事件响应函数代码，而 Epicc 需要分析所有的组件并且对单个组件内的所有代码进行分析，因此性能开销更大。

表 4　4 类标准测试应用

应用类型	公开组件 C_1	私有组件 C_2	是否存在 Intent 注入
A	有 source 和 sink	—	是
B	有 source	sink	是
D	无 source	—	否
E	有 source 无 sink	—	否

表 5　标准测试应用分析结果

应用类型	IntentTracer	Epicc	平均分析时间(s) (IntentTracer/Epicc)
A	√	√	45/58
B	√	漏报	52/56
D	√	误报	11/45
E	√	误报	47/52

5.3　功能测试

本文不仅针对标准测试应用进行分析，还从国内第三方应用市场中收集了 50 个 10M 以下的应用作为测试样本，包含安全管理类应用、社交类应用以及移动购物类应用。发现其中 8 个测试样本存在从 Source 到 Sink 的路径，并通过人工创建 exploit 触发，证明了确实存在 Intent 注入漏洞，测试结果如表 6 所列。从表中可以发现每个应用都存在大量公开组件，测试中发现样本 Sample2 的某个公开组件中存在 Intent 注入，可以实现加载任意网页，同时该组件中的 WebView 实现了 addJavaScriptInterface 接口[13]，因此可以执行嵌入网页中的 JavaScript 代码实现远程命令执行。样本 Sample5 的 Sink 点发生在私有组件内，需要进行组件间的污点数据跟踪。另外发现 Sample7 公开组件数为 10，其分析所用时间却明显高于其它应用，主要是由于 Sample7 的每个公开组件的代码复杂度明显高于其他应用。

表 6 第三方应用漏洞检测结果

样本	公开组件数/总组件数	Sink 类型	组件内/组件间	分析时间(s)
Sample1	11/58	开启网页	组件内	155
Sample2	5/37	开启网页(可执行远程命令)	组件内	65
Sample3	15/72	修改数据库	组件内	192
Sample4	9/63	执行命令	组件内	131
Sample5	13/51	开启网页	组件间	195
Sample6	8/49	修改数据库	组件内	55
Sample7	10/57	向指定号码发送短信	组件内	320
Sample8	7/44	写入指定文件	组件内	116

结束语 本文提出了一种基于静态污点分析的 Android 应用 Intent 注入漏洞检测方法,设计并实现了原型工具 IntentTracer。本方法同时考虑组件内和组件间的污点数据跟踪,可以有效减少误报和漏报;另外,组件分析只关注来自应用外部的 Intent 消息处理所经过的代码段而非盲目地分析所有应用组件,有效降低了性能开销。

目前的分析方法并没有考虑程序代码中的反射机制和 JNI 调用情况,因此静态分析结果存在一定误差,对于这类问题,今后将做进一步研究。

参 考 文 献

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

[2] Sasnauskas R, Regehr J. Intent fuzzer: crafting intents of death [C]// Proceedings of the 2014 Joint International Workshop on Dynamic Analysis (WODA) and Software and System Performance Testing, Debugging, and Analytics (PERTEA). ACM, 2014: 1-5

[3] 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

[4] Lu L, Li Z, Wu Z, et al. Chex: statically vetting android apps for component hijacking vulnerabilities [C] // Proceedings of the 2012 ACM Conference on Computer and Communications Security. ACM, 2012: 229-240

[5] 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]// USENIX Security 2013. 2013: 543-558

[6] Sagiv M, Reps T, Horwitz S. Precise interprocedural data flow analysis with applications to constant propagation[J]. Theoretical Computer Science, 1996, 167(1): 131-170

[7] Gallingani D, Gjomemo R, Venkatakrishnan V N, et al. Static detection and automatic exploitation of intent message vulnerabilities in Android applications[OL]. http://www.ieee-security.org/TC/spw2015/Most/papers/s3p1.pdf

[8] Enck W, Octeau D, McDaniel P, et al. A Study of Android Application Security[OL]. http://www.usenix.org/legacy/events/secll/tech/full_papers/Enck.pdf

[9] Takeshi Terada/Mitsui Bussan Secure Directions, Inc. Attacking Android browsers via intent scheme URLs[OL]. http://www.mbsd.jp/whitepaper/InterScheme.pdf

[10] Wang R, Xing L, Wang X F, et al. Unauthorized origin crossing on mobile platforms: Threats and mitigation[C]// Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. ACM, 2013: 635-646

[11] Lam P, Bodden E, Lhoták O, et al. The Soot framework for Java program analysis: a retrospective[C]// Cetus Users and Compiler Infastructure Workshop (CETUS 2011). 2011

[12] Bartel A, Klein J, Le Traon Y, et al. Dexpler: converting android dalvikbytecode to jimple for static analysis with soot[C]// Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program Analysis. ACM, 2012: 27-38

[13] Luo T, Hao H, Du W, et al. Attacks on WebView in the Android system[C]// Proceedings of the 27th Annual Computer Security Applications Conference. ACM, 2011: 343-352

(上接第 187 页)

[9] Dev H, Sen T, Basak M, et al. An Approach to Protect the Privacy of Cloud Data from Data Mining Based Attacks[C]// 2012 SC Companion: High Performance Computing, Networking Storage and Analysis. 2012: 1106-1115

[10] Zheng Xian. Status and Development of Web Mining[J]. Journal of Technology and Market, 2013, 20(3): 64-65(in Chinese)
郑弦. Web 挖掘的现状和展望[J]. 技术研发, 2013, 20(3): 64-65

[11] Li Ling-juan, Zheng Shao-fei. Analysis of Data Mining Privacy Preserving Technology Based on Data Processing [J]. Journal of Computer Technology and Development, 2011, 21(3): 94-97(in Chinese)
李玲娟, 郑少飞. 基于数据处理的数据挖掘隐私保护技术分析[J]. 计算机技术与发展, 2011, 21(3): 94-97

[12] He Yao, Wang Wen-qing, Xue Fei. Study of Massive Data Mining Based on Cloud Computing[J]. Computer Technology and Development, 2013, 23(2): 69-72(in Chinese)
贺瑶, 王文庆, 薛飞. 基于云计算的海量数据挖掘研究[J]. 计算机技术与发展, 2013, 23(2): 69-72

[13] Ding Yan, Yang Qing-ping, Qian Yu-ming. Architecture and Key Technology of a Data Mining Platform Based on Cloud Computing[J]. ZTE Technology Journa, 2013, 19(1): 53-57(in Chinese)
丁岩, 杨庆平, 钱煜明. 基于云计算的数据挖掘平台架构及其关键技术研究[J]. 中兴通信技术, 2013, 19(1): 53-57

[14] Abuhussein A, Bedi H, Shiva S. Evaluating Security and Privacy in Cloud Computing Services: A Stakeholder's Perspective[C]// The 7th International Conference for Internet Technology and Secured Transactions. 2012: 388-395