第 42 卷 第 10 期 计 算 机 工 程 2016 年 10 月
Vol. 42 No. 10 Computer Engineering October 2016

·安全技术· 文章编号:1000-3428(2016)10-0169-07 文献标志码:A 中图分类号:TP309.2

基于 TaintDroid 扩展的 WebView 隐私泄露检测

王伟平,林漫涛,李天明,吴 伟

(中南大学 信息科学与工程学院,长沙 410083)

摘 要: 针对 Android 应用程序与 WebView 交互可能导致隐私泄露的问题,分析 WebView 的 2 个隐私泄露主要通道 addJavascriptInterface 和 loadUrl,提出一种扩展 TaintDroid 的数据流跟踪框架 WTD。WTD 扩展了 Android 源码中的 addJavascriptInterface 和敏感 API,在敏感 API 中利用 Java 函数栈执行跟踪机制记录系统中的函数调用层次关系,从而判断 addJavascriptInterface 注册对象是否访问敏感 API,同时增加 loadUrl 加载页面参数的污点检测。实验结果表明,WTD 能有效检测 WebView 引起的隐私数据泄露。

关键词: Android 平台;WebView 组件;隐私泄露;数据流跟踪;污点检测

中文引用格式: 王伟平,林漫涛,李天明,等. 基于 TaintDroid 扩展的 WebView 隐私泄露检测[J]. 计算机工程,2016,42(10):169-175.

英文引用格式: Wang Weiping, Lin Mantao, Li Tianming, et al. Privacy Leakage Detection of WebView Based on TaintDroid Extension[J]. Computer Engineering,2016,42(10):169-175.

Privacy Leakage Detection of WebView Based on TaintDroid Extension

WANG Weiping, LIN Mantao, LI Tianming, WU Wei

(School of Information Science and Engineering, Central South University, Changsha 410083, China)

【Abstract】 Interaction between Android application program and WebView will lead to privacy leakage. Aiming at the problem, this paper analyzes two privacy leakage channel of WebView, addJavascriptInterface and loadUrl, and proposes a data flow tracking framework of extended WebView, named WebView Tainted Detection (WTD). WTD extends the addJavascriptInterface and sensitive API in the Android source code. In sensitive API, Java function stack executes trace mechanism to record function hierarchy call relation in the system to detect whether the addJavascriptInterface registering objects accessed Android privacy API. Besides, WTD increases the taint detection of loadUrl loading page parameters. Experimental result shows that WTD can effectively detect the private data leakage from WebView.

【Key words】 Android platform; WebView component; privacy leakage; data flow tracking; taint detection

DOI: 10.3969/j.issn.1000-3428.2016.10.030

1 概述

近年来,Android 平台由于其开源特性被广泛应用于智能移动终端领域,承担计算和资源管理的工作。Android 提供了包括权限、应用程序签名、程序沙箱等安全机制,保证了平台及其数据安全[1-2],但在实际应用中系统对于隐私数据保护,仍存在大量的安全隐患。文献[3-4]综述了 Android 面临的安全问题,指出恶意程序可以利用用户安装应用时授予的相关权限,获取隐私数据,并通过网络、短信等传输通道将这些敏感数据传播出去。

为提高 Android 平台的安全性,防止用户隐私数据泄露,研究人员提出一系列隐私数据保护机制和方法。针对 Android 系统粗粒度权限机制存在的缺陷,研究人员提出一些更严格或细粒度的访问控制机制。文献[5]分析 Android 自主访问控制策略的不足,提出运用强制访问控制策略保护 Android 系统中的资源。文献[6]提出基于上下文环境的访问控制模型,用户可以配置访问控制策略,使应用程序只能在特定环境下使用敏感数据。针对 Android 系统恶意应用程序的检测,一些研究人员通过应用程序权限申请来关联可能的恶意行为。文献[7]设计了挖掘 Android 应用权限之间关联性的权限频繁模式挖掘算法,通过挖掘权限之间

基金项目: 国家自然科学基金资助项目(61173169)。

作者简介: 王伟平(1969—),女,教授,主研方向为虚拟化安全、Web 安全;林漫涛、李天明、吴 伟,硕士研究生。

收稿日期: 2015-11-23 **修回日期:** 2015-12-25 **E-mail:** wpwang@mail.csu.edu.cn

的关联性检测未知的恶意应用程序。文献[8]研究了恶意应用申请权限的特点,并采用机器学习方法对恶意应用进行分类检测。另一些研究人员从应用程序数据流和控制流入手分析潜在的恶意行为。例如 CHEX 利用静态分析的方法研究应用组件间通信中的数据流和控制流,检测应用是否存在组件暴露漏洞[9]。FlowDroid 通过静态分析的方法构造控制数据流图对隐私数据进行跟踪,判断其是否泄露[10]。文献[11]提出的隐私数据污点跟踪系统 TaintDroid 完成了开创性的工作,通过对隐私数据进行污点标记并实时跟踪应用程序中隐私数据的传播,能够检测和记录隐私数据离开终端设备的行为。之后有许多相关研究是基于 TaintDroid 进行了扩展和改进[12-13]。

然而,上述文献都没有针对 WebView 交互的安全性展开研究。由于 Android 应用中大量采用 WebView,通过其提供的 API 能够加载并展示 Web 页面,实现页面中 JavaScript 代码与应用程序中 Java 代码的相互调用,完成应用程序与 Web 页面的交互。这种交互导致了 WebView 可能成为隐私数据泄露的通道。文献[14]研究并描述了 WebView 的安全漏洞和可能针对此漏洞实施的攻击行为。文献[15]提出基于白名单授权的访问控制策略,由 APP 开发者设定 Web 站点白名单,在白名单范围内的 Web 站点才能通过 WebView 获取相应的隐私数据,但该方案使决定权在 APP 开发者手中,恶意 APP 开发者可以将恶意网站列入白名单,从而窃取隐私数据。

据目前查阅到的文献,还没有一种有效的检测 WebView 隐私数据泄露的方法。为此,本文分析针对 WebView 的多种攻击通道,阐述产生安全问题的本质原因。在此基础上,提出一种结合污点数据跟踪、隐私数据操作溯源的 WebView 隐私数据泄露检测方法。针对 WebView 隐私泄露的数据通道,在 TaintDroid 污点数据跟踪框架的基础上,扩展 WebView 方法和 Android 平台隐私 API,利用 Java 反射机制[16]和函数执行栈跟踪机制[17-18],有效检测 WebView 可能存在的隐私数据泄露问题。

2 WebView 隐私泄露隐患分析

2.1 WebView 工作机制

WebView 是浏览器引擎 WebKit 所属的开发组件,能够在应用程序中提供加载和展示 Web 页面等浏览器基础功能,被广泛应用于基于 Web 服务的移动互联网应用程序开发。更重要地,WebView 提供了一组 API 能够实现应用程序与 Web 页面的交互。根据文献[14]给出的恶意应用与 Web 交互的威胁检测系统,WebView 存在 2 种交互方式。

图 1 中的方式 A 表示 Web 页面中的 JavaScript 代码调用 addJavascriptInterface 注册的 Java 对象方法 method()。当 method()方法具有隐私数据操作权限时,通过该交互机制,Web 页面中的代码就可能完成终端设备文件系统读写、通讯录和通话记录隐私数据获取等高权限操作。

图 1 中的方式 B 表示由应用程序通过 loadUrl()方式加载 Web 页面后,注入 JavaScript 代码到访问的 Web 页面中。借助于这种交互机制,注入的 JavaScript 代码能够操作页面中的 DOM 对象和 cookies,调用 JavaScript 代码,发送 AJAX 请求。WebView 解析执行注入的 JavaScript 代码,将可能发送隐私数据。

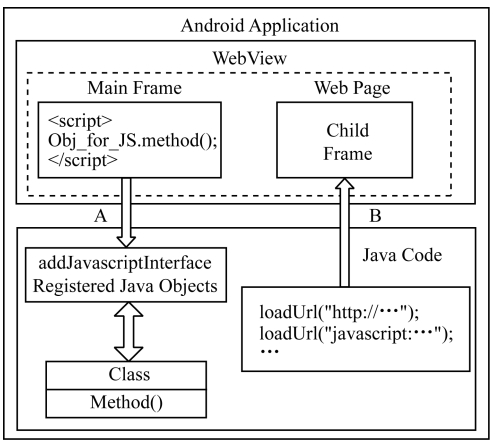

图 1 WebView 交互方式

2.2 WebView 隐私泄露通道

根据图 1 的 WebView 交互方式,得到隐私数据经过 WebView 可能泄露的通道,如图 2 所示。

(1)隐私数据传入 loadUrl 和 addJavascript Interface 注册的 Java 对象方法中。

(2)当 loadUrl 加载页面时,参数含有隐私数据,将隐私数据传入 JavaScript 环境中。

(3)在 JavaScript 中,调用 addJavascriptInterface 注册的 Java 对象方法,从返回值获取隐私数据。

(4)在 JavaScript 中,调用 addJavascriptInterface 注册的 Java 对象方法,参数含有隐私数据,将隐私数据传入 Java 环境中。

(5)addJavascriptInterface 注册的 Java 对象方法将隐私数据通过 Socket 发送到远程服务端。

(6)在 JavaScript 中,将隐私数据通过 http 发送到远程服务端。

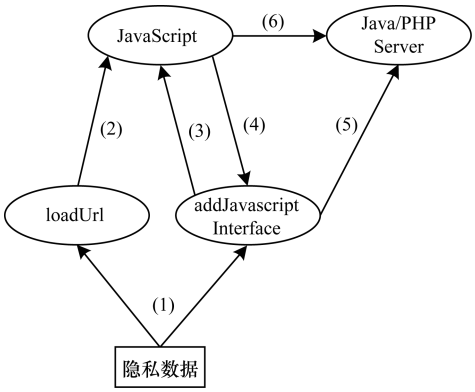

图 2　WebView 隐私数据流泄露通道

按照 WebView 泄露隐私通道的数据流分析,给出 4 种覆盖图 2 所有可能通道的 WebView 隐私泄露方式 C1 ~ C4。

(1)C1 对应的传输通道为(1)→(2)→(6)。

(2)C2 对应的传输通道为(1)→(3)→(6)。

(3)C3 对应的传输通道为(1)→(3)→(4)→(5)。

(4)C4 对应的传输通道为(1)→(5)。

C1 ~ C4 这 4 个样例的关键代码都以 WebView 泄露设备 IMEI 号为场景,其中 IMEI 是移动设备国际识别码,又称为国际移动设备标识,这是手机的唯一识别号码。当外部网站获取 IMEI 号后,可以对用户的访问行为进行跟踪。4 个样例均包含了 Web 服务端代码和使用 WebView 的 APP 客户端代码 2 个部分。Web 服务端代码包含前台 HTML 页面和 Java 或 PHP 后台处理代码;APP 客户端代码包含 WebView,负责与 Web 服务端的交互。

C1 关键代码如图 3 所示。APP 中调用 getDeviceId 获取 IMEI 号保存到局部变量 Imei,WebView 的 loadUrl 加载了一个 Web 页面,然后再次调用 loadUrl 注入一段 JavaScript 代码“javascript:sendIMEI(′″ + Imei + ″′)”,此时隐私数据 Imei 被传入 JavaScript 层。而 JavaScript 代码中 sendIMEI 方法会将传来的 IMEI 号先存放在变量 data 中,然后 post 传给后台 PHP 代码。后台 PHP 获取请求并存入 result.txt 文件。当用户安装该 APP 后,用户的 IMEI 号会通过上述方式发送给 Web 服务器端。

C2 关键代码如图 4 所示。在 APP 中,WebView 的 addJavascriptInterface 注册一个 Java 对象,定义 getIMEI 方法获取 IMEI 号。在 WebView 加载的 Web 页面中,JavaScript 调用 Java 注册对象的 getIMEI 方法,getIMEI 返回 IMEI 号,此时隐私数据 IMEI 号被传入 JavaScript 层。接下来的部分与 C1 相同,由 JavaScript 代码将 IMEI 号传给后台 PHP 代码。C1 与 C2 的不同点在于,C1 是通过图 1 的 B 方式,即 Java 调用 JavaScript 泄露隐私,而 C2 是通过图 1 的 A 方式,即 JavaScript 调用 Java 泄露隐私。

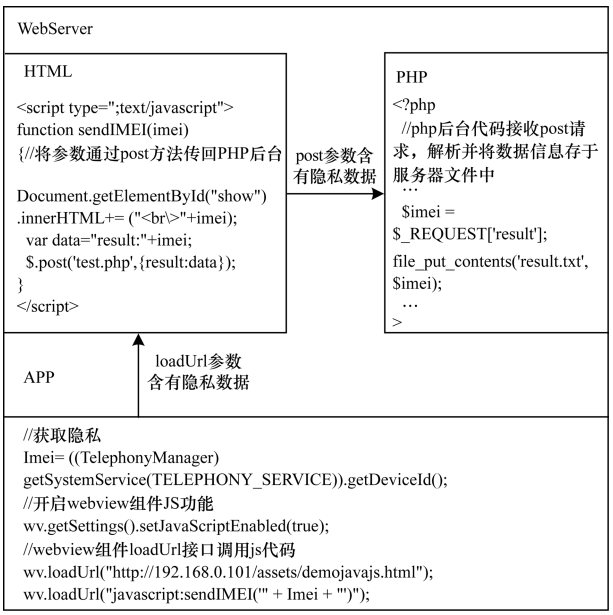

图 3　WebView 隐私泄露样例 C1

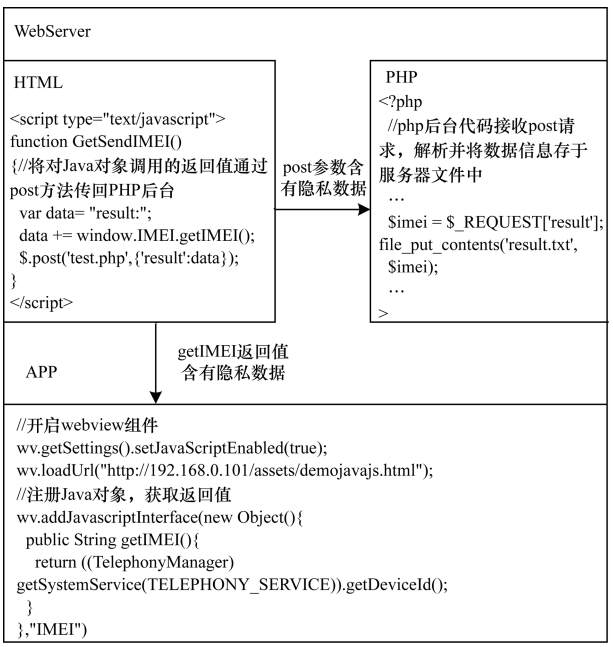

图 4　WebView 隐私泄露样例 C2

C3 关键代码如图 5 所示。与 C2 相同,依然由 Web 页面中的 JavaScript 调用 Java 注册对象的 getIMEI 方法,获取 IMEI 号。不同的是,JavaScript 在获取 IMEI 号之后,会调用 sendIMEI 方法又将 IMEI 号作为参数传回 Java 层。由 Java 函数通过 Socket 发送隐私数据到服务端。C3 比 C2 增加了从 JavaScript 层传回 Java 层的隐私数据处理流程。

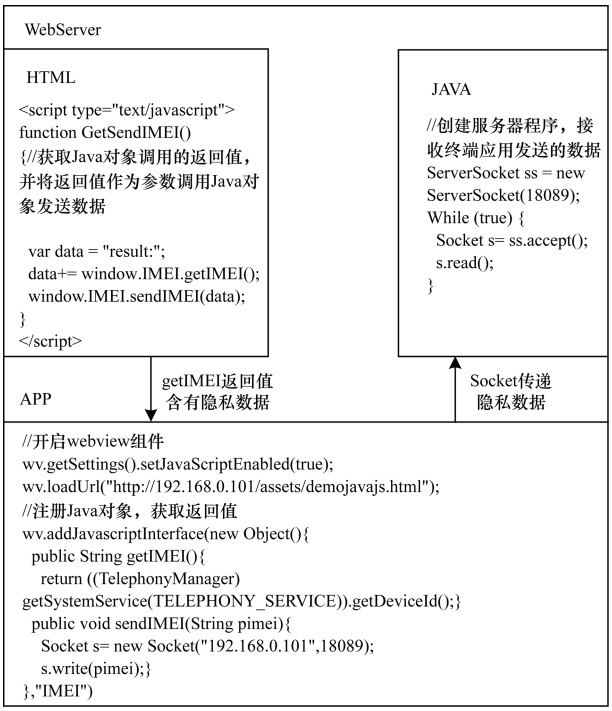

图 5 WebView 隐私泄露样例 C3

C4 关键代码如图 6 所示。C4 的处理逻辑与 C3 基本一致,JavaScript 代码调用 Java 代码操纵隐私数据,Java 代码直接把隐私数据发送到服务端。在该样例中隐私数据不会被传入 JavaScript 层。

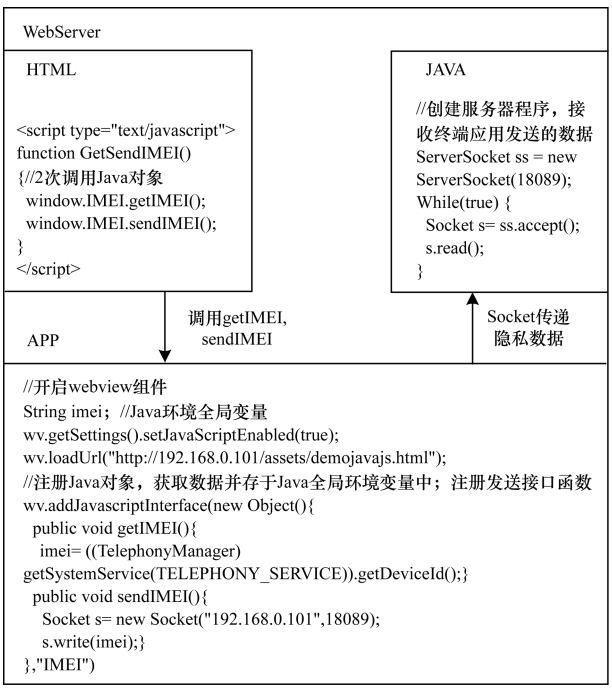

图 6 WebView 隐私泄露样例 C4

2.3 WebView 隐私泄露测试

为验证 C1 ~ C4 这 4 个样例隐私数据的传递情况,在原生 Android4.3 系统和 TaintDroid 系统进行测试,测试结果如表 1 所示,其中,“√”表示提出告警;“×”表示没有告警。

表 1 WebView 隐私泄露样例测试

样例	隐私传递代码层	原生 Android	TaintDroid
C1	JAVA→JS→PHP SERVER	×	×
C2	JAVA→JS→PHP SERVER	×	×
C3	JAVA→JS→JAVA →JAVA SERVER	×	×
C4	JAVA→JAVA SERVER	×	√

所有样例在原生 Android 系统上运行,可以在没有任何提示的情况下成功地将 IMEI 号传输到外部。在 TaintDroid 上,C1 ~ C3 均能在没有告警情况下成功地将隐私数据发送至远程服务器,而对 C4,TaintDroid 会在数据外传前提出告警。

究其原因,TaintDroid 在实现上修改了 Dalvik 虚拟机中的 Java 字节码,通过对敏感数据打标记的做法实现了对敏感数据的跟踪,当将隐私数据向外传输时提出告警。该机制决定了 TaintDroid 只能跟踪 Java 对象环境中的数据流。由于 JavaScript 运行时环境中的变量存储与宿主环境以及应用环境中的对象是隔离的,不满足 TaintDroid 在 Dalvik VM 中实现对 Java 字节码层面的数据跟踪条件,一旦隐私数据被传入图 2 所示的 JavaScript 对象环境,TaintDroid 的数据流跟踪机制将在 JavaScript 环境中失效,因此 TaintDroid 检测 WebView 隐私数据泄露时存在漏报。

3 WTD 设计与实现

3.1 WTD 设计思想

由图 2 可知,WebView 隐私泄露有 2 个主要通道 addJavascriptInterface 和 loadUrl,隐私数据由这2 个通道传入 JavaScript 环境,从而绕过 TaintDroid 污点数据跟踪。本文考虑在 TaintDroid 上进行扩展,增加对 Web View 泄露隐私的检测,当污点数据传入 JavaScript 环境时,使检测系统也能提出告警。本文提出一个基于 TaintDroid 扩展的 Android 平台 WebView 隐私泄露检测框架 WTD(WebView Tainted Detection)。WTD 检测思想如下:(1)addJavascriptInterface 通道检测。针对 C2,C3 样例的通道,隐私数据通过 addJavascript Interface 注册 Java 对象的方法传入 JavaScript 环境,污点标记无法跟踪。该检测思想是检测隐私数据的发起调用方法,当调用方法为 addJavascriptInterface 注册 Java 对象的方法时,则判定有隐私泄露风险。在实现上,本文改写了隐私数据 API,采用 Java 函数栈执行跟踪机制跟踪系统中的函数调用层次关系。Java 提供 getStackTrace 方法可以得到当前线程的函数栈,从中得到隐私 API 发起的调用方法。(2)loadUrl 通道检测。针对 C1 样例,WebView 通过 loadUrl 加载 Web 页面,loadUrl 参数带有隐私数据,增加 Taint Droid 对

WebView 的 loadUrl 方法的污点跟踪,在 loadUrl 加载 Web 页面时,对其参数进行隐私污点数据检测,当判定参数带有污点标记时,告警隐私数据被传入 Web 页面中。

3.2　WTD 系统架构

WTD 主要包括 4 种组件:TaintDroid 组件,扩展的 API,新增的组件以及其他组件,如图 7 所示。TaintDroid 组件包括 Source 节点污点标记、Sink 节点污点检测、进程间污点传播检测、存储污点传播检测组件,用于对隐私数据进行污点跟踪。WTD 扩展的 API 包括隐私 API、addJavascriptInterface 方法以及 loadUrl 方法,通过修改这些 Android 系统 API,使其在操作隐私数据时,WTD 能够跟踪隐私数据的传播。WTD 新增的组件包括记录内容提供者(Record Content Provider)和注册对象信息库,用于记录和查询 WebView 注册对象的方法。其他组件包括 Android 原生系统组件和待检测的 APP。

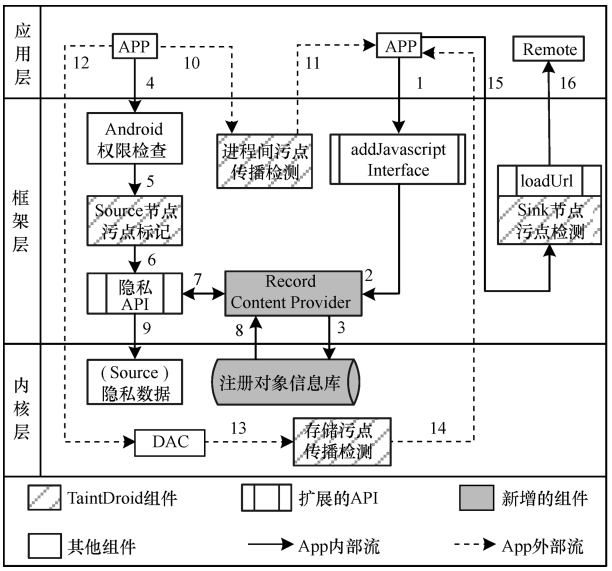

图 7　WTD 系统架构

3.3　WTD 工作流程

WTD 工作流程具体描述如下:

(1)当应用程序通过 addJavascriptInterface 注册可供 Web 页面代码调用的 Java 对象时,扩展的 addJavascriptInterface 方法利用 Java 反射机制获取 Java 对象信息并保存到 WebView 注册对象信息库,从而记录应用程序名与 Java 对象名的映射关系(步骤 1 ~ 步骤 3)。

(2)当应用程序通过敏感 API 获取隐私数据时,Android 安全机制对应用程序进行权限检查(步骤 4、步骤 5)。TaintDroid 框架在系统中对隐私数据进行污点标记,并在函数相互调用、进程间通信、文件存储等层面维持污点传播跟踪(步骤 5、步骤 6,步骤 10、步骤 11,步骤 13、步骤 14)。

(3)在扩展的隐私数据访问 API 中,利用 Java 函数栈跟踪机制跟踪此 API 的执行过程,并记录其上游函数调用层次关系;同时查询 WebView 注册对象信息库,如果上游函数是应用程序中 WebView 注册的 Java 对象方法,那么判定应用程序对此隐私数据 API 的操作存在隐私数据泄露的风险(步骤 7、步骤 8)。

(4)应用程序通过 WebView 的 loadUrl 方法执行 JavaScript 代码,扩展的 loadUrl 方法对传入的参数进行 TaintDroid 隐私数据污点检查,如果参数包含隐私数据污点标签,则可判定并提醒应用程序 WebView 存在隐私数据泄露的风险(步骤 15、步骤 16)。

3.4　关键模块

3.4.1　WebView 方法扩展模块

针对 WebView 提供的应用程序与 Web 页面的 2 种交互方式,WTD 扩展了 WebView 的 2 个方法:addJavascriptInterface 和 loadUrl。

图 8 描述了扩展的 addJavascriptInterface 方法的工作流程。应用程序在 WTD 框架中运行(步骤 1),假如应用程序是首次使用,则在注册对象信息数据库中初始化该应用记录,并当第一次使用 WebView 注册 Java 对象时(步骤 2),扩展的 addJavascriptInterface 方法通过 Java 对象的反射机制获取注册对象的方法属性,将类名和注册对象方法名保存到 WebView 注册对象信息库(步骤 3、步骤 4)。返回 addJavascriptInterface 方法,应用程序继续执行(步骤 5)。

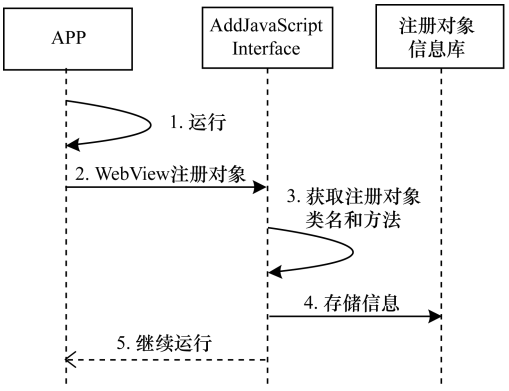

图 8　addJavascriptInterface 扩展方法的工作流程

图 9 描述了扩展的 loadUrl 方法的工作流程。WTD 将 WebView 的 loadUrl 方法添加到 TaintDroid 数据传输通道监控中,并在此方法中扩展了对隐私数据的污点检查。应用程序在 WTD 中运行(步骤 1)并进行隐私数据操作(步骤 2),TaintDroid 污点跟踪框架维护隐私数据标签在同一个应用程序内部代码之间或者同一终端上不同应用程序之间的扩散与传播(步骤 3)。应用程序在获取了终端设备上的隐私数据

后,通过 WebView 的 loadUrl 方法将这些带标签的隐私传输出去(步骤 4)。WTD 中的扩展 loadUrl 方法会获取此应用程序 UID、应用程序名称等信息(步骤 5)。利用 TaintDroid 对 Sink 节点进行污点数据检测,打印并返回隐私数据泄露日志,继续执行应用程序(步骤 6)。

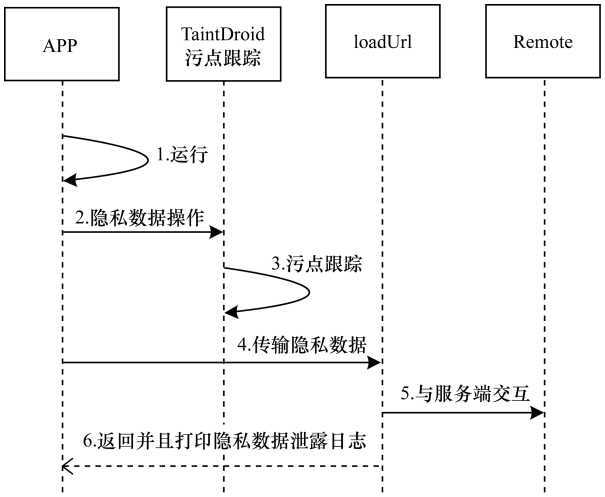

图 9 loadUrl 扩展方法的工作流程

3.4.2 Android 隐私数据 API 扩展模块

针对 WebView 能够注册 Java 对象方法供 WebView 加载的 Web 页面中 JavaScript 代码调用的特性,WTD 利用函数栈跟踪机制跟踪隐私数据操作 API 的执行过程。如果该 API 的调用上游是 WebView 注册的 Java 对象,那么就能确定对此敏感数据操作可能存在隐私数据泄露的风险。

图 10 为 WebView 调用注册的 Java 对象,该 Java 对象实现了获取终端设备隐私数据的方法,也就是在该方法中将调用 Android 敏感数据操作 API,TaintDroid 会对涉及到敏感数据操作的数据打上污点标记并进行污点跟踪(步骤 2、步骤 3)。在扩展的敏感 API 中,利用 getStackTrace 方法进行函数的执行过程跟踪,获取当前线程对敏感数据操作 API 的调用栈,同时获取执行敏感数据操作的类名和方法名(步骤 4、步骤 5)。查询该类名和方法名是否存在于 WebView 注册对象信息库中,如果有相关记录,则打印出可能存在敏感数据泄露风险的日志信息,并且返回主程序继续执行(步骤 7、步骤 8)。

表 2 列举了 WTD 中一部分扩展的敏感 API,这些 API 与 WebView 相关性较大,例如获取默认浏览器的收藏书签方法 getAllBookmarks,获取用户浏览器访问历史记录方法 getAllVisitedUrls。这些敏感数据与用户浏览习惯、关注喜好相关性较大,经常被广告商收集起来进行数据挖掘,达到精确广告投放的目的。

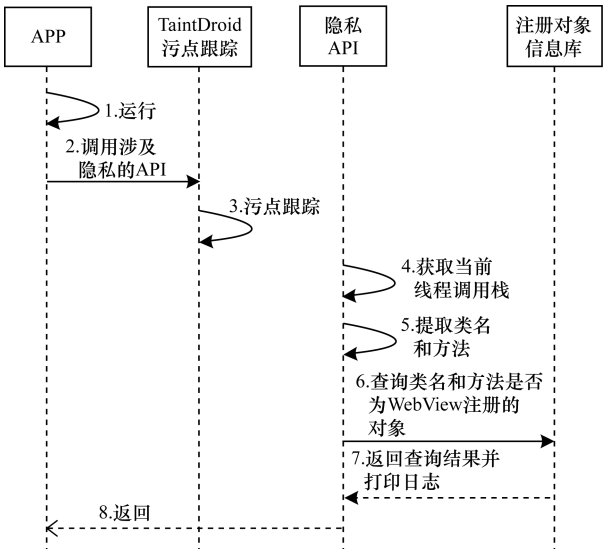

图 10 隐私数据 API 扩展模块流程

表 2 扩展的敏感 API 及其主要功能

敏感 API	主要功能
getCellLocation	获取当前地理位置
getDeviceId	获取设备 ID
getPhoneType	获取手机模式
getSimSerialNumber	获取 SIM 序列号
getAccounts	获取设备上的账户名
getAllBookmarks	获取默认浏览器的收藏书签
getAllVisitedUrls	获取浏览器访问历史记录
getAuthToken	获取账户授权令牌
getProfileConectionState	获取蓝牙连接状态
getLastKnownLocation	获取最近一次的已知地址位置

4 测试评估

为验证 WTD 的有效性,本文测试了覆盖通路的 4 个样例以及豌豆荚应用市场上使用到 WebView 的 25 个热门应用。在检测过程中使用的软硬件设备包括双核搭载 Windows7 系统的 3.2 GHz CPU、4 GB RAM 主机。利用 VMware 工具搭建的 Ubuntu11.10 虚拟机,版本为 4.3_r2.1 的 Android 源码,与 Android4.3 版本对应的 TaintDroid 框架。采用 Emulator 模拟器构建 Android 系统模拟环境。

本文在模拟器上分别用 TaintDroid 和 WTD 对样例 C1 ~ C4 进行检测,同时将样例 C1 ~ C4 上传到 6 个 Android 应用检测平台:Andrubis,virustotal,visualThreat,安全管家在线测,腾讯安全实验室以及网秦安全进行检测。表 3 给出了 4 个样例的检测结果,其中,“√”表示成功检测;“×”表示不能检测。结果表明,WTD 方法能够检测出所有 4 个样例泄露隐私数据,TaintDroid 仅能检测出编号为 C4 的测试样例泄露隐私数据,而其他 6 个 Android 应用检测平台都没有检测出 WebView 泄露隐私数据的危险。

表 3　样例 C1 ~ C4 检测结果

检测工具	C1	C2	C3	C4
WTD	√	√	√	√
TaintDroid	×	×	×	√
Andrubis	×	×	×	×
virustotal	×	×	×	×
visaulThreat	×	×	×	×
安全管家在线测	×	×	×	×
腾讯安全实验室	×	×	×	×
网秦安全	×	×	×	×

从豌豆荚应用市场上下载 25 个使用 WebView 的热门应用,分别用 TaintDroid 和 WTD 进行检测。检测结果如表 4 所示,WTD 一共检测出 12 个应用使用 WebView 泄露了 IMIE 号,其中,11 个应用利用 loadUrl 通道;1 个应用利用 addJavascriptInterface 通道。而 TaintDroid 并没有检测出 WebView 泄露隐私的行为。

表 4　豌豆荚检测结果

检测工具	loadUrl	addJavascriptInterface
WTD	11	1
TaintDroid	0	0

5　结束语

WebView 为 Android 应用中 Java 和 JavaScript 代码交互提供了便利,但同时也带来了隐私泄露的风险。本文分析 WebView 外传隐私数据的原理及其可能通道,提出一种扩展 TaintDroid 的数据流跟踪框架 WTD,用于检测 WebView 泄露隐私的行为。WTD 扩展了 WebView 页面加载方法的检测,同时利用 Java 反射机制和函数执行栈跟踪机制,实现 WebView 注册对象调用敏感 API 的检测。测试结果表明,WTD 能够有效检测出 WebView 隐私数据泄露。下一步工作将重点关注 WebView 所加载的 Web 页面代码,研究其中是否可能含有其他恶意行为。

参考文献

[1] Google Inc.. Android Security Overview[EB/OL]. [2015-03-15]. https://source.android.com/devices/tech/security/index.html.

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

[3] Tan D J J, Chua T W, Thing V L L. Securing Android: A Survey, Taxonomy, and Challenges[J]. ACM Computing Surveys, 2015, 47(4):1-45.

[4] 张玉清,王　凯,杨　欢,等. Android 安全综述[J]. 计算机研究与发展,2014,51(7):1385-1396.

[5] Smalley S, Craig R. Security Enhanced (SE) Android: Bringing Flexible MAC to Android[C]//Proceedings of Network and Distributed System Security Symposium. San Diego, USA: ISOC, 2013:20-38.

[6] Shebaro B, Oluwatimi O, Bertino E. Context-based Access Control Systems for Mobile Devices[J]. IEEE Transactions on Dependable and Secure Computing, 2015, 12(2):150-163.

[7] 杨　欢,张玉清,胡予濮,等. 基于权限频繁模式挖掘算法的 Android 恶意应用检测方法[J]. 通信学报,2013,34(1):106-115.

[8] Wang Wei, Wang Xing, Feng Dawei, et al. Exploring Permission-induced Risk in Android Applications for Malicious Application Detection[J]. IEEE Transactions on Information Forensics and Security, 2014, 9(11): 1869-1882.

[9] Lu Long, Li Zhichun, Wu Zhenyu, et al. CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities[C]//Proceedings of 2012 ACM Conference on Computer and Communications Security. New York, USA: ACM Press, 2012:229-240.

[10] Arzt S, Rasthofer S, Fritz C, et al. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps[C]//Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, USA: ACM Press, 2014:29-35.

[11] Enck W, Gilbert P, Han S, et al. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones[J]. ACM Transactions on Computer Systems, 2014, 32(2):5-10.

[12] Tang Y, Ames P, Bhamidipati S, et al. CleanOS: Limiting Mobile Data Exposure with Idle Eviction[C]//Proceedings of Symposium on Operating System Design and Implementation. Berkeley, USA: USENIX Association, 2012:77-91.

[13] Qian Chenxiong, Luo Xiapu, Shao Yuru, et al. On Tracking Information Flows Through JNI in Android Applications[C]//Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Washington D. C., USA: IEEE Press, 2014:180-191.

[14] Luo Tongbo, Hao Hao, Du Wenliang, et al. Attacks on WebView in the Android System[C]//Proceedings of the 27th Annual Computer Security Applications Conference. New York, USA: ACM Press, 2011:343-352.

[15] Yu Jing, Yamauchi T. Access Control to Prevent Attacks Exploiting Vulnerabilities of WebView in Android OS[C]//Proceedings of IEEE International Conference on High Performance Computing & Communications. Washington D. C., USA: IEEE Press, 2013:1628-1633.

[16] Georgiev M, Jana S, Shmatikov V. Breaking and Fixing Origin-based Access Control in Hybrid Web/Mobile Application Frameworks[C]//Proceedings of Network and Distributed System Security Symposium. San Diego, USA: ISOC, 2014:1-8.

[17] Oracle Inc.. Trail: The Reflection API[EB/OL]. [2015-03-15]. https://docs.oracle.com/javase/tutorial/reflect/.

[18] Tutorialspoint. Java. lang. Throwable. get Stack Trace() Method[EB/OL]. [2015-03-15]. http://www.tutorialspoint.com/java/lang/throwable_getstacktrace.htm.

编辑　陆燕菲