ISSN 1000-0054 清华大学学报（自然科学版） 2016 年 第 56 卷 第 5 期 3/16
CN 11-2223/N J Tsinghua Univ (Sci & Technol), 2016, Vol. 56, No. 5 468-477

Android 恶意广告威胁分析与检测技术

韩心慧， 丁怡婧， 王东祺， 黎桐辛， 叶志远

（北京大学 计算机科学与技术研究所，北京 100080）

摘　要：Android 第三方广告框架应用广泛，但 Android 系统漏洞和 Android 第三方广告框架的逻辑缺陷严重威胁着 Android 市场安全。攻击者可以通过恶意广告获取敏感数据、触发敏感操作，甚至是以应用程序的权限执行任意代码。该文总结了 4 种 Android 恶意广告攻击方式，并针对这 4 种方式设计了一种基于后向切片算法和静态污点分析的 Android 第三方广告框架静态测量方法，以及一种基于 API Hook 和靶向 API Trace 的 Android 恶意广告敏感行为动态检测方法。基于以上研究，该文设计并实现了 Android 恶意广告威胁分析与检测系统，通过实例证明该系统能够有效地分析 Android 第三方广告框架可能存在的安全隐患，并能够动态检测 Android 恶意广告的敏感行为。

关键词：Android；恶意广告；威胁；静态分析；动态分析

中图分类号：TP 393.08　　**文献标志码**：A

文章编号：1000-0054(2016)05-0468-10

DOI：10.16511/j.cnki.qhdxxb.2016.25.003

Android malicious AD threat analysis and detection techniques

HAN Xinhui, DING Yijing, WANG Dongqi, LI Tongxin, YE Zhiyuan

(Institute of Computer Science and Technology, Peking University, Beijing 100080, China)

Abstract: Android third-party advertising frameworks are deployed in almost every Android app. The vulnerabilities of the Android OS and these advertising frameworks greatly impact the security of the Android market. The attacker can get the users' private data, trigger sensitive operations and execute arbitrary code on the device. This paper summarizes four classes of attacks in Android third-party advertising frameworks and gives two detection algorithms to discover these four classes of vulnerabilities. The first detection algorithm statically analyzes the advertising frameworks using a backward slicing algorithm and a static forward tainting analysis. The second algorithm dynamically detects malicious behavior in advertising frameworks using API hooking and targeted API tracing. An Android malicious ad security threat analysis and detection system is designed and implemented based on these two algorithms. Tests show that this system effectively discovers potential vulnerabilities in advertising frameworks and dynamically detects malicious behavior in advertisements.

Key words: Android; malicious AD; threat; static analysis; dynamic analysis

随着移动互联网的发展，Android 手机市场处于快速增长的过程，人们对 Android 应用程序的需求与依赖也与日俱增。而 Android 开发者最主要的盈利方式是在应用程序中向用户展示广告[1]。

Android 广告由第三方广告平台负责分发。首先，广告商向广告平台提交广告。然后，广告平台向应用程序开发者提供广告框架，开发者将之嵌入自己开发的应用程序中。在应用程序运行时，其中的广告框架会向对应的广告平台请求广告，广告平台返回合适的广告页面给该广告框架，最终广告框架将返回的广告页面展示在应用程序内。

然而 Android 第三方广告框架却存在着一些安全隐患。一方面，Android 系统版本的碎片化问题使其被曝光的漏洞难以被完全修补，甚至 Google 不再针对 Android 4.3 及以下版本的操作系统提供漏洞补丁[2]，导致许多漏洞仍旧广泛存在于用户的手机中[3]。另一方面，广告框架与应用程序运行在同一个进程空间，没有严格的权限隔离，恶意广告可以利用 Android 系统漏洞或广告框架的逻辑缺陷[4]，获取敏感数据、触发敏感操作，甚至是以应用程序的权限执行任意代码。

目前，应对 Android 安全威胁的主要措施是利用 Android 软件分析技术检测恶意软件，也有部分研究针对第三方广告框架和应用程序的权限隔离给出了解决方案，但针对 Android 恶意广告带来的威胁进行分析与动态检测的研究很少。

本文结合 Android 系统版本碎片化问题、An-

收稿日期：2016-01-21

基金项目：国家自然科学基金资助项目(61402125)

作者简介：韩心慧(1969—)，男，高级工程师。

E-mail: hanxinhui@pku.edu.cn

droid 广告投放机制、Android 系统漏洞，以及 Android 第三方广告框架的逻辑缺陷 4 个因素，总结了 2 种已有的基于 Android 恶意广告的攻击方式——基于 JavaScript-Binding-Over-HTTP（JBOH）漏洞的 Java 反射调用攻击和 JavaScript（JS）侧门攻击，并设计了 2 种新的攻击方式——URL 侧门攻击和基于同源策略漏洞的本地文件访问限制绕过攻击。针对这 4 种攻击方式，设计了一种基于后向切片算法和静态污点分析的 Android 第三方广告框架静态测量方法，和一种基于 API Hook 和靶向 API Trace 的 Android 恶意广告敏感行为动态检测方法。基于以上方法，本文设计并实现了 Android 恶意广告威胁分析与检测系统。

1 Android 软件与第三方广告框架分析方法

1.1 Android 软件分析

Android 软件分析技术主要包括静态分析和动态分析 2 个分支。

在静态分析方面，许多系统被设计出来以检测漏洞和隐私泄露等问题。如 Scandroid[5]、Comdroid[6]、Epicc[7] 等分析了权限泄露问题，Flowdroid[8] 等分析了隐私泄露问题。其中，Flowdroid 是搭建于 Soot[9] 系统上的静态污点跟踪系统。但是在静态分析方面，广告相关分析还较少。

在动态分析方面，TaintDroid[10] 能够进行操作系统级别的动态污点跟踪[11]。CopperDroid[12] 通过监测 Android 组件与操作系统之间的交互，对上层的软件行为进行重建并发现其中的恶意行为。Aurasium[13] 能够自动进行应用程序重打包并且加入用户级沙箱策略代码，监测应用程序是否有侵犯用户隐私安全的行为并阻止提权攻击。但这些工作只能在 Java 代码空间里进行动态分析，无法进行涉及 JavaScript 代码空间的动态分析。

1.2 Android 第三方广告框架分析

AdRisk[14] 采用静态分析揭示了 Android 第三方广告框架潜在的安全威胁。但该工作主要分析广告框架自身敏感行为，未分析恶意广告带来的威胁。Stevens 等[15] 重点研究了第三方广告平台如何对用户进行定向以获得更精确的广告投放效果，该工作分析的也是广告框架自身的隐私泄露问题。

AdDroid[16] 针对 Android 系统提出了一个能够分离应用程序与广告框架权限的新的广告框架。AdSplit[17] 对 Android 系统进行了扩展，使应用程序和其集成的广告框架能以不同的 UID 运行在不同的进程中。SandAdBox[18] 将第三方广告框架作为一个独立的应用程序安装，隔离了广告。但这些旨在将二者进行权限分离的方法都难以大规模应用，在现有的机制下，第三方广告框架仍旧与应用程序享有同样的权限。

最后，Wei 等[19] 的研究工作通过中间人攻击，可以通过 Android 系统 JavaScript-Binding-Over-HTTP 漏洞以及 Android 第三方广告框架逻辑缺陷的 JavaScript 侧门攻击手机，或从流量中收集 Android 第三方广告框架本身在投放广告的过程中产生的用户隐私数据。

相比 Wei 等[19] 的研究工作，本文的研究工作除了考虑来自中间人攻击的威胁，还着眼于 Android 原生恶意广告被投放而带来的威胁。

2 Android 恶意广告攻击方式

2.1 基于 JBOH 漏洞的 Java 反射调用攻击

在 Android 4.1 及以下版本的操作系统中，如果 Android 应用程序通过 addJavascriptInterface() 函数向 WebView 提供了一个 Java 实例作为接口，WebView 所加载的页面中的 JavaScript 代码就能够利用该 JavaScript 接口和 Java 语言的反射机制访问该接口绑定的实例的全部公有函数，包括该实例继承的所有公有函数。一旦 WebView 加载了攻击者构造的包含恶意 JavaScript 代码的页面，或者攻击者通过中间人攻击在 HTTP 流量中注入的恶意 JavaScript 代码，上述漏洞将允许攻击者以应用程序的权限执行任意 Java 代码，这将对用户隐私数据的安全造成极其严重的威胁。

此外，在 Android 3.0 以上及 4.2 以下版本的操作系统中，系统会在 WebView 中自动添加一个名为“searchBoxJavaBridge_”的 JavaScript 接口，如果广告框架没有去除该 JavaScript 接口，则此时 Android 恶意广告仍旧能够利用上述漏洞实施攻击。

2.2 JavaScript 侧门攻击

Android 4.2 及以上版本的操作系统提供了@JavascriptInterface 注释机制，用于显式声明 JavaScript 接口所绑定的 Java 实例中允许 JavaScript 代码调用的公有函数。然而，一些 Android 第三方广告框架会在提供给 Java Script 代码调用的 Java 函数中调用 Android 系统敏感 API，那么攻击者同样可以利用恶意 JavaScript 代码对这些涉及敏感操作

的 JavaScript 接口函数进行任意调用。这种形式的 Android 系统敏感 API 的暴露被称为 JavaScript 侧门[20],故这种攻击被称为基于 JavaScript 侧门的攻击。

2.3 URL 侧门攻击

在 Android 4.1 及以下版本的操作系统中,第三方广告框架所使用的@JavascriptInterface 被认为是普通注释,JBOH 漏洞所带来的安全威胁依然存在。于是,一些 Android 第三方广告框架彻底摒弃了在 WebView 中添加 JavaScript 接口,而是通过 URL 与 JavaScript 代码空间进行通信。一方面,WebView 中的 JavaScript 代码只要将所需传递的信息放到一个 URL 中并让 WebView 加载该 URL 就能实现与 Java 代码空间的信息传递,Android 第三方广告框架实现的 ShouldOverrideUrlLoading() 函数会对 URL 进行解析并执行相应的操作;另一方面,Android 第三方广告框架只要将所需传递的信息放到一个 URL 中并通过 WebView.loadUrl() 函数并让 WebView 加载,就能实现与 JavaScript 空间的信息传递。

一些 Android 第三方广告框架会调用操作系统敏感 API,并通过上述方法将敏感数据传递给 WebView 中的 JavaScript 代码,同样攻击者也可以利用 JavaScript 代码构造特定的 URL 向第三方广告框架获取相同的敏感数据,攻击者还可以通过 URL 的构造触发系统敏感操作。本文将通过 URL 导致的 Android 系统敏感 API 的调用称为 URL 侧门,将对应的攻击称为 URL 侧门攻击。

2.4 基于同源策略漏洞的本地文件访问限制绕过攻击

同源策略保证了一个页面中的脚本只能访问来自同一站点的资源,阻止了其利用另一个页面的文档对象模型访问后者的敏感数据[21]。Android 4.3 及以下的操作系统存在浏览器同源策略绕过漏洞,该漏洞利用的是 Android 浏览器在解析 URL 时处理空字节的逻辑缺陷,在一个页面将一段恶意的 JavaScript 代码作为 URL 传递给 IFRAME 标签中的非同源页面打开时,只要在 URL 字符串起始位置添加"\u0000"就可绕过同源策略的限制[22]。若 WebView 未禁止对本地文件的访问,通过同源策略漏洞,恶意 Javascript 代码还可以访问本地文件。

3 基于后向切片算法和静态污点分析的 Android 广告框架测量方法

3.1 后向切片算法原理

后向切片算法[23]可以将 Android 应用程序反编译为 Dalvik 字节码并构建函数调用关系图,查看程序中是否有对特定函数的调用,利用定义使用链对调用函数时所传入的参数的值进行分析,判定该值是否导致了相应的漏洞,以检测 JBOH 漏洞和同源策略漏洞。在实际环境下,利用 Soot 提供的用于生成函数调用关系图和指令定义使用链的 API,实现基于后向切片算法的 Android 第三方广告框架分析。

3.2 静态污点分析原理

静态污点分析是一种基于数据流分析的软件分析技术,主要用于分析程序运行过程中敏感数据的传播路径,以及判断程序的运行是否导致了敏感数据的泄露。静态污点分析主要关注 3 个方面,分别是污点数据的源 Source 点、污点数据的传播过程以及污点数据的槽 Sink 点,针对 Android 应用程序的静态污点分析如下:

1) 一般将用于获取用户隐私数据的函数作为 Source 点,如用于获取设备标识信息的函数 getDeviceId()等。

2) 通过扫描并分析程序代码对污点标记的传播过程来进行记录与维护。

3) 一般将导致用户隐私数据泄露的函数作为 Sink 点,如发送短信的函数 sendTextMessage()等。

3.3 静态污点分析的回调函数指令扩展

1) FlowDroid 静态污点分析框架的局限性。

利用并扩展 FlowDroid 静态分析框架,对 Android 广告框架进行测量。FlowDroid 框架构建了一个虚拟的 Dummy Main 函数,用来模拟应用程序中每个组件的生命周期,并针对 Android 应用程序生成对应的控制流图。但 FlowDroid 只在 Java 代码空间内对应用程序的控制流进行分析,无法分析 Android 恶意广告可能带来的威胁。

针对这个情况,通过对 Dummy Main 函数中回调函数的指令扩展,针对第 2 部分总结的 Android 恶意广告可能带来的威胁,添加了模拟调用 JavaScript 接口回调函数以及 WebView 中敏感回调函数的指令。

2）Util 类的构建。

为了在 Dummy Main 函数中针对 JavaScript 接口和 WebView 中的回调函数进行模拟调用以及静态污点分析，除了将常见的 Android 系统函数按需要定义为 Source 点和/或 Sink 点，还需要构建自定义的 Source 点和 Sink 点。在集成了 Android 第三方广告框架的应用程序中添加一个通用的 Util 类，用来定义所需的 Source 点和 Sink 点。

图 1 展示了用于针对静态污点分析的回调函数指令扩展的 Util 类的实现，针对 Java 语言中常见类型分别定义了 Source 函数和 Sink 函数，其他类型污点对应的 Source 函数统一返回 Object 类的一个实例，对应的 Sink 函数的参数统一为 Object 类型。

```
public class Util {
    // 自定义 Source 点
    public byte sourceByte()          { return 1; }
    public short sourceShort()        { return 2; }
    public int sourceInt()            { return 4; }
    public long sourceLong()          { return 8; }
    public float sourceFloat()        { return 3; }
    public double sourceDouble()      { return 5; }
    public boolean sourceBoolean()    { return false; }
    public char sourceChar()          { return '6'; }
    public String sourceString()      { return "7"; }
    public Object sourceObject()      { return new Object(); }
    // 自定义 Sink 点
    public void sinkByte(byte param)          { }
    public void sinkShort(short param)        { }
    public void sinkInt(int param)            { }
    public void sinkLong(long param)          { }
    public void sinkFloat(float param)        { }
    public void sinkDouble(double param)      { }
    public void sinkBoolean(boolean param)    { }
    public void sinkChar(char param)          { }
    public void sinkString(String param)      { }
    public void sinkObject(Object param)      { }
}
```

图 1 Util 类的实现

3）Dummy Main 函数的指令扩展。

首先，将上述 Util 类定义的 Source 函数和 Sink 函数添加到原始的 Source 点和 Sink 点的集合中。接着，在 Dummy Main 函数中添加模拟调用 Android 第三方广告框架所添加的 JS 接口回调函数的指令，以及所创建的 WebView 中敏感回调函数的指令，利用 Util 类的 Source 函数创建污点数据，利用 Sink 函数将这些回调函数的返回值传播到一个 Sink 点。图 2 为在 Dummy Main 函数中添加回调函数模拟调用指令的伪代码。在实际情况下，实现上述模拟调用指令添加的代码较为复杂，需要利用 Soot 的 API。

```
1  Function createDummyMain() {
2      // 在 Dummy Main 函数中添加回调函数模拟调用指令的伪代码
3      util = new Util();
4      paramVar1 = util.source[callback_function_param1_type]();
5      paramVar2 = util.source[callback_function_param2_type]();
6      returnVar = [callback_function]( paramVar1, paramVar2);
7      util.sink[callback_function_return_type](returnVar);
8  }
```

图 2 Dummy Main 函数中回调函数模拟调用指令的伪代码

3.4 针对 Android 广告框架的测量方法

1）针对 JBOH 漏洞的静态分析。

在对集成了第三方广告框架的 Android 应用程序进行静态分析并判断其是否可能包含该漏洞的时候，需要以下 3 个步骤。

a）针对 WebSettings. setJavaScriptEnabled( ) 函数，利用后向切片算法查找参数来源，判断布尔参数是否为 true。

b）针对 WebView. removeJavascriptInterface( )函数，利用后向切片算法查找参数来源，判断字符串参数是否为“searchBoxJavaBridge_”。

c）判断 WebView. addJavascriptInterface( )函数是否出现在第三方广告框架中。

在 WebSettings. setJavaScriptEnabled( ) 函数参数为 true，未调用 WebView. removeJavascriptInterface( ) 移除 “searchBoxJavaBridge_”，或是 WebView. addJavascriptInterface( )存在的情况下，存在 JBOH 漏洞。

2）针对 JavaScript 侧门攻击的静态分析。

首先，对 Android 应用程序进行反编译得到 SMALI 文件。抽取 SMALI 文件中的 JS 接口函数，在 Dummy Main 中进行回调函数的指令扩展，模拟 JS 侧门的调用。利用 Util 类中自定义的 Source 点对 JS 侧门的参数污点化，利用 Util 类的 Sink 点模拟 JS 侧门的返回值的泄露。除此以外，加入针对 Android 系统预定义的 Source 点集合和 Sink 点集合，并将 WebView. loadUrl(String)函数额外作为 Sink 点进行标记。如果经过静态污点分析得到了从 Source 点到 Sink 点的传播路径，则该第三方广告框架可能存在获取用户的敏感信息或调

用系统敏感 API 的情况。

3) 针对 URL 侧门攻击的静态分析。

与分析 JavaScript 侧门类似,使用污点跟踪分析 URL 侧门中的敏感行为。首先通过分析 Android 程序的 SMALI 文件,找到 shouldOverrideUrlLoading()函数。在 Dummy Main 中模拟 shouldOverrideUrlLoad-ing(WebView, String)的调用,将参数中的 String 利用 Source 函数进行污点化,将 LoadUrl 函数作为 Sink 点进行分析。此外,还使用了针对 Android 系统敏感 API 定义的 Source 点集合和 Sink 点集合。通过这种方式,分析第三方广告框架通过 URL 侧门攻击获取用户的敏感信息和调用系统敏感 API 的情况。

4) 针对同源策略漏洞的静态分析。

对同源策略漏洞的分析需要针对 WebSettings. setAll-owFileAccess(boolean)函数,利用后向切片算法查找其参数的来源,如果参数的值是 true,则集成了该第三方广告框架的应用程序在 Android 4.3 及以下版本的操作系统中可能会因为同源策略漏洞的存在,面临本地文件被任意访问的威胁。

4 基于 API Hook 和靶向 API Trace 的 Android 恶意广告敏感行为动态检测方法

4.1 基于 API Hook 的 Android 广告框架运行时漏洞验证

API Hook 可以通过对 WebView 中特定函数的 Hook,在运行时对 Android 第三方广告框架是否引入了 JBOH 漏洞和同源策略漏洞进行准确的检测。本文基于 Xposed 动态劫持框架[24]实现了该功能。

1) 针对 JBOH 漏洞的动态验证。

基于 Xposed 框架实现针对 JBOH 漏洞的动态验证模块,该模块需要以下 3 个步骤:

a) 对 WebSettingsClassic. setJavaScriptEnabled(boolean)函数进行 Hook,分析程序运行时是否调用了该函数并且参数值为 true。

b) 对 WebView. removeJavascriptInterface(String)函数进行 Hook,分析程序运行时是否调用了该函数并且参数值为"searchBoxJavaBridge_"。

c) 对 WebView. addJavascriptInterface(Object, String)函数进行 Hook,分析程序运行时是否调用了该函数。

当 WebSettingsClassic. setJavaScriptEnabled(boo-lean)函数被调用并且参数值为 true,"search-BoxJavaBri-dge_"未被移除或者添加了 Java 对象供 Javascript 调用,则存在 JBOH 漏洞。

2) 针对同源策略漏洞的动态验证。

针对 WebSettingsClassic. setAllowFileAccess(boo-lean)函数进行 Hook,如果程序调用了该函数并且参数值为 true,那么集成了该第三方广告框架的应用程序在 Android 4.3 及以下版本的操作系统中会因为同源策略漏洞的存在,被攻击者实施基于该漏洞的本地文件访问限制绕过攻击从而访问任意的本地文件。

4.2 基于靶向 API Trace 的 Android 敏感 API 调用源分析

靶向 API Trace 方法能够在发生漏洞攻击时,在 JavaScript 接口、Android 系统敏感 API 等特定函数被调用时准确地分析其调用来源是否为 JavaScript 代码或 URL 通信。

1) WebView 中 JavaScript 接口调用机制。

通过分析 Android 4.1 版本操作系统源代码,总结了 WebView 中 JavaScript 代码对 JavaScript 接口的调用机制。如图 3 所示,对于 WebView 中的 JavaScript 代码对 JavaScript 接口函数的调用,Android 系统主要交由 JWebCoreJavaBridge 类进行处理。因此,将该 Java 类作为来自 WebView 中的 JavaScript 代码对第三方广告框架定义的 JavaScript 接口函数调用的标识。

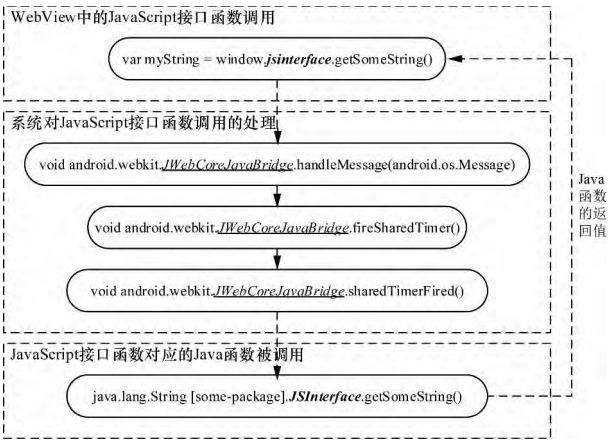

图 3 WebView 中 JavaScript 代码调用 JavaScript 接口函数的处理机制

2) 针对 Android 应用程序的靶向 API Trace 方法。

针对 Android 应用程序的靶向 API Trace 方法

基于 Xposed 框架和 API Trace 原理，用来分析 API 的调用是否来自某个特定的调用源，也就是所谓的“靶”。

靶向 API Trace 方法首先对 FlowDroid 提供的 Source 函数和 Sink 函数列表进行筛选和补充生成 Android 系统敏感 API 列表。其次，基于 Xposed 框架对列表中的敏感 API 进行 Hook，在每次敏感 API 执行之前获取函数调用栈并进行 API Trace，并结合“JWebCoreJavaBridge”关键字进行分析，判断敏感 API 的调用是否来自 WebView 中的 JavaScript 代码空间。

但是这种方法可能伴随着诸多问题，在实际情况下无法直接采用，需要改进与优化。

3）基于线程标记算法与计数器算法的靶向 API Trace 方法优化。

在 Android 应用程序运行的过程中，有些作为敏感 API 的 Android 系统函数会在正常情况下被广泛地调用。在使用靶向 API Trace 方法进行分析时，这些“常用”的系统敏感 API 被频繁地调用并分析，会导致系统资源的大量消耗。针对此问题可以进行 2 点优化，如图 4 所示。

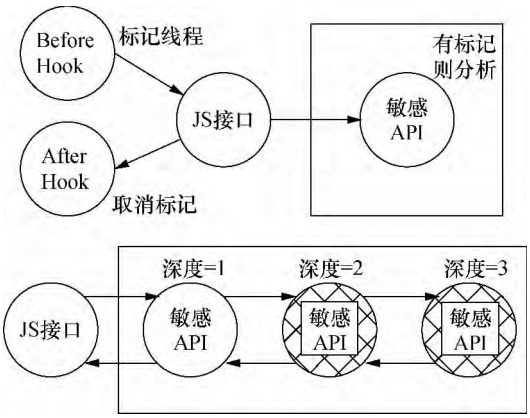

图 4 靶向 API Trace 优化方法

在线程标记算法中，为了将分析范围局限在 JavaScript 接口函数中的敏感 API，利用 Android 系统线程单步运行的原理，在对 JavaScript 接口函数进行 Hook 时，在 beforeHookedMethod()函数中对线程进行标记，在 afterHookedMethod()函数中取消标记，并只对拥有线程标记的敏感 API 进行分析。

在计数器算法中，为了消除敏感 API 的相互调用或递归调用并降低系统资源消耗，在对敏感 API 进行 Hook 的时候，只对线程函数调用栈内深度为 1 的敏感 API 调用进行进一步的处理。

4.3 Android 恶意广告敏感行为动态分析与检测

1）针对 Java 反射调用攻击的动态分析与检测。

作为 Java 反射调用攻击的跳板，Object.getClass()函数被用于通过 JavaScript 接口函数调用其他类的函数。对 getClass()函数进行 Hook，当针对该函数的调用来自 JavaScript 代码空间时，进行日志记录，可以实时发现 JBOH 漏洞攻击。

2）针对 JavaScript 侧门攻击的动态分析与检测。

如果 Android 应用程序集成的第三方广告框架开放了 JavaScript 接口并在接口函数中调用了系统敏感 API，那么 Android 恶意广告可以通过对这些接口函数的任意调用触发相应的敏感 API。针对这种 JavaScript 侧门攻击的动态分析与检测分为以下 3 个步骤。

a）对 WebView.addJavascriptInterface(Object, String)函数进行 Hook，通过第一个参数，也就是 JavaScript 接口类的实例，得到 JS 可以调用的 Java 对象。通过分析对象，得到该接口类的待分析的 JavaScript 接口函数。

b）对上一步得到的 JavaScript 接口函数进行 Hook，在函数执行前的 Hook 点 beforeHookedMethod()函数中，判断当前线程的函数调用栈中是否包含“JWebCoreJavaBridge”关键字，包含则将该线程标记为源自 JavaScript 代码空间的线程，否则直接返回。

c）对预定义的 Android 系统敏感 API 列表中的函数进行 Hook，当一个敏感 API 被 JavaScript 接口函数直接或间接地调用，并且相比其他所有敏感 API，该敏感 API 是第一个被调用的时候，说明是 JavaScript 接口函数触发了该敏感 API，并进行日志记录。

3）针对 URL 侧门攻击的动态分析与检测。

分析 URL 侧门与分析 JS 侧门类似，通过 HookWebView.setWebViewClient(WebViewClient)找到 WebViewClient 实例，该实例的 shouldOverrideUrlLoading()为待分析函数。之后，分析敏感 API 是否由 shouldOverrideUrlLoading 触发。

4）针对本地文件访问限制绕过攻击的动态分析与检测。

在 Android 4.3 及以下版本的操作系统中，Android 恶意广告能够通过利用同源策略漏洞发起本地文件访问限制绕过攻击。Android 系统会将用

于访问文件的 URL 交给 WebResourceResponse.shouldIn-terceptRequest(String)函数处理。检测时对该函数进行 Hook,当针对该函数的调用来自 JavaScript 代码空间时,进行日志记录。

5 系统实现与实验分析

5.1 系统实现

针对 Android 第三方广告框架的静态分析测量方法,以及 Android 恶意广告敏感行为的动态分析检测方法,进行了系统性的实现。

图 5 展示了 Android 恶意广告威胁分析与检测系统的架构图。图 5 的上半部分是预处理步骤,通过集成广告框架,添加污点跟踪所需 Util 类得到待分析 App 应用程序样本。之后,反编译分析 SMALI 文件,提取其中定义的所有关键 API。

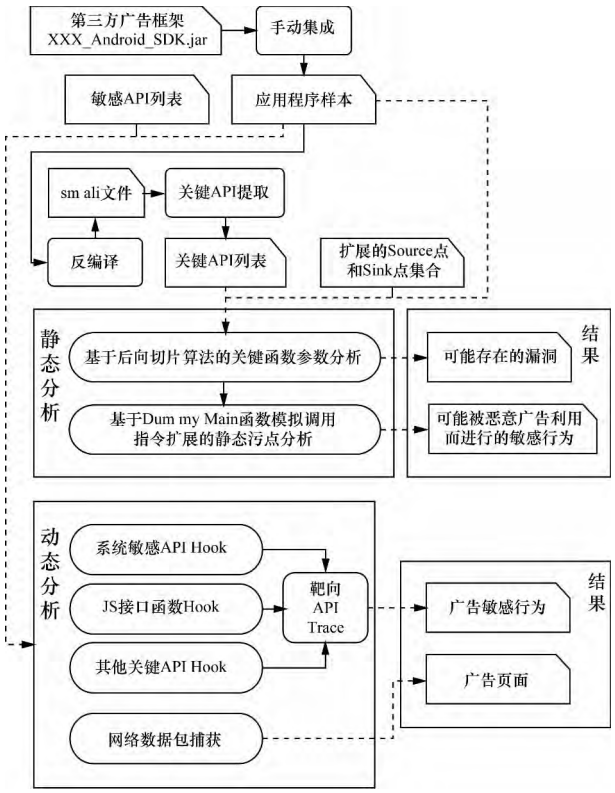

图 5 Android 恶意广告威胁分析与检测系统架构图

图 5 中间是静态分析部分。静态分析的输入主要包括应用程序样本、关键 API 列表、扩展的 Source 点和 Sink 点集合。检测系统在 FlowDroid 默认配置文件的基础上,添加了 Util 类中定义的 10 个 Source 函数和 10 个 Sink 函数,还添加了与 URL 侧门攻击相关的 loadUrl()函数作为额外的 Sink 点。在静态分析的过程中,系统采用后向切片算法对 WebView 相关设置函数的参数进行 JBOH 漏洞和同源策略漏洞的分析。通过扩展 FlowDroid 静态污点分析框架,分析 URL 侧门和 JS 侧门。

图 5 下方是动态分析部分。在动态检测时,将每个 Android 第三方广告框架手动集成到一个真实的应用程序中,为了通过广告平台的审核,选择当下较为火爆的 2 048 小游戏作为宿主应用程序。同时,为了分析 JBOH 漏洞,将动态分析部分的实验环境设置为 Android 4.1 版本的操作系统。通过对 FlowDroid 提供的 Source 点和 Sink 点列表的手动筛选,确定了一份较为完整的 Android 系统敏感 API 列表,并基于 Xposed 框架实现了对列表中所有敏感 API 的 Hook,同时基于靶向 API Trace 方法实现了对每个 Hook 点的函数调用源分析,并以此确定来自 JavaScript 接口以及 URL 通信的敏感操作。

5.2 基于静态分析的 Android 广告框架测量实验

针对国内常见的 37 个 Android 第三方广告平台所提供的广告框架进行测量实验,对每个广告框架可能存在的安全威胁进行静态分析,具体结果见表 1。

表 1 基于静态分析的 Android 广告框架测量实验结果

广告框架	JBOH 漏洞	JS 侧门	URL 侧门	同源策略漏洞	威胁数量
帷千支媒	√	√	√	√	4
万普世纪	√		√	√	3
有米	√			√	2
指盟	√			√	2
聚米					0
百度	√	√		√	3
酷果	√	√		√	2
趣米	√			√	3
多盟	√			√	2
果盟	√			√	2
力美	√			√	2
哇棒	√		√	√	3
广点通				√	1
易传媒	√		√	√	3
微盟聚无线	√			√	2
搜趣无线	√	√		√	3
APPFlood	√		√	√	3
畅思	√		√	√	3

（续表）

广告框架	JBOH 漏洞	JS 侧门	URL 侧门	同源策略漏洞	威胁数量
点入	√			√	2
91 斗金	√			√	2
百灵欧拓	√			√	2
架势	√			√	2
艾德思奇	√		√	√	3
乐点	√			√	2
云测					0
百通	√			√	3
亿动智道	√	√		√	3
点击移动	√			√	2
米迪					0
指点	√			√	2
易积分	√	√		√	3
第七传媒					0
安沃	√		√	√	3
威朋	√	√		√	3
飞云	√			√	2
优友	√	√	√	√	4
芒果	√		√	√	3

图 6 是本文研究的 4 种 Android 恶意广告威胁在上述 37 个广告框架中的分布情况，其中可能包含 JBOH 漏洞和同源策略漏洞的广告框架占比都在 85%以上，JavaScript 侧门和 URL 侧门则分别可能在 19%和 27%的广告框架中存在。

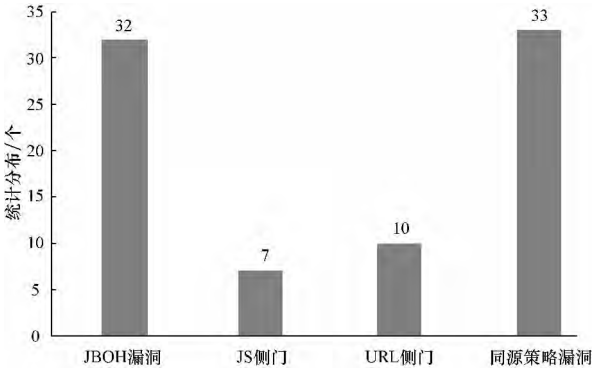

图 6 基于静态分析的 Android 广告框架测量实验结果统计

对于 JavaScript 侧门威胁，在静态分析实验中，根据污点传播路径，进行手动验证，发现可能触发定位、获取标识、startActivity、读写文件、访问日历等敏感行为。

对于 URL 侧门威胁，在静态分析实验中，根据污点传播路径，进行手动验证，发现可能触发发送短信、打电话、发送邮件、安装应用、打开 URL、下载等敏感行为。

综合上述实验结果，Android 第三方广告框架可能存在的 JavaScript 侧门和 URL 侧门涉及了很多系统敏感操作，如发送短信、打电话等，如果 Android 恶意广告对这些侧门进行滥用，则会严重威胁用户的隐私安全。

在静态分析中，JBOH 漏洞和同源策略漏洞未发现误报与漏报，JS 侧门和 URL 侧门通过对污点传播路径手工验证分析得出。

5.3 基于动态分析的 Android 广告敏感行为与检测实验

针对 11 个成功通过广告平台审核的应用程序进行动态分析的实验。

1）JBOH 漏洞和同源策略漏洞的验证。

通过对 WebView 相关设置函数的 Hook，记录函数的调用和实时传递的参数，根据程序运行时的日志，可以验证 JBOH 漏洞和同源策略漏洞是否存在。表 2 是关于广告框架漏洞验证的分析结果。可以看到，除广点通广告框架仅存在同源策略漏洞以外，其他广告框架均同时存在 JBOH 漏洞和同源策略漏洞。这与静态分析的结果一致，也验证了针对 Android 广告框架可能包含的漏洞进行静态分析的有效性。

表 2 基于动态分析的 Android 广告框架漏洞验证实验结果

广告框架	JBOH 漏洞	同源策略漏洞
帷千动媒	√	√
万普世纪	√	√
有米	√	√
百度	√	√
趣米	√	√
广点通		√
易传媒	√	√
艾德思奇	√	√
百通	√	√
亿动智道	√	√
落果	√	√

2）Android 恶意广告敏感行为的检测。

虽然恶意广告的危害十分严重，但由于实际网络中广告数量十分庞大，恶意广告难以捕获，因此本文并未实际捕获到恶意广告。但在实验过程中捕获到了一些广告平台为了广告的精准投放而在广告页面中主动添加的对 JavaScript 侧门和 URL 侧门

的调用。如图 7 所示，选取某广告框架进行实验，在针对该广告框架的动态分析实验中，捕获了大量的敏感 JavaScript 接口函数调用，图 7 中截取的是用于获取用户所在蜂窝网络位置的系统函数被调用的日志记录。

```
Call--- public native android.telephony.CellLocation android.telephony.TelephonyManager.getCellLocation() from JS
----dalvik.system.VMStack.getThreadStackTrace(Native Method)
----java.lang.Thread.getStackTrace(Thread.java:591)
----xposed.mods.game.Main.printStackOfSS(Main.java:70)
----xposed.mods.game.Main$1.beforeHookedMethod(Main.java:124)
----de.robv.android.xposed.XposedBridge.handleHookedMethod(XposedBridge.java:611)
----android.telephony.TelephonyManager.getCellLocation(Native Method)
----com.wqmobile.sdk.a.d.a(SourceFile:334)
----com.wqmobile.sdk.a.d.getMcc(SourceFile:370)
----de.robv.android.xposed.XposedBridge.invokeOriginalMethodNative(Native Method)
----de.robv.android.xposed.XposedBridge.handleHookedMethod(XposedBridge.java:631)
----com.wqmobile.sdk.a.d.getMcc(Native Method)
----com.wqmobile.sdk.a.d.getReqInfosString(SourceFile:892)
----de.robv.android.xposed.XposedBridge.invokeOriginalMethodNative(Native Method)
----de.robv.android.xposed.XposedBridge.handleHookedMethod(XposedBridge.java:631)
----com.wqmobile.sdk.a.d.getReqInfosString(Native Method)
----android.webkit.JWebCoreJavaBridge.sharedTimerFired(Native Method)
----android.webkit.JWebCoreJavaBridge.sharedTimerFired(Native Method)
----android.webkit.JWebCoreJavaBridge.fireSharedTimer(JWebCoreJavaBridge.java:92)
----android.webkit.JWebCoreJavaBridge.handleMessage(JWebCoreJavaBridge.java:108)
----android.os.Handler.dispatchMessage(Handler.java:99)
----android.os.Looper.loop(Looper.java:137)
----android.webkit.WebViewCore$WebCoreThread.run(WebViewCore.java:827)
----java.lang.Thread.run(Thread.java:856)
```

图 7 某广告框架敏感行为捕获

动态分析得到的日志记录可供研究人员进行后续分析，对于敏感 JavaScript 接口函数调用的记录，如果对应的接口函数语义明确，则可以直接得出恶意广告所触发的敏感行为；如果语义不明确，则需要借助日志中每次调用记录后附加的详细的调用栈信息来判断具体的敏感行为。

3) 通过某广告平台进行的恶意广告投放。

为了验证动态分析方法的有效性，选取某广告平台进行实验，利用该广告平台的自主广告投放功能，基于 4 种 Android 恶意广告攻击方式，自定义 4 种恶意广告页面，分别进行了自主广告的投放。根据日志，所有的攻击行为均被捕获，证明了使用基于 API Hook 和靶向 API Trace 的方法对 Android 恶意广告敏感行为进行动态检测的有效性。

4) 通过中间人攻击进行模拟实验。

此外，通过中间人攻击，本文向集成某广告框架的应用程序投放恶意广告，模拟了利用 JBOH 漏洞、JavaScript 侧门和 URL 侧门进行攻击。根据日志，所有恶意行为均被捕获，进一步证明了使用基于 API Hook 和靶向 API Trace 的方法对 Android 恶意广告敏感行为进行动态检测的有效性。

5) 动态分析的误报率与漏报率的分析。

尽管动态分析实验中所使用的 Android 系统敏感 API 列表较为完善，但不排除有个别遗漏。如果存在敏感 API 遗漏的情况，则可能会漏报恶意广告的敏感行为。在实验范畴内，动态分析部分没有漏报。另外，由于 Android 广告平台会自行向广告页面中添加敏感 JavaScript 接口函数或敏感 URL 通信机制的调用以实现广告的精准投放，因此在动态分析中会在针对 JavaScript 侧门和 URL 侧门的检测结果中产生误报。但由于广告平台行为模式相对固定，分析人员在分析的时候可以将误报的情况排除。同时，Android 广告平台所添加的敏感行为不会涉及 JBOH 漏洞和同源策略漏洞，因此对这 2 种漏洞的利用行为的动态检测不会出现误报，一旦捕获到这 2 种行为，即可证明攻击者在广告中添加了恶意的 JavaScript 代码实施了攻击。

6 结 论

本文对 Android 恶意广告可能带来的安全威胁进行了深入研究，结合 Android 系统版本碎片化问题、Android 广告投放机制、Android 系统漏洞，以及 Android 第三方广告框架的逻辑缺陷 4 个方面因素，总结了 4 种已有的基于 Android 恶意广告的攻击方式。针对以上攻击，设计了一种基于后向切片算法和静态污点分析的 Android 第三方广告框架静态测量方法，以及一种基于 API Hook 和靶向 API Trace 的 Android 恶意广告敏感行为动态检测方法，并实现了 Android 恶意广告威胁分析与检测系统。最后通过一些实例证明该系统能够有效地分析 Android 第三方广告框架可能存在的安全隐患，并能够动态检测 Android 恶意广告触发的敏感行为。

参考文献 (References)

[1] Manoogian J. How free apps can make more money than paid apps [Z/OL]. (2015-6-10). http://techcrunch.com/2012/08/26/how-free-apps-can-make-more-money-than-paid-app s/.

[2] Hruska J. Google throws nearly a billion Android users under the bus, refuses to patch OS vulnerability [Z/OL]. (2015-6-10). http://www.extremetech.com/mobile/197346-google-throws-nearly-a-billion-android-users-under-the-bus-refuses-to-patch-os-vulnerability.

[3] Vidas T, Votipka D, Christin N. All your droid are belong to us: A survey of current Android attacks [C]// Proceedings of the 5th USENIX Workshop on Offensive Technologies (WOOT 2011). San Francisco, USA: USENIX, 2011: 81-90.

[4] AVL 团队. 广告件发展现状分析 [Z/OL]. (2015-06-10). http://blog.avlyun.com/2015/01/2079/ malicious-adware/. AVL Team. Analysis of the development of adware [Z/OL]. (2015-06-10). http://blog.avlyun.com/2015/01/2079/malicious-adware/. (in Chinese)

[5] Fuchs A P, Chaudhuri A, Foster J S. Scandroid: Automated security certification of Android applications [R]. Maryland: University of Maryland, 2009.

[6] 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. Washington D C, USA: ACM, 2011: 239-252.

[7] 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]// Proceedings of the 22nd USENIX Security Symposium. Washington D C, USA: USENIX, 2013: 543-558.

[8] 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. Edinburgh, UK: ACM, 2014: **49**(6): 259-269.

[9] Soot Developers. Soot [Z/OL]. (2015-6-10). http://sable.github.io/soot/.

[10] 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* (*TOCS*), 2014, **32**(2): 5.

[11] Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software [C]//Proceedings of the 12th Network and Distributed System Security Symposium (NDSS'05). San Diego, California, USA: ISOC, 2005.

[12] Reina A, Fattori A, Cavallaro L. A system call-centric analysis and stimulation technique to automatically reconstruct Android malware behaviors [J]. *EuroSec*, April, 2013.

[13] Xu R, Saïdi H, Anderson R. Aurasium: Practical policy enforcement for Android applications [C]// USENIX Security Symposium. Tucson, Arizona, USA: USENIX, 2012: 539-552.

[14] Grace M C, Zhou W, Jiang X X, et al. Unsafe exposure analysis of mobile in-app advertisements [C]// Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks. Tucson, Arizona, USA: ACM, 2012: 101-112.

[15] Stevens R, Gibler C, Crussell J, et al. Investigating user privacy in Android ad libraries [C]// Workshop on Mobile Security Technologies (MoST). San Francisco, USA: IEEE CS Technical Committee on Security and Privacy, 2012.

[16] Pearce P, Felt A P, Nunez G, et al. Addroid: Privilege separation for applications and advertisers in Android [C]// Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security. Seoul, Korea: ACM, 2012: 71-72.

[17] Shekhar S, Dietz M, Wallach D S. AdSplit: Separating smartphone advertising from applications [C]// USENIX Security Symposium. Tucson, Arizona, USA: USENIX, 2012: 553-567.

[18] Kawabata H, Isohara T, Takemori K, et al. Sandbox: Sandboxing third party advertising libraries in a mobile application [C]// Communications (ICC), 2013 IEEE International Conference on IEEE. Budapest, Hungary: IEEE, 2013: 2150-2154.

[19] WEI Tao, ZHANG Yulong, XUE Hui, et al. Sidewinder targeted attack against Android in the golden age of ad libraries [C]//Proceedings of Black Hat USA 2014. Las Vegas, USA, 2014.

[20] Fireeye. JS-Binding-Over-HTTP vulnerability and JavaScript sidedoor: Security risks affecting billions of Android app downloads [Z/OL]. (2015-6-10). https://www.fireeye.com/blog/threat-research/2014/01/js-binding-over-http- vulnerability-and-javascript-sidedoor. html.

[21] Wikipedia Contributors. Same-origin policy [Z/OL]. (2015-6-10). http://en.wikipedia.org/wiki/Same-origin_policy.

[22] CVE Details. Vulnerability details: CVE-2014-6041 [Z/OL]. (2015-6-10). http://www.cvedetails.com/cve/CVE-2014-6041/.

[23] Bianchi A, Corbetta J, Invernizzi L, et al. What the app is that? Deception and countermeasures in the Android user interface [C]//2015 IEEE Symposium on Security and Privacy. San Jose, CA, USA: IEEE, 2015: 931-948.

[24] Xposed. Xposed module repository [Z/OL]. (2015-6-10). http://repo.xposed.info/module/de.robv.android.xposed.installer.