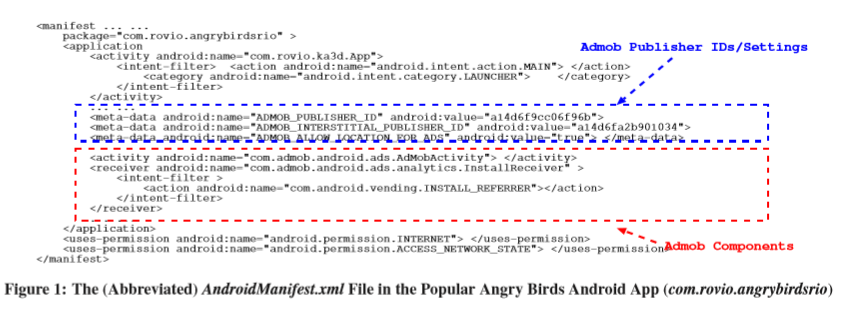
移动应用内广告的不安全曝光分析

摘要

近年来，智能手机销售出现了惊人的增长，伴随着大量的智能手机应用（或应用程序）的普及。最终用户或消费者被这些设备和相关应用提供的许多有趣的功能所吸引。这些应用程序的开发人员可以通过直接销售应用程序或通过嵌入智能手机平台上的许多广告库之一来获益。在本文中，我们关注这些嵌入式或应用程序内广告库（以下简称“广告库”）潜在的隐私和安全风险。为此，我们研究了流行的Android平台，并在2011年3月至5月期间从Android市场收集了10万个应用程序。在这些应用中，我们确定了100个代表性的应用内广告库（嵌入在52.1％的应用中），并进一步开发一个名为AdRisk的系统来系统地识别潜在的风险。特别是，我们首先将嵌入式广告库与其主机应用程序脱钩，然后应用我们的系统来静态检查广告库的风险，从上传敏感信息到远程（广告）服务器到从互联网来源执行不受信任的代码。我们的研究结果表明，大多数现有的广告库收集私人信息：其中一些数据可能用于合法的定位目的（即用户的位置），而其他数据更难证明，例如用户的通话记录，电话号码，浏览器书签，甚至安装在手机上的应用程序列表。此外，一些图书馆利用不安全的机制直接从互联网获取和运行代码，这将立即导致严重的安全隐患。我们的调查显示，嵌入式广告图书馆和主机应用程序之间的共生关系是这些暴露风险背后的主要原因。这些结果清楚地表明需要更好地规范广告库在Android应用程序中的集成方式。

1. 引言

近几年来，智能手机销量呈现爆炸式增长。据Gartner介绍，这些设备的销售额在2011年第二季度同比增长了74％[6]，去年底，智能手机已经超过了个人电脑历史上第一次[25]。显然，由于这些移动设备的便利性和强大功能，市场已经拥抱这些移动设备：这些传感器丰富的设备足够小，可以像传统的手机一样携带，而且为用户提供比简单的短信或基本电话更广泛的功能。而且，它们被下载和运行提供其他有用功能的第三方应用程序的能力定义。换句话说，智能手机用户不是仅限于电话制造商，运营商或有限的功能提供的功能，而是可以分配成千上万个应用程序，这些应用程序是为了在制作和分发设备方面不可预见的目的而设计的。此外，平台供应商（例如Google和Apple）也提供集中的应用程序市场，用户可以轻松浏览浏览，搜索，购买，下载和安装这些应用程序。作为移动生态系统的一部分，应用程序开发人员在很大程度上受到财务激励的驱动，将应用程序提交给集中的应用程序市场，供用户访问。请注意，在Android平台上，几乎三分之二的应用程序可以免费下载[5]。为了补偿他们的工作，许多应用程序开发人员在其应用程序中包含一个广告库（也称为广告库）。在运行时，广告库与广告网络的服务器进行通信，以请求广告进行展示，并可能会另外发送有关应用用户的分析信息。 （为了简单起见，我们使用术语“广告库”来表示广告库和分析库。）然后，广告网络会根据衡量每个应用给予网络及其广告客户的曝光程度的指标，持续支付开发人员费用。在本文中，我们的目的是研究现有的应用内广告图书馆，并评估潜在的风险。具体来说，我们专注于Android平台，并确定Android上流行的广告库可能对用户的隐私和安全带来什么风险。为此，我们在三个月内（即2011年3月至5月）收集了来自Android官方Android Market的100 000个应用程序。在这些应用程序中，我们识别并提取了100个代表性广告库，用于52,067（或52.1％）。为了方便我们的分析，我们进一步开发了一个名为AdRisk的静态分析工具来分析提取的广告库并报告可能的风险。特别地，我们目前的分析主要集中在标准Android框架中定义的那些“危险”权限（第2部分），旨在确定广告库可能（误）使用。

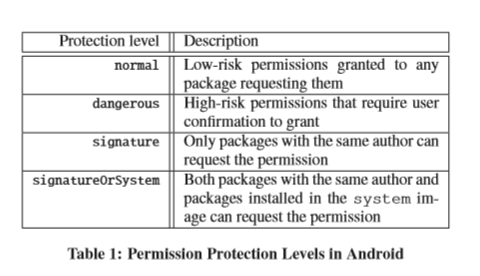


我们的分析揭示了100个代表性广告库中的一些隐私和安全问题。特别是，大多数广告库收集私人信息。虽然他们中的一些可能会将这些信息用于合法目的（即用户的定位广告位置），但我们注意到几个广告库可以隐藏地收集信息，例如用户的通话记录，帐户信息或电话号码。这样的信息可以用来推断用户的真实身份，从而能够更全面地跟踪用户的习惯 - 以牺牲隐私的方式为代价。一个特别受欢迎的广告库（在4190个应用程序中用于外部数据集）甚至允许广告客户直接访问各种个人信息，从而创造不必要的额外的滥用机会。我们还发现，一些广告库将在运行时从远程服务器下载额外的代码，并在正在运行的应用程序的环境中执行，为剥削和滥用打开机会，并且无法确保其完整性。事实上，我们已经确定了一种特定的情况，即提取和加载可疑的有效载荷。完成后，我们将受感染的应用程序（我们的数据集中的7个）报告给Google，并且所有这些应用程序现在已从Android Market移除。这些结果要求需要额外的机制来规范Android上的广告库的行为。本文的其余部分组织如下：第2节概述了Android框架的相关部分。第3节解释了系统设计来评估广告库的威胁，而第4节则包含了实施和评估结果。第5节考察了我们工作的影响和局限性，其次是对第6部分相关工作的调查。最后，我们在第7节总结了我们的论文。

1. 背景

为了了解一个广告库如何嵌入Android应用程序，我们将会考虑一个流行的应用程序，即Rovio的“愤怒的小鸟”。最初是付费的iPhone应用程序，当Rovio将其移植到Android时，“愤怒的小鸟”将移动到广告支持的模式。游戏可以免费下载，但在播放期间和加载新的级别时会定期显示广告;这些广告为Rovio的每月收入创造了100万美元。由于Rovio不在广告业务，该公司转而使用第三方广告网络通过Android上的“愤怒的小鸟”获利。这是智能手机应用开发人员的常见安排和自然选择。在向广告网络注册一些财务信息后，开发人员将收到开发人员标识符和SDK。 SDK的文档包含有关如何使用随附的广告库的说明。广告库旨在嵌入到使用它们的应用中，因此说明包括广告库所需的必要权限;开发人员必须确保广告支持的应用程序通过mak请求这些权限

对其清单文件进行必要的更改。同样，为了支付应用程式投放的广告费用，开发人员必须确保广告资源库具有开发人员标识符。 “愤怒的小鸟”表现文件（如图1所示）提供了这种安排的代表性例子。这个特定版本的“愤怒的小鸟”包含了Google流行的AdMob广告库，它从主机应用的清单中提取了一些控制数据。这些数据包括关键的发布商标识符，它们被存储为“ADMOB\_PUBLISHER\_ID”和“ADMOB\_INTERSTITIAL\_PUBLISHER\_ID”元数据值。此外，在清单文件中，AdMob通过注册com.admob.android.ads.analytics.InstallReceiver组件来监听程序包安装事件，并使用com.admob.android.ads.AdMobActivity定义其自己的Activity（屏幕）以显示全屏显示，屏幕广告一般来说，广告图书馆可以分为三个面向广告的类别：移动网络图书馆，富媒体图书馆和广告调解员。移动网络图书馆是基于网络的广告网络的前端。使用标准的网络技术请求，交付和显示内容，与设备的API交互很少。这些图书馆通常仅显示横幅或文字广告。在我们的研究中，我们发现超过一半的现有应用内广告库是这种类型的。丰富的媒体库具有类似的使命，但表现得更像强大的平台。特别地，它们为应用程序开发人员和广告商提供了丰富的API。虽然他们可以显示更简单的广告类型，但也可以支持更高级的类型，例如活动内容（即JavaScript），视频，插页式广告等。虽然作为富媒体图书馆的广告库比移动网络图书馆少，但许多最受欢迎的媒体库（包括AdMob）实际上是富媒体。第三类广告调解员与前两个不同之处在于展示了一个标准界面，通过该界面，应用程序开发人员可以与其他两种类型的其他广告库进行交互。由于广告库通常以非常不同的方式从应用程序开发人员请求类似的信息，所以这些调解器库可以使应用程序中的多个广告库捆绑在一起更容易。我们的经验表明，所有这三种广告图书馆都倾向于共享一些共同特征。例如，他们具有用户界面代码（呈现他们的广告）和网络代码（从广告网络的服务器请求广告）。它们也被设计为与主机应用程序紧密捆绑。这样，禁止广告功能或诈骗广告网络变得更加困难。同样的结果，一些广告库严重混淆其内部运作，以阻止逆向工程。 AdMob再次提供了一个代表性的例子。在AdMob广告库中，只有AdMob文档中描述的类，方法和字段才有意义的名称;所有其他的东西已经改名为字母表。此外，所有调试信息都从包中的所有类中删除。



在运行时，嵌入式广告库与同一运行时环境中的主机应用程序一起执行，即Dalvik [4]虚拟机（VM），最终在Linux中被实例化为用户级进程。不同的应用程序在不同的Dalvik VM中运行，彼此隔离。 Dalvik VM是从Java衍生出来的，但是已经大大修改（使用自己的机器操作码和语义）来满足手机的资源限制。当Android中安装了一个应用程序时，它被分配了自己的唯一用户标识符（UID），因为Android依赖于Linux进程边界，并且该特定的UID分配策略可以实现隔离或防止行为不端或恶意应用程序中断其他应用程序或访问其他应用程序的文件。不幸的是，这个策略并不会将主机应用程序与它们包含的应用内广告库分开，因为这些库居住在相同的Dalvik VM中，并且使用相同的UID执行。在我们的示例中，AdMob可以随时将用户的“愤怒的小鸟”分数发送给Google。 Android应用程序的结构与大多数平台上的程序的结构不同，这种情况更为复杂，因为它们可以包含多个入口点。响应于进程间通信（IPC）事件，这些入口点由框架调用;即使“运行”应用程序也是这样处理的。在技​​术上，每个应用程序由一个或多个不同的组件组成，每个组件可以独立调用。有四种类型的组件：活动，服务，广播接收器和内容提供商。活动是应用程序的可见用户界面的一部分。一个非常像Unix守护进程的服务在后台运行一段时间，服务请求。广播接收器接收并对广播通知作出反应，而内容提供商使数据可用于其他应用。每个Android应用程序都以压缩包（apk）的形式部署。这些apk文件包含一个清单文件（AndroidManifest.xml），用于描述应用程序的各种标准属性，如其名称，它暴露给系统其余部分的入口点（或接口）以及执行特权所需的权限动作。 “愤怒的小鸟”清单（图1）描述了AdMob定义的两个入口点，而不是“愤怒的小鸟”：一个活动（com.admob.android.ads.AdMobActivity）和一个广播接收器（com.admob.android.ads.analytics.InstallReceiver） 。该活动旨在由应用程序中的代码调用，但广播接收者对Android Market应用程序发送的com.android.vending.INSTALL\_REFERRER事件感兴趣。因此，可以在运行任何主机应用程序的代码之前直接调用广告库的代码。为了更好地保护个人信息和管理系统资源，Android定义了基于权限的安全模型[2]。在这个模型中，具有这些权限的主体是应用程序，而不是用户或库。 Android框架包含一组预先设置的权限，并允许开发者定义附加的权限，如他们所看到的那样。每个权限都有一个保护级别[1]，这决定了许可的“危险”

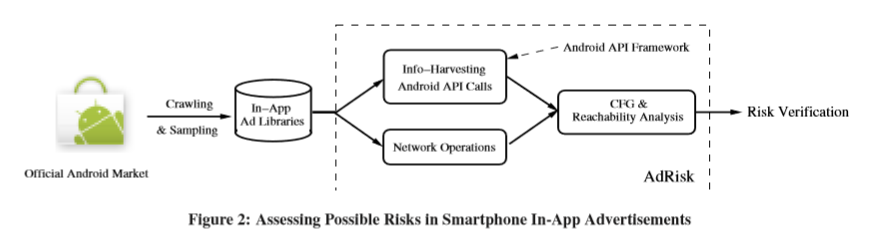
以及其他哪些应用可能会要求它。表1总结了Android中定义的保护等级。签名和signatureOrSystem权限保护级别被保留用于定义功能，这些功能不是由其他作者或作为系统映像一部分的应用程序编写的应用程序使用的。通过注释清单文件中定义的入口点或通过Android框架以编程方式检查权限。由于广告库不是主体，它们继承了嵌入其中的应用程序的权限。因此，许多广告素材库机会地检查并使用权限。有些可能允许主机应用程序的作者有一些控制他们的行为，而大多数广告库只是使用授予主机应用程序的权限。

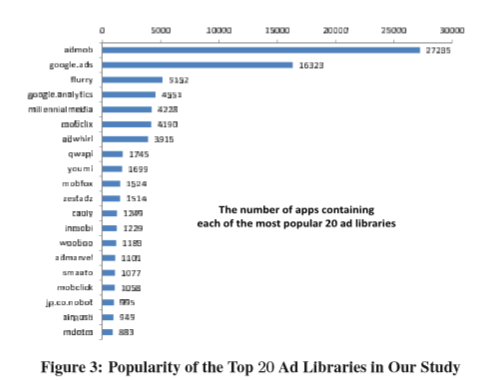
3.

系统设计这项工作的目标是评估嵌入式应用程序内广告图书馆可能存在的隐私和安全风险，并通过衡量其在Android上的流行性来量化这些风险。请注意，Android的基于权限的安全模型提供了一种方便的方式来衡量Android API中固有的风险，因为他们的文档通常提到是否需要权限检查才能成功进行呼叫。然而，如前所述，广告库没有以任何方式由Android框架注释。此外，围绕每个潜在危险的Android API调用的上下文在隐私问题上非常重要。例如，如果用户的电话号码被检索但从未发送到因特网，则不会发生隐私冲突。在这项工作中，我们会选择通过Android Market搜索和收集可用的应用。之后，我们系统地从这些应用程序中识别代表性的广告库，然后开发一个系统来彻底识别可能的风险。图2总结了我们研究中的方法。

3.1

对Android Market进行抽样我们在三个月（2011年3月至5月）中爬行了Android Market forapps，并选择了第一个100,000个下载的应用程序作为我们研究的数据集。与他们一起，我们构建了一个数据库，提取执行我们以后的分析所需的功能，即每个应用程序（在其清单文件中定义）所请求的权限以及应用程序代码中包含的Java类树层次结构。之后，在100,000个应用程序中，我们选择具有android.permission.INTERNET权限的应用程序，这是与广告网络的服务器通信所必需的，并将其组织成候选集。从候选集中，我们随机选择一个应用程序并进行反汇编。对新的广告库进行了拆分的字节码检查。特别是，在新广告库的搜索过程中，我们维护一个广告集，初始化为空。对于我们识别的每个新的广告库，我们将其添加到此集合。此外，我们提取其唯一的类树，并使用它作为模式来检测包含此特定广告库的主机应用程序列表。具体来说，我们从候选集中删除这些主机应用程序。我们重复选择过程，直到选择了100个不同的广告库。通过搜索数据库存储的每个广告库的包名称的类树，我们可以确定我们样本中的100,000个包含给定广告库的应用程序数量。排序和绘制前20个广告库中的图形产生图3中的图表（100个广告库的列表详见表2和表4 - 第4节）。我们研究中的100个广告库总共存在收集的10万个应用的52.1％。在这100个代表性的广告库中，Google自己的AdMob，AdSense和Google Analytics（分析）网络列在前五名。我们还注意到，其他几个网络 - Flurry，MillenialMedia，Mobclix和AdWhirl - 都出现在相对较多的应用中。鉴于这些广告网络在这些领先地位的成熟度





我们期望图书馆本身提供标准功能，也不要从事整个行业的不满。另一方面，这种共同部署的图书馆所造成的潜在隐私风险将会影响到许多用户。在其余的图书馆中，似乎是大量的小型广告网络，在Android上提供了应用内广告库。大量这样的图书馆，加上他们出现的应用程序的比例相对较小，使得他们的行为更容易让广告行业内外的看门狗组织变得更加困难。因此，分析这些图书馆对于广告网络将从事的行为范围的观点很重要。

3.2

分析广告库在识别100个代表性的广告库后，我们接下来尝试确定给定的广告库是否存在安全隐私的任何风险。为此，我们首先考虑标准Android框架定义的权限保护等级[1]。请注意，框架公开的各种标准API需要某些访问权限，这些权限已被保护级别注释。任何需要提高保护等级（即“正常”以上）权限的API都可能被认为是安全隐私的风险。不幸的是，API和权限之间的关系可能很难确定。 Android文档不包含这些关系的详尽列表，并且仅有条件地检查一些权限。例如，Android定义了允许访问用户位置数据的两个相关权限：android.permission.ACCESS\_COARSE\_LOCATION和android.permission.ACCESS\_FINE\_LOCATION.Both权限由android.location.LocationManager类的方法检查;

然而，这些方法确定了哪些权限通过它们被给出的参数进行检查。例如，调用LocationManager.getLastKnownLocation（“gps”）需要android.permission.ACCESS\_FINE\_LOCATION权限;与“wifi”的参数相同的调用将需要android.permission.ACCESS\_COARSE\_LOCATION权限。为了应对这些挑战，我们应用并扩展了Felt等。 [16]导出我们分析感兴趣的API调用列表。特别地，我们采用类似的方法，通过分析Android文档，源代码和反汇编的字节码来保守标注API所需的权限。然而，这项研究是独一无二的，我们的扩展程序还包括一组新的Android API调用，不需要任何许可（第4节）。特别是大部分与ClassLoader和反射机制有关。 ClassLoader部分负责在运行时动态加载代码。要详细说明，在Dalvik中，类引用在运行时解决。通常，由于存在查找未定义引用的验证码，可以将Dalvik应用程序视为仅包含固定的静态代码是安全的。当结合使用API​​时，可以在运行时使用数据来引用类，从而在验证程序运行后调用ClassLoader功能。由于ClassLoader本身只是一个类，因此可以覆盖其方法，以允许开发人员在运行时将原始字节码传递给Dalvik VM。以这种方式，可以下载并运行任意动态代码，使应用程序的任何静态分析都不完整。幸运的是，与基础的Dalvik VM的接口很好。我们将这些接口视为另一种API，不仅将动态代码加载隐含地标记为可疑行为，而且统一了我们的分析框架。总共我们目前的系统考虑到76个不同的权限（34个危险的，26个签名，11个签名系统和5个正常的第4节）。

3.3

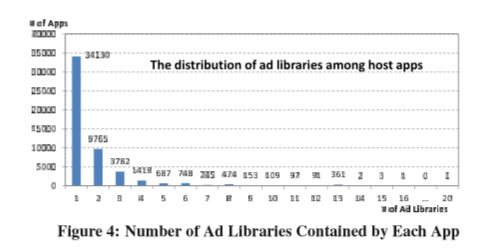
识别可能的风险在识别感兴趣的API集之后，我们对每个广告库执行可达性分析。我们对潜在危险行为的两个维度感兴趣，这意味着我们必须处理最多四个潜在的可达性条件。第一个维度涉及危险行为的沉淀事件;该行为可能来自Android的许多入口点之一，或者可能是响应于接收到的网络数据包。从任何一个起点到API的路径都可能表示危险的情况，但可能不一定;这是第二个维度发挥作用的地方。一些API调用本身是危险的（例如那些可能花钱的API），而另一些只是暴露出可以泄露给外部方的个人数据。在第一种情况下，确定从起始入口点或网络连接的路径是足够的，但在第二种情况下，我们必须进一步查找从危险呼叫到外部接收器（例如，网络API）的数据流路径。

在机械方面，我们的方法如下：首先扫描每个广告库样本的字节码，以查看我们以前注释的危险API调用。对于每个发现的API调用，我们通过库源向后追溯寻找潜在的入口点和任何减轻情况;例如，如果这样一个危险的API调用仅在设置了代表用户同意的标签时发生，我们注意到这种行为。某些API调用在任何情况下可能无法访问，因此可能会被安全忽略，但如果符合这些条件，则会记录所有其他API调用。对于那些泄露信息的电话，我们再通过字节码追踪寻找网络接收器。如果找到，则还会记录从API调用到网络的候选路径1。在算法术语中，我们产生一个控制流图，显示通过库的所有可能的执行路径，然后确定哪些路径确实可行。在我们的原型中，我们利用现有的baksmali Dalvik反汇编程序[3]来自动执行这一过程。作为更大的smali包的一部分，这允许我们访问方便的中间表示和一组有限的程序内静态分析工具。使用它作为基础，我们添加代码来导出控制流图，我们将遍历通过应用（以及因此的广告库）找到一组可行路径。由于Android应用程序和传统Java程序之间的关键区别，遍历派生的控制流程图在Android下带来了更多的挑战。具体来说，像其他许多语言一样的Java程序开始执行主要方法。 Android应用程序没有这样的方法，而是基于它们包含的组件（例如，服务和接收器对象中的某些方法）包含多个入口点。除了这些，库本身通常会向主机应用程序暴露一些方法以进行初始化。框架规定的入口点将根据其所属类自动识别，而库的初始化方法通过注释进入系统。然后我们依次对每个入口点进行分析的后续步骤，最终合并结果。我们的经验表明，由于本机代码和Android框架中的核心类别的影响（例如，使用线程 - Android中的常见技术来提高用户响应），我们观察到生成的调用图中的不连续性。为了解决这些不连续性，我们选择在图书馆旁边加载一组额外的类文件。这些文件用于核心类，并包含旨在捕获每个API调用语义的简单表达式。此外，这些文件包括系统应该识别的危险的API调用;每个危险的电话都包含一个哨兵指令，用于在分析的下一个阶段提醒我们的分析代码。给定这个控制流图，我们的算法接下来尝试从入口点到危险的API调用中找到可达到的路径。为此，我们执行传统的信息流分析，其中对变量进行约束，并通过分支指令进行检查。在所得到的可行控制流图中，我们验证每个危险的API调用是否在可行的代码区中。如果一个调用是，执行向后追溯并记住必要的约束以形成一个执行路径，然后被报告。然后验证我们系统报告的路径。 Android应用程序的可重入，多线程性质使得难以理解的分析，这反过来阻碍了准确地识别图书馆可行路径的努力。某些语言由于数据流分析所固有的复杂性，我们不会忽略不符合附加标准的呼叫。可以使用线程，缓存和其他行为引入数据流不连续性;我们选择参加一些额外的手动努力，以确保我们的学习准确性。

在我们目前的原型中，功能尚未完全支持。例如，Java Reject API（即java.lang.reflect。\* package）允许通过名称调用代码，如果没有完善的数据流分析工具，则会在生成的控制流图中引起不可调和的不连续性。为了适应这种情况，我们尽可能采取保守的做法，保持准确性，但在某些情况下需要额外的手动工作。特别是，我们报告在广告图书馆中使用反垃圾邮件来突出他们的存在进行进一步的调查（第4节）。

4.

原型和评估我们将静态分析工具基于开源的baksmali Dalvik反汇编器（1.2.6版）。在前一部分中实施的设计在原始的baksmali项目中需要2809条新的代码行和四个钩。如设计部分所述，我们的系统还要求每个注释的API进行注释，以便系统进行分析。因此，我们注释了与76个标准Android权限相关联的API。由于我们的静态分析方法相当标准，以下我们主要关注Android平台的特性和我们为风险分析添加的新扩展。特别地，除了报告潜在可行的路径之外，我们的原型已经扩展到报告其他五种感兴趣的代码模式：使用响应，动态代码加载，许可探测，JavaScript链接和阅读安装的软件包列表。由于存在这些模式中的一个或多个可能会使给定库的其他结果着色，因此我们选择让我们的工具自动将其与其可行路径输出一起报告。第一个这样的模式，使用反应，涉及使用java.lang.reflect包。如前所述，Java规范的这一部分允许对方法进行编程调用并访问字段，这使我们的静态分析变得复杂化。没有它，Dalvik字节码的静态分析是可靠和明确的。在理论上，反应本质上将应用程序的调用图解析为数据流动问题。在实践中，经常的反应似乎涉及恒定的字符串，因此不会引入新的含糊之处。不过，我们收集的广告样本并非总是如此。因此，我们的制度在作出进一步审查的情况下，做出了什么假设。类似地，Android应用通常适用于静态分析技术，因为它们被设计为整体加载并被框架本身静态验证。然而，另一个深奥的Java语言功能被转载到Dalvik：ClassLoader类。该类被框架用于根据需要查找代码资源。通常，静态验证阶段实际上会导致整个应用程序一次加载，因为验证程序尝试解析字节码中的所有引用。然而，使用响应，可能导致加载的类不被任何现有代码直接引用。由于ClassLoader类可以由开发人员扩展，因此可以编写此类的自定义版本以从非标准资源加载代码。每个这样的ClassLoader从类的父版本继承，直到类的基准“系统”实例。由于Dalvik与Java不同，不允许开发人员改变“系统”ClassLoader，所以动态代码加载是非常明确的：通用的反馈API不能用于第一次隐式引用一个类，而是使用自定义的ClassLoader必须明确查询。我们的原型引发了这种行为，并引起了严重的警告，因为它的存在否定了所有现有的静态分析工作，并表明了可疑的动态代码加载行为。我们的原型选择处理的更常见的模式是我们所说的“许可探测”。

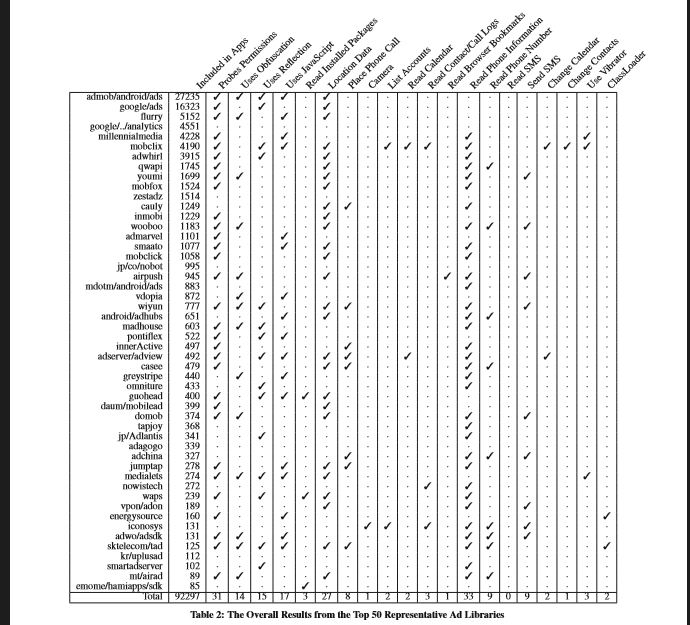


在这种模式下，广告库包含一些需要成功调用权限的API。而不是要求主机应用程序的开发者请求此权限，广告库可以通过事先检查它具有必要的权限，或通过处理大多数API抛出的SecurityException来代替机会主义地尝试使用API被称为不完全的许可。这些检查权限的方法在Android下得到很好的定义，因此我们可以检查控制流图来检测绕过危险的API调用的分支。同样，“富媒体”广告图书馆似乎常常提供JavaScript绑定功能，以向JavaScript广告公开其他功能。我们选择将这种做法包括在我们的发现中有两个原因。这种行为不仅表现为“富媒体”图书馆，而且也引起了有趣的隐私问题，我们将在4.2.2节深入讨论。最后，我们通过显示一个入侵API的一个实例来调整我们的结果，无论什么原因，它都不需要访问的特殊权限。我们研究的一些广告库收集设备上安装的所有应用程序的列表。这些信息与用户的浏览器历史一样是个人的，因为它揭示了他们的兴趣的一些信息。我们包括这种行为来展示广告主的广告主以及基于权限的系统的不完备性。在本节的其余部分，我们从100个代表性广告库的分析中提出我们的发现。我们首先总结了我们在4.1节中的结果，然后在4.2节中介绍了这些广告库中有关危险行为的分类结果。最后，我们在4.3节中评估我们的原型的性能。

4.1

总体结果在列出我们的发现之前，我们强调，我们的结果是针对广告库而不是应用程序。我们的100,000个样本中的某些应用程序将包含多个广告库，而其他应用程序根本不包含。对于我们研究中的100个代表性广告库，我们发现它们嵌入在我们的数据集中的52,067个不同的主机应用程序中。由于一个主机应用可能包含多个广告库，因此我们会显示每个应用包含多少个库的细目。结果如图4所示。从图中可以看出，超过三分之一的应用程序（或更准确地说是34,130）包含一个广告库，一小部分应用程序（大约3％）包含至少五个广告库货币化。一个特定的主机应用程序，即com.Dimension4.USFlag，嵌入不少于20个广告库！然而，目前尚不清楚是否包含更多的广告素材库对应用程序的开发人员有更多的帮助。我们的系统扫描每个代表性的广告库，以使用76个危险的API。前50名广告库的总体结果如表2所示，其余50个的结果如表4（附录）所示。在实践中，许多Android的危险API并没有被任何广告库使用，所以我们选择在我们的结果中省略它们。特别地，两个表包含

我们看到的至少有一个广告库使用的14个危险的API。在表中，我们还包括有关六种结构性质的数据，例如使用模糊处理，通过许可探测进行条件API使用，以及通过ClassLoader语言功能加载动态代码。总的来说，我们的系统报告了318个API使用和结构模式。经过进一步的验证，其中19人得到用户的许可，我们的系统正确地识别了15个这样的案例，这恰好与文本消息（SMS）API调用相关。尽管所有报告的API被标记为“危险”，但我们的结果显示，广告库通常使用一些API。这些包括位置API和单一的“读取电话信息”API调用，这两个API都由我们分析的至少一半的广告库使用。广告库使用这些API来定位信息：位置API可用于提供与用户地理区域相关的广告内容，而常用的电话信息呼叫返回一个唯一的标识符（手机的IMEI号码），可用于跟踪已向特定用户投放了哪些内容。在广告库的上下文中，这些用途似乎是合理的;不过，我们确实发现了两个将这些信息直接发布给广告客户的广告库（Mobclix和广告服务器），这更难证明。其余的危险API可能提供某些功能，或允许访问设备维护的更多侵入性数据。基于特征的API似乎大多是无害的。例如，许多广告库允许广告拨打电话，发短信或向日历添加活动。在所有这些情况下，这些功能仅在用户触发它们（即通过点击广告）并且确定其意图后执行。然而，更为隐蔽的是对广告定位不直接有用的信息请求。我们的分析揭示了一些广告库访问仅在与其他与用户有关的事实相关的信息时才有用的情况。这种相关信息是对用户隐私的直接威胁，因为它可以用于揭示用户的真实身份。例如，很难说用户的通话记录对他们感兴趣的广告有任何影响，但是我们发现一个广告库（sosceo）将一些信息传送到互联网（详见部分4.2.1）。类似地，大量的广告库使用API​​调用来检索用户的电话号码，另一个广告库（Mobus）特别读取用户的SMS消息，以确定他们使用哪个文本消息服务中心。最后，我们确定了一个特别令人担忧的使用其他无辜的API，其中一些广告库（如waps）上传了手机上所有已安装的应用程序的列表。除了隐私问题外，我们还可以确定使用ClassLoader功能的五个广告库，以便在运行时动态加载代码。因此，这些广告图书馆无法进行静态分析;他们的代码可以改变。恶意或妥协的广告网络可以命令其广告库下载僵尸网络有效载荷或根漏洞。我们后来的调查确实捕获了一个可疑的有效载荷，这主要将主机应用程序转变为可远程控制的机器人（第4.2.3节）。此外，广告图书馆的其他结构性质值得一提。我们研究的一半以上的广告图书使用模糊技术，大概是阻止逆向工程。虽然不改变库的功能，但是这些转换可以从方法和类中读取可读取的名称，同时可以通过添加无意义的冗余或重新排序指令来选择性地混合控制流。例如，我们在图5中列出了一个特定广告库（即AirAD）中包含的类。只有少数这样的类有名字带有任何意义;



所有其余的都是“l”（小写L）和“I”字符的字符串。在表2中被注意为使用混淆的广告库都使用一些方案来模糊其内部类，并且通常也以相似的方式混淆字段，方法等的名称。其他混淆器已知存在，但都是相同的目的;例如，字母字符后面的默认混淆器名称类，而另一个使用无意义词典，如“西瓜”和“铁路”。将这些技术应用到合理的大型广告库中隐藏了图书馆大部分内容的意图，而不是真正的保护广告网络的商业秘密 - 因为图书馆仍然可以被明确分析，而且

网络的核心功能驻留在其服务器上，无论从竞争对手的眼睛来看都是安全的。在另一种常见模式中，许多广告库在尝试使用受权限的API之前会探测到他们可用的权限。通常，如果Android应用程序调用API，则无权限访问，则会抛出SecurityException异常。如果此异常未被捕获，该应用程序将崩溃。为了防止这种情况发生，广告库可以先检查他们的权限，也可以默认地捕获抛出的异常。事实证明，超过一半的研究过的广告库（标记在表2和表4中）参与了这种行为。

最后，一些广告库使用Java反射语言功能，其基本上允许以他们的名称对方法和字段进行编程访问。 通常，当加载Dalvik字节码时，会有一个静态验证步骤，确保所有引用的代码元素都有效。 反过来，这种机制允许使用动态代码（在第4.2.3节中讨论的更多），但也可以用于访问任何不能保证在所有设备上正确解析的代码。 通过这种方式，广告库可以访问“实验性”API或供应商的特定API。 鉴于这些API的成熟度较低，广告库的使用是可疑的。

4.2

分类结果为了提供我们在分析中遇到的有问题的行为的更多细节，我们将它们分为三类。

4.2.1

不断收集个人信息第一类涉及可疑的个人信息收集。特别地，一些广告库大胆地要求不直接有用的信息来满足其目的。我们的研究结果表明，较大的广告联播网通常不会参与此类可疑活动，但较小的广告联播网可能会出现这种情况遗憾的是，应用程式使用者无法知道其中包含哪些广告联播网。这个行为的代表性例子可以在sosceo广告库中找到，这是最受欢迎的图书馆之一。像大多数广告库一样，sosceo由其托管应用程序实例化，制作一个用于显示广告的UI元素，在这种情况下是com.sosceo.android.ads.AdView对象。当创建此对象时，会发生相当冗长的混淆方法调用。这些方法调用最终查询设备的联系信息数据库以获取最近的电话。该信息适当地存储在AdView对象使用的数据对象的字段中;当AdView对象从后备广告网络请求广告时，该信息作为URL查询字符串包含在“dp”键下的广告服务器中。其他广告图书馆也采取类似的奇怪行为;例如，Mobus读取SMS（文本消息）数据库，查找有关用户的短消息服务中心（SMSC）的管理信息。该SMSC是后端服务提供商负责向用户和从用户路由文本消息。由于某些未知目的，Mobus将此信息传输到其服务器。类似地，Ponti fl ex对用户在设备上具有的帐户凭据感兴趣。这个信息不是直接的

广告库无法访问凭据本身，但它会查询用户手机管理的帐户列表，因此风险很大。有可疑的是，在这种情况下，危险的API调用是通过反向API来执行的，该API是允许通过数据字符串调用方法的语言功能。在这种情况下，可能会使用反射来抛出库的静态分析。

4.2.2

允许披露数据运行广告第二类涉及个人信息直接暴露在运行广告中。最受欢迎的广告库之一，Mobclix，首先显示为大多数其他广告库的功能。为了显示广告，Mobclix创建了一个android.webkit.WebView，它本质上是一个微型Web浏览器。然后，此网络浏览器呈现广告以进行显示，允许广告客户使用标准网络技术设计广告。然而，与其主要竞争对手不同，Mobclix尝试通过向其广告客户提供某些智能手机功能来获得优势。由于这些功能在HTML或JavaScript中没有标准钩子，因此Mobclix广告库具有将某些Android API绑定到JavaScript函数的类（com.mobclix.android.sdk.MobclixJavascriptInterface），然后暴露给WebView中呈现的广告。每个API调用都包含在简单简洁的JavaScript中。通过这样做，Mobclix公开了各种各样的API调用，并允许将大部分传感器和数据的广告投放到手机上。请注意，大多数这些访问包括适当的用户确认对话框。例如，当广告可以呼叫contactsAddContact（...）来向用户的通讯录添加联系人时，除非用户通过对话框同意，否则不会发生任何事情。

不幸的是，并非所有功能都以这种方式安全地包装。例如，gpsStart（...）函数允许JavaScript广告注册回调函数。该功能将立即被调用，并且每当用户从最后报告的位置移动超过定义的距离时，将再次调用此功能。用户永远不会被要求获得他们的同意，也没有以任何方式通知他们的手机的这个功能被广告使用。这个例子足以引起有趣的隐私问题。期望Mobclix广告库本身可以访问位置信息是合理的;这种信息通常用于将广告定位到某个地理区域。但是，此代码实际上并未将该信息用于Mobclix的广告定位目的。而是将信息提供给第三方广告客户。实际上，由于访问此功能，广告本身可以被认为是未知来源的动态加载代码（第4.2.3节）。

4.2.3

不安全地获取和加载动态代码第三类涉及到不安全的抓取和加载动态代码（可能来自互联网），这有两个原因造成更大的潜在威胁。一个是这个动态加载的代码不能被可靠地分析，有效地绕过现有的静态分析工作。另一个事实是下载的代码可以随时改变，严重损害预测或保护其行为的能力。在100个代表性的广告图书馆中，他们中的五个具有这种不安全的做法。一个特定的一个将下载可疑的有效载荷，这允许主机应用程序被远程控制。特别地，嵌入在主机应用程序中的该广告库的部分非常小：单个服务，com.plankton.device.android.service.AndroidMDKService。该服务通过授予主机应用程序和手机硬件标识符（IMEI）的权限列表与远程服务器联系;作为回报，远程服务器为其提供下载.jar文件的URL

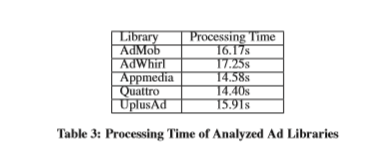


这个.jar文件包含了绝大多数浮游生物的代码，然后使用dalvik.system.DexClassLoader对象（Dalvik的Cla​​ssLoader Java语言特性的基本实现）动态加载。下载的.jar将收听远程命令，并将主机应用程序转换为机器人。基于这一发现，我们已经向Google报告了七个受影响的主机应用程序，该应用程序在同一天立即将其从官方Android Market中删除。这个行为很有趣，因为它突出了Dalvik的动态链接的性质。 Android应用程序以字节码分发，这使得应用程序分析更容易，因为格式的明确定义的语义。此外，在第一次加载类时，Java风格的字节码验证器确保类中的所有引用都被解析。这个验证步骤似乎排除了在运行时添加任意代码。但是，通过java.lang.reflect包，Java（因此Dalvik）可以在运行时通过名称加载类。将这种语言功能与控制Dalvik寻找这些类的定义（即DexClassLoader类）的能力相结合，允许应用程序加载应用程序包文件中不包含的任意代码。在这种情况下，下载的.jar文件具有预定义的入口点com.plankton.device.android.AndroidMDKProvider.init（...）。 DexClassLoader通过名称查找它，然后调用控制逻辑。在新下载的代码中，字节码验证工作正常，因为它现在使用修改的DexClassLoader来解析对不熟悉类的引用。另外四个广告库可以利用此功能，可能是版本控制和内容传递机制。打开达尔维克的完整表现力 - 充分利用应用程序授予的所有权限 - 以笨拙下载的动态代码，不利于隐私。再次，考虑到从互联网检索的代码自然会发生变化，不可能验证广告库只是从事图书馆中体现的行为。

4.3

性能测量

接下来，我们报告我们的原型的性能开销。在我们的测试中，我们选择了五个广告库，并运行我们的系统来分析其中的每一个10次。每个分析运行扫描给定的广告库，以获取我们原型处理的所有（80）API。在每次运行中，我们记录处理时间并报告平均值。我们的测试机是具有2GB内存的AMD Athlon 64 X2 5200+机器和一台Hitachi HDP72502 7200 rpm硬盘。我们总结了表3中的结果。选择测试用例库以提供广告库类型和复杂性的组合。每个图书馆平均需要15.66秒的时间。鉴于我们的工具旨在用于自动化，半自动化的能力，我们认为这种性能对于我们的目的是可以接受的。



5.讨论

我们的研究迄今已经发现了受欢迎的Anroid平台上现有的应用程序内广告图书馆的一些严重的隐私和安全风险.2鉴于此，重要的是检查可能的根本原因并探索未来的防御。首先，由于广告库被并入到使用它们的主机应用程序中，它们本质上形成了共生关系。基于这种关系，广告库可以有效地利用它，并自然地继承用户可以授予主机应用程序的所有权限，从而破坏基于应用的隐私和安全保护。因此，我们认为暴露的风险基本上植根于Android的权限模型中的粒度问题。在这种模式下，可以授予权限的最小实体是一个应用程序。尽管广告库来自不同的开发人员，并且具有与托管应用程序不同的意图，但它们具有相同的权限。正如我们所看到的，广告商本身有时被允许在一个应用程序中执行代码，另外还有一个不受信任的主体集合到单一权限策略所涉及的各方列表中。虽然应用程式访问私人信息的请求可能源自应用程式的代码，广告库的代码或两者，但用户或Android平台无法确定哪些方将使用该信息。第二，目前的情况也可能是一个中心紧张的产物：同样的解决方案，允许广告图书馆被沙盒也可以用来禁用它们，或者欺骗他们。即使Google在Android框架中提供了单独的广告客户模板（即与今天存在的服务，接收者，内容提供者和活动一起），广告网络也不会使用它。使用主机应用程序紧密耦合广告库更安全，以防止它们被轻易绕过。可能出于同样的原因，一些广告网络采取了我们观察到的令人担忧的动态代码加载行为的方法。特别是，由于广告库在框架中不是自己的实体，所以它们只能与其主机应用程序一起更新。广告网络无法控制其广告库捆绑的所有应用的发布时间表。因此，任何代码更新都需要沿侧面渠道推出。动态代码加载显然成为为移动用户提高隐私和安全问题的代价的选择。第三，我们也可以考虑如何设计符合广告客户，广告网络和用户需求的广告库[21,22,32]。与传统的基于网络的广告库一样，这些系统会展示有针对性的广告，并报告网络展示次数，点击次数等，向广告客户收费。但是，他们的目标是无耻地匿名地做这些事情。最终的目标是仅向广告网络提供计费所需的指标，同时允许用户保留完整和直接的所有权

2虽然我们只研究了一个特定的平台，由于将应用内广告整合到智能手机应用程序中的类似性质，我们预计在其他平台上也会存在类似的隐私和安全风险。

个人识别信息。不幸的是，目前提出的每种方法都需要额外的开销（额外的数据传输，设备上的额外存储等），组织转变（第三方广告“经销商”，无线提供商的直接参与等）或两者。由于某些广告库可能无法为其投放的广告做品牌，因此用户通常对某个应用所使用的广告网络无知。因此，这些缺点可能不会以以隐私保护方式运行的广告网络的竞争优势抵消。从另一个角度来看，我们目前的研究仅限于那些简单地“搭载”到主机应用程序中的广告库。特别地，当前的广告库通常是独立的（作为独立的包），以便应用程序开发人员可以轻松地包含它们。然而，有可能有更先进的机制（例如串通[26]，重新委派[18,20]或间接渠道[31]），可以避免使用由AdRisk建模的危险Android API，同时仍然访问各种个人电话信息。请注意，目前正在进行的一些研究项目旨在检测或减轻这些攻击[9,10,11,18]。如何扩展AdRisk无缝集成这些系统仍然是未来工作的有趣任务。

6.相关工作

智能手机的隐私和安全性最近引起了相当大的关注。研究人员使用各种技术来了解或评估这些风险。例如，PiOS [12]使用程序切片来检测iOS应用程序中的隐私泄露。 SCanDroid [19]分析了Android应用程序的源代码以及每个应用程序附带的清单文件，以生成描述应用程序使用信息的数据流策略规范。啄木鸟[20]使用过程间数据流分析来检测Android软体的可能的混乱攻击[23]。然而，它们都不是旨在了解或评估嵌入式广告库的信息泄露和安全风险。相比之下，AdRisk在隐私（例如，信息收集）和安全性（例如，不可信的代码下载和执行）的上下文中侧重于这些广告库的风险。在SCanDroid的情况下，它对应用程序的Java源代码的依赖使其无法分析大多数广告库，通常只能以编译形式进行分发。 TaintDroid [13]采取不同的方式来暴露和识别整个应用程序中的隐私泄露。通过使用修改的Android中间件内置的轻量级动态色调分析，系统会提醒用户泄漏的存在和性质。请注意，它只关心整个应用程序，而不是明确地包含它们包含的广告库。而且，作为一种动态技术，它可以精确地确定可能的泄漏，但是在不能探索所有可能的执行路径的情况下通常是不完整的。最近，Enck等人[14]撰写了Dalvik的反编译器来研究大约一千个流行的Android应用程序，并报告了一些关于它们的发现。在这项工作中，我们研究了十万个应用程序，这使我们能够系统地识别和评估更多种类的广告库。例如，没有一个包含动态代码加载的库（第4.2.3节）被包含在前面或之前报告。我们认为，这样的动态代码加载是危险的，特别是鉴于最近的Felt等人的发现[17]，这与识别Android应用程序中的“过度使用”有关。 （当应用程序请求的权限超过使用权限时，会出现一个特权。）特别是，在正在研究的940个Android应用程序中，超过三分之一被发现是超级用户。鉴于现有广告库中的许可探测行为，更多应用可能会请求不必要的权限，然后由其嵌入式广告库进行机会性的使用。动态代码加载描绘了更加严峻的图景，因为我们发现了一个广告库

在下载代码之前上传其主机应用程序被授予的权限。

在防御方面，已经提出了几个相关的解决方案，其中很多都围绕许可制度。例如，Kirin [15]根据权限分配策略检查正在安装的应用程序的清单，阻止任何请求某些潜在的不安全组合。 Saint [28]通过允许应用程序开发人员在运行时限制安装时的权限分配和权限使用来进一步采取这种方法。其他系统试图向许可系统增添更多的表现力，如Apex [27]，它修改框架，允许在运行时选择性地授予和撤销权限。 MockDroid [7]重写隐私敏感的API调用来模拟它们的失败。 TISSA [33]类似地保护用户信息，而是通过修改Android框架来支持用户定义的信息披露政策，敏感的API可以在这种方案下返回虚假信息，而不是简单的失败。 AppFence [24]进一步修改了这种方法，添加了污点跟踪，允许更细微的策略。然而，这些系统通常会将应用程序视为一段时间，而无需进一步区分嵌入式广告库与托管应用程序。更广泛地说，研究人员已经探索了如何提供有针对性的广告内容，而不会向广告客户或广告网络披露任何私人信息。例如，Adnostic [32]处理在线广告，并允许行为网络广告，而不向广告网络提供行为信息（通过使用专用的Firefox浏览器扩展来防止不必要的信息泄露）。 MobiAd [22]采用类似的方法，通过使用可用于无线供应商的广播模式来流式传输大量标记的广告内容，然后被移动设备过滤。 Privad [21]向客户展示广告选择，但目标是以对广告网络现有行业模式的破坏性较小的方式进行;特别是强调（包括后续工作[30]）保留广告商竞争网络上的广告位的拍卖机制。这些系统包含加密计费，以确保点击通过广告客户正确收费，而不会影响消费者的隐私，也不会鼓励点击欺诈。最后，有一些努力特别旨在解决位置信息所固有的隐私问题。 PrivStats [29]提供了一种机制，可以不间断地以隐私保护的方式收集聚合位置信息。 Bindschaedler等人[8]试图通过在拥挤的地区改变他们的身份来防止跟踪单个设备的移动。虽然这些系统在减轻隐私风险方面取得进展，但是还不清楚它们是否可以在我们的上下文中应用于处理应用程序内广告安全风险（第4.2.3节）。

7.结论

在本文中，我们系统地研究了应用内广告图书馆提出的安全隐私问题。 我们分析了从官方Android Market收集的100,000个应用样本中选出的100个广告库，并发现即使在部分广泛部署的广告库中存在安全隐私的威胁。 这种威胁的范围是收集不必要的侵入性用户信息，以允许在主机应用程序中执行未知来源的第三方代码。 由于Android的权限模型无法区分广告库执行的操作与其托管应用执行的操作，所以目前的Android系统几乎没有提供任何给定应用中存在这些威胁的迹象，这就需要改变现有广告库的方式 被集成到主机应用程序中。