**FlowDroid：精确的上下文，流，字段，对象敏感**

**以及针对Android应用程序的生命周期感知污点分析**

摘要：

今天的智能手机是私密和机密数据的无处不在的来源。 与此同时，智能手机用户受到无意中编程的应用程序的困扰，这些应用程序偶然泄漏重要数据，以及恶意应用程序利用其特定权限故意复制这些数据。 尽管现有的静态污点分析方法具有提前检测此类数据泄漏的潜力，但Android的所有方法都使用一些粗粒度近似值，这可能会产生大量漏检和误报。

因此，在这项工作中，我们展示了F LOW D ROID，这是一款针对Android应用程序的新颖且高度精确的静态污点分析。 Android的生命周期的精确模型允许分析正确的Android框架调用的回传，而上下文，流程，字段和对象敏感性允许分析减少虚假警报的数量。 新型按需算法帮助F LOW D ROID保持高效率和高精度。

我们还提出了D ROID B ENCH，这是一个开放的测试套件，用于评估专门针对Android应用程序的污点分析工具的有效性和准确性。 正如我们通过使用SecuriBench Micro，D ROID B ENCH和一套着名的An-droid测试应用程序进行的一系列实验所展示的，F LOW D ROID发现很大一部分数据泄漏，同时保持误报率较低。 OnD ROID B ENCH，F LOW D ROID可实现93％的召回率和86％的预判，远远超过商业工具IBM AppScanSource和Fortify SCA。 F LOW D ROID成功发现500个来自Google Play的应用的漏洞和来自VirusShare项目的约1,000个恶意软件应用泄漏。

1. 引言：

根据最近的一项研究[9]，Android在手机市场中的市场份额不断增加，目前该市场份额为81％。随着Android手机无处不在，它们成为攻击用户隐私敏感数据的重要目标。毛毡等。对不同类型的Android恶意软件进行了分类[12]，发现恶意Android应用所面临的主要威胁之一是隐私侵犯，这些隐私侵犯会向攻击者泄露位置信息，联系人数据，图片，短信等敏感信息。但即使是非恶意并且经过精心编程的应用程序也可能会出现这种泄漏，例如包含广告图书馆[16]。许多应用程序包括这些图书馆以获得一定的报酬，但他们中很少有人完全理解其隐私影响，他们能够完全控制哪些这些库处理的数据。普通图书馆提取识别针对目标广告的人的私人信息，例如唯一标识符（例如，IMEI，MAC地址等），国家或地点信息。

污点分析通过分析应用程序并将潜在的恶意数据流呈现给人类分析人员来自动检测恶意软件检测工具，然后可以确定漏洞是否构成违反政策的行为，从而解决了这个问题。 这些方法通过从预定义的源（例如API方法返回位置信息）开始，然后跟随数据流直到它到达给定接收器（例如方法编写）来跟踪敏感的“受污染”信息 将信息提供给asocket），从而提供有关哪些数据可能会泄漏的准确信息。 分析可以动态和静态地检查应用程序。 但是，动态程序分析需要许多测试运行到达适当的代码覆盖范围。 此外，当前的恶意软件可以在分析的应用程序执行时识别动态监视器，从而导致应用程序在这些情况下构成良性程序。

虽然静态代码分析不会共享这些问题，但它们会导致不准确的风险，因为它们需要从程序计数器抽象并近似运行时对象。运行时执行的精确建模对于Android应用程序来说尤其具有挑战性，因为这些应用程序不是独立应用程序，而是实际插入到Android框架中。应用程序由具有不同生命周期的不同组件组成。在应用程序执行期间，框架在应用程序内调用不同的回调，通知系统事件，这可以启动/暂停/恢复/关闭应用程序等。[17]。为了能够有效地预测应用程序的控制流，静态分析不仅必须对此生命周期进行建模，而且还必须为系统事件处理（例如，用于GPS等电话传感器），UI交互以及其他应用程序集成更多回调。正如我们在这项工作中所展示的，识别回调不过是微不足道的，需要专用算法。另一个挑战是由用户界面中的密码字段等敏感信息的来源提出。返回其内容的各个API调用不能仅基于程序代码来检测。相反，它们的检测需要存储在清单和layoutXML文件中的辅助信息的模型。最后但并非最不重要的，就像任何使用Java编写的应用程序一样，Android应用程序也包含别名和虚拟调度结构。针对Java的典型静态分析通过一定程度的上下文和对象敏感性来处理这些问题。 Android的框架本质使得这个问题比平常更难，因为我们发现它显示了非常深的别名关系。

过去用于Android的数据流分析方法[14,15,24,40]以不理想的方式使用粗粒度过度以及近似下处理上述挑战。 通常由于缺乏忠实的生命周期模型而导致的低估现象可能导致这些分析错过重要的数据流。 然而，实际情况更糟糕的是，这些工具的过度逼近可能会导致很多虚假的警告，从而使安全分析人员无法完全停止使用分析工具。

因此，在这项工作中，我们展示了F LOW D ROID，这是一款专门针对Android平台而设计的新型静态分析系统，并基于新颖的按需算法，可在保持可接受性能的同时产生高精度。 F LOW D ROID通过分析应用程序的字节码和配置文件来发现潜在的隐私漏洞，无论是由于疏忽造成的，还是由恶意注入造成的。与之前的分析相反，F LOW D ROID是第一款静态污点分析系统，它在完全模拟完整的Android Lifecycle的同时，对应用程序中的回调和用户定义的UI小部件进行了正确处理，具有完全的上下文，流程，字段和对象敏感性。这种设计最大限度地提高了精度和召回率，即旨在最大限度地减少泄漏和错误警告的数量。为了在保持可接受的性能的同时获得深层次的上下文和对象敏感性，F LOW D ROID使用按需别名分析。分析算法受到了Andromeda [37]的启发，但在精确度方面优于Andromeda。我们在2013年夏季开放源代码F LOW D ROID。该工具已被多个研究小组挑选出来，并且正与一家领先的反病毒工具生产商联系，在分析后端高效地使用F LOW D ROID 。

对于我们和其他人来说，要衡量这个重要研究领域的科学进展，需要研究人员对Android污点分析工具进行比较研究。遗憾的是，到目前为止，还没有可以进行系统研究的基准。 作为这项工作的另一个贡献，我们提供了D ROID B ENCH，这是一种新颖的开源微基准套件，用于比较Android的污点分析的有效性。 我们已经在2013年春季在线提供了D ROID B ENCH，并且已经知道一些研究小组已经使用它来衡量和提高他们的Android分析工具的有效性[19]。 第一批外部研究人员已经同意为该套件提供进一步的微基准。

F LOW D ROID可用于保护内部开发的Android应用程序，并协助对Android恶意软件进行分类。 这两种用例都不需要完美但却相当低的假阳性和假阴性率。 使用SecuriBench Micro，D ROID B ENCH和一些知名应用程序（包含数据泄漏）进行的一组实验表明，F LOW D ROID发现数据泄漏的分数很高，同时保持误报率较低。在D ROID B ENCH 1.0，F LOW D ROID召回率达到93％，精度达到86％，远远超过商业工具AppScanSource [2]和Fortify SCA [3]。 进一步的实际应用实验证实了F LOW D ROID在实践中的实用性。总之，这项工作提出了以下原创性贡献：

•F LOW D ROID，这是第一个考虑Android应用程序生命周期和UI小部件的完全上下文，字段，对象和流敏感污点分析，以及哪些功能是按需别名分析的新颖，特别精确的变体;

•F LOW D ROID的完整开源实现，

•D ROID B ENCH，一款用于Android流程分析的新颖，开放和全面的微型基准测试套件，

•与商业工具AppScan Source和Fortify SCA相比，一组实验证实了F LOW D ROID的出众精确度和召回率，以及

•一组将F LOW D ROID应用于来自Google Play的500多款应用程序以及来自VirusShare项目的大约1000种恶意软件应用程序的实验[1]。

我们在线提供我们作为开源项目的全面实施，以及所有基准和脚本，以重现我们的实验结果：

空间限制使我们无法包含完整再现我们方法所必需的一些细节。 因此，我们发布了一份技术报告，[13]将F LOW D ROID的转移功能正式化，并给出了实施的更多细节。

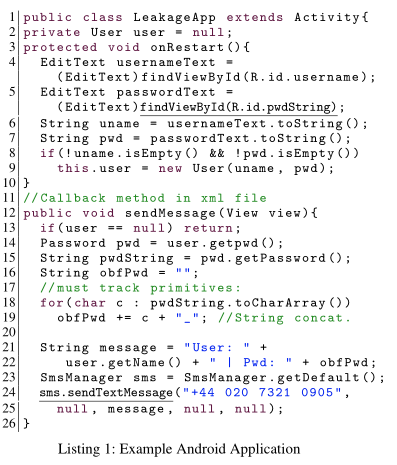
该文件继续如下。 第2部分给出了一个motivatin实例，并解释了Android安全性的必要背景。第3部分解释了F LOW D ROID如何模拟Android生命周期，而第4部分则给出了有关实际异常分析的重要细节。 在第5节中，论文讨论了实施细节和限制，而第6节评估了F LOW D ROID。 第7节讨论相关工作，第8节结束。

1. 背景和实例

我们首先给出一个激励的例子，然后解释这个工作假设的攻击者模型。清单1中的例子（从现实世界的恶意软件应用程序[42]中抽象出来）实现了一个活动，inAndroid代表用户界面中的一个屏幕。应用程序只要框架重新启动应用程序就从文本字段（第5行）读取密码。当用户点击活动的按钮时，通过SMS发送密码（第24行）。这构成了从密码字段（源）到SMS API（接收器）的污染数据流。在此示例中，sendMessage（）与应用程序UI中的按钮相关联，该按钮在用户单击该按钮时触发。在Android中，监听器直接在代码中或在布局XML文件中定义，如此处所述。因此，单独分析源代码是不够的 - 分析还必须处理元数据文件以正确关联所有回调方法。在此代码中，如果在执行sendMessage（）之前调用onRestart（）（初始化用户变量），则会发生泄漏。

为了避免错误的否定，一个污点分析必须正确地模拟应用程序的生命周期，认识到在应用程序重新启动后，用户可能确实击中了按钮。为避免错误肯定，本示例的分析必须是字段敏感的：用户对象包含两个字段 用户名和密码，但只有后者应被视为私密值。 虽然这个例子不是必需的，但对象敏感性对于区分源自不同分配地址但到达相同代码位置的对象是非常重要的。 在我们的实验中，我们发现一些需要深度物体灵敏度的情况才能够自动消除误报。 这是由于Android框架的相对较长的调用和分配链。

像字符串连接这样的操作（第19行）需要一个模型来定义是否以及如何通过这些操作传输数据。像普通方法调用和分析应用程序代码库这样的操作可能是不准确的（因为它忽略了操作的语义） 我们发现，在实践中往往是非常昂贵的。



攻击者模型F LOW D ROID可用于检测一般数据流，无论它们是否由粗心或恶意的意图引起。 对于恶意案件，我们假设以下攻击者模型。 攻击者可能会提供任意恶意的Dalvik字节码的应用程序。 通常，攻击者的目标是通过用户授予的危险广泛的权限泄露私人数据[4]。 F LOW D ROID对安装环境和应用程序输入进行合理的假设，这意味着攻击者可以自由地篡改这些内容。 然而，F LOW D ROID确实认为攻击者无法规避Android平台的安全措施或利用旁路。 此外，我们假设攻击者不使用隐式流[20]来掩饰数据泄漏。 鉴于当前的可用软件类型，这是一个非常合理的假设。

1. 精确模型生命周期：

在下文中，我们将解释F LOW D ROID对生命周期的精确建模，包括入口点以及异步执行组件和回调。

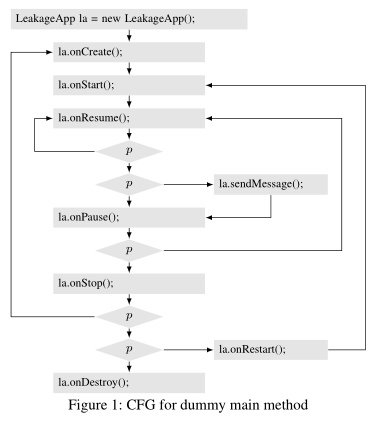
多个入口点与Java程序不同，Android应用程序没有主要方法。相反，应用程序包含许多入口点，即由Android框架隐式调用的方法。 Android操作系统为应用程序中的所有组件定义完整的生命周期。应用程序开发人员可以定义四种不同类型的组件：活动是单一用户操作，服务执行后台任务，内容提供者定义类似数据库的存储，以及广播接收者监听全局事件。所有这些组件都是通过从预定义的操作系统类派生自定义类来实现的，将其注册到AndroidManifest.xml文件中并覆盖生命周期方法。 Android框架根据环境需要调用这些方法来启动或停止组件，或者暂停或恢复组件。例如，它可以由于内存耗尽而停止应用程序，并且稍后在用户返回时重新启动它[17]。因此，在构建调用图时，Android分析不能简单地通过检查预定义的“主要”方法开始。相反，Android生命周期中所有可能的转换必须精确地进行建模。为了解决这个问题，F LOW D ROID构造了一个模拟生命周期的自定义虚拟主方法。在下面我们解释如何构造这个方法。

异步执行组件应用程序可以包含多个组件，例如三个活动和一个服务。虽然这些活动按顺序运行，但不能预先确定其顺序。例如，一项活动可能是用户最初看到的主要活动，然后根据用户的输入启动其中一项活动。服务作为平行的后台任务运行。 F LOW D ROID通过假定应用程序内的所有组件（活动，服务等）可以以任意顺序运行（包括重复）来执行此执行。一些静态分析是路径敏感的，即分别考虑每个可能的程序路径。在这种情况下，考虑所有可能的排序会非常昂贵.F LOW D ROID将其分析基于IFDS [32]，该分析框架不是路径敏感的，而是在任何控制流合并点立即加入分析结果。因此F LOW D ROID可以生成并高效地分析虚拟主要方法，无论组件生命周期和回调的任何顺序是否可行;它不需要遍历所有可能的路径。

回调Android操作系统允许应用程序注册各种类型信息的回调，例如位置更新或UI交互。 F LOW D ROID将这些回调模拟为虚拟主要方法，例如，以识别应用程序将位置数据存储到框架传递给回调的位置数据作为参数的情况，并在稍后将活动停止时将此数据发送到Internet。通常可以预测回调被调用的顺序，这就是为什么F LOW D ROID假设所有回调都可以以任何可能的顺序调用。但是，回调只能在父组件（例如活动）正在运行时发生。为了精确，F LOW D ROID因此将组件（活动，服务等）与其注册的回调联系起来。例如，一个activity可以注册一个按钮被调用时调用的回调函数。然后，相应的回调处理程序必须仅在thisactivity的onResume（）和onPause（）事件之间进行beanalyzed。

有两种不同的方式在Android平台上注册回调处理程序。首先，可以在活动的XML文件中声明性地定义回调。或者，他们也可以通过使用众所周知的对特定系统方法的调用进行命令注册.F LOW D ROID支持两种方式。此外，对于恶意软件，攻击者可能会通过Android基础结构的重写方法注册未记录的回调，其中一些甚至可以由本机代码调用。 F LOW D ROID可以识别这些重写的方法，并像普通的回调处理程序一样处理它们，例如按钮点击。

为了查找在应用程序代码中注册的回调，F LOW D ROID首先计算每个组件的一个调用图，从各个组件类中实现的生命周期方法（onCreate（），onStop（）等）开始。这个调用图然后用于扫描forcalls到Android系统方法，使用一个众所周知的回调接口作为形式参数类型。之后，调用图被逐步扩展以包含这些新发现的回调，并且扫描再次运行，因为回调处理程序自由注册新回调，可能需要F LOW D ROID重新扩展调用图并重新分析，直到固定点达到了。虽然这种方法比扫描实现回调接口的类更昂贵，但它在组件和回调之间提供了更精确的映射。这不仅可以减少误报，还可以显着减少以下污点分析的运行时间。一旦构建dummymain方法，F LOW D ROID就会使用此方法计算最终调用图作为应用程序的入口点。

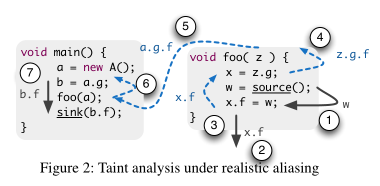


对于在布局XML文件中定义的回调，使用相应的布局控件将相应的XML文件映射到一个或多个应用程序组件。例如，按钮单击处理程序仅对承载相应按钮的活动有效。 F LOW D ROID分析每个活动以查看它注册的XML文件中的哪些标识符。这个信息然后用于创建映射。

示例请注意，为了获得最大精度，F LOW D ROID会为每个分析的应用程序生成一个新的虚拟主方法。每个主方法只会涉及应用程序的XML配置文件根据应用程序的XML配置文件实际发生atruntime的部分生命周期。禁用的活动会自动过滤，回调方法仅在其实际所属组件的上下文中调用。例如，按钮单击处理程序仅在其各自活动的上下文中分析。在图1中，我们展示了我们前面例子中虚拟主方法的控制流图。该图模拟了用sendMessage回调增强的通用活动生命周期。在这个图中，p表示其中的anopaque谓词。

4精确的流敏感分析：

分析的一个主要困难是如何实现高对象灵敏度来有效地解决混叠。图2（摘自现实世界的案例）显示了F LOW D ROID如何将一个前瞻性分析和一个按需后向别名分析结合起来，从而推断b.f被污染了。在步骤1中，污染变量w向前传播，污染堆对象x.f。 Step2继续对w和x.f进行污点跟踪。重要的一步是3：Whenevera堆对象受到污染，向后分析向上搜索各个对象的别名（本例中为x.f）。 At7，thealias b.f被发现，然后作为正常的污点向前传播。



F LOW D ROID模拟了跨程序分布式子集问题的IFDS [32]框架内的污点分析问题。第4.1节解释了分析使用的传递函数。大部分功能都比较标准。然而，有一个重要的分析，其中F LOW D ROID的分析与标准的污染分析算法不同，即在赋予堆的语句，即字段或数组的语句。这种情况会导致后向别名分析被调用，我们将在4.2节中解释这些细节。由于空间限制，我们将流程功能的描述保留在非正式的层面上。为了让其他人重现我们的方法，随附的技术报告包含完整的正式化[13]。

4.1污点分析：

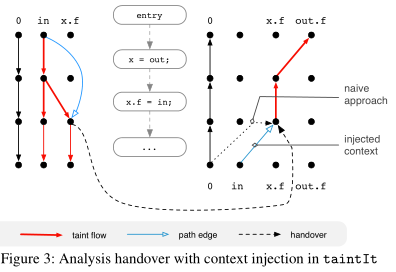
向前和向后分析都传播访问路径。访问路径的格式为x.f.g，其中x是局部变量或参数，f和g是字段。访问路径可以具有不同的长度，直至用户可定制的最大长度（默认为5）。长度为0的Anaccess路径是一个简单的局部变量或参数，例如x。在F LOW D ROID中，访问路径隐含地描述了通过该路径可达到的所有对象集合，例如，x.f包括x.f.g，x.f.h，x.f.g.h等。

如果右侧的任何操作数被污染，则分配的传递函数会污染左侧。分配数组元素通过污染整个阵列来进行保守处理。将一个“新”表达式分配给一个变量x会删除以x为根的访问路径建模的alltaints。方法通过替换实际参数和形式参数来调用对被调用者上下文的访问路径;反向翻译发生在方法返回时，包括返回值（如果存在）。像往常一样使用基于IFDS的分析，F LOW D ROID还包含一个呼叫回拨流程功能（绕过呼叫方侧的每个方法呼叫）。这个函数传播与呼叫无关的污点，生成新的污染源，汇报污点并为本地调用传播污点。第5节给出了关于后者的更多信息。

4.2按需别名分析

无论何时将污染值分配给堆位置（如字段或数组），F LOW D ROID都会向后搜索目标变量的别名，然后再将其污染。在清单2中，fornow考虑第一次调用taintIt（第3行），这会污染其中的正式参数。在第10行中，由于赋值x.f = in，这会导致访问路径x.fto受到污染。在这种情况下（通常在堆的所有分配中），F LOW D ROID将开始向后搜索xf的别名，在第9行中找出out.f。在此处，从此语句开始out.f的新向前污点传播，后者最终会发现第11行的泄漏。然而，后向分析也将继续向后搜索，尽管发现了主要的别名pf，然后它产生了一个前向分析，导致第4行的第二个污点流量报告。

保持上下文敏感性算法1和2显示了正向和反向分析求解器伪代码的主循环。该算法表示假设该算法熟悉原始IFDS算法的算法描述[32]。两个求解器都在自己的工作列表上运行，其中包含所谓的路径边缘，用于汇总到目前为止的语句/节点n所计算的数据流。一个edgehs p，d 1 i→hn，d 2 i有效地表明，分析得出结论：如果d 1在n的过程p的起点sp处保持，则d 2保持在n。在我们的具体实现中，抽象域值di非常有效地访问描述对受污染值的引用的路径。两次分析之间的移交是相当不重要的。如果以一种天真的方式进行协调，人们很容易就可以获得两个独立的分析结果，每个独立分析可能都是上下文敏感的，但是会组合产生分析信息，用于不可实现的路径，即存在相互冲突的上下文。例如，请注意，在清单2中的例子中，x.f的别名只有在以前受到怀疑的情况下才会受到污染。特别是，分析不应该报告泄漏线6，其相应的污点只能传播字符串“public”。



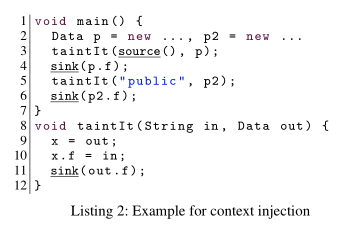
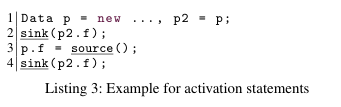


图3显示了一个天真的实现如何导致假阳性，以及F LOW D ROID如何通过将上下文从一个分析注入到另一个分析来处理问题。该图假定对IFDS框架内流量函数的典型符号[32]有一定的熟悉度。这里的黑色节点代表相应语句之前/之后的数据流事实，黑色和红色边代表数据流。事实0是重复的事实，它总是真实的，这就是为什么一个0节点总是连接到下一个节点。图的左侧显示前向污点分析如何确定x.f被污染。在处理转换为x.f时，正向分析会生成一个反向分析实例，如右侧所示。这种分析的天真方法是用0到x.f（虚线）的边来初始化它。这个实现虽然直截了当，但会导致不精确，因为它的语义状态表明x.f的别名无论如何都被保留。在清单2中，即使对于p2.f，这也可能导致analysisto错误地报告污点违规。因此，正确的方法是在后向分析中注入前向分析的内容：F LOW D ROID向x.f查询“路径边缘”，IFDS算法存储该路径作为其总结计算的副作用。然后它将整个边缘注入后向求解器（参见算法1，第16行）。上下文注入发生在两个方面。在例子中的第9行，当后向分析产生了对out.f的前向分析时，它将原始内容注入到前向分析中。 （见算法2，第17行）在语义上，对于这个例子来说，这意味着两个分析都发现了所有taintItare条件w.r.t.最初受到污染。

第二个问题是避免由于不可实现的路径导致的误报：F LOW D ROID需要防止反向分析返回到前向分析未分析的上下文中（反之亦然）。为了实现这个约束，F LOW D ROID中的落后分析实际上根本不会返回到调用者。相反，无论何时找到别名，它都会触发对该别名的前向分析，例如第9行中的out.f。然后，正向分析的任务是将任何相关的污点映射回调用者的上下文。在该示例中，正向分析知道它所调用的调用上下文，这就是为什么它可以很容易地确保只将影子映射回正确的上下文。在这个例子中，forwardpass只会在第3行将out.f映射到p.f，而不会映射到第5行的p2.f。Inessence，反向分析可以下到被调用者中，但不会返回给调用者;所有的回报都是由forwardanalysis处理的。当向后分析进入呼叫时，它会在到达方法头时最终产生前向分析。 （见算法2，第13行）因此，前向分析可以确保只返回到正确的调用者，因为它的上下文是由后向分析注入的。 （技术上，它的incomingset [26]被注入。）每当前向分析将调用者映射到与堆对象关联的污点时，它会在调用者内部产生一个新的aliassearch。



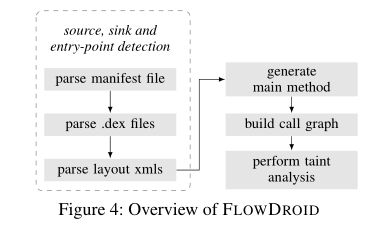
保持流量灵敏度Andromeda [37]是另一种技巧分析工具，它启发了F LOW D ROID的按需别名分析.Andromeda的分析可能会导致流程不敏感的结果。在清单3中的例子中，分析会报告两个泄漏在第2和第4条，即使第一次调用沉没肯定发生在p2.f变得污染之前。事实上，同样的问题也会出现在F LOW D ROID的分析中，正如我们上面所描述的那样：后面的分析会发现第1行的被污染的别名p2.f并且触发一个具有该值的正向传递，从而导致今后p2.f泄漏的任何地方。

F LOW D ROID通过跟踪几个呼叫激活语句来解决这个问题。每当产生后向别名分析的实例时，相应的访问路径将增加当前语句，即别名的激活语句。另外，受污染的别名被标记为不活动。在语义上，只有活动到达接收器时会导致泄漏。未激活的污点是尚未污染的内存位置的别名。只要后向分析再次产生前向分析，并且当前向分析在其激活声明上传播混淆污染时，污染就会激活，从而获得其实际导致泄漏报告的能力。在该示例中，激活声明在第3行，因此分析仅报告在随后的第4行的阿雷克，避免第2行的误报。

通常，激活语句是调用树的代表。假设清单3中的堆分配包含在方法调用中，例如清单2中第10行的分配情况，该方法调用发生在方法调用taintIt中。在这个例子中，当前向分析处理返回到main的第3行时，全局分析将调用与激活语句联系起来（第3行），因为每当这个调用完成时，激活语句也被处理，因此污点将会积极点。换句话说，激活语句用于查找发生它们的呼叫树将其转换回（传递）呼叫者。

据我们所知，F LOW D ROID是第一种实现按需分析的方法，可以充分保持对称流动的灵敏度。在未来，我们计划进行调查以清楚地说明这里解释的原理可以在污点分析的范围之外重复使用，理想情况下产生IFDS的相当普遍的扩展。

进一步的算法细节我们对IFDS的实现使用了由Naeem和Lhot'ak [26]解释的扩展。有了这个扩展，IFDS实现就可以即时计算程序的超图，这意味着在我们的情况下，我们只为那些确实已经保留的变量/访问路径计算taintinformation。我们的实现使用了IFDS解算器的两个实例，每个实例都经过微调，如算法1和2所解释。每个实例都包含一个独立的汇总函数表，如同原始IFDS算法一样，用于避免对相同被调用者进行重新计算在相同的背景下。



5实验

F LOW D ROID扩展了Soot框架[21]，它为精确分析提供了重要的先决条件，特别是三地址代码中间表示Jimple和准确呼叫分析框架Spark [22]。一个名为Dexpler [5]的插件允许F LOW D ROID将Android的Dalvik字节码转换为Jimple。在Soot和Dexpler的基础上，F LOW D ROID进一步使用Heros [7]，这是一种可扩展的，高度多线程的IFDS框架实现[32]。接下来我们将解释F LOW D ROID的体系结构，而后续章节将解释重要的实施细节和F LOW D ROID目前的局限性。

架构图4显示了F LOW D ROID的架构。 Android应用程序打包在apk文件（Android包）中，这些文件实质上是zip压缩的归档文件。解压缩无法解析后，F LOW D ROID将搜索应用程序的生命周期和回调方法以及调用源和接收器。这是通过解析各种Android特定的文件，包括布局XML文件，包含可执行代码的dex文件以及定义应用程序中的活动，服务，广播接收器和内容提供者的清单文件来完成的。接下来，F LOW D ROID从生命周期和回调方法列表中生成虚拟主要方法。这个主要方法然后被用来生成一个调用graphand程序间控制流图（ICFG）。从检测到的来源开始，污点分析通过遍历ICFG来跟踪污点，如第4节所述。F LOW D ROID配置了我们的SuSi项目[30]推断的源和汇，这是目前最全面的一个。 F LOW D ROID网站提供了信息来源和汇的具体清单。最后，F LOW D ROID报告从源到汇的所有发现流量。其中包括完整路径信息。为了获得这些信息，实现将数据流抽象对象链接到它们的前辈和它们的生成语句。这允许F LOW D ROID的报告组件完全重构所有相关分配语句的图表，这些分配语句可能会导致在该宿舍内发生污点违规。

定义快捷方式在分析中包括完整的JRE或Android平台运行时间不仅需要大量的分析时间和内存，而且由于在图书馆分析过程中进行近似处理，还会导致不必要的不​​精确。 F LOW D ROID因此包含用于外部库模型的接口。这些工具支持简单的文本文件格式来定义某些“shortcutrules”。预定义的规则处理集合类，字符串缓冲区和类似的常用数据结构，例如指定将集合元素添加到集合会污染整个集合。从技术上讲，这些快捷方式是使用call-to-return边缘实现的。如果库调用没有关联的规则，则会对其进行全面分析。

本机调用Java和Android平台都支持使用C或其他非托管语言编写的本地方法的调用。对于基于Java的分析，此类方法是无法分析的黑匣子。 F LOW D ROID为最常见的本地方法提供了explicittaint传播规则，例如System.arraycopy。在此示例中，如果在调用之前第一个参数（输入数组）被污染，则此规则将第三个参数（输出数组）定义为污染。对于没有明确规则的本地方法，F LOW D ROID假设一个合理的默认值：调用参数和返回值在至少一个参数被污染之前变为污染。这既不完全合理，也不是最精确的，但可能是黑箱设置中最好的实际近似值。

组件间通信F LOW D ROID通过将发送意图作为接收者的接收方法和接收意图资源的回调过度近似于显式的组件间通信。 Android还支持隐式的基于意图的通信，例如，通过设置被调用活动的结果值，然后由操作系统自动返回到调用活动。将这些行为与更精确的组件间连接模型一起支持未来的工作。特别是，我们正在将F LOW D ROID与EPICC [27]结合起来，这是一种使用Soot和Heros进行串行分析以更精确地解决应用程序间通信的静态分析。

局限性尽管F LOW D ROID通常旨在进行分析，但它与其他静态分析工具确实存在一些固有的局限性。例如，F LOW D ROID只有在它们的参数是字符串常量时才会解析反射调用，这并非总是如此。在Java平台上，TamiFlex [8]等反射分析工具可用于使静态分析工具了解运行时发出的反射调用。不过，这样的工具需要通过java.lang.instrument加载时间管理，而Android平台目前不支持。如果Android生命周期包含我们并不知晓的回调，或者通过我们的规则模型模型不正确的本地方法，那么不稳定性也可能出现。目前F LOW D ROID也没有注意到多线程：它使用随机顺序的顺序排序，这通常也是不合理的。完全结合多线程的良好支持对我们来说是未来工作的一大挑战。

7相关工作

有几种方法可用于静态分析Android应用程序，其精度，运行时间，范围和焦点不同。

其中最复杂的是CHEX [24]，这是一种通过在外部可访问的接口和敏感资源或接收器之间追踪垃圾来检测Android应用程序中的组件劫持漏洞的工具。虽然不是为这项任务而建造的，但CHEX原则上可用于污点分析。 CHEX不会将调用分析到Android框架本身，而是需要一个（希望完成的）框架模型。在F LOW D ROID中，这样的模型是可选的，除了本地调用外，仅用于提高精度和性能。因此，用户可以完全省略模型，但仍然不会失去污点。 CHEX的入口点模型要求枚举所有可能的“分割次序”，这在F LOW D ROID中不是必需的。此外，CHEX仅限于大多数1对象灵敏度，而F LOW D ROID的需求驱动的分析允许任意长度的上下文（使用默认值5）。我们发现1对象敏感性在实践中太不精确。

LeakMiner [40]与我们的技术观点类似：像F LOW D ROID，它基于Soot，使用Spark forcall-graph生成，实现Android生命周期，并且该报告指出可以分析应用程序平均2.5分钟。然而，分析不是上下文敏感的，这排除了D ROID B ENCH中大多数测试案例的精确分析。

AndroidLeaks [15]也声明了处理AndroidLifecycle的能力，包括回调方法。它基于WALA的对文本敏感的系统依赖图，具有用于堆跟踪的contextinsensitive覆盖，但不像F LOW D ROID那样精确，因为如果受污染的数据存储在其中一个字段中，它就会污染整个对象，即，既不是字段或对象敏感。这排除了许多实际情况的精确分析.ScanDroid [14]是另一种推理Android应用程序中的数据流的工具。它的主要焦点是组件间（例如同一应用程序中的两个活动之间）和应用程序间数据流。这提出了将意图发送者连接到其他应用中的各个接收者的挑战。

SCanDroid修剪allcall边缘到Android操作系统方法，并保守地承担base对象，参数和返回值以继承参数。这比F LOW D ROID的处理要精确得多; F LOW D ROID仅适用于未明确建模的本地调用的默认规则。 F LOW D ROID目前将intentsending模式作为信号源和意图接收模式来源，产生对应用程序间通信的良好处理。未来，我们计划将F LOW D ROID与EPICC [27]相结合，这是一种新颖的静态分析，它使用字符串分析来精确地解决应用程序间通信。

像CopperDroid [31]等其他方法动态观察Android组件和底层Linux系统之间的交互，以重构更高层次的行为。特殊的刺激技术被用于运用应用程序来发现恶意活动。然而，攻击者可以很容易地修改应用程序来检测它是否在虚拟机内运行，然后在此期间不泄露数据[29]。或者，数据泄漏只会在某个运行时间阈值后发生。 Aurasium [38]和DroidScope [39]在很大程度上与静态泄漏检测相同的缺点。

TaintDroid [11]是迄今为止最复杂的Android tainttracking系统之一。然而，作为一种动态的方法，与F LOW D ROID相比，它产生了一些完全不同的折衷。例如，TaintDroid通过反射方法调用跟踪污点没有问题，因为TaintDroid是作为执行环境的扩展来实现的，对于这些扩展来说，方法是否通过反射来调用并不重要。另一方面，如果用于在安装时间之前对恶意软件进行分类，那么TaintDroid只有与产生不错的代码覆盖率的动态测试方法配对时才能成功检测到恶意软件。像F LOW D ROID这样的静态提前分析不会分享这个缺点，因为它们覆盖了所有的执行路径。其次，像TaintDroid这样的动态方法可能会被恶意应用程序所迷惑，这些恶意应用程序认识到它正在被分析，在这种情况下，应用程序可能会简单地避免执行任何恶意活动[29]。如果动态分析安装在最终用户的手机上（这种情况下，恶意软件将被有效地驯服），这并不是问题，但如果动态分析仅用于恶意软件的提前发现，那么稍后可能会发生这种问题安装在不受动态分析保护的系统上（在这种情况下，应用程序可以恢复其恶意活动）。诸如F LOW D ROID之类的静态方法不会分享这个特殊的缺点，因为它们实际上执行应用程序。

F4F [36]是一个框架，用于使用名为WAFL的规范语言来执行污点分析基于框架的应用程序，以描述相应框架的功能行为。虽然最初是为Web应用程序创建的，但它也可以扩展为通过为Android添加WAFLgenerator对Android框架进行建模。 F LOW D ROID的虚拟主代具有很大的优势，只包含应用程序确实可以访问的组件和回调。然而，这需要应用程序清单的语义模型，布局XML文件，编译后的资源文件和应用程序的源代码，它们都是交错式的.F4F最多可以用来粗略近似建模所有可能应用程序的常见分母。

F LOW D ROID目前通过过度逼近处理异常流量。最近Kastrinis和Smaragdakis提出了一种新颖且特别有效的方法来分析感染和分析结合[18]。看到F LOW D ROID是否可以从这些概念的一体化中轻松获益，这将是有趣的。

Rountev等人。已经提出了一种预先计算大型库的摘要的方法，目的是加速对客户端代码的重复分析[33]。对于Android应用程序来说，这个方法非常有意义。 Rountev的工作基于IDE框架[34]，F LOW D ROIDalso内部使用它来执行基于IFDS的分析。因此，应该完全有可能将作者的想法纳入“低落差”。

Dillig等人开发了一种更精确地分析集合和数组内容的方法[10]。所需的分析手段并非易事，但考虑到我们的结果，很明显，F LOW D ROID可以通过实施分析支持进一步提高其精度。

8总结

我们提供了F LOW D ROID，这是一款适用于Android应用的新颖且高度精确的静态分析工具。与以前的方法不同，F LOW D ROID充分模拟了Android特定的挑战，例如应用程序生命周期或回调方法，这有助于减少错漏或误报。新型按需算法允许F LOW D ROID尽管具有强大的上下文和对象敏感性，仍然可以保持效率。为了评估分析工具的有效性，我们提出了特定于Android的基准测试套件D ROID B ENCH，并将其用于比较F LOW D ROID与AppScan Source和Fortify SCA的商业工具，结果表明还发现了更多的真实泄漏（所有泄漏的93％总数），F LOW D ROID也具有更高的精确度（86％），从而导致伪造数量减少。我们希望未来的D ROID B ENCH将成为Android恶意分析的标准测试集。

为了分析来自Google Play商店的前500个真实世界的应用程序，F LOW D ROID每个应用程序只花费了不到一分钟的时间，发现了一些泄漏。大约每分钟16秒钟分析约1000个恶意软件样本，平均每个样本发现2个泄漏。对SecuriBench Micros的F LOW D ROID评估结果显示96％的召回率只有9次误报。