访问控制是软件安全的关键支柱之一[38]。许多访问控制模型用于有选择地限制对软件系统对安全敏感的资源和功能的访问。在这些模型中，基于权限的访问控制近年来得到了广泛的关注，部分原因是它在很多流行平台中得到广泛应用[24]，包括Android。

在Andriod中，权限授予应用程序。 Android运行时环境可防止缺乏适当权限的应用程序访问敏感系统资源（例如传感器）以及其他受保护的应用程序。最初，Android采用了静态权限系统，这意味着在安装之前提示用户同意应用程序所请求的所有权限，并且授予的权限不能在接下来被撤销。为了让用户更好地控制他们的设备，2015年，从API级别23开始，Android切换到动态授权系统，允许用户在运行时更改授予应用程序的权限[3]。

然而，引入动态权限系统对测试Android应用程序提出了重要挑战。在应用程序上执行的测试可能会在授予的任务组合下通过，但在不同的组合下失败。正如Android最佳实践所建议的：“从Android 6.0（API级别23）开始，用户在运行时授予和撤销应用程序权限，而不是在安装应用程序时这样做。因此，您必须在更广泛的条件下测试您的应用程序。“[3]。

在最新技术水平上，正确地测试Android应用程序的权限保护行为需要重新执行每个测试对应用程序请求的所有可能的权限组合，因为没有可用的工具来协助确定测试和权限之间的相互作用。这种详尽的方法非常耗时，而且通常不切实际，特别是在回归测试的情况下，整个测试套件的执行需要针对指数数量的权限组合重复执行。

为了缓解这一挑战，我们开发了PATDroid，这是AnDroid的权限感知GUI测试的简称。引导我们研究的见解是，给定的测试可能不会与应用程序请求的所有权限进行交互，这意味着某些权限（无论它们是被授予还是被撤销）可能不会影响应用程序在特定测试下的行为。通过排除不与测试相互影响的许可，我们可以实现测试工作量的显着减少，但仍能实现与彻底测试相当的覆盖率和故障检测能力。

PATDroid利用混合程序分析方法来确定应用程序的GUI测试与其性能之间的交互。它首先动态地确定每个测试用例所执行的应用程序的入口点。然后静态检查可从已识别的入口点访问的代码部分以找到受权限保护的代码片段。之后，它会静态确定控制执行受权限保护的代码片段的应用程序输入（即GUI小部件）。最后，它在测试脚本中静态标识应用程序输入的用法。采用足够精确且可扩展的技术，PATDroid能够有效地确定在应用程序的许可权组合下应执行哪些测试。

我们的实验表明，PATDroid能够平均减少71％的测试次数和执行时间，同时保持与所有权限组合测试的详尽执行类似的覆盖率。另外，通过使用PATDroid，我们能够识别真实世界中的应用程序中的一些缺陷，正如其开发人员所确认的那样，它们只能在特定的权限设置下公开，进一步证明了PATDroid在实践中的实用性。

本文做出如下贡献：

•理论：就我们所知，第一种方法是考虑程序，测试套件和访问控制模型之间的依赖关系，以减少测试工作量;

•工具：实现Android程序方法的全自动化环境，并公开发布[10];

•实验：对大量真实世界的Android应用程序进行实证评估，展示其效果。

本文的其余部分安排如下。第2节介绍了一个说明性的例子来激励研究。第3节提供了PATDroid的概述，更多细节在第4-7节中介绍。第8节提供了与该工具相关的实施信息，以实现我们的方法。第9节介绍了这项研究的实验评估。最后，本文概述了相关的研究并结束了对未来工作的讨论。

2例证

我们使用名为Suntimes的Android应用程序的简化版本来激励研究并说明我们的方法。 Suntimes计算并显示特定位置的日出，日落和暮色时间。 它开发的目标是Android版本6.图1中捕获了此应用的示例屏幕截图。如果用户授予“地点”权限，则应用程序会根据当前用户的位置定期计算并更新日出，日落和暮色时间。或者，用户可以通过手动提供特定的纬度和经度，或使用GPS获取位置数据（图1c），从选项菜单（图1b）按需更新当前位置。然而，当用户先前拒绝了所请求的位置权限时，Suntimes会尝试使用GPS更新当前位置，因为那时该应用程序既没有被授予所需的权限（即位置）来完成此任务，也没有再次要求。

为了验证它的行为，Suntimes提供了一个GUI测试套件，图2显示了其中的一个子集。与单元测试相比，这些测试运行在硬件设备或仿真器上，通常称为仪器测试[4]。无论测试框架如何（例如Espresso [5]，Robotium [13]），仪器化测试都会被编译并打包为独立的apk文件，并与主应用程序的apk一起安装。为了在整篇文章中区分这两个软件构件，我们将包含测试套装和测试库的apk作为测试用途应用程序（THA），并将主应用程序的apk作为App Under Test（AUT）进行调用。

在图2所示的测试案例中，testSunTimesNavigation（Test＃1）验证了不同日期和日期之间的平滑导航，testSettingLocationToUserDefined（Test＃2）验证了基于GPS数据添加新的用户定义位置和testExportLocations（Test＃3）确保将检索到的位置信息导出到存储的正确性。自Android版本6以来，建议使用授予和撤销权限的各种组合来测试应用程序，以确保应用程序在不同条件下的正确行为[3]。例如，testSettingLocationToUserDefined只有在开发人员在运行测试之前撤销了“位置”权限时才会显示上述崩溃。

2893/5000

作为另一个例子，考虑图2的测试＃3，它需要位置和存储权限来保存用户的位置。根据授予Suntimes的权限，测试＃3可以显示不同的行为：

1）两个必需的权限已被授予，Suntimes能够成功地将用户的位置保存在外部存储上。

2）只有位置许可已被授权。因此，Suntimes要求获得存储许可。在拒绝的情况下，Suntimes会将位置信息保存在应用程序的内部存储中，而不需要存储权限。

3）只有存储权限已被授予。因此，Suntimes要求提供位置权限。如果拒绝，该应用程序不采取任何行动。

4）以前并未授予所需的权限。因此，Suntimes要求他们两人。如果拒绝，该应用程序不采取任何行动。

在任何情况下，如果用户拒绝任何请求的权限，

Suntimes不应该崩溃。

随着应用配置文件a.k.a. Manifest中定义的许可数量的增加，在不同的许可设置下测试应用行为的问题变得更加复杂。一种方法是随机授予和撤消权限并运行测试套件。虽然很简单，但这种方法无法彻底测试应用程序的行为，并容易漏掉重要的缺陷。另外，开发人员可以手动查看应用程序的测试脚本和源代码，以确定应在哪些应用程序权限下执行哪些测试。但是，这种方法非常麻烦，特别是考虑到每次应用程序的源代码发生更改时，开发人员都需要手动建立应用程序测试与其权限之间的关系。

另一种方法是在请求的权限的所有可能组合下彻底运行测试套件。在这个方法中，如果一个应用程序需要p权限，每个测试都应该执行2p次，因为每个权限都取两个{ndranted，revoked}的值.1例如，Suntimes在它的表单中请求四个许可，位置，存储，闹钟和互联网）。考虑到图2中的三个测试，详尽的方法运行每个测试24 = 16次。对于图2中显示的3个测试，我们需要总共3×16 = 48个测试运行。显然，这样的方法不会随着请求的权限数量和测试套件的大小的增加而扩展。

指导我们研究的见解是，对所有权限组合的测试的详尽执行过于保守。例如，我们发现测试＃2只需要位置权限，因为此测试执行的代码不需要访问由其他权限保护的功能。因此，此测试只能执行两次 - 无论是否具有“位置”权限 - 而非详尽情况下所需的16次。

3 APPROACH OVERVIEW

正如前一节所述，在所有流行的Android测试框架（例如[5]，[13]）中，都会编译和打包测试套件以生成测试套件应用程序（THA），该应用程序与App Under测试（AUT）。给定一对THA和AUT，PAT- Droid标识AUT的许可组合的最小数量，应该针对THA中嵌入的每个测试用例进行测试。图3描述了PATDroid的概述，由四个主要组件组成。

PATDroid首先标识AUT中可嵌入THA中的测试用例的那些部分。然而，这是一个具有挑战性的任务，因为测试套件和测试主题是以两个独立的软件工件（apk文件）的形式实现的。此外，与例如单元测试相比，THA由需要更多参与分析的仪器化测试用例组成。与没有Android框架依赖的单元测试并直接调用AUT的方法相比，装备测试运行在硬件设备或仿真器上，并通过GUI事件间接触发一系列操作。触发的GUI事件最初由测试框架，然后是Android运行时环境，并最终将其委托给AUT中的某些方法，称为入口点。由于这种隐式依赖性，静态分析无法解析由THA执行的AUT的部分。

为了缓解解决AUT和THA之间静态关系的困难，PATDroid采用了混合（静态和动态）方法，以两种粒度级别跟踪AUT和THA之间的依赖关系。首先，在方法级别，动态分析标识AUT的入口点方法，运行嵌入THA的测试结果（表示为图3中的TE集合）。其次，在子方法级别，静态分析组件将动态分析发现的入口点缩小到特定测试用例可执行的块。入口点方法的选定块是进一步静态分析的目标。

要了解限制分析范围的必要性，请回想一下图2所示的Suntimes应用程序和测试套件。第二个测试（testSettingLocationToUserDefined）通过从主菜单（第11行）选择Location选项来触发事件，通过图4所示的入口点方法进行处理。该方法是动态分析器为测试＃2识别的入口点方法。但是，仔细检查onOptionsItemSelected方法，显然只有第三种switch语句（即图4中的第10-12行）可由test＃2执行，因为其他情况旨在处理从未触发的其他选项通过这个测试。包括整个方法，而不是在第10-12行上进行讨论，在搜索相关权限时会增加我们分析的假阳性率。

以上示例演示了GUI事件处理程序的执行流程由触发这些事件的小部件控制。因此，精确的分析也应该使GUI小部件考虑到应用程序的控制，否则它会过度接近每个测试可能执行的代码段。为此，THA分析仪确定每个测试用例中使用的小部件（在图3中表示为TW），而AUT分析仪确定在AUT中执行每个代码块所需的权限（如果有的话）以及小部件操作这些块的可达性（在图3中表示为EWP）。

最后，Interaction Detector集成了静态和动态组件的输出，并为每个测试用例（如图3中的地图TP所示）生成相关权限。以下各节更详细地介绍了PATDroid的四个组件。

4 DYNAMIC ANALYSIS

与使用单一主要方法的传统Java程序不同，Android应用程序包含由框架隐式调用的几种方法，通常称为入口点。入口点负责处理各种事件，包括GUI事件（例如，图4中所示的onOptionsItemSelected，用于处理菜单选项的选择），以及更改应用程序的状态，即生命周期事件（例如onResume激活已暂停的应用程序）。

作为运行测试的结果，应用程序的入口点由Android框架调用。这些由Dynamic App Analyzer组件识别。为此，PATDroid首先自动测试给定的AUT，并在应用程序每个可能的入口点的开始处注入记录器，这些记录器可以通过实现Android框架的特定接口（例如onOptionsItemSelected，onResume等）。对于Android入门级接口的综合列表，我们依赖于之前的研究成果[20,39,40,50]。

PATDroid随后使用任意权限设置在指导应用程序上运行整个测试套件。由于入口点的调用独立于权限设置，因此我们的方法有效地找到了AUT中依赖于THA的入口点。与测试脚本不同，入口点内的代码取决于测试执行期间的权限设置。因此，我们使用第6节中描述的静态分析技术来进一步研究入口点方法中的逻辑。

最后，处理通过应用程序入口点的检测获得的日志，以捕获每个测试用例的执行入口点。生成的这个阶段的输出，称为TE，是一组元组⟨test，entryPoint⟩，其中第一个元素是测试标识符，第二个元素是测试执行过程中暴露的入口点。图5提供了图2的测试＃2的生成输出的子集。

5 STATIC ANALYSIS OF TEST HARNESS APP

正如在第3节中所讨论的，PATDroid以两个粒度级别跟踪AUT和THA之间的依赖关系。如前一节所述，在粒度较高的情况下，方法级别的依赖性通过动态分析来识别。在粒度较低的情况下，在入口点方法中，依赖关系通过静态分析得到。

为了静态追踪AUT和THA之间的依赖关系，PAT-Droid解析了应用程序输入，即GUI小部件，它们是测试脚本执行的操作的目标。在运行示例中，action\_location\_add是THA和AUT工件（图2和4中分别为第11行和第10行）中使用的小部件标识。为此，PATDroid的静态分析组件从AUT和THA中提取窗口小部件信息。提取的信息应该在整个应用程序的实现中唯一标识小部件，因此通常包含小部件标识符或密钥。虽然本节主要关注从THA中提取小部件，但第6.2节介绍了我们的方法如何应用于AUT。

每个Android测试框架（例如，Espresso [5]，Robotium [13]等）都基于框架的API和模式以自己独特的方式对小部件交互进行编码。为了概括找到使用的小部件的问题并使我们的方法不受测试框架的影响，我们将这个问题定义为一般的数据流分析。因此，我们的目标是从测试程序中找出从某些来源到接收器的数据。为此，数据源被定义为用于通过特定属性检索小部件的一组测试框架API，例如，使用Espresso中的ViewMatcher.withId（int）和Solo.findViewById（String）API基于ID发现小部件和Robotium框架。类似地，数据接收器被定义为用于对小部件执行动作的测试框架API集合，例如由ViewActions.click（）和Solo.clickOnButton（）API定义的点击动作。

以这种方式解决问题使我们能够独立于测试框架执行静态分析。为了支持一个新的测试框架，只需要提供框架的API列表来检索和执行对小部件的操作。稍微快一点但不太精确的找到小部件的方法是只查找小部件检索API（即只有源集合）并简单地返回提取的信息。但是，这种方法可能会增加误报率，因为除了执行某个操作（例如，断言）之外，可能会为了某些目的而检索某些小部件。出于这个原因，我们选择了一个精确的分析。

为了解决数据流问题，我们采用了Android兼容的数据流分析框架FlowDroid [20]，但有一个重要的修改，使我们能够对THA进行分析。默认情况下，FlowDroid旨在分析符合Android框架所期望的常规结构的应用程序，例如由Android组件组成。与AUT相比，THA不遵循这种传统结构，因此不受FlowDroid的支持。因此，我们用定制的创建者替换了FlowDroid的默认入口点创建者，该创建者专门为THA分析量身定制。对于每个THA，PATDroid都会创建一个dummy主方法，负责准备在@Before方法中编码的测试环境，然后调用嵌入在THA中的@Test方法。回想一下在图2所示的测试脚本示例中使用这些注释。

解决数据流问题时，THA分析器生成输出TW，它是一组元组，其中第一个元素是测试标识符，第二个元素是一个小部件，它是测试所执行的动作的目标。图6提供了针对图2的测试＃2生成的分析输出的子集。

6 STATIC ANALYSIS OF APP UNDER TEST

在任意权限设置下运行，动态应用分析器部分探索每个测试可执行的AUT代码。随后，PATDroid利用AUT分析器静态地检验每个测试可以执行的代码的所有部分。如图3所示，AUT分析仪接收AUT和TE作为输入并生成EWP作为输出。生成的输出是一组元组，每个元组包含三个元素：⟨entryPoint，wid，et，permission，指示在执行测试期间调用的入口点方法，一个可以提供受权限保护的代码的可访问性的小部件在该入口点以及相应的许可范围内。算法1总结了AUT分析仪的主要步骤。

分析过程执行几个步骤来生成输出。最初，PermissionAnalysis子过程（第2行）为测试套件执行的所有应用程序的入口点方法标识了执行每个语句所需的权限（如果有的话）。这个子程序的细节在6.1节中描述。随后，在第3行中调用WidgetAnalysis过程来确定由每个Widget控制的语句，其细节在6.2节中描述。

对于每个入口点方法（第4行）和其中的每个语句（第5行），算法都会确定它是否是受权限保护的方法调用语句（第6行）（第7-8行）。这些可以是Android API调用或用户定义的方法。对于每个受权限保护的方法调用语句，检索控制该语句执行的所有小部件（第9行）。最后，该算法添加了由方法，小部件和权限信息组成的元组来设置EWP，除非它们已经存在于这个集合中（第14-15行）。如果受权限保护的语句不受任何小部件控制，则在相应生成的元组中将小部件元素设置为Null（第10-11行）。

6.1 Permission Analysis

对于给定AUT中定义的每种方法，权限分析过程通过执行程序内点分析（在算法2中总结）捕获执行该方法所需的所有权限，称为权限摘要（PS）。

在第一步中，权限分析会构建整个应用程序的调用图（CG）（第2行）。然而，由于Android平台的事件驱动结构，传统的CG生成方法不会连接隐含通话所对应的呼叫站点。在之前的研究中广泛讨论了为Android应用程序生成调用图的挑战，并为此提出了几种适用于PAT-Droid的技术[20,46]。图7描绘了Suntimes应用程序的调用图的子集。在此图中，隐式调用由虚线表示。例如，方法GetFixHelper.getFix（）通过调用execute（）接口来启动一个AsyncTask，即GetFixTask。因此，任务类的doInBackground（）方法由Android框架间接调用。

Permission Analysis迭代遍及给定应用程序调用的所有Android框架API（第3-7行），并将API所需的权限添加到调用该API的方法的权限摘要（PS）.3我们依赖权限 - 在之前的工作中产生的API映射[21,23]，以确定Android API所需的权限。

最后，Permission Analysis使用广度优先搜索（BFS）方法遍历构造的调用图（CG）（第9-15行）。从给定的入口点方法（EE）开始，它在图中传播权限。在每次迭代中，算法通过用被调用方法的PS（第13行）增加它们的PS来更新调用当前方法的所有方法的权限摘要（PS）。重复此过程直至获得许可摘要的固定点（第16行），这意味着PS在进一步迭代中不会改变。在图7中，权限摘要显示在每个调用图节点的左上角。

6.2 Widget Analysis

回想一下图4中介绍的入口点方法。为了处理选择的菜单选项，这个方法（onOptionsItemSelected）调用其他几个方法，每个方法在与菜单选项相对应的情况下。例如，Set Alarm（图1b中的第三个选项）由图4所示的第一个case语句处理，其中scheduleAlarm（）方法因此被调用（第5行）。因此，如果GUI测试仅点击Set Alarm选项，则不会执行其他情况下调用的方法，因此对其他方法（例如showSetting（），configLocation（）等）的总结许可不相关到这个测试。为了排除不相关的许可，我们需要确定哪些小部件等控制哪些程序语句，特别是调用带有非空许可摘要的方法的语句。在算法3中总结的小部件分析过程提供了此功能。

对于给定的方法，Widget Analysis过程执行分支敏感的部分程序间数据流分析并生成Widget Summaries（WS）作为输出。为此，首先构造一个修剪版本的程序间控制图（ICFG）（第3行）。 ICFG是在所有呼叫站点相互连接的一组控制流图。然而，我们的分析专门针对应用程序小部件，因此，只有通过小部件对象的调用站点才包含在修剪的ICFG中，表示为ICFGT。通过ICFGT而不是ICFG进行分析，显着提高了我们方法的可扩展性，但保持了可接受的精度。

之后，通过迭代每个方法的每个语句（第4-11行）来填充gen集。我们只关注条件语句，即程序的控制，即IF（第5-6行）和SWITCH（第7-10行）语句，以widget作为条件。例如，图4中的switch语句可以作为我们分析的目标，因为它是（1）控制程序流的一个条件语句，（2）一个控件，即MenuItem用作语句的条件。

最后，该算法以广泛的第一种搜索方式遍历ICFGT，并通过图形传播小部件信息。通过这样做，在每个声明中，我们都有可以从开始到该声明控制程序控制的所有小部件的信息。例如，如图7所示的onOptionsItemSelected方法的控制图中突出显示的那样，使用location\_add作为选定的菜单选项，程序的控制权将到达第11行和第12行。因此，location\_add被添加到小部件节点11和12处语句的摘要。小部件分析在达到小部件摘要（WS）的固定点时终止。

必须注意6.1节和6.2节中描述的两个子程序的精确度和范围之间的差异，即权限分析和小组件分析。由于分支和分支的灵敏度，Widget分析比许可分析的成本更高。另一方面，虽然通过遍历其调用图对应用程序中的每个方法执行权限分析，但Widget分析的范围仅限于通过运行测试执行的几个入口点方法。这种区别让PAT-Droid能够保持应用程序分析的准确性并且可扩展。

结合权限分析和Widget分析子程序的输出，主程序（算法1）产生AUT分析器组件的最终输出，即EWP。图8提供了Suntimes应用程序生成的EWP子集。

7 BUILDING PERMISSION COMBINATIONS

如图3所示，交互检测器生成最终输出TP，它是从测试到相关权限集合的映射。它通过将其他组件的输出（即TE，TW和EWP）关联如下来实现。

交互检测器程序，在算法4中总结，它对三个输入集（TE，TW，EWP）进行了描述，并匹配元组

这些集合的成员基于共享元素，即入口点，测试和小部件。唯一的例外发生在没有找到EWP的小部件时（即，没有小部件用于控制对受权限保护的代码的访问一个入口点），在这种情况下，保守地认为整个入口点方法可以通过单个测试执行，因此算法不会尝试匹配EWP.widдet和TW.widдet（line5）。基于匹配的元组，测试权限被添加到输出TP（第9行）。最后，将一个空集分配给那些没有相关权限的测试（第14-17行）。图9提供了Suntimes应用程序测试集的生成输出T P。

此算法的输出可以对给定应用程序进行有效的权限感知测试。总的来说，对于由T测试和P权限组成的应用程序，PATDroid的测试运行次数计算如下：

其中TP [t] .perms表示PATDroid确定的所有适当的权限。正如我们的实验所显示的那样，这个数字显然比| T |小得多×2 | P |在详尽的方法下执行所需的测试。

8 IMPLEMENTATION

PATDroid是由2,500多行Java代码和800行Python脚本实现的。 它还依赖于一些第三方库，其中最值得注意的是Soot [53]用于Android应用程序的静态分析，IC3 [46]用于解析ICC通信，Xposed [17]用于运行时处理根Android过程。

PATDroid以两种模式运行：（1）开发者模式和（2）测试者模式。 当主题应用程序（AUT）及其GUI测试（THA）的源代码可用时，第一种模式适用。 当只有apk文件（AUT）可用时，可以使用第二种模式。 PATDroid目前支持主要的Android GUI测试框架，即Espresso，Robotium和Monkey。

与PATDroid相关的工件，包括可执行工具及其用户手册可从PATDroid的网页下载，可通过以下链接访问：<http://www.ics.uci.edu/~seal/projects/patdroid/>

9 EVALUATION

我们对PATDroid的评估解决了以下问题：

RQ1。 效率：在测试运行时间和测试执行时间方面，PATDroid如何与替代方法进行比较？

RQ2。 覆盖范围：How doesPATDroid比较代码覆盖方面的替代方法？

RQ3。 有效性：PATDroid是否能够揭示真实世界应用程序中的缺陷，特别是那些仅在特定权限设置下才暴露的应用程序？

RQ4。 性能：PATDroid如何根据应用程序的大小进行缩放？

9.1 Experiment Setup

为了评估我们在现实主题方面的做法，我们抓取了Google Play和GitHub存储库，并使用以下条件搜索Android应用：

（i）应针对Android API级别≥23;否则，该应用程序不支持运行时权限修改，因此不适用于我们工作重点所在的问题。

（ii）由于其他类型的权限在运行时不可调，因此用少于两个可调权限解决问题是微不足道的。

根据上述标准，我们收集了110个应用程序：（1）来自Google Play的100个热门应用程序，以及（2）来自Github的10个开放源代码应用程序（列于表1），因为调查RQ2即测量代码覆盖率，需要源代码的可用性。

对于开源应用程序，我们使用Espresso [5]或Robotium [13]框架手动创建或扩展了现有的GUI测试，以实现至少50％的语句覆盖率。对于Google Play应用，我们使用Monkey [8]来生成黑盒GUI测试。

我们将PATDroid与三种替代策略进行了比较，如下所示：

详尽无遗地包括所有权限组合。按成对技术成对生成[44];也就是说，对于任何两个权限，所有可能的权限设置对（即，授予，撤销）都应该在输出集中。所有 - 无 - 包括两种组合，一种授予所有许可，另一种授予所有权限。

正如我们将在第10节中讨论的那样，现有的测试套件缩减工具都不支持Android框架，也没有考虑其访问控制模型，因此未包含在我们的评估中。

9.2 Efficiency

为了回答RQ1，我们将PATDroid的测试运行时间和测试执行时间与穷举，成对和全和无比较，如表2所示。测试运行大小表示每个运行所需的累积测试次数技术。该数字通过表2中显示的公式计算，在相应的列下。此外，表格显示了与PATDroid相比，每个报告指标的减少或增加的百分比。

表2中的结果确认PA​​TDroid可以显着减少测试运行次数和测试执行时间。平均而言，PATDroid要求的测试执行次数分别比穷举和成对要少71.35％和41.78％。同样，平均而言，PAT-Droid的执行时间分别比穷举和成对要少71.07％和39.07％。然而，与全无相比，结果是混合的，在某些情况下，PATDroid实现更高的降低（例如预算监督），而在其他情况下，PAT-Droid实现更低的降低（例如RadioBeacon）。虽然在某些情况下全部和部分都没有达到更高的缩减比例，但下一部分显示它不会像其他方法那样保持相同的覆盖范围。

图10绘制了所有110个主题应用程序的测试执行时间。如图所示，测试执行时间相对于穷举方法中的权限数量而言是增长的。因此，与具有更多权限的应用程序相比，穷尽方法的降低率更高。例如，拥有9个权限的A2DP Vol应用程序的情况下减少了99％以上，而具有2个权限的Budget Watch应用程序的情况则减少了近70％。

9.3 Coverage

为了回答RQ2，我们比较了PATDroid所实现的声明和分支覆盖率与其他技术所达到的覆盖率。如表3所示，PATDroid与所有主题应用程序中的详尽覆盖范围相同。 PATDroid达到与详尽覆盖范围相同的事实尤为重要，因为它表明PATDroid不会产生很多错误的否定，即未能通过相关的应用程序的权限组合执行测试。

此外，平均而言，PATDroid的成功覆盖率分别高于成对和全部和非成熟技术14％和10％。值得注意的是，虽然在7款应用程序中实现了与PATDroid相同的覆盖率，但在3款应用程序中，它的覆盖范围显着降低。仔细观察一下PATDroid表现优于两者的应用程序表明，当应用程序提供的某些功能依赖于两个以上权限时，会发生这些情况。例如，AlwaysOn应用程序要求提供四个权限，如果未授予这些权限，则该应用程序的功能将显着降级。由于成对技术不包括授予所有四个权限的组合，因此与PATDroid相比，它的语句覆盖率降低了77％，分支覆盖率降低了93％。

总之，RQ1和RQ2的结果证明PATDroid能够显着减少没有交易代码覆盖率的测试次数。

9.4 E ectiveness

为了回答RQ3，我们调查了我们的方法在识别真实应用程序中与权限相关的缺陷时的强大功能。为此，我们仔细分析了Android日志以及在PATDroid生成的权限组合下执行的测试的输出。特别是，我们对碰撞或意外行为感兴趣，这些行为只能通过在某些权限组合下运行测试来验证。

在110个应用程序集上运行PATDroid，我们发现有14个应用程序（即13％）存在由于动态权限处理不当而导致的缺陷。我们通过GitHub问题跟踪器向开发人员报告了开源应用程序的缺陷以及重现错误和缺陷建议的信息。表4总结了报告的缺陷以及提供公共问题跟踪器的应用程序的每个问题的当前状态。截至本文提交之日，大部分缺陷都由应用程序开发人员进行验证。

请注意，详尽的和成对的方法也能够识别报告的缺陷，除非它们执行的时间明显较长，如9.2节所示。另一方面，无论如何都无法揭示这些问题。例如，在开放笔记扫描仪应用程序中，最初询问所需权限时，在授予相机权限的同时撤消存储权限会导致应用程序崩溃。这种行为是不可重复使用全和无技术。此外，详尽的方法无法找到PATDroid错过的缺陷，进一步证明了PATDroid在揭示许可相关缺陷方面的功效。

9.5 Performance

要回答RQ4，我们测量了运行PAT-Droid超过主题应用程序的性能。实验在配备Intel Core i7 2.4 GHz CPU处理器和16 GB主存储器的PC上运行。根据实验结果，花在识别相关权限上的平均时间为356秒，与缩短测试运行时间所节省的时间相比可以忽略不计（见第9.2节）。

图11显示了运行PATDroid的性能测量结果。 PATDroid每个阶段的分析时间，即静态和动态分析，在图中分别绘制。平均而言，静态和动态分析分别需要97秒和259秒。根据图表，静态分析时间随着应用程序大小的增加而增加，而动态分析时间与应用程序大小之间没有关联。动态分析时间取决于主题应用程序的逻辑和工作负载。例如，应用程序从互联网上下载的数据量可能会影响应用程序系统测试的执行时间。

10 RELATED WOR

减少测试是多个研究领域的研究目标。在本节中，根据我们的研究，我们提供了这些工作的讨论。

组合相互作用测试（CIT）。组合相互作用测试提出了一套技术来减少软件系统的测试空间，同时保持整个测试空间的有效性[47,56]。 CIT方法可以分为贪婪[27,29,30]，启发式[31,45]，遗传[51]和基于搜索的算法[52]。

CIT的另一项研究重点包括识别和消除配置约束。尽管研究的主体[25,28,55]侧重于硬约束 - 那些使配置不可行或不被允许的问题，软约束问题是有效的，但是不合需要的或不相关的配置，先前的研究。就我们所知，PATDroid是第一次试图通过在访问控制的上下文中解决软约束来减少测试空间，即排除与测试用例无关的应用程序权限。

测试软件产品线（SPL）。构成软件产品线（SPL）的程序的系统测试是昂贵的，因为它需要检查测试功能的组合。因此，一些先前的作品[34-37]试图减少SPL的测试空间。例如，SPLat [37]及其前身[35]利用动态和静态程序分析技术来识别和排除每个测试用例的不相关特征。

尽管具有类似的目标，但由于多种原因，所提议的用于减少SPL测试的技术不适用于Android应用程序的权限感知GUI测试。与代码中明确指定的SPL功能相比，应用程序权限并未嵌入到应用程序代码中。而且，SPL测试缩减技术的目标是单元测试。尽管单元测试可以从程序中进行追踪，但在Android程序中，源代码和GUI测试套件之间没有明确的依赖关系，如第5节所述。

回归测试选择（RTS）。大量的先前研究集中于加速连续集成过程中的回归测试[32,48,49,57]。与这些在le，class或method级别跟踪依赖关系的RTS技术相反，PATDroid在控制级捕获依赖关系。应用精确但可扩展的分析，PATDroid能够显着缩短测试时间。尽管用于跟踪测试和被测系统之间依赖关系的技术相似，但我们的方法与RTS的目标相当不同。尽管RTS技术旨在确定与软件修订版中引入的代码库变更相关的测试，但PATDroid的目标是发现GUI测试的相关权限。

上述三类相关研究都不适用于Android。接下来简要介绍Android GUI测试中的相关工作。

Android GUI测试。 Android GUI测试近年来受到了广泛的关注[26]。提出的方法采用多种技术，包括随机[18,19,33]，基于行为的[41]，基于模型的[22,54]和基于搜索的[42]方法。

尽管如此，只有少数研究方法将组合测试缩减技术应用于Android测试领域。最值得注意的是，TrimDroid [43]提取小部件之间的依赖关系，以减少GUI测试中的组合数量。然而，TrimDroid与我们的方法有很多不同之处。首先，TrimDroid对应用程序代码执行静态分析以生成新的测试用例，而PATDroid对待测应用程序和测试用具应用程序使用混合分析来确定用于运行现有测试的许可组合的子集。其次，TrimDroid捕获窗口小部件之间的依赖关系，而PATDroid则追踪应用窗口小部件和权限之间的依赖关系。据我们所知，访问控制模型，特别是权限，在先前的任何测试生成或减少研究中都没有考虑到。

11 CONCLUSION AND FUTURE WORK

最近在Android中引入动态权限系统使得有必要在各种权限设置下测试Android应用程序的行为。如果没有一个自动化解决方案来推断哪些测试应该在什么样的权限组合下执行，那么开发人员必须手动进行这些测试，或者在许多权限组合下彻底重新运行每个测试。两种方法都不切实际，耗时且麻烦。

为了克服这个问题并帮助开发人员在各种权限设置下有效测试Android应用，我们推出了PAT-Droid。通过对Android应用程序及其测试套件的混合程序分析，PATDroid能够识别每个测试用例的相关权限。通过排除不相关的权限，PAT-Droid能够显着减少测试运行次数和测试执行时间，而无需交易覆盖范围和测试故障检测能力。我们的实验结果表明，与详尽的方法相比，PATDroid在测试执行时间方面可以减少71％，代码覆盖率没有任何降低。而且，使用PATDroid，我们能够在真实世界的应用程序中识别出以前未知的许多与权限相关的缺陷。

我们目前的PATDroid实施有两个限制，这将是我们未来工作的主题。首先，假定GUI小部件由一个唯一的标识符（如ID或密钥）检索。虽然这种假设在绝大多数情况下都是正确的，但小部件偶尔会通过多个属性的组合来检索，例如位置和父母的ID。

其次，假设程序的GUI依赖流程直接使用小部件进行控制。这意味着，如果程序块的执行取决于特定的GUI事件，则开发人员会在小部件上指定一个条件防护来触发该事件。然而，另一个不太常见但合理的方案是开发人员间接使用该小部件。例如，通过在全局变量中保存对小部件的引用，并使用该引用来控制程序的流程。

通过构建更高级的静态分析可以解决上述限制。然而，额外的精度很可能会降低我们方法的可扩展性和性能，从而在总体环境中获得小的收益。研究这些行业将成为我们未来研究的重点。

我们计划扩展我们的方法，在Android中纳入其他可配置参数，这些参数可以反映程序的行为，例如网络和电池使用情况的设置。将PAT- Droid概括为超出许可范围将需要使用将API映射到其他Android配置（例如网络设置）的输入来替换许可 - API映射（算法2）。

最后，由于PATDroid的理论贡献适用于具有基于权限的访问控制模型的任何软件，我们打算调查我们的方法适用于其他使用基于权限的安全模型的平台的适用性。