2 基于流分析技术的静态漏洞检测

静态分析的第二个方面主要对Android应用可能存在的漏洞进行检测，这些Android应用的漏洞虽然可能并不含有恶意代码，不会直接对用户产生攻击行为，但如果被攻击者恶意利用的话，同样可以对用户产生相当严重的威胁。比如在Android应用开发中对证书校验部分的代码编写存在问题而没有实现证书的有效校验，则可能产生中间人劫持攻击，攻击者通过拦截正常的网络通信数据，可以进行数据篡改和嗅探，而通信的双方却毫不知情。

2.1 Android应用漏洞检测方案

基于静态分析的漏洞检测是在不运行app的情况下对反编译后的APK进行分析，通过特征匹配来发现应用中的漏洞。传统的检测方案是利用反编译工具将dex文件转换为smali中间代码，之后对关键词以及特征参数的匹配都是在smail代码上基于字符匹配完成的，这种分析方法虽然简单但同样存在局限性，首先在精度上会比较低，并且只能对特征单一，容易检测的漏洞进行特征匹配，无法对需要结合上下文语义，逻辑复杂的漏洞实现高精度的识别。为了解决这种问题，本系统的漏洞检测模块使用Java静态分析框架Soot进行漏洞检测，Soot反编译所生成的Jimple中间代码只含有15种语句，另外在反编译初始化时，Soot同时会同时进行两项工作，一是对中间代码进行数据填充以支持使用者结构化的分析这些中间代码，二是为分析的代码生成控制流图，允许使用者采用数据流技术来解决问题，整个初始化过程如图x.所示。本系统的漏洞检测模块将在Soot所提供的静态分析框架的基础上，基于过程内数据流分析技术设计并实现从变量到相关常量的追溯方法，提高漏洞的检测精度。



图 静态漏洞检测的准备工作流程

2.3基于过程内数据流分析实现变量到常量的追溯

在对漏洞特征进行匹配时，经常将特定的常量参数做为漏洞特征，然而应用开发中有很多情况是先将常量赋值到一个变量对象里，在进行处理之后再作为参数传值到漏洞特征函数中，一个最简单的例子就是使用StringBuilder连接字符串，最后调用toString()方法将生成的字符串作为参数传给函数，对于这种情况，传统基于字符匹配的检测方案是无法匹配到特征的，本文基于过程内的流分析技术，实现变量到常量的追溯从而解决这种问题，提高漏洞检测的精确度。

系统中为了实现变量到常量的追溯，首先使用基于前向的过程内数据流分析技术对函数内的局部变量行为进行详细记录，流分析结束后，再根据变量的行为记录进行变量到常量的追溯。在数据流分析中，使用Soot提供的过程内数据流分析框架模拟完整控制结构下的语句遍历分析，并在此过程中详细记录过程中所有变量的初始化，变量的方法、成员变量调用，变量赋值，变量值传递等所有变量相关的行为，力求在函数内为所有局部变量的操作建立详细准确的模型，为之后变量到常量的追溯分析做好基础。

在框架中主要实现两个接口函数如下

表1 过程内数据流分析框架的两个主要接口

|  |  |
| --- | --- |
| 方法名 | 作用 |
| flowThrough(d.inFlow,d.data,d.outFlow) | 对传入语句d.data进行语法分析，并在d.inFlow参数的记录基础上，添加新的记录并作为d.outFlow传出 |
| merge(in1,in2,out) | 对分支结构两条路径的记录结果in1、in2进行合并，生成整合的记录结果out传出 |

flowThrough(d.inFlow, d.data, d.outFlow)函数的具体工作是对传入语句d.data进行分析，并添加新的记录，系统实现中根据不同的语句类型进行不同的解析操作来生成新的记录，几种常见的语句类型对应的操作如下：

1) IdentityStmt为变量初始化语句，对左边的被初始化变量以及右边的初始化类型进行记录。

2)DefinitionStmt为定义类型语句，左操作书是变量的话，根据右边的表达式种类，可能的操作有使用工厂函数进行的变量初始化，变量赋值，变量方法调用操作，根据右侧表达式类型进行分析记录。

3)InvokeStmt为函数调用类型语句，里面可能存在变量的方法调用，将调用的方法解析出来并存放到对应变量的调用记录中去。

merge(in1,in2,out)函数用于将分支结构中两条路径的记录进行合并以整合成一条记录，实现中为了保证所有分支路径下变量调用记录的完整性，对两条记录做并集来生成整合记录，即out = in1 ∪ in2。

过程内数据流分析结束后会输出函数内所有变量的行为记录信息，变量到常量的追溯就基于这些信息来完成，系统实现中在如下位置提取常量作为变量的常量特征：

1)变量的函数调用中所应用的常量参数，如果参数为变量的话则进行递归查询。

2)变量的定义处涉及到的常量，若定义源头来自另一个变量的话，则对此变量进行递归查询

2.4 以WebView远程代码执行漏洞的检测为例说明漏洞检测过程

WebView远程代码执行漏洞是Android应用中相当常见的一个漏洞，危害也相当广泛，本文将以此为例基于Soot提供的Jimple中间代码，以及结合数据流分析技术下的WebView远程代码执行漏洞的检测方案。

Android系统中提供JavaScript调用Java代码的方法，用户只需要在WebView中调用setJavaScriptEnabled(boolean)函数将Js代码的运行开关打开，甚至都不需要向其中提供Java实例对象，系统就默认会在WebView中自动添加searchBoxJavaBridge\_的Java实例对象，对象中的所有方法在JavaScript中均可调用，因此攻击者可以通过反射调用攻击实现任意代码执行。在API>17之后，Android系统通过添加JavaScript注解机制来主动声明Java实例中允许被调用的方法，但实际上如果提供调用的方法中含有敏感API的话，同样可能对用户造成威胁。

检测前首先提取出应用所使用的API版本号，对于API<17的应用，首先遍历出函数内所有WebView类型的局部变量，然后使用2.3说明的过程内数据流分析技术生成过程内所有变量的行为集合，在集合中分别寻找setJavaScriptEnabled(boolean)

removeJavascriptInterface(java.lang.String)

addJavascriptInterface(java.lang.Object,java.lang.String)

三个方法的调用情况，对于将JavaScript调用设置为允许的WebView，则进行进一步的检查，对于没有移除默认接口”searchBoxJavaBridge\_”，以及又添加了新Java实例的WebView，均可判定为存在WebView远程代码执行漏洞。对于API>17的应用，因为提供了注解机制，所以首先寻找addJavascriptInterface(java.lang.Object,java.lang.String)函数的调用，从其中提取出传入的Java实例对象，之后对此Java对象中存在JavaScript注解的方法进行分析，如果方法内含有危险API的话，同样判定此WebView存在漏洞。



图 API<17时的检测方案



图 API≥17时的检测方案

2.5 测试

基于上述检测手段，本系统对11类常见漏洞编写了检测方案，并从公开的恶意软件库VirusShare中下载了6801款恶意Android应用程序进行批量测试，检测结果如下：

表 对VirusShare中6801款恶意应用的漏洞检测结果

|  |  |
| --- | --- |
| 漏洞描述 | 总漏洞数 |
| WebView开启了Javascript，并没有移除默认接口，在API<17时可能导致Web组件远程代码执行漏洞 | 2524 |
| API>17时，javascript注解机制所提供的接口中检测到含有危险API | 55 |
| 自实现的HostnameVerifier返回值恒为true，非有效校验 | 704 |
| 自实现的校验证书的X509TrustManager接口的checkServerTrusted()方法实现为空， | 1706 |
| 证书校验中接受任意域名，可能受到中间人劫持攻击 | 39 |
| WebView忽略SSL证书错误检测，可能导致中间人攻击的威胁，可能导致隐私泄露 | 472 |
| 应用程序加解密时密钥使用硬编码，攻击者通过反编译拿到密钥即可解密APP通信数据 | 1595 |
| 检测到应用存在代码动态加载的行为，应用可能隐藏有未检查到的恶意功能 | 3366 |
| AndroidManifest.xml文件中allowBackup属性值被设置为true | 6686 |
| AndroidManifest.xml文件中debuggable属性值被设置为true | 3643 |
| 检测到存在SYSTEM\_ALERT\_WINDOW(系统弹窗)权限，该应用可能在应用外弹窗 | 839 |
| 应用可能尝试使用toast实现全局弹窗，此项全局弹窗并不需要权限声明 | 127 |
| WebView并没有调用setSavePassword(false)，存在WebView明文存储密码漏洞 | 3372 |
| 应用使用WebView，同时支持File协议，在特定情况下可能利用File协议获得应用的敏感数据 | 1515 |
| 文件读写使用全局模式，可能造成隐私数据泄露 | 542 |

3 隐私泄露的研究与检测

静态分析的第三部分着手于Android应用的隐私泄露问题的研究与检测，应用所产生的隐私泄露问题虽不会让用户直观感觉到影响，然而在移动端发展迅猛的当下，手机支付，手机社交都与移动设备紧密相关，可以说手机比电脑中存放着更多用户的隐私信息，这些信息一旦被应用软件恶意获取，将可能对用户带来无法挽回的损失，因此Android应用的隐私泄露问题的研究与检测在Android恶意应用检测系统中是十分必要的一环。

在隐私泄露的检测上，我们首先对Android系统所提供的涉及到用户隐私的API，即污点分析算法中的Sources点，以及可能发送出这些隐私数据的API，即污点分析算法中的Sinks点，进行了详细的分析和归类，之后使用过程间数据流分析技术寻找从Sources点到Sinks点的传播路径，从而检测出可能存在的隐私泄露。

3.1 对Android系统中涉及隐私的API以及有关数据发送API的分析归类

Android系统中含有相当多涉及到获取用户隐私的API，比如能返回用户设别MEID的getDeviceId()，能获得用户精确位置的getLastKnownLocation()，获得浏览器书签信息的的getAllBookmarks()，获得浏览器历史记录的信息getAllVisitedUrls()等等，通过归类整理我们将其大致分为以下6类

表 对Android中涉及用户隐私API的分类

|  |  |
| --- | --- |
| 隐私类别 | Android系统中相关API条目数 |
| 用户位置信息 | 7 |
| 用户唯一标志信息 | 6 |
| 用户私人数据 | 9 |
| 用户网络访问相关信息 | 10 |
| 应用组件信息 | 9 |
| 程序运行间数据信息 | 98 |

在恶意应用获取到用户隐私数据之后，同样可以使用多种手段将数据传输到攻击者手中，诸如通过sendTextMessage()发送短信，通过openConnection()的http请求发送数据，通过Log.d()以日志形式记录数据，通过FileOutputStream.write()将数据写入到本地文件等等，通过归类整理我们将其大致分为以下5类

表 对Android中数据发送相关API的分类

|  |  |
| --- | --- |
| 数据发送方式类别 | Android系统中相关API条目数 |
| 通过短信发送数据 | 3 |
| 通过网络请求发送数据 | 13 |
| 通过运行日志记录数据 | 14 |
| 通过本地文件文件记录数据 | 21 |
| 通过应用组件传递数据 | 108 |

基于对Sources点和Sinks点的归类整理，之后便可以使用过程间数据流分析技术寻找应用运行中从获取隐私信息的Sources点到发送隐私数据的inks点的传播路径，来发现这些隐私泄露问题。

3.3 基于过程间数据流分析技术寻找污点传播路径

Android应用程序的运行方式并不像其他程序一样有个固定的入口点，然后顺序执行下去，Android应用中每一个组件都有自己完整的生命周期，组件在任何时候都可能被触发调用，因此要对Android应用程序进行过程间数据流分析，就需要先为Android应用程序的生命周期建模，并构造一个虚拟的main方法，方法内模拟触发所有可能在运行中触发到的行为，然后再对这个虚拟的main方法进行数据流分析来寻找Sources到Sinks间的路径。本系统使用德国研究人员开发的FlowDroid进行隐私泄露的测试分析，FlowDroid是基于流分析技术的隐私泄露分析工具，它对Android应用程序的生命周期做了完整建模，并构建了一个虚拟的dummyMainMethod方法来模拟生命周期，之后使用流分析技术寻找Sources点与Sinks点间的传播路径，本系统基于FlowDroid所提供的分析接口以及3.2归类的Sources点与Sinks点进行隐私泄露分析。

3.4 测试

我们基于3.1中整理的隐私及泄露方式，并使用 FlowDroid提供的API，对3053款VirusShare中下载的恶意应用做了隐私泄露检测，并对检测结果基于3.1中的分类做了整理，在这3053款应用中，总共发现了1218条对用户隐私的窃取行为

表 泄露的隐私类别统计

|  |  |
| --- | --- |
| 隐私类别 | 在测试中的发现数量 |
| 用户位置信息 | 79 |
| 用户唯一标志信息 | 145 |
| 用户私人数据 | 176 |
| 用户网络访问相关信息 | 270 |
| 应用组件信息 | 409 |
| 程序运行间数据信息 | 139 |

表 隐私数据发送方式统计

|  |  |
| --- | --- |
| 隐私类别 | 在测试中的发现数量 |
| 通过短信发送数据 | 388 |
| 通过网络请求发送数据 | 223 |
| 通过运行日志记录数据 | 267 |
| 通过本地文件文件记录数据 | 157 |
| 通过应用组件传递数据 | 183 |