3 静态检测

3.1 静态检测模块综述

静态检测模块位于 Android 应用安全性评估模型的服务器端，在动态检测模块识别出应用运行中出现的异常行为后，用户可以选择将此应用上传到服务器端，通过服务器端的静态检测模块，对应用进行更加详细、全面的安全性评估，并向 Android 客户端返回该应用的检测报告。

静态检测模块主要划分为三个功能模块，分别从三个不同的方面进行应用安全性的评估。第一个功能模块基于权限与敏感 API 的机器学习分类算法实现恶意软件鉴别，并返回应用是否为恶意软件的判断结果。第二个功能模块用于对 Android 应用进行漏洞扫描，以检测出影响应用安全性的漏洞代码，此外，模块结合数据流分析技术对应用进行漏洞检测，可以有效提高漏洞检测的准确性。第三个功能模块用于检测 Android 应用中可能存在的隐私泄露问题，系统在对 Android 的隐私 API 与隐私泄露 API 进行归类整理后，使用 FlowDroid 寻找隐私数据的传播路径。

通过上述的三个功能模块，服务端在静态分析的基础上，分别从应用是否具有恶意行为、应用的代码编写是否存在漏洞、应用是否存在隐私窃取行为三个角度来综合衡量被上传的 Android 应用的安全性。

3.2 基于权限与敏感 API 的恶意应用鉴别模块

本模块针对 Android 应用程序的权限机制与敏感 API 的调用情况，分别建立基于权限与敏感 API 的特征向量，然后使用 Native Bayes 与 Random Forest 两种机器学习算法建立分类模型。通过对两种分类模型的评估，提出了基于敏感 API 调用的随机森林算法分类模型，从而完成对恶意软件与正常应用的基本鉴别，并反馈回鉴别结果。

3.2.1 特征选取

1)权限信息

Android 应用程序的一系列行为都需要它所申请的权限作为支撑，所以权限在一定程度上反应了应用程序的行为模式[10]。通过对 Android 应用程序的配置文件 AndroidManifest.xml 的分析，提取了关于应用注册的权限信息。

由于 Android 应用程序的权限信息太过冗杂，而且权限之间有较大的相关性，例如在发送短信过程中，必然伴随着接收短信，所以本文对提取出的权限信息进行分类处理，去除权限之间的相关性。这里选取皮尔逊相关系数这种度量方法计算两种权限的相似度[11]，皮尔逊相关系数是一种线性相关系数，用来反应两个变量线性相关程度。计算公式如下所示：

$$p_{XY}=\frac{\sum XY-\frac{\Sigma X\Sigma Y}{N}}{\sqrt{\left(\sum X^2-\frac{(\Sigma X)^2}{N}\right)\left(\sum Y^2-\frac{(\Sigma Y)^2}{N}\right)}}$$

X 与 Y 表示两个变量，p_{XY}表示两个变量的相似度。如果两个权限的相关系数过高，那么就把这两个权限归为一个权限簇，表 6 是经过分类后得到的 7 个权限簇，其中列举了部分权限。

表 6 部分高危权限簇展示

类别	权限簇
1	android.permission.WRITE_CONTACTS
	android.permission.GET_ACCOUNTS
2	android.permission.READ_CALENDAR
	android.permission.WRITE_CALENDAR
3	android.permission.READ_CALL_LOG
	android.permission.READ_PHONE_STATE
	android.permission.CALL_PHONE
4	android.permission.ACCESS_FINE_LOCATION
	android.permission.ACCESS_COARSE_LOCATION
5	android.permission.READ_EXTERNAL_STORAGE
	android.permission.WRITE_EXTERNAL_STORAGE
6	android.permission.RECORD_AUDIO
7	android.permission.READ_SMS
	android.permission.RECEIVE_WAP_PUSH

通过去除权限之间的相关性，可以提高分类器的准确率，而且也在一定程度上减轻了分类器的工作量，提高效率。

b) 敏感 API 调用

敏感 API 包括涉及窃取用户隐私行为的高危函数接口，通过这些函数的调用，程序可以直接或间接地获取一些敏感数据，例如手机联系人、短信、账号、密码等。本文通过对应用程序反编译之后的中间代码进行提取，整理出 53 个调用次

数较多的敏感 API[12]，其中部分 API 如表 7 所示：

表 7 部分敏感 API 列表

类别	敏感 API
1	obtainMessage()
2	getLastKnownLocation(java.lang.String)
3	void execSQL(java.lang.String)
4	sendMultipartTextMessage()
5	getAccounts()
6	setDataAndType(Uri,String)
7	killBackgroundProcesses(java.lang.String)

将敏感 API 的调用次数作为特征值构建特征向量，由于不同大小的应用程序相对调用 API 的次数不同，所以本文将应用程序的大小与 API 调用次数做了加权平均，来降低应用程序大小带来的影响。

3.2.2 机器学习分类算法

不同分类算法的分类性能直接影响了鉴别恶意应用的准确率，本文使用的机器学习算法[13]包括朴素贝叶斯(Native Bayes)与随机森林(Random Forest)算法。

a) 朴素贝叶斯

贝叶斯算法是基于概率框架下实施决策的基本方法，对分类任务来说，在所有相关概率都已知的理想情况下，贝叶斯决策论考虑如何基于这些概率和误判损失来选择最优的类别标记，实现公式如下所示：

$$y = \arg\max_{c \in \mathrm{y}} P(c) \prod_{i=1}^{d} P(x_i \mid c)$$

使用贝叶斯判定准则来最小化决策风险，首先要获得后验概率$P(c|x)$。然而，机器学习要实现的是基于优先的训练样本，尽可能准确地估计出后验概率$P(c|x)$，朴素贝叶斯模型采取“判别式模型”策略：即给定 x，通过直接建模$P(c|x)$来预测 c。

b) 随机森林

鉴于决策树容易过拟合的缺点，随机森林采用多个决策树的投票机制来改善决策树，是一种重要的基于 Bagging 的集成学习方法，可以用来做分类，回归等问题。

对于本系统，由于随机森林算法具备善于选取最优特征的能力，所以很适合从具备大量特征的权限与敏感 API 中找出最具代表性的恶意特征，所以本文选取了随机森林算法作为恶意软件鉴别的主要依据。

3.2.3 模型分析与评估

取正常应用与恶意应用各 1500 个作为训练集，另取 250 个作为测试集，其中恶意应用来自公开的恶意软件库 VirusShare，正常应用爬取自 GooglePlay 商店，基本涵盖了所有应用类别。

首先从应用程序的配置文件提取权限信息，然后将权限信息与高危权限簇匹配，生成特征向量，最后将所有训练集与测试集的特征向量输入到分类器中。实验结果如表 11 所示：

表 11 基于权限特征的实验结果

算法类型	ACC	pre	recall
朴素贝叶斯	0.888	0.958	0.812
随机森林	0.926	0.946	0.907

由表中数据可见，朴素贝叶斯算法预测的精确度较高，但召回率较低，即对所有恶意软件的鉴别效果较差，而随机森林算法的分类性能普遍优于前者，pre 与 recall 分别达到了 94.6%与 90.7%。

不同于权限，敏感 API 信息包含每个 API 的平均调用次数，所以恶意特征的表现方式会更加明显，通过反编译模块得到每个应用的 API 调用情况，结合调用次数建立基于敏感 API 的特征向量，并输入到分类器中。实验结果如表 12 所示：

表 12 基于敏感 API 特征的实验结果

算法类型	ACC	pre	recall
朴素贝叶斯	0.860	0.855	0.867
随机森林	0.957	0.966	0.947

可以看出，随机森林算法的分类性能依然优于朴素贝叶斯算法，无论准确率(ACC)还是召回率(recall)都接近 95%，精确率(pre)也接近 97%，说明利用敏感 API 作为鉴别特征，基于随机森林分类算法能达到很好的分类效果。

3.2.4 样本容量对分类性能的影响

本文在进行模型测试过程中，发现并不是所有模型的分类性能都随样本容量的增加而提升，有时

会出现负增长，鉴于此现象，本文通过控制样本容量进行循环测试，测试过程中依然保证正常应用与恶意应用的比例为 1:1，以每次增加 100 个应用的速率递增，测试结果如图 5 所示:

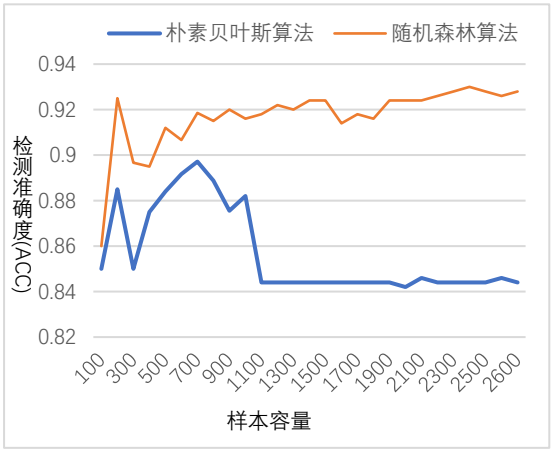

图 5 基于权限信息的样本容量对分类性能的影响图

从图中可以明显看出使用朴素贝叶斯分类器在鉴别恶意应用时准确率出现了负增长，这是因为在样本容量达到 1000 后出现了过拟合问题。

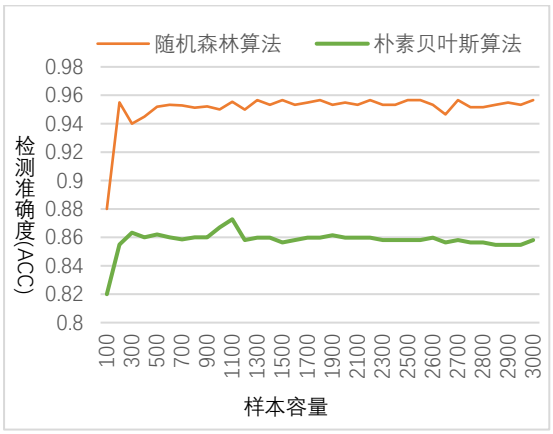

图 6 基于敏感 API 的样本容量对分类性能的影响图

结合样本容量对检测准确度的影响曲线发现

(1) 样本容量的增加并不意味着分类性能提升

(2) 随机森林分类器的分类性能由于朴素贝叶斯，而且不会发生过拟合现象

(3) 敏感 API 的恶意特征比权限特征更加明显，即两种分类器对敏感 API 的分类结果普遍优于权限特征。

综上所述，通过对不同算法的性能分析，本模块决定使用随机森林算法，结合敏感 API 特征鉴别恶意应用。

3.3 Android 应用漏洞检测模块

在 3.2 所述的恶意应用鉴别模块判别出是否为恶意应用后，紧接着运行的是静态检测中的第二个模块——漏洞检测模块。Android 应用中或多或少都可能存在一些编写上存在漏洞的代码，这些 Android 应用漏洞虽然可能并不含有恶意代码，无法直接被 3.2 的恶意应用鉴别模块检测出来，但在被攻击者恶意利用的情况下，同样可以对用户产生相当严重的威胁，降低应用的安全性。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。基于上述情况，本模型从静态分析角度入手，设计了 Android 应用漏洞检测模块。

在下面的小节中，将首先描述漏洞检测模块的工作流程，之后分别详细介绍流程中使用到的详细技术，然后通过 WebView 远程代码执行漏洞为例介绍检测过程，最后则展示本模块在批量测试中的检测效果。

3.3.1 Android 应用漏洞模块的工作流程

漏洞检测模块的工作流程如图 7 所示，首先使用 Soot 对 APK 进行初始化工作，输出的两个结果一个是 Jimple 中间代码，另一个则是控制流图，其中控制流图可以结合数据流技术，用于生成函数内的变量行为记录。结合生成的几种数据可以有效的提高漏洞检测精度。

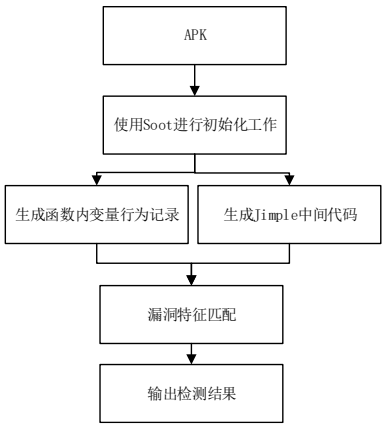

图 7 漏洞检测模块的工作流程

3.3.2 使用 Soot 进行初始化工作

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基于 Soot 进行。

Soot 在初始化过程中会进行两项工作，一是对应用进行反编译生成 Jimple 中间代码，此外，Soot 会对中间代码进行数据填充，并提供丰富的 API 供使用者结构化的分析这些中间代码，二是会基于 Jimple 代码生成控制流图[6]，支持使用者采用数据流分析技术来设计问题的解决方案。在使用 Soot 初始化，生成了中间代码以及控制流图之后，便可以接着进行下面的工作。

3.3.3 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，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，本小节提出的算法则可以解决这种问题以提高漏洞检测精度。

利用 Soot 初始化生成的控制流图，漏洞检测模块可以基于过程内数据流分析技术，为函数内的变量行为生成详细记录，基于此记录实现从变量到相关常量的追溯工作，下面首先描述的是基于过程内数据流分析技术，生成函数内变量行为记录的算法。

在控制流图构建完毕的基础上，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所有变量的初始化、变量的方法和成员变量调用、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。

表 3 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，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

框架主要实现的两个接口函数如表 3 所示。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相应的解析来生成新的记录，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，左操作数是变量的话，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$out = in_1 \cup in_2$。

通过如上定义的过程内数据流分析，即可计算出函数内局部变量的行为集合。行为集合生成完毕后，即可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

上面介绍的变量行为集合，除了可以用于实现从变量到常量的追溯外，在漏洞分析的其他步骤也能够起到重要的作用，具体应用在 3.3.4 所给出的示例中可以看到。

3.3.4 以 WebView 远程代码执行漏洞的检测为例介绍漏洞检测过程

WebView 远程代码执行漏洞[7]是 Android 应用中相当常见的一个漏洞，其危害也相当广泛，本文将以此为例介绍基于 Jimple 中间代码，并结

合变量到常量的追溯算法下的 WebView 远程代码执行漏洞的检测方案。

检测的第一步使用 Soot 进行初始化工作，一方面进行反编译，另一方面则为应用构造控制流图。初始化分析结束后，需要提取出应用所使用的 API 版本号，对于 API<17 的应用，使用 3.3.3 中说明的算法生成函数内所有变量的行为集合，并在集合中分别寻找 WebView 类型以及 WebViewSetting 类型的变量，并在行为集合中定位到三个成员函数的调用

1) setJavaScriptEnabled()

2) removeJavascriptInterface()

3) addJavascriptInterface()

没有调用 setJavaScriptEnabled()来开启 JavaScript 调用的 WebView 变量则不存在此漏洞，而对于将 JavaScript 调用设置为允许的 WebView，则需要进行进一步的检查，对于没有移除默认接口 searchBoxJavaBridge，以及移除默认接口却添加了新 Java 实例的 WebView，均可判定为存在 WebView 远程代码执行漏洞。另外，对特征函数 removeJavascriptInterface(“searchBoxJavaBridge”)的判定因为需要检查字符串参数是否匹配 “searchBoxJavaBridge”，如果代码中参数并非字符串常量的话（比如使用 StringBuilder 进行拼接所得），则使用 3.3.3 中说明的算法进行变量到常量的追溯，然后再来进行判定。

对于 API≥17 的应用，因为提供了注解机制，所以直接查找 addJavascriptInterface()函数的调用位置，之后从其中提取出传入的 Java 实例对象，并对此 Java 对象中使用 JavaScript 注解的成员方法进行分析，如果方法内含有危险 API 的话，同样判定此 WebView 存在漏洞。

3.3.5 漏洞检测模块的运行测试效果

本文基于 3.3 中展示的漏洞检测方案，针对 11 种常见的 Android 应用漏洞编写了检测方案，对在公开样本库 VirusShare 中下载的 6801 款 Android 应用进行了批量测试，结果整理如表 8 所示。

表 8 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

3.4 隐私泄露检测模块

在智能设备大量普及的当下，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恶意获取，就可能给用户带来无法挽回的损失。因此用户隐私安全性与 Android 应用的安全性是紧密相关的，隐私泄露检测模块便从隐私泄露的角度入手，进行 Android 应用安全性的评估

在下面的 3.4.1 小节中，首先对 Android 系统中涉及到用户隐私的 API（即污点分析算法中的 Sources 点，以下简称 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，以下简称 Sinks 点）进行详细的分析和归类。

在 3.4.2 小节中，介绍基于过程间数据流分析技术[8]，寻找从 Sources 点到 Sinks 点的传播路径的污点分析算法。在最后的 3.4.3 小节中，则展示本模块在批量测试中，对应用隐私泄露问题的检测效果。

3.4.1 对 Sources 点及 Sinks 点的分析归类

Android 系统中含有很多涉及到获取用户隐私的 API，例如能返回用户设别 MEID 的 getDeviceId()、获取用户精确位置的 getLastKnownLocation()、获取浏览器书签信息的的 getAllBookmarks()、获取浏览器历史记录的信息 getAllVisitedUrls()等，通过归类整理本文将其大致分为以下 6 类，如表 4 所示

表 4 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发送到攻击者手中，例如通过 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，在进行归类后我们将其大致分为以下 5 类

表 5 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
通过日志记录数据	14
通过本地文件记录数据	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，之后便可以使用基于过程间数据流分析技术的污点传播算法，寻找从 Sources 点到 Sinks 点的传播路径，从而发现这些隐私泄露问题。

3.4.2 使用污点分析算法寻找传播路径

Android 应用程序的运行方式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，组件在运行过程中随时可能被触发调用，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，在方法内模拟触发所有组件，然后再对这个虚拟的 main 方法进行数据流分析，寻找 Sources 到 Sinks 间的路径。

模型使用 FlowDroid[9]进行实现污点传播路径的寻找。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，在对应用内的 Sources 点及 Sinks 点做了标记后，使用过程间数据流分析技术寻找这些点间的传播路径。

本模型基于 FlowDroid 所提供的分析接口以及 3.1 归类的 Sources 点与 Sinks 点进行 Android 应用中的隐私泄露分析。

3.4.3 隐私泄漏检测模块的运行测试效果

基于 3.4.1 中整理的 Sources 点和 Sinks 点，模型对 3053 款恶意应用进行了隐私泄露检测，并对检测结果做了分类整理。在这 3053 款应用中，总共发现了 1218 条窃取用户隐私的行为，统计结果如表 9、10 所示。

表 9 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 10 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