基于动静态结合分析的Android应用安全性多重检测模型

张钦尧 解梦飞 刘俊 陈明翔

(中国民航大学计算机科学与技术学院 天津 300300)

摘要

本文在对大量Android应用程序的研究基础上，提出并设计一种评估Android应用安全性的多重检测模型。模型从动态分析，静态分析，流量分析，三个衡量软件安全性的方向入手，结合机器学习算法，对Android应用中可能存在的恶意代码攻击，漏洞代码利用，隐私泄露三种问题进行了研究并设计检测方案。动态分析基于Android API Hook技术进行敏感行为监控，并自行设计风险评估模型进行预警，此外还基于代理和VPN设计实现了流量分析模块用于进行应用流量的分析拦截。静态分析则分为三方面，首先提取权限信息与敏感API建立特征向量，结合对不同机器学习算法的性能评估，提出了一种基于随机森林算法的恶意软件鉴别模型，第二方面则辅助以数据流分析技术实现漏洞扫描，以提高漏洞检测精度，第三方面则采用FlowDroid提供的污点分析算法实现隐私泄露的检测，

关键词 Android，Hook，机器学习，静态分析，数据流，漏洞扫描，隐私泄露。

0 引言

近年来，Android系统智能设备在全球市场的占有率越来越高，Android系统的安全问题却层出不穷，一方面各大应用市场宽松的审查机制导致Android恶意软件泛滥，在网络上流传不止；另一方面开发者的水平不一也导致Android应用自身的安全性良莠不齐。随着时间的推移，各大厂商的手机大多都已安装了自家的安全软件，并配套了具有审查机制的应用市场。但是，由于Android系统的开源性，许多智能手机爱好者会为系统进行root提权，从而对手机进行高度定制。然而，各大厂商原生的安全机制对root手机不再适用，用户在没有对Android应用的安全性进行确认下，经过root提权的手机（下简称root手机）更容易被恶意软件以高权限入侵。

与此同时，目前常见的Android应用安全性检测方案主要使用动态和静态两种检测技术，但大多数设计方案多只针对其中一项技术进行改进或直接应用，而少有对这些技术检测效果的相关性与互补性进行研究。这明显无法对root手机进行全面、高效的安全防护。

在此背景下，本文提出并设计一种综合多种检测技术的Android应用安全性检测模型，对现有的检测技术进行改进，并利用不同检测方案间的互补性来设计检测流程，一方面可以提高检测的精度，另一方面则可以扩展应用安全性的评估面。此外，本系统针对root手机的特点，在应用安全性评估模型中还着重实现了动态的风险评估和危险行为拦截，具有全面性、精确性、高效性。

1 基于动静态结合分析的Android应用安全性多重检测模型的总体设计

1.1 整体功能

本系统包括：动态监测部分和静态检测部分，分别运行在本系统的Android客户端和服务器端。动态监测部分包括敏感行为监测系统和流量监控系统，静态检测部分包括恶意软件鉴别系统、漏洞检测系统和隐私泄露检测系统。

图1为本系统的各子系统和它们工作模块。

C:\Users\qyzhang_nj\Desktop\图片1.tif

1.2 系统工作流程

图2为本系统工作流程图。



本系统Android客户端启动后，动态监测系统即开始工作，在后台监测所有第三方应用API调用和流量使用状况，若发现某非白名单应用恶意行为，则询问用户是否信任该应用，若用户选择信任，则该应用会被添加到白名单中；若用户选择不信任，则客户端会将应用上传到服务器端，同时对该应用进行恶意软件鉴定、漏洞扫描和隐私泄露检测，并生成一份鉴定结果和分析报告，发送回客户端反馈给用户，同时部分结果和报告内容加入分析规则。

2 动态检测模块

2.1 动态检测模块综述

本系统动态监测运行在Android客户端，在用户运行第三方程序时为监控应用行为，并向用户预警恶意行为。动态监测部分主要划分为两个子系统，分别从两个不同方面进行Android第三方应用行为监控。一方面基于API Hook对Android应用敏感API调用进行钩取、分析，并自设计高效风险评估算法对应用风险进行评估；另一方面使用代理接管Android设备所有网络通信，按规则对Android应用进行流量监控和拦截。

2.2 基于API Hook的敏感行为监测系统

Android 操作系统维护着自己的一套事件分发机制，Android应用程序、及其触发事件和后台逻辑处理，都是根据该机制一步步向下执行，恶意软件的敏感行为也不例外。若能在事件传送到终点前将事件截获并对其操作进行一定修改，就可以有效地抑制恶意行为的效果，这就是Hook机制。

本系统设计一种基于Xposed框架的Hook模块，运行于Android客户端上，利用Xposed框架对Zygote进程及其创建的Dalvik虚拟机的劫持[1]，在开机时完成对所有的Hook方法的重定义或改写，实现对于应用中调用的系统敏感API的钩取、分析、预警。

2.2.1 确定敏感行为

通常，移动平台的恶意行为包含：恶意控制设备、强行推送广告、偷跑流量、窃取用户隐私、获取用户实时动态等[2]，根据以上恶意行为，Hook模块预设表1行为为高风险行为：

表1 高风险操作

|  |  |  |
| --- | --- | --- |
| 行为分类 | 具体行为 | 相关类 |
| 账户操作 | 获取本机账户 | AccountManager |
| 应用管理 | 关闭、重启应用 | ActivityManager |
| 蓝牙操作 | 打开蓝牙、蓝牙连接 | BluetoothAdapter |
| 浏览器操作 | 获取用户书签、历史记录 | Broswer |
| 执行Shell | 各种方式执行Linux Shell | Runtime |
| 系统数据库操作 | 对于联系人、信息、通话记录、  浏览器书签的增删改查 | ContentResolver |
| 下载操作 | 设置URL、下载 | DownloadManager |
| HTTP连接 | 设置URL、连接 | HttpUrlConnection |
| 音频录制 | 录音、设置存放位置 | MediaRecoder |

2.2.2 系统工作流程

模块工作流程包含API Hook模块、风险评估模块、风险预警模块。

a) API Hook

Hook模块随Android客户端安装后托管于Xposed模块，运行于后台。由于不少敏感API会在系统运行的过程中被大量的系统应用调用，所以Hook模块为用户提供选择接口。为防止恶意API之间过度耦合地相互、递归调用而使Hook模块占用大量系统资源，采用在Xposed框架提供的beforeMethodHook和afterMethodHook方法中分别对调用敏感API的线程进行标记和消除标记的方法，只对含有线程标记的敏感API进行分析[3]。

API Hook模块每钩取到一个API，就会记录其调用时间、重要参数，加入到相应Android应用程序的API调用链中，若该应用程序调用链长度达到阈值或者API调用链中出现了固定的恶意API组合，系统就会启动恶意风险评估流程。

b) 风险评估

本系统提出并设计一种风险评估模型，对APP敏感API调用链进行分析、评估，分以下两种情况执行：

1) 若无匹配恶意行为特征API集的调用出现：

Hook模块设计前期，通过对大量恶意软件的反编译并提取其中恶意API调用代码，计算出敏感API集在恶意软件中的分布，表2给出测试统计出的部分敏感API分布情况：

表2 部分敏感API列表

|  |  |  |
| --- | --- | --- |
| 序号 | 敏感API | 调用次数 |
| 1 | android.net.Uri: parse(): | 107121 |
| 2 | android.database.sqlite.SQLiteDatabase: execSQL(): | 31549 |
| 3 | android.content.Intent: getAction(): | 27054 |
| 4 | org.apache.http.client.HttpClient: execute(): | 24474 |
| 5 | android.telephony.TelephonyManager: getDeviceId(): | 20301 |
| 6 | android.content.Intent: setDataAndType(): | 18813 |
| 7 | java.net.URLConnection: connect(): | 17393 |
| 8 | android.content.ContentResolver: query(): | 16597 |
| 9 | android.os.Handler: obtainMessage(): | 15431 |

Hook模块提出一种风险评估算法，定义为调用关系，若在运行的过程中调用了，定义

若经过层调用了，则称

若无任何敏感API调用，则称

定义为该调用关系权重，则有

,

设为某APP调用的敏感API集，为该集大小，定义

为该应用的带权恶意系数，若，则判定该应用为恶意软件。

2) 若匹配到恶意行为特征API集：

定义为被调用的时间集，定义

为间的调用时间差。设为该App调用的固定组合API集，定义

为该App的恶意API组合时间恶意系数，若，则将该App评估为恶意App。

使用现有的App自动化检测系统得出在100款恶意软件上，该风险评估算法成功率为87%。

评估结束后Hook模块将App放入后续处理流程。

c) 风险预警

对于判定为恶意App的应用，API Hook模块提醒用户卸载，并将其包名、签名添加到黑名单中。Hook模块监听系统安装程序，对于黑名单中的应用直接予以卸载处理。

对于未判定为恶意但进入风险评估流程的App，本系统会在用户允许下将其上传到服务器，启动静态检测流程。

2.2.3 敏感行为监测系统总结

本系统的优势在于：

一、 对Xposed框架提供的SDK进行封装后，Hook代码容易管理，便于bug修复和系统更新。

二、Hook模块给每个用户应用生成完善、精确的敏感API调用日志，对于系统保护、0day漏洞发掘、研究提供可靠数据保障。

三、位于客户端的Hook模块代码托管于Xposed框架， Xposed框架对于系统内存、设备电量占用、消耗极少，并且保证了较高的恶意软件检测成功概率。

* 1. Android流量监控

2.3.1 Android流量监控方案简介

1) 基于代理服务器的监控方案

通过在Android设备指定端口上创建代理服务器，无线网环境下设置WIFI代理为本机(127.0.0.1：代理端口)，3G/4G环境下新建一个接入点APN，设置APN的代理和端口。Android系统会让所有流量都通过代理服务器，代理服务器即可对流量进行监控。代理服务器还可以安装证书，通过mitm中间人攻击的方式支持对https流量的解析。Android可用的代理工具有fiddler，browsermob-proxy等。

2) 基于VPNService的监控方案

VPNService是Android SDK中提供的开发Android VPN应用的API。Andriod开启VPN后，应用程序发出的数据包先发送到真实网络设备上，真实网络设备再通过iptables，使用NAT，将数据包转发到虚拟网络设备上。VPN应用打开虚拟网络设备，读取数据即可获得几乎所有的流量包。VPN应用可对获得的流量包进行拦截修改等操作，再发送给真实网络设备。Android一款名为packet capture的应用采用VPNService实现了流量监控。



图3 流量在VPN中的传递过程

3) 使用Android tcpdump

Android tcpdump是一个命令行窗口工具集，可以抓取WIFI、蜂窝网络等环境下的数据流量。tcpdump是Linux下一款强大的网络数据采集分析工具，而Android tcpdump则是tcpdump针对ARM架构处理器的编译版本。tcpdump通过libpacp来抓取报文，在Linux中libpacp使用AF\_PACK套接字实现对流量的抓取分析。Android设备必须取得root权限才能使用tcpdump。

4) 借助PC端软件

PC端具有功能强大的流量分析软件，如wireshark、fiddler、burpsuite、charles等，分析人员可采用这些软件间接抓取和分析Android设备的网络流量。首先PC先建立一个热点连接，只让待分析Android设备连接该热点。设置wireshark等流量分析软件监听这个热点即可间接分析Android设备的网络流量。

2.3.2优缺点分析

1) 基于代理服务器的方案中可用到的fiddler提供了可供开发者二次开发的SDK，而browsermob-proxy源代码开源，并且Android设备仅需设置代理IP和端口即可，因此实现较为容易。局限性是代理服务器只能在应用层上进行流量监控和拦截修改，无法在传输层上进行，因此不能区分开不同Android应用的流量。

2) VPNService方案可获得Android设备中网络层的IP报文，因此可对不同Android应用进行流量监控和拦截修改，功能更为强大。局限性是目前没有稳定优质的Android开源VPN项目，实现需要大量底层网络通信代码的编写。

3) Android tcpdump需要root权限，取得root权限后Android设备会面临更多安全问题，而且root Android设备的过程也充满了风险，对普通用户不友好。此外，tcpdump无法拦截修改报文，只能监听报文，无法满足拦截恶意流量的要求。

4) 建立热点借助PC端软件的方案是分析人员临时分析Android流量时最方便的方法。若Android设备上暂无流量分析软件，那么此方案最为有效。若使用fiddler，可在应用层监控、拦截、修改报文；若使用wireshark，可监控链路层以上的所有报文，但不能拦截修改报文。但是此方案仅限于分析人员临时分析流量采用。

综上所述，为了满足监控和拦截恶意应用流量的需要，并尽量对用户友好，本文采用代理服务器和VPNService两种方案在Android设备上搭建流量监控、分析、拦截平台。

2.3.3 Android流量监控、分析、拦截框架的实现



图4 Android流量监控、分析、拦截平台架构图

如图4所示，Android流量监控、分析、拦截平台采用C-S架构，在客户端上使用代理服务器和VPN相结合的方式控制流量，服务器端负责分析客户端上传的流量样本并更新恶意URL模式数据库。客户端和服务器的数据库定期保持更新。

Android客户端中流量分析模块采用VPNService方案。VPNService能获得网络层IP报文，拿到传输层TCP、UDP报文中的端口信息。Android客户端因此能够识别此报文是哪一个应用程序发出的，从而把不同应用程序的流量区分开，利于对特定应用的流量分析。客户端第一次启动时，请求用户输入密码以安装该客户端的数字证书，此证书主要用于代理服务器和VPN通过mitm的方式对https流量的解析。用户在使用过程中反馈具有恶意行为的应用，触发客户端在每次可疑应用运行时开启流量监控功能，抓取这些可疑应用运行期间的流量包，并在用户同意的情况下封装为pacp格式发送至服务器进行分析。

Android客户端中流量拦截模块采用代理服务器方案。代理服务器采用browsermob-proxy。browsermob-proxy开源项目在效率和稳定性上有较为可靠的保证，满足流量拦截模块的需求。为了保持代理服务器长期正常运行不被系统回收，把代理服务器写入Andriod系统前台服务中并在通知栏里显示运行状态。代理服务器运行在本机IP为127.0.0.1的一个特定端口上，需要用户在Andriod网络设置中手动设置代理服务器的IP和端口号，设置完毕那么Andriod所有流量都会经过代理服务器。客户端中维护一个使用轻量级数据库sqlite管理的恶意URL模式表。代理服务器根据这个模式表检查每一个Android应用的请求报文，如果请求报文的URL成功匹配了模式表中的某一项，则将此请求报文拦截，立即向发出该请求报文的应用返回一个内容为空的回应报文，从而阻断恶意应用的流量。如果请求报文URL没有匹配模式表中任何一项，则放行该请求报文至互联网。

服务器中对客户端上传的可疑流量进行分析。客户端上传至服务器的可疑流量需要以下信息：可疑应用的包名、客户端类型、监控起止时间、可疑流量pacp包。服务器端流量分析人员根据包名对应可疑应用程序，使用wireshark等软件复现pacp格式的流量，从中提取恶意流量的URL模式，添加至服务器端恶意URL模式数据库。服务器与客户端的数据库依据版本号，定期更新，保持同步。随着服务器对恶意URL模式数据库的不断完善，逐渐增强对恶意流量拦截的广度和准确度。

3 静态检测模块

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：

*X*与*Y*表示两个变量，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包括朴素贝叶斯(Native Bayes)与随机森林(Random Forest)算法[13]。

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：



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。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，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。

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下面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 |

4 工作流程样例

5 结束语

**参考文献：**

[1]于航.基于模拟器的沙箱系统研究[A]. 中国计算机学会.第30次全国计算机安全学术交流会论文集[C].中国计算机学会:,2015:5.

[2]任伟,柳坤,周金.An Da:恶意代码动态分析系统[J].信息网络安全,2014(8):28-33.

[3]蔡林,陈铁明.Android移动恶意代码检测的研究概述与展望[J].信息网络安全,2016(09):218-222.

[4]华保健,周艾亭,朱洪军.Android内核钩子的混合检测技术[J].计算机应用,2014,34(11):3336-3339+3343.

[5]Arni Einarsson，Janus Dam Nielsen.A Survivor’s Guide to Java Program Analysis with Soot

[6]王旭.基于控制流分析和数据流分析的Java程序静态检测方法的研究

[7]韩心慧，丁怡婧，王东祺，黎桐辛，叶志远.Android恶意广告威胁分析与检测技术

[8] Eric Bodden. Inter-procedural Data-flow Analysis with IFDS/IDE and Soot

[9] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel等. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps

[10]周裕娟,张红梅,张向利,李鹏飞.基于Android权限信息的恶意软件检测[J].计算机应用研究,2015,32(10):3036-3040.

[11]卜义云.基于机器学习的Android恶意软件静态检测系统的设计与实现[D].电子科技大学,2016.49-53.

[12]邵舒迪,虞慧群,范贵生.基于权限和API特征结合的Android恶意软件检测方法[J].计算机科

学,2017,44(04):135-139.

[13]张家旺.基于机器学习算法的Android恶意程序检测系统.计算机应用研究.2016(6).1776-1782

[14]朱月俊,文爽,李剑.改进随机森林在Android恶意检测中的应用[J].信息安全研究,2017,3(11):1020-1027.

[15]杨宏宇,徐晋.Android恶意软件静态检测模型[J].吉林大学学报(工学版),2018,48(02):564-570.