**2018年全国大学生信息安全竞赛**

**作品报告**

**作品名称：基于动静态结合分析的Android应用安全性多重检测系统**

**电子邮箱：**

**提交日期：**

填写说明

1. 所有参赛项目必须为一个基本完整的设计。作品报告书旨在能够清晰准确地阐述（或图示）该参赛队的参赛项目（或方案）。

2. 作品报告采用A4纸撰写。除标题外，所有内容必需为宋体、小四号字、1.5倍行距。

3. 作品报告中各项目说明文字部分仅供参考，作品报告书撰写完毕后，请删除所有说明文字。(本页不删除)

4. 作品报告模板里已经列的内容仅供参考，作者可以在此基础上增加内容或对文档结构进行微调。

5. 为保证网评的公平、公正，作品报告中应避免出现作者所在学校、院系和指导教师等泄露身份的信息。

# 

# 目 录

[目 录 1](#_Toc515576787)

[摘 要 3](#_Toc515576788)

[第一章 作品概述 5](#_Toc515576789)

[1.1 背景概述 5](#_Toc515576790)

[1.2 作品意义和特点 6](#_Toc515576791)

[第二章 作品总体设计 7](#_Toc515576792)

[2.1 设计目标 7](#_Toc515576793)

[2.1 系统总体功能 7](#_Toc515576794)

[2.2 系统工作流程 8](#_Toc515576795)

[第三章 系统各模块详细设计与实现 10](#_Toc515576796)

[3.1 基于Android API Hook的敏感行为监控模块 10](#_Toc515576797)

[3.1.1 Hook技术简介 10](#_Toc515576798)

[3.1.2 敏感行为监控模块的设计 12](#_Toc515576799)

[3.2 Android流量监控、分析、拦截模块 17](#_Toc515576800)

[3.2.1 Android流量监控方法 17](#_Toc515576801)

[3.2.2 流量监控方法的优缺点分析 20](#_Toc515576802)

[3.2.3 设计目标和功能 20](#_Toc515576803)

[3.2.4 Android流量监控、分析、拦截平台的实现 21](#_Toc515576804)

[3.3 基于权限与敏感API的机器学习分类算法静态鉴别模块 25](#_Toc515576805)

[3.3.1特征选取 26](#_Toc515576806)

[3.3.2 机器学习分类算法 28](#_Toc515576807)

[3.3.3 实现思路与流程 29](#_Toc515576808)

[3.3.4 模型评价指标 31](#_Toc515576809)

[3.3.5 模型分析与评估：选取合适的机器学习算法 32](#_Toc515576810)

[3.3.6 继续分析样本容量对分类性能的影响 33](#_Toc515576811)

[3.4 基于数据流分析技术的应用漏洞检测模块 34](#_Toc515576812)

[3.4.1 Soot的基本介绍 35](#_Toc515576813)

[3.4.2 数据流分析技术及Soot提供的过程内数据流分析框架 37](#_Toc515576814)

[3.4.3 基于过程内数据流分析技术的对象行为记录算法 38](#_Toc515576815)

[3.4.4 漏洞检测模块的工作流程 40](#_Toc515576816)

[3.4.5 举例：WebView远程代码执行漏洞[16]的检测 41](#_Toc515576817)

[3.5 隐私泄露检测模块 42](#_Toc515576818)

[3.5.1 静态污点分析算法与Sources点、Sinks点的分类 43](#_Toc515576819)

[3.5.2 对Android应用的生命周期建模与IFDS问题求解 44](#_Toc515576820)

[3.5.3 使用FlowDroid进行隐私泄露行为检测的流程 46](#_Toc515576821)

[第四章 作品测试与分析 47](#_Toc515576822)

[4.1测试大纲 47](#_Toc515576823)

[4.2 功能测试 48](#_Toc515576824)

[4.2.1敏感行为监控模块功能测试 48](#_Toc515576825)

[4.2.2流量监控模块功能测试 50](#_Toc515576826)

[4.2.3漏洞检测模块功能测试 52](#_Toc515576827)

[4.2.4隐私泄露检测模块功能测试 53](#_Toc515576828)

[4.2.5服务端完整静态检测流程测试 55](#_Toc515576829)

[4.3 性能测试 58](#_Toc515576830)

[4.3.1 流量监控模块性能测试 59](#_Toc515576831)

[4.3.2 恶意软件鉴别模块准确率测试 60](#_Toc515576832)

[4.3.3 漏洞检测模块性能测试 61](#_Toc515576833)

[4.3.4 隐私泄露检测模块性能测试 61](#_Toc515576834)

[第五章 总结与展望 63](#_Toc515576835)

[5.1 工作总结 63](#_Toc515576836)

[5.2 创新点 64](#_Toc515576837)

[5.3 未来展望 64](#_Toc515576838)

[参考文献 66](#_Toc515576839)

# 

# 摘 要

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、有效的安全防护。

在此背景下，我们学习了现有的检测技术，在加以改进后进行综合应用，最终设计并实现了一种综合多种检测技术的Android应用安全性检测系统。通过不同检测方案间的互补性来设计检测流程，扩展检测的评估面，同时部分检测模块改进后的检测方案可以有效提高检测精度和性能。本系统具体工作包括：

（1）在Android设备上设计并实现动态监控相关的两个模块。动态监控通过捕获Android应用在运行过程中的系统调用特征和网络流量特征做判断，第一个模块基于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，另一个模块则基于代理和VPN设计实现了应用流量的监控。通过这两个动态监控模块，可以在恶意行为及恶意流量两方面对运行中应用实现实时预警，同时可以弥补由于加壳等原因导静态分析无法进行的问题

（2）在服务器端设计并实现静态检测相关的三个模块。第一部分通过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，第二方面辅助以数据流分析技术实现高精度的漏洞扫描，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。通过静态检测部分的三个模块，可以有效检测出恶意代码攻击，漏洞代码利用，隐私泄露三种问题，扩展应用安全性的评估面。

本文设计实现的动静态结合的Android应用安全性检测系统，经过测试，能够较好的处理应用在运行过程中可能存在的恶意流量、恶意行为、漏洞利用，隐私窃取等一系列威胁用户数据安全，降低软件安全性的行为，可为root手机的用户提供较为完善的应用安全保障，保护用户的信息安全。

# 第一章 作品概述

## 1.1 背景概述

Google在2007年11月发布Android系统以来，凭借着Android系统的开源性和高度的开放性，Android迅速占领移动终端市场。2016 年 IDC 发布最新数据显示，Android 全球市场份额超过 80%，并预计 2019 年 Android 市场份额将达到 82.6%，占据绝对优势[1]。另外，据互联网监测公司StatCounter的研究显示，在全球网络用户中，从移动设备访问互联网的用户数量在2016年11月首次超过桌面端用户。来自手机和平板电脑的网络流量已经合计达51.2%，超过了桌面端的48.7%。这种趋势已经不可逆转，促使越来越多的厂商向移动应用产业发展。随着移动应用行业的快速发展，在巨大经济利益的诱惑下，灰产和黑产纷纷盯上这块大蛋糕，导致各种恶意应用层出不穷。大致看来，Android应用有如下几大类影响软件安全性的问题：

1）应用自身含有恶意行为影响用户的信息安全。Android恶意应用通过推送恶意广告诱骗欺诈、私自下载其他应用偷跑流量、私发短信恶意吸费、出卖用户隐私等方式非法获取利益。更有甚者，在应用安装后下载木马后门程序，让用户的Android手机沦为“肉鸡”。产生这种现状的原因，一方面来源于国内各种Android应用市场宽松的审查机制，另一方面则源于Android系统自身的问题，正如Sundar Pichai 曾在MWC 大会上所说——“android 不是为了安全而设计的，他是为了开放而设计的”，正是这种开放的设计导致了Android平台恶意应用的肆虐。

2）应用自身编写存在问题而被恶意利用。由于应用开发者的水平参差不齐以及应用测试的不完善，导致了很多应用自身都存在安全问题，容易被攻击者恶意利用从而对用户产生威胁。要防护这类问题，还要从应用的安全开发做起。

3）应用过度的隐私收集。为了提高用户体验，或是提高广告投放的精准度，大多数应用都尽可能多的收集用户隐私，但作为移动设备普及的当下，用户的手机中存储着大量个人隐私，一旦应用过度收集用户个人数据，甚至是恶意窃取，就造成了隐私泄露问题。

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

## 1.2 作品意义和特点

与PC端相对成熟的安全防护相比，移动安全领域还处于劣势，因此研究对Android应用安全性的研究显得十分必要。目前Android应用的检测方法主要分为动态监控和静态检测[2]，这两种技术各有优缺点。静态检测涉及到二进制相关的技术，其中包括反编译、逆向分析、模式匹配和静态系统调用分析等，静态检测技术有一个共同的特点：应用程序不被执行，它的优点是可以实现对应用程序所有功能的完整检查，但缺点同样也很明显，首先软件加壳，代码混淆，动态加载等技术可以有效防止应用被静态分析，无法准确获得应用内部代码的话，静态分析就无从进行，另一方面静态分析大多基于特征匹配，难以胜任未知行为的检测工作，因此需要另一种检测技术——动态监控的配合应用。动态监控的核心过程是在应用运行过程中动态监控应用的行为，如文件权限改变、进程和线程运行情况、系统调用情况、网络访问情况等，基于这些动态运行时收集的特征来进行恶意行为的识别，可以有效应对动态加载，加壳等应用防护方案，相应的，其缺点则是无法保证对完整应用行为的监控，只要还有行为没有在用户手机上运行，就无法保证是安全的。综合看来，动静态检测各有其优缺点，在实际中应结合使用。在本作品中，通过客户端运行的动态监控，以及服务端的静态检测来结合这两种技术，提供更为完善的安全防护。

另一方面，本作品中使用的检测手段都是已经推广使用开来并被证实有效的，但这些检测技术或多或少都还存在一些改进的空间，因此本作品的另一个着重点则是对作品中使用的检测技术进行改进，以达到更高的精确度与检出率。在敏感行为监控模块里，我们设计了风险评估算法，以识别应用运行过程中可能存在的敏感行为并进行实时预警；在恶意软件鉴别模块中，我们选取敏感API作为特征向量，并在此基础上进行微调，以提高恶意应用的检出率；漏洞检测模块中则结合数据流分析技术进行，使用过程内数据流分析技术设计了函数内对象行为记录算法来提高漏洞检测率。

# 第二章 作品总体设计

## 2.1 设计目标

本小组研究现有的检测技术，针对现有技术的不足，提出改进措施，实现对Android恶意应用的精准高效检测。具体设计目标有：

（1）通过Hook技术监控软件运行时调用的敏感API，并通过风险评估算法判断软件是否发生恶意行为。

（2）监控Android应用的网络通信，建立规则拦截恶意流量，向用户提供观察管理流量的工具；

（3）通过机器学习算法训练大量恶意样本，实现对恶意应用的基本鉴别；

（4）检测出应用中可能存在被恶意利用的代码漏洞，以及隐私泄露行为，并将检测报告反馈给用户。

（5）界面设计友好，操作简便，采用模块化设计，便于后期系统功能完善。

## 2.1 系统总体功能

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。图2-1为本系统的各模块和它们工作组成部分。动态监测部分包括敏感行为监控模块和流量监控模块，静态检测部分包括恶意软件鉴别模块、漏洞检测模块和隐私泄露检测模块。

敏感行为检测模块包括敏感API钩取模块、风险评估算法和高危预警模块；流量监控模块包括流量抓包模块、流量控制模块和规则同步模块；恶意软件鉴别模块包括权限与敏感API特征提取模块、随机森林算法分析模块、结果分析评估模块；漏洞检测模块包括soot反编译与控制流图构造模块、特征函数参数常量化追溯模块和漏洞特征匹配模块；隐私泄露检测模块包括虚拟函数构造模块、Sink点Source点扫描模块和污点传播路径扫描模块。

图2-1为系统功能模块组成图，图2-2为本系统工作流程图。



图2-1 系统功能模块组成

## 2.2 系统工作流程



图2-2 本系统工作流程图。

本系统客户端启动后，动态监测模块即开始工作，在后台监控所有用户选择的应用API调用和流量使用状况，若发现某应用恶意行为，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。

动态监测部分运行在Android客户端，在用户运行第三方程序时为其监控应用行为，并向用户示警恶意行为。动态监测部分主要划分为两个模块，分别从两个不同方面进行Android第三方应用行为监控。敏感行为监控模块基于API Hook对Android应用敏感API调用进行钩取、分析，并设计风险评估算法对应用风险进行评估；恶意流量监控模块使用代理接管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。

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。

# 第三章 系统各模块详细设计与实现

## 3.1 基于Android API Hook的敏感行为监控模块

### 3.1.1 Hook技术简介

**1). Android系统的进程沙箱机制**

Android顺其自然地继承了Linux内核的安全机制，同时结合移动终端的具体应用特点，进行了许多有益的改进与提升。

Windows与Unix/Linux等传统操作系统以用户为中心，假设用户之间是不可信的，更多考虑如何隔离不同用户对资源（存储区域与用户文件，内存区域与用户进程，底层设备等）的访问。

Android系统则是假设应用软件之间是不可信的，甚至用户自行安装的应用程序也是不可信的，因此，首先需要限制应用程序的功能，也就是将应用程序置于“沙箱”[3]之内，实现应用程序之间的隔离，并且设定允许或拒绝API调用的权限，控制应用程序对资源的访问，如访问文件，目录，网络，传感器等。

应用程序进程之间，应用程序与操作系统之间的安全性由Linux操作系统的标准进程级安全机制实现。在默认状态下，如图 所示应用程序之间无法交互，运行在进程沙箱内的应用程序没有被分配权限，无法访问系统或资源。因此，无论是直接运行于操作系统之上的应用程序，还是运行于Dalvik虚拟机的应用程序都得到同样的安全隔离与保护，被限制在各自“沙箱”内的应用程序互不干扰，对系统与其他应用程序的损害可降至最低。图3-1 为Android进程沙箱机制的原理图。

C:\Users\qyzhang_nj\Desktop\Ads GG\设计报告\沙箱.tif

图3-1 Android进程沙箱机制的原理

**2). Android Hook技术**

由于Android进程沙箱机制的存在，Android设备上的安全软件极难获取到其他软件的资源调度、事件触发，这使得对Android应用进行动态监控十分困难，而Hook技术[4]的出现很好地解决了这个问题。

Hook（钩子）是一个很形象的词语，这种技术使得Hook程序可以 “钩住”系统或第三方软件需要用到的资源，如消息或者API，并把自身代码融入到这些资源的调度中去，从而实现对系统或第三方应用的运行间跨沙箱劫持。

本系统通过对XposedBridge API的利用及再封装实现了Android API Hook模块开发，并通过Xposed框架在手机上实现Hook模块的运行。

**3). Xposed框架及其工作原理**

Xposed框架是一款可以在不修改APK的情况下影响程序运行(修改系统)的框架服务，基于它可以制作出许多功能强大的模块，且在功能不冲突的情况下同时运作。

Xposed框架通过替换/system/bin/app\_process程序控制zygote进程，使得app\_process在启动过程中会加载XposedBridge.jar，从而完成对Zygote进程及其创建的Dalvik虚拟机的劫持[5]。

在Android系统中，应用程序进程都是由Zygote进程孵化出来的，而Zygote进程是由Init进程启动的。Zygote进程在启动时会创建一个Dalvik虚拟机实例，每当它孵化一个新的应用程序进程时，都会将这个Dalvik虚拟机实例复制到新的应用程序进程里面去，从而使得每一个应用程序进程都有一个独立的Dalvik虚拟机实例。这也是Xposed选择替换app\_process的原因。

Zygote进程在启动的过程中，创建一个Dalvik虚拟机实例同时，还会将Java Runtime库以及一些Android核心类的JNI方法到前面创建的Dalvik虚拟机实例中去。值得注意的是，一个应用程序进程被Zygote进程孵化出来的时候，不仅会获得Zygote进程中的Dalvik虚拟机实例拷贝，还会与Zygote一起共享Java Runtime库。这也就是可以将XposedBridge.jar加载到每一个Android应用程序中的原因。

C:\Users\qyzhang_nj\Desktop\Ads GG\设计报告\Xposed原理.tif

图3-2 Xposed框架的工作原理

如图3-2所示，Xposed在开机的时候完成对所有的Hook Function的劫持，在原API执行的前后加上自定义代码。原API执行前执行的代码写在beforeHookedMethod方法中，该方法包含于XposedBridge.jar中，用户可以对该方法进行定制，同样的API执行后执行的代码写在afterHookedMethod方法中。

### 3.1.2 敏感行为监控模块的设计

**1). 研究背景**

Hook是一个实用而强大的工具，目前常见的Hook程序大多是为了实现系统个性化、应用功能定制的功能，但是用它做安全分析的程序却少之又少。为数不多的带有安全分析功能的Hook程序却只生成应用日志，只有专业的安全分析人员能阅读，对于普通的手机用户来说只是鸡肋。

本系统研究了部分带有安全分析功能的Hook程序，发现它们有以下几个缺点：

①. Hook全部应用，不让用户选择，生成大量日志造成系统负担。

②. 没有示警功能，只有呆板的日志记录。

③. 没有风险评估算法，仅仅在风险调用达到一定数量时判定应用为恶意应用，存在大量误判。

④. 判定为恶意应用后没有后续流程，不对恶意软件中的特征进行处理，对于误判没有解决方案。

基于此背景，敏感行为监控模块的设计关注了以上几点问题，并较好地解决了它们。下面模块执行流程、白名单和选择监控功能、Hook代码设计、风险评估算法、示警流程介绍该模块。

**2). 敏感行为监控模块的运行流程**



图3-3 敏感行为监控模块工作流程

如图3-3所示，敏感行为监测模块总的Hook代码随Android客户端安装后托管于Xposed框架，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，所以该模块为用户提供选择要监控的应用功能，用户选择好要监控的应用后，重启手机，Hook代码被注入相应应用的API调用中，模块正式开始工作。

Hook代码每钩取到一个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。API Hook模块只记录固定时间内的API调用，若超过该固定时间，API调用链中过早调用的API会被删去。若该应用在固定时间内程序调用链长度达到阈值模块就会启动恶意风险评估流程。

下面先介绍模块预设白名单监控应用选择功能。

**3). 预设白名单**

每个Android应用都会有其自己的签名，同一个厂商、同一款应用的不同版本之间的签名都是一样的。

签名过程需要数据摘要、签名文件、证书、密钥库的参与，只有掌握了私钥、拥有有效证书的厂商才能提供正确的签名。客户只能通过公钥和对原始数据的Hash判别签名的正确性。

所以，如果可以验证一个应用的包名和其签名与已知其安全版本是一致的，就能说明这个应用是安全应用。

模块设计之初，项目组建立了一个长度为125的<安全应用包名-签名>的键值对，即白名单。在用户选择要Hook的App时，应用列表中会注明哪些应用在白名单中，可以放心使用。这不仅防止了用户对所有应用进行监控而带来的系统负担，更防止了大型应用运行时复杂的操作带来的误判。

用户选择监控应用界面时，可以看到白名单中的应用已被标识了“安全”字样，用户只需选择标有“未知”字样的应用进行监控即可。应用选择后，重启系统，Hook代码开始运行。

**4). 对XposedBridge.jar中的API再封装和恶意应用“自首”功能设计**

由于敏感行为监控模块需要Hook大量的API，同时对于每个不同的API都有不同的分析、处理方式，所以本系统对XposedBridge中的API进行了再封装。

以下代码会在设备开机时由Xposed框架注入Android系统。

// 代码写在继承与Hookers类的initAllHooks方法中，便于格式化代码的压缩与管理  
**Method** execMethodWithString = **Reflector**.**findMethod**(**clazz**, **methodName**, **paramClazz**);

// 封装Reflector类用于反射查询功能方法、构造器、成员变量

// MethodHookHandler类的methodHookImpl定义在父类Hookers中，MethodHookHandler类实现了hookMethod和hookAllConstructors方法，方法中分别调用了XposedBridge.hookMethod(method, callback)和XposedBridge.hookAllConstructors(clazz, callBack)。  
methodHookImpl.hookMethod(execMethodWithString, **new MethodHookCallBack**(*packageParam*) {

// MethodHookCallBack回调类继承于XC\_MethodHook抽象类，重载其中重要方法并添加新抽象方法  
 **@Override** // 重载beforeHookedMethod方法  
 **protected void** beforeHookedMethod(**MethodHookParam** *param*) **throws Throwable**   
 {**super**.beforeHookedMethod(*param*); }  
 **@Override** // 重载afterHookedMethod方法，方法中 **protected void** afterHookedMethod(**MethodHookParam** *param*) **throws Throwable**   
 {**super**.afterHookedMethod(*param*); }  
 **@Override** // 新增recordAPICalling方法，用于在日志中记录方法的调用  
 **protected void** recoredAPICallingi(**MethodHookParam** *param*) **throws Exception**   
 {

**super**.recoredAPICalling(*param*);

//Xposed注入的代码的执行发生在目标App进程中，而不是Xposed进程或是本系统的进程中，所以在以下函数中实现了当目标进程恶意值达到峰值，向本系统程序发出广播告知自己发生了恶意行为，实现了恶意应用的“自首”

processWarning(hookMonitorCallbackHandler(packageName, action);

}  
 **@Override** // 新增recordAPIReturn方法，用于记录方法的返回内容 **protected void** recoredAPIReturn(**MethodHookParam** *param*) **throws Exception** {  
 **super**.recoredAPIReturn(*param*);  
 logger.openLogFile(**packageName**); // 打开按包名区分的日志文件  
 logger.logAPICalling(*param*); // 记录API调用信息  
 logger.logParam(**"logContent"**); // 记录自定义内容，如返回内容、重要参 数、行为特征  
 }  
});

模块会将每个被Hook的应用的风险API调用时间与其重要细节如参数或返回值分应用记录，以备安全分析使用。

**5). 风险评估流程**

风险评估流程的设计基于自创的算法，算法的设计基于API调用链。

①. API调用链的概念

提出API调用链的概念。API调用链一种自定义的数据结构，该数据结构中包含一个LinkedHashMap成员变量，用于有顺序地记录调用时间与调用API的对应关系，一个记录敏感行为发生次数的数组成员变量，一个风险值成员变量。

API调用链可以准确地记录某App一定时间内调用的API顺序和次数和它们对应的风险行为，便于带权风险值的计算和日志记录。当App运行时间超过1分钟，该链会自动剔除过早的调用记录。

风险评估流程就是基于API调用链设计。

②. 计算带权恶意值

计算带权恶意值为风险评估流程的关键，带权恶意值的大小直接判定被Hook的软件是否为恶意软件。以下是计算带权恶意值的方案。

首先，本系统人为地给所有监控的API分了类，并给每个风险API设定了风险权值。如下表所示

表3-1 风险API与其权值

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 类别 | 描述 | 风险权值 |
| 1 | 高危行为、漏洞利用 | Shell操作、JS注入操作、root系统、root shell | 0.08-0.1 |
| 2 | 高危隐私获取行为 | 获取浏览器访问URL、浏览器书签、手机号、联系人通话信息、设备ID、网络ID、IMSI码、设备位置 | 0.05-0.07 |
| 3 | 常见隐私获取行为 | 获取账户信息、获取Mac地址、读取信息、接收信息、蓝牙操作、 | 0.02-0.04 |
| 4 | 可疑常见行为 | 注册ContentResolver、数据库操作、下载行为、加解密操作、终止广播、SQLite操作、使用反射 | 0.001-0.002 |
| 5 | 大量发生的常见行为 | http连接、https连接、Intent操作 | 0.00 只记录，不算入恶意值 |

本系统预设敏感行为相关API集为

|  |  |  |
| --- | --- | --- |
|  |  | (3-1) |

其中为某一API。

定义

|  |  |  |
| --- | --- | --- |
|  |  | (3-2) |

为某App调用的敏感API集，包含于。

定义

|  |  |  |
| --- | --- | --- |
|  |  | (3-3) |

为该App中API 执行过程中调用的敏感API集。

定义

|  |  |  |
| --- | --- | --- |
|  |  | (3-4) |

为该API的调用权重，特别地若，则。

定义

|  |  |  |
| --- | --- | --- |
|  |  | (3-5) |

为该App的带权恶意值。

通过大量测试，设定当时，判定该App为恶意软件，模块进入示警流程。

**6). 示警流程**

当某App被判定为恶意应用后，该App“自首”广播发出的同时，还附带其运行期间发生的所有恶意行为，本系统客户端接收到广播后会发送浮动通知，告知用户正在运行恶意应用并关闭应用进程，用户根据通知提示操作可以看到该应用的监控报告，其中包括发生的恶意行为和通过对该应用日志扫描得到的可疑网址，用户可以选择卸载该应用、禁用这些网址（通过流量监控模块）、将应用上传到本系统服务器端进行恶意应用鉴别、代码漏洞分析和隐私泄露分析或选择将该应用添加到白名单，模块将不再对该应用进行Hook。

示警流程为用户提供了选择，禁用网址操作可以防止类似恶意应用的部分恶意行为，由于服务器端的静态检测效果较好，上传检测基本做到对误判的防范。

## 3.2 Android流量监控、分析、拦截模块

Android恶意应用的获利方式几乎不能离开移动互联网，如果切断恶意应用的恶意流量，就可以阻止恶意应用进一步的恶意行为，从而保护用户的流量和隐私。

### 3.2.1 Android流量监控方法

拦截恶意流量的前提是获得Android网络通信流量。目前Android开发者通过下面几种方法获得经过Android设备的网络报文。

**1). 使用代理服务器**

开发者在Android设备中运行代理服务器程序，绑定特定端口，设置当前网络环境的代理地址为127.0.0.1，端口为代理程序的绑定端口。一般情况下Android手机的网络环境为无线WLAN或蜂窝移动网络。无线WLAN环境下进入系统无线设置，填写代理地址和端口，保存即可。蜂窝移动网络环境下需要新建一个接入点APN，修改APN的代理和端口，选择这个新建的APN连接。正确设置网络环境的代理后，Android系统将整个系统流量交给代理服务器，开发者在代理服务器中就可以得到经过Android系统的网络报文。



图3-4 Android代理服务器

常用的代理服务器有FiddlerCore、browsermob-proxy等。FiddlerCore是fiddler流量分析软件的嵌入式集成模块。开发者在自己的应用中集成FiddlerCore ，使用它提供的API即可获得fiddler的功能，如抓包过滤等。browsermob-proxy是一个开源的代理工具，提供了一套接口实现对网络流量的控制。这两种代理服务器都能够在用户信任代理服务器证书的前提下通过中间人攻击的方式解析Https流量，获得加密通信的内容。

**2). 基于VPNService编写VPN程序**

Google在Android API level 14中引入了VPNService，为开发者提供了开发Android VPN程序的框架和接口。Android开启VPN后，VPN程序可以获得几乎所有网络流量。对于一个向外发送的数据包，一般会在Android VPN服务框架中经历以下4个转发过程。进入Android设备的数据包经历的过程与之类似。



图3-5 数据包在VPN中的转发过程

①. 应用程序使用socket，将数据包发送到真实网络设备上。

②. Android系统通过iptables，使用NAT，将所有的数据包转发到TUN虚拟网络设备上去，端口是tun0。

③. VPN程序通过打开/dev/tun设备，并读取该设备上的数据，可以获得所有转发到tun虚拟网络设备上的IP包。因为设备上的所有IP包都被NAT转成原地址是tun0端口发送，所以VPN程序可以获得进出该设备的几乎所有的数据，只有本地回环数据等少数流量无法获得。

④. VPN程序可以对拿到的数据包做一些处理，然后将处理过后的数据包，通过真实网络设备发送出去。为了防止发送的数据包再被转到tun虚拟网络设备上，VPN程序所使用的socket必须先被明确绑定到真实的网络设备上去。

Android一款名为packet capture的应用使用VPNService实现了抓包功能，此外开发者助手中也实现了VPN抓包功能。

**3). 使用Android tcpdump**

Android tcpdump是一个命令行窗口工具集，可以抓取WLAN、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中的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). 改造Linux防火墙[6]**

Android系统基于Linux，可借助现有的Linux防火墙，使用iptables/Netfilter链钩子技术，劫持系统调用，在内核中获取数据包并隐藏劫持行为，实现Android内核层次的流量控制功能。

**5). 借助PC端软件**

PC端具有功能强大的流量分析软件，如wireshark、fiddler、charles等。一般方法是PC开启一个热点，待分析的Android设备连接这个热点，PC端流量分析软件监听这个热点即可获得Android网络通信流量。

### 3.2.2 流量监控方法的优缺点分析

基于代理服务器的方案中可用到的FiddlerCore提供了丰富的API接口，而browsermob-proxy是优质的开源项目，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，无法在传输层上进行，因此不能区分开不同Android应用的流量。

使用VPNService框架编写的VPN程序可获得Android设备中网络层的IP报文，因此可通过IP报文中传输层的端口信息区分出不同应用的流量并对流量进行控制，功能更为强大。局限性是VPN程序需要对IP数据包进行不同协议下的封包拆包操作，需要大量底层网络报文控制代码的编写，实现较为困难。

改造Linux防火墙，在内核级取得对Android流量控制。这种方式用户可以不进行任何操作如设置代理、开启VPN等，就可以获得所有流量。局限性是需要研究和修改系统源码，实现比较困难。

Android tcpdump命令行工具不适用于嵌入Android应用程序中。此外tcpdump只能抓取流量，无法拦截修改流量，因此tcpdump无法用于本模型。

建立热点借助PC端软件的方案是分析人员临时分析Android流量时最方便最有效的方法。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。局限性是这种方法只能间接分析使用。

### 3.2.3 设计目标和功能

本模块搭建一个Android流量监控、分析、拦截平台，实现以下4点功能。

①. Android设备上流量抓包，根据应用、进程分类显示。

②. Android流量拦截功能。建立URL模式规则库，根据规则库中的黑名单自动拦截恶意流量。

③. 获取hook模块输出的可疑URL，由用户授权决定是否进行拦截。

④. 用户自定义拦截规则，服务器维护共享规则、同步优质规则。

鉴于上述功能的需要，考虑现有Android流量监控方法的优缺点，对于流量抓包功能，向用户展示具体每个应用的流量是抓包的最终目的，所以流量抓包功能采用VPN方法实现；对于流量监控拦截功能，不必展示详细的信息，而browsermob-proxy开源项目除了提供较为完善的接口外，开发者还能修改源码实现更特殊的需求，所以流量监控拦截功能采用browsermob-proxy代理实现十分合适。

### 3.2.4 Android流量监控、分析、拦截平台的实现



图3-6 Android流量监控、分析、拦截平台架构图

Android流量监控、分析、拦截平台采用C-S模式，Android客户端实现流量的抓包和拦截，服务器实现共享规则的汇总、选优、同步。

**1). 客户端的设计**

Android客户端中流量分析模块采用VPNService实现。用户选择想要分析的应用，启动VPN程序，VPN程序启动完毕后，再启动想要分析的应用。在这之后待分析应用的所有流量都会被VPN程序获得并显示。对于VPN程序拿到每一个IP数据包，检查其中是否包含了传输层协议，如果是，则根据传输层中TCP或UDP中的端口信息将IP数据包与待分析应用绑定。



图3-7 IP数据包与特定应用绑定流程

Android中通过Runtime.getRuntime().exec()方法可以执行Linux的命令，从返回结果的Process中获得输出流，从输出流中获得Linux命令的执行结果。通过上述方法，执行ps命令获得进程的PID和NAME，执行cat/proc/net/tcp、cat/proc/net/udp命令获得进程PID和绑定的运输层端口。两个命令的输出通过PID联系起来，可将进程名NAME和进程端口对应，从而区分每个IP数据包到底是哪个应用程序的。

将IP数据包与应用程序对应后，解析IP数据包的传输层以及应用层，将数据包的基本信息如源和目的IP地址、端口、应用层协议等展示给用户。通常情况下应用传输的数据会使用Http或Https协议，抓包程序突出显示Http键值对和图片等信息，更加方便地让用户追踪应用行为的流量信息。

Android客户端中的流量拦截模块采用browsermob-proxy代理实现。browsermob-proxy是为ajax应用和selenium测试开发的代理服务器，由于browserm-proxy完全使用java编写，可方便移植到Android平台上运行。为了支持对Https流量的解析，修改代理的证书部分并安装代理服务器的证书，以支持Android平台上的MITM。

图3-8 MITM中间人攻击解析Https流量

Https协议在传输层之上增加了TLS/SSL层，对应用层明文信息进行加密，普通抓包方法只能得到加密后的密文。MITM中间人攻击方式作为本地Android应用和应用对应服务器的中间人，使用用户信任的代理证书，既充当本机应用的服务器又充当远端服务器的客户端。代理在这过程中可从两个SSL连接中解密得到明文，实现对Https的解密抓包。

客户端中使用SQLite存储恶意URL模式表。代理根据这个数据库匹配每个从Android设备中发出的请求报文URL，如果命中了规则，代理就将请求报文拦截并向应用返回内容为空的回应报文，从而切断远端服务器发来的恶意请求。如果请求报文没有命中任何规则，就将此请求报文放行至互联网。



图3-9 恶意URL正则模式表结构

由于代理必须检查所有从Android设备中发出去的请求报文，在网络负载较大的情况下，要求代理的拦截部分具有较高的效率，才能保证不明显影响网络通信速率。代理判断一个请求报文的URL是否为恶意的，必须要保证这个URL不能命中任何一个恶意规则库中的正则表达式，这相当于每一个请求都会对数据库进行一次遍历。影响效率的主要因素是数据库的访问比较耗时，为了平衡访问数据库的次数和运行内存的占用，采用了折衷的方法。



图3-10 代理拦截URL匹配流程图

每次访问数据库都取出X条记录，参数X需要根据数据库的规模动态调整。假设数据库规模较小，完全可以取出数据库中所有记录放入内存，匹配速度最快。随着数据库规模的增大，X可取数据库总记录数的二分之一、四分之一等等。这样可以减小内存的占用，控制访问数据库的次数，提高匹配效率。

当Hook模块判断当前用户使用的应用为恶意应用，Hook模块将恶意应用的运行日志传入流量拦截模块。拦截模块使用正则表达式提取Hook日志中恶意应用曾经向外发出的所有流量的URL，添加到恶意URL黑名单中，拦截标志全部设为true，代理默认全部拒绝放行。



图3-11 恶意URL黑名单结构

用户如果需要继续使用恶意应用的某些功能，选择信任该应用，拦截标志变为false，代理则放行流量。用户后续能够修改拦截标志字段限制恶意流量。

用户可使用自定义规则模块操作数据库，自由控制代理的拦截行为。Hook模块提取的日志和抓包模块的结果为用户提供数据流量信息，用户编写的自定义规则存放在本地恶意规则库中，结构与恶意URL模式表相同。在用户同意分享自定义规则的条件下，平台定期上传用户自定义规则到服务器。

**2). 服务器的设计**

服务器收集汇总客户端分享的用户自定义规则，存放到服务器端的共享恶意URL模式数据库中，向所有用户开放。用户可以尝试共享的规则，并对其进行评分评价。服务器定期对具有良好口碑的高评价规则添加到公共恶意URL模式数据库中，定期向所有客户端推送更新。随着用户共享的优质恶意URL规则的积累，逐步加强对恶意流量拦截的广度和深度。

## 3.3 基于权限与敏感API的机器学习分类算法静态鉴别模块

传统的Android恶意软件检测主要基于特征码匹配技术和启发性信息扫描技术。特征码通过匹配应用的签名和二进制代码，并依据特征库匹配判断是否为恶意应用，在分析和更新特征库的这段时间内，新型恶意应用很可能已经在网络流传，也就是说，特征码技术无法有效检测新型恶意应用。启发式扫描技术根据恶意应用的行为特征规律来检测未知恶意软件，然而这种方法的最大缺陷是只有当恶意应用造成危害时，系统才能返回相应的检测结果。

因此，机器学习恰好适用于在未知应用感染用户之前就能准确检测和判断Android恶意应用，本模块针对Android恶意应用的权限机制与敏感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Naive 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性能评估，提出了基于敏感API调用的随机森林算法分类模型，从而完成对恶意应用与正常应用的基本鉴别，并反馈用户鉴别结果。具体流程图如图3-12 所示

图3-12 机器学习检测模块流程图

### 3.3.1特征选取

（1）权限信息：

Android安全机制的核心是权限机制，应用程序的一系列行为都需要它所申请的权限[7]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。比如当应用程序申请了发送短信和读取短信权限，就可能在用户不知道的情况下获取用户的隐私。权限按照危险权限可以分为两种类型，一类是普通权限，一类是危险权限。普通权限指的是那些不会直接威胁到用户安全和隐私的权限，例如访问网络、设置系统时间等；危险权限则表示那些可能会触及用户隐私或者对设备安全性造成影响的权限，例如获取设备联系人信息、地理位置、读取SD卡等。

通过对Android应用程序的配置文件AndroidManifest.xml的分析，提取了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。这里使用皮尔逊相关系数[8]这种度量方法计算两种权限的相似度，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：

|  |  |  |
| --- | --- | --- |
|  |  | (3-6) |

*X*与*Y*表示两个权限，表示X和Y的相似度，它的值从-1到1，如果两个权限的大于0.6，说明这两个权限具有强相关性，那么就把这两个权限归为一个权限簇，表3-2是经过分类后得到的7个权限簇，其中列举了部分权限。

表3-2 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。

（2）敏感API

敏感API[9]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，其中部分API如表3-3所示：

表3-3 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。

### 3.3.2 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(Naïve Bayes)与随机森林(Random Forest)算法。

1. 朴素贝叶斯算法

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：

|  |  |  |
| --- | --- | --- |
|  |  | (3-7) |

其中，为是类“先验概率”，是类“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。对于本模型，为该样本是存在某一权限的概率，表示已确定样本类型的情况下，存在某一权限的概率，将与输入到(3-6)中，就能得到该应用为恶意应用的概率。

1. 随机森林算法

随机森林[10] [11]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。随机森林使用CART决策树作为弱学习器，在此基础上，对决策树的建立做了改进，对于普通的决策树，会在其节点上所有的样本特征中选择一个最优的特征来做决策树的左右子树划分，而随机森林通过随机选择节点上的一部分特征，再从这些特征中选取一个最优的特征作为决策树的左右子树划分，这样进一步增强了模型的泛化能力，避免过拟合问题。随机森林算法的执行过程如下：

1.从原始训练集中随机有放回采样选出m个样本，共进行n\_tree次采样，生成n\_tree个训练集

2.对于n\_tree个训练集，我们分别训练n\_tree个决策树模型

3.对于单个决策树模型，假设训练样本特征的个数为n，那么每次分裂时根据信息增益/信息增益比/基尼指数选择最好的特征进行分裂

4.每棵树都一直这样分裂下去，直到该节点的所有训练样例都属于同一类。在决策树的分裂过程中不需要剪枝

5.将生成的多棵决策树组成随机森林。对于分类问题，按多棵树分类器投票决定最终分类结果；对于回归问题，由多棵树预测值的均值决定最终预测结果

对于本模块，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。

### 3.3.3 实现思路与流程

（1）特征提取

权限特征：首先通过获取android应用配置文件AndroidManifest.xml文件中的权限信息，建立基于权限信息的集合，然后从数据库中获取所有高危权限集合，对所有高危权限进行遍历，如果权限在该应用程序已调用，就置为1，否则置为0。最后形成如下所示的特征向量：

[0,1,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0]

特征提取对应流程图如图3-13所示：



图3-13 特征值提取流程图

敏感API：通过反编译应用程序得到的中间代码整理出的敏感API以json格式存储，对json数据进行解析，获取敏感API的调用次数，建立基于敏感API的特征向量如下所示（这里每个API的调用次数都已经根据应用大小进行加权平均）：

[0.0, 0.0, 3.99, 0.89, 0.0, 11.97, 0.0, 3.55, 0.0, 0.0, 0.44, 0.0, 0.0, 0.44, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.44, 0.0, 0.0, 0.0, 0.0, 0.44, 0.0, 0.0, 0.0, 1.77, 0.0, 0.0, 0.89, 0.0, 0.0, 0.44, 0.0, 0.0, 4.88, 1.33, 0.0, 0.44, 0.0, 1.33, 2.66, 0.0, 0.0, 0.0, 0.89, 2.22, 0.0, 0.0]

（2）使用朴素贝叶斯算法与随机森林算法训练样本

sklearn是机器学习中一个常用的python第三方模块，其中对一些常用的机器学习方法进行了封装，在进行机器学习任务时，并不需要实现所有的算法，只需要简单的调用sklearn里的模块，就能实现大多数机器学习任务。

对于朴素贝叶斯分类器，建立模型关键代码如下所示**def** TextClassifier(train\_feature, test\_feature, train\_class, test\_class):  
 classifier = BernoulliNB(alpha=0.1).fit(train\_feature, train\_class)  
 test\_accuracy = classifier.score(test\_feature, test\_class)  
 **return** test\_accuracy *# 返回预测结果的精确度*

首先将训练集与其对应标签输入到分类器，朴素贝叶斯分类器有三个分类算法类，分别是高斯分布（GaussianNB）、多项式分布（MultinomialNB）和伯努利分布（BernoulliNB），由于权限信息的特征值只有0和1，所以使用BernoulliNB，而建立基于敏感API的分类器模型使用MultinomialNB。分类器的参数alpha表示拉普拉斯平滑，默认为1.0，是为了防止零概率的问题。然后通过分类器的fit()函数建立模型，此时可以调用score()函数，使用测试集进行测试，函数会返回测试的准确率。

随机森林算法同样类似，通过给定训练集与测试集可以建立模型并进行简单的测试：

**def** TextClassifier(train\_feature, test\_feature, train\_label, test\_label):  
 classifier = RandomForestClassifier(n\_estimators=100).fit(train\_feature, train\_label)  
 accuracy = classifier.score(test\_feature, test\_label)  
 **return** accuracy

其中随机森林分类器的参数n\_estimators表示决策树个数，一般与特征数相近，如果过大的话会影响训练效率，这里本作品取决策树个数为100个。

1. 存储训练模型并对测试样本进行预测

sklearn模块中包含对训练模型进行存储的函数，使用joblib.dump()函数可以将模型存储为pkl文件以便读取。

当对一个未知样本进行判断时，首先获取该位置样本的特征，然后读取模型文件并进行预测，具体代码如下所示：

*# 加载训练模型,并判断1个apk是否为恶意软件***def** judgeVirus(fileName):  
 fr = open(fileName)  
 line = fr.readline()  
 api\_dict = json.loads(line)  
 label = fr.readline()  
 dataList = list(api\_dict.values())  
 labelList = label.strip()  
 clf = joblib.load(**'train\_model\_byRandomForest\_API.pkl'**)  
 **return** clf.predict(np.mat(dataList))

如果返回值为“1”，说明该应用为恶意应用；返回值为“0”，说明该应用为正常应用。

### 3.3.4 模型评价指标

采用准确率(ACC)、精确率(Precision)、召回率(Recall)作为分类器的评估指标。ACC表示所有被正确分类应用占总应用的比例，Precision表示在预测为恶性的应用样本中，真正有恶性的样本所占的比例。即

|  |  |  |
| --- | --- | --- |
|  |  | (3-8) |

其中，*TP*表示恶意应用被检测正确（被检测为恶意应用）的数量，*FP*表示良性应用被检测错误的数量，*TN*表示良性应用被检测正确的数量，*FN*表示恶意应用被检测错误的数量。Recall是指所有真正的恶意样本中，预测为恶意样本所占的比例，即

|  |  |  |
| --- | --- | --- |
|  |  | (3-9) |

通常情况下，ACC只反应分类器的整体分类性能，不能反应不平衡数据集的分类性能。所以使用更具代表性的精确率(pre)与召回率(recall)进行模型评价[12],pre与recall越高，说明分类器性能越好。

在python中，sklearn库提供了评价指标的库函数，通过调用confusion\_matrix函数，返回一个包含tp、tn、fp、fn的矩阵，然后根据公式计算得出评价指标，具体实现方式如下所示：

**def** evaluate(real\_v, predict\_v):  
 AUC = metrics.roc\_auc\_score(real\_v, predict\_v)  
 threshold = 0.5  
 **for** i **in** range(0, len(predict\_v)):  
 **if** predict\_v[i] > threshold:  
 predict\_v[i] = 1  
 **else**:  
 predict\_v[i] = 0  
 confusion\_matrix = metrics.confusion\_matrix(real\_v, predict\_v)  
 tp = confusion\_matrix[0][0]  
 tn = confusion\_matrix[1][1]  
 fp = confusion\_matrix[1][0]  
 fn = confusion\_matrix[0][1]  
  
 acc = float(tp + tn) / (tp + tn + fp + fn)  
 recall = float(tp) / (tp + fn)  
 pre = float(tp) / (tp + fp)

### 3.3.5 模型分析与评估：选取合适的机器学习算法

取正常应用与恶意应用各1500个作为训练集，另各取250个作为测试集，其中恶意应用来自公开的恶意软件库VirusShare[13]，正常应用使用爬虫爬取自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。采用精确率(pre)与召回率(recall)作为衡量分类性能的指标，实验结果如表3-4所示：

表3-4 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3-5所示：

表3-5 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。

### 3.3.6 继续分析样本容量对分类性能的影响

我们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，关键代码图3-14下所示：

图3-14 部分关键代码

测试结果如图3-15所示:

图3-15 基于权限信息的样本容量对分类性能的影响

从图3-15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3-16 基于敏感API的样本容量对分类性能的影响

结合样本容量对检测准确度的影响曲线(图3-16)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。

## 3.4 基于数据流分析技术的应用漏洞检测模块

在3.3所述的恶意应用鉴别模块判别出是否为恶意应用后，紧接着运行的是服务端静态检测中的第二个模块——漏洞检测模块。Android应用中或多或少都存在有编写漏洞的代码，这些Android应用漏洞虽然可能并不含有恶意行为，无法直接被3.3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，本作品从静态分析角度入手，设计了Android应用漏洞检测模块。另一方面，现有的基于Java源码的静态检测技术在检测精度和运行效率上都仍有很大的提高空间，在漏洞检测模块中，我们利用Soot[14]提供的流分析技术[15]来进行漏洞检测，自行设计并实现了基于过程内数据流分析技术的对象行为记录算法，以实现函数内的变量参数到相关常量的追溯等功能，提高漏洞检测精度。

### 3.4.1 Soot的基本介绍

Soot是一款在Github上开源的Java代码分析框架，最初的功能只是为Java程序的分析优化提供一款方便实用的工具，之后Soot工作组在此基础上进行了延伸，增加了诸如程序间控制流图构建，过程内数据流分析框架，基于IFDS问题的数据流分析框架等功能，支持使用者在静态代码分析过程中结合流分析技术。

Soot提供的最基础的功能是将程序转化为一种易于分析的中间表示形式，并将程序语句和逻辑关系转化成一系列Soot定义的对象，使用者在分析中可以基于这些Soot定义的对象来结构化的对代码进行优化或分析。Soot提供了四种Java字节码的中间表示形式：

* Baf 一种易于操作的，精简的字节码表示形式
* Jimple 提供一种适合分析优化的三地址码表示形式
* Shimple Jimple中间代码的SSA变体
* Grimp 适用于反编译和代码分析的Jimple汇总版本

在作品中，我们使用Jimple类型用于漏洞分析，Jimple中间代码将复杂的Java字节码都转化为三地址指令，并且只有15种基本语句类型，分类如下所示：

表3-6 基本语句分类表

|  |  |
| --- | --- |
| 类型 | 语句类 |
| 核心语句 | NopStmt, IdentityStmt, AssignStmt |
| 有关过程内控制流的语句 | IfStmt Goto-Stmt,  TableSwitchStmt LookupSwitchStmt |
| 有关过程间控制流的语句 | InvokeStmt, ReturnStmt, Return-VoidStmt |
| 监控语句 | EnterMonitorStmt, ExitMonitorStmt |
| 其他 | ThrowStmt, RetStmt |

Jimple中间代码除了对不同语句类型进行了分类，还对语句内部进行了基本数据结构的填充，如类，方法，函数体，语句，表达式，变量等，按照常见的数据结构分类方式进行整理后如下表所示：

表3-7 常见结构分类方式表

|  |  |
| --- | --- |
| 数据类型 | 描述 |
| Scene | 用于提供反编译后完整的程序环境 |
| SootClass | Soot对反编译后Java类的表示 |
| SootField | Soot对Java类中成员变量的表示 |
| SootMethod | Soot对反编译后函数的表示 |
| Body | Soot对函数方法体的表示 |
| Local | Soot对函数内定义的局部变量的表示 |
| Unit / Stmt | 两种都可以用来表示一条Jimple语句 |
| Expr | Jimple语句中的表达式，如AddExpr |
| ValueBox | Jimple语言对语句中元素块的封装 |

一些常见类型之间的调用关系如下图所示：



图3-17 一些常见类型之间的关系

通过上述定义的基本数据类型，使用Soot可以结构化的对反编译后的Java代码进行分析。

### 3.4.2 数据流分析技术及Soot提供的过程内数据流分析框架

数据流分析技术是一种用于收集程序在不同点计算的值的技术，用来确定对变量的赋值可能传播到程序中的哪些部分，通过Soot提供的数据流分析框架，我们可以很方便的模拟程序在运行中的走向，进而在模拟的过程中求解问题。流分析技术从大体上可以分为两类——控制流分析与数据流分析，在应用中，控制流分析和数据流分析需要结合使用，对程序的控制流分析是对源程序或者源程序的中间表示形式的直接操作，形成控制流图；数据流分析是在控制流分析后得出的控制流图的基础上，将程序中包含数据的变量沿着控制流图的路径进行赋值和传递，直至程序完成、变量回收或者未被回收。

数据流分析的基础是控制流图，在构造完成控制流图之后，即可沿着控制流图进行传递并求解答案。控制流图可以分为两个大类，过程内的控制流图（CFG）和过程间的控制流图（ICFG），对它们的解释如下：

* 过程间控制流图——ICFG(interprocedural control flow graph)

基于函数间的调用关系以及函数内的控制流所得出的，可以用于描述主程序的执行逻辑

* 过程内控制流图——CFG(control flow graph)

主要表示为程序中的某个程序块内部顺序执行逻辑图的表示，例如主程序中的某个函数是一个具有单独逻辑功能的程序块，它内部的控制流图就是过程内控制流图

* 调用图——CG(call graph)

函数的调用关系图，主要用于表示程序执行逻辑块之间的相互调用关系，例如主程序在执行到某条语句时，调用了具有某个功能的函数，之后程序的执行顺序就会从主程序跳转至该函数块中，当函数执行完成后，通过一定的执行动作或返回值跳转回主函数，继续执行主函数的顺序逻辑

由此可见，控制流图间的基本关系为ICFG = CFG + CG，如果想要进行过程间控制流图分析的话，需要首先求出CFG与CG，然后分步进行求解

Soot提供了三种功能不同的过程内数据流分析框架，分别有对流向敏感的，对分支敏感的，前向分析，后向分析等不同类别，它们之间的类关系如下入所示：



图3-18 Soot提供的过程内数据流分析框架的继承关系

在作品中，我们选则继承前向数据流分析框架——ForwardFlowAnalysis类来实现数据流分析功能，在框架中，需要实现的方法有：

* flowThrough()

在数据流的每个节点，都会调用一次flowThrough函数，传过来上一个节点的记录值，并返回下一节点的记录值，而flowThrough函数要做的，就是对节点中的语句进行分析，适当修改记录值并返回

* merge()

Soot提供的数据流分析框架会遍历代码中的每个分支，因此在出现不同分支的时候，节点的记录值也相应会出现分支，而merge函数需要做的，就是在出现两个不同记录值的分支下，进行适当操作将两个记录值合并为一个，例如取并集，或是或集

* copy()

如merge函数中所述，在数据流进入分支时，记录值将复制成两份进行记录，但记录值是可以自定义类型的，因此需要使用者提供记录值类型的copy方法

在实现上述方法后，调用doAnalysis()即可开始执行函数内的数据流分析，在数据流分析结构中，通过getFlowAfter(Stmt)或者getFlowBefore(Stmt)即可获得运行后的记录值情况。

### 3.4.3 基于过程内数据流分析技术的对象行为记录算法

利用上面介绍的数据流分析技术，我们设计并实现了一种函数内的对象行为记录算法，通过Soot提供的过程内数据流分析框架，可以方便的实现函数内基于控制结构的完整遍历，由此可以完整的为函数内的所有变量对象的行为进行建模，数据流分析框架执行所产生的记录值中将包含每个变量的初始化、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，对这些行为的建模往往是很有用的。例如可以用于变量到常量的追溯，一个最常见的例子就是使用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，这样就将参数的常量特征隐蔽掉了，对于这种情况，不使用数据流技术的静态检测方案是很难匹配到特征的，而使用本小节设计的算法则可以有效对抗这些混淆手段。对象行为记录算法另一方面的应用是用于梳理漏洞利用链中的关系，在一些较为复杂的漏洞利用中，漏洞识别并不能只靠几个单一的特征函数就确定，依靠对对象行为的建模，可以将不同漏洞特征函数对应到每一个对象的函数调用下面，依靠对象间的相互作用关系达到更精准的识别。

为了使用Soot的数据流分析框架实现本算法，首先需要确定数据流分析中所需要使用的记录值。因为目的是为函数内的对象行为建模，所以使用HashMap<Local, LocalInfo>来进行记录，其中Local为Soot中定义的局部变量对象，LocalInfo则设计用于存储此Local对象在函数中的行为，其数据结构如下：

**public boolean isDeriveFromOrtherObject** = **false**;  
**public** Value **rootObject** = **null**;  
**public** Map.Entry<Local, InvokeExpr> **rootLink** = **null**;  
**public** List<Value> **invokeInfo** = **new** LinkedList<Value>();

rootObject记录此对象的初始化类型，isDeriveFromOrtherObject，rootLink两个变量则用于刻画不同对象之间的关系，其中isDeriveFromOrtherObject记录此对象初始化时是否来自其他对象，rootLink则在isDeriveFromOrtherObject值为true时记录此对象初始化时所用到的其他对象信息，最后invokeInfo是一个Value对象的列表，用于记录此对象在函数内所有的函数调用情况。

在记录值的数据结构建立完成后，还需要实现数据流分析框架中的几个方法，其中最关键的是flowThrough()以及merge()方法的实现，其中为了保证对象行为建模的准确性，在分支情况下，对不同的记录值需要做并集处理，因此merge()函数中对记录值做并集处理即可，下面将详细介绍flowThrough()函数的实现。

如3.4.2小节所述，flowThrough()函数用于对数据流分析中每个节点的语句进行分析处理来增减记录值，为了实现对象行为的建模，需要基于Jimple语言的15种基本语句来对语句进行解析，从中读取出有效信息并添加到记录值中，在作品中我们初步对对象相关语句的语义进行如下分类：

1. IdentityStmt

变量初始化操作

1. InvokeStmt

如果可以从中提取出Local的话，即为变量对象的方法调用操作

3） DefinitionStmt

表3-8 对Jimple语句语义解析的对照表

|  |  |  |
| --- | --- | --- |
| 左操作数 | 右操作数 | 含义 |
| Local | Local | 对象间的赋值操作 |
| Local | Constant | 使用常量对对象的赋值操作 |
| Local | ArrayRef | 使用数组对对象的赋值操作 |
| Local | AnyNewExpr | 使用new语句进行对象初始化 |
| Local | InvokeExpr(含Local) | 使用函数调用进行的赋值操作 |
| Local | InvokeExpr(无Local) | 使用工厂函数进行初始化 |
| Local | AddExpr | 使用运算表达式的赋值操作 |
| Local | FieldRef | 使用对象成员对象进行的赋值操作 |

基于对上述规则，我们将语句按照不同的语义划分，并分别从中提取有效信息录入到记录值中，在借助Soot的数据流分析框架实现函数内所有语句的结构化完整遍历后，即可生成函数内所有局部变量对象的行为建模，供之后的漏洞分析使用。

### 3.4.4 漏洞检测模块的工作流程

漏洞检测模块的完整工作流程如下图所示。首先使用Soot对APK进行初始化工作，输出的两个结果一个是反编译之后的Jimple中间代码，另一个则是控制流图，其中控制流图使用上一小节中介绍的算法为函数内的变量生成行为记录。初始化工作结束后，则开始进行漏洞特征匹配，最后输出检测结果



图3-19 漏洞检测模块工作流程

### 3.4.5 举例：WebView远程代码执行漏洞[16]的检测

WebView远程代码执行漏洞是Android应用中相当常见的一个漏洞，因其任意代码执行的功能，所造成的危害也相当广泛。本文将以此为例介绍基于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(true)函数将JavaScript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。

在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，然后使用前面介绍的变量行为记录算法，生成函数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()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，流程图如下所示：



图3-20 WebView远程代码执行漏洞检测方案

## 3.5 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，用户的手机中往往比电脑存放着更多用户的隐私信息，而另一方面，应用不论是为了改善用户体验，还是为了实现更精准的广告投放，都需要尽可能获得更详细的用户个人信息，一旦这些应用过量收集用户隐私，甚至是恶意加以利用，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应用对用户隐私的收集行为入手，进行Android应用安全性的评估。

### 3.5.1 静态污点分析算法与Sources点、Sinks点的分类

应用的隐私收集行为可以大致分为以下步骤：

1. 调用获取用户隐私的函数，这些函数的返回值均包含用户的隐私信息，这些函数的调用点称为Source点，即意为污点数据的源点
2. 携带用户隐私数据的变量在程序运行中传播，其中可能包括的行为有赋值给其他变量，销毁等，这些污点的传播过程是需要进行跟踪分析的
3. 调用发送隐私数据的函数，这些函数一般将导致用户隐私数据泄露，如发送短信的函数sendTextMessage，这些函数的调用点称为Sink点，即意为污点数据发送出去的点

静态污点分析算法是一种基于数据流分析的软件分析技术，主要用于分析程序运行过程中敏感数据的传播路径，以及判断程序的运行过程中是否导致了敏感数据的泄露，通过追踪程序中与敏感信息相关的数据流，分析关键函数之间的数据可达路径。在隐私泄露检测中，静态污点分析问题的理论基础即为IFDS问题：假设数据流集合是一个确定的有限的集合，在这个数据流集合上的数据流函数对于集合操作符是可分配的，并且数据流的追踪是可跨函数的，那么这一类问题称之过程间(Interprocedure)，有限的(Finite)，可分配的(Distributive)子集(Subset)问题。

在静态污点分析过程中，首先要做的就是从代码中收集所有的Sources点与Sinks点，然后才能接着分析任意两点间的传播路径。在Android系统中包含有很多涉及到用户隐私的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，如下表所示

表3-9对污点分析算法中Sources点的整理分类

|  |  |
| --- | --- |
| **隐私类别** | **在测试中的发现数量** |
| 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Android系统中，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3-10 对污点分析算法中Sinks点的整理分类

|  |  |
| --- | --- |
| **数据发送方式** | **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 |

在对Sources点与Sinks点归类整理后，便可以使用污点分析算法开始寻找从Sources点到Sinks点的传播路径，从而发现这些隐私泄露问题。

### 3.5.2 对Android应用的生命周期建模与IFDS问题求解

污点分析算法在实际情况中的解决方案是基于过程间数据流分析技术进行的，首先需要为待分析程序建立过程间控制流图，然后在此图上使用过程间数据流分析技术进行IFDS问题的求解，以寻找污点传播的路径。Android应用程序的运行方式不像其他程序一样有个固定的main函数入口点，然后顺序执行下去。Android应用中组件大都有自己完整的生命周期，组件在运行过程中以触发调用的方式运行，有区别于一般应用程序的顺序执行，下图为Activity组件的完整生命周期。



图3-21 Activity生命周期模型

因此，想要对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进行完整建模，通过构造一个虚拟的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。

在控制流图制作结束后，接着就需要在其上使用过程间数据流分析技术进行IFDS问题的求解。得益于FlowDroid[17]项目为Android生命周期做的完整建模，以及Soot项目提供的通用的IFDS问题求解框架[18]，我们可以很方便的在Android应用程序中使用静态污点分析算法来发现潜在的隐私泄露行为，具体操作步骤在下一小节中描述 。

### 3.5.3 使用FlowDroid进行隐私泄露行为检测的流程

FlowDroid的运行需要提供一系列的配置，具体如下：

1. 所有回调函数的列表：FlowDroid会在应用程序中提取出所有列表中记录的回调函数，并将其添加到虚拟的dummyMain方法中去。
2. Sources点与Sinks点的列表：FlowDroid会在应用程序中提取列表中记录的Sources点与Sinks，并调用污点分析算法分析传播路径。
3. Android各版本的SDK：可以在Android官网中下载，为了保证分析结果的准确性，至少需要提供1-26版本的API。
4. Soot运行结果：FlowDroid的运行需要基于Soot的运行结果，为了提高运行效率，我们将上一部分漏洞分析中Soot的初始化结果保存下来，在隐私泄露检测中再次利用
5. Config信息：在测试中发现在组件的回调分析中使用递归的方式，在单个组件的的回调函数过多时会因为递归深度上限而产生异常，因此将配置中的MaxCallbacksPerComponent设置为5防止深度递归带来的异常。

在所有配置完毕且均正确后，调用runInfoflow()函数即可开始进行隐私泄露的分析，分析结果存储在自定义的results中，从中取出分析结果并加以解析即可，完整的工作流程如下：



图3-22 隐私泄露检测模块工作流程

# 第四章 作品测试与分析

## 4.1测试大纲

#### 4.1.1 测试目的

通过对恶意样本的综合分析，指定系统测试大纲，对系统各项功能进行测试，对综合结果进行评估分析。

#### 4.1.2 测试内容

测试内容主要分为功能测试和性能测试两个部分：

功能测试包括：

1. 敏感行为监控模块功能测试
2. 流量监控模块功能测试
3. 漏洞检测模块功能测试
4. 隐私泄露检测模块功能测试
5. 服务端全部流程模块功能测试

性能测试主要包括：

1. 流量监控模块测试
2. 恶意软件鉴别模块测试
3. 漏洞检测模块测试
4. 隐私泄露模块测试

#### 4.1.3 测试环境

（1）硬件环境

神舟战神Z7笔记本电脑，Intel® CORE™ i7-6700HQ CPU，主频2.60GHz，内存16.00GB

红米Note4X，2.0GHz高通Snapdragon 625处理器，3G内存

（2）软件环境

Microsoft Windows 10 专业版 (64位)

Android 7.1.2 AOSP Extended

Xposed Framework Version 89

Mysql 5.7.18

Python 3.6.3

Java SE Runtime Environment (build 1.8.0\_131-b11)。

## 4.2 功能测试

### 4.2.1敏感行为监控模块功能测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 001 | 模块名称 | 敏感行为监控模块 |
| 测试目的 | 客户端会在应用列表中标注在白名单中的应用，用户可以根据提示选择应用进行敏感行为监控。 | | |
| 操作过程 | 1、启动App并登录，用户选择需要进行敏感行为监控的应用；  2、根据提示重启手机  3、重启后用户正常使用手机，若被监测的应用被模块评估为恶意应用，模块会通知用户并关闭应用。  4、用户可以点击通知查看监控报告，并根据报告页面的提示进行相应操作。 | | |
| 测试结果 | 选择应用进行上传    用户选择要监控的应用 用户根据提示选择是否立即重启    发现风险立即弹窗，同时杀死风险进程  用户根据提示进行选择 | | |
| 测试结论 | 用户可以选择想要监控的应用，敏感行为监控模块可以即时发现风险较高的软件操作，关闭软件并向用户生成监测报告，并给用户提供相应操作提示。 | | |

### 4.2.2流量监控模块功能测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 002 | 模块名称 | 流量监控模块功能测试 |
| 测试目的 | 检查流量代理是否正常运行以及是否能够拦截规则中的恶意流量。 | | |
| 操作过程 | 1. 启动动态监控模块，开启流量代理； 2. 运行恶意应用，观察流量代理中恶意广告的请求URL特征； 3. 自定义拦截规则，添加到恶意URL模式数据库； 4. 再一次运行恶意应用，观察恶意广告是否能加载。 | | |
| 测试结果 | 计算机生成了可选文字: 1．21K/s0誦《《夕-77％ 下午5：15 填写信息一家小贷 家家丨家小贷 有银行卡即可申请，最快3分钟到账 说 30000亓 日息0，05％起按日计息随借随还 请输入姓名 请输入身份证号 请输入手机号 请输入验证码 发送验证码 ．同意《汽车之家借贷平台用户服务使用协议》 注：用户信息受法律和以上《协议》保护，请放心填写。 申请借款 联系我们：400一858一0580400一897一3589  某恶意广告页面的恶意广告请求URL  下 午 5 ： 18  · 0 ． 02K / s 0 誦 《 《 夕 -77 ％  加 载 失 败  连 接 失 败 占 击 屏 幕 重 新 加 载  编写自定义拦截规则 恶意广告拦截成功 | | |
| 测试结论 | 流量代理能根据规则成功拦截恶意流量。 | | |

### 4.2.3漏洞检测模块功能测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 003 | 模块名称 | 漏洞检测模块功能测试 |
| 测试目的 | 在批量测试中观察漏洞检测模块对应用漏洞的检测情况以及运行稳定性 | | |
| 操作过程 | 针对11类，共计16项常见且危害性较大的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，检测结果如下表所示 | | |
| 测试结果 | |  |  | | --- | --- | | **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 | | | |
| 测试结论 | 从上表中可以看到，在应用中存在大量的编写不规范导致的漏洞，其中最多的当为Manifest文件中的allowBackup属性置为true，以及debuggable属性置为true这两个常见的配置问题，另外WebView相关的安全漏洞，证书安全相关漏洞也不在少数，这些问题均可通过本模块检测出来，帮助提高应用安全性。 | | |

### 4.2.4隐私泄露检测模块功能测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 004 | 模块名称 | 隐私泄露检测模块功能测试 |
| 测试目的 | 在批量测试中观察隐私泄露检测模块对应用隐私泄露的检测情况以及运行稳定性 | | |
| 操作过程 | 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中的分类作了整理，分类情况如下表所示 | | |
| 测试结果 | 所有测试软件中，共发现了1218条隐私泄露情况，整理如下：  对所发现的隐私泄露中的Sources点进行的归类统计如下表所示   |  |  | | --- | --- | | **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 |   对所发现的隐私泄露中的Sinks点进行的归类统计如下表所示   |  |  | | --- | --- | | **隐私发送方式** | **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 | | 通过本地文件记录数据 | 157 | | 通过应用组件传递数据 | 183 | | | |
| 测试结论 | 在3053款应用中，共发现了1218条对用户隐私的窃取行为，平均每不到3个软件就会有一条隐私泄露行为产生，其中不乏getLastKnownLocation() ，getAccounts()等敏感的用户数据，这些隐私泄露行为均可通过本模块有效的检测出来。 | | |

### 4.2.5服务端完整静态检测流程测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 005 | 模块名称 | 服务端全部检测模块 |
| 测试目的 | 用户可以顺利上传本机应用至服务器，服务器对应用进行检测，生成检测结果及分析报告并反馈给用户。 | | |
| 操作过程 | 1、启动客户端并登录；  2、用户选择应用进行上传；  3、查看上传进度，当应用上传完会提示用户；  4、服务器开始分析应用；  5、用户访问服务器，查看检测报告。 | | |
| 测试结果 | 选择应用进行上传    用户查看上传进度  Screenshot_2018-05-24-20-19-49-828_com  用户在上传记录界面中上拉刷新，当服务器分析完成后会返回该应用的检测报告  Screenshot_2018-05-24-20-18-38-635_com  用户查看分析报告  Screenshot_2018-05-24-20-18-47-249_com | | |
| 测试结论 | 用户可以成功上传应用到云端，云端能完成对应用的鉴别和分析；  用户可以收到并查看云端反馈的检测报告。 | | |

## 4.3 性能测试

在上述功能测试环境基础上，分别对服务器端静态检测模块中的恶意软件鉴别准确率、漏洞分析以及隐私泄露模块的分析效率方面进行了详细的性能测试。下面给出各个功能的测试结果。

### 4.3.1 流量监控模块性能测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 006 | 模块名称 | 流量监控模块 |
| 测试目的 | 测试代理在较高网络负载下的运行能力。 | | |
| 操作过程 | 1. 手机启动动态监测模块，开启代理； 2. 向恶意URL模式数据库添加一万条URL； 3. 电脑开启Android Studio性能分析器； 4. 手机开启直播软件，清晰度由低到高，观察性能分析器数据。 | | |
| 测试结果 | 网速80kb/s左右的性能状况    网速350kb/s左右的性能状况    网速1.29MB/s左右的性能状况 | | |
| 测试结论 | 低速情况下，代理CPU占用率小于10%，高速情况下CPU占用率峰值小于40%。低速性能较好，高速对手机性能有一定影响。 | | |

### 4.3.2 恶意软件鉴别模块准确率测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 007 | 模块名称 | 恶意软件鉴别模块 |
| 测试目的 | 测试恶意软件鉴别模块对恶意应用识别的准确率 | | |
| 操作过程 | 取正常应用与恶意应用各300个，通过反编译分析获取敏感API，使用随机森林算法得出结果，对600个样本进行批量检测。 | | |
| 测试结果 | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 样本类别 | 正常样本 | 恶意样本 | 总量 | 准确率 | 误判率 | | 正常样本 | 287 | 13 | 300 | 95.7% | 4.3% | | 恶意样本 | 290 | 10 | 300 | 96.7% | 3.3% | | 总计 | 577 | 23 | 600 | 96.2% | 3.8% |   表4-1 对600个应用的检测结果  将上述600个应用分别运行于360安全卫士，安全管家等主流安全软件平台，与本作品的检测效果进行对比   |  |  |  |  | | --- | --- | --- | --- | | 软件类型 | 检测率 | 误判率 | 耗时 | | 360安全卫士 | 90% | 10% | 176s | | 腾讯安全管家 | 88% | 12% | 194s | | 百度手机卫士 | 91% | 9% | 205s | | 本作品 | 96.2% | 3.8% | 40s |   表4-2 本作品的检测系统与各大主流平台检测效果对比 | | |
| 测试结论 | 从表中数据可见，本作品的检测系统相较其他主流检测平台具有较高的检测准确率，而且在实际测试中，主流检测平台在进行扫描时会感觉卡顿且检测时间较长，相比之下，本作品的检测耗时极少，仅需读取训练模型，然后得出结果，具有优良的检测性能。 | | |

### 4.3.3 漏洞检测模块性能测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 008 | 模块名称 | 漏洞检测模块 |
| 测试目的 | 测试恶意软件鉴别模块在检测中的时间花费 | | |
| 操作过程 | 对随机挑选的100款应用进行了漏洞检测，并记录运行时间，统计如右图所示 | | |
| 测试结果 | |  |  | | --- | --- | | 应用平均大小(MB) | 平均运行时间(s) | | 12 | 28 | | | |
| 测试结论 | 从表中数据可见，本作品的检测系统相较其他主流检测平台具有较高的检测准确率，而且在实际测试中，主流检测平台在进行扫描时会感觉卡顿且检测时间较长，相比之下，本作品的检测耗时极少，仅需读取训练模型，然后得出结果，具有优良的检测性能。 | | |

### 4.3.4 隐私泄露检测模块性能测试

|  |  |  |  |
| --- | --- | --- | --- |
| 测试编号 | 009 | 模块名称 | 隐私泄露检测模块 |
| 测试目的 | 对随机挑选的100款应用进行了隐私泄露检测，并记录运行时间，统计如下表所示。 | | |
| 操作过程 | 对随机挑选的100款应用进行了漏洞检测，并记录运行时间，统计如右图所示 | | |
| 测试结果 | |  |  | | --- | --- | | 应用平均大小(MB) | 平均运行时间(s) | | 12 | 75 | | | |
| 测试结论 | 在系统中，隐私泄露模块使用的是漏洞检测模块初始化后的数据，因此应用反编译的时间不包括在平均运行时间内。隐私泄露模块的运行时间主要取决于Sources点以及Sinks点发现的数量，之后使用污点分析算法查找传播路径是基于图算法的，因此点的数量越多，运行时间就越长，需要的内存就越多。 | | |

# 第五章 总结与展望

## 5.1 工作总结

本作品针对当前Android恶意应用泛滥，市面上安全软件对Android应用安全性的检测手法单一等问题进行了研究，最终设计并实现了一种评估Android应用安全性的多重检测系统，模型从动态分析，静态分析两个衡量软件安全性的方向入手，结合机器学习算法，共设计了动态敏感行为监控，动态流量监控，静态恶意应用鉴别，应用漏洞扫描，隐私泄露检测五个检测模块并进行综合运用。经过测试证明，可以较好的解决Android应用中可能存在的恶意代码攻击，漏洞代码利用，隐私泄露这三种影响Android手机安全性的问题，而且可以避免过于单一的检测手法所可能造成的漏报，误报等问题。其中本作品的具体工作包括：

1. 小组成员通过阅读相关论文以及开源安全工具的实现代码，基本掌握以下安全技术作为本作品的实现基础
2. Android系统的Hook技术
3. Android系统下的代理及VPN技术
4. 朴素贝叶斯算法和随机森林算法，以及基本的模型评估方案
5. Java静态代码审计框架（Soot），及其提供的过程内数据流分析

框架

1. 污点分析算法以及过程间数据流分析技术
2. 对常用的安全检测技术加以改进，以实现本作品预期的功能，或是达到更高的检测率，其中包括：
   1. 在敏感行为监控模块设计实现风险评估算法来进行Android应用程序恶意行为的告警。
   2. 在静态恶意应用鉴别模块，通过提取应用中敏感API作为特征并加以优化，最高精确度接近了97%，有效提高了恶意应用的识别率。
   3. 在静态漏洞扫描模块基于数据流分析技术，设计实现了函数内的变量行为记录算法并在检测过程中加以运用，以提高漏洞检测的精确度。

3）在设计的五个检测模块的基础上，对不同模块的检测技术，不同模块负责检测的安全性方面进行综合运用，通过客户端动态检测，服务器端静态检测的总体结构加以安排，实现了基于动静态结合的Android应用安全性多重检测系统。

## 5.2 创新点

1.采取本地检测与服务器端检测相结合的方式，解决本地检测样本数量少、效率低的弊端，同时在服务器端静态检测模块中提供更加全面的检测功能。

2. Hook模块给每个被监控的应用生成完善、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。

3.预设白名单，为用户提供监控选择的参考，防止大型应用（如支付宝、QQ）被误报；同时设计风险评估算法进一步提高示警的准确率。

4. API调用链更新机制保证了多数安全APP在联网操作时不会被误判。

5.结合数据流分析技术进行漏洞检测，通过设计的函数内变量行为记录算法来对砍漏洞检错中可能存在的代码混淆，以及用于梳理漏洞利用链中的关系。

6.对流量的规则匹配采用高效的算法保证匹配过程不明显影响性能。

7.通过多种机器学习算法的性能分析，结合权限特征与敏感API特征建立模型，并对各种模型进行比较，得出具备最优效果的基于敏感API的随机森林算法检测模型

## 5.3 未来展望

由于小组成员学历尚浅，Android应用新的攻防技术又层出穷，因此在本作品中还有很多可以完善并加以改进的地方，具体整理下来有如下几项：

1）通过对更多的恶意软件进行逆向，分析其源码，对恶意行为的API调用与组合进行更细致的研究从而更新Hook规则、修正风险阈值使得误判率更低、查杀率更高。

2）通过结合数据流分析技术的代码审计，可以完成传统特征匹配方法所难以完成的任务，在本作品中对数据流分析技术的应用还比较有限，将来可以在此基础上，更完善的将漏洞检测与数据流分析技术结合起来。

3）对现有的基于敏感API特征的随机森林鉴别算法进行改善，改用多种恶意特征相结合的方法鉴别恶意软件，例如设定一个权重系数，将权限与敏感API特征相结合得出特征向量，然后通过相应机器学习算法训练，来提高恶意应用鉴别的准确率。

4）完善VPN抓包程序，支持更多协议以显示更多流量信息。提高VPN程序的稳定性，可代替代理程序实现网络层的流量拦截功能。使用机器学习算法区分正常流量和恶意流量，实现对未知流量的鉴别。

# 参考文献

[1]陈鹏,赵荣彩,单征,韩金,孟曦.基于动静结合的Android恶意代码行为相似性检测[J].计算机应用研究,2018,35(05):1534-1539.

[2]文伟平,梅瑞,宁戈,汪亮亮.Android恶意软件检测技术分析和应用研究[J].通信学报,2014,35(08):78-85+94.

[3]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.

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

[5]李伟,陈忠红.安卓Xposed框架安全应用研究[J].电脑知识与技术,2016,12(10):49-50.

[6]刘淼. Android流量监控技术研究与实现[D].华中科技大学,2012.

[7]朱佳伟,喻梁文,关志,陈钟.Android权限机制安全研究综述[J].计算机应用研究,2015,32(10):2881-2885.

[8]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.电子科技大学.2016(03).49-53

[9]邵舒迪.基于权限和API特征结合的Android恶意软件检测方法.计算机科学.2017(4).136-138

[10]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.

[11]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

[12]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.

[13]VirusShare.https://virusshare.com

[14]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

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

[16]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

[17] 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

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