计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(7):26−33 [doi: 10.15888/j.cnki.csa.006418]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

安卓恶意软件的静态检测方法①

陈红闵, 胡江村

(中南民族大学 电子信息工程学院, 武汉 430074)
通讯作者: 陈红闵, E-mail: minny_stu@163.com

摘　要: 近几年, Android 平台的恶意软件数量几乎以几何式的速度增长, 故提出一种恶意软件检测方法是必要的. 本文利用现如今疯涨的 Android 恶意样本量和机器学习算法建立分类预测模型实现对恶意软件的静态检测. 首先, 通过反编译 APK 文件获取 AndroidManifest.xml 文件中权限特征, baksmali 工具反编译 class.dex 成 smali 文件得到危险 API 特征. 然后运用机器学习中多种分类和预处理算法比较每一特征和联合特征检测的准确率. 实验结果表明, 联合特征检测准确率高于单独特征, 准确率达到 97.5%.

关键词: API; 权限; APK; 静态检测; 恶意软件

引用格式: 陈红闵,胡江村.安卓恶意软件的静态检测方法.计算机系统应用,2018,27(7):26–33. http://www.c-s-a.org.cn/1003-3254/6418.html

Static Detection Method of Android Malware

CHEN Hong-Min, HU Jiang-Cun

(College of Electronics and Information Engineering, South-Central University for Nationalities, Wuhan 430074, China)

Abstract: In recent years, the number of malware on the Android platform has a geometrical growth. Therefore, it is very necessary to have a method to detect Android malware. This study experiments with a large number of Android malware samples and machine learning technology to establish a prediction model for malware classification, which is run in the static detection process. First, we obtain the permissions and the dangerous API information of Android applications, the permissions feature in its AndroidManifest.xml file by decompiling APK files and its dangerous API features by translating decompiles class.dex files into smali files together with the baksmali tool. Then, we use multiple classification algorithms and preprocessing algorithm to compare the accuracy rate of the single detection and the conjoint detection. The experimental results show that the accuracy rate of the conjoint detection is higher than that of the single detection, and the accuracy rate reaches up to 97.5%.

Key words: API; permission; APK; static detection; malware

在如火如荼的高科技时期, Android 软件的开发呈现了爆发式增长. 根据日前 App Annie 发布的《全球移动应用市场 2016 年回顾报告》显示数据表明, 2015 年至 2016 年两年全球应用下载量增长率为 15%. 可惜不幸的是, 这样的受欢迎程度也会吸引恶意软件开发者, 预置应用程序、捆绑下载、过度获取权限、山寨应用等防不胜防. 恶意应用程序的盛行却让用户的个人隐私逐渐走向透明, 在《2017Q1 中国手机安全市场研究报告》[1]中提到, 89.6% 的受访用户表示曾遭受过个人隐私信息泄露, 诈骗电话等, 如今信息安全成为很多用户的心腹大患. 360、金山等是深受用户喜爱的安全厂商也投入到移动安全领域, 而这些软件进行杀毒基本原理是通过匹配已知的病毒木马特征来确认入侵行为, 以防火墙、动态监控等方式进行主动防御, 但缺点是依赖病毒特征库的更新, 学习新型病毒能力较弱[2,3]. 本文在前人的基础上进行进一步研究, 提取不

① 收稿时间: 2017-11-03; 修改时间: 2017-11-27; 采用时间: 2017-12-01; csa 在线出版时间: 2018-05-24

同的特征组合进行检测. 首先, 通过反编译提取权限和高危 API, 经过预处理组成权限——API 特征集合, 利用机器学习分类算法进行检测, 并与单独特征集合进行分类的准确率进行比较, 得出相应的优化检测方案. 并将服务器端的这一静态检测过程整合至开发的安卓恶意软件检测与防御系统中, 以向用户返回静态检测报告的方式返回检测结果.

1 Android 应用程序安全机制

1.1 权限机制

Android 是一个“权限分离”的系统, 在开发 App 过程中, 要想使用 Android 系统受限资源, 需在 Android.xml 文件中申请相关资源的权限, 利用唯一字符串来分别表示每一资源的权限. 权限主要由以下组成: 权限的名称; 属于的权限组; 保护级别. 每个权限通过 protectionLevel 来标识保护级别: normal, dangerous, signature, signatureorsystem[4]. 使用该权限时就要根据不同的保护级别进行认证, 如 normal 的权限只要申请了就可以使用, 而 dangerous 权限需要用户确认才能被使用. 在 AndroidManifest.xml 中会通过一些标签如 <permission>标签, <permission-group>标签 <permission-tree>等标签来指定 package 的权限信息. 为此将所有应用程序的权限提取出来作为特征值, 具有一定的实际意义. 但是不同的 App 会申请不同的权限, 故使用频率有一定的差异, 同时恶意软件和良性软件在申请权限上也互不相同, 因此将恶意软件和正常软件中所申请的权限全部提取出来, 根据使用次数分别提取恶意和良性软件中排名前 20 的使用权限, 同时进行归类, 组成相应的权限特征集.

1.2 签名机制

应用程序通过签名机制唯一区别 APK 文件, 这样就解决了 Android 程序重名的问题, 因此开发者必须对开发的应用程序进行数字签名, 通过签名将应用程序作者和应用程序之间建立一种信任关系. 应用程序的签名文件和证书两者缺一不可, 当用户在安装应用程序的过程中, 首先会被系统的安装程序检查, 确定该应用程序是否被签名, 未被签名的应用程序, 系统安装程序将阻止该应用程序的安装, 同样的, 在应用程序需要升级时, 新版的应用程序也会被系统安装程序检查, 确定其签名与旧版本的应用程序签名是否一致, 一致则更新, 不一致则被认为是新的应用程序来安装, 同时也防止了被恶意软件替换的风险. Android 有 jarsigner 和 signapk 签名两种方式: jarsign 对 APK 签名, 因其是 Java 内置的一个签名工具, 所以通过 jarsign 签名需要安装 JDK. 而 signapk 是为 Android 应用程序签名专门开发的工具. jarsigner 和 signapk 签名的签名算法大同小异. 通过上面的签名后会生成一个 META-INF 文件夹, 这里有三个文件: MANIFEST.MF、CERT.RSA、CERT.SF[5].

1.3 Android 恶意软件分析检测方法及理论

恶意软件分析和检测技术可以分为三类: 静态分析, 动态分析和混合方法. 静态检测利用相应的反编译工具提取程序的静态特征如语法语义、签名等特性等进行分析, 评估软件安全. 在动态分析技术中, 应用程序被部署在模拟器上或被控制的设备上进行模拟和监控[6,7]. 而近年来国内外很多学术研究人员及一些商业软件公司已经意识到传统的基于签名的静态分析方法很容易受到攻击, 也无法实现对未知恶意软件的检测. 特别是, 常见的隐形技术, 例如加密技术, 代码转换, 以及环境意识的方法等都具有生成恶意软件的能力[8]. 文献[9]提出的基于特征码的恶意代码检测方法利用特征值匹配技术进行检测与国外著名的 Androguard Android 恶意代码检测工具一样都是基于签名的检测方法, 但不能对未知恶意应用检测. 在文献[10]中作者对网络行为、短信等进行监控, 考虑到 Android 平台实际资源需求, 一般单独使用 Monkey 程序进行动态测试, 实用性不强. 在大数据时代, 数据挖掘技术越来越成熟, 云端处理能力越来越强, 很多研究者也将机器学习算法和巨大的恶意样本量结合起来进行相应的研究[11–13]. Justin Sahs[14]利用 Androguard 提取 APK 包特征, 利用分类器训练这些特征来分类 Android 软件. 王超[15]设计了基于机器学习算法的 Android 恶意软件检测系统, 他将静态、动态、客户端和云端结合成一体, 实行全方位的控制与监听检测. 解决了 Android 系统在权限控制的缺点, 实现了细粒度的权限控制. 并通过实验数据比较不同机器学习分类算法, 从而提高检测效率. 徐欣等人[16]设计并实现了一个恶意软件动态分析云平台, 通过基于虚拟化沙箱机制来判断目标软件是否是恶意软件, 但对网络的实时性和并发性要求较高.

2 方案介绍

本文在邵舒迪等人[17]提出的基于权限和 API 特征结合的 Android 恶意软件检测方法基础上进行相应的

预处理过程及提取不同的特征集合, 同时运用不同的分类算法提高检测准确率. 然后将相应的静态检测过程整合至开发的安卓恶意软件检测与防御系统中, 以恶意软件检测报告详情方式返回给用户, 供用户知晓.

2.1 Android 恶意程序检测流程

本文提出的 Android 恶意软件检测与防御系统, 将静态检测与动态监控相结合, 实现在服务器端利用机器学习算法对已知正常和恶意软件进行训练学习, 建立相应的分类模型, 对未知的恶意软件进行静态检测, 并返回相应的静态检测报告. 客户端通过对未知样本进行扫描, 与已存在的病毒库通过 MD5 值进行比对过滤, 同时通过监听广播发送与接收实时监听软件的行为信息, 并提供给用户防御与处理措施. 用户通过上传本地的 APK 文件至服务器端进行静态检测, 该系统能有效地用于对已知未知应用的恶意性进行检测. 整个系统检测流程如图 1 所示.

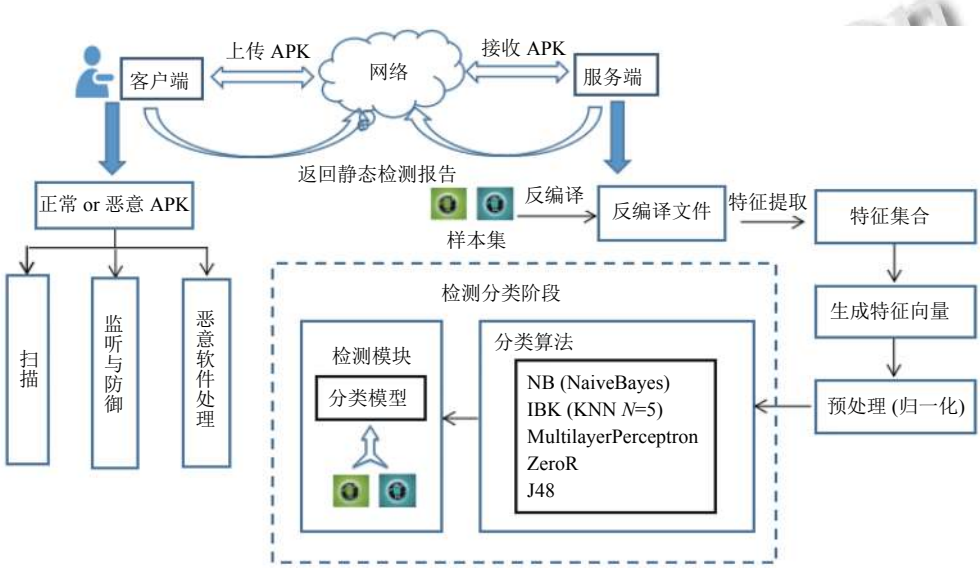

图 1 系统检测框架

2.2 特征提取

在权限特征提取阶段, 采用静态分析方法, 分别收集正常和恶意 APK 软件作为训练样本库, 利用 Java 编写的解压应用程序对收集的样本进行批量解压, 利用 Android 自带的 aapt(aapt.exe 在 SDK 的 platform-tools 目录下) 工具反编译 APK 同时获取其 AndroidManifest.xml 文件中申请的权限信息, 将提取的权限进行格式化处理, 每个样本抽象为 $1\times(n+1)$ 维向量, 其数据格式为(packagename,$\chi_1,\chi_2,\chi_3,\cdots,\chi_n$), 其中 packagname 是应用程序的名称, $\chi_1 \sim \chi_n$ 是统计的 Android 应用程序申请的权限, 并将其存入 Oracle 数据库中, 为了统计恶意样本和正常样本申请权限的使用情况, 分别将得到的权限信息在数据库中存入两张不同的表 permissiondesc 和 permissiondescmal 中, 并分别选择将排名前 20 的权限提取出来, 通过程序抽象为 1×20 维向量$(\alpha_1,\alpha_2,\alpha_3,\cdots,\alpha_{20})$, $(\beta_1,\beta_2,\beta_3,\cdots,\beta_{20})$该程序使用的权限标记为 1, 未使用的则标记为 0. 由于在此实验过程中要引入 weka 数据挖掘工具, 其处理数据集格式为 Arff 格式的数据, arff 格式是 weka 专用的文件格式, 全称 Attribute-Relation File Format. 它是一个 ASCII 文本文件, 记录了一些共享属性的实例. arff 格式文件主要由两个部分构成, 头部定义和数据区. 头部定义包含了关系名称 (relation name)、一些属性 (attributes) 和对应的类型. 数据区以@data 开头, 故数据区一横行就代表一个样本实例, 竖行是作为属性和变量. 遍历样本生成权限特征向量集如下所示:

```
@data
1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, Malware
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, Malware
0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, Malware
0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, Malware
0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, Benign
1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, Benign
1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, Benign
1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, Benign
```

如图 2 和图 3 所示, 是正常样本和恶意软件排名

前 20 的权限, 对比其使用情况发现, 两者重合申请的权限比较多, 故将其进行组合生成 1×23 维共有特征向量$(\delta_1,\delta_2,\delta_3,\cdots,\delta_{23})$生成权限数据特征集如表 1 所示, 将其输入 weka 中进行训练. 在预处理模块中选择信息增益 InfoGainAttributeEval 特征选择算法来衡量各权限在判断恶意软件上的贡献值大小. 在信息增益中, 重要性的衡量标准就是看特征能够为分类系统带来多少信息, 带来的信息越多, 该特征越重要. 即某权限特征的信息增益值越大, 即它为恶意值的贡献越大. 设置一定的阈值, 选择相应的特征重新进行训练, 比较其实验结果.

表 1 权限数据特征

序号	特征值
1	READ_PHONE_STATE
2	ACCESS_COARSE_LOCATION
3	WRITE_SETTINGS
4	MOUNT_UNMOUNT_FILESYSTEMS
5	RESTART_PACKAGES
6	RECORD_AUDIO
7	INTERNET
8	ACCESS_NETWORK_STATE
9	SEND_SMS
10	ACCESS_WIFI_STATE
11	SYSTEM_ALERT_WINDOW
12	CAMERA
13	VIBRATE
14	CHANGE_WIFI_STATE
15	CHANGE_NETWORK_STATE
16	READ_EXTERNAL_STORAGE
17	GET_TASKS
18	WAKE_LOCK
19	ACCESS_FINE_LOCATION
20	INSTALL_SHORTCUT
21	READ_LOGS
22	WRITE_EXTERNAL_STORAGE
23	RECEIVE_BOOT_COMPLETED

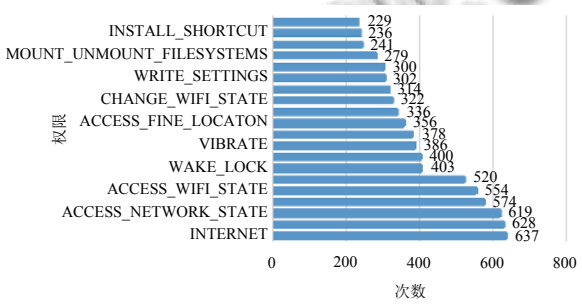

图 2 正常样本权限统计直方图

在高危 API 特征提取阶段, 参照张锐[4]在其论文 Android 环境下恶意软件静态检测方法研究中提出的通过逆向解析 Android 软件安装包, 获取代码级的 41 种高危 API 作为另一种检测特征. 将得到的高危 API 特征存入数据库 dangerousapi 表中, 在提取高危 API 特征时, 将获取样本的 APK 文件进行解压得到文件如表 2 所示.

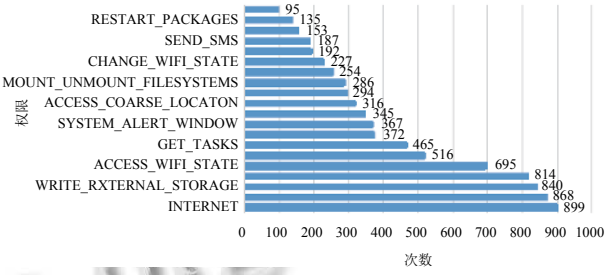

图 3 恶意样本权限统计直方图

表 2 APK 文件

目录名	作用描述
assets 目录	存放需要打包到 APK 中的静态文件
lib 目录	程序依赖的 native 库
res 目录	存放应用程序的资源
META-INF 目录	存放应用程序签名和证书
AndroidManifest.xml	应用程序的配置文件
Class.dex	Dex 可执行文件
resources.arsc	资源配置文件

classes.dex 是 java 源码编译后生成的 java 字节码文件, 在 Dalvik 虚拟机上执行, 它包含了所有的源码信息, 利用反编译工具 Baksmali 解析 APK 的 classes.dex 文件, 即可获得以 smali 为后缀的文件, 遍历所有 smali 文件获取每一 APK 高危 API 的使用次数生成特征空间向量. 根据 Android 官网提供的 API 信息及张锐[4]在其论文中统计的具备高危行为的 41 种 API, 根据 smali 语法规则生成相对应的 java 函数形式, 遍历所有 smali 文件得到 41 种 API 使用次数, 每使用一次 count++最后每一样本便生成 1×41 维特征向量$(\psi_1,\psi_2,\psi_3,\cdots,\psi_{41})$, smali 文件中方法的表现形式: Lpackage/name/ObjectName;->MethodName(III)Z Lpackage/name/ObjectName;表示类型, MethodName 是方法名. III 为参数 (在此是 3 个整型参数), Z 是返回类型 (bool 型). 方法的参数是一个接一个的, 中间没有隔开.

代码实现如下所示:

```
/**
 * 根据 smali 语法规则提取包名+函数名
 * 方法调用的表示格式: Lpackage/name/ObjectName;->MethodName(III)Z
```

```
        * 字段调用的表示格式: Lpackage/name/ObjectName;->FieldName:Ljava/lang/String;
        */
        public String GetMethodAllPath(String str) {
            String result = "-1";
            if (str.lastIndexOf(";->") != -1) {
            String tempStr=str.substring(0, str.lastIndexOf(";->"));
                str = str.substring(str.lastIndexOf(";->") + 3);
                if (str.contains(":")) { // 如果截取的字符串中包含冒号 则代表这个是变量 否则为方法} else {
            String packegeName = tempStr.substring(
                tempStr.lastIndexOf(", ") + 1).trim();
                packegeName = packegeName.replaceAll("/", ".").substring(1);
            String methodName = str.substring(0, str.indexOf("("));
            result = packegeName + "." + methodName;
                }
            }
            return result;
        }}
```

将其转为 arff 格式数据作为高危 API 数据特征集输入 Weka 中进行训练. 遍历样本得到 API 特征向量集如下所示:

```
@data
0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 5, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 2, 0, 0, 4, 0, 3, Malware
0, 0, 0, 3, 14, 0, 7, 2, 0, 0, 0, 0, 0, 0, 2, 8, 3, 20, 0, 0, 9, 0, 5, 0, 0, 0, 0, 4, 0, 3, 0, 0, 0, 4, 0, 5, 0, 0, 36, 0, 5, Malware
0, 0, 0, 9, 3, 0, 1, 34, 0, 0, 0, 0, 0, 0, 35, 64, 3, 23, 0, 0, 24, 0, 2, 0, 0, 0, 0, 8, 0, 1, 0, 0, 0, 3, 0, 4, 0, 0, 47, 0, 8, Benign
0, 3, 0, 42, 33, 0, 16, 79, 0, 0, 0, 0, 0, 0, 89, 80, 42, 77, 0, 0, 75, 0, 20, 0, 0, 0, 0, 26, 0, 4, 0, 0, 0, 27, 0, 10, 0, 3, 76, 0, 23, Benign
```

由于单一权限和高危 API 信息能在一定程度上对恶意软件进行检测, 但是正确率不太高, 同时也存在一定的局限性, 所以考虑将其合并在一起作为统一特征集进行检测, 然后对得到的特征集进行相应的预处理过程选出最优特征子集, 提高分类器的正确率.

3 实验过程及结果分析

Weka 自带了许多机器学习算法, 它能够用来进行模型的训练和预测. 当使用这些算法来构建模型的时候, 我们需要一些指标来评估这些模型的性能, 为此定义一组相关指标从各方面衡量实验效果. 针对一个二分类问题, 将实例分成正类 (Postive) 或者负类 (Negative). 但是实际中分类时, 会出现 4 种情况:

(1) 真正类 (True Positive, TP): 被模型预测为正类的正样本.

(2) 假正类 (False Positive, FP): 被模型预测为正类的负样本.

(3) 假负类 (False Negative, FN): 被模型预测为负类的正样本.

(4) 真负类 (True Negative, TN): 被模型预测为负类的负样本.

1) 正确率 (Accuracy)

正确率是我们最常见的评价指标, $Accuracy = (TP+TN)/(P+N)$, 这个很容易理解, 就是被分对的样本数除以所有的样本数, 通常来说, 正确率越高, 分类器越好.

2) 错误率 (Error Rate)

错误率则与正确率相反, 描述被分类器错分的比例, $Error\ Rate = (FP+FN)/(P+N)$, 对某一个实例来说, 分对与分错是互斥事件, 所以 $Accuracy = 1 - Error\_Rate$.

3) 精度 (Precision)

精度是精确性的度量, 表示被分为正例的示例中实际为正例的比例, $Precision=TP/(TP+FP)$.

4) 召回率 (Recall)

召回率是覆盖面的度量, 度量有多个正例被分为正例, $Recall=TP/(TP+FN)=TP/P=Sensitivity$.

5) 灵敏度 (Sensitivity)

灵敏度是将正样本预测为正样本的能力, $Sensitivity=TP/(TP+FN)$, 可以看到召回率与灵敏度是一样的.

6) 特异度 (Specificity)

特异度是将负样本预测为负样本的能力, $Specificity=TN/(TN+FP)$.

7) ROC 曲线

ROC (Receiver Operating Charateristic): ROC 的主要分析工具为画在 ROC 空间的曲线, 横轴为 1–*Specificity*, 纵轴为 *Sensitivity*. 在分类问题中, 一个阀值对应于一个特异性及灵敏度, 一个好的分类模型要求 ROC 曲线尽可能靠近图形的左上角.

8) AUC 面积

AUC (Area Under roc Curve) 值指处于 ROC 曲线

下方的那部分面积大小, 一个理想的分类模型其 AUC 值为 1, 通常其值在 0.5 至 1.0 之间, 较大的 AUC 代表了分类模型具备较好的性能[18].

3.1 实验过程

本次实验选取了 552 个正常样本和 475 个恶意样本作为实验的样本数据集, 为了避免数据的不均衡性, 正常样本分别从木蚂蚁安卓市场 (http://www.mumayi.com/) 和安卓市场 (http://apk.hiapk.com/) 分类别通过自己编写的爬虫程序抓取, 包括影音、生活、社交、教育、购物等多个应用类别. 而 475 个恶意样本则来自于 https://virusshare.com/国外一个共性病毒库网站上所收集的恶意软件样本集, 通过发邮件联系管理员注册接收邀请获得样本来源. 在分类算法的选择上, 使用基础分类器 NB (NaiveBayes)、IBK (KNN *N*=1)、MP (MultilayerPerceptron)、J48、ZeroR 五种分类器进行分类, 特征采用 10 折交叉检验来评估模型. 其中 ZeroR 分类器作为基分类器基准衡量其它分类器效果.

3.2 实验结果

步骤 1. 只选取权限特征集 permission 进行训练, 此时样本数据为 24×1027 维向量, 其中 23 维为前文所述正常和恶意样本的组合权限特征, 另一维为分类属性, 1027 为总样本数量 (后续实验相同), 不经过特征属性选择, 得到实验结果如表 3 所示.

表 3 五种分类算法检测结果 (permission)

	TP	*FP*	*Precision*	*Recall*	*F-Measure*	*ROC*
NB	0.691	0.302	0.698	0.691	0.692	0.744
IBK	0.760	0.232	0.768	0.760	0.760	0.832
MP	0.765	0.236	0.766	0.765	0.766	0.828
J48	0.779	0.222	0.779	0.779	0.779	0.807
ZeroR	0.537	0.537	0.289	0.537	0.376	0.499

步骤 2. 对权限特征集进行特征属性筛选 (InfoGainAttributeEval Ranker -T 0.0 -N -1) 得到权限优化特征集 permissionInfoGain, 此时数据集为 17×1027 向量, 其中 16 维为经过信息增益算法得到各特征风险性评分筛选出的权限特征如图 4 所示, 所得实验结果如表 4 所示.

表 4 和表 3 实验结果对比可知, 将权限特征集用信息增益选择算法进行预处理之后, 虽然整体的正确率没有明显的改善, 但是 ROC Area 有明显提高, 而分类器中 ROC 指标对分类器预测准确性具有重要意义, 故对权限作预处理对分类效果改善具有促进作用.

```
average merit      average rank  attribute
 0.134 +- 0.008     1   +- 0       1 android.permission.CHANGE_WIFI_STATE
 0.119 +- 0.005     2   +- 0       2 android.permission.CHANGE_NETWORK_STATE
 0.09  +- 0.005     3   +- 0       3 android.permission.WRITE_EXTERNAL_STORAGE
 0.031 +- 0.002     4.8 +- 0.87    4 com.android.launcher.permission.INSTALL_SHORTCUT
 0.03  +- 0.002     4.9 +- 0.7     5 android.permission.CAMERA
 0.029 +- 0.002     5.4 +- 0.92    6 android.permission.GET_TASKS
 0.026 +- 0.003     6.9 +- 0.3     7 android.permission.WAKE_LOCK
 0.018 +- 0.002     8.4 +- 0.66    8 android.permission.ACCESS_NETWORK_STATE
 0.015 +- 0.001     9.8 +- 0.87   10 android.permission.RECORD_AUDIO
 0.015 +- 0.001     9.8 +- 1.17    9 android.permission.MOUNT_UNMOUNT_FILESYSTEMS
 0.015 +- 0.002    10   +- 0.89   11 android.permission.SYSTEM_ALERT_WINDOW
 0.011 +- 0.002    12.6 +- 0.66   12 android.permission.ACCESS_FINE_LOCATION
 0.01  +- 0.002    12.9 +- 0.83   13 android.permission.READ_LOGS
 0.008 +- 0.001    13.7 +- 0.78   14 android.permission.READ_EXTERNAL_STORAGE
 0.005 +- 0.003    14.9 +- 0.7    15 android.permission.VIBRATE
 0.002 +- 0.003    15.9 +- 0.3    16 android.permission.ACCESS_WIFI_STATE
```

图 4 权限特征信息增益结果

表 4 五种分类算法检测结果 (permissionInfoGain)

	TP	*FP*	*Precision*	*Recall*	*F-Measure*	*ROC*
NB	0.703	0.288	0.712	0.703	0.703	0.751
IBK	0.757	0.235	0.765	0.757	0.757	0.826
MP	0.758	0.243	0.758	0.758	0.758	0.829
J48	0.773	0.229	0.773	0.773	0.773	0.821
ZeroR	0.537	0.537	0.289	0.537	0.376	0.499

步骤 3. 对高危 API 特征集 API41, 此时样本数据为 42×1027 向量, 其中 41 维为前文所述的 41 种高危 API, 1027 为总样本数量, 利用上述 5 种算法进行实验, 得到实验结果如表 5 所示.

表 5 五种分类算法检测结果 (API41)

	TP	*FP*	*Precision*	*Recall*	*F-Measure*	*ROC*
NB	0.612	0.350	0.679	0.612	0.589	0.721
IBK	0.797	0.203	0.798	0.797	0.797	0.818
MP	0.762	0.242	0.761	0.762	0.762	0.820
J48	0.781	0.221	0.781	0.781	0.781	0.785
ZeroR	0.537	0.537	0.289	0.537	0.376	0.499

步骤 4. 将权限和 API 特征集 permission-API 组合至至一起进行实验, 此时样本数据为 58×1027 向量, 其中 57 维为 16 种权限和 41 种高危 API 的组合, 102 为总样本数量得到实验结果如表 6 所示.

表 6 五种分类算法检测结果 (permission-API)

	TP	*FP*	*Precision*	*Recall*	*F-Measure*	*ROC*
NB	0.768	0.218	0.786	0.768	0.767	0.859
IBK	0.875	0.122	0.877	0.875	0.876	0.949
MP	0.975	0.018	0.975	0.975	0.975	0.959
J48	0.963	0.034	0.963	0.963	0.963	0.965
ZeroR	0.537	0.537	0.289	0.537	0.376	0.499

据上述 4 张表的结果均可以看出, 选择 NB、IBK、MP、J48 四种分类器效果均比基分类器 ZeoR 效果好, 说明上述实验结果是有实际意义的. 同时将表 3 和表 5 的实验结果与表 6 的实验结果对比可

以看出, 将权限和高危 API 进行联合检测比单独使用权限 (permission) 特征和高危 API 特征进行检测的模型有较高的检测率和准确率. 故后续实验过程中选择经过预处理的权限特征和 API 组合成整体特征集进行训练具有实际意义.

3.3 结果比较

为了评估本方案的有效性, 本文将提出的方法与近年来的相关工作进行对比, 魏理豪、艾解清等人[5] 2016 年提出了将 Android 权限、敏感 API 调用、MD5 等特征进行分析处理, 共同协作完成对 Android 应用程序的安全检测. 如表 7 所示是其实验结果.

表 7 魏理豪等方案实验结果

	TP	*FP*	*Precision*	*Recall*	*F-Measure*	*ROC*
NB	0.807	0.195	0.807	0.807	0.807	0.874
RT	0.822	0.179	0.822	0.822	0.822	0.821
J48	0.853	0.148	0.853	0.853	0.853	0.862
IBK	0.871	0.134	0.873	0.871	0.871	0.909
SMO	0.883	0.119	0.883	0.883	0.883	0.882
RF	0.903	0.100	0.903	0.903	0.902	0.970

由表 7 结果与本方法中表 6 中所得到的实验结果对比可知, 在所选取的分类算法大体相同的情况下, 本文选择较少的特征得到较好的检测准确率, 稍显优势.

4 实验分析

由于在 weka 中得到的 ROC 曲线图不够清晰而且无法导出, 因此通过 result 对象得到 TP Rate 和 FP Rate 数组, 利用 SPSS 软件绘制了其 ROC 曲线图. 图 5、图 6 和图 7 分别是根据上述实验结果画出的 ROC 曲线.

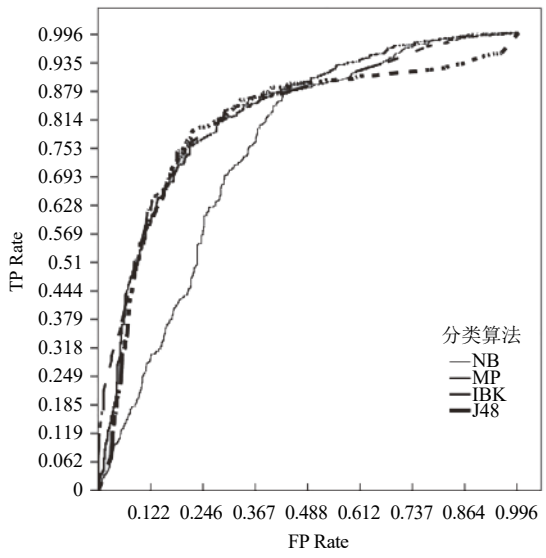

图 5 权限 ROC

从上述 ROC 曲线图中可以看出, 当选择单独权限和 API 特征时, IBK 和 J48 分类器分类效果明显好于其他两类分类器, 而将两特征结合起来时, MultilayerPerceptron 分类效果优于另外三种分类器.

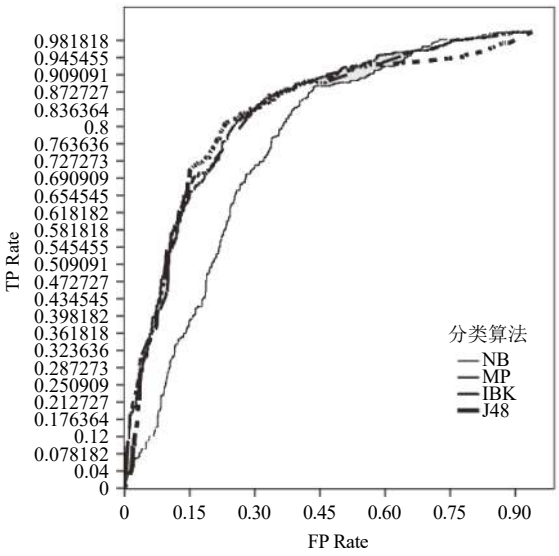

图 6 permissionInfoGain ROC 曲线

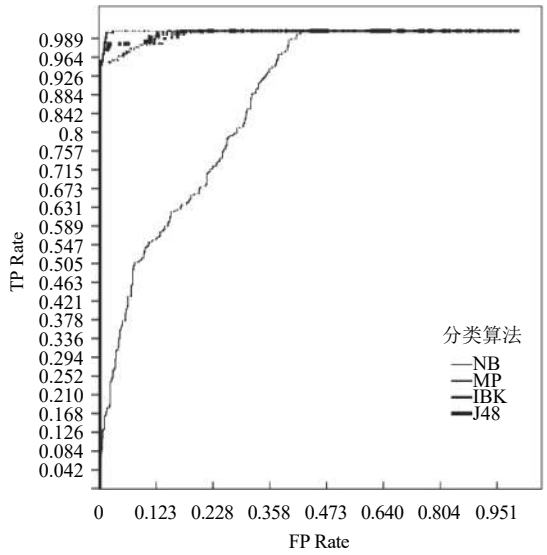

图 7 permission-API ROC 曲线

5 结论

当我们生活越来越依赖手机时, Android 平台安全问题也越来越受大众关注. 而现有的安全防护软件缺乏对未知恶意软件的检测, 故针对 Android 移动平台恶意软件进行快速有效的分析检测成为当务之急, 本文提出的将权限和 API 特征结合起来对 APK 进行静

态检测过程只是作为设计与开发的安卓恶意软件防御与检测系统中的一部分, 后续会将这部分功能嵌入至自己开发的系统中使其能实现完整的检测与防御功能, 使恶意行为能防患于未然. 但是本文提出的方案还存在一些待改进的地方: 如在特征集预处理过程中利用多种预处理算法选取子特征集后进然后比较各分类器的准确率, 样本集数量不够多也不够均衡, 如果收集更多的样本进行训练可能会得到更好的分类效果. 后续将会在更大的样本空间及提取不同的静态特征上进行检测研究, 提高分类器的检测率及准确率.

参考文献

1 艾媒咨询. 2017 上半年中国手机安全市场研究报告. http://www.iimedia.cn/57234.html. [2017-09-27].

2 王菲飞. 基于 Android 平台的手机恶意代码检测与防护技术研究[硕士学位论文]. 北京: 北京交通大学, 2012.

3 蔡昌. Android 平台恶意软件动态检测系统的设计与实现[硕士学位论文]. 北京: 北京交通大学, 2013.

4 张锐, 杨吉云. 基于权限相关性的 Android 恶意软件检测. 计算机应用, 2014, 34(5): 1322–1325. [doi: 10.11772/j.issn.1001-9081.2014.05.1322]

5 魏理豪, 艾解清, 邹洪, 等. Android 恶意软件的多特征协作决策检测方法. 计算机工程与应用, 2016, 52(20): 5–13. [doi: 10.3778/j.issn.1002-8331.1605-0171]

6 Hsieh WC, Wu CC, Kao YW. A study of Android malware detection technology evolution. International Carnahan Conference on Security Technology. Taipei, China. 2016. 135–140.

7 Tam K, Feizollah A, Anuar NB, *et al*. The evolution of Android malware and Android analysis techniques. ACM Computing Surveys, 2017, 49(4): 76.

8 Faruki P, Bharmal A, Laxmi V, *et al*. Android security: A survey of issues, malware penetration, and defenses. IEEE Communications Surveys & Tutorials, 2015, 17(2): 998–1022.

9 张巍, 任环, 张凯, 等. 基于移动软件行为大数据挖掘的恶意软件检测技术. 集成技术, 2016, 5(2): 29–40.

10 路程. Android 平台恶意软件检测系统的设计与实现[硕士学位论文]. 北京: 北京邮电大学, 2012

11 刘晓明. 基于 KNN 算法的 Android 应用异常检测技术研究[硕士学位论文]. 北京: 北京交通大学, 2016.

12 苏志达, 祝跃飞, 刘龙. 基于深度学习的安卓恶意应用检测. 计算机应用, 2017, 37(6): 1650–1656. [doi: 10.11772/j.issn.1001-9081.2017.06.1650]

13 Wu S, Wang P, Li X, *et al*. Effective detection of Android malware based on the usage of data flow APIs and machine learning. Information &Software Technology, 2016, 75(C): 17–25.

14 Sahs J, Khan L. A machine learning approach to Android malware detection. European Intelligence and Security Informatics Conference. Odense, Denmark, 2012: 141–147. [doi: 10.1109/EISIC.2012.34]

15 王超. 基于机器学习分类算法的 Android 恶意软件检测系统[硕士学位论文]. 南京: 南京邮电大学, 2015.

16 徐欣, 程绍银, 蒋凡. 恶意软件动态分析云平台. 计算机系统应用, 2016, 25(3): 8–13.

17 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. 计算机科学, 2017, 44(4): 135–139. [doi: 10.11896/j.issn.1002-137X.2017.04.029]

18 张骁敏, 刘静, 庄俊玺, 等. 基于权限与行为的 Android 恶意软件检测研究. 网络与信息安全学报, 2017, 3(3): 51–57. [doi: 10.11959/j.issn.2096-109x.2017.00159]