多维敏感特征的 Android 恶意应用检测

谢念念[1] 曾凡平[1,2] 周明松[1] 秦晓霞[1] 吕成成[1] 陈 钊[1]
(中国科学技术大学计算机科学与技术学院 合肥 230026)[1]
(安徽省计算与通讯软件重点实验室 合肥 230026)[2]

摘 要 应用程序的行为语义在 Android 恶意应用检测中起着关键作用。为了区分应用的行为语义,文中提出适合用于 Android 恶意应用检测的特征和方法。首先定义广义敏感 API,强调要考虑广义敏感 API 的触发点是否与 UI 事件相关,并且要结合应用实际使用的权限。该方法将广义敏感 API 及其触发点抽象为语义特征,将应用实际使用的权限作为语法特征,再利用机器学习分类方法自动检测应用是否具有恶意性。在 13226 个样本上进行了对比实验,实验结果表明,该方法的分析速度快且开销小,选取的特征集使 Android 恶意应用检测得到很好的结果;经机器学习分类技术的比较,我们选择随机森林作为检测方案中的分类技术,所提特征策略的分类准确率达到 96.5%,*AUC* 达到 0.99,恶意应用的分类精度达到 98.8%。

关键词 安卓恶意应用检测,静态分析,语法特征,语义特征,机器学习

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.02.015

Android Malware Detection with Multi-dimensional Sensitive Features

XIE Nian-nian[1] ZENG Fan-ping[1,2] ZHOU Ming-song[1] QIN Xiao-xia[1] LV Cheng-cheng[1] CHEN Zhao[1]
(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China)[1]
(Anhui Province Key Lab of Software in Computing and Communication, Hefei 230026, China)[2]

Abstract The behavior semantics of applications play a key role in Android malware detection. In order to distinguish the behavior semantics of applications, this paper presented suitable features and method for Android malware detection. This paper first defined the generalizdd-sensitive API, and emphasized to consider whether the trigger point of the generalized-sensitive API is UI-related as well as combined the really-used permission. The approach first abstracts the generalized-sensitive API and their trigger points as the semantic feature, extracts the really-used permission as the syntax feature, and then leverages machine learning-based classification method to automatically detect whether the application is benign or malicious. Comparative experiments were conducted on 13226 samples. The experimental results show that the proposed approach costs little time and the feature set is reasonable, and it can get good classification results. Through the comparison of several machine learning-based techniques, Random Forest is chosen as the classification method, and the results demanstrate that the accuracy achieves 96.5%, *AUC* reaches 0.99, and a classification precision of malware reaches 98.8%.

Keywords Android malware detection, Static analysis, Syntax feature, Semantic feature, Machine learning

1 引言

Android 设备在全球手机市场的占有率已经高达 87.7%。极高的市场占有率使得 Android 第三方应用市场不断扩大,Android 已经成为恶意应用开发者的主要攻击目标。由于恶意应用已经严重威胁到用户的个人隐私和财产安全,因此 Android 应用的安全问题亟需解决。Android 系统的权限机制规定应用必须显式地在清单文件 AndroidManifest.xml 中申请权限才能够访问相关资源。然而,开发人员可能会为应用申请过多的权限;恶意应用开发者也可能申请许多无关的正常权限以隐藏自己的恶意性;恶意应用甚至可以进行权限提升攻击,利用其他应用的权限访问敏感资源。因此,

到稿日期:2018-01-18 返修日期:2018-04-12 本文受国家自然科学基金项目(61772487)资助。
谢念念(1993-),女,硕士生,主要研究方向为信息安全;**曾凡平**(1967-),男,博士,副教授,主要研究方向为信息安全,E-mail:billzeng@ustc.edu.cn(通信作者);**周明松**(1994-),男,硕士生,主要研究方向为信息安全;**秦晓霞**(1994-),女,硕士生,主要研究方向为信息安全;**吕成成**(1994-),男,硕士生,主要研究方向为信息安全;**陈 钊**(1995-),男,硕士生,主要研究方向为信息安全。

Android 权限机制并不能完全避免恶意应用，仅仅通过申请的权限判断应用行为的安全性也是不可靠的。

检测 Android 恶意应用的技术大致可分为 3 类：1）动态分析[1-3]需要运行应用，对实时性要求高，时间和资源消耗较多，且无法保证代码覆盖率；2）静态分析[4-5]不需要运行应用，能耗低且代码覆盖率较高，但是无法从动态加载代码和混淆代码中抽取特征；3）基于机器学习[6-7]的检测方法主要通过大量的数据集训练出最优分类模型，此类方法的关键在于选择合适的特征集，主要有权限、API 方法及行为语义特征。Drebin[6]选取了大量样本进行实验，然而它仅考虑了权限、API 等语法特征，并没有考虑到应用的语义特征。权限和 API 固然是应用行为的基础，但是如果仅提取权限和 API 作为特征，检测结果是不够准确的，因为不同触发条件调用的同一个 API 行为的安全性可能是不同的。正常应用和恶意应用之间的主要区别在于后者往往在用户不知情的情况下操作敏感 API 进行恶意行为。因此，我们需要针对应用的行为语义进行分析。AppContext[7]通过找到敏感行为的激发事件和环境条件构成语义特征，以分析该行为是否是恶意操作。他们提取应用的敏感行为构造出训练集，其方法虽然粒度细，但实现的过程较为复杂。由于恶意应用演变速度快，基于机器学习的检测方法需要新数据集重复训练分类模型，但该方法计算开销大，可扩展性不强。受此启发，本文选择相对粗粒度但开销较小的方法来检测恶意应用。

为了克服以上方法的不足，更加准确地反映出应用的行为，文中通过静态分析提取应用的语义特征和语法特征，并利用机器学习方法对 Android 应用进行自动分类。本文定义广义敏感 API，包括权限保护的敏感 API 以及两类“近似敏感 API”（静态分析无法处理的反射和动态加载代码、代码混淆以及第三方库相关的 API 等，涉及敏感数据的 Source/Sink 方法）；并将三类敏感 API 与对应的触发点抽象表示构成语义特征集。语法特征集则是由应用实际使用的权限构成，包括两类：1）应用代码中敏感 API 对应的权限；2）对于应用采用动态加载代码等方式导致代码中找不到相关敏感 API 的情况，本文根据近似敏感 API 的功能和类名定义其对应的“近似权限”。

本文的主要贡献如下：

（1）提出一种新的能够有效检测应用安全性的语义特征：“广义敏感 API”及其触发点；使用应用实际使用的权限构成语法特征，综合考虑语法和语义特征检测应用。

（2）提出并实现新的 Android 恶意应用检测系统，实验结果显示所提特征提取策略高效且计算开销小，分类准确度为 96.5%，恶意应用的分类精度为 98.8%。

本文第 2 节介绍 Android 恶意应用检测的相关工作；第 3 节具体定义两类特征，并详细介绍特征提取和应用分类方法；第 4 节展示实验过程并分析结果；最后总结全文，分析不足之处并提出未来工作。

2 相关工作

现有的通过静态分析对 Android 恶意应用检测的研究工作一般与特征匹配或机器学习相结合，其中常用的特征大致可分为 3 类：

（1）清单文件中申请的权限[8-10]。这类方法过于表面，因为应用普遍存在过分申请权限的现象。恶意应用还可能通过权限提升完成恶意行为，单纯依靠申请的权限检测应用存在较大的误报率。

（2）API 等息。Cen 等[11]提取 API 方法作为特征并加入权限特征进行实验对比，发现结合 API 和权限的特征分类效果更好。Aafer 等[12]的方法与 Yerima 等[13]的方法类似，根据权限和 API 特征的出现频率构造特征集。Arp 等[6]通过提取权限、组件、intent 和 API 等 8 类特征，结合 SVM 算法实现了一个轻量级的检测工具 Drebin。以上这些工作都没有考虑到应用的语义特征，而语法特征并不能够抵抗代码混淆等技术。

（3）图类信息或敏感操作的上下文情况等语义特征。工具 Apposcopy[14]的实现原理与 DroidSIFT[15]的实现原理类似，通过构建应用的组件间调用图，利用特征匹配的方式检测应用。然而这种方式无法检测出新型恶意应用，结合语义特征与机器学习的方法检测恶意应用更加可靠。同样是利用图作为特征，Gascon 等[16]则是用 SVM 算法训练出分类模型进行检测。Yang 等[7]发现恶意应用为了不让用户发现其恶意行为，常在进行敏感行为之前检测时间、位置等系统环境。他们实现了 AppContext，通过构造过程间控制流图定位到敏感行为、激活事件和环境因素，形成上下文特征集训练分类器，最终检测恶意行为和恶意应用的准确率分别为 84.8% 和 95.0%。由于训练集样本是单个敏感行为，细粒度但构造过程复杂，因此该方法计算开销大，可扩展性不强；并且，一旦恶意行为出现误报，就会影响该应用的判断结果。文献[17]的方法与 AppContext 类似，准确率高，区别在于训练集样本是应用且构建的是组件间函数调用图。该工作同样存在恶意样本数量少的问题，并且敏感 API 考虑得不充分。

3 方法

3.1 恶意应用例子

图 1 是一个恶意应用的 AndroidManifest.xml 清单文件的关键代码段，该应用的功能是窃取短信内容并发送出去。如图 1 所示，该恶意应用在 SmsReceiver 组件中订阅了接收短信的广播事件，并设置最高优先级，一旦用户设备接收到短信，该广播就会激活 SmsReceiver.onReceive() 系统回调方法执行。然后，该应用解析收到的短信内容，并发送到开发者设定的手机号码，达到窃取短信的目的。其中，发送短信方法 SmsManager.sendTextMessage() 是敏感 API，因此需要在该应用的清单文件中申请对应的敏感权限 android.permission.SEND_SMS，如图 1 所示。该过程的简化调用图如图 2 中

M-①分支所示。这里的发送短信功能由 onReceive()方法触发调用。然而，正常的发送短信事件应该由用户点击发送短信按钮触发，即 onClick()之类的 UI 事件回调方法，短信内容及短信接收者也都应该从用户界面的编辑框内获取，该过程的简化调用图如图 2 中 B 分支所示。因此，我们需要关注敏感 API 的触发点是否是 UI 相关事件。另外，恶意应用常会以动态加载代码或加载第三方库的形式进行恶意行为，如图 2 中 M-②所示，通过 System. loadlibrary()加载第三方动态链接库，具体的发送短信操作在库中用 C++实现，静态分析就无法分析到其中的敏感 API。因此，考虑将动态加载代码、加载第三方库等相关 API 以及涉及敏感数据的 Source/Sink 方法也当作敏感 API 处理。为了考虑得更加全面，我们还需要通过代码中的敏感 API 找到应用实际使用的权限，并为近似敏感 API 定义相关“权限”。

```
〈application〉
  〈activity android:name=". MainActivity"〉
    〈intent-filter〉
      〈action android:name="android. intent. action. MAIN"/〉
      〈category
      android:name="android. intent. category. LAUNCHER"/〉
    〈/intent-filter〉
  〈/activity〉
  〈receiver android:name=". SmsReceiver"
    〈intent-filter android:priority="1000"〉
      〈！ ——订阅广播事件类型——〉
      〈action android:name="android. provider. Telephony. SMS_RECEIVED"/〉
    〈/intent-filter〉
  〈/receiver〉
〈/application〉
〈！ ——发短信的权限——〉
〈uses-permission android:name="android. permission. SEND_SMS"/〉
```

图 1　恶意应用的 AndroidManifest. xml 的关键代码段

Fig. 1　Core code snippet of malware's AndroidManifest. xml

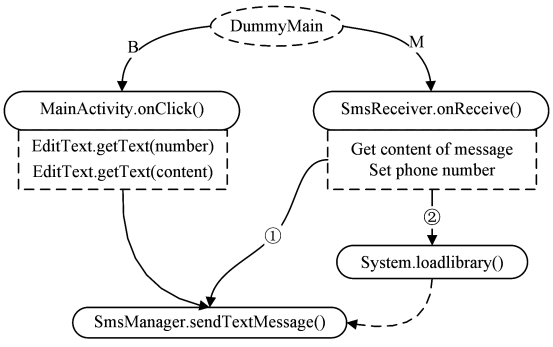


图 2　正常应用与恶意应用的简化调用图

Fig. 2　Simplified CG of benign and malicious

3.2　方法概述

本文方法的整体流程如图 3 所示，由特征提取、分类模型的训练和待测应用分类 3 个步骤组成。首先，借助工具 FlowDroid[3] 生成样本应用的函数调用图，通过函数调用图提取出训练集的两类特征（语义特征和语法特征），形成特征向量矩阵；然后，利用训练集数据对机器学习分类算法进行训练，确定特征集和最优分类模型；对于待测应用，根据特征集提取出特征向量，并将之输入到确定的分类模型进行检测以得到分类结果。

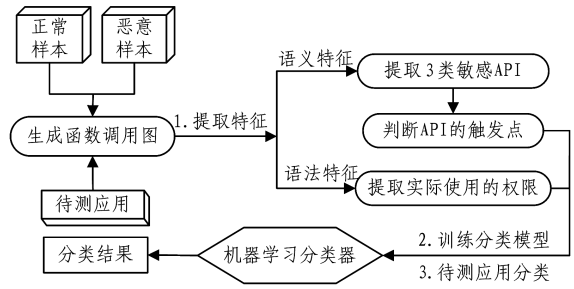


图 3　本文方法流程图

Fig. 3　Flowchart of proposed method

换言之，所提方法包含提取特征模块和分类器模块。下面根据这两个模块介绍本文方法。

3.3　提取特征模块

该模块考虑两类特征，包括语义特征和语法特征。

3.3.1　语义特征

语义特征具体由广义敏感 API G 和触发点 T 两部分构成。

(1)广义敏感 API G

本文定义广义敏感 API 包括以下 3 部分：

1)敏感权限保护的 API 方法

这些方法访问敏感资源，涉及用户隐私和设备的信息安全。PScout[18] 给出了 Android 系统中权限和对应敏感 API 的匹配结果。本文汇总 5 个 Android 版本下 PScout 的结果，使权限和 API 的对应关系更加完整；然后利用汇总结果提取出 59 个敏感权限及对应的敏感 API。

2)反射和动态加载代码等方法

许多恶意应用表面上不包含恶意代码，实际上会加载外部恶意代码进行恶意操作，从而躲避杀毒工具的检测。由于静态技术无法精确分析这些代码，其中又常会进行敏感操作，因此将它们看作敏感 API 处理。反射主要涉及类 Class，Constructor，Method 和 Field，动态加载代码主要涉及类 DexFile 和 ClassLoader[19]。另外，我们还考虑了恶意应用常会调用的其他方法，如加载第三方库的方法 System. loadLibrary()、用于代码混淆的类 Crypto. Cipher 中的常用方法及执行外部命令的方法 Runtime. exec()等。

3)Source/Sink 方法

数据流中的 Source/Sink 方法同样涉及敏感数据，它们和权限保护的 API 不完全相等，因此需要单独考虑。类 Intent 中的方法就属于这种情况，如 Intent. getStringExtra()是从 intent 消息中获取数据的 Source 方法，但是不受权限保护。除此之外，还包括其他类的一些方法，如 URL，Activity 和 IO 等。SUSI[20] 统计出了 Android 系统中的 Source/Sink

方法，我们利用该结果提取出权限保护的 API 以外的 Source/Sink 方法。

本文将第 1 类权限保护的敏感 API 记作集合 A，第 2 类和第 3 类敏感 API 记作近似敏感 API 集合 $\tilde{A}$，即 $G=A\cup\tilde{A}$。最终收集到的广义敏感 API 共 10 815 个，其中第 1 类包含 10 476 个，第 2 类和第 3 类分别包含 290 个和 49 个。

(2)触发点 T

由于本文以敏感 API 是否由用户触发作为出发点考虑应用安全性，因此不需要用具体的方法名表示触发点。将触发点方法抽象成 *UIevent* 和 *NonUIevent*，分别表示 UI 相关事件和非 UI 相关事件，即 $T=\{UIevent, NonUIevent\}$。这样的处理还可以减少特征个数，降低特征冗余度。如图 2 中的方法 onClick()和 onReceive()在语义特征中分别用 *UIevent* 和 *NonUIevent* 表示。

当用户点击界面上的一个控件时，类 android.view 中的方法 onClick()就会被触发。因此要找到 UI 事件回调方法，就是要分析它的类是否是顶层 UI 相关接口的子类[15]。我们共统计出 88 个 UI 事件相关的触发点方法。

借助工具 FlowDroid[3] 和 PScout[18] 提取出应用的语义特征集，提取过程如算法 1 所示，具体解释如下：

第 2 行，给定应用 APK α，首先利用 FlowDroid 解析该 APK 生成应用的函数调用图 $cg=(N,E)$，接下来分析调用图中的节点得到敏感 API 及其对应的触发点集合。

第 4－12 行，找到调用图中所有的入口点方法及其可达的节点。第 4－7 行，对于调用图中的每一条边 $n_{src}\rightarrow n_{dest}\in E$，若该源节点 n_{src} 不是任何一条边的目的节点，也不是虚拟 main 函数，则该源节点就是入口点，加入入口点集合 $\breve{T}$，对应的目的节点 n_{dest} 加入 n_{src} 的可达节点集合 $n_{src}.DestNode$ 中。敏感 API 对应的触发点一定是入口点集合 $\breve{T}$ 的子集，因此接下来要找到 $\breve{T}$ 中所有可到达敏感 API 的节点以构成触发点集合。第 8－12 行，对于每一个入口点 $n_{src}\in\breve{T}$，从调用图中寻找它所有的可达节点，包括直接到达和间接到达的节点。即如果存在边 $n_{src}\rightarrow n_{dest}$ 和 $n_{src}\rightarrow n_{dest}\rightarrow n_{dest}$，那么 n_{dest} 和 $n_{dest'}$ 都加入 n_{src} 的可达节点集合 $n_{src}.DestNode$ 中，直至遍历到叶子结点终止循环。

第 13－22 行，找到所有的触发点及其对应的敏感 API 方法节点并形式化表示。第 13－14 行，对于每一个入口点 $n_{src}\in\breve{T}$，首先判断其对应的可达节点 g 是否属于敏感 API G，如果是则把该节点 n_{src} 加入触发点集合，再判断该触发点是否是 UI 相关事件。第 15－22 行，根据 n_{src} 是否是 UI 相关事件，分别把每一条触发点 n_{src} 及其可达的敏感 API g 的对应关系表示成 $UIevent\rightarrow g$ 或 $NonUIevent\rightarrow g$。

第 23 行，经过以上遍历过程，即可得到该样本 APK 的语义特征集 $S^{\{T\rightarrow G\}}$：触发点是否是 UI 相关事件→广义敏感 API。其中，触发点 T 用值 *UIevent* 或 *NonUIevent* 表示，每一个敏感 API $g\in G$ 用具体方法名表示，即方法签名〈类名：返回类型 方法名(参数 1 类型，参数 2 类型，…)〉。

如图 2 中的 3 种情况抽象表示成语义特征形式分别为(此处 API 用方法名简写)：*UIevent*→sendTextMessage()，*NonUIevent*→sendTextMessage()和 *NonUIevent*→loadLibrary()。最终收集到的语义特征集共包含 688 个特征项。

算法 1 语义特征集提取算法

输入：APK α，广义敏感 API 集合 $G=A\cup\tilde{A}$，UI 事件相关的触发点集合 U

输出：入口点集合 $\breve{T}$，语义特征集 $S^{\{T\rightarrow G\}}$

```
1. begin
2.   FlowDroid parse α→CG cg=(N,E)
3.   foreach n_src→n_dest∈E, n_src, n_dest∈N do
4.     if srcNode (n_src)=null & n_src∉DummyMain then
5.       //找到调用图中所有的入口点方法及其可达的目的节点
6.       T̆.addNode(n_src)
7.       n_src.DestNode.addNode(n_dest)
8.       while n_dest∉LeafNode do
9.         foreach n_dest→n_dest'∈ E, n_dest, n_dest'∈N do
10.            n_src.DestNode.addNode (n_dest')
11.        end
12.      end while
13.  //找到所有的触发点及其对应的敏感 API 方法节点并形式化表示
14.      foreach g∈n_src.DestNode & g∈G do
15.        if n_src∈U then
16.          S^{T→G}.addFeature (UIevent→g)
17.        else
18.          S^{T→G}.addFeature (NonUIevent→g)
19.        end if
20.      end
21.    end if
22.  end
23.  return S^{T→G}
24. end
```

3.3.2 语法特征

语法特征由应用实际使用的权限表示，分为以下两类：

(1)代码中敏感 API A 对应的权限

由于简单地提取应用申请的权限作为语法特征并不可靠，因此本文提取的是应用代码中敏感 API 对应的权限。我们根据 PScout 的结果汇总出每个 API 与权限(包括敏感权限和普通权限)的对应关系，共有 24147 条匹配结果。对于每个 APK，遍历它的函数调用图中所有权限保护的 API，与 PScout 的结果进行匹配，从而提取出对应的权限。如图 2 中的敏感 API sendTextMessage()，我们用 *SEND_SMS* 表示其对应的权限。

(2)代码中近似敏感 API $\tilde{A}$ 对应的“权限”

由于恶意应用常会利用代码混淆、动态加载代码或加载第三方库的形式进行恶意行为，静态分析无法从代码中分析

到相关的敏感 API，也就会遗漏这部分实际使用的权限，因此本文还考虑加上近似敏感 API 的“近似权限”。由于 SUSI 中给出的 Source/Sink 的类别不完整，因此我们没有用 SUSI 中规定的 Source/Sink 类别，而是根据两类近似敏感 API 的功能和类名，手动地统计并定义对应的权限名，如图 2 中的近似敏感 API loadlibrary()，我们用 *LoadLibrary* 表示其权限。

以上两类实际使用的权限共同构成语法特征集，$S^{\{P\}}$ 共包含 57 个特征项，第一类敏感权限和第二类近似权限的特征元素数量分别为 41 和 16。

综上，汇总样本集中所有 APK 的特征集，广义敏感 API G 及其触发点 T 构成语义特征集 $S^{\{T\to G\}}$，应用代码中实际使用的权限 P 构成语法特征集 $S^{\{P\}}$，则特征集为 $S=S^{\{T\to G\}}\cup S^{\{P\}}$，共 745 个特征项。

3.4　分类器模块

由于没有绝对最优的机器学习分类算法，因此我们利用特征提取模块得到的特征集，选择经典的机器学习分类技术进行实验以训练出最优分类模型，包括决策树 C4.8、随机森林 RF、支持向量机 SVM、K-最近邻 KNN 和反向传播神经网络 BPNN。

4　实验评估

4.1　实验环境

实验在处理器为 Intel® Core™ i5-4460 3.20GHz、内存为 16GB、操作系统为 Windows 10 的机器上完成。基于工具 FlowDroid 的代码以及 PScout 和 SUSI 的结果，我们在 JDK 8.0 和 Eclipse Mars 开发环境下，通过 Java 语言实现应用特征的提取以及所有的数据分析过程。机器学习分类器使用工具 Weka[20] 3.8.0 版本实现。

4.2　数据集

为了避免数据集的分析时间过长，剔除了分析时间超过 10min 的 APK，最终共选取了 13 226 个样本进行实验。其中，正常样本由网站 AndroZoo[1] 上的 6 219 个 Google Play 应用构成；7 007 个恶意样本分别有 1 661 个来自 Genome 恶意应用库[22]，5 346 个来自 VirusShare 恶意应用库[2]。AndroZoo 上的应用都被几十个病毒检测软件分析过，从该网站获得的正常样本可以保证是安全的。实验中对 APK 进行分析提取出特征向量的平均用时约为 78s，最快的仅需几秒，具体分析时间受应用大小影响。

4.3　评估标准

本文采用的评估分类器性能的标准如下：

准确度 $Accuracy=(TP+TN)/(TP+TN+FP+FN)$

误报率 $FPR=FP/(FP+TP)$

漏报率 $FNR=FN/(FN+TP)$

召回率 $Recall$（真正例率 TPR）$=TP/(TP+FN)$

精度 $Precision=TP/(TP+FP)$

由于召回率 R 和精度 P 往往是矛盾的，因此我们利用 $F1$ 值进行综合评估：$F1=2*P*R/(P+R)$。

ROC 曲线反映了分类器的泛化性能，分别以 TPR 为纵轴，FPR 为横轴，AUC 表示 ROC 曲线的面积：$AUC=Area$（ROC）。AUC 值越大，分类器性能越优。

其中，T 表示正确分类，F 表示错误分类，正样本 P 为恶意应用，负样本 N 为正常应用。下面的分析结果中评估指标的前缀 $P_$ 和 $N_$ 分别表示正样本和负样本。

4.4　实验结果及分析

通过对所有样本进行特征提取，最终形成了 745 维的特征集，其中语义和语法特征集分别包含 688 和 57 个元素。在应用的特征向量中，特征项出现的特征值用 1 表示，未出现的特征值用 0 表示。我们通过十折交叉验证实验方法评估本文特征及方法的性能，并提出以下 3 个研究问题：

问题 1：最适合本文特征集的机器学习分类算法是什么？使用本文方法检测 Android 恶意应用的整体性能如何？

问题 2：本文定义的语义特征中的广义敏感 API 对 Android 应用分类的结果有何影响？特征集及其表示形式是否合理？

问题 3：与恶意应用检测工具 Drebin 相比，本文方法的分类结果如何？

4.4.1　问题 1：整体性能

借助工具 Weka，尝试了 5 个机器学习分类算法以寻找最适合本文特征集的分类技术，包括决策树 C4.8、随机森林 RF、支持向量机 SVM、K-最近邻 KNN 和反向传播神经网络 BPNN。对比结果如表 1 所列。

表 1　分类算法的对比结果

Table 1　Comparative results of classification algorithms

Indicator	RF	SVM	C4.8	KNN	BPNN
Accuracy/%	96.5	93.4	94.6	96.0	92.2
FNR/%	5.5	10.0	6.1	5.1	12.5
FPR/%	1.3	2.9	4.5	2.8	3.0
P_Presscision/%	98.8	97.3	95.9	97.4	96.7
P_Recall/%	94.5	90.0	93.9	94.9	87.5
P_F1/%	96.6	93.5	94.9	96.2	91.9
N_Precision/%	94.1	89.6	93.3	94.5	88.5
N_Recall/%	98.7	97.1	95.5	97.2	97.0
N_F1/%	96.6	93.2	94.4	95.8	92.6
AUC	0.99	0.94	0.95	0.97	0.95

表 1 结果显示，随机森林算法 RF 的分类结果最好，准确度为 96.5%，AUC 为 0.99，正常应用和恶意应用的 $F1$ 值都是最高的，误报率和漏报率也都是最低的。RF 本身就适合于属性多的数据集，并且在决策树的训练过程中引入了随机属性选择，使得随机森林中的决策树都能够彼此不同，提升了系统的多样性，从而提升分类性能。

4.4.2　问题 2：特征集表示形式

为了评估特征集的有效性，我们进行了 3 组对比实验。

第 1 组实验主要探讨广义敏感 API 的选取是否合理，结果如表 2 所列。将 3 类敏感 API 分别用 $G1$，$G2$，$G3$ 表示，利

1) https://androzoo.uni.lu,2016

2) https://virusshare.com

用 RF 算法和 3 个特征集 $S^{\{T\to G1\}}+S^{\{P\}}$，$S^{\{T\to G1+G2\}}+S^{\{P\}}$，$S^{\{T\to G1+G2+G3\}}+S^{\{P\}}$ 进行对比实验，在表 2 中分别用 $G1+P$，$G1+G2+P$，$G1+G2+G3+P$ 表示。结果显示，当敏感 API 为 $G1+G2+G3+P$ 时的分类结果最好，即第 2 类和第 3 类敏感 API 对于 Android 应用的分类过程是重要的，我们定义 3 类敏感 API 是合理的。

表 2 不同敏感 API 的对比结果

Table 2 Comparative results of different sensitive APIs

（单位：%）

Indicator	$G1+G2+G3+P$	$G1+G2+P$	$G1+P$
Accuracy	96.5	95.1	92.9
P_Precision	98.8	97.4	97.0
P_Recall	94.5	93.3	89.5
P_F1	96.6	95.3	93.1
N_Precision	94.1	92.8	89.1
N_Recall	98.7	97.2	96.9
N_F1	96.6	95.0	92.8

第 2 组实验探讨特征集的选取是否合理。我们利用 Android SDK 中的 AAPT 工具提取出应用申请的权限，并且仅结合敏感权限保护的 API 构成特征集进行对比实验，结果如表 3 中的 $RP+API$ 列所示。与 $G1+G2+G3+P$ 列的结果相比，结合 3 类敏感 API 及对应触发点作为语义特征，加上实际使用的权限作为语法特征的结果更好。因此，我们的特征集的选取及定义方法能够更好地帮助检测出 Android 恶意应用。

表 3 不同特征集的对比结果

Table 3 Comparative results of different feature sets

（单位：%）

Indicator	$G1+G2+G3+P$	$RP+API$
Accuracy	96.5	94.5
P_Precision	98.8	96.0
P_Recall	94.5	93.4
P_F1	96.6	94.7
N_Precision	94.1	92.8
N_Recall	98.7	95.7
N_F1	96.6	94.2

第 3 组实验探讨特征集中语义特征的表示形式对实验结果的影响，如表 4 所列。第 1 组实验中的初始特征集 $G1+G2+G3+P$ 在表 4 中用 $G+P$ 表示。

我们将广义敏感 API 粗粒度地用对应权限表示，代替具体的方法签名，该特征集用 $G+P_C$ 表示。另外，我们考虑到敏感 API 用粗粒度的方式表示可能会导致特征细节丢失，所以我们又加入了一组特征集，用 $G+P_CS$ 表示，其中语义特征部分的特征值用出现次数表示。表 4 中的结果显示，敏感 API 用粗粒度的权限表示时，无论特征向量中特征值用二元表示还是频次表示，分类结果均没有原表示形式的分类结果好。

DroidAPIMiner[12] 筛选出恶意样本中出现频次更多的 API 形成特征集并得到了 99% 的准确度。因此第 3 组实验中，我们还借鉴该方法对特征集预先做了特征选择，只保留恶意应用中出现次数多于正常应用的特征以形成特征集，从表 4 中可以看到，$G+P_Dif$ 列中每一项评估指标都略低于 $G+P$ 列。因为这样简单地对特征集进行缩减，可能会丢失一些相关的特征信息，正常应用中出现而恶意应用中不出现的特征同样可以用于帮助区分应用。

表 4 不同特征表示形式的对比结果

Table 4 Comparative results of different feature representations

（单位：%）

Indicator	$G+P$	$G+P_C$	$G+P_CS$	$G+P_Dif$
Accuracy	96.5	95.4	96.0	96.3
P_Precision	98.8	97.0	98.0	98.6
P_Recall	94.5	94.1	94.4	94.3
P_F1	96.6	95.6	96.2	96.4
N_Precision	94.1	93.6	94.0	93.9
N_Recall	98.7	96.7	97.9	98.5
N_F1	96.6	95.1	95.9	96.1

以上 3 部分实验都印证了本文特征集的选取以及特征向量的表示形式对于 Android 恶意应用检测的有效性。

4.4.3 问题 3：相关方法比较

Drebin[6] 是一个轻量级的 Android 恶意应用自动检测工具。它主要是通过静态分析提取应用的 8 类特征，包括权限、intent、组件、硬件、敏感及可疑 API 等，再通过训练出 SVM 分类模型来检测恶意应用。

我们用本文方法分析了 Drebin 的数据集，共 5560 个恶意应用，结合 6219 个 GooglePlay 的正常应用，分别用 RF 和 SVM 训练出新的分类模型。实验结果中得到的整体分类准确度分别为 97.7% 和 95.8%，关于恶意应用的检测准确度 P_Recall 的对比结果如表 5 所列。

表 5 所提方法与 Drebin 的对比结果

Table 5 Comparative results between proposed method and Drebin

（单位：%）

Indicator	Drebin	本文方法+RF	本文方法+SVM
P_Recall	93.9	96.2	94.8

由表 5 可以看出，本文选取特征的方法加上随机森林 RF 训练出的分类模型，检测 Android 恶意应用的准确度最高，达到 96.2%。为了避免机器学习分类技术对分类结果的影响，同样采用了 Drebin 使用的支持向量机 SVM 进行了实验，得到的恶意应用分类准确率为 94.8%，依然高于 Drebin 的结果 93.9%。实验结果表明，本文选取特征的方法合理有效，结合 API 及其触发点作为特征比单独使用 API 作为特征进行应用分类的结果更好。

结束语 文中提出了一种新的开销较小的特征表示形式，综合考虑了应用的语法和语义特征，并利用机器学习方法检测恶意应用。实验共分析了 13226 个样本，应用的平均分析用时为 78s。评估结果显示，本文特征集的抽象和特征向量的表示使 Android 恶意应用检测得到了很好的结果，整体的分类准确度为 96.5%，恶意应用的分类精度为 98.8%。

由于本文方法的基础是静态分析，不能对反射和动态加载代码方法做精确处理，因此会产生漏报和误报。此外，PScout 也是基于静态分析的，不能捕捉到反射调用相关的所有 API，所以找到的敏感 API 可能并不完整。然而，我们综合了 PScout 多个版本的结果，考虑的敏感 API 也不仅限于权限

保护的 API，最终选择的广义敏感 API 是合理的，一定程度上弥补了静态分析的缺陷。

未来我们将会在以下 3 个方面做一些工作：1）结合动态分析检测应用，提高分类准确度和精度。2）考虑有关第三方库，使选择的特征集更加完善；去除仅在第三方库中出现的可疑行为，避免正常应用因第三方库中的恶意行为而被误判为恶意应用[12]。3）尝试更多经过优化的分类算法，以探索更精确的分类结果。

参 考 文 献

[1] ENCK W, GILBERT P, HAN S, et al. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones[J]. ACM Transactions on Computer Systems (TOCS), 2014, 32(2): 1-29.

[2] WEI F, ROY S, OU X. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps[C]// Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2014: 1329-1341.

[3] YAN L K, YIN H. DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis[C]// USENIX Security Symposium. 2012: 569-584.

[4] ARZT S, RASTHOFER S, FRITZ C, et al. FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps[J]. Acm Sigplan Notices, 2014, 49(6): 259-269.

[5] LI L, BARTEL A, BISSYANDE T F, et al. Iccta: Detecting inter-component privacy leaks in android apps[C]// Proceedings of the 37th International Conference on Software Engineering-Volume 1. IEEE Press, 2015: 280-291.

[6] ARP D, SPREITZENBARTH M, HUBNER M, et al. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket[C]// Network and Distributed System Security Symposium. 2014.

[7] YANG W, XIAO X, ANDOW B, et al. AppContext: Differentiating malicious and benign mobile app behaviors using context [C]// IEEE/ACM, IEEE International Conference on Software Engineering. IEEE, 2015: 303-313.

[8] PENG H, GATES C, SARMA B, et al. Using probabilistic generative models for ranking risks of android apps[C]// Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012: 241-252.

[9] MOONSAMY V, RONG J, LIU S. Mining permission patterns for contrasting clean and malicious android applications[J]. Future Generation Computer Systems, 2014, 36: 122-132.

[10] TALHA K A, ALPER D I, AYDIN C. APK Auditor: Permission-based Android malware detection system[J]. Digital Investigation, 2015, 13: 1-14.

[11] CEN L, GATES C S, SI L, et al. A probabilistic discriminative model for android malware detection with decompiled source code[J]. IEEE Transactions on Dependable and Secure Computing, 2015, 12(4): 400-412.

[12] AAFER Y, DU W, YIN H. DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android[C]// International Conference on Security and Privacy in Communication Systems. Springer International Publishing, 2013: 86-103.

[13] YERIMA S Y, SEZER S, MCWILLIAMS G, et al. A new android malware detection approach using bayesian classification [C]// 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA). IEEE, 2013: 121-128.

[14] FENG Y, ANAND S, DILLIG I, et al. Apposcopy: semantics-based detection of Android malware through static analysis[C]// ACM Sigsoft International Symposium on Foundations of Software Engineering. ACM, 2014: 576-587.

[15] ZHANG M, DUAN Y, YIN H, et al. Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs[C]// ACM. 2014: 1105-1116.

[16] GASCON H, YAMAGUCHI F, ARP D, et al. Structural detection of android malware using embedded call graphs[C]// ACM Workshop on Artificial Intelligence and Security. ACM, 2013: 45-54.

[17] MIAO X C, WANG R, XU L, et al. Security Analysis for Android Applications Using Sensitive Path Identification [J]. Journal of Software, 2017, 28(9): 2248-2263. (in Chinese)
缪小川，汪睿，许蕾，等. 使用敏感路径识别方法分析安卓应用安全性[J]. 软件学报，2017，28(9)：2248-2263.

[18] AU K W Y, ZHOU Y F, HUANG Z, et al. PScout: analyzing the Android permission specification[C]// ACM Conference on Computer and Communications Security. ACM, 2012: 217-228.

[19] LI L, OCTEAU D, KLEIN J. DroidRA: taming reflection to support whole-program analysis of Android apps[C]// International Symposium on Software Testing and Analysis. ACM, 2016: 318-329.

[20] FRANK E, HALL M, HOLMES G, et al. Weka-a machine learning workbench for data mining[M]// Data Mining and Knowledge Discovery Handbook. Springer, Boston, MA, 2009: 1269-1277.

[21] RASTHOFER S, ARZT S, BODDEN E. A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks[C]// Proc. NDSS, 2014.

[22] ZHOU Y, JIANG X. Dissecting Android Malware: Characterization and Evolution[C]// IEEE Symposium on Security and Privacy. IEEE Computer Society, 2012: 95-109.