doi :10.3969/j.issn.1671-1122.2016.08.012

基于 *N*–gram 算法的恶意程序检测系统研究与设计

张家旺，李燕伟
（国家计算机网络应急技术处理协调中心，北京 100029）

摘 要：文章针对恶意程序检测中难以检测未知恶意程序等问题，提出了一种提取恶意程序语义特征的方法。该方法使用 *N*–gram 算法对提取的 Android 应用程序的权限和 API 特征建立语义特征序列，并对特征序列进行筛选处理，获得了更具代表性的行为特征序列。首先，为了增加特征的有效性，经验丰富的恶意程序分析专家为每个 Android SDK 中的 API 函数添加相应的权重，并使用出现频次和权重值重新计算 *N*–gram 序列中每个元素的特征值，从而构建了改进的 *N*–gram 序列模型。然后，使用多种机器学习算法进行分类检测，验证其有效性。实验结果表明，提取的特征及改进的 *N*–gram 算法可以有效检测 Android 平台上的恶意程序。

关键词：机器学习；恶意代码检测；*N*–gram；Android 应用
中图分类号：TP309 **文献标识码：**A **文章编号：**1671–1122（2016）08–0074–07

中文引用格式：张家旺，李燕伟 . 基于 *N*–gram 算法的恶意程序检测系统研究与设计 [J]. 信息网络安全，2016（8）：74–80.

英文引用格式：ZHANG Jiawang，LI Yanwei. Research and Design on Malware Detection System Based on *N*–gram Algorithm[J]. Netinfo Security, 2016 (8) : 74–80.

Research and Design on Malware Detection System Based on *N*-gram Algorithm

ZHANG Jiawang, LI Yanwei
(*National Computer Network Emergency Response Technical Team Coordination Center of China*, *Beijing* 100029, *China*)

Abstract: It is difficult to detect malware detection of unknown malicious programs, Aiming at solving this problem, this paper proposes an approach for extracting the dynamic features of malicious code semantics. This method extracts the permissions and API features of Android application to set up the semantic feature sequence with the *N*-gram algorithm. With screening of the feature sequence, the behavior sequence becomes more representative. First, in order to increase the effectiveness of the characteristics, analysis of experienced malware experts for each Android API function in SDK to add the corresponding weights, and the use of frequency and the weight value of each element of the *N*-gram sequence characteristics of re-calculated values in order to build a *N*-gram series model improved. Then, using a variety of machine learning algorithms for classification and detection, verify its effectiveness. The experimental results show that the improved *N*-gram algorithm and features in this paper can effectively detect malicious programs under Android platform.

Key words: machine learning; malicious code detection; *N*-gram; Android application

收稿日期：2016–06–10
基金项目：国家自然科学基金 [61402125]
作者简介：张家旺（1981—），男，北京，工程师，硕士，主要研究方向为恶意代码分析、漏洞发现与利用；李燕伟（1983—），男，河北，工程师，博士，主要研究方向为恶意代码分析、漏洞发现与利用。
通信作者：张家旺 16362505@qq.com

0 引言

随着移动互联网的发展，智能手机在生活中的地位不断提升，已成为必不可少的部分。由于智能手机涉及到个人生活的方方面面，具有一定的秘密性，因此安全问题显得尤为重要。根据阿里移动安全的分析统计[1]，2015年移动恶意代码数量和用户感染量虽然呈现出一定程度的下降趋势，但全年仍有18%的设备感染过病毒，且病毒木马攻击手法的专业性较2014年有了明显的提升。智能手机感染恶意程序不仅会造成用户经济财产的损失，同时也存在泄露用户重要信息的隐患。

目前Android系统已经成为智能手机的主要操作系统之一，据数据研究公司IDC报告显示，2015年全球Android系统OEM厂商的手机出货量将达到惊人的11.6亿台，这个数字至2019年将增长至15.4亿台，届时Android系统将占据全球82.6%的移动系统市场份额。考虑到庞大的经济利益，Android系统已经成为黑客攻击的主要目标，也是恶意代码程序与各杀毒软件的主要战场。以勒索软件为例，Android勒索软件可能造成的后果比在PC上更加严重，因为移动设备中存储了更多没有备份过的个人信息或者企业数据，由于这些数据的重要性，用户更倾向于支付费用。Bidifender发现的Android勒索软件家族Adnroid.Trojan.Slocker已经在2015年下半年攻击了德国、澳大利亚和英国的用户。在德国，超过33.58%的恶意软件报告都与其相关，在澳大利亚和英国也得出了相同的结论，30.25%和22.39%的报告与之相关。全球45.53%的勒索软件都来自美国，德国是全球第二大勒索软件国，总量占到32.87%。

恶意代码危害巨大，随着恶意代码数量、种类的不断增加，技术不断升级，使得基于签名或行为分析的传统分析方法成为短板。签名方法首先需要事先获得恶意代码样本，然后将签名信息应用于恶意程序的检查过程中，无法有效检查未知恶意代码。另外，基于行为的检测方法解决了静态分析时的代码混肴和加密的问题，但无法覆盖程序的全部有效路径，导致功能分析不够全面。故本文采用API函数调用和权限做为行为特征，结合多种机器学习算法建立模型进行检测。主要成果：1）建立了一种改进的*N*-gram模型，通过添加特征出现的频次和权值来生成更加有效的*N*-gram序列；2）提出了恶意程序行为特征分析检测框架；3）利用多种机器学习方法对*N*-gram特征进行分类检测，评价指标良好。

1 研究现状

恶意程序是Android系统面临的最大威胁，最早出现的是基于签名和特征码的检测方法，其缺点是无法检测未知恶意代码。CHRITODORESCU[2]等人开始使用静态分析技术检测恶意代码，SAFE可以有效处理大多数恶意代码采用的混淆技术，但检测时间较长。随着基于签名和特征码的静态检测方法的失效，越来越多的研究人员开始研究基于动态行为特征的检测方法。徐曾春[3]等人设计了一套沙箱系统，使用动态方法分析Android应用的系统调用、文件等信息，从而判断是否为恶意代码。

近年来大数据相关技术不断成熟，也有研究者开始将数据挖掘和机器学习的方法引入到恶意代码检测中。在判别基于Windows系统的恶意软件中，YE[4]等人使用API调用作为特征，从API调用中提取出特征，在此基础上使用数据挖掘技术，对含有12214个良性程序以及1736个恶意软件的数据集进行分析。李盟[5]等人提出了一种恶意代码特征选取和建模的方法。

针对Android恶意程序可以借助机器学习方法进行分类和检测[6-10]，杨欢等人设计了一套基于多个特征并采用三层混合系综算法（THEA）的检测系统；AMOS等人提出了STREAM框架，加速了Android恶意软件的机器学习分类的快速大规模验证；SAHS等人提出了基于机器学习的检测系统，提取大量特征并采用离线的方式训练一类支持向量机。

使用机器学习方法构成的分类器可以对Android应用行为进行模拟，区分应用是否为恶意软件。分类器通过分析应用的静态或动态行为特征做出判断，静态特征可以通过反编译Android Dalvik字节码获得，动态特征则通过监控程序运行时的行为来获取。通过提取申请的权限、反编译的代码或运行的行为，可以获取各种不同类型的特征。常用的机器学习方法包括*k*-Means、逻辑回归（Logistic Regression）、*K*-最近邻（*K*-nearest-neighbor，KNN）、直方图（Histogram）、决策树（Decision tree）、贝叶斯网络（Bayesian Network）、朴素贝叶斯（Naïve Bayes）、随机树（Random Tree）、随机森林（Random Forest）、支持向量机（Support

Vector Machine，SVM）等。

由于 Android 应用程序的特殊性，本文使用 DVM（Dalivk Virtual Machine）中的 API 和应用权限作为特征，应用 *N*-gram 算法建立特征集合，最后使用机器学习算法进行分类，获得了较好的分类与检测效果。

2 特征提取与算法改进

2.1特征选取

1）API

API 是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力，而又无需访问源码或理解内部工作机制的细节。在 Linux 中，用户编程接口 API 遵循了 UNIX 中最流行的应用编程界面标准 POSIX 标准。POSIX 标准是由 IEEE 和 ISO/IEC 共同开发的标准系统。该标准基于 UNIX 实践经验，描述了操作系统的 API，用于保证应用程序可以在源程序一级在多种操作系统上移植运行。这些系统 API 调用主要是通过 C 库（LIBC）来实现的。由于 Android 和 Linux 系统的同源性。使得 API 存在一定的一致性。API 是操作系统为应用程序提供的服务性接口，应用程序在完成文件读写、网络访问以及其他重要资源的访问时都会调用 API。早在 2004 年 XU[11] 等人就提出了应用程序的 API 调用大致可以反映程序的行为。而 YE[4] 等人通过更进一步的研究，在他们的论文中使用 API 调用作为特征，并利用数据挖掘的方法对数据进行分析。

2）权限特征

权限管理是 Android 系统中最重要的安全措施之一，所有应用程序对权限的申请和声明都被强制标识于 AndroidManifest.xml 文件之中，通过 <permission>、<permission-group>、<permission-tree> 等标签指定。需要申请某个权限，可以通过 <uses-permission> 指定。应用程序申请的权限在安装时提示给用户，用户可以根据自身需求和隐私保护的考量决定是否允许对该应用程序授权。

Android 系统通过权限来控制应用程序想要执行的功能，程序在默认情况下没有权限去进行特定的操作，如发送短信、访问地址位置、访问通信录等，想要进行这些操作就必须正式地申请该操作对应的权限。如果没有申请相应的权限而执行该功能时，应用程序会在运行时抛出一个 SecurityException 异常。申请权限的方法比较简单，只要在配置文件 AndroidManifest.xml 中加入相应的代码即可。例如，发送短信功能时，需要添加的内容如下：

<uses-permission android:name=”android.permission.SEND_SMS:/>

Android 系统具有非常严格的权限管理机制，应用程序实现某个功能或访问某些敏感数据时都要申请某个使用权限。例如，吸费类恶意应用会申请 SEND_SMS 权限发送短信。根据分析大量的恶意程序样本发现，存在恶意行为的应用往往会使用大量敏感权限，由此可见，权限的使用从一个方面反映了恶意程序的某些行为特征，因此把权限特征作为本文中提取的特征之一。

2.2 提取方法

1）提取权限特征

对应用程序自身 APK 文件进行解析分析，提取应用程序在配置文件中申请的敏感权限信息。Androguard[12] 是基于 Python 的 Android 恶意应用程序检测工具，可以提取大量敏感信息，其中 androapkinfo.py 模块分析并列出应用程序的文件类型、权限、4 大组件、是否 NDK 反射等信息。本文利用 androapkinfo.py 提取 APK 申请的敏感权限，加入权限特征列表。下面简单介绍 Androguard 的安装过程。

（1）从 Github 上下载其最新的源码，安装环境必须要有 Python，并且版本不能低于 2.6，且不能高于 3.0。

（2）安装基础依赖库。

```
sudo apt-get install python-dev python-bzutils libbz2-dev libmuparser-dev libsparsehash-dev python-ptrace python-pygments python-pydot graphviz liblzma-dev libsnappy-dev
sudo pip install traitlets
sudo pip install -U jupyter
```

（3）用 pip 来安装 magic 库。

```
sudo pip install python-magic
```

（4）源码安装 pyfuzzy 库。

源码下载，解压后进入其文件夹：

```
tar -zxf pyfuzzy-0.1.0.tar.gz
cd pyfuzzy-0.1.0
sudo python setup.py install
```

（5）安装 chilkat。

从其官网选择合适的版本下载，如 chilkat-9.5.0-python-2.7-x86_64-linux.tar.gz。

解压至：

sudo cp chilkat.py _chilkat.so /usr/local/lib/python2.7/dist-packages/

（6）从 Github 上下载 dorzer，将其文件夹修改为 dorzer，然后直接复制到 Androguard 源码根目录，进入其源码目录，直接执行 make 即可。

（7）修改 Androguard makefile。

打开 elsim/elsign/formula/Makefile 添加如下代码：

```
CFLAGS += -I/usr/include/muParser
```

打开 elsim/elsign/libelsign/Makefile 添加如下代码：

```
CFLAGS += -I/usr/include/muParser -I/usr/include/python2.7
```

（8）编译 Androguard，在源码根目录直接执行 make 即可。

2）提取 API 特征

由于 Android 系统调用的复杂性，本文结合静态和动态方法提取 API 特征，将静态和动态两种方式提取的特征分别加入静态 API 特征列表和动态 API 特征列表。

（1）静态方法

Android 基于 Linux 内核，在架构上分为 5 个部分，包括 Linux Kernel、Android Runtime、Libraries、Application Framework 和 Application。Android 应用程序以 APK(Android Package) 文件结构发布，APK 文件是一个压缩包，其结构如图 1 所示，重点关注 classes.dex，它是 DVM 字节码文件，可运行于 Android 虚拟机 Dalvik 上。

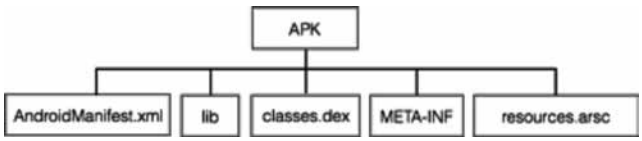

图1 APK文件结构

目前 DEX 可执行文件主流的反汇编工具有 Baksmali 与 Dedexer。本文提取 APK 应用的 API 调用是基于 DEX 的反汇编代码，将 DEX 文件反编译为 smali 文件，然后对每个 smali 文件进行解析分析，提取其中的 API 调用序列。为了去除无用调用，参考 MIN[13] 等人的做法，从 Android SDK 框架的 Android.jar 文件中提取全部 API 的 Class 路径和函数名称，并对每个 API 方法进行 hex 编码。最终形成 API 调用初始化表。例如，Android/accessibilityservice/AccessibilityService;-><init> 的编码为 0x0000，对比 Android SDK 中的 API 列表，删除无用调用，加入 API 特征列表，流程图如图 2 所示。

图2 静态API提取流程

（2）动态方法

为了更好地反映 API 调用特征，解决静态分析中的不足，本文使用 DroidBox[14] 工具箱中的 APIMonitor 进行动态监控，其原理是向目标 APK 中插入桩代码，实现对特定 API 的监控，将重新打包的程序在模拟器中直接运行，通过查看 logcat 中的 Tag 标记为“DroidBox”的信息，使用正则表达式提取其中有关 API 特征。另外，可以通过修改 config/default_api_collection 中默认监控的 API 列表来添加监控的函数。

基本命令如下。

运行 APIMonitor：

./apimonitor.py ./app.apk

运行模拟器：

emulator -avd Android4.3

安装 app 样本：

adb install ./app.apk

运行 app 样本：

adb shell am start -n 包 (package) 名 | / | 包名 | .{ 活动 (activity) 名称 }

提取 API 调用：

adb logcat -s DroidBox:V | grep pattern

流程图如图 3 所示。

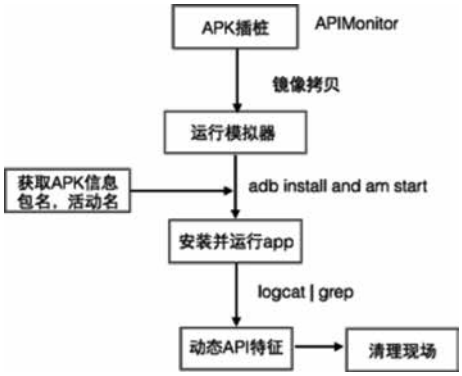

图3 动态API特征提取流程

2.3 *N*-gram算法

N-gram是自然语言处理领域的概念，是大词汇连续语音识别中常用的一种语言模型。早期的语音识别技术和统计语言模型与它密不可分。该模型基于这样一种假设，第*N*个词的出现只与前面*N*-1个词相关，而与其他任何词都不相关，整句的概率就是各个词出现概率的乘积。其切分算法的基本思想是将文本按字节流从第一个字符开始从左向右移动，每次移动一个字符单位，窗口中出现的*N*个字符即为一个*N*-gram单元。如果*N*=1，就是以一个汉字做为一个语言单元；*N*=2，即为双汉字单元。例如，当*N*=2时，对于句子“我们去北京看奥运会”的*N*-gram切分结果为“我们”、“们去”、“去北 ”、“北京”、“京看”、“看奥”、“奥运”、“运会”8个语言单元。这些概率可以通过直接从语料中统计*N*个词同时出现的次数得到，常用的是二元的Bi-Gram和三元的Tri-Gram。

N-gram已经被应用到众多领域，最早是应用于文本分析中，将一串数据转化为*N*-gram，可以将该数据表示为向量形式，从而能更好地与其他数据串进行比较。另外，还可以将两组数据串的*N*-gram分布进行比较，从而判断这两组数据的相似度。对于一段二进制程序*P*，它可以被表示为由*w*个字节*b*组成的一段序列，即$P=(b_1,b_2,b_3,\cdots,b_{w-1},b_w)$，一段字节*N*-gram序列*g*被定义为一个可执行文件中一小段连续字节，这里$g\in P$，并且$g=(b_1,b_2,b_3,\cdots,b_{N-1},b_N)$，其中*N*即为字节*N*-gram的长度。程序*P*可用N-gram表示为$P=(g_1,g_2,g_3,\cdots,g_{w-N},g_{w-N+1})$。类似于上面的中文语句，某个代码段的控制指令P=('ADC','LDR','LDR', 'ADC', 'LDR', 'MOV','ADC', 'LDR', 'MOV')可以产生的3-gram序列有如下7个：g_1=('ADC','LDR','LDR'), g_2=('LDR','LDR', 'ADC'), g_3 =('LDR', 'ADC', 'LDR'), g_4=('ADC', 'LDR', 'MOV'), g_5=('LDR', 'MOV','ADC'), g_6=('MOV','ADC', 'LDR'), g_7=('ADC', 'LDR', 'MOV')。

该算法应用于恶意代码识别的想法最早源于ASSALEH[15]等人提出的一种基于ByteCode的检测方法。2008年，MOSKOVITCH[16]等人提出利用OpCode代替ByteCode的检测方法，更加科学且检测效果更好。

文本基于androapkinfo.py工具提取的权限特征，权限存在的方式是0或1，因此处理该特征时仍然使用未改进的*N*-gram算法。由于敏感权限不会在一个APK中反复申请，并不需要使用高纬度的滑动窗口，因此使用滑动窗口*N*为1建立1-gram序列。例如，应用程序使用了INTERNET、INSTALL_PACKAGES、SEND_SMS，则*N*-gram中的特征元素序列为[[INTERNET:10],[INSTALL_PACKAGES:5],[SEND_SMS:8]]。

2.4 改进的*N*-gram模型

与2.3节介绍的*N*-gram的基本概念和其他研究人员的相关成果相比，本文使用权限和API调用来建立*N*-gram特征序列。由于权限特征的特殊性，不适用改进*N*-gram算法，因此本文只针对API特征。*N*-gram只是将一串数据转化为*N*-gram向量形式，只是简单地统计*N*-gram中元素出现的频次，并不能从实质上代表应用程序的恶意行为，仅反映了在恶意程序中出现的元素并没有在非恶意程序中出现。为了更好地区分和描述恶意行为特征，文本将提取的静态和动态API特征加入权重后重新计算每个*N*-gram特征元素的特征值。

通过对大量恶意应用程序进行分析和动态监控，提取反映恶意行为的API函数，根据不同函数的功能赋予不同的权重。例如，getDeviceId()获取设备的IMEI，权重为10；getSubscriberId()获取设备的IMSI，权重为10；openConnection()访问网站，权重为6等。本文使用滑动窗口为3建立3-gram序列，即每个元素中包括3个函数，并获得每个元素的出现频次。分别对静态API和动态API数据进行处理，同时出现在两个序列中的以动态特征为主。为了更好地反映API特征，还要对生成的*N*-gram序列进行筛选，去除明显无用调用，如[[init,init,init]:100]，然后计算每个元素的特征值。API特征值计算方法为：

$$V=P\times(W_{API1}+W_{API2}+W_{API3}) \quad (1)$$

公式（1）中 V 为特征值；P 为元素出现的频次；（W_{API1}，W_{API2}，W_{API3}）为每个 API 函数的权重。以恶意程序 DroidKongFu 为例，某个调用序列为 getDeviceId()、getSubscriberId()、getInstance() 且出现 2 次，则其特征值为 V=2×(10+10+1)=42，即 N–gram 序列中的一个元素为 [[2c4e, 2c5c, 0a24]:42]。

3 检测架构

通过描述如何对恶意程序进行建模和分类，并以此为基础提出一个完整的恶意程序检测框架。为了获取应用程序的动态 API 特征，我们使用了Android SDK 中的模拟器程序，其优点是避免恶意程序运行后带来的不良后果、部署方便且系统快速还原；缺点是运行速度较慢。恶意程序检测框架如图 4 所示。

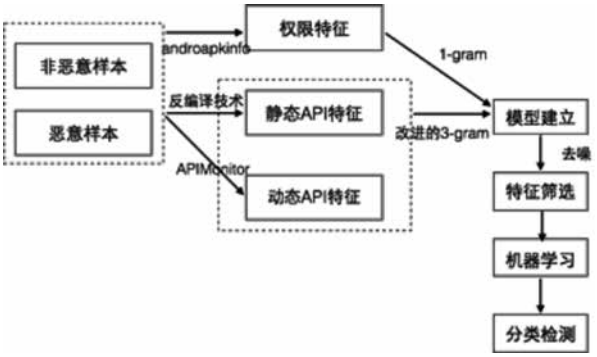

图4 恶意程序检测框架

1）使用静态方式提取 APK 文件的权限特征。

2）使用静态反汇编技术对 APK 中的 DEX 文件进行反汇编，并提取每个 smali 文件中的静态 API 调用特征。

3）搭建用于动态获取 API 调用序列的模拟器环境，用于获取动态特征。

4)向 APK 文件中插桩、重新打包，安装 APP 应用、运行，监控并提取相应的动态 API 调用特征。

5）提取语义特征并建立模型，对原始、未处理的语义信息进行加工，对权限特征使用原始 N–gram 算法，对 API 特征使用改进的 N–gram 算法。

6)对生成的 N–gram 序列使用机器学习方法进行检测。

4 实验与结果分析

4.1 实验环境

本文设计的检测框架由 Python 语言实现，涉及的工具包括 baksmali.jar、androguard、DroidBox。为了检验有效性，对现实中的 811 个非恶意程序和 826 个恶意程序进行检测。物理主机为 4 GB 内存，处理器为 Intel Core i5–2400，系统开发环境为 32 位 ubuntu 15.10。

4.2 实验结果与分析

实验中采用了两类样本，811 个非恶意程序和 826 个恶意样本，共计 1637 个。将所有样本提取特征建立模型后，使用 60% 的样本进行训练，而 40%（655 个）样本作为独立集合进行测试。使用静态、动态技术提取 APK 的 2 类特征，应用传统 N–gram 和改进 N–gram 算法分别对权限特征及 API 特征进行建模，最后应用机器学习算法进行分类检测。实验中选择朴素贝叶斯（Naive Bayes，NB）、逻辑回归算法（Logistic Regression，LR）、随机森林（Random Forest，RF）、决策树（Decision Tree，DT）、GBDT 算法（Gradient Boosting Decision Tree，GBDT）5 种分类器进行测试。

下面使用 Accuracy（准确率）、Precision（精确度）、Recall（召回率）、F1–score 等指标来衡量本文提出的检测方法的有效性。其中，召回率反映了被正确判定的正例占总的正例的比重。F1–score，即 F1 分数，是统计学中用来衡量二分类模型精确度的一种指标，它同时兼顾了分类模型的准确率和召回率，是模型准确率和召回率的一种加权平均。实验结果如表 1 所示。

表1 综合指标

分类	Precision	Recall	F1–score	Accuracy
NB	0.92	0.92	0.92	0.924
LR	0.92	0.92	0.92	0.918
RF	0.94	0.94	0.94	0.936
DT	0.92	0.92	0.92	0.924
GBDT	0.96	0.96	0.96	0.959

TPR（True Positive Rate）表示所有恶意代码被检测出的概率，用公式表示即为 $TPR=TP/(TP+FN)$，其中 TP 为正确检测到的恶意代码数量，FN 表示被判定为良性程序的恶意代码数量。FPR（False Positive Rate）表示良性程序被检测为恶意程序的概率，用公式表示即 $FPR=FP/(FP+TN)$，其中 FP 为被检测为恶意程序的良性程序的数量，TN 为被正确认定为良性程序的数量。漏警概率（Missing Alarm，MA）反映有多少个正例被漏判，用公式表示即为 $MA = FN/(TP+FN)$ 。

表2 检测指标

分类	*TPR*	*FPR*	*MA*
NB	0.95	0.10	0.05
LR	0.94	0.10	0.06
RF	0.91	0.03	0.08
DT	0.94	0.09	0.06
GBDT	0.95	0.03	0.05

由表 1 和表 2 可以看出，本文提出的 Android 恶意程序检测方法无论是从综合指标还是检测指标来看都具有较好的表现。对于文中提取的特征和改进算法，不同的分类器都具有较高的准确率，全部在 92% 以上。各方面指标最高的是 GBDT 算法，且 *FPR* 非常低，只有 0.03，充分显示了本文所提方法的优势。

由表 3 可以看出，准确率最高的 GBDT 算法使用了较长的时间，在平衡准确率和性能时，可以根据不同的需求进行算法的选择。

表3 检测时间

分类	时间
NB	0.085s
LR	0.228s
RF	0.334s
DT	1.115s
GBDT	385.490s

5 结束语

本文通过静态提取权限特征以及动、静两个方面提取 API 调用特征，综合大量实验和人工分析结果，对重要 API 函数进行权重设置，针对 API 特征使用改进的 *N*-gram 算法建立了特征模型，提高了特征的代表性。本文还进一步提出了用于恶意程序检测的基本框架，最后利用多种的机器学习算法对样本进行分类检测。

本文主要贡献是针对传统的 API 特征加入了权重，改进了 *N*-gram 算法，使特征更具代表性；提出了恶意程序检测框架并实现了相关功能；使用多种机器学习算法进行分类检测。实验中对由 811 个非恶意程序和 826 个恶意程序组成的集合进行检测，结果表明该方法在多种分类算法上都有较高的准确率，在各项指标上都表现良好。下一步工作是更加准确地提取特征并去除噪声，进一步改进模型算法，提高检测准确率，以完善检测框架。（责编 程斌）

参考文献：

[1] 阿里安全 . 阿里安全发布 2015 移动安全病毒年报 [EB/OL].http://www.freebuf.com/articles/terminal/97563.html，2016-03-02.

[2] CHRITODORESCU M, JHA S. Static Analysis of Executables to Detect Malicious Paterns[EB/OL]. http://xueshu.baidu.com/s?wd=paperuri%3A%28f1f8580140b17a89535c66c63c1d8d90%29&filter=sc_long_sign&tn=SE_xueshusource_2kduw22v&sc_vurl=http%3A%2F%2Fdl.acm.org%2Fcitation.cfm%3Fid%3D1251365&ie=utf-8&sc_us=16082281130912334676，2016-05-15.

[3] 徐曾春，卢洲，叶坤 . 一种检测可疑软件的 Android 沙箱系统的研究与设计 [J]. 南京邮电大学学报（自然科学版），2015，35（4）：104-109.

[4] YE Y, WANG D, LI T, etal. An Intelligent PE Malware Detection System Based on Association Mining[J]. Journal in Computer Virology, 2008, 4 (4) : 323-334.

[5] 李盟，贾晓启，王蕊，等 . 一种恶意代码特征选取和建模的方法 [J]. 计算机应用与软件，2015（8）：266-271.

[6] 杨欢，张玉清，胡予濮，等 . 基于多类特征的 Android 应用恶意行为检测系统 [J]. 计算机学报，2014，37（1）：15-27.

[7] AMOS B, TURNER H, WHITE J. Applying Machine Learning Classifiers to Dynamic Android Malware Detection at Scale[C]//IWCMC 2013.9th International Wireless Communications and Mobile Computing Conference ,July 1-5,2013.Sardinia.New York:IEEE,2013:1666-1671.

[8] SAHA J, KHAN L. A Machine Learning Approach to Android Malware Detection[C]//IEEE. 2012 European Intelligence and Security Informatics Conference (EISIC),August 22-24，2012，Odense.New York:IEEE,2012:141-147.

[9] PEIRAVIAN N, ZHU X. Machine Learning for Android: Malware Detection Using Permission and API Calls[C]//IEEE. 2013 IEEE 25th International Conference on Tools with Artificial Intelligence.November 4-6, 2013, Washington DC, USA.New York:IEEE,2013:300-305.

[10] SHABTAI A, FLEDEL Y, ELOVICI Y. Automated Static Code Analysis for Classifying Android Applications Using Machine Learning[C]//IEEE. 2010 International Conference on Computational Intelligence and Security.December 11-14，2010，Nanning，China. New York:IEEE,2010: 329-333.

[11] XU J Y，SUNG A H，CHAVEZ P，et al. Polymorphic Malicious Executable Scanner by API Sequence Analysis[C]//IEEE. 4th International Conference on Hybrid Intelligent Systems. June 20, 2004, Puszczykowo, Poland. New York:IEEE, 2004 : 378-383.

[12]PENG H, GATES C, SARMA B, et al. Using Probabilistic Generative Models for Ranking Risks of Android Apps[C]//ACM. 2012 ACM conference on Computer and communications security.October 16-18, 2012, Raleigh, NC, USA.New York:ACM，2012: 241-252.

[13] MIN Z, SUN M S , JOHN C S. DroidAnalytics: A Signature Based Analytic System to Collect, Extract, Analyze and Associate Android Malware[C]//IEEE. 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom).July 16-18, 2013, Melbourne, Australia. New York:IEEE, 2013: 163-171.

[14] GitHub.Dynamic Analysis of Android Apps[EB/OL].https://github.com/pjlantz/droidbox,2016-05-15.

[15] ASSALEH, CERCONE N, KESELJ V, et al. N-gram-based Detection of New Malicious Code[C]//IEEE.Computer Software and Applications Conference, 2004. COMPSAC 2004. 28th Annual International.September 28-30，2004，Hong Kong, China.New York:IEEE, 2004 : 41-42.

[16] MOSKOVITCH R, FEHER C, TZACHAR N, et al. Unknown Malcode Detection Using OPCODE Representation[C]//Springer.Intelligence & Security Informatics, First European Conference.December 3-5, 2008, Euroisi, Esbjerg, Denmark.Berlin Heidelberg:Spring, 2008: 204-215.