文章编号:1000-5862(2019)02-0147-07

基于 ICC 的 Android 恶意程序检测方法

郭 帆[1],黄 硕[1],王昌晶[1],涂风涛[2]
(1. 江西师范大学计算机信息工程学院,江西 南昌 330022;2. 豫章师范学院,江西 南昌 330103)

摘要: 结合语法和语义特征的检测方法可有效识别 Android 恶意程序. 对现有以污点传播路径为语义特征的方法进行扩展,将不同组件内的 Source 方法和 Sink 方法对定义为跨组件(ICC)污点传播路径,并进一步抽象为基于类的路径集合作为新的语义特征,按照它们在不同样本集中出现次数的比例关系进行特征值规范化,采用 SVM 进行分类和检测. 针对 295 个样本的检测结果表明,准确率和虚警率均有一定改善.

关键词: 程序检测; 跨组件通信; 污点传播; 语义特征
中图分类号: TP 311 **文献标志码:** A **DOI:** 10.16357/j.cnki.issn1000-5862.2019.02.06

0 引言

随着智能手机的普及,全球已进入无线互联网时代. 智能手机成为人们生活中必不可少的工具,保护其不受安全性威胁显得极为重要. 而接近 85% 的手机搭载 Android 系统,对 Android 程序进行恶意行为的分析与检测是非常有必要的.

现有检测方法通常使用动态或静态方法提取程序特征,然后使用机器学习算法对程序进行归类以实现检测目的. 动态方法指在程序运行过程中收集其行为信息,如 Crowdroid[1] 监控程序运行过程中 Linux 内核的系统 API 调用情况. 静态方法分析程序的源代码、二进制代码和 Manifest 文件,提取特征数据. 特征可分为语法特征和语义特征,语法特征包括程序权限、签名和意图等,语义特征包括数据流、控制流、依赖图等. 如 SigPID[2] 提取程序声明的权限、DREBIN[3] 提取程序的权限和 Intent 等、Androguard[4] 提取程序的签名、Mudflow[5] 提取从 Source 方法到 Sink 方法之间的路径组成的数据流.

在使用动态或静态方法提取特征数据并规范化后,可使用机器学习算法建立模型. 常见的机器学习算法有 k 均值算法(k-means)[1,4]、支持向量机算法(Support Vector Machine,SVM)[3] 等.

本文在现有研究基础上,提出一种基于跨组件通信(Inter-Component Communication,ICC)的检测方法,抽取程序声明的权限和 Intent-action 作为语法特征,以组件内污点传播路径和 ICC 污点传播路径为语义特征,并使用 SVM 算法进行训练和检测. 实验结果表明,相比现有方法准确率和虚警率均有一定改善.

1 相关工作

现有方法使用动态或静态分析方法提取特征,有的只提取语法特征,有的只提取语义特征,还有的综合提取 2 种特征,然后使用机器学习算法建立模型进行检测.

DREBIN[3] 以权限、可疑 API 和网络地址等作为语法特征,获得较好的检测结果. SigPID[2] 进一步减少权限集合,仅以 22 个权限作为特征,准确率与 DREBIN[3] 不相上下. 但是本文实验结果表明,将它们应用于包含 ICC 的样本集时,准确率仅为77.3%. Crowdroid[1] 通过动态方法统计每个内核 API 的调用次数作为语义特征,使用 k-means 算法实现聚类和检测,但是只能检测与正常程序名字和版本相同的木马. Zhou Wu 等[7] 以程序依赖图为语义特征,仅能检测 2 次打包的程序,且特征集合过大. Mudflow[5] 和 HFDroid[6] 提取 Source 方法到 Sink 方法之间的污点传播路径作为语义特征,其中 MudFlow[5] 将正常程序的污点传播路径作为训练集,HFDroid[6] 在语义特征基础上增加权限、Intent-action 等语法特

收稿日期:2018-06-17
基金项目:国家自然科学基金(61762049,61562040),江西省自然科学基金(20171BAB202013)和江西省教育厅科技课题(GJJ161305,GJJ151330)资助项目.
作者简介:郭 帆(1977-),男,江西南昌人,副教授,博士,主要从事网络与信息安全的研究. E-mail:121171528@qq.com

征,并使用 k-means 算法对不同种类的恶意程序进行聚类,通过欧式距离判断待测程序属于正常还是恶意,相比 MudFlow[5] 获得了更好的检测效果. 但是,这 2 种方法只分析了组件内污点传播路径,没有考虑可能经过不同组件的污点传播路径,而样本分析结果表明有超过 20% 的程序包含 ICC 路径,它们对此类程序的分析结果精确度不高,虚警率超过 30%.

本文基于 SigPID[2] 和 HFDroid[6] 的工作提出一种改进的静态检测方法,主要特点如下:(i) 增加 ICC 污点传播路径作为语义特征,因为污点传播路径反映程序对外部数据的处理流程,将其与权限结合,能更准确地描述程序的意图;(ii) 根据每条 ICC 污点传播路径在所有样本中出现次数的比例关系对特征值进行规范化处理,改善了准确率和虚警率.

2 方法设计

基于 ICC 的静态检测方法框架如图 1 所示. 主要包括:(i) 基于 FlowDroid[3] 分析 APK,获得组件内污点传播路径集合作为语义特征,基于 IC3[9] 分析,获得声明的权限集合、每个组件的 Intent 和过滤器信息,并从 Intent 中提取 Intent-action 值作为语法特征;(ii) 提出一种方法用于匹配不同组件的 Intent 和过滤器,产生ICC路径,并结合组件内污点传播路径,生成 ICC 污点传播路径集合,作为语义特征;(iii) 将语法特征和组件内污点传播路径参照 HFDroid[6] 的方法进行规范化,将 ICC 污点传播路径按照在所有样本中出现次数的比例关系进行规范化;(iv) 使用 SVM 算法进行分类训练,获得检测模型,判断待测程序与已有模型是否匹配,进而区分正常与恶意程序.

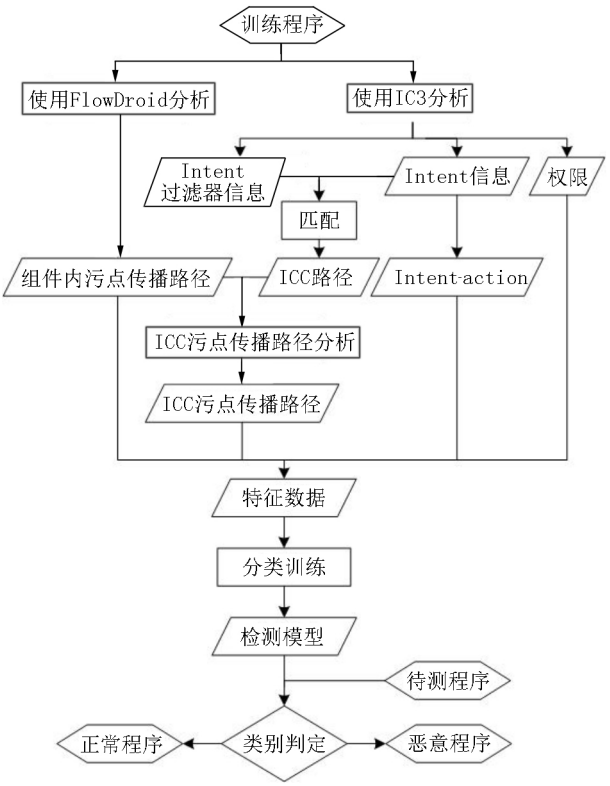

图 1 恶意 Android 程序检测方法框架

该方法采用的语法特征包括危险权限[10] 和 Intent-action. 危险权限是会对用户的隐私数据或其它程序构成威胁的权限. 恶意程序为了实现恶意操作,通常会声明危险权限,而正常程序对危险权限的需求相对来说要少,因此将危险权限作为区分正常和恶意程序的语法特征.

Zhou Yajin 等[11] 指出许多恶意程序通过劫持 Android 系统事件来执行恶意代码,如监听并拦截系统收到短信后发出的“SMS_RECEIVED”广播事件,随后删除运营商发出的通知短信. Android 系统的事件机制通过 Intent-action 实现,因此方法将 Intent 类中的 action 常量值作为第 2 种语法特征.

使用污点传播路径作为语义特征. 污点传播路径是从获取程序外部数据的 Source 方法到将数据送出程序外部的 Sink 方法的 API 调用序列集合. 由于每条 Source 到 Sink 路径之间的 API 调用节点较多,存储和计算每条路径的所有节点需要的时空代价过大,因此现有研究方法通常以(Source,Sink) 对代表具有相同 Source 和 Sink 的污点传播路径集合[5-6].

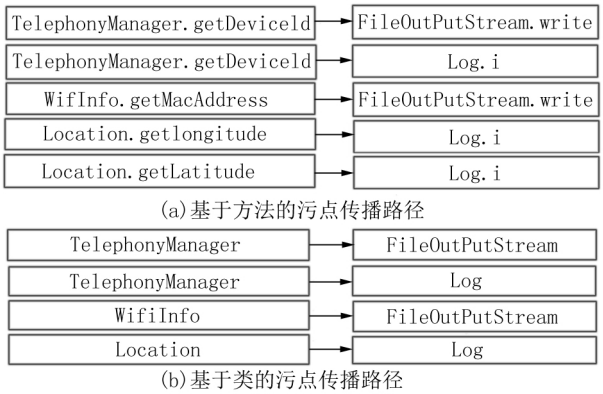

图 2 污点传播路径示例

SUSI[12] 统计出 Android 系统有 156 个 Source 方法和 164 个 Sink 方法,理论上存在 25 584 对. 进一步将基于方法的(Source,Sink) 对抽象为基于类的对,使得特征数量大大减少,但是检测准确率不会降低[5-6]. 图 2(a) 表示基于方法的(Source,Sink) 对,图 2(b) 表示相应的基于类的(Source,Sink) 对.

Android 程序有 4 大组件[13]:表示用户界面的 Activity、执行后台任务的 Service、管理应用数据的 Content Provider 和在系统范围内发送通知的 Broad-

cast Receiver. ICC 指不同组件通过 Intent 对象进行信息传递,Intent 可分为显式和隐式 2 种类型. 显式 Intent 通过指定名称来启动组件; 隐式 Intent 在发出后,系统会搜索所有程序的 Intent 过滤器,若某个程序的组件与之匹配[14],则会将该 Intent 发送给该组件.

基于 ICC 的污点传播路径指路径中的节点至少跨越 2 个不同组件,组件内污点传播路径指路径中的所有节点属于同一个组件. 如图 3 所示, Activity *A* 和 *B* 中的 Source 到 Sink 的路径称为组件内污点传播路径. 当 *A* 中的 Source 方法获取的外部数据被传递到组件内的 Sink 后,若进一步通过 Intent 发出,可能会被 *B* 的 Source 方法接收,然后进一步传递到 *B* 的 Sink 方法. 这样,从 *A* 的 Source 方法到 *B* 的 Sink 方法之间的污点传播路径即为一条 ICC 污点传播路径.

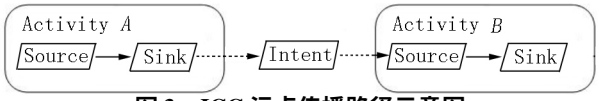

图 3　ICC 污点传播路径示意图

3　方法实现

ICC 通信分为显式和隐式通信,图 4 给出的代码片段说明如何通过隐式 Intent 产生 ICC 通信. 图 5 给出图 4 代码对应 APK 的 Manifest 文件信息,其中 5 ~ 9 行表示声明的权限,23 ~ 26 行表示组件 FooActivity 中的 Intent 过滤器信息.

```
1  public class MainActivity extends Activity {
2      protected void onCreate(Bundle savedInstanceState) {
3          …                                  发送信息的组件
4          TelephonyManager tel=
5              (TelephonyManager) getSystemService(TELEPHONY_SERVICE)
6          String imei=tel.getDeviceID();  ← source方法
7          Intent i=new Intent("android.intent.action.DIAL");
8          i.putExtra("data",imei);
9          startActivity(i);  ← sink方法
10     }
11     …
12     void sendSMS(String phoneNumber){
13     …
14     WifiManager wifiManager=
15         (WifiManager)getSystemService(Context.WIFI_SERVICE);
16     …
17     wifiInfo wifiInfo=wifiManager.getConnectionInfo();
18     int ipAddress=wifiInfo.getIpAddress();  ← source方法
19     String ip=intToIp(ipAddress);
20     android.telephony.SmsManager smsManager=
21                 android.telephony.SmsManager.getDefault();
22     smsManager.sendTextMessage(phoneNumber,null,ip,sentPI,deliverPI);
23 }                                          sink方法
24 }
25                          接收信息的组件
26 public class FooActivity extends Activity{
27     protected void onCreate(Bundle savedInstanceState){
28     …
29     Intent i=getIntent();  ← source方法
30     String imei=i.getStringExtra("data");
31     Log.d("deviceid",imei);  ← sink方法
32 }
33 }
```

图 4　隐式 ICC 代码片段

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android" >
3      package="com.ksu.implicitl"                    声明的权限
4      …
5      <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
6      <uses-permission android:name="android.permission.SEND_SMS"/>
7      <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
8      <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
9      <uses-permission android:name="android.permission.WAKE_LOCK"/>
10     <application
11         …
12         <activity                          组件名          Intent过滤器信息
13             android:name="com.ksu.implicitl.MainActivity"
14             android:label="@string/app_name" >
15             <intent-filter>
16                 <action android:name="android.intent.action.MAIN" />
17                 <category android:name="android.intent.category.LAUCHER" />
18             </intent-filter>
19         </activity>
20         <activity
21             android:name="com.ksu.implicitl.FooActivity"  ← 组件名
22             android:label="@string/app_name" >
23             <intent-filter>
24                 <action android:name="android.intent.action.DIAL" />
25                 <category android:name="android.intent.category.DEFAULT" />
26             </intent-filter>
27         </activity>
28     </application>                          Intent过滤器信息
29 </manifest>
```

图 5　Manifest 文件片段

图 4 中 MainActivity 调用构造方法生成 Intent 对象并设置 action 字段的值,这里没有显式指定接收组件的名称,属于隐式 ICC 通信,FooActivity 根据在 Manifest 文件中定义的过滤器信息可以接收 MainActivity 发出的 Intent[14]. MainActivity(第 7 行)生成 Intent 对象 *i*,并调用 startActivity 将 *i* 发送给其它组件(第 9 行),FooActivity 中的 getIntent 方法根据预定义的过滤器信息可接收该对象(第 29 行).

3.1　语法特征提取

语法特征提取过程基于 IC3[9] 实现,它解决了 ICC 通信中的多值复合常量传播问题,能够确定每个组件的 Intent 值、声明的权限以及过滤器中各个字段的可能取值,但是 IC3[9] 不能直接分析 Android 字节码,需要先使用 Dare[15] 将 Android 字节码转换为 Java 字节码.

图 6 是 IC3 对图 4 代码的分析结果片段,其中 3 ~ 7 行表示声明的权限,第 9 行和第 23 行分别表示 2 个不同组件名,第 11 ~ 20 行给出每个组件内的详细过滤器信息,第 27 ~ 36 行表示在 MainActivity 组件的 Intent 信息,其中第 29 ~ 30 行表示 Intent-action 的值. 基于图 6 的分析结果,方法可以方便地提取 APK 声明的权限和各组件使用的 Intent-action 值. 从图 6 提取的权限集合为 READ_PHONE_STATE、SEND_SMS、ACCESS_WIFI_STATE、CHANGE_WIFI_STATE 和 WAKE_LOCK,提取的 Intent-action 为"action. DIAL".

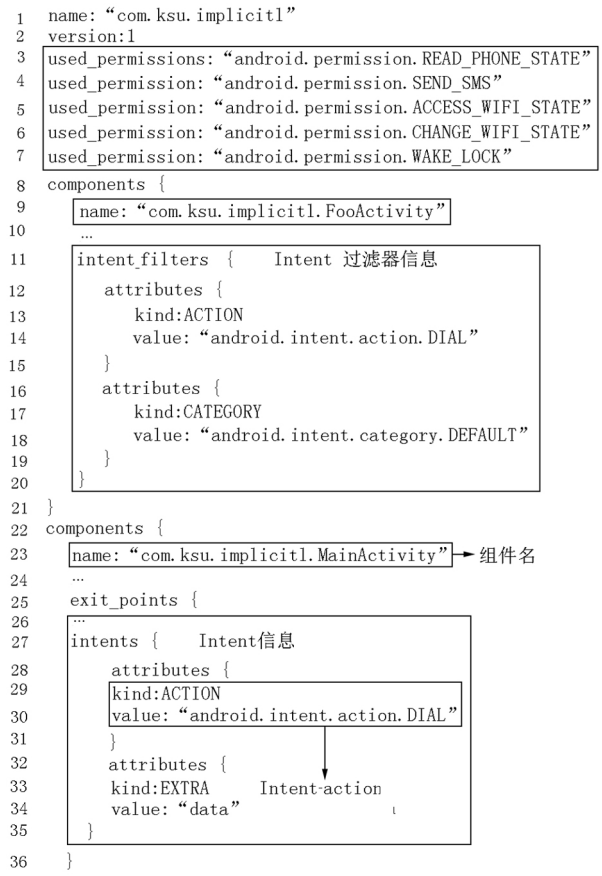

图 6 IC3 分析结果片段

3.2 组件内污点传播路径特征提取

基于 FlowDroid 框架[8]的静态污点分析方法可以获取组件内污点传播路径，通过构造 Dummy Main 方法模拟 APK 的生命周期，获取每个组件内部从 Source 方法到 Sink 方法的所有污点传播路径. 分析图 4 代码可以获得 3 条基于方法的组件内污点传播路径.

(i) (getDeviceId，startActivity). MainActivity 首先获取 IMEI 号并赋值给变量 imei(第 6 行)，然后创建 Intent 对象 i，并将 imei 中的数据存入 i 的 data 字段(第 8 行)，最后将 i 发送给其它组件(第 9 行)，该条完整污点传播路径为“6-8-9”.

(ii) (getIpAddress，sendTextMessage). 首先获取用户的 IP 地址并赋值给变量 ipAddress(第 18 行)，然后将该整型值转换为字符串并赋值给 ip(第 19 行)，最后将 ip 的值以短信形式发送到指定电话号码(第 22 行)，该条完整污点传播路径为“18-19-22”.

(iii) (getIntent，Log. d). 首先调用 getIntent 方法获取 Intent 并赋值给变量 i(第 29 行)，然后获取 i 中 data 字段的值并赋值给 imei(第 30 行)，最后将 imei 的值输出到日志(31 行)，该条完整污点传播过程为“29-30-31”.

将上述 3 条基于方法的(Source，Sink)对抽象为基于类的对，结果分别为(TelephonyManager，Activity)、(WifiInfo，SmsManager)和(Activity，Log).

3.3 ICC 污点传播路径分析

FlowDroid[8]可以获取 APK 的组件内污点传播路径，而 IC3 可以获取 APK 的 ICC 组件通信信息，方法将 2 者的分析结果相结合，并进行匹配来获取 ICC 污点传播路径.

ICC 通信需要满足 2 个条件:(i)对于显式 ICC 通信，发送 Intent 中指定的组件名必须与目标组件的组件名相同. 对于隐式通信，发送 Intent 与接收 Intent 的过滤器按照文献[14]定义的规则匹配;(ii)发送组件的 Sink 方法是发送 Intent 的方法，如 startActivity 和 startService 等;接收组件的 Source 方法是接收 Intent 的方法，如 getIntent 等. 根据上述通信规则对不同组件的 Intent 和 Intent 过滤器进行匹配，可以获得任意 2 个组件之间的 ICC 路径集合，然后将该集合与组件内污点传播路径集合进行拼接，即可获得 ICC 污点传播路径集合.

具体拼接算法如图 7 中伪码所示:(i)获取 APK 的所有组件列表;(ii)针对每个组件 comIndex 获得可能与该组件显式或隐式匹配的其它组件列表 ssMatchComList，然后尝试将 comIndex 中的每对(Source，Sink)与其它组件 ssMatchComIndex 的每对(Source，Sink)进行拼接，若满足拼接条件，则找到了一条长度为 2 的路径;(iii)重复步骤(ii)，只是获取与组件 ssMatchComIndex 可能匹配的其它组件列表，得到长度为 3 的路径. 方法只截取了路径长度为 2 和 3 的 ICC 污点传播路径集合，算法复杂度为 $O(n^3)$，n 为组件内污点传播路径的数量. 因为一般情况下 ICC 路径很少经过 3 个以上的组件，同时遍历这些较长的路径所需要的计算代价过高. 实验结果表明，以图 7 算法获取的 ICC 污点传播路径集合作为语义特征可以有效改善准确率和虚警率.

应用图 7 方法可以对图 4 代码提取 ICC 污点传播路径，只有 2 条组件内污点传播路径(getDeviceId，startActivity)和(getIntent，Log. d)可以进行拼接，产生 ICC 污点传播路径(getDeviceId，Log. d)，并进一步抽象为基于类的路径(TelephonyManager，Log)，其它路径不满足 ICC 通信条件.

```
begin:
  for ComIndex = θ to N
    ssList = getSourceLinkList( comIndex) ;
    for ssIndex = θ to ssList. Size( )
      ( source,sink)  = ssList. get( ssI) ;
    ssMatchComList = getMatchComponentList( ComIndex) ;
    for ssMatchIndex = θ to ssMatchComList. Size( )
    ssMatchComIndex = ssMatchCom. get( ssMatchComIndex) ;
    ssMatchComIndex = ssMatchCom. getIndex( ) ;
    ssMatchList = getSourceSinkList( ssMatchComIndex) ;
    for ssMatchIndex = θ to ssMatchList. Size( )
    ( sourceMatch,sinkMatch)  = ssMatchList. get( ssMatchIn-
dex)
    if( sink = = startActivity or sink = = startService)
    and( sourceMatch = = getIntent)
    addICCPath( source,sinkMatch,2)
    ssMatchComList = getMatchComponentist( ssMatchComIndex) ;
    for ssMatchIndex2 = θ to ssMatchComComList. Size( )
    ssMatchComCom = ssMatchComComList. get( ssMatchIndex2) ;
    ssMatchMatchList = getSourceSinkList( ssMatchComComIndex)
    for ssMatchMatchIndex = θ to ssMatchMatchList. Size( )
    ( sourceMatchMatch, sinkMatchMatch)  = ssMatchMatch-
List. get( ssMatchMatchIndex)
    if( sinkMatch = = startActivity or sinkMatch = = startService)
    and( sourceMatchMatch = = getIntent)
    addICCPath( source,sinkMatchMatch,3)
```

图 7　ICC 污点传播路径拼接算法

3.4　数据规范化

方法使用 WEKA [16] 进行模型训练与检测,获取特征数据后,需要对数据进行规范化,将每个样本的特征数据组织为一个多维度的特征向量. 方法对各类特征的规范化处理如下:

(i) 由于权限和 Intent-action 在程序中只需声明 1 次,因此将此类特征设置为 Nominal 类型,若样本中存在该特征则赋值为 TRUE,否则为 FALSE.

(ii) 由于每对(Source,Sink) 在程序中出现的次数不同,表示程序的不同数据流分支,因此将组件内污点传播路径设置为 Numeric 型更能精确表示程序的语义,即特征值为(Source,Sink) 对在程序中出现的次数.

(iii) 若将 ICC 污点传播路径也设置为在程序中出现的次数,准确率和虚警率改善不明显. 方法将 ICC 污点传播路径特征按照它们在不同样本集中出现次数的比例关系进行规范化,如某条路径在恶意样本集中总共出现 1 000 次,而在正常样本集中总共出现 500 次,则将其在恶意样本中的值设为 2,在正常样本中的值设为 1. 若某条路径只在某个样本集中出现,则将特征值设置为无穷大,在实验中设置为 1 000 000.

4　实验与分析

4.1　实验结果

实验使用的 APK 样本集包含 2 000 个正常样本和 2 000 个恶意样本,分别从包含 2 万多个恶意 Android 程序的 VirusShare [17] 中随机选取和从豌豆荚下载. 实验环境为 Dell PowerEdge-R730 服务器,CPU 型号是 E5-2667,主频 3. 2G Hz,内存 256 G,操作系统为 64 位 Ubuntu 14. 04,JDK 版本为 1. 8.

实验将 FlowDroid 和 IC3 分析 APK 文件的最长时间设置为 60 min,若超时则停止分析,结果显示绝大部分 APK 在 30 min 内分析完毕. 最终实验样本集合剔除了以下情况: (i) IC3 或 FlowDroid 无法完成分析; (ii) 没有找到组件内污点传播路径,具体样本数如表 1 所示.

表 1　实验样本数　　个

类别	恶意样本	正常样本
总样本数	2 000	2 000
实验样本数	834	590

因为不是每个程序都包含 ICC 路径,所以将样本集进一步分割为包含 ICC 和没有 ICC 的样本集合,如表 2 所示.

表 2　有 ICC 和无 ICC 样本数量　　个

类别	正常	恶意	总数
所有实验样本	590	834	1 424
有 ICC 样本	92	203	295
无 ICC 样本	498	631	1 129

实验首先对没有 ICC 的样本集使用基于语法特征的 SigPID [2] 和结合语法语义特征的 HFDroid [6] 进行分析,结果如表 3 所示. HFDroid [6] 相比 SigPid [2] 有一定改善,准确率提高了 2. 8%,漏报率和虚警率分别降低了 1. 9% 和 4. 0%. 接着对有 ICC 的样本集使用 HFDroid [6] 进行分析,结果如表 5 所示,虚警率高达 37%. 将 ICC 污点传播路径特征按照出现次数进行规范化并加入语义特征后,准确率提高了 4. 0%,但是虚警率仍然高达 28. 3%.

为了进一步提高准确率并改善虚警率,提出一种新的特征值规范化方法,按特征在所有样本中出现次数的比例关系进行规范化赋值.

表 3 无 ICC 样本集分析结果

特征	准确率/%	漏报率/%	虚警率/%
SigPid[3]	83.6	9.8	24.7
HFDroid[6]	86.4	7.9	20.7

表 4 比值前 10 的值

$S_{um_{正常}}/S_{um_{恶意}}$	$S_{um_{恶意}}/S_{um_{正常}}$
928.00	130.00
780.00	18.13
757.00	16.25
396.00	9.64
384.00	8.86
313.50	8.00
212.82	6.44
169.27	6.44
156.75	6.44
92.47	5.26

在包含 ICC 的样本集中,ICC 污点传播路径的数量是 339 条,其中 125 条路径只存在于正常样本集,还有 109 条路径只存在于恶意样本集,剩余 105 条路径同时存在于 2 个样本集中. 表 4 以 $S_{um_{正常}}$ 和 $S_{um_{恶意}}$ 分别表示单条路径在样本集中出现的总次数,对 105 条路径分别计算 $S_{um_{正常}}/S_{um_{恶意}}$ 和 $S_{um_{恶意}}/S_{um_{正常}}$,给出比例关系排在前 10 位的数值,例如"928.00"表示某条路径在正常样本中出现总次数是在恶意中出现总次数的 928.00 倍.

从表 5 结果可看出,将 ICC 污点传播路径特征规范化(称为 nICC)后,准确率进一步提高 4.8%,虚警率下降到 20.0%. 实验还单独使用 nICC 特征对包含 ICC 的样本集进行分析,结果表明准确率提高到 94.6%,漏报率仅为 0.5%,虚警率降到 16.3%. 将 HFDroid 和 nICC 相结合的方法,说明若以组件内传播路径作为语义特征分析包含 ICC 的样本,反而会降低分析结果的准确率和虚警率,因此应该单独采用 nICC 方法对包含 ICC 的程序进行分析,获得更精确的分析结果.

表 5 ICC 样本集分析结果

特征	准确率/%	漏报率/%	虚警率/%
SigPID[3]	77.3	8.9	53.3
HFDroid[6]	82.4	8.9	37.0
HFDroid[6] + ICC	86.4	6.9	28.3
HFDroid[6] + nICC	91.2	3.9	20.0
nICC	94.6	0.5	16.3

实验进一步分别使用 HFDroid[6] 和 nICC 结合 HFDroid[6] 的方法对所有样本集进行分析,表 6 中结果表明,将 HFDroid[6] 与 nICC 结合后,准确率、漏报率和虚警率均有一定改善.

表 6 所有样本集的分析结果

	准确率/%	漏报率/%	虚警率/%
HFDroid[6] + nICC	88.1	6.1	18.3
HFDroid[6]	86.7	6.7	22.5

最后,在表 7 中对比本文方法(HFDroid[6] + nICC)的实验结果与 Mudflow[5] 和 HFDroid[6] 在文献中给出的最终结果. 相比 HFDroid[6] 准确率提高 3.5%,漏报率降低了 6.5%,虚警率差别不大,但是本文实验的样本数量是 HFDroid[6] 的 3.76 倍. 相比 Mudflow[5] 准确率提高 6.9%,漏报率降低 14.2%,虚警率略有增加. 总体而言,方法的准确率和漏报率有较大改善.

表 7 相关工作对比

方法	测试集 正常	测试集 恶意	准确率/%	漏报率/%	虚警率/%
HFDroid[6] + nICC	590	834	88.1	6.1	18.3
HFDroid[6]	192	187	84.6	12.6	18.1
Mudflow[5]	1 533	2 579	81.2	20.3	17.3

4.2 案例分析

在包含 ICC 通信的样本中,正常样本集的 ICC 污点传播路径的种类不多,但是每条路径出现的总次数多,通常大于 10 次,最高为 928 次. 而恶意样本集中的路径种类较多,但是每条路径出现的总次数较少,通常小于 5 次. 因此,方法将 ICC 污点传播路径规范化为 nICC,能够更好地区分正常和恶意程序的语义. 例如,若仅使用 HFDroid[6] 对包名为"jp.co.canon.ic.eos.eosremote"的正常样本进行分析,它会被误报为恶意程序. 但是增加 ICC 污点传播路径为语义特征后,分析发现有几条路径分别出现过 169、92 和 58 次,而这些路径在恶意样本集中很少出现,因此它被正确判定为正常程序.

再如,对包名为"net.mingyihui.kuaiyi"的正常程序分析发现,它仅包含 2 条路径,它们在不同样本集合中出现的总次数比例关系分别是 6.0 和 1.7,与表 4 中 $S_{um_{恶意}}/S_{um_{正常}}$ 结果的前 6 ~ 10 位的比例关系值非常接近,说明它与许多恶意样本的 nICC 特征值十分相似,因此被错误归为恶意程序.

5 结语

提出一种基于 ICC 污点传播路径的恶意 Android 程序检测方法,以权限和 Intent-action 为语法特征,在以组件内污点传播路径为语义特征的基础

上增加 ICC 污点传播路径,并提出一种新的特征规范化方法,针对 ICC 特征在不同样本集中出现的总次数的比值进行规范化赋值,改善了检测结果的准确率和虚警率. 不足之处主要包括: (i) 若 APK 程序的 ICC 传播路径征较少,则容易产生虚警; (ii) 对于不存在 ICC 污点传播路径的 APK,方法无法分析.

未来工作主要是研究 nICC 特征与恶意行为的相关度,从而增加相关度高的特征权重,进一步降低虚警率.

6　参考文献

[1] Burguera I,Zurutuza U,Nadjm-Tehrani S. Crowdroid: behavior-based malware detection system for Android [EB/OL]. [2018-03-11]. 10.1145/2046614.2046619.

[2] Sun Lichao,Li Zhiqiang,Yan Qiben,et al. SigPID: significant permission identify-cation for android malware detection [EB/OL]. [2018-03-09]. http://doi.ieeecomputersociety.org/10.1109/MALWARE.2016.7888730.

[3] Arp D,Spreitzenbarth M,Hübner M,et al. DREBIN: Effective and explainable detection of android malware in your pocket [EB/OL]. [2018-03-09]. 10.14722/ndss.2014.23247.

[4] GitHub,Inc. Androguard [EB/OL]. [2018-03-09]. https://github.com/androguard/androguard.

[5] Avdiienko V,Kuznetsov K,Gorla A,et al. Mining apps for abnormal usage of sensitive data [C]. IEEE/ACM IEEE International Conference on Software Engineering,2015:426-436.

[6] 徐林溪,郭帆. 基于混合特征的恶意安卓程序检测方法[J]. 计算机工程与科学,2017,39(10):1837-1846.

[7] Zhou Wu,Zhou Yajin,Grace Michael,et al. Fast,scalable detection of "Piggybacked" mobile applications [EB/OL]. [2018-04-11]. 10.1145/2435349.2435377.

[8] 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.

[9] Octeau D,Luchaup D,Dering M,et al. Composite constant propagation: application to Android inter-component communication analysis [C]. IEEE/ACM IEEE International Conference on Software Engineering,2015:77-88.

[10] Google inc. Android 官方权限说明 [EB/OL]. [2018-04-11]. https://developer.android.google.cn/guide/topics/permissions/overview#viewing.

[11] Zhou Yajin,Jiang Xuxian. Dissecting Android Malware: characterization and Evolution [C]. Security and Privacy (SP),2012 IEEE Symposium on,2012:95-109.

[12] Rasthofer S,Arzt S,Bodden E. A Machine-learning approach for classifying and categorizing Android sources and sinks [C]. Network and Distributed System Security Symposium,2014.

[13] Google inc. Android 应用基础知识 [EB/OL]. https://developer.android.google.cn/guide/components/fundamentals#Components.

[14] Google inc. Intent 和 Intent 过滤器匹配规则 [EB/OL]. https://developer.android.google.cn/guide/components/intents-filters.

[15] Octeau D,Jha S,Mcdaniel P. Retargeting Android applications to Java bytecode [C]. ACM Sigsoft International Symposiumon the Foundations of Software Engineering,2012:1-11.

[16] Waikato inc. WEKA [EB/OL]. [2018-03-27]. https://www.cs.waikato.ac.nz/ml/weka/.

[17] VirusShare. Because sharing is caring [EB/OL]. [2018-05-11]. https://virusshare.com/.

The ICC-Based Detection Method on Android Malware

GUO Fan[1],HUANG Shuo[1],WANG Changjing[1],TU Fengtao[2]

(1. College of Computer Information Engineering,Jiangxi Normal University,Nanchang Jiangxi 330022,China;
2. Yuzhang Normal University,Nanchang Jiangxi 330103,China)

Abstract: The detection methods of combining syntax and semantic features can effectively identify Android malware. An improved static approach with two key-points is presented. Firstly,the new approach adds a new semantic feature called Inter-Component Communication Taint Propagation Path,which covers at least two components and is formally defined as a pair of methods,Source and Sink respectively. Moreover,the path is further abstracted as a pair of classes where the methods are defined. Then,every new feature is normalized according to the proportion of its total counts found in different sample sets. At last,a model based on SVM is created and used for classification and detection. The final experimental results show on 295 samples that the accuracy rate and the false positive rate are much better.

Key words: malware detection; ICC; taint propagation; semantic features　　(责任编辑: 冉小晓)