静动态结合的恶意 Android 应用自动检测技术

黄浩华[1, 2, 3], 崔展齐[1, 5], 潘敏学[4], 王林章[1, 2, 3], 李宣东[1, 2, 3]

[1] 计算机软件新技术国家重点实验室(南京大学), 南京 中国 210023
[2] 江苏省软件新技术与产业化协同创新中心 南京 210023
[3] 南京大学计算机科学与技术系 南京 210023
[4] 南京大学软件学院 南京 210023
[5] 北京信息科技大学计算机学院 北京 100101

摘要 随着移动互联网的快速发展, 移动终端及移动应用在人们日常生活中越来越重要, 与此同时, 恶意移动应用给网络和信息安全带来了严峻的挑战。Android 平台由于其开放性和应用市场审查机制不够完善, 使其成为了移动互联网时代恶意应用的主要传播平台。现有的恶意应用检测方法主要有静态分析和动态测试两种。一般而言, 静态分析方法代码覆盖率高、时间开销小, 但存在误报率较高的问题; 而动态测试准确度较高, 但需要实际运行应用, 所需的时间和计算资源开销较大。针对上述情况, 本文基于静动态结合的方法, 自动检测恶意 Android 应用。首先, 使用静态分析技术获取应用 API 的调用情况来判定其是否为疑似恶意应用, 特别是可有效检测试图通过反射机制调用 API 躲避静态分析的恶意应用; 然后, 根据疑似恶意应用 UI 控件的可疑度进行有针对性的动态测试, 来自动确认疑似恶意应用中是否存在恶意行为。基于此方法, 我们实现了原型检测工具框架, 并针对吸费短信类恶意行为, 对由 465 个恶意应用和 1085 个正常应用组成的数据集进行了对比实验。实验结果表明, 该方法在提高恶意应用检测效率的同时, 有效地降低了误报率。

关键词 Android 应用; 静态分析; 动态测试; 恶意行为

中图分类号: TP311.5 DOI 号 10.19363/j.cnki.cn10-1380/tn.2017.10.003

Automatic Malicious Android Application Detection Approach by Combining Static Analysis and Dynamic Testing

HUANG Haohua[1, 2, 3], CUI Zhanqi[1, 5], PAN Minxue[4], WANG Linzhang[1, 2, 3], LI Xuandong[1, 2, 3]

[1] State Key Laboratory of Novel Computer Software Technology, Nanjing University, Nanjing 210023, China
[2] Jiangsu Novel Software Technology and Industrialization, Nanjing 210023, China
[3] Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China
[4] Software Institute, Nanjing University, Nanjing 210023, China
[5] Computer School, Beijing Information Science and Technology University, Beijing 100101, China

Abstract Mobile devices and mobile applications are becoming more and more important with the rapid development of mobile Internet. Meanwhile, malicious applications have brought serious challenges for the security of network and information. Because the openness and poor review mechanism of the Android platform, it becomes the main transmission platform of malicious applications. At present, static analysis and dynamic testing can be used to detect malicious Android applications. Generally speaking, static analysis has high code coverage and low time costs, but it could cause high false alarm rates. While dynamic testing has high accuracy, but it has high time costs and requires much resource. Therefore, this paper combines static and dynamic detection technology to detect malicious applications automatically. Firstly, this paper uses static analysis to determine whether an application is potentially malicious according to sensitive API calls. Especially, to prevent hidden malwares from static analysis, we take into consideration the reflection call and can detect them effectively. And then, this paper confirms whether the application contains malicious behavior using dynamic testing base on suspicious degree of UI controls. Focus on malicious SMS applications, this paper implements a tool and makes experiments on 465 malicious and 1085 non-malicious applications in real. The experimental results show that the proposed method can effectively improve the detection efficiency and reduce the false alarm rate.

通讯作者: 王林章, 博士, 教授, Email: lzwang@nju.edu.cn。

本课题得到国家重点研发计划项目课题(No.2016YFB1000802), 国家自然科学基金项目(No.61472179, No.61572249, No.61632015, No.61561146394), 计算机软件新技术国家重点实验室开放课题(No.KFKT2016B12)资助。

收稿日期: 2017-04-28; 修改日期: 2017-08-09; 定稿日期: 2017-08-23

Key words Android Application; Static Analysis; Dynamic Detection; Malicious Behavior

1 引言

近年来, 随着无线通信技术和移动互联网迅猛发展, 移动终端已经成为人们日常生活中必不可少的一部分。据 CNNIC 第 39 次《中国互联网络发展状况统计报告》[1], 截至 2016 年 12 月, 我国手机网民规模已达到 6.95 亿, 台式电脑、笔记本的上网比例则继续呈下降态势, 网民上网设备进一步向移动端集中。当前主流移动端设备主要有 Android 和 iOS 操作系统, 其中 Android 设备相对于 iOS 而言用户量更大。据 CNNIC《2015 年中国手机网民网络安全状况报告》[2], 使用 Android 操作系统的手机占 67.4%。然而, 由于 Android 系统自身的开放性和应用的分发渠道的不规范性, 造成该平台已成为了移动互联网恶意程序传播的主要平台。如: CNCERT 数据显示[3], 2015 年检测到的 147.2 万个移动互联网恶意程序中, 基于 Android 平台的占比在 99.6%以上; 又如: 360 互联网安全中心 2016 年共截获 Android 平台新增恶意程序样本 1403.3 万个, 平均每天新增多达 3.8 万个恶意程序样本[4]。因此, 如何高效检测恶意 Android 应用受到了学术界和工业界的广泛关注。

恶意 Android 应用是指可在 Android 平台上运行的含有恶意代码的移动应用。其中, 恶意代码是指在用户不知情或未授权的情况下, 在移动终端中安装、运行以达到不正当目的, 或具有违反国家相关法律法规行为的可执行文件、代码模块或代码片段[5]。恶意应用的出现使得用户数据受到严重威胁, 恶意应用的类型主要包括恶意扣费、隐私窃取、远程控制、恶意传播、资费消耗、系统破坏、诱骗欺诈等行为, 可能会造成用户经济上的损失或隐私数据的泄漏。

现有的恶意 Android 应用检测方法主要可分为: 静态分析[6,7]、动态测试[8]和基于数据挖掘的恶意应用检测[9,10]三类。其中静态分析主要通过分析程序安装包或源码来判断是否为恶意应用, 此类方法的代码覆盖率高, 但由于没有实际执行程序, 误报率较高; 而且恶意制造者会通过代码混淆等技术来躲避静态分析。动态测试通过在模拟器或真机上实际运行应用, 收集运行过程中的程序行为信息来判断应用恶意性, 所发现的都是真实的恶意应用, 其缺点是代码覆盖率较低和开销较大。基于数据挖掘的检测方法通过学习权限、函数调用等特征, 构造分类器, 从而对 Android 应用的恶意性进行判定。此类方法同样不需要实际运行应用, 因此也存在较高的误报率。此外, 这类方法是基于已有恶意软件去挖掘恶意特征的, 因此无法检测新类型的恶意应用。

针对上述问题, 本文基于静动态结合的方法, 自动检测恶意 Android 应用。首先, 利用静态分析, 根据 API 调用行为和申请权限等特征信息, 初步筛选出疑似恶意应用; 其次, 对静态分析所报告的疑似恶意应用进行有针对性的动态测试, 来自动确认疑似恶意应用; 最后, 在上述框架的基础上, 开发原型工具, 针对特定恶意行为, 进行实验研究。

本文的主要创新点和贡献如下:

(1) 在通过静态分析检测恶意应用时, 为了防止某些恶意应用躲避静态检测, 本文通过静态分析检测使用反射机制调用 API 来实现的疑似恶意行为, 同时还首次覆盖了通过数组赋值方式进行的反射调用, 从而能有效检测出试图通过反射机制躲避静态分析的恶意应用;

(2) 基于该方法实现了一个原型工具, 并针对恶意短信扣费行为, 在一组由真实的存在恶意吸费行为的应用和正常应用所组成的实验对象集上进行了实例研究, 以评估本文方法的有效性。

本文第 2 节介绍相关研究背景; 第 3 节详细介绍静动态结合的恶意应用自动检测方法; 第 4 节介绍原型工具的实现并进行实验和评估; 第 5 节介绍与本文相关的工作; 第 6 节总结本文工作并进行展望。

2 研究背景

本节主要对 Android 应用市场、APK 文件构成及 Android 应用中的反射调用进行介绍。

2.1 Android 应用市场

在市场上发布一个 Android 应用需要接受应用商店的审核, 这样能够控制应用软件的质量以及进行安全保护。为此, Google 在 2012 年 2 月份发布了审核服务 Bouncer[11], 在一定程度上减少恶意软件的传播, F-Secure[12]发现的 Google 市场的恶意软件仅占总数的 0.1%。但是很多第三方 Android 应用市场缺乏这样的审查机制, 因此滋生了大量的恶意软件, 据 F-Secure[13]报告显示, 中国用户常用的四大应用商店: 安智、木蚂蚁、百度和优亿中, 恶意软件检测样本比例近 10%, 显然受到了恶意软件的极大威胁。

2.2 Android 应用程序构成

当前的 Android 应用基本上都是用 Java 语言开发的。Android SDK 工具编译源代码, 将数据和资源

打包成一个 APK 包文件, 最后的 APK 文件可以直接在 Android 设备上安装运行。

Android 应用程序 APK 文件主要包括: Android Manifest.xml, class.dex (字节码文件, 由 Dalvik 虚拟机解释), 其他二进制或 XML 格式的资源文件都保存在 res/和 assets/目录下。

APK 文件的 manifest 文件是 Android 程序的全局配置文件, 是每个 Android 程序中的重要文件。它的主要功能有包括: 指定应用程序包名; 描述应用程序的组件构成: 活动(Activity)、服务(Service)、广播接收者(Broadcast Receiver)、内容提供者(Content Provider); 声明该应用程序访问 API 以及与其他应用程序进行交互的权限。

2.3 Android 应用中的反射调用

反射是一些编程语言的一种特性, 它允许运行中的程序观察自身和当前的运行环境并做出相应的改变[14]。Java 的反射机制表现为动态获取信息以及动态调用对象的方法。例如: 在运行状态中, 对于任意一个类, 都能够知道这个类的所有属性和方法; 对于任意一个对象, 都能够调用它的任意一个方法和属性。

Java 反射机制起源于 java.lang.Class 这个类, 利用类名来获取对应的运行时数据结构, 当一个类被加载后, Java 虚拟机自动生成 Class 对象, 通过该 Class 对象能够获取加载到虚拟机中的该对象所对应的成员、方法以及构造方法的声明和定义等信息。

通过反射可以使程序代码访问装载到 JVM 中的类的内部信息; 此外, 利用反射机制能够增加程序的灵活性, 避免将程序固定化, 允许程序创建和控制类对象, 无需提前硬编码目标类, 能够很好地达到解耦的效果; Android 拥有众多 Java 语言的性质, 它继承了 Java 反射 API, 在 Android SDK 中提供给开发者调用。

近年来, 恶意软件制作者将反射机制作为隐藏软件中的恶意行为的一个重要途径[15,16]。恶意应用为躲避静态分析, 可在运行时刻通过反射来调用敏感方法, 以此来传播恶意代码。Martina Lindorfer 等人[16]对 2010 年到 2014 年以来的恶意软件的行为进行统计, 结果表明 57.08%的恶意软件样例都调用了反射机制, 而且从 2010 年开始就保持这个占比, 未得到缓解。

由于反射机制的特性, 给众多静态分析工具都带来了挑战, 目前针对反射方法调用的检测是比较欠缺的, 尤其是针对 Android 应用, Barros 等人[17]虽然提出了检测 Android 应用中反射调用的方法, 但是需要基于源码, 显然在实际中是不适用的。据文献[18]统计, 超过 90%的 Android 应用静态分析技术未考虑反射调用问题, 近期报告[19]还指出反射调用问题是现在的很多恶意软件检测工具表现不佳的一个重要因素。

当前针对 Android 应用通过反射调用 API 隐含恶意行为的研究不多, 其中 Li 等人[20]对直接通过字符串作为参数进行反射调用的恶意程序检测进行了研究, 并实现了工具 DroidRA[18]。例如, 针对图 1 中的方法, DroidRA 可以检测出第 6-8 行使用了反射调用方法“setImei”, 第 9-10 行使用了反射调用方法“getImei”。但 DroidRA 并不完善, 还不能检测出复杂的反射调用方法, 例如, 通过数组赋值方式传递的参数就不能得到有效检测。如图 2 代码所示, DroidRA 无法检测出第 7-9 行通过反射调用方法“setImei”, 因为其参数 method_name 的值是通过字符串数组的方式赋值的。

```
1   TelephonyManager telephonyManager = //default;
2   String imei = telephonyManager.getDeviceId();
3   Class c =
4       Class.forName(“de.ecspride.ReflectiveClass”);
5   Object o = c.newInstance();
6   Method m = c.getMethod(“setIme” + “i”,
7       String.class);
8   m.invoke(o,imei);
9   Method m2 = c.getMethod("getImei");
10  String s = (String) m2.invoke(o);
11  SmsManager sms = SmsManager.getDefault();
12  sms.sendTextMessage("+49 1234", null, s, null, null);
```

图 1 文献[12]Fig. 1 中的代码片段

```
1   TelephonyManager telephonyManager = //default;
2   String imei = telephonyManager.getDeviceId();
3   Class c =
4       Class.forName(“de.ecspride.ReflectiveClass”);
5   Object o = c.newInstance();
6   char[] method_name={‘s’, ‘e’, ‘t’, ‘I’, ‘m’, ‘e’, ‘i’};
7   Method m = c.getMethod(new String(method_name),
8       String.class);
9   m.invoke(o,imei);
```

图 2 数组赋值方式进行反射调用的示例代码片段

3 静动态结合的恶意 Android 应用自动检测技术

3.1 方法框架

针对静态检测误报率高、动态检测代码开销大问题, 我们基于静动态结合的方法, 自动检测恶意 Android 应用, 通过静态分析检测疑似恶意行为, 并通过动态测试确认来最终判定一个应用是否包含恶意行为。该方法的架构如图 3 所示。该方法主要由两大部分组成: 基于特征分析的恶意行为检测和基于可疑度分析的恶意行为自动确认。

(1) 恶意行为检测:

- 敏感 API 调用分析: 通过对 Android 应用反编

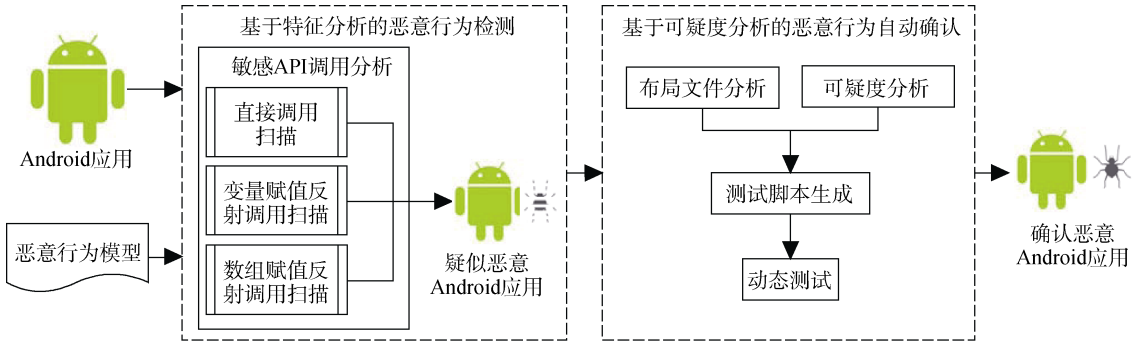

图 3　静动态结合的恶意 Android 应用自动检测及确认技术框架

译阶段得到的 smali 文件进行静态扫描分析, 确定应用中是否调用了相关敏感 API, 对于 API 的调用检查主要从两个角度出发: 直接调用、反射调用, 其中反射调用中又分为基于普通变量赋值的反射调用和基于数组赋值的反射调用, 具体在 3.2 节中描述;

● 疑似恶意行为检测: 若检测到恶意行为模型中定义的调用敏感 API 行为, 则报告存在疑似恶意行为; 否则认为该应用无恶意行为。根据给定的不同恶意行为模型, 可以检测不同类型的恶意应用。恶意行为模型主要包含两方面: 1)代码中与恶意行为相关的包含敏感操作的函数调用, 如发送短信、发送地理位置、拨打电话等, 可能会导致用户隐私数据泄漏等风险; 2)应用实际运行过程中敏感函数的调用, 即恶意行为的触发条件。

(2) 恶意行为确认:

● 可疑度分析: UI 控件的 ID 和可疑度有着一定的联系, 因此利用自然语言处理对 UI 控件 ID 进行可疑度分析, 将最有可能触发恶意行为的 UI 控件最先执行;

● 测试脚本生成: 按照可疑度排序, 依次为 UI 控件生成自动化测试脚本;

● 恶意行为触发: 执行测试脚本, 监测应用运行过程中的行为信息, 若发现恶意行为, 则确认应用为恶意应用。

3.2　基于特征分析的恶意行为静态检测

在此阶段通过静态抽取应用的特征来判断它是否可能为恶意应用。用来检测恶意行为的静态特征有很多, 例如应用权限[21,22]、应用组件间通信(Inter-Component Communication, ICC)机制[23]以及敏感 API 调用[24,25]等, 其中敏感 API 调用对于恶意行为来说是最本质、最关键的信息, 为此本文将它作为静态特征来辨别恶意行为。

为了躲避常规静态检测, 黑客开发恶意应用技术在不断发展, 调用敏感 API 的方式也在不断地演化, 从早期的简单直接调用变得越来越复杂和隐蔽。为此, 对应的检测技术的要求也越来越高。下面对本方法中针对不同 API 调用方式所采取的不同检测方法进行介绍:

(1) 直接调用敏感 API

采用此方式的应用, 如需要发送短信, 则调用对应函数 sendTextMessage(*destinationAddress*, *scAddress*, *text*, *sentIntent*, *deliveryIntent*), 本方法采取直接扫描所有 smali 文件中 API 使用情况的方式来检查是否存在相关函数调用, 主要包括以下几个步骤:

● 从所有 smali 文件中排除部分无关文件, 例如 R.smali、R$attr.smali、R$layout.smali 等文件, 因为这类文件并不属于主要功能文件, 不会有敏感函数的调用, 没有必要进行分析;

● 逐个扫描剩余的 smali 文件, 逐行分析文件内容, 如果程序包含 invoke-virtual 以及对应的敏感函数, 则表明该应用中确实调用了相关敏感 API。图 4 描述了文献[9]使用的实验数据集中 0a68a27d0298bdb226dac7a5ab4853e8794b3ca8b2d20ee6a413559315d29f35.apk 应用中通过调用 sendTextMessage 函数发送短信的 smali 代码片段, 其中第 1 行中 invoke-virtual 表示虚方法调用, 调用的方法运行时确认实际调用, 一般 API 都采用这种方式进行调用;

```
1  invoke-virtual/range {v0 .. v5},
2  Landroid/telephony/SmsManager;->sendTextMessage
3  (Ljava/lang/String;Ljava/lang/String;
4  Ljava/lang/String;Landroid/app/PendingIntent;
5  Landroid/app/PendingIntent;)V
```

图 4　发送短信 smali 代码片段

● 如果所有文件扫描结束也没找到敏感函数的调用, 说明该应用中不存在直接调用敏感 API 的情况, 需要进一步分析有没有采用比较隐蔽的方式调用敏感 API。

(2) 通过变量赋值的反射调用

图1中的代码片段即为此类反射调用方式, 该案例[①]被很多测试工具用来评估性能, 该反射案例中首先利用 forName 方法取得被调用方法所在类 ReflectiveClass (行3-4), 并且初始化得到对象(行5), 最后利用getMethod方法获取反射调用的API并且利用 invoke 方法去执行(行6-10)。其中被调用的 setImei 通过字符串拼接的方式形成, getImei 直接赋值, 此外, 可以先将字符串赋值给变量, 再将该变量作为 getMethod 的参数, 无论何种方式, 本质上都统一为变量赋值类型的反射调用。

通过直接扫描 smali 文件无法检测出通过这种方式实现的敏感 API 调用, 为此, 本文基于文献[17]中的方法框架去检测这类隐蔽调用。

该方法中首先对 APK 文件进行预处理: 转换成 Jimple 代码、确定分析入口以及确保覆盖到所有应用代码。

然后对预处理后的代码进行反射分析, 采用过程间的、基于上下文、基于数据流的分析方法来检测 Android 应用中是否包含反射调用, 并且给出反射调用的参数(类名、方法名以及方法参数)。

(3) 通过数组赋值反射调用

为了躲避检测, 恶意应用会采取更复杂的利用数组赋值的方式进行反射调用敏感函数, 文献[9]中给出的恶意数据集中就有大量的数组赋值反射调用, 图 2 中的代码片段即为此类反射调用方式, 和图 1 中的变量赋值反射调用相比, 调用过程是一致的, 但它的区别主要在于通过数组来赋值函数名称(行 6-8), 相比于普通变量赋值的反射调用更复杂。针对这一问题, 本方法首先检测 smali 文件中是否有存在数组赋值语句, 然后分析数组所存放的字符串是否就是所要关注敏感 API 函数名。图 5 为采用数组赋值方式调用 sendTextMessage 函数对应的 smali 代码, 其中第 1 行指明一个数组赋值开始, 第 17 行代表数组赋值结束, 第 2-16 行分别为数组中每个元素的 ASCII 码表示。

通过这种方式调用敏感 API 的检测过程主要包括以下几个步骤:

- 扫描 smali 文件, 如果有关键字.array-data 和.end array-data, 则提取两个关键字间的内容进行分析;
- 还原 smali 文件中使用 16 进制 ASCII 码表示字符;
- 查看还原后的数组存放的字符串是否为需要关注的敏感 API 函数名。

```
1   .array-data 1
2       0x65t
3       0x78t
4       0x74t
5       0x4dt
6       0x65t
7       0x73t
8       0x73t
9       0x61t
10      0x67t
11      0x65t
12      0x73t
13      0x65t
14      0x6et
15      0x64t
16      0x54t
17  .end array-data
```

图 5 采用数组赋值方式调用 sendTextMessage 函数对应 smali 代码片段

上述 3 种检测敏感 API 调用的方法类似且相互独立, 为提高检测效率, 避免重复扫描 smali 文件, 本方法将上述 3 种检测步骤合并处理, 即在分析 smali 文件时一并检查是否存在上述方式实现的敏感函数调用。若通过上述 3 种方式检测到应用中调用了敏感 API, 则认为该待检测应用可能包含恶意行为, 为疑似恶意应用, 需要进行下一步的动态测试进行确认; 反之, 则认为该待检测应用为非恶意软件, 不需要进行动态测试。

3.3 基于可疑度分析的恶意应用动态确认

在此阶段通过动态测试来确认静态分析报告的疑似恶意应用是否为真实的恶意应用。在这一过程中, 主要通过在模拟器运行 Android 软件, 然后再生成自动化测试脚本对软件进行操作以触发恶意行为, 通过分析日志信息来监测整个过程中的各类行为信息, 特别是 API 调用信息。为提高自动测试效率, 本方法对应用的 UI 控件进行可疑度分析, 在生成测试脚本时, 优先对可疑度较高的 UI 控件进行测试。基于可疑度分析的恶意应用自动确认流程如图 6 所示。

若在测试过程中确认了恶意行为被触发, 则该应用即为恶意软件; 若未检测到恶意行为, 则为非恶意软件。

3.3.1 页面布局文件分析

在自动化测试中, 需要通过测试脚本执行应用。因为测试脚本就是对应用中页面 UI 控件的一系列操

① https://github.com/secure-software-engineering/DroidBench/blob/master/apk/Reflection/Reflection3.apk

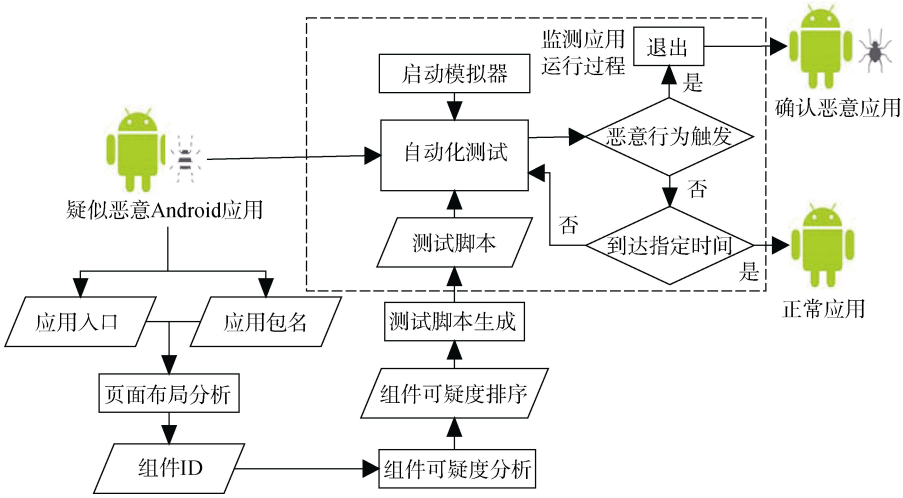

图 6　基于可疑度分析的恶意应用动态确认

作, 为生成测试脚本, 需要知道页面的 UI 控件信息。当前有很多的工具可以查找 UI 控件, 例如 Hierarchy Viewer①、uiautomatorviewer②等, 但启动这些工具会影响效率、占用额外的计算资源。在本文的方法工作中, 为生成测试脚本只需要获取某些 UI 控件(例如 Button), 并不需要查看所有 UI 控件信息。因此, 本方法从代码分析的角度去分析页面布局情况。具体步骤如下:

(1) 根据 AndroidManifest.xml, 查找 Activity, 然后根据 Activity 组件查找 smali 文件夹中对应文件, 如图 4 应用中 FirstActivity 页面对应的 smali 文件即为 FirstActivity.smali;

(2) 分析该 smali 代码, 获取该 Activity 对应的布局文件的 16 进制表示, 在该例中, 对应的 16 进制表示为 0xf030001;

(3) 扫描 R.layout 文件, 查找该 16 进制数对应页面布局 XML 文件, 如图 7 中第 2 行所示, main 即为该 Activity 对应的页面布局名;

(4) 解析相应的 XML 文件, 得到所有 UI 的 ID, 如需要找页面上所有按钮的 ID, 则找到“<Button>”标签, 即可找到对应的 ID, 如图 8 所示, 第一个按钮的 id 为 firstScreenRulesButton, 第二个按钮的 id 为 firstScreenAcceptButton。

```
ield public static final finish:I = 0x7f030000
ield public static final main:I = 0x7f030001
ield public static final member:I = 0x7f030002
ield public static final question:I = 0x7f030003
ield public static final rules:I = 0x7f030004
```

图 7　示例应用 R.layout 文件代码片段

```
<Button android:id="@id/firstScreenRulesButton"
<Button android:id="@id/firstScreenAcceptButton"
```

图 8　示例应用 main.xml 示例

3.3.2　可疑度分析

经过对文献[9]数据集中大量恶意 Android 应用的分析发现, 对于实际触发恶意行为的 UI 控件, 其名称具有一定规律, 例如: 类似“Next”、“Accept”、“OK”这类名称的 UI 控件相比“Cancel”、“Back”这类名称的 UI 控件更有可能触发恶意行为产生。因为这些恶意应用会弹出一个能够使得应用得以继续运行的接受条件, 那一般来说用户都会选择接受, 为此恶意应用会利用这点实施它们的恶意行为。为此, 在生成测试脚本之前, 本方法先对 Android 应用界面中的 UI 控件进行可疑度分析, 并根据控件 ID 的可疑度进行排序, 将最有可能触发恶意行为的 UI 控件优先执行测试。这将有助于节省动态测试恶意软件的时间开销, 对于界面较为复杂的大型应用尤为有效, 本文采取的可疑度分析方法具体步骤如下:

(1) 将所有 UI 控件 ID 进行规范化处理, 如统一大小写等;

(2) 按照可疑度来给 UI 控件 ID 排序, 根据分析恶意应用获取的历史经验, 对 UI 控件 ID 中含有“Next”、“Accept”、“OK”等名称的控件赋以较高可疑度值, 其他控件赋以较低可疑度值, 从而保证最有可能触发恶意行为的控件优先执行。若多个控件的 ID 都包含相同的敏感单词, 则按照控件在页面布局文件中出现顺序执行。

① Hierarchy Viewer: http://developer.android.com/tools/debugging/debugging-ui.html
② uiautomatotviewer: http://www.guru99.com/uiautomatorviewer-tutorial.html

3.3.3 动态测试

在完成上述分析后, 将对 Android 应用进行动态测试。为驱动疑似恶意应用执行, 需要自动生成测试脚本, 模拟用户点击、输入等操作。算法 1 为根据 UI 控件序列自动生成测试脚本的算法, 根据可疑度降序对 UI 控件进行对应操作。

其中对 UI 控件的操作可以是点击、输入等, 如: 点击操作对应的脚本就为: device.touch(By.id('id1'), MonkeyDevice.DOWN_AND_UP), 点击返回操作对应的脚本为 device.press('KEYCODE_BACK', MonkeyDevice.DOWN_AND_UP)。

利用 AndroidManifest.xml 得到的包名、入口 Activity 以及生成的自动化测试脚本对应用进行安装和自动化测试。

算法 1 测试脚本生成算法

```
1  输入: List:    IDs//UI 控件 ID 数组
2  输出: File:    script//测试脚本
3  过程: Target_Guided_Search(list IDs){
4      for(int i := 0); i<= sizeOf(IDs) ; i++ {
5          id := IDs[i]   //取出第 i 个 UI 控件
6          script.write(对 UI 控件的操作)
7          script.write(点击返回)
8      }
9  }
```

4 实现与评估

在上述方法基础上, 我们实现了一个静动态结合的恶意 Android 应用自动检测工具 MalDet[①], 并在一组真实的 Android 应用上进行了实验和评估。

4.1 原型工具实现

原型工具使用 Java 和 Python 语言实现, 其中 DroidBox[②]工具是基于 Python 语言的, 因此使用 Python 语言对 DroidBox 进行了修改和扩充。此外, 自动化测试脚本是使用 Python 语言描述的, 工具其他部分均用 Java 语言来实现。实验和开发该工具的硬件环境为 Intel(R) Core(TM) i3-3227U 1.90GHz 处理器、4GB 内存, 软件环境为 Ubuntu 16.04、JDK 1.7、Python 3.4。

如图 9 所示, MalDet 主要由两个模块组成。其中, 静态检测模块给出待测应用的权限信息, 并给出静态检测结果即待测应用是否为潜在恶意应用; 动态确认模块首先启动模拟器以及动态监测工具 DroidBox, 然后开始对应用进行自动化测试给出最终确认结果。图 10 为 MalDet 运行界面截图。

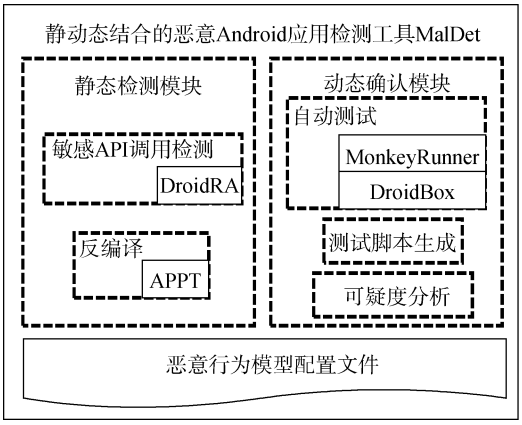

图 9 MalDet 架构图

图 10 MalDet 运行界面截图

敏感 API 调用检测、UI 控件可疑度分析、测试脚本生成、自动化测试等内容已在第 3 节介绍, 下面介绍 MalDet 的输入配置文件以及在工具中使用到的其他工具。

恶意行为模型配置文件: 为提高方法的可扩展性和通用性, MalDet 使用恶意行为模型配置文件来描述需要检查的恶意行为。为检测不同类型的恶意应用, 只需提供相应的恶意行为模型配置文件即可。

基于 AAPT 反编译: 本方法利用 Android SDK 自带的 AAPT 工具对 Android 应用 APK 文件进行反编译, 提取应用中的 AndroidManifest.xml 文件和 smali 文件夹, 具体包括以下步骤:

- 对 Android 应用 AndroidManifest.xml 进行分

① MalDet: https://github.com/malwareTesting/Android_Malware
② DroidBox: https://github.com/pjlantz/droidbox

析, 得到应用的包名和启动页面, 后续疑似恶意应用动态确认过程需要使用上述信息;

● 对 AndroidManifest.xml 文件进行分析, 得到待测应用的需要获取的权限。此外, 还会提取各组件使用情况, 例如 receiver 等, 为后续疑似恶意应用确认过程提供参数;

● 将.dex 文件反编译为 smali 文件夹, 作为静态分析的主要对象。

基于 DroidRA 检测变量赋值反射调用: 由于部分恶意应用尝试使用反射机制躲避静态扫描检测, 在本方法中, 通过反射实现的敏感 API 调用受到了重点关注。其中, 我们除实现了检测通过数组赋值方式实现的反射调用以外, 还在 DroidRA 的基础上进行修改和扩展, 检测通过变量赋值方式实现的反射调用, 主要包括:

● 在 DroidRA 的基础上, 首先, 删除其中的获取应用的使用权限、包名、入口页面等初始化工作, 因为这类工作在反编译阶段已经完成; 其次, 为提高检测效率, 不再把所有反射函数调用结果输出到文件, 直接检测是否存在恶意行为相关的敏感 API 调用;

● 调用 DroidRA, 其中给定两个参数: 待检测 Android 应用(APK 格式或 ZIP 格式)及对应的 android.jar。

基于 MonkeyRunner 进行自动化测试: 当前已经有多种针对 Android 应用的自动化测试工具[26]。由于 DroidRA 是基于 MonkeyRunner 进行自动化测试的, 为此, 从一致性角度出发, 我们最终选用 MonkeyRunner 作为自动化测试工具, 生成 Python 语言格式的测试脚本。

可疑度分析: 对实验数据中的 406 个第一类恶意吸费软件进行统计, 结果表明其中 151 个应用的恶意行为是由 ID 包含了关键字“accept”、“yes”、“next”的按钮触发的, 为此本文在实现中将这三个关键字作为我们的可疑控件 ID。在生成测试脚本时, 本文优先执行按钮控件 ID 中包含这些可疑名字的; 若没有任何按钮控件 ID 包含这些关键字, 则按照按钮控件在 XML 文件中的顺序执行。

基于 DroidBox 监测应用执行: 为追踪应用在动态执行过程中敏感 API 的调用情况, 需要实时监测整个执行过程, 一旦发现确实存在调用敏感函数的情况, 即停止运行并报告为恶意应用。我们对 Android 应用检测工具 DroidBox 进行了修改, 使其更适用于本方法, 具体步骤如下:

● 将其中对 AndroidManifest.xml 的分析删除, 因为反编译阶段已经完成相关工作, 不用重复分析;

● 将其中与要检测的恶意行为无关的内容删除, 因为 DroidBox 是个综合性工具, 关注的应用执行过程中的行为信息较多, 本方法只需要关注指定敏感 API 调用, 不需要关注所有运行信息;

● 原 DroidBox 将无限运行, 直到手动停止, 这与实际测试需求不符。为此, 我们指定测试终止时间为检测到恶意行为或执行完所有测试脚本后 3 秒。

4.2 实验设计

为评估静动态结合方式检测恶意软件的效率和准确率等方面情况, 本文采取对真实的恶意扣费短信应用和非恶意应用进行了实验, 分别采用静态分析、动态测试和静动态结合的恶意应用检测 3 种方法对待检测程序进行分析, 并且从检测时间、误报和漏报情况等方面来进行比较。

4.2.1 度量参数

为对实验结果进行有效的量化分析, 引入如下参数: *t_pos* (true positives)表示恶意应用被正确识别为恶意应用的数目; *t_neg*(true negatives)表示正常应用被正确识别为正常应用的数目; *f_pos*(false positives)表示正常应用被错误识别为恶意应用的数目; *f_neg*(false negatives)表示恶意应用被错误识别为正常应用的数目。

实验结果通过误报率和漏报率进行评价。公式(1)为误报率计算方法, 即为误报数/所有非恶意应用数, 公式(2)为漏报率的计算方法, 即为漏报数/所有恶意应用数。

$$FP\_Rate = \frac{f\_pos}{(t\_pos+f\_pos)} \tag{1}$$

$$FN\_Rate = \frac{f\_neg}{(t\_neg+f\_neg)} \tag{2}$$

4.2.2 实验对象

据 360 公司发布的《2016 年安卓恶意软件专题报告》[3], 2016 年 Android 平台新增的 1403.3 万个恶意程序样本中, 高达 90.7%的恶意应用与资费消耗和恶意扣费相关, 其中恶意扣费短信是最常见的一类资费消耗和恶意扣费应用。有鉴于此, 我们针对发送恶意扣费短信行为进行了实验。如果要实现其他恶意行为的分析, 那么只需从恶意行为提取出对应的敏感 API, 然后将其输入恶意行为配置文件, 即可利用 MalDet 分析应用中是否存在相应恶意行为。

在此次实验中, 我们提取出 sendTextMessage、sendMultipartTextMessage 等敏感 API 作为恶意行为模型输入配置文件, 以检测应用中隐藏的恶意扣费

短信行为。

恶意扣费短信应用一共包含两种类型[9,27]: 安装时刻发送恶意扣费短信和应用运行过程中发送恶意扣费短信, 本文主要检测第一类恶意吸费短信应用, 下面具体介绍下这两种类型的表现形式和差别:

(1)安装时刻触发恶意吸费行为(以下简称第一类恶意应用): 在应用安装时刻添加一个引导界面, 通知用户是否确定安装, 一旦用户点击“确定安装”, “下一步”类按钮, 恶意扣费行为即触发, 此时, 应用已经在用户不知情下发送恶意短信或者拨打恶意电话。如图 11 所示, 当用户点击“Next”按钮之后将会发送恶意付费短信。

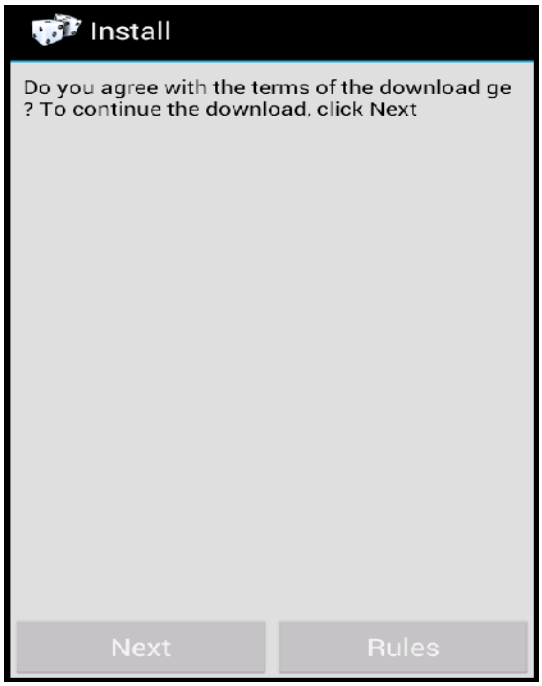

图 11 恶意扣费短信应用安装界面

(2) 运行过程中触发恶意吸费行为(以下简称第二类恶意应用): 在应用执行中某个操作触发恶意扣费行为, 如果电信商发来通知短信的话, 恶意软件一般会进行阻塞, 确保用户不知情。如图 12、13 所示, 运行过程中恶意吸费行为通过调用 sendTextMessage API, 向订阅号码发送短信, 并通过重写 onReceive 方法阻塞系统向用户提示收到的订阅成功短信。

第一类恶意吸费短信应用相对于第二类恶意吸费短信应用更普遍, 据文献[9]统计, 恶意吸费短信应用中第一类占 87.3%, 为此, 目前我们的工具暂时只支持对第一类恶意吸费短信应用进行检测, 可通过扩展自动测试脚本生成方法, 模拟对应用运行过程中进行随机测试, 以尝试触发第二类的恶意吸费行为。

```
SmsManager localSmsManager = SmsManager.getDefault();
localSmsManager.sendTextMessage("1066185829", null,
"921X1", intent);
```

图 12 恶意扣费短信应用调用敏感 API

```
Public class SmsReceiver extends BroadcastReceiver
Public void onReceive(Context paramContext, Intent
paramIntent)
If (( "10086".equals(str)) || ( "1000" .equals(str)))
abortBroadcast();
```

图 13 恶意扣费短信应用阻塞电信商通知

4.2.3 实验数据集

为了进行实验, 需要选取大量的数据集, 从全面性和适用性角度出发, 本文选择了文献[9]中的恶意数据集, 其中一共包含了 5560 个恶意软件, 涵盖了 179 种不同的恶意软件类型。本文将其中含有发送恶意扣费短信行为的所有恶意软件作为实验对象, 共计 465 个。其中 406 个为前面提到的第一种类型恶意吸费软件, 59 个为前面提到的第二种类型恶意吸费软件。

同时, 本文还爬取了豌豆荚①上电话通讯类、社交类、生活类、视频类、旅游类、购物类 6 种不同类型的应用共 600 个, 其中每种类型分别爬取了用户数量最多的前 100 个应用; 同时, 从安卓市场②上爬取了最新发布的 500 个应用。除去 15 个不能正常安装的应用后, 使用恶意软件检测工具 VirusTotal③、Virscan④进行扫描确认, 均为正常软件, 一共收集到 1085 个非恶意应用。

4.3 实验结果分析与评价

在实验过程中, 我们将实验数据集分别采用静态分析、动态测试和静动态结合的恶意应用检测 3 种方法进行检测, 并且根据时间及准确度量化指标进行分析比较。

① 豌豆荚: http://www.wandoujia.com/apps
② 安卓市场: http://apk.hiapk.com/apps
③ VirusTotal: https://www.virustotal.com/
④ Virscan: http://www.virscan.org/

4.3.1 效率

表 1 对 3 种检测方式对实验数据集中的 1550 个 Android 应用进行检测所需的运行时间进行比较。从表中可以看出, 静态分析方法所需时间最短, 动态测试方法所需时间最长, 静动态结合的方法比单独使用动态测试节省了 34.8%的时间。

这是因为静态分析方法不需要实际运行程序, 较为快速, 而实际执行应用时间消耗较多, 尤其是对于大型应用更加明显。静态分析方法虽然消耗时间少, 但并不能实际触发和确认恶意应用。采用静动态结合的方式可以首先利用静态分析方法排除大量正常应用, 只针对静态分析报告的疑似恶意应用进行动态执行。如在此例中, 静态分析排除了 1027 个正常应用, 只需要对 523 个应用进行动态确认。因此, 静动态结合的方法和完全使用动态测试相比起来节省了大量时间。

表 1 运行情况

	静态分析	动态测试	静动态结合
运行时间(s)	34100	108500	70710
静态分析应用数	1550	0	1550
动态测试应用数	0	1550	523

4.3.2 准确度分析

表 2 对 3 种方法的误报和漏报情况进行比较。此外, 工具目前只支持第一类恶意吸费短信应用的动态确认, 为此, 本文对结果进行了分开统计, 表中(a)(b)分别表示第一类恶意吸费短信应用和两类恶意吸费短信应用的结果。

静态分析方法误报率为 5.3%, 因为该方法将所有符合敏感 API 调用方式的应用都归为恶意软件。在实验数据集中, 例如短信助手、安卓短信等 58 个正常应用被该方法作为恶意应用报告。经人工核查发现, 原因为上述应用也调用了发送短信的 API。而动态测试或静动态结合的方法因为需要实际触发应用发送恶意扣费短信行为, 才会将该应用报告为恶意应用, 因此, 能在很大程度上避免误报。经人工核查, 动态测试和静动态结合的方法在实验数据集中不存在误报情况。

动态分析和静动态结合的方法漏报了 85 个恶意应用, 其中包含 26 个第一类恶意吸费短信应用, 59 个第二类恶意吸费短信应用, 本文目前主要针对的是第一类, 其漏报率为 6.4%, 两类恶意应用总漏报率 18.3%。而静态分析方法在实验中报告了所有恶意应用。这是因为静态分析的方法代码覆盖率高, 在此实验中, 将所有使用了敏感 API 的应用都报为恶意软件, 因此没有产生漏报, 但将会导致误报率较高。而动态测试方法根据生成的测试脚本自动执行应用, 如果运行过程中监测到实际调用了发送短信的 API, 且接收号码为订阅号, 则确认为恶意吸费短信应用。动态测试代码覆盖率较低, 所生成的测试用例难以确保能有效触发应用中隐藏的恶意行为。此外, 我们的原型工具目前只支持动态确认第一类恶意吸费短信应用。因此, 动态分析和静动态结合的方法漏报率较高。

从上面的数据和分析可以看出, 通常情况下, 静态方法误报率较高, 动态测试和静动态结合的方法能有效避免误报, 但由于其覆盖程度不如静态方法, 会不可避免的增加漏报。

表 2 误报及漏报情况对比

	静态	动态		静动态结合	
		(a)	(b)	(a)	(b)
误报数	58	0	0	0	0
误报率	5.3%	0%	0%	0%	0%
漏报数	0	26	85	26	85
漏报率	0%	6.4%	18.3%	6.4%	18.3%

(注: 动态测试和静动态结合方法中, (a)列只统计第一类恶意吸费短信应用, (b)列统计两类恶意吸费短信应用)

4.3.3 反射调用数据分析

此次实验的目标是检测应用是否为恶意吸费软件, 我们将 sendTextMessage 等相关敏感 API 作为检测目标, 若检测到的反射方法中包含此类 API, 则认为包含恶意行为。

使用 3.2 节中介绍的反射调用分析方法对实验数据集中的 1550 个 Android 应用进行敏感 API 分析, 发现其中有 294 个应用利用了反射机制调用 API, 占比达 19.0%。若是采取直接扫描 smali 代码的方式, 将无法检测到这些 API 的调用。

在检测出的 380 个第一类恶意吸费应用中, 有 11 个采取数组赋值的方式进行反射调用, 占比 2.9%。这类恶意行为是 DroidRA 工具不能检测到的。

因此, 本文提出的方法能够检测出比较隐蔽的利用反射机制调用敏感 API 的恶意软件, 这是目前很多恶意软件检测工具所忽略的部分。

4.4 有效性影响因素分析

基于静动态结合的恶意 Android 应用自动检测技术有效提高了检测效率和准确度, 但该方法的有效性还受到以下方面的限制:

(1) 方法的通用性。目前本文只针对吸费短信这一类恶意应用实现了原型工具, 并进行了实验, 此方

法对其他类型恶意应用的适用性还需要进一步验证;

(2) 实验数据集的代表性。目前我们的实验数据中, 恶意数据集主要来源于文献[9], 该数据集主要集中在 2014 年以前, 可能会缺少最新出现的恶意应用, 而且在类型上还只集中于恶意扣费类应用, 需要进一步提高数据集的多样性和代表性。

5 相关工作

由于 Android 系统的日益流行, 以及其系统的开源特性使得系统遭受了大量恶意软件的攻击, 该问题也引起了国内外信息安全领域诸多学者的注意, 目前对如何预防以及检测手机恶意应用程序这一课题也已经做出了大量的分析和研究工作。卿斯汉[28]对当前的 Android 安全进行了总结, 对恶意软件进行了归类, 并且指出目前的恶意软件检测方式主要可以分为静态和动态。而两种分析方法一般都会基于特征, 其中, 两个主流方向是基于代码特征[29-38]和基于行为特征[39-44]。这些方法大多依赖大规模的静态或者动态分析技术, 然后利用得到的信息来判断一个应用是否为恶意应用。

5.1 基于静态分析的方法

首先, 静态检测技术是通过直接查看分析程序安装包、源码等方式, 而不必实际执行程序的一类检测技术, 该类方法以特征匹配为基础, 检测已知恶意代码的代码样本常见的方法有二进制程序分析、源码分析等。

Chen[6]从代码相似性的角度来判断一个应用是否为恶意应用, 他将一个应用的页面和方法分别映射为 v-cores 和 m-cores, 并且建立相应的已有应用的数据库值, 然后将应用的 m-cores 和 v-cores 和数据库中的进行比较, 以此来确认一个应用是否为恶意应用, 但是他们这种方式仅限于采用重打包的方式形成的恶意软件。Zhang[7]基于 API 的依赖关系构造 API 依赖图, 然后利用图相似性来和数据库中已有图进行相似度比较, 确认应用中是否包含恶意行为。采用静态分析的技术是比较快速的, 但是仅仅利用静态特征不全面, 因为有些恶意行为是在运行时刻触发的, 所以准确率有受到影响。

5.2 基于动态测试的方法

Xiao[8]采用动态检测技术, 利用 Monkey 产生 1000 个随机事件去执行应用, 然后在这个过程中用 strace 工具追踪进程获得所有的系统调用函数, 然后以此作为特征来分析应用是否包含恶意行为。Abela[45]、Teubert[46]也采用类似的思想从系统调用的角度来区分恶意软件和非恶意软件。Cho[47]通过分析大量恶意软件的 API 调用序列得出一个模式, 然后通过动态执行一个应用将这个过程中得到的 API 序列和恶意模式进行匹配, 以此来确认一个应用是否为恶意软件。通过动态检测的方式可以覆盖到那些只有在运行时刻才触发的恶意行为, 但是很难保证测试用例完全, 而且这种方式开销大。

Huang[48]采用静动态结合的方式检测 Virustotal 提交者上传的应用是否为恶意软件, 静态分析方法和页面, 类似于文献[6]中的方法, 然后进行包分析, 主要是分析其中的时间戳和证书信息等来初步判定应用是否为潜在恶意应用, 最后动态分析来确认恶意行为。这样静动态结合的方式能够有效的综合两种方式的优点和减小缺点, 但是该方法是针对 Virustotal 中的恶意软件检测, 有着局限性。

5.3 基于数据挖掘的方法

最近基于数据挖掘的方式检测恶意软件比较流行。Arp[9]分析恶意应用的多个静态特征例如 permission, app components, filtered intents 等, 然后根据这些特征构造 SVM 分类器, 当检测一个应用时, 只需将对应的特征放入分类器中检测即可得到结果。Burguera[10]利用动态方法提取动态特征, 然后利用 K-means 算法来区分恶意代码和非恶意代码, 但是该方法局限于检测同一个应用的不同版本变种。杨欢等人[49]对应用程序进行自动化静态分析, 提取各个组件特征和函数调用序列; 采用沙盒技术动态分析, 自动提取各个应用程序的系统调用序列特征, 然后综合两者特征提出三层混合系综算法(第一层选出最优分类器, 第二层分类效果提升, 第三层进行综合评判)进行恶意软件检测。同样的还有很多基于机器学习的方法[50,51]去检测恶意软件。这类方法需要大量的已有恶意软件作为训练集, 而且这类方法无法检测新类型的恶意软件。

6 结论

针对现有的静态检测恶意软件误报率高, 动态检测效率低的问题, 本文采用静动态结合的方式检测恶意 Android 应用, 静态分析 smali 代码, 查看应用中是否直接调用敏感 API, 或通过变量赋值及数组赋值方式反射调用敏感 API, 以此来判定一个应用是否为疑似恶意应用; 再根据可疑度分析结果生成测试脚本, 驱动应用执行, 并对其运行行为信息进行监测, 来确认该应用中是否有恶意行为被触发; 最后, 本文实现了原型工具, 并针对含有恶意扣费短信行为的 Android 应用进行了实验。

在本文研究基础上, 我们计划下一步工作优化

可疑度分析, 利用 UI 控件和 Android 代码映射分析方法, 将静态分析阶段识别的敏感 API 调用代码与对应的 UI 控件建立关联, 从而进一步提高动态确认效率。此外, 我们将原型工具扩展到检测更多类型的恶意行为, 并将结合机器学习方法优化 UI 控件可疑度计算方式, 以提高恶意行为检测效率和准确度。

参考文献

[1] 中国互联网络信息中心, 第 39 次《中国互联网络发展状况统计报告》, 2017 年 1 月, http://www.cnnic.net.cn/hlwfzyj/hlwxzbg/hlwtjbg/201701/P020170123364672657408.pdf

[2] 中国互联网络信息中心, 2015 年中国手机网民网络安全状况报告, 2016年9月, http://www.cnnic.net.cn/hlwfzyj/hlwxzbg/ ydhlwbg/201610/P020161012494271880676.pdf

[3] 中国互联网协会, 国家互联网应急中心,《中国移动互联网发展状况及其安全报告》, 2016 年 5 月

[4] 360 手机卫士,《2016 年安卓恶意软件专题报告》, 2017 年 2 月

[5] 国家计算机网络应急技术处理协调中心,《移动互联网恶意代码描述规范》, YD/T 2439-2012, 行业标准.

[6] K. Chen, P. Wang, Y. Lee, XF. Wang, N. Zhang, HQ. Huang, W. Zou and P. Liu. Finding unknown malice in 10 seconds: mass vetting for new threats at the Google-play scale[C]// *Usenix Conference on Security Symposium (USENIX)*. pp. 659-674,2015.

[7] M. Zhang, Y. Duan, H. Yin and Z. Zhao. Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs[C]// ACM, pp.1105-1116, 2014.

[8] X. Xiao, X. Xiao, Y. Jiang and R. Ye. Identifying Android malware with system call co-occurrence matrices[J].//*Transactions on Emerging Telecommunications Technologies*, pp.675-684,2016.

[9] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon and K. Rieck. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket[C]// Network and Distributed System Security Symposium. 2014.

[10] I. Burguera, U. Zurutuza and S. Nadjm-Tehrani . Crowdroid: behavior-based malware detection system for Android[C]// *ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, pp.15-26,2011.

[11] KASSNER, M. Google play: Android's bouncer can bepwned. http://www.techrepublic.com/blog/it-security/-googleplay-androids-bouncer-can-be-pwned/, 2012.

[12] F-SECURE. F-secure: Internet security for all devices. http:// fsecure.com, 2014.

[13] F-SECURE. Threat report h2 2013. Tech. rep., f-secure, http://www.f-secure.com/documents/996508/1030743/Threat Report H2 2013.pdf, 2014.

[14] IR. Forman and N. Forman. Java Reflection in Action[J]. // Willodom Com, pp.296–326.2004

[15] M. Kazdagli, L. Huang, V. Reddi, and M. Tiwari. Morpheus: Benchmarking computational diversity in mobile malware. //*In Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*. pp.1–8. 2014.

[16] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio and C. Platzer. ANDRUBIS -- 1,000,000 Apps Later: A View on Current Android Malware Behaviors[C]// *Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. pp. 3-17.2014.

[17] Paulo Barros, Ren´e Just, Suzanne Millstein, Paul Vines, Werner Dietl, Marcelo d'Armorim, and Michael D. Ernst. Static analysis of implicit control flow: Resolving java reflection and android intents.// In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE,Lincoln, Nebraska, 2015.

[18] L. Li, Bissyand, Tegawend, D. Octeau and J. Klein. DroidRA: taming reflection to support whole-program analysis of Android apps[C]// *International Symposium on Software Testing and Analysis*. ACM, pp.318-329.2016.

[19] V. Rastogi, Y. Chen, and X. Jiang. Droidchameleon: Evaluating android anti-malware against transformation attacks. //*The 8th ACM SIGSAC Symposium on Information, Computer and Communications Security(ASIA CCS)*.pp.329–334.2013.

[20] L. Li, Bissyand, Tegawend, D. Octeau and J. Klein. Reflection-aware static analysis of Android apps[C]// *Ieee/acm International Conference on Automated Software Engineering*. ACM, pp.756-761.2016,

[21] U. Pehlivan, N. Baltaci, C. Acartürk and N. Baykal. The analysis of feature selection methods and classification algorithms in permission based Android malware detection[C]// *IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pp.1-8,2014.

[22] K. Chen, N. Johnson, V. D'Silva, S. Dai, K. MacNamara, T. Magrino, E. Wu, M. Rinard and D. Song. Contextual Policy Enforcement in Android Applications with Permission Event Graphs[J]. //*Symposium on Network and Distributed System Security (NDSS)*, pp.586, 2013.

[23] Xu K, Li Y, Deng R H. ICCDetector: ICC-Based Malware Detection on Android[J].//*IEEE Transactions on Information Forensics & Security,* pp.1252-1264,2016.

[24] C. Yang, Z. Xu, G. Gu, V. Yegneswaran and P. Porras, "DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in Android applications[J]" //*European Symposium on Research in Computer Security(ESORICS)*. pp. 163–182, 2014.

[25] Y. Aafer, W. Du and H. Yin . DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android[M]// *Security and Privacy in Communication Networks. pp*.86-103,2013.

[26] N. H. Saad and N. S. Awang Abu Bakar. Automated testing tools for

mobile applications[C]// *The International Conference on Information and Communication Technology for the Muslim World (ICT4M)*. IEEE, pp. 1-5,2015.

[27] C. Yang, V. Yegneswaran, P. Porras and G. Gu. Detecting money-stealing apps in alternative Android markets[C]//*ACM Conference on Computer and Communications Security (CCS)*. ACM, pp.1034-1036,2012.

[28] S.H Qing, Security research progress of Android, Journal of software,vol 27,no 1,pp.45-71,2016
(卿斯汉,“Android 安全研究进展”, 软件学报, 2016, 27(1): 45-71)

[29] K. Griffin, S. Schneider, X. Hu and TC. Chiueh. Automatic Generation of String Signatures for Malware Detection[C]// *Recent Advances in Intrusion Detection(RAID)* .pp.101-120.2009.

[30] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen and D. Song. Juxtapp: a scalable system for detecting code reuse among android applications[C]// *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. pp.62-81.2012.

[31] C. Platzer, M. Lindorfer, S. Volanis, A. Sisto, M. Neugschwandtner, E. Athanasopoulos, F. Maggi, S. Zanero and S. Ioannidis. AndRadar: Fast Discovery of Android Applications in Alternative Markets[C]// *Conference on Detection of Intrusions and Malware & Vulnerability Assessment*. pp. 51-71. 2014.

[32] S. Arzt, S. Rasthofer, C. Fritz, E. Boden, A. Bartel, J. Klein, YL. Traon, D. Octeau and P. McDaniel. Flow Droid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps[J]. //*Acm Sigplan Notices,* pp.259-269.2014.

[33] Y. Feng, S. Anand, I. Dillig and A. Aiken. Apposcopy: semantics-based detection of Android malware through static analysis[C]// *The, ACM Sigsoft International Symposium(SIGSOFT FSE)* .pp.576-587.2014.

[34] T. Vidas, J. Tan, J. Nahata, N. Christin and P. Tague. A5: Automated Analysis of Adversarial Android Applications[C]// *ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, pp.39-50.2014.

[35] M. Grace, Y. Zhou, Q. Zhang, S. Zou and X. Jiang. RiskRanker: scalable and accurate zero-day android malware detection[C]// *International Conference on Mobile Systems, Applications, and Services*. ACM, pp.281-294.2012

[36] Q. Zhang and D. S. Reeves. MetaAware: Identifying Metamorphic Malware[C]// *Twenty-Third Annual Computer Security Applications Conference (ACSAC)*. pp.411-420. 2007.

[37] W. Zhou, Y. Zhou, M. Grace, X. Jiang and S. Zou. Fast, scalable detection of piggybacked mobile applications.//*ACM Conference on Data and Application Security and Privacy (CODASPY)*. pp. 185-196. 2013.

[38] Y. Zhou., Z. Wang., W. Zhou and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alterna-tive android markets. //*The 19th Network and Distributed System Security Symposium (NDSS)*. 2012.

[39] W. Enck, P. Gilbert, BG. Chun, LP. Cox, J. Jung, P. McDaniel and AN. Sheth . TaintDroid: an information flow tracking system for real-time privacy monitoring on smartphones[C]// *Usenix Symposium on Operating Systems Design and Implementation(OSDI)*. pp. 1–6.2010.

[40] P. Gilbert, BG. Chun, LP. Cox and J. Jung. Vision: automated security validation of mobile apps at app markets[C]// *International Workshop on Mobile Cloud Computing and Services*. ACM, pp.21-26.2011

[41] A. Reina, A. Fattori and L. Cavallaro . A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors[J]. //Eurosec, 2013.

[42] C. Wu, Y. Zhou, K. Patel,Z. Liang and X. Jiang. AirBag: Boosting Smartphone Resistance to Malware Infection[C]// *Network and Distributed System Security Symposium(NDSS)*. Pp.23-26.2014.

[43] LK. Yan and H. Yin . DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis[C]// *Proceedings of the 21st USENIX conference on Security symposium(USENIX)*. pp.29-29.2012.

[44] V. Rastogi, Y. Chen and W. Enck . Appsplayground: Automatic security analysis of smartphone applications.//*The third ACM conference on Data and application security and privacy* . pp. 209–220. 2013.

[45] KJ. Abela, DK. Angeles, JRD. Alas and RJ. Tolentino. An automated malware detection system for android using behavior-based analysis amda[J].//International Journal of Cyber-Security and Digital Forensics, 2(2), 2013.

[46] D. Teubert, F,. Grossmann and U. Meyer . Anomaly-based Mobile Malware Detection: System Calls as Source for Features[C]// *International Conference on Information Systems Security and Privacy(ICISSP)*. pp. 26-36,2016.

[47] IK. Cho and EG. Im . Extracting representative API patterns of malware families using multiple sequence alignments[C]// Conference. pp.308-313,2015.

[48] H. Huang, C. Zheng, J. Zeng, W. Zhou, S. Zhu, P. Liu, S. Chari and C. Zhang. Android malware development on public malware scanning platforms: A large-scale data-driven study[C]// *IEEE International Conference on Big Data(Big Data)*. IEEE, pp.1090-1099, 2016.

[49] H.Yang,Y.Q Zhang,Y.P Hu and Q.X Liu, A malicious behavior detection system of Android applications based on multi class features, Journal of Computer Science,vol 37,no 1,pp.15-27,2014
(杨欢, 张玉清, 胡予濮, 刘奇旭, “基于多类特征的 Android 应用恶意行为检测系统”, 计算机学报, 2014, 37(1): 15-27.

[50] S. Wu, P. Wang, X. Li and Y. Zhang. Effective detection of android

malware based on the usage of data flow APIs and machine learning[J]. //*Information & Software Technology*, pp.17-25, 2016.

[51] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer and Y. Weiss. "Andromaly": a behavioral malware detection framework for android devices[J].//Journal of Intelligent Information Systems, pp. 161-190, 2012.

黄浩华 于 2015 年在南京大学计算机科学与技术专业获得学士学位。现在南京大学计算机科学与技术专业攻读硕士学位。研究领域为软件测试、软件安全。研究兴趣包括: 模型驱动自动化测试、恶意软件检测。Email: hhh@seg.nju.edu.cn

崔展齐 于 2011 年在南京大学计算机软件与理论专业获得博士学位。现任北京信息科技大学讲师。研究领域为软件工程、 信息安全。研究兴趣包括: 软件分析与测试技术。Email: czq@bistu.edu.cn

潘敏学 于2014年在南京大学计算机软件与理论专业获得博士学位。 现任南京大学助理研究员。研究领域为形式化方法和软件工程。研究兴趣包括嵌入式系统、实时系统与高可信系统的分析与验证。Email: mxp@nju.edu.cn

王林章 于 2005 年在南京大学计算机软件与理论专业获得博士学位。现任南京大学计算机科学与技术系教授。 研究领域为软件工程、 信息安全。研究兴趣包括: 软件安全性测试、模型驱动的软件测试及验证、自动化软件测试工具。Email: lzwang@ nju.edu.cn

李宣东 于 1994 年在南京大学计算机软件与理论专业获得博士学位。现任南京大学计算机科学与技术系教授。 研究领域为软件工程、信息安全。研究兴趣包括: 软件建模与分析、软件测试与验证、自动化软件测试工具。Email: lxd@nju.edu.cn