基于数据流深度学习算法的 Android 恶意应用检测方法

朱大立[1,2], 金 昊[1,2], 吴 荻[1], 荆鹏飞[1], 杨 莹[1,2]

[1] 中国科学院信息工程研究所第四研究室 北京 中国 100093
[2] 中国科学院大学网络空间安全学院 北京 中国 100093

摘要 目前针对未知的 Android 恶意应用可以采用机器学习算法进行检测, 但传统的机器学习算法具有少于三层的计算单元, 无法充分挖掘 Android 应用程序特征深层次的表达。文中首次提出了一种基于深度学习的算法 DDBN(Data-flow Deep Belief Network)对 Android 应用程序数据流特征进行分析, 从而检测 Android 未知恶意应用。首先, 使用分析工具 FlowDroid 和 SUSI 提取能够反映 Android 应用恶意行为的静态数据流特征; 然后, 针对该特征设计了数据流深度学习算法 DDBN, 该算法通过构建深层的模型结构, 并进行逐层特征变换, 将数据流在原空间的特征表示变换到新的特征空间, 从而使分类更加准确; 最后, 基于 DDBN 实现了 Android 恶意应用检测工具 Flowdect, 并对现实中的大量安全应用和恶意应用进行检测。实验结果表明, Flowdect 能够充分学习 Android 应用程序的数据流特征, 用于检测未知的 Android 恶意应用。通过与其他基于传统机器学习算法的检测方案对比, DDBN 算法具有更优的检测效果。

关键词 机器学习; Android 恶意应用检测; 深度学习; 数据流特征
中图法分类号: TP181/TN929.5 DOI 号 10.19363/J.cnki.cn10-1380/tn.2019.03.06

Android malware detection method based on data-flow deep learning algorithm

ZHU Dali[1, 2], JIN Hao[1, 2], WU Di[1], JING Pengfei[1], YANG Ying[1, 2]

[1] The 4th Laboratory, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100093, China

Abstract At present, machine learning algorithm is always used to detect unknown malicious applications of Android. As traditional machine learning algorithm has less than three computing layers, it could not fully mine the deep characterizations of features in an Android application. For this problem, a Data-flow Deep Belief Network Algorithm (DDBN) is proposed, which learns data flow features deeply to detect Android malware. Firstly, we combine the analysis tools FlowDroid and SUSI to extract static data flow features, which can reflect malicious behaviors of an Android application. Then, we design DDBN to construct a deep model and transform the data flow features from the original representation space to a new feature space layer by layer, so as to achieve higher classification accuracy. Finally, we implement an automated tool named Flowdect based on DDBN to detect a number of benign and malicious applications in real. The experimental results show that Flowdect can fully learn the data flow features to detect unknown Android malware. What's more, DDBN performs better than other machine learning-based approaches on the accuracy and efficiency.

Key words machine learning; android malware detection; deep learning; data flow feature

1 引言

市场调研机构 Gartner 发布的统计报告显示[1], 2016 年度全球 Android 智能设备的销量超过了 16 亿台。同时, 随着 Android 智能设备的普及, Android 应用市场快速发展。Sensor Tower 的数据显示[2], 在 2017 年第一季度, 用户从 Android 应用市场(如 Google Play 和第三方应用市场)下载的应用数量达到

通讯作者: 吴荻, 博士, 助理研究员, Email: wudi@iie.ac.cn。
本课题得到国家自然基金(No. 61701494), 中科院信工所青年之星(No. Y8YS016104), 和中国科学院战略性先导专项项目(No. XDA06010703)资助。
收稿日期: 2017-07-24; 修改日期: 2018-01-28; 定稿日期: 2019-02-27

172 亿。然而, Android 操作系统的开放性使其成为恶意应用开发者的活跃地盘。除此之外, 由于国内外第三方应用市场, 甚至 Google Play, 都缺乏严密的安全审核机制, 使得恶意应用开发者可以将其开发的恶意应用程序发布到应用市场中。一旦用户下载并安装了这些应用, 将会造成严重的危害。恶意应用通常采取资费消耗、隐私窃取和远程控制等手段实施攻击, 其中造成危害最大的恶意攻击手段就是窃取用户的隐私数据。安全机构 F-secure[3]揭示了 2015 年度被检测到的排名前十的恶意应用程序, 如: SMSSEND, SLOCKER, FAKEINST, GINMASTER, SMSPAY, SMSKEY 等, 均实施了恶意的敏感数据泄露攻击, 并给用户带来巨大的损失。

研究表明[4], 在恶意应用程序中, 敏感数据总是流向几种固定类型的数据泄露点(如: 短信和网络)。同时, 针对敏感数据, 恶意应用往往采取与安全应用不同的处理方式。基于此, 本文首先从大量的恶意应用和安全应用中提取敏感数据流, 然后对这些数据流特征进行分析, 构建检测模型, 用于判断未知应用程序的安全性。

在应用敏感数据流提取方面, 早期学术界主流的方法是动态污点追踪技术。该技术通过对敏感数据进行标记, 在应用程序运行期间跟踪其流向, 从而判断是否发生泄露[5,6]。然而这种机制存在一定的缺陷, 如果有部分应用行为没有被触发, 会导致提取的数据流信息不够全面。为了避免这个问题, 研究者提出了静态数据流分析技术[7-9]。该技术通过静态分析应用程序的源代码, 构建应用程序控制流图, 从而提取应用中可能存在的敏感数据泄露路径。

从应用程序中提取敏感数据流后, 需要对这些数据流特征进行分析以判定其安全性。因为机器学习算法能够从大量的数据中学习出有意义的信息, 研究者提出大量利用机器学习算法训练从应用程序中提取的特征从而判断其安全性的技术方案, 这些特征包括静态特征(如: 权限, API 调用, 静态数据流等), 以及动态特征(如: 可疑行为序列, 系统调用, 网络流量等)。学术界在恶意应用检测领域常用的机器学习算法包括 Naïve Bayes, Decision Tree, Kmeans, Support Vector Machine 和 Multi-Layer Perceptron 等。

基于机器学习算法的恶意应用检测技术能够自动检测未知的恶意应用程序。然而, 传统的机器学习模型具有少于三层的计算单元, 这种浅层结构在一定程度上影响了检测的准确率[10]。深度学习[11]作为机器学习的分支, 从 2006 年开始受到学术界的广泛关注以来, 已经成为互联网大数据和人工智能的一个研究热潮。深度学习通过建立类似于人脑的分层模型结构, 对输入数据逐级提取从底层到高层的特征, 从而很好地建立从底层信号到高层语义的映射关系。区别于传统的浅层学习, 深度学习强调了模型结构的深度, 通常有 5 层、6 层、甚至 10 多层的隐层节点; 同时, 深度学习明确突出了特征学习的重要性, 通过逐层特征变换, 将样本在原空间的特征表示变换到一个新特征空间, 从而使分类或预测更加容易。目前, 深度学习被广泛应用于图像分析, 语音识别, 自然语言处理等领域, 并取得了显著的成效。

由 Guy Caspi 和 Eli David 创立的 Deep Instinct[12]是第一个在安全领域应用深度学习算法的安全公司。他们通过训练深度学习模型分析应用程序的结构和函数, 进而实现对 PC 端恶意软件的检测。当前, 在学术界应用深度学习技术检测 Android 平台恶意应用的工作较少。文献[10]通过提取 Android 应用程序的权限、API 调用、动态行为等特征, 训练深度学习模型来区分恶意应用和安全应用。研究者选择了深度信念网络(DBN, Deep Belief Network)来构建深度学习模型并表征 Android 应用程序, 检测准确率高达 96.76%, 远远超越了传统的机器学习模型。

基于以上分析, 本文使用基于数据流的静态分析方法, 并结合深度学习算法实现 Android 恶意应用检测。首先, 利用静态数据流分析工具 FlowDroid[7]对 Android 应用程序进行自动分析, 提取其内部所有从敏感数据源(*source*)到敏感数据泄漏点(*sink*)的数据流。然后, 使用 SUSI 技术[13]对数据流特征进行处理, 并构建数据流特征库。最后, 使用深度学习算法对大量恶意应用与安全应用的数据流特征进行训练分析, 构建深度学习检测模型。每当用户提交一个待测应用程序时, 提取其数据流特征并处理成规范化的格式, 输入到训练好的深度学习检测模型中以判断其安全性。为了提高对不断涌现的新型恶意应用的检测率, 本文持续爬取来自 Google Play 的安全应用, 以及来自 VirusShare[14], Android Malware Gnome Project[15]等开源库的恶意应用作为深度学习检测模型的训练样本。

本文主要贡献如下: 首次将深度学习算法应用到基于数据流的 Android 恶意应用检测, 设计并提出了数据流深度学习算法 DDBN(Data-flow Deep Belief Network), 并根据提出的原型系统实现了自动化的检测工具 Flowdect。Flowdect 利用 DDBN 的深层架构, 能够更好地描述 Android 应用程序的数据流特征, 从而为用户提供安全抉择。本文设计了多个实验对从 VirusShare, Android Malware Genome Project 以及

Drebin 等开源库爬取的 5000 个恶意应用和从 Google Play 爬取的 5000 个安全应用进行训练和检测。同时, 通过对某安全厂商提供的最新恶意应用和实验室自主研发的恶意应用共计 90 个(不属于任何开源恶意应用库)进行检测, 验证了本文提出方法检测未知恶意应用的有效性。同时, 通过与基于传统机器学习算法的检测方案对比, 表明了数据流深度学习算法 DDBN 的优势, 为进一步应用深度学习算法进行基于其他特征的恶意应用检测提供了启发性思考。

本文第 2 节介绍相关工作; 第 3 节总体介绍检测系统 Flowdect; 第 4 节阐述系统的关键技术和算法, 包括: 静态数据流特征提取技术 FlowDroid、数据流特征处理技术 SUSI 以及数据流深度学习算法 DDBN; 第 5 节给出实验设计和结果分析; 第 6 节讨论本文提出的系统存在的局限性, 以及下一步工作; 最后第 7 节对本文进行总结。

2 研究现状

自 Fuchs 等人[16]在 2009 年首次提出基于签名的检测技术以来, 该项技术被广泛应用于各种商用病毒扫描工具中。基于签名的检测技术能够准确地检测已知的恶意应用, 并使用较少的计算资源, 但不能检测未知的恶意应用或恶意应用变体。同时, 签名提取的过程需要人工参与, 效率不高。

考虑到这些问题, 近年来国内外研究人员提出很多新的检测技术。这些技术在进行恶意应用检测时大体可以分为两个步骤。第一步分析应用程序, 提取相关信息, 根据获取应用信息的方式是否需要运行应用作为标准, 可以将恶意应用检测技术分为静态检测技术和动态检测技术。第二步处理第一步提取的信息, 从而判断应用的安全性。根据处理应用信息方式, 可以将恶意应用检测技术分为基于启发式的检测技术和基于机器学习的检测技术。

基于启发式的检测技术通过分析应用程序本身, 或动态运行应用程序, 检查其所有可能的执行路径, 并收集这些路径的执行结果。根据分析方式的不同可以分为语法分析、控制流分析和数据流分析。在基于静态分析的启发式检测技术方面, Enck, W.等人[17]最早在 2009 年提出一个开源的静态分析工具 Androguard。Androguard 能够反编译 Android 应用, 提供其静态分析结构, 包括基本模块, 指令和每个方法的控制流图。2014 年, Wei, F.等人[8]提出一个 Android 应用静态分析框架 Amandroid, 通过捕捉应用的语义行为, 如组件间数据流和控制流信息, 对 Android 应用进行安全审计。Amandroid 可以处理一个或多个应用中的多个组件交互引起的安全问题。2015 年, Debiaze, T.等人[18]提出静态代码分析工具 Androwarn, 主要用于检测 Android 应用潜在的恶意行为。Androwarn 对应用程序的 Dalvik 字节码进行静态分析, 可以发现以下几种恶意行为: 设备信息泄露、位置信息泄露、网络连接信息泄露、Telephony 服务滥用以及 Audio/Video 被监听等。为了对抗静态分析存在的混淆代码和本地代码无法检测的问题, 近几年研究者们先后提出了一些解决方案[19-21]。

在基于动态分析的启发式检测技术方面, 2009 年, Enck, W.等人[5]提出了一个污点跟踪系统 TaintDroid。TaintDroid 通过修改 Android 操作系统, 能够追踪应用程序的敏感数据泄露。TaintDroid 实现四个层面的追踪: 变量级, 方法级, 信息级和文件级。基于 TaintDroid 的染色分析技术, 出现了大量动态行为分析工具, 如: DroidBox[22]、AppsPlayground[23]和 DroidScope[24]。2016 年, Wong 等人[25]提出一种动态分析工具 IntelliDroid, 通过生成可疑 API 的集合, 能够根据执行路径触发可疑 API, 从而获取更全面的动态行为。另外一些动态行为分析工具基于系统调用和 Binder 通信行为重构应用的高级语义行为, 不需要修改 Android 操作系统, 如 CopperDroid[26]。

在基于动态特征的机器学习检测技术方面, 2013 年, Ham, H. S.等人[27]将应用行为特征分为 7 类: 网络、短信、CPU 和电量消耗、进程信息(ID、进程名等)、本地内存、Dalvik 内存以及其他虚拟内存, 并使用 Naïve Bayes, Random Forest, 10-cross SVM 进行训练分析。结果表明, SVM 在判定安全应用方面能取得接近 100%的准确率, 但是存在将恶意应用当作安全应用的情况。Random Forest 在所有算法中表现最好。2014 年, Sahib, S.等人[28]将应用的系统调用作为特征, 并运用 KNN, Decision Tree, Multi-Layer Perceptron, Random Forest, Naïve Bayes 五种分类器进行训练。研究对比发现, MLP 的准确率最高。2016 年, Narudin, F. A.等人[29]评估了五种类型的分类器在恶意应用检测方面的效率, 包括 Decision tree J48, Bayes Network, MLP, kNN 和 Random Forest。以从 Android Malware Genome Project 获取的 1000 个恶意应用为样本, 研究利用上述算法来分析应用程序的网络流量特征。

在基于静态特征的机器学习检测技术方面, 2014 年, Arp, D.等人[30]提出了 DREBIN, 通过收集应用静态特征, 包括权限、应用组件、可疑 API 调用、网络地址等, 将这些特征存储在一个联合向量空间,

然后采用基于机器学习的检测方法, 自动识别一些典型的恶意应用模式。DREBIN 在静态分析时并没有对代码进行深入的分析, 因此比较轻量级, 可以直接在 Android 终端上检测恶意应用。2015 年, Avdiienko 等人[4]发现恶意应用的敏感数据流与安全应用存在很大的不同。基于此, Avdiienko 利用 FlowDroid 提取 2866 个安全应用的“正常数据流集合”, 通过将不同应用的敏感数据流与该集合的几何差作为特征训练一个恶意应用判别模型 MUDFlow。MUDFlow 对于新型的恶意应用具有 86.4%的检测率, 并发现 90.1%的新型恶意应用存在泄露隐私数据的行为。在 MUDFlow 的基础上, Pengbin Feng 等人[31]指出, 某些关键性的数据流通常会同时出现在安全应用和恶意应用中, 将这些数据流作为特征进行训练将不利于模型的准确率。同时, 另外一些关键性的数据流出现在恶意应用和安全应用的频率存在很大的差异。仅将这些关键性的数据流作为模型训练的特征, 能够在很大程度上提高模型检测的准确率。基于以上发现, Pengbin Feng 设计了特征选择算法 CflowSel, 并训练了一个改进的恶意应用判别模型 SCDFlow。当测试集与 MUDFlow 相同时, SCDFlow 具有高于 MUDFlow 5.73%的检测率。2016 年, Yuan, Z.等人[10]首次将深度学习算法应用于移动恶意应用检测, 提出一种新型的恶意应用检测模型 DroidDetector。DroidDetector 首先提取应用程序的权限和 API 调用等特征, 然后将特征数据处理后输入深度学习模型 DBN 中进行训练, 得到最终的检测模型。研究表明, 当采用深度学习算法时, 检测准确高达 96.76%, 远远高于传统浅层的机器学习算法。2017 年, F. Idrees 等人[45]将集成学习算法用于 Android 恶意应用检测, 并提出一种新型的检测工具 PinDroid。PinDroid 提取应用程序的权限和 intent 组件信息作为分类特征, 并使用集成学习算法优化检测结果。

基于以上的分析, 本文将深度学习算法应用到 Android 平台恶意数据流检测中, 通过对 Android 应用程序的数据流特征进行训练分析, 判断其安全性。

3 Flowdect 检测系统

本文提出了一种基于深度学习算法的 Android 恶意应用检测原型, 并实现了检测系统 Flowdect, 可以利用数据流特征自动化地检测未知的恶意应用程序。该系统首先利用静态数据流分析工具 FlowDroid 提取 Android 应用程序的数据流特征, 其后用敏感 API 分类工具 SUSI 对数据流特征进行处理, 最后基于数据流深度学习算法 DDBN 对数据流特征进行训练, 从而构建 Android 应用程序自动分类模型, 实现对恶意应用的检测。

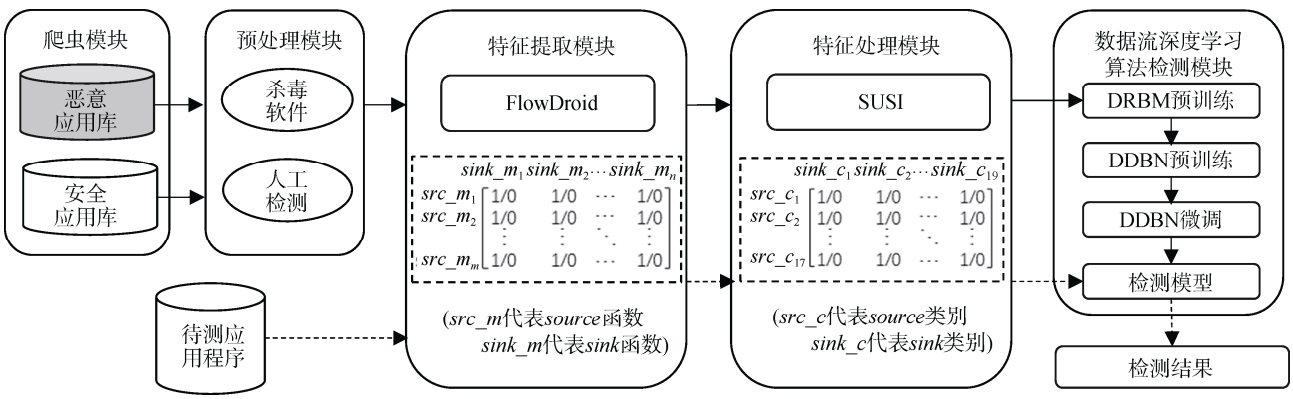


图 1 Flowdect 检测系统框图

Figure 1 Flowdect detection system diagram

Flowdect 检测系统框图如图 1 所示, 系统由 5 个功能模块组成: 爬虫模块、预处理模块、特征提取模块、特征处理模块和数据流深度学习算法检测模块, 分别介绍如下:

(1) 爬虫模块。本文设计了两个爬虫模块, 并分别创建了安全应用库和恶意应用库。其中, 安全应用从 Google Play 爬取, 并覆盖了各类别中当前最流行的应用程序; 恶意应用库主要包括 Android Malware Genome Project 和 VirusShare 等数据源提供的部分恶意应用, 以及 2016 年度最新被检测到的 100 个恶意应用。

(2) 预处理模块。文献[32]发现 Google Play 上的 52208 个应用程序中只有 2 个为恶意应用, 因此可以认为 Google Play 上的应用程序基本属于安全应用。但是为了进一步确保结果的准确性, 本文对从 Google Play 上批量爬取的应用程序使用杀毒软件进行预处理, 以确保安全应用库中的应用程序均为安全应用。

(3) 特征提取模块。本文利用静态数据流分析工具 FlowDroid 提取 Android 应用的数据流特征。本文认为, 从安全应用中提取的数据流特征反映了敏感数据的“正常”处理方式, 相反, 从恶意应用中提取的数据流特征揭示了敏感数据的“异常”使用方式。因此, 本文针对安全应用库和恶意应用库中的应用程序分别提取数据流特征。

(4) 特征处理模块。FlowDroid 提取的数据流特征是函数级的, 由此构建的特征矩阵过于稀疏, 不利于训练。因此, 本文使用敏感 API 分类工具 SUSI 将数据流特征转换成类别级, 依此对提取的数据流特征集合进行处理。

(5) 数据流深度学习算法检测模块。使用数据流深度学习算法 DDBN 对特征数据进行训练, 创建最优的检测模型, 实现对待测应用程序的检测。

4 关键技术与算法

4.1 数据流特征提取技术

静态数据流分析技术通过构建应用的函数调用图, 并对其中可到达函数进行逐一分析, 将源信息进行传播来监控其在整个系统中的信息流流向。静态数据流分析技术存在以下两大挑战: (1)需要对应用运行流程进行准确的建模; (2)由于 Android 应用程序运用了大量的组件间通信, 因此需要准确地获取组件间通信的目标组件。Arzt, S.等人[7]提出的开源静态数据流分析工具 FlowDroid 有效地解决了以上难点, 因此本文基于 FlowDroid 提取应用的静态数据流特征。

一个 Android 应用程序包含多个组件, 如 Activity, Service, Content Provider, Broadcast Receiver, 其中 Activity 是静态数据流分析的主要入口。与传统的 Java 程序不同, Android 应用程序中不存在主函数, 因此在分析的过程中, 不能简单的通过主函数找到程序的入口和出口来构建控制流图。但是, Android 应用程序的每个组件都有函数来反映该组件的生命周期, 可以根据这些组件的生命周期构建 Android 应用程序的控制流图。

图 2 描述了 Activity 组件的生命周期。从图中可以看出, Activity 组件可以从任何节点按照任何顺序执行, 并不是传统的单入口单出口结构。除了 *onCreate()*, *onStart()*, *onDestroy()*等生命周期函数, Activity 组件还带有各种事件处理器的回调函数, 用于处理 UI 交互等事件。为了生成应用程序的控制流图, FlowDroid 将 Activity 生命周期和所有回调函数之间的控制流调用关系用一个虚主函数来模拟。下面, 用一个例子来详细描述 FlowDroid 的数据流提取过程。

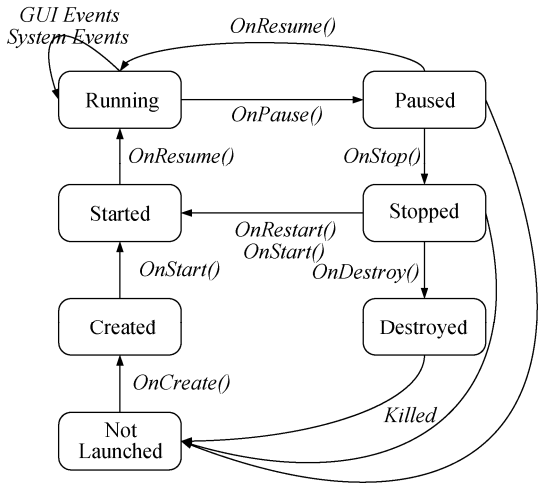


图 2 Activity 组件生命周期

Figure 2 The life cycle of Activity

图 3 所示的应用程序通过调用百度地图提供的 API, 读取用户的位置信息, 包括经度、维度以及详细的地址, 然后通过短信将位置信息发送到手机号“+1 11”上。这个简单的程序反映了常见的隐私攻击方式, 敏感数据从 *LocationClient (source)*流向 *SMS API (sink)*, 从而导致了隐私泄露。

```
public class Location extends Application {
  LocationClient mLocationClient = new LocationClient(this);
  TelephonyManager tm = (TelephonyManager)
     this.getSystemService(TELEPHONY_SERVICE);
  public class MyReceiveListenner implements ReceiveListener {
    public void onReceive(String strData) {
      JSONObject jsonObject = new JSONObject(strData.trim());
      JSONObject jsonjingweidu =
         jsonObject.getJSONObject("content").getJSONObject("point");
      JSONObject jsonaddr =
         jsonObject.getJSONObject("content").getJSONObject("addr");
      String longitude =jsonjingweidu.getString("y");
      String latitude =jsonjingweidu.getString("x");
      String detail = jsonaddr.getString("detail");
      String location = loc(longitude,latitude,detail);
      SmsManager sm = SmsManager.getDefault();
      sm.sendTextMessage("+1 11",null,location,null,null);
    }
  }
  String loc(String a,String b,String c){
    return l = "longitude"+a+"latitude"+b+"address"+c;
  }
```

图 3 Android 恶意应用攻击案例

Figure 3 The case of Android malicious application attack

FlowDroid 的数据流分析过程是与执行顺序相关的, 分为两个部分: 一部分是向前的污点分析, 用于找出被污染的变量传递到了何处; 一部分是向后的请求式别名分析, 用于查找 *sink* 之前所有对同一被污染的 heap 位置的别名。如图 4 所示, FlowDroid 采用以下的步骤提取图 3 所述应用程序中从 *source* 到 *sink* 的敏感数据流:

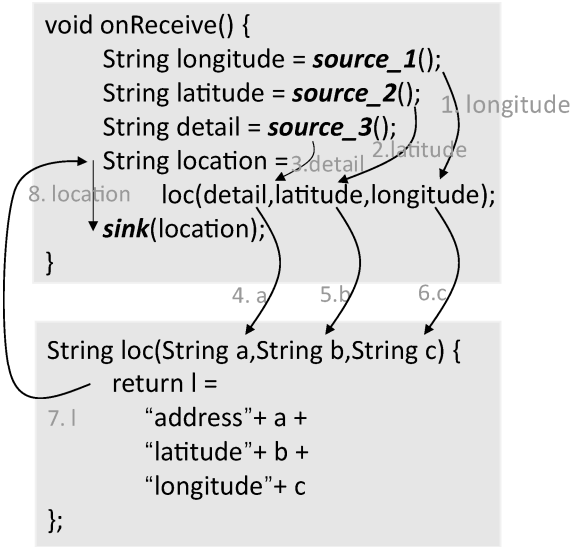


图 4 FlowDroid 提取数据流案例

Figure 4 The case of obtaining data flow by FlowDroid

步骤 1. 从 *source* 获取的 *longitude*, *latitude* 和 *detail*, 作为被污染了的变量, 向前传递给函数 *loc*(*detail, latitude, longitude*);

步骤 2. 当函数 *loc*(*detail, latitude, longitude*)被调用时, *longitude*, *latitude* 和 *detail* 是实际参数。执行后向分析, 发现对应的形式参数分别是 *a*, *b* 和 *c*, 因此对 *a*, *b* 和 *c* 进行污染。

步骤 3. 继续追踪 *a*, *b* 和 *c*, 对 *l* 进行污染;

步骤 4. *l* 是被调用函数的返回值, 继续进行后向分析, 对 *location* 进行污染;

步骤 5. 继续前向分析, 发现 *location* 被传递到 *sink*。

基于以上步骤, FlowDroid 完成了对图 3 所述应用程序内部敏感数据流的提取。面对实际环境中更复杂的隐私窃取场景, FlowDroid 采取了以下三种技术手段: (1)FlowDroid 的分析过程是上下文相关的, 因此能够有效地区分对同一函数基于不同参数的不同调用; (2)FlowDroid 采用了特殊的方式处理对库函数的调用; (3)FlowDroid 采用大量的优化手段拓展操作规模, 同时降低噪音。

由于没有专门针对 Android 应用程序的测试集, 为了测试 FlowDroid 在数据流提取方面的准确性和高效性, Arzt, S.等人[7]开发了 DroidBench。该测试集包含了 39 个 Android 应用程序, 可用于 Android 静态数据流分析的评估。结果表明, FlowDroid 能够检测到 93%的隐私泄露, 检测的精确度高达 86%, 在检测性能上优于商业性检测工具 AppScan Source 和 Fortify SCA[7]。主要表现在 AppScan 和 Fortify 为了降低误报率, 牺牲了召回率, 因此会漏报一些实际存在的导致隐私泄露的数据流; 而 FlowDroid 的召回率比上述两个工具有显著提高, 在数据流提取精度上也有小幅优势。本文在实际环境中应用 InsecureBank 进一步对 FlowDroid 的数据流提取精度进行评估。实验在一台内存为 4GB, CPU 为 Intel Core 2 Centrino 的笔记本电脑上完成。InsecureBank 是 Paladion Inc.制作的用于评估分析工具的 Android 应用程序, 它包含了多种与实际应用程序类似的漏洞和数据泄露[33]。在学术界和工业界, 大量安全学者和开发者通过 InsecureBank 测试分析工具, 并学习 Android 应用渗透性测试的相关安全知识[7,34,35]。实验结果表明, FlowDroid 能够提取 InsecureBank 中所有的隐私泄露数据流, 既没有误报, 也没有漏报。因此, 我们可以认为, FlowDroid 能够准确、全面地提取 Android 应用程度的数据流, 从而为本文的数据流深度学习算法提供了大量可训练的数据流特征。

4.2 数据流特征处理技术

FlowDroid 提取的原始数据流特征包含完整的 *source* 和 *sink* 函数名。如果用这些函数之间的数据流映射关系作为特征数据, 特征矩阵可以表示成:

$$\begin{array}{c} \\ src\_m_1 \\ src\_m_2 \\ \vdots \\ src\_m_m \end{array} \begin{array}{c} \begin{array}{cccc} sink\_m_1 & sink\_m_2 & \cdots & sink\_m_n \end{array} \\ \begin{bmatrix} 1/0 & 1/0 & \cdots & 1/0 \\ 1/0 & 1/0 & \cdots & 1/0 \\ \vdots & \vdots & \ddots & \vdots \\ 1/0 & 1/0 & \cdots & 1/0 \end{bmatrix} \end{array}$$

其中, *src_m* 表示 *source* 函数, *sink_m* 表示 *sink* 函数, *m* 和 *n* 分别表示 Android 函数库中 *source* 和 *sink* 函数的个数。由于 Android 函数库中存在成千上万的 *source* 和 *sink* 函数, 因此 *m* 和 *n* 非常大。当一个 *source* 函数 src_m_i 和一个 *sink* 函数 $sink\_m_j$ 之间存在数据流映射关系时, 在特征矩阵中对应的 src_m_i 行, $sink\_m_j$ 列的值为 1, 否则,该值为 0。以恶意应用 *Android.Dendoroid.B* 为例[36], 该恶意应用获取设备的屏幕信息, 联系人, 通话记录, 短信, 设备 ID 和账户 ID 等敏感信息, 并泄露至远程服务器, 其特征矩阵如下所示。

	sendDtmf(char)	...	*URL.openConnection()*	...	*onPrepared(MediaPlayer)*
getDeviceId()	0	...	1	...	0
getSubscriberId()	0	...	1	...	0
getVoiceMailNumber()	0	...	0	...	0
⋮	⋮	⋱	⋮	⋱	⋮
View.getDrawingCache()	0	...	1	...	0
getContentResolver	0	...	1	...	0
openFileInput()	0	...	1	...	0
⋮	⋮	⋱	⋮	⋱	⋮
getContentDisposition()	0	...	0	...	0

可以看出, 如果以 *source* 函数和 *sink* 函数之间的映射关系作为特征数据, 特征矩阵中大部分的值都是 0。本文认为需要对这样的稀疏矩阵进行处理以改进训练结果。

本文利用 Rasthofer, S.等人[13]提出的 SUSI 技术对 FlowDroid 提取的数据流特征进行处理。SUSI 技术基于机器学习算法, 能够根据函数代码判断其属于 *source* 函数或者 *sink* 函数。同时, SUSI 技术将当前存在的 *source* 函数和 *sink* 函数分别划分为 17 个 *source* 类和 19 个 *sink* 类, 如表 1 所示。不难发现, 通过 SUSI 分类可以更直观地判断某恶意应用泄露的敏感数据种类, 以及敏感数据泄露的路径。

表 1 SUSI 提出的 *source* 类和 *sink* 类

Table 1 Proposed source class and sink class by SUSI

	Source 类	*Sink* 类
1	UNIQUE_IDENTIFIER	LOCATION_INFORMATION
2	LOCATION_INFORMATION	PHONE_CONNECTION
3	NETWORK_INFORMATION	VOIP
4	ACCOUNT_INFORMATION	PHONE_STATE
5	FILE_INFORMATION	EMAIL
6	BLUETOOTH_INFORMATION	BLUETOOTH
7	DATABASE_INFORMATION	ACCOUNT_SETTING
8	EMAIL	AUDIO
9	SYNCHRONIZATION_DATA	SYNCHRONIZATION_DATA
10	SMS_MMS	NETWORK
11	CONTACT_INFORMATION	FILE
12	CALENDAR_INFORMATION	LOG
13	SYSTEM_INFORMATION	SMS_MMS
14	IMAGE	CONTACT_INFORMATION
15	BROWSER_INFORMATION	CALENDAR_INFORMATION
16	NFC	SYSTEM_SETTING
17	NO_CATEGORY	NFC
18	----------	BROWSER_INFORMATION
19	----------	NO_CATEGORY

本文结合 FlowDroid 和 SUSI 两种技术, 从每个应用程序中提取 323 个数据流特征, 特征矩阵可以表示为:

$$\begin{array}{c} \\ src\_c_1 \\ src\_c_2 \\ \vdots \\ src\_c_{17} \end{array} \begin{array}{c} \begin{array}{cccc} sink\_c_1 & sink\_c_2 & \cdots & sink\_c_{19} \end{array} \\ \begin{bmatrix} 1/0 & 1/0 & \cdots & 1/0 \\ 1/0 & 1/0 & \cdots & 1/0 \\ \vdots & \vdots & \ddots & \vdots \\ 1/0 & 1/0 & \cdots & 1/0 \end{bmatrix} \end{array}$$

当一个 *source* 类 src_c_i 和一个 *sink* 类 $sink\_c_j$ 之间存在数据流映射关系时, 在特征矩阵中对应的 src_c_i 行, $sink\_c_j$ 列的值为 1,否则, 该值为 0。基于此, 恶意应用 *Android.Dendoroid.B* 的数据流特征矩阵可以表示为:

$$\begin{array}{c} \\ \text{UNIQUE\_IDENTIFIER} \\ \text{LOCATION} \\ \vdots \\ \text{FILE} \\ \text{BLUETOOTH} \\ \text{DATABASE} \\ \vdots \\ \text{IMAGE} \\ \vdots \\ \text{NO\_CATEGORY} \end{array} \begin{array}{c} \begin{array}{ccccc} \text{LOCATION} & \cdots & \text{NETWORK} & \cdots & \text{NO\_CATEGORY} \end{array} \\ \begin{bmatrix} 0 & \cdots & 1 & \cdots & 0 \\ 0 & \cdots & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & \cdots & 0 \\ 0 & \cdots & 1 & \cdots & 0 \\ 0 & \cdots & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 \end{bmatrix} \end{array}$$

4.3 数据流深度学习算法 DDBN

在提取应用程序的数据流特征并对其进行处理后, 本文利用机器学习算法训练数据流特征, 以构建检测模型实现对待测应用程序安全性的自动判别。

传统的机器学习算法, 如 Naïve Bayes, SVM, MLP 等, 通常具有少于三层的计算单元。深度学习算法, 顾名思义, 具有超过三个隐藏层的深层次架构。它旨在创建输入数据的抽象层次表示, 从而为传统的机器学习算法提供高效的训练数据[37]。目前应用广泛的深度学习算法包括自动编码器、深度信念网络、卷积神经网络以及长短期记忆网络等。其中, 卷积神经网络由于其局部感知的特性, 被广泛应用于图像处理和语音识别领域; 长短期记忆网络属于递归神经网络, 具有时序特征, 因此在自然语言处理领域取得了较好的效果。而自动编码器和深度信念网络关注的是对输入进行重构以寻找新的输入表达。其中, 自动编码器强调对输入本身的重构, 而深度信念网络强调对输入分布的重构。在以往的研究中, 深度信念网络由于理论发展成熟、训练过程相对简单, 被广泛应用于安全领域[10, 12]。除此之外, 与其他深度学习算法相比, 深度信念网络对一维特征向量或特征矩阵的分类能力更强且训练速度更快[38]。而在恶意代码检测领域, 从恶意应用样本中提取的特征往往用特征向量或特征矩阵的形式表达。因此, 本文基于深度信念网络 DBN (Deep Belief Network), 设计了数据流深度学习算法 DDBN (Data-flow Deep Belief Network), 学习恶意应用和安全应用内的数据流特征, 并以此构建基于数据流的 Android 恶意应用检测系统, 实现对 Android 未知恶意应用的自动检测。

算法设计思想为: Android 应用程序内的数据流特征在一定程度上反映了该应用程序的行为模式, 通过学习这些特征可以有效检测未知恶意应用。以往研究者对大量恶意应用内的数据流进行分析, 得出恶意应用内普遍都存在的数据流映射关系。但是, 某个恶意应用程序内的数据流特征一方面与其他恶意应用程序类似, 另一方面显著不同于安全应用程序。本算法的思想就是通过深度学习这些数据流特征之间存在的差异性和相似性, 检测未知恶意应用。

4.3.1 相关概念

定义 1. 限制玻尔兹曼机[39](Restricted Bolzmann Machine, RBM)。限制玻尔兹曼机由 Smolensky 于 1986 年在玻尔兹曼机(Bolzmann Machine, BM)的基础上提出, 是一种可用随机神经网络解释的概率图模型。其中, “随机”是指这种网络中的神经元是随机神经元, 其输出只有两种状态(未激活、激活), 一般用二进制的 0 和 1 表示, 而状态的具体取值根据概率统计法则来决定。

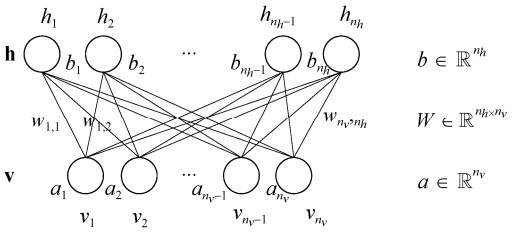


图 5 RBM 网络结构图

Figure 5 Network structure of RBM

如图 5 所示, RBM 的网络结构是一个二部图, 其中一层是可视层(visible layer, 简称 **v**), 即输入数据层, 用来描述输入数据的特征; 一层是隐藏层(hidden layer, 简称 **h**), 即特征提取层, 用来获取可视层单元对应变量之间的依赖关系。可视层与隐藏层满足每一层的节点之间没有连接, 但是层间存在全连接。

图中相关变量定义如下:

- n_v, n_h: 分别表示可见层和隐藏层的神经元的数目。
- $\boldsymbol{v} = (v_1, v_2, \mathrm{L}\ , v_{n_v})^T$: 可见层的状态向量, $\boldsymbol{v}_i$ 表示可见层第 i 个神经元的状态向量。
- $\boldsymbol{h} = (h_1, h_2, \mathrm{L}\ , h_{n_h})^T$: 隐藏层的状态向量, $\boldsymbol{h}_j$ 表示隐藏层第 j 个神经元的状态向量。
- $\boldsymbol{a} = (a_1, a_2, \mathrm{L}\ , a_{n_v})^T$: 可见层的偏置向量, $\boldsymbol{a}_i$ 表示可见层第 i 个神经元的偏置向量。
- $\boldsymbol{b} = (b_1, b_2, \mathrm{L}\ , b_{n_h})^T$: 隐藏层的偏置向量, $\boldsymbol{b}_j$ 表示隐藏层第 j 个神经元的偏置向量。
- $\boldsymbol{W} = (w_{i,j})$: 可见层与隐藏层之间的权值矩阵, $w_{i,j}$ 表示可见层第 i 个神经元与隐藏层第 j 个神经元之间的连接权重。
- $\theta = (W, a, b)$: RBM 中的参数集。

RBM 满足如下性质:

性质 1. 当给定可见层神经元的状态时, 各隐藏层神经元的激活条件独立; 反之, 当给定隐藏层神经元的状态时, 各可见层神经元的激活条件独立。

性质 2. 一个 RBM 中, **v** 和 **h** 的全概率分布 p(**v**, **h**) 满足 Boltzmann 分布。

定义 2. 反向传播算法[40](Back Propagation, BP)。反向传播指的是在训练神经网络的时候, 根据最终输出结果的误差来调整倒数第二层, 倒数第三层直至第一层的参数的过程。

首先, 定义如下变量:

- $(x^{(i)}, y^{(i)})$: 表示训练样本集。
- n^l: 表示神经网络的层数。
- $h_{W,b}(x)$: 表示神经网络模型, 其中 W 和 b 为拟合样本数据的参数。
- $J(W, b; x, y) = \frac{1}{2}\left\| h_{W,b}(x) - y \right\|^2$: 表示单个样本 (x, y) 的代价函数。

反向传播算法的基本思路可以描述为: 对于给定的样本 (x, y), 首先进行“前向传导”运算, 计算出网络中所有的激活值, 包括 $h_{W,b}(x)$ 的输出值。之后, 针对第 l 层的每一个节点 i, 计算其残差 $\delta_i^{(l)}$, 该残差表明了该节点对最终输出值的残差产生的影响。对最终的输出节点, 可以直接计算出网络产生的激活值与实际值之间的差距 $\delta_i^{(n_l)}$。

用 $\bullet$ 表示向量乘积运算符, 例如 $a = b \bullet c$ 表示 $a_i = b_i c_i$。同时, 扩展 $f(\bullet)$ 和 $f'(\bullet)$ 的定义, 使其包含向量运算, 例如: $f'([z_1, z_2, z_3]) = [f'(z_1), f'(z_2), f'(z_3)]$。反向传播算法用矩阵-向量表示法可以描述为以下步骤。

步骤 1. 进行前馈传导计算, 利用前向传导公式, 得到 $l_2, l_3, \mathrm{L}\ , l_{n_l}$ 的激活值;

步骤 2. 对输出层计算:

$$\delta^{(n_l)} = \frac{\partial}{\partial z^{n_l}} J(W, b; x, y) = \frac{\partial}{\partial z^{n_l}} \frac{1}{2}\left\| h_{W,b}(x) - y \right\|^2$$

$$= \frac{\partial}{\partial z^{n_l}} \frac{1}{2}\left\| a^{n_l} - y \right\|^2 = \frac{\partial}{\partial z^{n_l}} \frac{1}{2}\left\| f(z^{n_l}) - y \right\|^2$$

$$= -(y - f(z^{n_l})) \mathrm{g} f'(z^{n_l}) = -(y - a^{n_l}) \mathrm{g} f'(z^{n_l})$$

步骤 3. $l = n_l - 1, n_l - 2, \mathrm{L}\ , 2$ 的各层, 计算: $\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \mathrm{g} f'(z^{(l)})$

步骤 4. 计算最终需要的偏导数值:

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T$$

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}$$

注意到步骤 2 和步骤 3 中都需要计算 $f'(z^{(l)})$。假设 $f(z)$ 是 sigmoid 函数, 即 $f(z) = 1/(1 + \exp(-z))$,

可以得到其导数是 $f'(z)=f(z)(1-f(z))$。在前向反馈中可以得到 $a^{(l)}$ 的值, 可以计算得到 $f'(z^{(l)})=a^{(l)}(1-a^{(l)})$。

定义 3. 深度信念网络[41](DBN)。使用层叠波尔兹曼机组成深度神经网络的方法, 被称作深度信念网络 DBN。经典的 DBN 网络结构由若干层 RBM 和一层 BP 组成, 如图 6 所示。

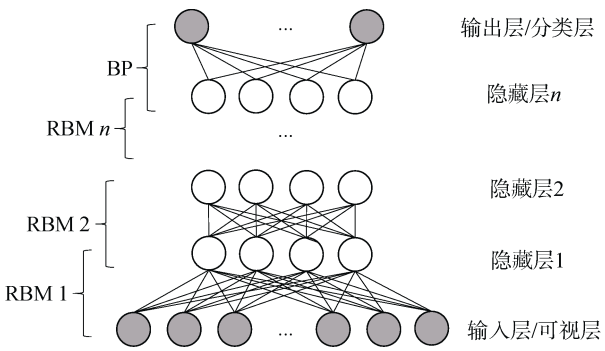


图 6 DBN 网络结构图

Figure 6 Network structure of DBN

4.3.2 DDBN 算法

图 7 为本文设计的数据流深度学习算法 DDBN 的模型结构图与训练过程图。其具体思想为: 首先将从训练应用样本中提取的数据流数据输入底层的 DRBM (Data-flow Restricted Bolzmann Machine), 底层的 DRBM 提取输入数据的简单特征, 并作为上一层的输入。如此自下而上、无监督地训练每一层 DRBM; 然后, 在最后一层设置分类器, 接收其上一层 DRBM 的输出, 并利用相同的应用样本数据有监督地进行训练。这里的 DRBM 是以数据流特征作为输入的限制玻尔兹曼机, 每个可见层单元和隐藏层单元的值代表对应的数据流特征。

因为每一层 DRBM 的训练过程是相互独立的, 所以对每一层 DRBM 的训练只能得到这一层的隐藏层对可见层最好的表达。但是, 整个 DDBN 模型不是对输入数据流特征数据的最好表达。因此, 在 DDBN 模型的最后一层设置有监督学习网络, 本文通过对比实验, 最终选取了 SVM 作为分类层算法。这样可以将整个 DDBN 模型看作一个多层的 BP 神经网络, 自顶向下地进行微调, 这个过程可以看作对一个深层 BP 网络权值参数的初始化。DDBN 的这种训练过程有效地改善了采用随机初始化权重参数时导致的可能出现的局部最优和训练时间长的问题[42]。

在 DDBN 算法中, DRBM 层数、每层的 DRBM 单元数、DRBM 重构迭代次数、SVM 微调迭代次数以及 DDBN 微调迭代次数等参数的选取直接关系到检测算法的漏报率和误报率。通过反复实验, 针对不同的参数组合评估其检测率, 选取最优的参数组合, 保持较高的检测率和较低的漏报率、误报率。

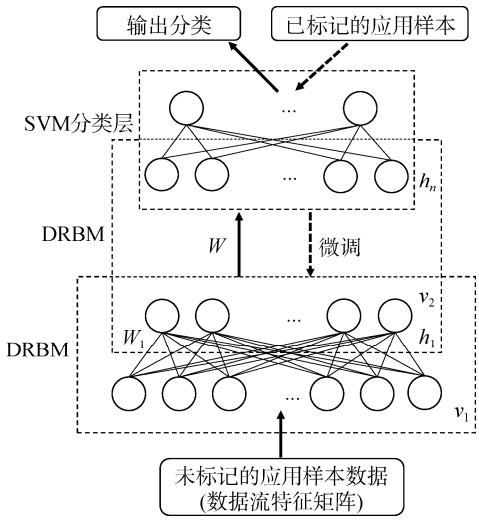


图 7 DDBN 模型结构及训练过程图

Figure 7 Model structure and training picture of DDBN

DDBN 算法的伪代码描述如下。

算法 1. DDBN.

输入:

n: 安全和恶意应用样本数

(x_i, y_i): 安全和恶意应用样本的数据流特征集合, $i=1,2,\mathrm{L}\ ,n$

n_R: DRBM 的层数

R_j: 第 j 层 DRBM 的单元数, $j=1,2,\mathrm{L}\ ,n_R$

ε: 学习率

T: 最大训练周期

输出:

D_dbn: 生成的分类器

方法:

//建立 DDBN 网络

1) $D\_initial = ddbnsetup(n_R, R_j)$;

//训练 DDBN 网络

2) $D\_train = ddbntrain(D\_initial)$;

//利用 D_train 的参数初始化 D_dbn, 然后微调

3) $D\_dbn = dbnunfoldtonn(D\_train)$;

过程 1: $ddbntrain(D\_initial)$

4) FOR $k=1,2,\mathrm{L}\ ,n_R$

//分别训练每层 DRBM

5) $drbmtrain(D\_initial.rbm(k))$;

过程 2: $drbmtrain(D\_initial.rbm(k))$;

6) 初始化: 令第 k 层 DRBM 的可视层单元的初始状态 $\mathbf{v}_k=\mathbf{x}_i$, 其中 $\mathbf{x}_i$ 为某个训练样本; $\boldsymbol{W}$、$\boldsymbol{a}$ 和 $\boldsymbol{b}$ 为随机的较小数值。

7) FOR $t=1,2,\mathrm{L}\ ,T$

8) FOR $q=1,2,\mathrm{L}\ ,R_k$ (对 k 层所有隐单元)

9) $P(\mathbf{h}_{kq}=1|\mathbf{v}_k)=\sigma(b_q+\sum_p v_{kp}W_{pq})$

10) 从条件分布 $P(\mathbf{h}_{kq}|\mathbf{v}_k)$ 中抽取 $\mathbf{h}_{kq}\in\{0,1\}$

11) ENDFOR

12) FOR $p=1,2,\mathrm{L}\ ,R_{k-1}$ (对 k 层所有可见单元)

13) $P(\mathbf{v}_{(k+1)p}=1|\mathbf{h}_k)=\sigma(a_p+\sum_q W_{pq}h_{kq})$

14) 从条件分布 $P(\mathbf{v}_{(k+1)p}|\mathbf{h}_k)$ 中抽取 $\mathbf{v}_{(k+1)p}\in\{0,1\}$

15) ENDFOR

16) FOR $q=1,2,\mathrm{L}\ ,R_k$ (对 k 层所有隐单元)

17) $P(\mathbf{h}_{(k+1)q}=1|\mathbf{v}_{k+1})=\sigma(b_q+\sum_p v_{(k+1)p}W_{pq})$

18) ENDFOR

//按下式更新各个参数

19) $$W\leftarrow W+\varepsilon(P(\mathbf{h}_k=1|\mathbf{v}_k)\mathbf{v}_k^T-P(\mathbf{h}_{k+1}=1|\mathbf{v}_{k+1})\mathbf{v}_{k+1}^T;$$

20) $\mathbf{a}\leftarrow\mathbf{a}+\varepsilon(\mathbf{v}_k-\mathbf{v}_{k+1})$;

21) $\mathbf{b}\leftarrow\mathbf{b}+\varepsilon(P(\mathbf{h}_k=1|\mathbf{v}_k)-P(\mathbf{h}_{k+1}=1|\mathbf{v}_{k+1}))$;

过程 3: $dbnunfoldtonn(D\_train)$

//利用 D_train 初始化一个大的 NN

22) $NN=nnsetup(D\_train)$;

//对 NN 进行微调生成最终的分类器 D_dbn

23) $D\_dbn=nntune(NN)$;

过程 4: $nnsetup(D\_train)$

24) FOR $k=1,2,\mathrm{L}\ ,n_R$

//训练好的 D_train 中第 k 层 $DRBM$ 的权重用于初始 化 NN 的第 k 层权重

25) $NN\{k\}.W=D\_train.drbm\{k\}.W$

26) ENDFOR

//第 n_R+1 层为分类层, 选择 SVM 作为分类器

27) FOR $i=1,2,\mathrm{L}\ ,n$

28) $NN\{n_R+1\}=trainsvm(\mathbf{x}_i,\mathbf{y}_i)$

29) ENDFOR

5 实验设计与结果分析

本文设计了多组实验验证数据流深度学习算法 DDBN 的正确性和先进性, 从而评估 Flowdect 系统在检测 Android 恶意应用方面的有效性。下文分别介绍了实验环境、实验样本数据以及评估参数, 并对实验结果进行分析。

5.1 实验环境

实验在 RBM 为 24GB, CPU 为 Intel(R) Xeon(R) E7-4809 v2(15M Cache, 2G Hz)的服务器集群上完成。通过搭建基于 Spark/Hadoop 的大数据分析平台, 并在其上集成 Yahoo 开源的大规模分布式深度学习框架 CaffeOnSpark[43], 完成对应用样本特征数据的存储、训练和分析。其中, 应用程序数据流特征的提取和处理部分主要由 Java 语言实现, 通过结合使用分析工具 FlowDroid 和 SUSI, 实现对大规模应用程序数据流的自动提取。

5.2 实验样本

实验样本包括 50000 个来自 Google Play 和第三方应用商店的安全应用程序以及 5000 个来自 Android Malware Gnome Project, Drebin 和 VirusShare 等恶意应用开源库的恶意应用程序。由于安全应用样本数远大于恶意应用样本数, 本文对安全应用集采用不充分抽样的方法, 取 5000 个安全应用样本的一个随机抽样, 与所有的恶意应用样本一起形成训练集和测试集。

这些应用程序被划分为两个应用程序集:

- 标记的应用程序集: 包括 6000 个应用程序, 其中 3000 个应用程序是恶意的, 其他是安全的。本文采用三重交叉验证技术将应用程序集划分为以下部分, 包括 2000 个恶意的训练样本、2000 个安全的训练样本、1000 个恶意的测试样本和 1000 个安全的测试样本。
- 未标记的应用程序集: 包括 4000 个应用程序, 其中 2000 个应用程序是恶意的, 其他是安全的。

另外, 为了验证 Flowdect 检测未知恶意应用的有效性, 本文使用了某安全厂商提供的最新恶意应用, 以及实验室自主研发的恶意应用共计 90 个(不属于任何开源恶意应用库), 这些恶意应用的详细分类及对应数量如表 2 所示。我们采用的分类依据是动态行为提取工具 CopperDroid 针对其提取的行为特征进行的分类[26]。

表 2 90 个最新恶意应用样本分类

Table 2 90 latest malicious application sample classification

样本行为特征分类	样本数量
S_1: 文件访问	13
S_2: 特权提升	18
S_3: 网络	14
S_4: 获取隐私信息	25
S_5: 发送短信	11
S_6: 接听/拨打电话	9

5.3 评估参数

本文采用以下的评估参数衡量系统 Flowdect 的性能:

定义 4. 混淆矩阵(Confusion Matrix)。混淆矩阵是一种特定的矩阵, 用来呈现算法性能的可视化效果, 通常用于监督学习。其每一列代表实际的类别, 每一行代表预测的类别。表 3 为本文在评估实验中采用的混淆矩阵。

表 3 混淆矩阵

Table 3 Confusion matrix

	恶意应用	安全应用
恶意应用	TP	FP
安全应用	FN	TN

其中 True Positives(TP)表示被正确识别为恶意应用的恶意应用数; True Negatives(TN)表示被正确识别为安全应用的安全应用数; False Positives(FP)表示被错误识别为恶意应用的安全应用数; False Negatives(FN)表示被错误识别为安全应用的恶意应用数。由混淆矩阵可以计算得到以下的分类器评估参数:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\mathrm{Re}call = TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

$$\mathrm{Pr}ecision = \frac{TP}{TP + FP}$$

$$F\text{-}score = \frac{2 \times \mathrm{Re}call \times \mathrm{Pr}ecision}{\mathrm{Re}call + \mathrm{Pr}ecision}$$

5.4 实验结果

5.4.1 DDBN 模型参数

在对 DDBN 模型进行训练时, 需要设置模型参数, 包括模型层数, 每层的神经元数以及迭代次数等。根据以往研究者在训练深度学习模型时相关参数的选取规则, 并且在实验过程中通过对比训练误差和测试误差不断调节迭代次数, 本文最终将 DRBM 预训练迭代次数固定为 50 次, DDBN 无监督微调迭代次数为 200 次, BP 微调迭代次数为 500 次。研究[10]表明, 当模型层数为 4 时, 往往能取得更优的检测效果。基于此, 本文分别设置不同的模型层数, 以及每层的神经元数, 并分别训练对应的 DDBN 模型, 部分检测结果如表 3 所示。

由表 4 可知, 当设置 DDBN 模型层数为 4, 并且每层的神经元数分别为 200, 150, 50 和 20 时, 分类精度最优, 最高的 *F-score* 接近 95.05%。

表 4 不同 DDBN 模型结构检测效果比较

Table 4 Detection effect comparison for different DDBN model structures

层 1	层 2	层 3	层 4	层 5	安全应用		恶意应用		*F-score* (%)
					Pr*ecision* (%)	Re*call* (%)	Pr*ecision* (%)	Re*call* (%)	
200	**150**	**50**	**20**	—	**97.22**	**94.26**	**92.69**	**96.13**	**95.05**
200	100	50	20	—	95.14	91.82	92.12	96.32	93.81
200	100	100	20	—	95.48	89.98	90.51	95.72	92.85
200	150	100	50	—	94.74	94.20	94.59	94.91	94.61
100	100	50	20	—	90.66	88.18	89.81	91.80	90.10
200	150	50	—	—	88.74	87.04	87.13	89.84	88.17
200	150	50	20	10	89.18	88.35	89.32	90.56	89.35

表 5 DDBN 算法与传统机器学习算法和 DBN 算法检测效果对比

Table 5 Comparisons of DDBN and traditional machine learning algorithms

		Naïve Bayes	PART	Logistic Regression	MLP	SVM	DBN	DDBN
安全应用	Pr*ecision* (%)	90.73	88.26	96.89	91.95	93.78	96.69	**97.22**
	Re*call* (%)	75.05	73.3	79.02	52.06	90.54	92.57	**94.26**
恶意应用	Pr*ecision* (%)	78.65	75.12	82.57	68.63	92.11	92.52	**92.69**
	Re*call* (%)	91.36	89.79	96.91	93.87	94.02	95.78	**96.13**
F-score (%)		83.34	80.94	72.89	88.11	92.59	94.35	**95.05**

5.4.2 数据流特征

此外, 本文对恶意应用和安全应用的数据流特征展开了深入分析。我们总结了在恶意应用和安全应用中分别排名较高的数据流特征, 如图 8 所示。我们发现, 在不考虑 *source* 或 *sink* 属于 NO_CATEGORY 的情况下, 在恶意应用和安全应用中均排名较高的 *source* 类有 UNIQUE_IDENTIFIER 和 NETWORK_INFORMATION, *sink* 类主要有 LOG。但是存在部分在恶意应用中频繁出现, 而在安全应用中却极少见的数据流特征。例如: LOCATION_INFORMATION 是恶意应用中最常见的 *source* 类之一。除此之外, SMS_MMS, FILE 和 NETWORK_INFORMATION 频繁被恶意应用作为敏感数据泄露的方式, 这表明大部分恶意应用会获取用户的位置信息并使用 SMS, 文件或网络作为泄露用户隐私数据的手段。

	Malware		Benignware
1.	NO_CATEGORY -> LOG	1.	UNIQUE_IDENTIFIER -> NO_CATEGORY
2.	UNIQUE_IDENTIFIER -> SMS_MMS	2.	NO_CATEGORY -> NO_CATEGORY
3.	NO_CATEGORY -> NO_CATEGORY	3.	NO_CATEGORY -> LOG
4.	UNIQUE_IDENTIFIER -> NO_CATEGORY	4.	UNIQUE_IDENTIFIER -> LOG
5.	LOCATION_INFORMATION -> LOG	5.	NETWORK_INFORMATION -> LOG
6.	NO_CATEGORY -> NETWORK_INFORMATION	6.	CALENDAR_INFORMATION -> LOG
7.	NETWORK_INFORMATION -> LOG	7.	BROWSER_INFORMATION -> NO_CATEGORY
8.	UNIQUE_IDENTIFIER -> FILE	8.	NETWORK_INFORMATION -> NO_CATEGORY
		9.	FILE_INFORMATION -> NO_CATEGORY

图 8 恶意应用及安全应用中出现频率高的数据流

Figure 8 Frequently occurred data flow in malware and benignware

其后, 本文展开实验挖掘在恶意应用和安全应用中出现频率差异最大的十种数据流特征, 结果如图 9 所示。从图 9 我们可以看出, 数据流如“UNIQUE_IDENTIFIER → SMS_MMS”, “LOCATION_INFORMATION → LOG”和“NETWORK_INFORMATION→LOG”是三种在恶意应用和安全应用中存在最大出现频率差的数据流特征。其他的数据流, 如: “LOCATION_INFORMATION → NETWORK”, “DATABASE_INFORMATION → SMS_MMS”, “SMS_MMS → SMS_MMS”, “DATABASE_INFORMATION → NETWORK_INFORMATION”, “EMAIL → NETWORK_INFORMATION”和“CONTACT_ INFORMATION →SMS_MMS”, 尽管在恶意应用和安全应用中的出现频率均没有排名前十, 但它们通常只出现在恶意应用中。我们可以总结得到, 恶意应用往往通过窃取用户的敏感数据(如: 位置, 联系人, SMS, 呼叫记录等)并发送 SMS 或网络数据包实现泄露。此外, 我们发现数据流“UNIQUE_IDENTIFIER→LOG”在安全应用中的出现频率高于恶意应用。这意味着安全应用总是将诸如 IMEI, IMIS, 电话号码等信息视为用户设备标识符, 但不会让这些信息离开设备。相反地, 恶意应用通常利用这些标识符来执行隐私攻击。

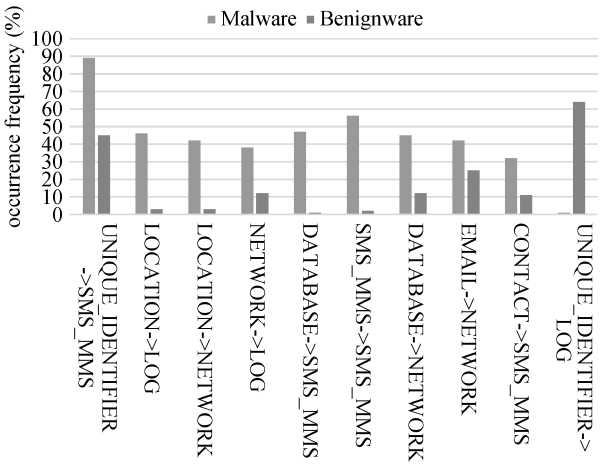


图 9 恶意应用及安全应用中出现频率差异前十的数据流

Figure 9 Top 10 most different data flow in malware and benignware

5.4.3 DDBN 算法评估

为了评估本文提出的 DDBN 算法的优势, 在本节中, 我们比较了 DDBN 算法与基于传统机器学习模型的检测方法的分类精度, 结果如表 5 所示。在实验中我们选择了 Naïve Bayes, PART, Logistic Regression, SVM 和 MLP 作为对比算法。对于 Naïve Bayes 算法, 先验概率 P(*C*=恶意应用)=P(*C*=安全应用)=0.5, 假设类条件概率满足伯努利分布, 对于其可能出现的因为某个属性的类条件概率为 0 导致整个类的后验概率为 0 的情况, 我们使用 *m* 估计方法来估计条件概率, 其中 *m* 最优值为 5000。对于决策树算法 PART, 选取“熵”作为不纯性度量, 从而在每次构建分支时选取最佳划分, 最优决策树的树深为 78。对于 SVM 算法, 我们分别测试了线性内核、多项式内核、径向量内核和 sigmoid 核等核函数, 当选取径向量内核作为核函数时能取得最优检测效果。对于 MLP 算法, 输入层神经元个数为 136, 每个权重的初始值为 0.01, 偏置因子 t=0.08, 学习率 λ 是自适应的。通过训练, 最终根据各特征的重要性, 其权值范围在 0.003~0.012 之间(共有 136 个数据流特征, 此处不一一列举), 最优偏置因子 t=0.092。同时, 我们比较了以 SVM 作为分类层算法的 DDBN 算法与传统意义上以 BP 作为分类层算法的 DBN 算法的分类精度。

从表 5 我们可以发现, 决策树算法 PART 的分类精度明显低于其他几种模型, 原因是用于训练的特征不够丰富。此外, 贝叶斯分类算法 Naïve Bayes, 回归算法 Logistic Regression 和人工神经网络算法 MLP 在恶意应用的召回率 Re*call* 和安全软件的准确率 Pr*ecision* 方面具有一定的优势。然而, 当使用这几种算法进行训练时, 对恶意应用检测的准确率会随着恶意应用召回率的增加而快速降低; 同样地, 对安全应用检测的召回率会随着安全应用准确率的增加而快速降低。基于核的学习算法 SVM, 深度学习算法 DBN, 以及本文提出的数据流深度学习算法 DDBN 的检测精度优于其他模型。但是 DDBN 同时具有 DBN 算法对输入的特征进行深度训练的特性, 以及 SVM 算法在平衡召回率和准确率方面的优势, 因此, DDBN 比 SVM 和 DBN 具有更高的 *F-score*, 在所有的检测模型中表现最优。

值得注意的是, 文献[10,38]运用深度学习算法 DBN 对 Android 应用程序的权限、API 调用、动态行为等特征进行深度训练, 从而构建检测模型。作者在训练 DBN 模型的分类阶段使用 BP 算法作为分类算法, 因此可以从表 4 看出, 其检测精度低于本文的 DDBN 算法。

5.4.4 检测未知恶意应用性能评估

为了评估 DDBN 算法在检测未知恶意应用方面的性能, 在本节中, 我们利用训练好的最优 DDBN 模型对某安全厂商提供的最新恶意应用, 以及实验室自主研发的恶意应用共计 90 个进行检测。需要注意的是, 这 90 个恶意应用不属于任何开源的恶意应用库。结果表明, DDBN 模型能够成功检测出其中 74 个未知的恶意应用, 也就是说, DDBN 算法对于未知恶意应用的检测率能够达到 82.2%。分析发现, 未被成功检测的 16 个恶意应用程序在特征提取阶段, 均未被提取到敏感数据流映射关系。

6 讨论

本文结合 FlowDroid 和 SUSI 技术提取 Android 应用程序的静态数据流特征, 并设计数据流深度学习算法 DDBN 构建检测模型 Flowdect。

在 DDBN 模型中, DRBM 的训练过程实际上是求出一个最能产生训练应用样本数据流特征的概率分布。也就是说, 通过训练获得一个分布, 在这个分布里, 训练应用样本数据流特征的概率最大。之后的 DDBN 微调阶段, 通过不同层次之间的不断调节, 使得生成模型可以重构出具有较低误差的原样本, 这样就可以得到应用样本的本质特征, 也就是 DDBN 模型的最高抽象表示形式。综上所述, DDBN 模型中底层的 DRBM 接收原始的数据流特征向量, 在自底向上的传递过程中, 从具体的数据流特征向量逐渐转化为抽象的数据流特征向量, 从而在顶层的神经网络形成更易于分类的组合特征向量。

DDBN 算法明确突出了特征学习的重要性, 通过逐层特征变换, 将应用程序样本的静态数据流在原空间的特征表示变换到一个新特征空间, 从而得到了更高层、更抽象的静态数据流的表示, 能够更好地描述应用程序, 也使得对恶意应用的预测更加容易[10, 48]。

但是, 基于 DDBN 模型构建的 Flowdect 系统存在一定的局限性。在本节中, 我们总结了基于机器学习算法设计 Android 平台恶意应用检测系统时的三个关键点, 并分别介绍 Flowdect 在每个关键点存在的问题和下一步工作的方向。

(1) 选取的机器学习算法。本文提出的 DDBN 算法是基于深度学习算法实现的。深度学习算法具有不同于传统机器学习算法的深层结构, 明确突出了特征学习的重要性, 通过逐层特征变换, 将样本在原空间的特征表示变换到一个新特征空间, 从而使分类或预测更加容易。本文选取深度学习模型 DBN 作为分类模型并取得了较好的检测效果。在下一步工作中, 可以选取更适用于恶意应用检测场景的深度学习模型, 从而完善系统。

(2) 选取的样本特征。应用程序的数据流特征直观地反映了该应用有关敏感数据的行为, 因此本文将应用程序的数据流作为训练特征。但是本文仅实现了对数据流特征的检测, 由于特征类型的单一性, 造成了一定的漏报率。同时, 在文献[31,46,47]中, 作者研究发现, 尽管 FlowDroid 能够准确地跟踪敏感数据流的轨迹, 以及处理对象、域和上下文敏感等因素, 但是对于代码混淆而形成的隐式信息流, 由于敏感流信息转化的多样性和复杂性, 使得 FlowDroid 无法完全从代码分析层面进行准确推断。因此, FlowDroid 针对混淆代码存在漏报的情况。

研究表明, 复杂度越大、粒度越细的特征越能覆盖安全应用程序和恶意应用程序的各方面, 从而能够更好地表征应用程序。因此在下一步工作中, 可以利用深度学习算法训练更丰富的应用程序动静态特征, 例如: 函数调用、系统调用、网络行为等, 从而提高对混淆代码的检测率。

(3) 样本数。显然, 用于训练的样本数越大, 基于机器学习算法的分类模型将取得越准确的分类效果。尤其是对于深度学习模型, 输入的训练样本数直

接决定了输出的结果准确率[10]。本文选取的应用样本分别来自 Google Play 以及 Android Malware Gnome Project 和 VirusShare 等恶意应用开源库, 然而这些应用并不足以训练一个完美的恶意应用检测模型, 获取更多数量的安全应用样本和恶意应用样本势在必行。当前, 对 Android 平台恶意应用的实时收集是一个巨大的挑战, 需要全球各地的研究者相互合作, 构建并不断完善类似于 Android Malware Gnome Project 和 VirusShare 的恶意应用公开库。

7 结论

当前, Android 平台恶意应用的迅猛增长严重威胁着用户的隐私安全。传统的检测方法基于静态签名或动态异常行为识别恶意应用。然而, 基于签名的方法难以检测不断涌现的零日恶意应用。同时, 恶意攻击技术发展迅速, 新型的恶意应用能够隐藏恶意行为, 以躲避动态检测。为了解决这些问题, 研究者提出了基于机器学习的检测方法, 通过静态和动态分析技术提取应用程序的特征, 并自动学习恶意应用与安全应用之间的差异, 以实现对恶意应用的自动检测。

本文结合 FlowDroid 和 SUSI 技术提取 Android 应用程序的静态数据流特征, 通过设计数据流深度学习算法 DDBN 构建检测模型, 并实现检测系统 Flowdect。实验结果表明, Flowdect 在检测的准确率和执行效率上表现良好, DDBN 算法显著优于传统的机器学习算法。

下一步工作将应用深度学习算法分析更丰富、更细粒度的特征来表征 Android 恶意应用程序。除了本文的静态数据流特征, 可以将其他静态特征(如权限, API 调用等)与动态特征(如可疑行为序列, 网络流量等)相结合进行分析。除此之外, 如果有更多的样本数据, Flowdect 将可以实现更高的分类精度。

参考文献

[1] "Gartner says Android has surpassed 1.6 billion shipments of devices," Gartner, http://www.gartn-er.com/newsroom/id/ 2954317, 2016.

[2] Sensor Tower, https://sensortower.com.

[3] "Threat report 2015," F-Secure, https://www.f-secure.com/documents/996508/1030743/Threat_Report_2015.pdf, 2015.

[4] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden, "Mining Apps for Abnormal Usage of Sensitive Data," *in Proc. 37th IEEE International Conference on Software Engineering* (ICSE'15), pp. 426-436, 2015.

[5] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B. G. Chun, L. P. Cox, and A. N. Sheth, "TaintDroid: an Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones," *in Proc. Usenix Symposium on Operating Systems Design and Implementation* (OSDI'10), pp.393-407, 2010.

[6] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These Aren't the Droids you're Looking For: Retrofitting Android to Protect Data from Imperious Applications," *in Proc. 18th ACM Conference on Computer and Communications Security* (CCS'11), pp. 639-652, 2011.

[7] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, and P. McDaniel, "Flowdroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps," *ACM Sigplan Notices*, vol. 49, no. 6, pp. 259-269, 2014.

[8] F. Wei, S. Roy, and X. Ou, "Amandroid: A Precise and General Inter-Component Data Flow Analysis Framework for Security Vetting of Android Apps," *in Proc. ACM Conference on Computer and Communications Security* (CCS'14), pp. 1329-1341, 2014.

[9] Z. Yang, and M. Yang, "Leakminer: Detect Information Leakage on Android with Static Taint Analysis," *in Proc. World Congress on Software Engineering* (WCSE'12), pp. 101-104, 2012.

[10] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: Android Malware Characterization and Detection Using Deep Learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114-123, 2016.

[11] Deep Learning, https://en.wikipedia.org/wiki/Deep_learning.

[12] Deep Instinct, www.deepinstinct.com.

[13] S. Rasthofer, S. Arzt, and E. Bodden, "A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks," *in Proc. Network and Distributed System Security Symposium* (NDSS'14), 2014.

[14] VirusShare, http://virusshare.com/.

[15] Y. Zhou, and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," *in Proc. IEEE Symposium on Security and Privacy* (S&P'12), pp. 95-109, 2012.

[16] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "SCAndroid: Automated security certification of android applications," Department of Computer Science, University of Maryland, College Park, Technical Report CS-TR-4991, November 2009.

[17] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in Proc. 16th ACM Conf. Comput. *Commun. Security*, 2009, pp. 235–245.

[18] T. Debiaze, "Detecting malicious behavior for android applications by static analysis," https://github.com/maaaaz/androwarn, 2015.

[19] V. Afonso, A. Bianchi, Y. Fratantonio, A. Doupé, M. Polino, and P. D. Geus, "Going Native: Using a Large-Scale Analysis of Android Apps to Create a Practical Native-Code Sandboxing Policy," *in Symposium on Network and Distributed System Security* (NDSS'16), 2016.

[20] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "DroidSieve: Fast and Accurate Classification of Obfuscated Android Malware," *in ACM on Conference on Data and Application Security and Privacy*, pp. 309-320, 2017.

[21] J. Seo, D. Kim, D. Cho, T. Kim, I. Shin, and J. Seo, "FLEXDROID: Enforcing In-App Privilege Separation in Android," *in Symposium on Network and Distributed System Security* (NDSS'16), 2016.

[22] A. Desnos, and P. Lantz, "Droidbox: An android application sandbox for dynamic analysis," Available: https://code.google.

com/p/droidbox/, 2013.

[23] V. Rastogi, Y. Chen, and W. Enck, "AppsPlayground: automatic security analysis of smartphone applications," *in ACM Conference on Data and Application Security and Privacy*, pp.209-220, 2013.

[24] L. K. Yan, and H. Yin, "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis," *in Proceedings of the 21st USENIX conference on Security symposium* (USENIX'13), pp.29-29, 2013.

[25] M. Y. Wong, and D. Lie, "IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware," *in Symposium on Network and Distributed System Security* (NDSS'16), 2016.

[26] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "CopperDroid: Automatic Reconstruction of Android Malware Behaviors," *in Network and Distributed System Security Symposium*, 2015.

[27] H. S. Ham, and M. J. Choi, "Analysis of Android Malware Detection Performance Using Machine Learning Classifiers," *in Proc. IEEE International Conference on ICT Convergence* (ICTC'13), pp. 490-495, 2013.

[28] M. Z. Mas'Ud, S. Sahib, M. F. Abdollah, and S. R. Selamat, "Analysis of Features Selection and Machine Learning Classifier in Android Malware Detection," *in Proc. IEEE International Conference on Information Science & Applications* (ICISA'14), pp.1-5, 2014.

[29] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of Machine Learning Classifiers for Mobile Malware Detection," *Soft Computing*, vol. 20, no. 1, pp. 1-15, 2016.

[30] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," *in Proc. Network and Distributed System Security Symposium* (NDSS'14), 2014.

[31] P. Feng, J. Ma, and C. Sun, "Selecting Critical Data Flows in Android Applications for Abnormal Behavior Detection," *Mobile Information Systems*, pp. 1-16, 2017.

[32] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: scalable and accurate zero-day android malware detection," *in Proc. International Conference on Mobile Systems, Applications, and Services* (Mobisys'12), pp.281-294, 2012.

[33] https://github.com/dineshshetty/Android-InsecureBankv2

[34] L. Li, A. Bartel, J. Klein, and Y. L. Traon, "Using a path matching algorithm to detect inter-component leaks in android apps," *Giornale Di Malattie Infettive E Parassitarie*, vol. 23, no. 5, pp. 275-8, 2014.

[35] H. Dalziel, and A. Abraham, "Automated Security Analysis of Android and iOS Applications with Mobile Security Framework," Syngress Publishing, 2015.

[36] "360 mobile security report," 360 mobile security lab, http://blogs.360.cn/360mobile/2015/04/14/analysis_of_dendoroid_b/, 2015

[37] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," Nature, vol. 521, no. 7553, pp. 436-444, 2015.

[38] Z. D. Su, Y. F. Zhu and L. Liu, "Android Malware Detection Based on Deep Learning," *Journal of Computer Applications*, vol. 37, no. 6, pp. 1650-1656, 2017.

[39] C. X. Zhang, N. N. Ji, and G. W. Wang, "Restricted Boltzmann Machine," *Chinese Journal of Engineering Mathematics* (2), pp. 159-173, 2013.

[40] E. D. Karnin, "A Simple Procedure for Pruning Back-propagation Trained Neural Networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 239-42, 1990.

[41] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring Strategies for Training Deep Neural Networks," *Journal of Machine Learning Research*, vol. 1, no. 10, pp. 1-40, 2009.

[42] G. E. Hinton, S. Osindero, and Y. W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, no. 7, pp. 1527-1554, 2006.

[43] CaffeOnSpark, https://github.com/yahoo/CaffeOnSpark/.

[44] M. D. Zeiler, and R. Fergus, "Visualizing and understanding convolutional networks," 8689, pp. 818-833, 2013.

[45] F. Idrees, M. Rajarajan, M. Conti, T. Chen, and Y. Rahulamathavan, "Pindroid: a novel android malware detection system using ensemble learning methods," *in Computers & Security*, 68, 2017.

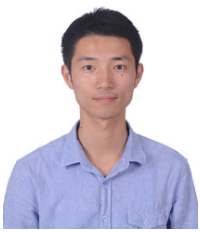
[46] C. K. Guo, J. Xu, G. N. Si and S. H. Xu, "Model Checking for Software Information Leakage in Mobile Application," *Chinese Jounal of Computers*, vol. 39, no. 11, pp. 2324-2343, 2016.

[47] J. Garcia, M. Hammad, and Malek. Sam, "Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware," *Transactions on Software Engineering and Methodology*, 2017.

[48] Z. Yuan, Y. Lu, "Droid-Sec: deep learning in android malware detection," *in Acm Sigcomm Computer Communication Review* (Sigcomm'14), pp. 371-372, 2014.

朱大立 于 2007 年在华中科技大学获得计算机应用技术专业博士学位。现任中国科学院信息工程研究所正研级高级工程师，博士生导师。研究领域移动互联网安全和无线网络攻防技术，研究兴趣包括：智能终端安全、应用安全、无线管控等技术。Email: zhudali@iie.ac.cn

金昊 于 2013 年在合肥工业大学计算机科学与技术专业获得学士学位。现在中国科学院信息工程研究所通信与信息系统专业攻读博士学位。研究领域为移动互联网安全。研究兴趣包括：移动恶意代码检测与分析等。Email: jinhao@iie.ac.cn



吴荻 于 2014 年在北京交通大学通信与信息系统专业获得博士学位。现任中科院信息工程研究所助理研究员。研究领域为移动网络、传感网等。研究兴趣包括: 物联网、隐私保护、无线资源分配等。Email: wudi@iie.ac.cn

荆鹏飞 于 2013 年在北京理工大学电子与通信工程专业获得硕士学位。现任中国科学院信息工程研究所第四研究室工程师。研究领域为移动互联网安全。研究兴趣包括: 智能手机系统安全防护、手机多维管控、手机恶意代码检测与分析等。Email: jingpengfei@iie.ac.cn

杨莹 于 2008 年在广西师范大学计算机应用技术专业获得工学硕士学位, 现在中国科学院信息工程研究所信息安全专业攻读博士学位。研究领域为信息安全。研究兴趣包括: 操作系统安全、移动智能终端安全。Email: yangying@iie.ac.cn