一种基于深度学习的强对抗性 Android 恶意代码检测方法

李鹏伟,姜宇谦,薛飞扬,黄佳佳,徐 超
(南京审计大学信息工程学院,江苏南京 211815)

摘 要: 针对现有 Android 恶意代码检测方法容易被绕过的问题,提出了一种强对抗性的 Android 恶意代码检测方法. 首先设计实现了动静态分析相结合的移动应用行为分析方法,该方法能够破除多种反分析技术的干扰,稳定可靠地提取移动应用的权限信息、防护信息和行为信息. 然后,从上述信息中提取出能够抵御模拟攻击的能力特征和行为特征,并利用一个基于长短时记忆网络(Long Short-Term Memory,LSTM)的神经网络模型实现恶意代码检测. 最后通过实验证明了本文所提出方法的可靠性和先进性.

关键词: 恶意代码; 静态分析; 动态分析; 深度学习; 长短时记忆网络
中图分类号: TP391 **文献标识码:** A **文章编号:** 0372-2112(2020)08-1502-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2020.08.007

A Robust Approach for Android Malware Detection Based on Deep Learning

LI Peng-wei,JIANG Yu-qian,XUE Fei-yang,HUANG Jia-jia,XU Chao
(*School of Information Engineering,Nanjing Audit University,Nanjing,Jiangsu* 211815,*China*)

Abstract: Conventional Android malware detection method can easily be evaded. In this study,we propose a detection method of Android malicious code based on short-term memory network(LSTM),which makes malware more difficult to evade from detection. In this method,a program analysis framework that combines static and dynamic analysis is proposed at first to get the permission information,protection information and behavior information. Secondly,entrenched features such as ability features and behavior features are extracted from the information that provided by the program analysis framework. With the entrenched features,we design a malware detection method based on LSTM model to distinguish benign applications from the malicious ones. Experimental results demonstrate that our approach is more effective and robust in Android malware detection than the state-of-the-art methods.

Key words: android malware;static analysis;dynamic analysis;deep learning;LSTM

1 引言

以智能手机为代表的智能移动设备已成为人们日常生活中不可或缺的工具. 移动互联网应用的种类和数量呈爆发式增长,越来越多地渗透到人们生活、工作的各个领域,正逐渐成为用户信息数据的主要入口和核心载体. 同时,恶意移动应用的数量也急剧增长[1],严重侵犯用户权益、影响产业发展,甚至威胁国家安全.

近年来,机器学习特别是深度学习迅速发展,已经开始被用于恶意代码检测,并取得了良好效果[2]. 但是,恶意代码检测是一个典型的攻防对抗问题,恶意代码会尝试通过各种方法逃避检测[3]. 对于基于机器学习的恶意代码检测,如果攻击者能够阻断、干扰特征提取或者直接改变自身的特征,则可能绕过检测.

恶意代码检测往往需要通过静态分析或者动态分析实现特征提取,对此,恶意代码可以通过加壳[4]、反射、动态代码加载等方法绕过静态分析[5],可以通过感知或绕过行为监控模块来逃避动态分析[6]. 现有检测

收稿日期:2019-03-03;修回日期:2020-05-15;责任编辑:孙瑶
基金项目:国家自然科学基金(No.61802194,No.61902190);江苏省高等学校自然科学研究(No.17KJB520015,No.19KJB520040);审计信息与工程协同创新中心资助项目(No.18CICA06)

方法往往会因为以上对抗机制的存在而出现漏报. 因此,本文通过脱壳重打包来提高静态分析的准确性,并设计实现了基于解释器修改的具有较强对抗性的行为监控方法,以期实现稳定可靠的行为分析.

另一方面,现有面向恶意代码检测的机器学习模型所使用的特征包含了大量字符特征和结构特征. 但是,对于恶意应用的开发者,改变程序大小、组件数量、权限数量都比较容易实现的. 例如,攻击者可以通过模拟正常应用的权限申请情况实现对恶意代码检测系统的模拟攻击(Impersonate Attack),从而绕过检测. 针对该威胁,本文试图设计一种能够剔除易变特征,利用不容易更改的特征来区分正常和恶意应用的方法. 同时,针对本文所使用的特征较为复杂多样的特点,设计实现基于深度学习的分类模型来实现恶意代码检测.

2　相关工作

2.1　Android 应用行为分析

现有行为分析方法可分为静态分析和动态分析. 静态分析方法在不实际运行程序的情况下,通过分析程序代码获得程序所有的可能执行路径. 例如,SAAF[7]、FlowDroid[8]、IccTA[9]等工具通过静态污点跟踪分析来发现隐私泄露. 动态行为分析是指实际运行分析目标,并在样本运行过程中监控系统调用、文件操作、网络访问等行为,最后根据监测到的行为信息进行进一步的分析. 例如,TaintDroid[10]通过动态污点跟踪来检测隐私泄露. 对于静态分析,恶意移动应用可以通过加壳、加密、反射、动态代码加载等方式防止或逃避分析;对于动态分析,移动应用可以通过分析环境检测、触发条件设置来逃避分析. 因此,分析者需要尽可能提高分析方法的鲁棒性和对抗能力.

2.2　基于机器学习的 Android 恶意代码检测

面对来自海量恶意代码的威胁,现有研究和应用提出了多种基于机器学习的恶意代码检测方案. 例如,Drebin[11]将组件、权限、硬件功能需求、组件关联关系、API 调用、网络地址等作为特征实现恶意代码检测;文献[12]将组件间通信载体 Intent 作为特征进行恶意代码检测;SSdroid[13]利用“后续行为特征”实现恶意行为检测;王兆国等人[14]提出了基于行为链的应用隐私窃取行为检测方法;张鹏等人[15]提出基于资源签名的 Android 应用相似性快速检测方法,为检测盗版应用提供支持;王蕊等人[16]通过动态污点分析提取特征,实现抗混淆的恶意代码变种识别. 近年来,深度学习迅速发展,越来越多的研究人员通过深度学习方法来检测恶意代码. 例如,Droid-sec[17]通过静态分析方法提取 opcode 序列作为特征,用深度卷积神经网络(Convolutional Neural Networks, CNN)实现恶意代码检测;Droiddetector[18]通过动静态结合的分析提取了 192 个特征,然后利用深度信念网络(Deep Belief Networks, DBN)实现恶意代码检测. 由于深度学习模型可以根据输入数据自行训练多层神经网络模型,每层可以提取原始数据不同特征,能够更好地表达高维复杂函数,在分析处理特征复杂多样的 Android 应用方面具有很大优势.

3　移动应用行为分析

本文通过动静态分析相结合的行为分析框架实现强对抗性的移动应用行为分析. 如图 1 所示,该框架包括脱壳重打包、静态分析、动态分析 3 个模块. 其中,脱壳重打包模块的目的是获取脱壳后的程序安装包,从而在静态分析时获得原本无法获得的隐藏信息;静态分析模块能够提取移动应用的大小、结构、权限申请、防护情况以及类名等信息;动态分析模块分为动作触发和行为监控两部分,能够获得移动应用执行时的行为序列且不易被绕过.

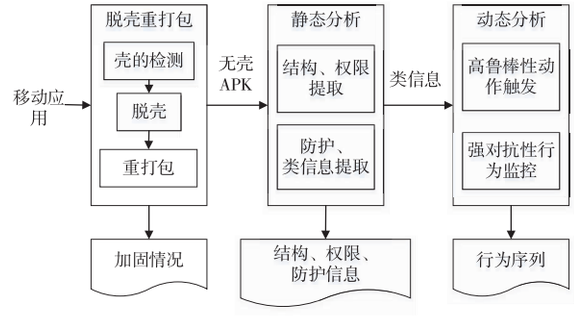

图1　移动应用行为分析框架

3.1　脱壳重打包

移动应用普遍存在的加壳现象会对静态分析造成干扰. 因此,首先分析移动应用的加壳加固情况,如果发现有壳,则进行脱壳处理,以确保后续分析的准确性.

本模块选取 Dex2oat 作为脱壳点. Dex2oat 是位于 Android 系统/system/bin 目录下的一个系统文件,其作用是将 Android 程序中的 DEX 文件编译为 oat 可执行文件. 在应用运行时,壳的主体会以 Native 调用的方式调用 Dex2oat,将应用的真实 DEX 编译为 oat 文件,再提供给系统去执行. 因此,如果可以在调用 Dex2oat 时,从内存中读取 DEX 文件,就可以实现脱壳. 基于上述原理,本文通过修改 Dex2oat 函数源码,编译定制 Android 系统的方式实现脱壳. 为了对抗某些加固厂商对于脱壳的抵抗,例如监视应用所在目录的和监控特定的系统 API,本模块还实现了根据配置文件灵活调整脱壳输出文件的存放位置以及调整所使用的系统读写 API 的功能.

完成脱壳操作后,得到了未加密的 DEX 文件. 由于后续的静态分析还需要应用包中的其他文件作为输入

信息,因此还需要将解密后的 DEX 重新打包回应用包中. 重打包的具体方法如下:首先将 DEX 文件反编译为 smali 代码,然后修改 AndroidManifest. xml,去除加固厂商插入的无效干扰信息,最后将 smali 源码文件编译回 DEX 文件,并与其他应用资源一起,打包为新的 APK 文件,为静态分析提供脱壳后样本.

3.2 静态分析

静态分析模块首先实现了常规静态分析的基本功能:提取移动应用的结构信息、权限信息. 在此基础上,增加了对加固情况、反射和 Native 代码使用情况的记录,并获取了动态分析依赖的信息,为实现完备的行为监控提供支持. 其中,加固情况的分析通过特征码匹配实现,类的数量和名称通过依次分析当前移动应用中所有 DEX 文件的字节码来得到.

静态分析模块的建立在移动应用分析框架 AndroGuard[19] 的基础上. 本文的结构信息分析、权限信息分析、防护信息分析模块通过对 Androguard 的 Androapkinfo. py、Androaxml. py 等模块进行扩展实现. 其中,反射和 Native 代码的检测通过调用 Androguard 的 Dx 分析对象实现. 获取类信息的方法如下:首先调用 Androguard 的 API 将样本的 DEX 文件加载进内存(在 Android 4. 4 版本之后出现的 MultiDex 技术,移动应用中有可能存在多个 DEX 文件,需要逐一处理所有 DEX 文件),然后通过递归扫描的方式,获取每个 DEX 文件中所有的类的字节码,并存放在内存中的列表中. 最后遍历列表,将这些类的类名逐行写入到配置文件中提供给动态分析模块.

3.3 动态分析

动态分析模块分为动作触发和行为监控两部分. 行为触发方面,现有研究和应用中已经存在 Monkey[20]、Droidbot[21]、APPCrawler[22] 等自动化测试软件,但是本文通过虚拟机进行批量动态分析时,以上工具均存在无法输入、闪退、卡顿等情况. 因此,本文设计实现了一种简单、鲁棒的 Android 应用行为触发工具 Traverse UI,该工具能够在自动安装运行程序的基础上,自动模拟用户对移动应用的操作,并实现了自动重启崩溃或跳出的应用、指定最大递归次数避免无限递归、首次启动应用时尝试滑动操作、尝试跳过点击过的控件、避免点击可能导致退出应用的控件、尝试进行文本输入、多次点击错误后自动重启应用等功能. Traverse UI 在自动化测试工具 Appium[23] 的基础上实现,能够规避和处理可能出现的错误,具有较高的鲁棒性.

行为监控方面,现有研究对 Android 应用进行行为监控的主要方法是应用层的插装和系统层的敏感 API 监控. 插装通过解包应用,增添输出信息的代码后重打包,然后在运行应用过程中获取插装代码的输出信息实现,其优势在于可以监控应用的任何方法调用,缺点是实现较为复杂,且容易被干扰或者阻断. 敏感 API 监控通常是通过 Xposed 等框架去 Hook 特定 API,从而获得应用运行时的 API 的调用信息. 其主要缺陷是穷举所有敏感 API 是困难的,容易被绕过;同时,移动应用可能通过检测框架的指纹信息发现监控模块,进而避免运行恶意载荷,其对抗性较差. 因此,本文设计实现了一种新的监控方式,采用动静态结合的方式,在不改写移动应用的情况下实现行为监控. 行为监控功能主要通过对 Android 系统中的解释器进行插装实现:读取静态分析所提供的应用所含类、SDK 中敏感 API 所在类等信息,在指定的方法被调用时,记录方法的调用信息. 行为监控模块的实现建立在 AppSpear[24] 基础上. 该监控方法不需要对各个应用进行插装,但是效果类似于插装,可以监控应用的所有行为,攻击者无法通过调用冷僻的 API 绕过监控,也很难发现或对抗监控模块.

4 恶意代码检测

4.1 深度学习框架

通过上文所设计实现的移动应用行为分析方法,能够从移动应用中提取结构信息、权限信息、防护信息以及行为信息. 在此基础上,首先进行特征提取,在尽可能剔除易变因素的基础上,提取出能够反应移动应用能力范围的能力特征和能够体现其行为模式的行为特征;然后对所提取的特征进行预处理,将每个移动应用所对应的特征都转换为按统一方式编码的一行数据,方便后续操作;最后,通过一个 5 层 LSTM 神经网络模型实现正常和恶意应用的分类. 深度学习框架主要包括特征提取、预处理、分类模型 3 个模块,其结构如图 2 所示.

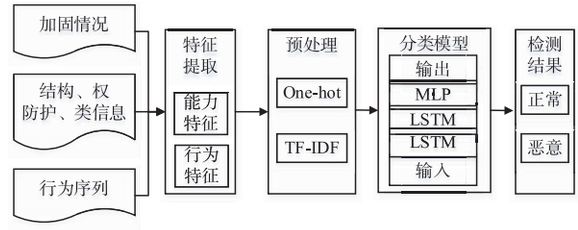

图2 基于深度学习的恶意代码检测框架

4.2 特征提取

对所提取的信息进行初步分析后发现,恶意应用往往是具有功能较为简单的小程序,具体表现为体积小、组件数量少、申请权限数量少(恶意代码往往不会申请其攻击行为不需要的权限,如蓝牙、录音等)等方面. 对于恶意应用的开发者,改变程序的大小、增加组件数量、申请权限都非常容易实现且不会影响移动应用原有的功能,很容易通过这些变换绕过相关检测方法.

因此,本文试图设计一种能够剔除易变特征,通过“非易变特征”准确区分正常和恶意应用的方法,以保持检测的鲁棒性.

基于上述思路,本文从能力、行为两方面进行特征提取. 其中,能力特征可以刻画程序的能力范围,揭示程序“能干什么”. 该特征与现有检测方法所用的权限特征相类似,但存在以下两方面的区别:一方面,能力特征只保留了与实现恶意功能关联较为密切的权限特征;另一方面,能力特征包含了权限之外的,能够体现防护能力的特征,包括加固类型,是否存在反射和 Native 代码以及类的数量等. 行为特征体现特定类型行为出现的情况,揭示程序在实际运行中“干了什么”. 本文将动态分析所得到的信息行为划分为短信、电话、加密、网络、位置等 58 类,所提取行为特征为 58 维向量,向量中各个元素分别表示不同类型行为的出现次数. 能力特征、行为特征均与程序的核心功能紧密相关,与结构特征、权限特征相比,攻击者实现模拟攻击的难度更高.

4.3　预处理

能力特征方面,针对不用类型特征,采用不用的编码方式. 对是否申请某个权限以及用反射和 Native 代码,采用 One-hot 模式进行编码,即用 0,1 分别表示是否具有该能力;加固类型的值域为{0,1,2,…,9},分别表示未加固、加百度壳、加腾讯壳等的 10 种不同情况.

对于行为特征,我们参照现有文本分析方法,通过 TF-IDF(Term Frequency-Inverse Document Frequency)表示行为特征. 其中,TF(Term Frequency)为某个关键词在当前移动应用动态特征数据中出现的频率,IDF(Invers Document Frequency)为计算倒排文本频率,此处文本频率是指某个关键词在所有动态数据中出现的次数,倒排文本频率主要用于降低所有数据中一些常见却对影响不大的词语的作用. TF-IDF 的计算如式(1)所示.

$$tfidf_{ij} = \frac{n_{ij}}{\sum_k n_{kj}} \times \log \frac{|D|}{N_i} \tag{1}$$

其中,$tfidf_{ij}$表示第 j 个 App 中第个 i 关键词对应的 TF-IDF 值,n 表示特定关键词的出现次数,$|D|$表示移动应用的数量,N_i表示含有关键词 i 的移动应用的个数. 这里的关键词为加密、网络、位置等 58 种不同类型的行为.

最后,我们对提取的特征集合进行格式化处理,统一处理为 CSV(Comma Separated Values)格式,每一行为一个移动应用样本所对应的特征.

4.4　基于 LSTM 的分类模型

长短时记忆网络(Long Short-Term Memory,LSTM)是一种循环神经网络,适用于处理和预测时间序列中间隔和延迟相对较长的重要事件. LSTM 在算法中加入了一个判断信息有用与否的“处理器”. 而我们所使的多维特征中,存在大量无用的、并不能在区分正常和恶意应用的特征,所以 LSTM 能够遗忘一定信息的特点适合此处恶意代码检测的场景.

选用 LSTM 的另一个原因是决策树、支持向量机(Support Vector Machine,SVM)等传统分类方法往往假设特征之间是相互独立的. 它们更善于刻画特征之间的浅层关系. 然而本文所提出的特征之间并非相互独立,而是具有较强且复杂的依赖关系,例如体现目标应用在测试过程中发送短信次数的行为“特征依赖”于目标应用能否发送短息的“权限特征”. 因而本文选用深度神经网络,以期能够刻画特征之间难以定义的深层次关系.

本文通过一个多层神经网络模型来实现正常和恶意软件的分类,其结构如图 3 所示. 从输入到输出依次为输入层、若干个 LSTM 层、全连接层和 Relu 二分类输出层. 在训练模型时,预先训练一个单层 LSTM 网络以获取各层结点数、学习率及结点初始权重的最佳值,实验结果表明当输出结点数设置为 60,全连接结点数设置为 50,学习率设置为 0.01 时,特征分类准确率最高. 本文提出的基于 LSTM 的神经网络模型通过 Deeplearning4j[25] 实现. 本文所设计的多层神经网络的激活函数均为 ReLU(Rectified Linear Unit),其余参数使用 Deeplearning4j 中的默认值.

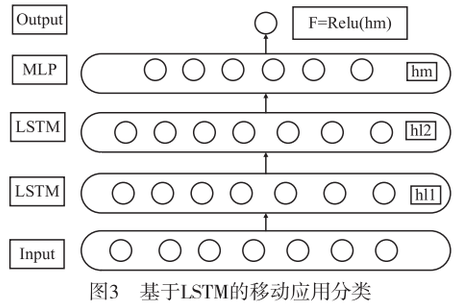

图3　基于LSTM的移动应用分类

5　实验与分析

5.1　实验对象与统计分析

本文选取了 44798 个移动应用作为静态分析的目标. 其中,正常样本共计 32175 个,包括下载自 360 移动市场的应用 11806 个,下载自豌豆荚的移动应用 18674 个,以及 Arash Habibi Lashkari 等人[26] 所提供的来自 Google play 的样本 1695 个. 恶意样本共计 12623 个,包括安天公司提供的恶意样本 973 个,来自 Gennome[27] 的恶意样本 1260 个,来自 AMD[28] 的恶意样本 10390 个. 表 1 总结了正常和恶意代码在包大小以及各类组件数

量方面的区别,各列均为平均数值.

表 1 正常和恶意代码结构特征的对比

类别	大小(M)	Activity	Service	Content Provider	Broadcast Receiver	权限
正常	15.21	50.01	5.34	1.09	3.29	21.34
恶意	1.06	5.27	1.54	0.05	2.16	14.76

能够看出,恶意代码在程序大小、组件组成、权限申请等方面和正常应用存在较为明显的区别,权限特征和结构特征可以被用来进行恶意代码检测. 但是,更改程序大小、增加组件数量、额外申请权限等操作对于恶意代码开发者而言是比较容易实现的,因此,此类检测方法面临模拟攻击的威胁,其对抗性较差.

5.2 基于动静态结合分析的恶意代码检测

由于动态行为分析需要在模拟器中安装、执行每个应用,分析的效率较低,所以只对 2724 个移动应用进行了动态分析,其中包括正常和恶意应用各 1362 个. 然后使用 K 最近邻、J48 决策树,SVM 以及 LSTM 等 4 种现有研究中表现较好的算法来区分正常和恶意应用. 对于 LSTM 算法,除使用全部的能力特征和行为特征外,还分别尝试用能力特征和行为特征进行分类,实验结果如表 2 所示. 结果显示,算法方面,本文所设计的 LSTM 模型具有最高的检测能力;特征方面,能力特征和行为特征都具有良好的区分正常和恶意应用的能力.

表 2 基于能力-行为特征的恶意应用检测结果

算法	特征	ACC	Precision	Recall	F-1
K 最近邻	能力-行为特征	0.954	0.954	**0.954**	0.954
J48 决策树	能力-行为特征	0.938	0.939	0.938	0.938
SVM	能力-行为特征	0.929	0.920	0.940	0.930
LSTM	能力特征	0.925	0.931	0.919	0.925
LSTM	行为特征	0.898	0.890	0.907	0.899
LSTM	能力-行为特征	**0.961**	**0.974**	0.949	**0.961**

对基于 LSTM 和能力-行为特征分类出现的 106 个漏报和误报的样本,手动分析发现漏报的主要原因是恶意代码设定了较为复杂的行为触发机制或者虚拟机检测机制,动态分析模块未能触发其恶意载荷. 例如,ADRD 家族的恶意代码会检测程序被安装的时间;bankbot 家族的恶意代码会进行虚拟机检测. 此外,钓鱼型恶意代码也造成了一定漏报,这是因为此类恶意代码主要通过诱骗、欺诈实现攻击意图,其程序行为特征与正常应用不存在明显区别. 误报的主要原因是部分正常样本也存在较为敏感的行为,例如,来自豌豆荚的不少壁纸、美化类应用在程序启动的时候会获取用户的设备号、电话号码等信息并上传.

5.3 和其他工作的比较

表 3 为本文方法与类似工作的比较. 可以看出,本文方法具有较强的对抗性和较高的检测正确率.

表 3 本文方法与类似工作的比较

方法	程序分析方法	特征	分类算法	对抗性	准确率(%)
Shiqi Li 等人提出的方法[29]	静态分析	结构,文本和权限	DBN	使用了易变的结构,文本和权限特征	95.7
F Martinelli 等人提出的方法[30]	动态分析	系统事件,UI 事件	CNN	动作序列长度较短,易逃避	85-95
DDefender[31]	混合分析	系统调用数量、权限、组件信息	MLP	使用了易变的结构和权限特征	95
R Vinayakumar 等人提出的方法[32]	混合分析	权限、电池、内存、CPU 等	CNN LSTM	使用了易变的权限特征	93.9-97.5
本文工作	混合分析	权限,防护,系统调用	LSTM	破解了部分对抗静态分析、动态分析方法,剔除了部分易变特征	96.14

6 总结

本文设计实现了一种基于长短时记忆网络的 Android 恶意代码检测方法. 程序行为分析方面,通过脱壳重打包破解了加壳对静态分析的干扰;设计实现了具有较强鲁棒性的动作触发方法和具有较强对抗性行为监控方法. 恶意代码检测方面,剔除或避开了攻击者容易更改的特征;设计了基于 TF-IDF 的行为特征表示方法以及基于 LSTM 的神经网络模型来实现分类. 本文所设计的方法对 1362 个正常应用和 1362 个恶意应用的分类准确率达到了 96.14%,能够为 Android 恶意代码的检测提供理论和技术支持.

参考文献

[1] 腾讯移动安全实验室. 2019 年上半年手机安全报告[OL]. https://m.qq.com/security_lab/news_detail_517.html,2019-07-18/2020-03-01.

[2] Naway A,Li Y. A review on the use of deep learning in android malware detection[J]. International Journal of Computer Science and Mobile Computing,2018,7(10):42-58.

[3] Huang H,Cong Z,Zeng J,et al. Android malware develop-

ment on public malware scanning platforms: A large-scale data-driven study[A]. Proceedings of the IEEE International Conference on Big Data[C]. Washington D. C: IEEE, 2016. 1090 – 1099.

[4] Yang W, Zhang Y, Li J, et al. Appspear: Bytecode decrypting and dex reassembling for packed android malware [A]. International Workshop on Recent Advances in Intrusion Detection[C]. Cham: Springer, 2015. 359 – 381.

[5] 乐洪舟,张玉清,王文杰,等. Android 动态加载与反射机制的静态污点分析研究[J]. 计算机研究与发展,2017,54(2):313 – 327.
Yue Hongzhou, Zhang Yuqing, Wang Wenjie, et al. Android static taint analysis of dynamic loading and reflection mechanism[J]. Journal of Computer Research and Development, 2017, 54(2): 313 – 327. (in Chinese)

[6] Wang X, Zhu S, Zhou D, et al. Droid-AntiRM: Taming control flow anti-analysis to support automated dynamic analysis of android malware[A]. Proceedings of the 33rd Annual Computer Security Applications Conference[C]. USA: ACM, 2017. 350 – 361.

[7] Hoffmann J, Ussath M, Holz T, et al. Slicing droids: Program slicing for smali code[A]. Proceedings of the 28th Annual ACM Symposium on Applied Computing[C]. USA: ACM, 2013. 1844 – 1851.

[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] Li L, Bartel A, Klein J, et al. I know what leaked in your pocket: Uncovering privacy leaks on android apps with static taint analysis[J]. arXiv Preprint, 2014, arXiv: 1404. 7431.

[10] Enck W, Gilbert P, Han S, et al. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones[J]. ACM Transactions on Computer Systems(TOCS), 2014, 32(2): 5.

[11] Arp D, Spreitzenbarth M, Hubner M, et al. DREBIN: Effective and explainable detection of android malware in your pocket[A]. Proceedings of the Network and Distributed System Security Symposiu[C]. SanDiego: ISOC, 2014. 1 – 12.

[12] Feizollah A, Anuar N B, Salleh R, et al. AndroDialysis: Analysis of android intent effectiveness in malware detection[J]. Computers & Security, 2017, 65: 121 – 134.

[13] Pengwei LI, Jianming FU, Chao XU, et al. Differentiating malicious and benign android App operations using second-step behavior features[J]. Chinese Journal of Electronics, 2019, 28(5): 944 – 952.

[14] 王兆国,李城龙,张洛什,等. 一种基于行为链的 Android 应用隐私窃取检测方法[J]. 电子学报,2015,43(9):1750 – 1755.
WANG Zhao-guo, LI Cheng-long, ZHANG Luo-shi, et al. A privacy stealing detection method based on behavior-chain for android applications[J]. Acta Electronica Sinica, 2015, 43(9): 1750 – 1755. (in Chinese)

[15] 张鹏,牛少彰,黄如强. 基于资源签名的 Android 应用相似性快速检测方法[J]. 电子学报,47(9):1913 – 1918.
ZHANG Peng, NIU Shao-zhang, HUANG Ru-qiang. A fast and resource-based detection approach of similar android application[J]. Acta Electronica Sinica, 2019, 47(9): 1913 – 1918. (in Chinese)

[16] 王蕊,苏璞睿,杨轶,等. 一种抗混淆的恶意代码变种识别系统[J]. 电子学报,2011,39(10):2322 – 2330.
WANG Rui, SU Purui, YANG Yi, et al. An anti obfuscation malware variants identification system[J]. Acta Electronica Sinica, 2011, 39(10): 2322 – 2330. (in Chinese)

[17] Mclaughlin N, Rincon J M D, Kang B J, et al. Deep android malware detection[A]. Proceedings of the ACM Conference on Data & Application Security & Privacy[C]. Scottsdale, AZ: ACM, 2017. 301 – 308.

[18] Zhenlong, Yuan, Yongqiang, et al. Droid detector: Android malware characterization and detection using deep learning[J]. Tsinghua Science & Technology, 2016, 21(1): 114 – 123.

[19] Desnos A. Androguard: Reverse Engineering, Malware and Goodware Analysis of Android Applications and More[OL]. https://code. google. com/p/androguard/, 2013-03-26/2020-03-01.

[20] UI/Application Exerciser[OL]. http://Monkey. developer. android. com/guide/developing/tools/monkey. html, 2020-03-01.

[21] Li Y, Yang Z, Guo Y, et al. Droidbot: A lightweight UI-guided test input generator for android[A]. IEEE/ACM 39th International Conference on Software Engineering Companion(ICSE-C)[C]. USA: IEEE, 2017. 23 – 26.

[22] Seveniruby. 基于 appium 的 app 自动遍历工具[OL]. https://github. com/seveniruby/AppCrawler, 2020-03-01.

[23] Jonathan. Appium: Mobile App Automation Made Awesome[OL]. http://appium. io/, 2020-03-01.

[24] Li B, Zhang Y, Li J, et al. AppSpear: Automating the hidden-code extraction and reassembling of packed android malware[J]. Journal of Systems and Software, 2018, 140: 3 – 16.

[25] Alex Black. Deep Learning for Java, Scala & Clojure on Hadoop, Spark & GPUs [OL]. https://github. com/eclipse/deeplearning4j, 2020-03-01.

[26] Arash Habibi Lashkari, Andi Fitriah A Kadir, Hugo Gonzalez, et al. Towards a network based framework for android malware detection and characterization[A]. Pro-

ceedings of the 15th International Conference on Privacy, Security and Trust[C]. Calgary: IEEE, 2017. 233 – 234.

[27] Jiang X, Zhou Y. Dissecting android malware: Characterization and evolution[A]. Proceedings of the IEEE Symposium on Security and Privacy[C]. San Francisco' Bay Area: IEEE, 2012. 95 – 109.

[28] Wei F, Li Y, Roy S, et al. Deep ground truth analysis of current android malware[A]. Proceedings of the International Conference on Detection of Intrusions and Malware, And Vulnerability Assessment[C]. Bonn: SIDAR, 2017. 252 – 276.

[29] Shiqi L, Shengwei T, Long Y, et al. Android malicious code classification using deep belief network[J]. KSII Trans on Internet Inf Syst, 2018, 12(1): 454 – 475.

[30] Martinelli F, Marulli F, Mercaldo F. Evaluating convolutional neural network for effective mobile malware detection[J]. Procedia Comput Sci, 2017, 1(112): 2372 – 2381.

[31] Alshahrani H, Mansourt H, Thorn S, et al. DDefender: Android application threat detection using static and dynamic analysis[A]. Proceedings of the International Conference on Consumer Electronics[C]. Las Vegas: IEEE, 2018. 1 – 6.

[32] Vinayakumar R, Soman K P, Poornachandran P, Sachin Kumar S. Detecting Android malware using Long Short-term Memory(LSTM)[J]. J Intell Fuzzy Syst, 2018, 34(3): 1277 – 1288.

作者简介

李鹏伟 男,1987 年 7 月出生于山西省五寨县. 现工作于南京审计大学信息工程学院. 主要研究方向为系统安全和软件安全.
E-mail: pwli@nau.edu.cn

姜宇谦 男,1998 年出生于上海. 现为南京审计大学计算机科学与技术专业在读本科生. 主要研究方向为网络与信息安全.
E-mail: jyq9898@163.com