中国科学: 信息科学 2020 年 第 50 卷 第 8 期: 1148–1177
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
评述

安卓恶意软件检测方法综述

范铭[1], 刘烃[1*], 刘均[3], 罗夏朴[2], 于乐[2], 管晓宏[1]

1. 西安交通大学网络空间安全学院, 西安 710049
2. 香港理工大学计算机系, 香港 999077
3. 西安交通大学计算机学院, 西安 710049
* 通信作者. E-mail: tingliu@mail.xjtu.edu.cn

收稿日期: 2019–07-12; 修回日期: 2019–09–30; 接受日期: 2020–02–03; 网络出版日期: 2020–07–31

国家重点研发计划 (批准号: 2016YFB1000903)、国家自然科学基金 (批准号: 61902306, 61632015, U1766215, 61772408, 61833015)、国家自然科学基金创新群体 (批准号: 61721002)、教育部创新团队 (批准号: IRT_17R86)、中国博士后科学基金站前特别资助 (批准号: 2019TQ0251) 资助项目

摘要 在过去的十几年间, Android 系统由于其开源性、丰富的硬件选择性以及拥有百万级别应用市场等优点, 已经迅速成为了目前最流行的移动操作系统. 与此同时, Android 系统的开源性也使其成为了恶意软件的主要攻击目标. 恶意软件的快速增长给移动智能手机用户带来了巨大的危害, 包括资费消耗、隐私窃取以及远程控制等. 因此, 深入研究移动应用的安全问题对智能手机生态圈的健全发展具有重要意义. 本文首先介绍了恶意软件检测所面临的问题与挑战, 然后综述了近些年的恶意软件检测所使用的数据集信息以及相关方法, 将现有方法分为了基于特征码、基于机器学习以及基于行为 3 大类, 并针对各方法所使用的技术进行了归纳总结, 全面比较和分析了不同技术的优缺点. 最后, 结合我们自身在恶意软件检测方面的研究基础对未来的研究方向和面临的挑战进行了探索与讨论.

关键词 安卓, 恶意软件检测, 家族识别, 机器学习

1 引言

Android 是由 Google 公司以及开放手机联盟于 2007 年领导以及开发的一种基于 Linux 的自由及开放源代码的操作系统, 它被广泛使用于移动设备. 第一部 Android 智能手机发布于 2008 年 10 月, 随后, Android 逐渐扩展到平板电脑以及其他领域上, 如数码相机、电视、游戏机、智能手表等. 相比于传统的塞班 (Symbian) 移动操作系统, Android 具有以下 3 个优点: (1) 开源性, 开源的代码库、免费的开发软件、社区、第三方开源共享等特点吸引越来越多的 Android 系统开发者, 在带来巨大的竞

引用格式: 范铭, 刘烃, 刘均, 等. 安卓恶意软件检测方法综述. 中国科学: 信息科学, 2020, 50: 1148–1177, doi: 10.1360/SSI-2019-0149
Fan M, Liu T, Liu J, et al. Android malware detection: a survey (in Chinese). Sci Sin Inform, 2020, 50: 1148–1177, doi: 10.1360/SSI-2019-0149

争的同时也使得 Android 系统在开放的平台中显得日益成熟. (2) 丰富的硬件选择性, 由于其开源性, 很多厂商为了吸引更多的用户, 对 Android 系统加以改造, 推出功能特色各具的各种产品, 而不会影响到数据同步、软件兼容等功能, 从而不断丰富用户体验. (3) 百万级别 app 应用市场, 无论国内国外均有能够为用户提供各种丰富 app 下载的 Android 应用市场, 例如官方的 Google Play[1)], 第三方市场安智[2)]等. 截至 2018 年 8 月, Google Play 上可供下载应用数量超过 280 万款, 总下载量近 3300 亿次. 以上优点使得 Android 迅速成为了最流行的移动操作系统. 根据研究公司 Gartner 提供的数据, 在 2018 年第一季度, Android 已经占据了 85.9% 的移动操作系统市场份额[3)].

然而, Android 的开源性以及流行性也使它成为了 97% 恶意软件的主要攻击目标[4)], 现如今各大平台市场上充斥着各种各样的恶意软件. 根据 360 公司 2017 年 Android 恶意软件年度专题报告, 2017 年全年, 360 互联网安全中心累计截获 Android 平台新增恶意软件样本 757.3 万个, 平均每天新增 2.1 万个[5)]. 从用户感染恶意软件情况看, 2017 年 Android 用户感染恶意软件数量为 2.14 亿. 除此之外, 根据 Wang 等 [1] 在 16 个中国安卓市场以及 Google Play 官方市场上共计 600 万应用的大规模分析结果, 中国安卓市场上大约 12.30% 的 app 被至少 10 个反病毒引擎报告为恶意, 即使是 Google Play 官方市场, 这一数据仍可达 2.09%. 恶意软件的快速增长给移动智能手机用户带来了巨大的危害, 包括资费消耗、隐私窃取以及远程控制等. 资费消耗主要是针对手机用户的资费, 强行定制服务并从中牟利. 隐私窃取主要实现短信、通讯录和通话记录的获取, 定位以及拍照等功能, 收集用户的隐私数据、社交数据以及设备数据. 远程控制主要是使用 http 协议接收 C&C (command-and-control) 服务器的控制指令实现指令操纵、信息回传以及本地恶意代码更新等.

由此可见, 深入研究移动应用的安全问题对智能手机生态圈的健全发展具有重要意义. 近些年移动恶意软件检测技术的快速发展使其迅速成为了学术界软件安全领域中的热点问题, 并已有大量优秀的相关研究成果发表于顶级的国际会议和期刊上, 如 ICSE 会议 [2~4]、FSE 会议 [5]、ASE 会议 [6,7]、S&P 会议 [8]、SECURITY 会议 [9~11]、CCS 会议 [12~14]、NDSS 会议 [15~17]、TIFS 期刊 [18~20] 等. 目前针对 Android 安全研究进行相关综述的有 2018 年 Liu 等 [21] 在软件学报上发表的 “软件与网络安全研究综述”, 2016 年 Qing [22] 在软件学报上发表的 “Android 安全研究进展”, 以及 2014 年 Zhang 等 [23] 在计算机研究与发展上发表的 “Android 安全综述”. 但是 Liu 等 [21] 主要从恶意软件、软件漏洞以及软件安全机制 3 个方面进行综述, 其针对移动恶意软件的检测方法部分缺乏更详细的归纳总结. 而 Qing [22] 以及 Zhang 等 [23] 主要从 Android 体系结构以及安全机制等面临的威胁出发探讨了 Android 安全的研究现状, 其发表时间较早, 缺乏近些年新型移动恶意软件检测技术的讨论分析. 因此本文综述了近些年的恶意软件检测所使用的数据集信息以及相关技术, 针对该技术进行了归纳总结, 全面比较和分析了不同技术的优缺点. 最后, 结合我们自身在恶意软件检测方面的研究基础对未来的研究方向和面临的挑战进行了探索.

本文结构如下: 第 2 节对恶意软件检测相关定义以及所遇到的主要挑战进行了介绍; 第 3 节介绍现有的恶意软件数据集; 第 4 节介绍基于特征码的恶意软件检测方法; 第 5 节介绍基于机器学习的恶意软件检测方法; 第 6 节介绍基于行为的恶意软件检测方法; 第 7 节探讨了恶意软件检测所遇到的一些新问题以及未来的研究方法; 最后第 8 节总结了全文.

1) Google play. 2019. https://play.google.com/store.

2) Anzhi market. 2019. http://www.anzhi.com/.

3) Gartner says worldwide sales of smartphones returned to growth in first quarter of 2018. 2018. https://www.gartner.com/newsroom/id/3876865.

4) 97% of mobile malware is on Android. This is the easy way you stay safe. 2014. http://goo.gl/MYDBKC.

5) Report of smartphone security in China. 2017. http://zt.360.cn.

2 恶意软件检测的问题与挑战

2.1 分析对象

本小节首先对本文中的 Android 恶意软件以及恶意家族给出相关定义.

恶意软件. 本文从恶意行为的实现过程将恶意软件分为两大类: (1) 一类是指在 Android 系统上执行恶意任务的病毒、蠕虫和特洛伊木马的程序, 它们通过破坏软件进程实施控制. 该类恶意软件由恶意制作者具有特定目的地完成并可以自行完成恶意行为, 从而对用户直接造成经济损失或隐私泄露. 例如近些年 Android 平台上出现的挖矿木马程序, 该木马指在用户不知情的情况下利用其手机的计算能力来为攻击者获取电子加密货币的应用程序, 仅在 2018 年 1 月被检测出的挖矿木马约有 400 个[6]. (2) 另一类是指由正常开发者自主完成的程序, 该类程序并不会直接对用户造成危害, 但是其自身存在漏洞或者缺陷, 从而会被第一类恶意软件所利用, 导致用户的隐私和财产间接受到威胁. 例如文献 [24] 中发现的一款名为 Qunar 的应用 (37000000+ 下载量) 在用户付账时使用了不安全的 SSL 方法来传输账户相关信息, 该缺陷的存在可能会被其他恶意软件加以利用从而造成巨大的经济损失, 因此该类应用的安全问题也是目前主要的分析对象.

恶意家族. 同一个恶意家族中的样本完成相似的恶意行为, 并且大部分新截获的恶意样本是已有家族样本的变种. 例如 plankton 家族, 它的首个家族成员出现在 2011 年, 但目前仍是传播范围最广的 Android 恶意软件之一 [25]. 即使是在官方 Android 应用程序商店 Google Play, plankton 恶意代码也出现在为数众多的应用程序中, 其家族成员主要为不同版本的广告软件, 主要恶意行为是在手机中下载不需要的广告或更改移动设备的浏览主页, 或添加新的快捷方式与书签. 近些年, 已有工作发现将新检测的恶意样本划分至其相应的家族并在每个家族中选择代表性样本供安全人员进行进一步地详细分析可以大大减小安全人员的工作量 [26~28], 因此恶意家族识别也成为了现如今恶意软件检测中的主要研究问题之一.

2.2 面临的挑战

针对恶意软件检测问题, 现如今的恶意软件检测方法存在以下 4 个主要挑战:

(1) 恶意代码形态多样化. 为了绕过现有的恶意软件检测技术, 恶意代码制作者会通过不断修改恶意软件源码制造出多种类型变种. 文献 [8] 中研究结果表明 86% 的恶意软件是通过重打包方式制作而成, 即通过将恶意加载代码植入到原 app 的 Dalvik 代码中, 然后通过重新打包并上传至 app 市场上吸引用户免费下载. 同一个家族中的恶意加载代码部分由于制作者的不断修改, 其相同的恶意行为会具有不同的代码实现方式.

除此之外, 日趋成熟的混淆技术也使得经过混淆后的恶意代码在保留恶意行为正常执行的前提下, 表现形式发生很大改变, 从而使得恶意软件的检测越来越难. 现有的针对 Android 程序的混淆技术可以分为基本混淆技术与高级混淆技术. 基本混淆技术包括布局调整、重命名以及控制流混淆等 [29, 30]. 高级混淆技术包括反射机制 [31]、加固技术 [32][7] 以及将恶意行为隐藏在 native 代码中 [33~35]. 现有的混淆工具也有很多, 典型工具有 DashO[8], ProGuard[9], DexGuard[10], Allatori[11], Bangcle[12] 等.

6) Android platform mining horse research report. 2018. https://www.freebuf.com/articles/paper/161741.html.
7) 360 Packer. 2019. http://jiagu.360.cn/index.html.
8) DashO. 2019. https://www.preemptive.com/products/dasho/overview.
9) ProGuard. 2019. http://proguard.sourceforge.net/.
10) DexGuard. 2019. https://www.guardsquare.com/en/products/dexguard.
11) Allatori Java Obfuscator. 2019. http://www.allatori.com/.
12) Bangcle Security. 2019. https://www.bangcle.com/.

表 1 3 类方法针对已有挑战的解决能力描述

Table 1 The ability of three kinds of methods to handle existing challenges

Method	Diversity of malicious code	Huge samples to analyze	Difficulty of labeling	Bad interpretability
Signature-based method	Weak	Strong	Weak	Medium
Machine Learning-based method	Strong	Strong	Weak	Weak
Behavior-based method	Strong	Weak	Strong	Strong

针对 Android 程序混淆技术的研究也是现如今非常热门的课题, 感兴趣读者可以参见文献 [36~38].

(2) 待分析样本数量大. 现实情况下安全检测系统每天面临的是成千上万的待分析样本, 详细地分析每一个样本是一件耗时费力的工作. 除此之外, 近些年还出现了恶意软件工厂, 即自动变形恶意软件从而能够全自动化生成大量恶意软件变种, 使得恶意软件数量急剧增加. 例如在 2017 年首次在国际黑客会议 BlackHat 上公开的 AVPASS [39], 它是一个可探测 Android 杀毒软件的检测模型, 通过结合探测到的信息和混淆技术构造特定 APK 来绕过杀毒软件检测. AVPASS 实现了自动化免杀, 这种技术已经衍生出多种类型软件的一键生成器, 比如勒索软件和广告软件等. 因此, , 如何有效降低样本分析的时间开销, 使得检测工具能够处理大规模样本集是现如今检测方法所要考虑的主要问题.

(3) 样本数据标注难. 大部分恶意软件检测方法均依赖监督学习方法, 也就是基于大量已标注样本数据构建分类器, 从而对未标注样本进行分类. 但是, 获得大量已标注信息的样本是一件非常耗时费力的工作. 并且, 实际情况中部分新截获的恶意样本隶属于未知家族, 此时基于有监督学习的家族识别方法无法处理该类型样本. 除此之外, 新的恶意样本持续被发现, 基于旧样本的恶意行为相关信息去预测判断未知恶意行为是十分具有挑战性的, 即概念漂移现象 [40] (在第 7 节中详细讨论).

(4) 可解释性差. 大部分已有方法仅仅提供待分析样本是否为恶意样本的决策结果. 然而, 已有决策结果无法给安全分析人员提供足够有效的信息去解释该样本具有哪些恶意行为, 以及其恶意行为代码如何实现. 因此, 已有方法的可解释能力较差, 离产业应用还有较大距离. 在准确率和效率的前提下, 一个方法具有好的结果解释性可以帮助安全人员对分析方法做进一步改进, 并提醒用户如何预防恶意行为的发生 [41, 42]. 因此, 可解释性能力也逐渐成为了现如今方法所要考虑的因素之一.

针对上述 4 个主要挑战, 有效的恶意软件检测技术需要满足 4 点, 即检测率高、运行时开销小、所需标注样本数量少、可解释能力强. 但是现有的方法往往只能满足其中的一条或两条, 因此在恶意软件检测领域亟需提出行之有效的解决方法.

2.3 已有技术分类

本文根据应用场景与技术手段的不同, 将已有的恶意软件检测方法划分为 3 大类: (1) 基于特征码方法; (2) 基于机器学习方法; (3) 基于行为方法. 表 1 对各类方法相对于上述挑战的解决能力进行了归纳总结. 其中解决能力分为强、中、弱 3 个级别, 强表示该方法能够较好地解决相应挑战, 弱则表示相应挑战会对该方法带来一定限制.

具体而言, 基于特征码方法通过将待检测软件的特征码与已有恶意软件的特征码进行匹配达到检测目的, 因此其运行时开销小, 并且能够提供一定的证据 (所检测的恶意特征码) 来解释说明. 但是, 该类方法容易被经过变形的恶意代码所绕过, 因此无法有效应对挑战 1. 除此之外, 该类方法依赖特征码数据库, 因此挑战 3 对其也带来一定影响. 相关技术在第 4 节详细讨论分析.

基于机器学习方法能够通过特征工程将程序转换为特征向量进行表示, 然后构建分类器从而快速准确地对恶意软件进行检测和分类. 该类方法能够在一定程度上应对不同的恶意代码变种, 所需的时间开销也较少. 但是该类方法需要大量标注样本进行学习, 并且所构建的分类器对用户而言是不透明的, 可解释能力弱. 相关技术在第 5 节详细讨论分析.

基于行为方法针对特定的恶意行为提出定制化的检测方法, 该类方法能够对检测出的恶意软件行为进行有效的解释说明. 但是它们往往采用了高开销的数据流分析、自然语言处理等技术, 因此无法处理大规模样本集. 相关技术在第 6 节详细讨论分析.

3 恶意软件数据集

恶意软件检测离不开有效恶意样本集的支撑. 现如今常用的恶意样本集根据其来源主要分为两类, 一类是恶意样本共享网站, 典型的如 Contagio[13)] 与 VirusShare[14)], 另一类是具有家族信息的常用数据集, 包括 Genome 数据集 [8]、Drebin 数据集 [15]、FalDroid 数据集 [26]、DroidBench 数据集[15)]、AMD 数据集 [43] 以及 RmvDroid 数据集 [44].

恶意样本共享网站上提供了大量恶意样本供研究人员分析使用, 大大推动了恶意软件检测技术的发展. 但是该类样本具有两个不足之处: (1) 不包含家族相关信息, 无法被用于恶意家族识别工作, 需要人工对样本数据进行重新标注. (2) 掺杂其他类型恶意软件, 经研究发现, 该网站上提供的样本集中并不完全是 Android 相关的恶意样本, 即包含其他类型恶意样本, 因此需要对样本进行初步筛选.

第 2 类数据集则可直接用于恶意软件检测与家族识别工作, 它们的详细介绍如下.

Genome 数据集由 Zhou 与 Jiang 的团队 [8] 花费了 1 年多的时间构建而成, 它包含了 49 个家族, 共计 1260 个恶意软件, 样本分布时间为 2011~2012 年. 该团队花费大量人力物力对该数据集中的样本进行了详细的分析, 分析结果发现: (1) 86% 的恶意软件通过重打包方法制作而成, 即制作者选择流行应用程序进行反编译, 植入恶意加载代码, 然后重新编译并上传至市场上吸引用户免费下载; (2) 36.7% 的恶意软件利用已知的平台漏洞提升特权; (3) 93% 的恶意软件使用基于 http 协议来接收 C&C 服务器的控制指令. 针对该数据集的分析结果也成为了后续许多相关工作的研究背景.

Drebin 数据集由 Arp 等 [15] 在 Genome 数据集的基础上扩展而成. 该数据集包含 179 个家族, 共计 5560 个恶意软件, 样本分布时间为 2011~2014 年.

FalDroid 数据集 [26] 由我们自己构建而成, 主要原因是尽管 Genome 与 Drebin 数据集包含足够多的恶意家族, 但是其中部分家族仅仅包含少量样本, 例如 Genome 数据集中 16 个家族仅包含 1 个样本, Drebin 数据集中 47 个家族仅包含 1 个样本. 因此我们通过对 VirusShare 中的恶意样本进行家族信息标注, 从而构建出一个包含 36 个家族共 8407 个样本的有效数据集, 样本分布时间为 2013~2014 年. 该数据已经公开.

DroidBench 数据集由 Fritz 与 Arzt 等构建并发布. 该数据集是一个开放的测试集, 包含 119 个样本, 分布时间为 2014~2016 年. 该数据集主要用于评估针对 Android 应用的数据流分析工具的有效性. 具体而言, 它可用于评估动态和静态的数据流分析效果, 尤其是它包含一些特定分析问题的测试用例, 例如字段灵敏度、回调机制、反射技术以及组件间通信等.

AMD 数据集由 Wei 等 [43] 在 2017 年构建并发布. 该数据集包含 71 个家族共计 24553 个恶意样

13) Contagio Mobile-mobile malware mini dump. 2019. http://contagiominidump.blogspot.com/.
14) VirusShare. 2019. http://virusshare.com/.
15) DroidBench Benchmarks. 2013. https://github.com/secure-software-engineering/DroidBench.

表 2 测试数据集相关信息

Table 2 Descriptions of datasets

Dataset	#Sample	#Family	Average file size (MB)	Time
Genome dataset [8]	1260	49	1.3	2011∼2012
Drebin dataset [15]	5560	179	1.3	2011∼2014
FalDroid dataset [26]	8407	36	1.9	2013∼2014
DroidBench dataset	119	–	0.2	2014∼2016
AMD dataset [43]	24553	71	2.1	2010∼2016
RmvDroid dataset [44]	9133	56	4.8	2014∼2018

本, 样本分布时间为 2010∼2016 年. 相比于之前的数据集, Wei 等通过人工分析每个家族的恶意行为, 在数据集中补充了各个家族的所属类型、出现时间、检测时间以及其安装方式等具体信息.

RmvDroid 数据集由 Wang 等 [44] 在 2019 年构建并发布. 该数据集包含 56 个家族共计 9133 个恶意样本, 主要是由 Google Play 上的下架样本组成, 样本分布时间为 2014∼2018 年. 该数据集不仅仅包含家族以及恶意样本信息, 还包含 app 相关的描述、评级、下载次数、所在类别等信息.

表 2 列出了上述 4 个常用数据集的相关信息, 包括样本数量、家族数量、平均样本大小以及样本收集时间. 值得注意的是 DroidBench 数据集中样本主要是用于测试不同特定行为, 因此不包含复杂的应用功能, 样本平均大小仅为 0.2 MB.

除了上述常用数据集之外, 许多研究人员根据相应需求构建数据集 [45, 46]. 但是数据集的构建是一件非常困难的事情, 除了样本的收集外, 还需要额外的样本家族信息标注工作 [47]. 但是现如今各个数据集的标注方法存在较大差异. 具体而言, Genome 数据集是通过人工分析进行家族信息标注, 其可信度较高. 但是由于人工分析恶意样本行为是一件非常耗时费力的工作, 因此不适合大量样本数据集的标定. Drebin 数据集则是分析 VirusTotal 中 10 个反病毒引擎的检测结果, 如果超过 2 个引擎结果表明其为恶意, 则将其添加入数据集中, 并用引擎返回的标签作为其家族信息. 其中 VirusTotal[16] 是一个集成了 53 个反病毒引擎 (例如 AVL[17], McAfee[18] 以及 ESET-NOD32[19] 等) 的系统网站. 类似地, FalDroid 数据集与 AMD 数据集也是基于 VirusTotal 的返回结果对恶意软件家族信息进行标定, 二者方法较为类似, 即采取投票策略, 如果一半以上的反病毒引擎给定的家族标签一致, 那么就将该标签赋予相应样本. 与上述数据集标注方式不同的是, RmvDroid 数据集首先通过收集不同时期 Google Play 上的应用, 并筛选出下架的 app, 然后结合 VirusTotal 的检测结果来标注该 app 是否为恶意样本, 最后利用 AVClass 方法 [47] 对其家族信息进行标定. 下面我们以 FalDroid 数据集构建为例, 介绍一下数据标注的基本思路, 其主要包含 3 个步骤.

(1) 首先从 VirusShare 上下载了大约 15000 个恶意样本. 通过工具 apktool [48] 对其进行反编译操作, 并检查其 smali 文件夹, 查看是否能够获取其有效的 Dalvik 代码. 值得注意的是, 有些恶意样本可能被加固, 因此其真实 DEX 文件被隐藏, apktool 无法正常进行反编译, 此类加固样本不在我们的数据集范围中. 针对该类型样本现如今有一些专门进行脱壳分析的研究工作, 典型的有 PackerGrind [49], DexHunter [50], DroidUnpack [51] 等.

16) VirusTotal. 2019. https://www.virustotal.com.
17) AVL. 2019. https://www.avlsec.com/.
18) McAfee. 2019. http://us.mcafeestore.com/.
19) ESET-NOD32. 2019. https://www.eset.com/us/home/antivirus/.

表 3 家族标签字典部分信息列表

Table 3 Part of the family label dictionary

Family label	Other similar labels
basebridge	bridge
droiddreamlight	ddlight/lightdd/drdlightd/
droidkungfu	kungf/gongf/droidkungf/droidkungfu2
fakeinst	fakeinstall/fakeins
plankton	planktonc/plangton
geinimi	geinim/geinimia/geinimix

(2) 将每一个样本上传至 VirusTotal. 对于 VirusTotal 中每一个反病毒引擎而言, 它都会对待检测的样本进行分析并给出一个相应的家族标签. 但是我们发现这些反病毒引擎存在两个问题. 首先, 不同的反病毒引擎标定的家族信息命名往往不同 (例如, plankton/plangton/planktonc 等), 这主要是由于不同反病毒引擎具有各自的命名规则; 其次, 标定的家族信息往往不一致 (例如, plankton/adrd 等), 这主要是因为不同的引擎其病毒库不同, 会存在误报和漏报.

(3) 为了解决上述两个问题, 我们首先基于字符串编辑距离 [52] 构造了一个家族标签字典, 如表 3 所示, 然后将相近的家族标签归为一类并选择一个代表性标签作为最终的家族标签. 最后采取投票策略, 即如果一半以上的反病毒引擎给定的家族标签是一致的, 那么就将相应的家族标签赋予相应的样本.

现在使用的数据集其家族标注方式基本上均依赖于 VirusTotal 上的反病毒引擎结果, 采取不同的投票策略进行标注. 但是, 我们认为它们还存在不足之处, 即一个真实可靠的数据集还应包括恶意样本的详细恶意行为信息. 例如, 当某一个样本被标注为 geinimi 家族成员, 其判断依据是什么, 所对应的恶意代码位于哪个部分. 如果现有的数据集能对该类型信息进行补充和完善, 那么将对未来的恶意软件检测领域起到很好的促进作用.

4 基于特征码的恶意软件检测方法

基于特征码的恶意软件检测方法基本原理是利用每个软件特有的特征信息进行匹配, 即在已知恶意软件指定特征码的情况下, 通过与目标待检测软件的特征码进行匹配, 如果在已有的恶意软件特征码数据库中找到相同的特征码则将目标软件判定为恶意软件, 否则为良性软件. 对于该类方法而言, 已有的恶意软件特征码数据库是关键. 现有的特征码可以根据其是否包含程序语义信息分为两类: 传统特征码 [53~56] 以及语义特征码 [5, 16].

4.1 传统特征码

最早期的特征码即每个 Android 应用的数字签名. 根据 Android 系统相关要求, 每一个应用软件必须拥有有效的数字签名才能被上传至应用市场. 每一个数字签名唯一标识该应用, 因此最原始的恶意软件检测方法是将待检测软件的数字签名与已知恶意软件的数字签名进行匹配, 判断其是否在已知数字签名的数据库中. 但是通过重打包技术可以对程序进行修改并重新签名, 因此该检测方法非常容易被绕过.

Enck 等 [53] 在 2009 年提出了一种基于权限使用规则的轻量级检测方法 Kirin. Kirin 构造了 9 条理

想安全权限规则, 如果待检测软件所申请的权限违反了其中任意一条规则, 则将该软件判别为恶意软件并拒绝安装该软件. 例如, 其中一条安全规则为: 不能同时申请权限 SEND_SMS 以及 WRITE_SMS. 该方法在一定程度上可以识别恶意软件, 但是存在很高的误报率, 容易将需要使用发短信功能的良性应用判别为恶意软件. 除此之外, 通过修改配置文件 AndroidManifest.xml 中的权限申请信息则可轻松绕过该检测. 类似地, Zhou 等 [54] 提出了一种基于权限脚本的恶意软件检测方法 DroidRanger. 该方法在权限使用规则的基础上额外添加了 API 信息以及关键字符串等信息. 除此之外, 该方法还结合了远程代码的加载情况以及软件的动态执行信息, 从而通过动静结合的方法使得检测效果有所提升.

除了将权限使用规则当做特征码, Seo 等 [55] 提出的 DroidAnalyzer, 通过分析恶意软件中常用的 API 函数 (例如获取电话号码的 getLine1Number()) 以及提升权限相关命令 (例如重启设备的 reboot 命令), 将它们作为特征码进行匹配检测. Zheng 等 [56] 提出了一种基于 3 层不同粒度的特征码检测方法 DroidAnalytics, 第 1 层特征码为每个方法体中 API 调用序列的哈希 (Hash) 值, 第 2 层特征码为在第 1 层特征码的基础上构建的每个类的相应哈希, 第 3 层特征码为在第 2 层特征码的基础上构建的每一个软件应用的相应哈希. 通过不同粒度的特征码表示, 该方法可以有效针对不同类型的恶意代码, 例如方法级别、包级别以及 APK 级别.

4.2 语义特征码

早期的特征码不包含程序语义, 因此容易被现有混淆技术修改从而绕过检测. 后续工作开始尝试将程序语义信息融入到特征码中, Feng 等 [5] 在 2014 年提出了一种基于语义特征码的恶意软件检测方法 Apposcopy. 与上述方法不同的是, Apposcopy 的特征码包含 3 类信息: 组件类型信息、控制流信息以及数据流信息. 除此之外, 该方法使用一种高级描述语言来描述各个恶意家族的特征码信息, 例如, golddream 家族具有特征码 flow(s, DeviceId, s, Internet), 表示该类家族中的样本包含将设备 ID 信息发送至网络中的数据流. 然而, 该方法需要依赖安全专家仔细分析每一个家族中的样本并总结恶意行为的特征码, 该过程需要大量人力物力. 并且, 该方法构建的家族特征码仅仅对已知家族有效, 无法检测未知家族样本.

在 2017 年, Feng 等 [16] 对 Apposcopy 方法做了进一步改进, 提出了 Astroid. 该方法首先通过分析软件中的控制流信息以及数据流信息构建组件间调用图 ICCG (inter-component call graph). 然后对该图进行形式化表达, 将特征码表示成合取范式. 针对同一个家族中样本的 ICCG, 分析其相应的合取范式并挖掘出最大公共子图, 用其表示该家族的共性行为. 针对待检测恶意样本, 通过子图匹配方法判断该样本是否包含某个家族的子图特征码. 尽管该方法能够自动从家族样本中构建其子图特征码, 但是还是需要一定的人工参与, 即如何从家族中挑选用于构建特征码的代表性样本.

以上方法没有对程序的 native 代码进行分析, 因此对于将恶意行为隐藏在 native 代码中的恶意软件会存在漏报情况. 根据文献 [57,58] 结果显示, 市场上 37% 的应用程序以及 86% 的流行应用程序均包含 native 代码. 为了解决 native 代码对于恶意软件检测的局限性, Alam 等 [59] 提出了 DroidNative, 一种支持跨平台 (支持 x86 与 ARM 两种架构) 分析的恶意软件 native 恶意代码检测方法. 该方法首先解析 native 代码然后将其用一种中间语言 (MAIL [60]) 进行描述, 并构建相关行为模式. 基于该中间语言, DroidNative 在控制流图 CFG 的基础上将每一条状态语句用中间语言进行描述并构建了带注释的控制流图 ACFG. 然后, 通过在 ACFG 上使用滑动窗口将其进行切分并构建特征码用于恶意软件检测.

综上所述, 传统的基于权限或者 API 特征码的方法检测分析速度快, 但是该类方法容易被攻击者通过混淆技术改变软件语法结构, 从而使得特征码发生改变. 并且, 此类方法虽然可以通过与已有特征

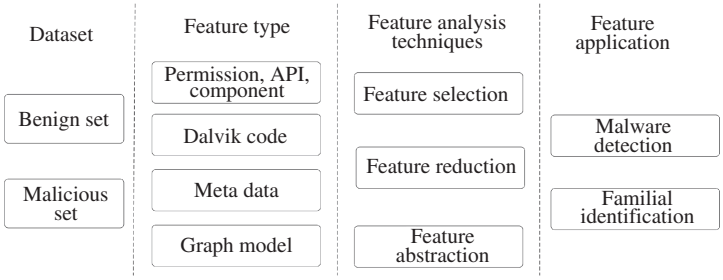

图 1　基于机器学习的恶意软件检测方法框架图

Figure 1　Overview of the machine learning-based malware detection methods

码的精确匹配而达到恶意检测的目的, 但是无法处理未知恶意软件. 包含语义信息的特征码能够在一定程度上抵抗混淆技术, 但是该类特征码需要一定程度上的人工分析, 因此自动构建语义特征码是未来可行的研究方向.

5　基于机器学习的恶意软件检测方法

基于机器学习的恶意软件检测方法的基本原理是通过程序分析等技术提取不同的特征描述待分析样本的不同行为, 然后每一个样本均用一个固定维度向量表示, 最后借助于现有的机器学习算法对已知标签的样本进行训练并构建分类器, 从而能够对未知样本进行预测判断. 本节从特征工程角度对已有方法进行回顾, 包括 3 方面: 特征类型、特征分析技术以及特征应用. 如图 1 所示, 特征类型主要可分为 4 种, 权限、API、组件等基本元素, Dalvik 字节码, 元数据, 以及包含程序语义的各种图模型; 特征分析技术主要可分为特征选择、特征降维以及特征抽象 3 种; 特征应用主要可分为恶意软件检测以及恶意家族识别两种.

5.1　特征类型

5.1.1　权限、API 以及组件

基本元素特征主要包括权限、API、组件以及 Intent. 权限是一种安全机制, 主要用于限制应用程序内部某些具有限制性特性的功能使用以及应用程序之间的组件访问. 例如, 如果一个程序要使用网络, 则其需要申请网络相关权限 INTERNET. Android API 是 Android 系统预先定义好的函数, 或指系统不同组成部分衔接的约定. 主要提供应用程序与开发人员基于软件访问一组例程的能力, 无需访问源码或理解内部工作细节. API 相比于权限而言是一种粒度更细的特征. 应用组件是 Android 应用的基本构建块. 每个组件都是一个入口点, 系统或用户可通过入口点进入应用. Android 系统共有 4 种应用组件类型, Activity, 服务 (service), 内容提供者 (content provider) 以及广播接收器 (broadcast receiver). Intent 则是一个非常重要且常用的类, 可以用来在一个组件中启动当前程序中的另一个组件或者是启动另一个程序的组件.

文献 [19, 61~65] 通过对 Android 应用程序配置文件 AndroidManifest.xml 的分析, 提取出权限申请信息, 并将每一个权限当做特征. Au 等 [14] 提出的 PScout 通过静态分析 Android 系统源码, 抽取 API 函数及其所需权限的映射关系, 从而得到各个 API 函数的权限规范. Aafer 等 [66] 提出了 DroidAPIMiner, 通过静态分析提取了 5 种不同类型 API 作为特征, 包括应用相关 API、Android 框架 API、DVM (Dalvik virtual machine) 相关 API、Linux 系统调用 API 以及工具 API. 与静态 API 特征

不同的是, 文献 [67,68] 通过动态执行程序记录其执行的系统调用 API 作为特征. 除了分别使用权限以及 API 作为特征, 文献 [13,69,70] 将权限特征与 API 特征组合在一起, 进一步提升检测效果.

除了经典的权限以及 API 特征之外, 许多研究工作 [15,71~75] 发现 Android 的相关 Intent、字符串以及组件也可以作为恶意软件的有效特征. 典型工作为 Arp 等 [15] 提出的 Drebin, 在权限和 API 特征的基础上, 该方法将 Intent 信息 (例如重启手机后触发恶意行为的 BOOT_COMPLETED)、硬件组件信息 (例如 GPS 与网络模块)、Android 4 大组件信息以及网络地址信息 (例如 IP 地址、主机名以及 URL) 作为特征. Garcia 等 [76] 提出了 RevealDroid, 在上述特征的基础上对反射函数以及 native 代码做了分析, 将反射函数出现的频次以及 native 调用函数作为特征, 从而增强了该方法对混淆技术的弹性能力.

5.1.2 Dalvik 字节码

Android 程序通常是由 java 语言编写, 通过编译成 Dalvik 代码 (DEX) 存储在命名为 classes.dex 的文件中. 通过反编译 APK 文件可获得相关的 Dalvik 字节码, 它存储在 smali 文件中, 与 java 中的类一一对应, 它具有自己的一套指令集, 与汇编语言类似. 一条 Dalvik 指令包含相应的操作码 (opcode) 以及操作数 (operand). 除此之外, 将一个函数内部的 Dalvik 指令根据其执行顺序关系可划分为基本块. 具体而言, 基本块指的是程序一一顺序执行的语句序列, 其中只有一个入口和一个出口, 入口就是其中第 1 个语句, 出口就是其中的最后一个语句. 对基本块而言, 执行时只从其入口进入, 从出口推出, 不包含跳转语句.

在字节码特征中, 最为常见的是将 k-gram 技术 [77] 应用到操作码序列上, 通过步长为 k 的滑动窗口将操作码序列划分为以 k 个操作码为单位组成的特征, 从而提高特征的鲁棒性 [78,79]. 但是该类方法随着 k 值的增大, 其特征个数呈指数型增长. 一般而言, k 的取值范围会在 1 到 5 之间, 并且在不同的数据集上, 其特征空间会存在较大差异. 而以基本块为特征的典型方法为 Suarez-Tangil 等 [80] 提出的 Dendroid, 首先将每一个样本表示为基本块集合, 然后设计了一种语法对基本块元素进行表征并构建特征向量实现家族识别.

5.1.3 元数据

元数据指的是与 app 应用程序本身代码无关的额外描述信息, 例如该 app 的下载量、功能描述、类别信息等. 该类信息可以从另外一个新的角度对现有的特征进行完善, 从而提升检测效果.

Teufl 等 [81] 抽取 app 的多种元数据作为特征, 包括最后一次修改时间、价格、描述信息、类别、下载量、用户评级、安装包大小、截图数量、包名信息、版本号、开发者 ID 以及其联系方式等. Grampurohit [82] 与 Wang 等 [83] 将应用的类别信息作为特征与已有的权限和 API 特征进行组合, 从而对检测效果有一定程度上的提升. 与直接使用 app 默认类别信息不同, Gorla 等 [84] 提出了 CHABADA, 首先通过对应用的描述信息进行聚类分析, 重新将应用划分为 29 个类别. 他们认为如果一个天气类别应用调用了 API 函数 getLastKnownLocation() 获取位置信息则属于正常现象, 而其他无关类别的应用程序调用该函数则可能会被当做异常应用. 该方法有效地将描述信息与软件行为进行结合, 从而在各个类别中挖掘异常应用程序. 我们在 2019 年提出了自动特征工程方法 CTDroid [85], 与已有方法不同的是, 它能够从大量恶意软件相关的技术博客中抽取出敏感行为特征, 表示为动宾短语形式, 从而与程序行为相匹配. 该方法利用已有的知识库实现了行为特征的自动构建, 大大减少了特征工程的时间开销.

表 4 基于图模型特征的方法列表

Table 4 Descriptions of the graph feature-based methods

Graph model	Typical method	Node type	Granularity
Function call graph	Adagio [86], MaMaDroid [17], MIGDroid [87], DAPASA [18], FalDroid [26]	Function name	Medium
Control flow graph	Centroid [3], GroupDroid [88], ADAM [89], SMART [90]	Basic block	Fine
Data flow graph	DNADroid [91], PVCS [92]	Statement	Fine
UI graph	ViewDroid [93], ResDroid [94], MassVet [10], SmartDroid [95]	View	Coarse
Package dependency graph	PiggyApp [96]	Package name	Coarse
Class dependency graph	DR-Droid [97], Droidlegacy [98]	Class name	Coarse
API dependency graph	DroidSIFT [28]	API	Medium
Heterogeneous information network	HinDroid [99]	API, app	Medium

5.1.4 图模型

上述 3 种类型特征均以字符串形式存在, 该形式容易被现有的混淆技术所篡改, 从而绕过恶意软件检测. 因此, 越来越多的研究工作在保留程序语义的基础上为了提高特征的鲁棒性, 采用了图模型方式, 也就是将程序相关语义用各种图模型进行表示. 常用的图模型可以按照其节点属性分为函数调用图 FCG (function call graph)、控制流图 CFG (control flow graph)、数据流图 DFG (data flow graph) 以及用户接口交互图 UIG (user interface graph). 除此以外, 还有一些根据不同需求自定义的图模型. 表 4 [3, 10, 17, 18, 26, 28, 86~99] 列出了不同的图模型及相关信息, 其中最后一列表示分析粒度, 根据分析对象的不同分为 3 个等级: 粗、中以及细.

粒度较细的主要是以程序语句为分析对象的图模型. 典型工作为 Centroid [3] 中构建的控制流图, 图中节点表示基本块, 边表示基本块之间的跳转关系. DNADroid [91] 则构建数据流图进行细粒度分析, 图中节点表示带有数据信息的程序语句, 而边则表示语句之间的数据依赖关系.

粒度适中的主要为以函数为分析对象的图模型. 典型工作为 Adagio [86] 中构建的函数调用图, 图中节点表示函数, 边表示函数之间的调用关系. 在函数调用图基础上, DroidSIFT [28] 仅分析其中的 API 调用, 构建了 API 依赖图, 图中节点表示 API 函数, 边表示 API 函数之间的依赖关系. 而我们则特别标识了函数调用图中的敏感 API 节点, 并提出了基于敏感子图分析的恶意软件检测方法 DAPASA [18] 与恶意家族识别方法 FalDroid [26]. 敏感 API 函数主要指操作敏感数据的相关 API 函数, 例如返回手机号码的 getLine1Number() 以及发送短信消息的 SendTextMessage(). 上述图模型均属于同构信息网络, 即图中节点均为函数或 API, 而 HinDroid [99] 构建了异构信息网络, 图中节点分为两类, API 以及 app 应用, 边根据 4 种不同情况进行构建: (1) app 调用某个 API; (2) 两个 API 隶属于同一个包; (3) 两个 API 存在于同一个基本块中; (4) 两个 API 使用相同的调用方法.

粒度较粗的主要是以包、类以及视图元素等作为分析对象的图模型. 典型工作为 PiggyApp [96] 中构建的包依赖图, 图中节点为程序中的包名, 边为包之间的依赖关系, 主要分为 4 种: 类继承关系、同源包关系、方法调用关系以及成员字段引用关系. 类似地, Droidlegacy [98] 中构建了类依赖图, 图中节点为程序中定义的类名, 而边则为类之间的依赖关系, 也就是上述 4 种关系中除了同源包关系

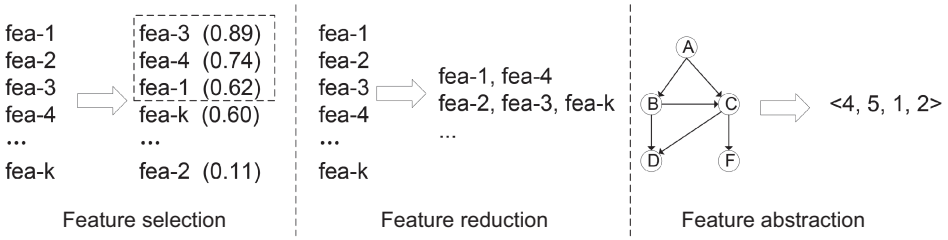

图 2 3 种常用特征分析技术原理示意图

Figure 2 Overview of three feature analysis techniques

的其余 3 种. 通过分析用户交互界面, MassVet [10] 构建了用户接口交互图, 图中节点主要为 activity 组件以及不同类型的对话框, 边则表示不同类型的事件消息, 例如点击事件 onClick()、焦点改变事件 onFocusChange() 以及触屏事件 onTouch().

总体来说, 粒度越细, 包含的语义信息越丰富, 但是反之其模型构建时间开销就越久, 并且后续图分析复杂度也越高. 因此, 针对不同的应用需求应构建不同粒度的图模型.

5.2 特征分析技术

在经过特征抽取后, 针对不同的需求, 例如减少特征数量、提高检测精度以及加快处理速度等, 往往需要对特征进行进一步的分析. 常用的分析技术可以分为 3 大类: (1) 特征选择, 通过选取已有属性的子集进行建模的一种方式. 进行特征选择的主要目的有: 简化模型, 缩短训练时间, 避免维数灾难, 增强模型泛化能力等. (2) 特征降维, 通过对原有的特征进行重新组合, 形成新的特征, 选择其中的主成分进行分析. (3) 特征抽象, 在保留原有特征程序语义的基础上对特征做进一步分析, 构建新的特征. 图 2 展示了 3 种方式的基本原理.

5.2.1 特征选择

特征选择常用方法为特征排序, 给每一个特征赋予相应权重然后选择权重最高的特征子集. Wang 等 [19] 提出了基于权限特征的恶意检测方法, 通过 3 种特征排序方法, 即互信息、皮尔森 (Pearson) 相关系数以及 T 检验, 对每一个权限赋予相应的权重, 从而在权限集合中选择权重最高的 40 个权限作为特征子集. 类似地, Wang 等 [74] 提出的多特征检测方法由于考虑了组件名、URL 以及 IP 地址等信息, 其特征维数达到了 230 万维. 如此高维度的特征向量可能会造成维数灾难以及过拟合等问题, 因此他们也采取了 3 种特征排序方法, 即互信息、卡方检验以及单向方差分析, 选择了 14000 个特征用于分类模型的构建. Hu 等 [87] 提出的 MIGDroid 通过统计分析 API 在恶意数据集中出现的频次, 根据频次的高低, 人工赋予相应的权重, 表示其敏感系数. 然而, 由于存在一些 API 在良性样本和恶意样本均被高频使用的情况, 我们提出了 DAPASA [18], 结合 API 在良性数据集与恶意数据集中分别出现的频次, 并提出了一种 TF-IDF-like 的方法给敏感 API 赋予了不同的权重, 表示其相对于不同类别应用的敏感性系数.

5.2.2 特征降维

常用方法为关联分析, 通过挖掘特征之间的潜在关系组成新的特征. Moonsamy 等 [62] 通过关联分析对权限特征进行重新组合, 形成新的特征, 每一个新的特征均由若干个权限组合而成. 该方法还设置了最小支持度阈值 0.05, 即仅保留被至少 5% 样本所使用的权限组合特征. Avdiienko 等 [2] 提出

的 MUDFLOW 通过数据流分析挖掘 API 之间的数据传递关系, 将 API 重新组合为 source-sink 形式的新特征, 其中 source 表示获取敏感数据的 API 调用, 例如获取用户号码的 getLine1Number(), sink 表示将敏感数据作为参数的 API 调用, 例如用于发送短信的 sendTextMessage() [100].

5.2.3 特征抽象

常用的特征抽象方式为哈希处理以及图分析技术. 由于目前应用市场上的应用规模已经达到了百万级别, 因此对特征之间的相似性计算带来了较高的效率要求. Juxtapp [101] 对操作码特征进行了哈希处理, 大大降低了程序之间的相似性计算开销. 常用的哈希算法包括 MD5, SHA1, SHA256 等. 然而, 普通的哈希操作面临一个局限性, 即可能会由于个别操作码的修改而导致整个哈希结果的巨大差异. DroidMOSS [102] 对操作码特征采用模糊哈希算法, 使得哈希结果对于个别操作码改动仍然具有高度相似性. 模糊哈希算法又叫基于内容分割的分片哈希算法, 其对文件的部分变化 (包括多处修改、增加、删除部分内容) 仍然能够发现与源文件的相似关系.

相比于字符串类型特征, 图模型尽管能够有效地保留程序的语义信息, 但其复杂度则大大高于字符串类型特征. 例如, 常用的函数调用图中往往包含成千上万个节点, 直接地去分析整个图的时间开销是非常大的. 因此, 针对图模型特征, 图分析是非常重要的步骤.

为了减小待分析图的规模, 一些研究工作 [26,28,96,98] 将复杂的初始原图划分为子图集合. PiggyApp [96] 通过对包依赖图中的节点进行聚类操作, 从而将待分析样本的所有包划分至不同的模块中. 聚类条件是如果两个包节点之间存在紧密关系, 即包内代码存在较多的引用关系, 则将两个包划分至同一个模块中. Droidlegacy [98] 划分算法与 PiggyApp 类似, 区别在于其节点表示类, 粒度相对包而言更细. DroidSIFT [28] 则是通过分析 Android 程序的入口点, 例如 onClick() API 调用, 从而在功能上将 API 依赖图划分为不同的子图, 每一个子图完成特定的功能. 因此, DroidSIFT 相比于前者其保留的语义信息更丰富. 类似地, 我们提出的 FalDroid [26] 发现函数调用图也具有显著的社团结构特征, 即位于同一个社团中的函数节点具有较强的关联性并且常常位于同一个包中, 从而共同完成一个程序功能. 因此我们基于现有的社团划分算法将函数调用图划分为子图集合, 然后仅仅保留其中至少包含一个敏感 API 节点的子图, 并将其定义为敏感子图. 敏感子图的构建大大减少了图分析复杂度.

还有一些工作 [3,10,86,94] 将图模型抽象为哈希值或者向量等低维表示形式. Adagio [86] 首先统计每一个函数内部常用的 15 种操作符的出现频次, 然后基于该频次用 15 维向量表示该函数. 接下来, 通过计算每一个函数与其邻接函数之间向量操作后的哈希值来唯一标识该函数节点, 从而达到了降低图分析复杂度的目的. 类似地, ResDroid [94] 对于每一个 activity 的布局视图, 采用先序遍历算法将其转换为元素序列形式, 然后将视图对象使用预先对应的英语字母进行替换, 从而将图模型转换为字母序列形式. 然而上述工作在降低图复杂度的同时损失了图中的结构信息, 为了弥补该缺陷, Chen 等 [3,10] 提出了 Centroid, 将控制流图 CFG 用 3D 形式进行表示, 每一个节点在三维坐标系上用 $\langle x, y, z\rangle$ 唯一标识, 其中 x 表示该节点在所有节点中的序列号, y 表示节点的出度, z 表示节点的循环深度. 然后, 结合每个节点内部语句的数量, 通过加权平均计算该图的质心. 但是该方法的缺陷是节点数较少的 CFG 其质心表示太相近. 近两年, 随着深度学习的快速发展, 许多工作将其应用到图分析上. Narayanan 等 [103] 提出了 subgraph2vec, 基于图嵌入技术将 CFG 抽象为固定维度向量, 从而将高复杂度的图匹配问题转换为低复杂度的向量相似性计算问题. 类似地, 我们也将图嵌入技术应用到无监督的 Android 恶意家族分析中并提出了 GefDroid [104], 首先基于 struc2vec 技术 [105] 将敏感子图抽象为固定维度向量, 然后基于该向量构建了 SRA 特征, 用于描述敏感子图中敏感 API 节点的结构相似性关系. 该特征在保留敏感子图中敏感信息的同时进一步提高了子图匹配的速度.

5.2.4 特征应用

基于机器学习的方法主要应用在两个场景上, 即恶意软件检测与恶意家族识别. 恶意软件检测即给定一个待分析样本, 判断其是否具有恶意行为, 属于二分类问题. 恶意家族识别即对于已经检测的恶意软件如何将其划分至相应的家族当中, 属于多分类问题.

恶意软件检测. 现如今大部分的方法主要应用于恶意软件检测场景上, 其数据输入包含恶意样本数据集与良性样本数据集. 该类方法常常将数据集划分为两部分, 训练集和测试集. 训练集中的数据用于学习并构建分类器, 而测试集中的数据则用于检验方法的效果. 对于训练集中的每一个样本, 通过特征抽取, 将其表示成固定维度的特征向量 $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_k \rangle$, 其中 k 表示向量维数, 也就是特征数. 对于训练集中的样本数据由于已知其是否恶意, 因此在其向量后添加相应的标签信息, 例如用标签 0 表示该样本为良性样本, 而用标签 1 表示该样本为恶意样本. 然后借助于已有的机器学习算法, 例如支持向量机、随机森林等, 对训练样本的特征空间进行学习并构建相应的分类器. 对于测试数据中的样本, 将其特征向量输入至分类器中, 返回的标签结果表示其是否具有恶意行为. 对于恶意软件检测, 常用的度量指标为真正率 TPR (true positive rate)、假正率 FPR (false positive rate)、召回率 Recall、准确率 Precision、F-measure 值等. 每一个恶意软件检测方法的主要目的是为了获得较高的 TPR 以及较低的 FPR.

恶意家族识别. 与恶意软件检测不同的是, 恶意家族识别中仅需要恶意数据集作为训练集, 不需要良性数据集. 除此之外, 恶意数据集中的样本需要包含其相应的家族信息标签. 与恶意软件检测类似, 每一个家族中的样本被划分为训练集和测试集, 但是不同的是, 对于训练集中的样本, 其向量后添加的是家族标签信息, 通常由英文单词表示, 例如 adrd, geinimi 以及 droidkungfu 等. 然后通过构建分类器并输入测试样本的特征向量, 将返回的标签结果表示测试样本的相应家族信息. 对于恶意家族识别, 常用的度量指标为识别准确度, 表示被正确划分至相应家族中样本的百分比.

表 5 列出了典型的基于机器学习方法的结果, 其中 MD 表示恶意软件检测应用场景, FC 表示家族识别应用场景, #B 表示良性样本数量, #M 表示恶意样本数量, ACC 表示家族识别准确度. 从表中可以看出, 随着时间推移, 各种检测方法中所使用的样本数量逐渐增加, 从最开始的几百到现在的十万级别. 前面提到, 恶意样本的标注是一件相对耗时费力的工作, 但是目前的工作大部分依赖于大量样本的有监督学习才能训练出较好的模型, 因此如何基于小样本训练集实现半监督学习是目前基于机器学习的恶意检测方法的可研究方向之一. 除此之外, 从该表中还可以发现各个方法均可以达到较高的预测准确率. 然而, 基于机器学习的方法存在一个普遍的共性缺陷, 即可解释性差. 通过构建好的分类器可以有效地预测待检测样本是否为恶意样本, 但是该分类器对于安全人员而言是一个黑盒, 无法提供有效信息去解释该样本具体包含何种恶意行为, 可解释性分析是恶意软件检测未来的重点研究方向之一, 我们将在第 7.1 小节详细介绍.

6 基于行为的恶意软件检测方法

行为检测主要指针对特定恶意行为的检测方. 典型的恶意行为有隐私泄露、权限提升以及描述与实际行为不一致, 其描述如下.

- **隐私泄露.** 隐私泄露的原因主要可分为两类, 一类是由 Android 系统的漏洞所导致, 另一类是由 Android 应用漏洞所导致. Android 系统漏洞是由其本身框架缺陷所引起的, 影响范围较广, 能够对所有安装相应系统的智能手机造成危害. 例如 Android 系统的 WebView 漏洞主要是 JS 脚本对 Java 和

表 5 基于机器学习的恶意软件检测方法的结果统计

Table 5 Performance of existing machine learning-based methods[a)]

Method	Time	Task	#B	#M	Detection performance (%)
Puma [61]	2012	MD	1811	249	TPR = 91, FPR = 19
DroidMat [71]	2012	MD	1500	238	TPR = 87, FPR = 0.4
SCSdroid [106]	2013	MD	100	49	Precision = 95.97
DroidAPIMiner [66]	2013	MD	16000	3987	TPR = 99, FPR = 2
Adagio [86]	2013	MD	135792	12158	TPR = 89, PFR = 1
PVCS [92]	2014	MD	2436	1433	TPR = 96.52, FPR = 1
V.Grampurohit [82]	2014	MD	24335	1530	TPR = 91.8, FPR = 11.4
W.Wang [19]	2014	MD	310926	4868	TPR = 94.62, FPR = 0.6
Drebin [15]	2014	MD	123453	5560	TPR = 94, FPR = 1
Droidlegacy [98]	2014	MD/FC	48	1052	Precision = 97, ACC = 92.9
DroidSIFT [28]	2014	MD/FC	13500	2200	TPR = 98, FPR = 5.15, ACC = 93
Dendroid [80]	2014	FC	–	1260	ACC = 94.2
MUDFLOW [2]	2015	MD	2866	15338	TPR = 86.4, FPR = 18.7
AndroidTracker [107]	2015	MD	51179	4554	Precision = 90
K.Allix [108]	2016	MD	51800	1200	Precision = 94
SMART [90]	2016	MD	223170	5560	Precision = 97
DAPASA [18]	2017	MD	44921	2551	TPR = 95, FPR = 0.7
X.Wang [74]	2017	MD	166365	18363	TPR = 96, FPR = 0.06
MaMaDroid [17]	2017	MD	8500	35500	F-measure = 99
HinDroid [99]	2017	MD	15000	15000	TPR = 98.33, FPR = 0.87
FalDroid [26]	2018	FC	–	8407	ACC = 94.2
W.Wang [83]	2018	MD	107327	8701	Precison = 99.39
RevealDroid [76]	2018	MD/FC	24679	30203	Precision = 98, ACC = 95

a) MD denotes the malware detection task; FC denotes the familial identification task; #B denotes the number of benign samples; #M denotes the number of malicious samples; and ACC denotes the prediction accuracy of FC.

native 代码中的 API 的不安全调用所引起的. 攻击者可以制作恶意网页诱惑用户进行点击访问, 然后执行网页中恶意脚本, 触发漏洞并调用相关 API 窃取用户隐私信息. 由 Android 应用漏洞造成的隐私泄露相比于系统漏洞而言更常见, 主要是因为大部分开发者为了追求功能实现的高效性而忽略了应用的安全性, 也就是应用中对外暴露的组件缺乏合法性验证, 从而引起用户敏感信息的泄露.

• **权限提升.** Android 软件在开发过程中应该遵循最小权限原则, 即软件应当声明那些用到的权限, 不应声明与软件功能无关的权限, 否则会造成权限提升问题 [109]. 权限提升产生的原因主要是 Android 应用之间存在调用关系, 随着 Android 应用数量的增多以及功能的越来越丰富, 许多应用可以调用其他应用的功能. 例如在淘宝上购买商品然后调用微信进行支付. 但是该类行为会带来一定的安全隐患, 恶意应用会通过合法调用公开接口而获取敏感资源, 从而实现权限提升.

• **不一致行为.** 该问题特指软件的实际执行行为与其声明的行为不一致. 例如, 一个应用申请了 READ_CONTACTS 权限, 但是其相关描述中没有关于读取联系人方式相关的语句, 则该应用具有不一致行为. 造成该问题主要有两个原因: 首先一个程序往往由多人共同开发而成, 撰写应用描述的人员

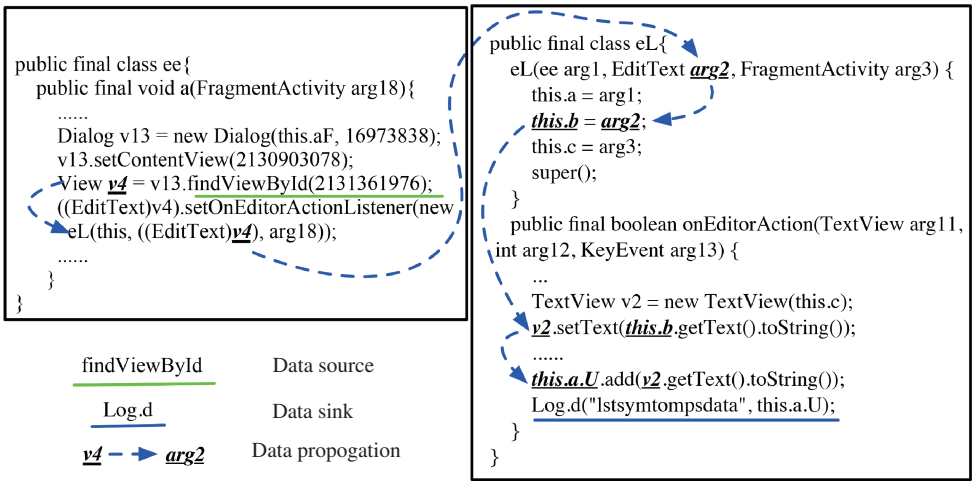

图 3　(网络版彩图) 数据流分析技术原理示意图

Figure 3　(Color online) An example of data flow analysis

可能没有参与完整的程序开发过程, 因此对程序功能理解不够清晰. 其次, 大部分 app 在开发时会嵌入第三方包, 而这些第三方包不包含源码, 因此开发者无法描述其具体行为.

针对上述 3 个特定问题, 其相应的主流分析技术分别为数据流分析技术、组件接口检测技术以及自然语言处理分析技术.

6.1　数据流分析技术

数据流分析技术主要通过确定敏感数据的来源以及追踪信息流, 判断敏感数据的最终去向. 图 3 展示了数据流分析技术的基本原理, 主要由 3 部分组成: (1) 数据产生点, 程序中需要追踪的敏感数据产生位置; (2) 数据传播, 根据特定规则跟踪数据在程序中的传播过程; (3) 数据检查点, 在程序中最终判断敏感数据是否到达检查位置. 在该图中, 敏感数据为 API 调用 findViewById() 的返回值, 通过数据传播过程, 敏感数据最终通过 API 调用 Log.d() 记录下来. 现有的数据流分析技术可以分为动态和静态两大类.

典型的动态数据流分析技术为 Enck 等 [110] 提出的 TaintDroid, 该方法对 Android 系统进行了特别定制, 在追踪隐私数据在系统中的传播过程中为每一个数据对象增加了额外的存储空间, 用于标定该数据的隐私属性, 如果隐私数据传播到预先设定的系统边界外后, 则认为有隐私数据泄露问题存在. 后续工作例如 Hornyack 等 [111] 提出的 AppFence 在 TaintDroid 上进行了改进, 不仅仅可以对隐私数据实现数据流追踪, 还可以根据用户自定义设定对隐私数据的传播进行拦截, 防止数据外泄. 但是动态数据流分析技术面临的主要问题是路径覆盖不全以及效率问题, 路径覆盖不全指的是难以将程序中所有的路径遍历执行, 因此会丢失信息. 除此之外, 动态分析由于需要实时执行程序, 因此其所需时间开销较大, 无法处理大规模数据集.

典型的静态数据流分析技术为 Arzt 等 [112] 提出的 FlowDroid, 该工具通过构建准确的 Android 生命周期模型, 使其能够有效处理 Android 程序中的回调函数, 并使用 IFDS 算法实现准确高效的数据流传播分析, 具有上下文敏感、流敏感、域敏感以及对象敏感等特性, 其准确性较高. 但是 FlowDroid 无法有效处理组件间的数据传递, 为了解决该缺陷, Octeau 等 [9] 在 FlowDroid 基础上提出了 Epicc, 一种能够有效分析组件间数据流传递的分析工具. 然而 Epicc 可以识别 Intent 的 action, category 以

及 data MIME type, 但是无法解析 data scheme. Kliber 等在 Epicc [9] 和 FlowDroid [112] 的基础上进行了整合并增强, 从而提出了 Android 信息流泄露检测工具 DidFail [113]. 利用前者识别字符串形式的 intent 的相应属性, 利用后者识别组件内的数据流信息. 因此 DidFail 能够有效分析应用之间或者单个应用的多个组件之间的潜在数据流信息. Li 等 [114] 提出了 IccTA, 利用 Epicc 与 IC3 [115] 对 intent 的相关字符串进行分析, 找到 intent 的目标, 然后利用 FlowDroid 进行数据流分析.

基于上述数据流分析技术, 典型的隐私泄露检测方法有 Nan 等 [24] 提出的 UIPicker 与 Huang 等 [116] 提出的 SUPOR. 二者均主要检测 Android 应用程序 UI 界面中的用户输入隐私数据是否可能被泄露. 首先通过解析 UI 界面相关文件识别用户输入数据, 并通过自然语言处理技术判断该输入数据是否为敏感数据, 如果是, 则将该数据作为数据产生点, 并追踪该数据流信息. 如果该数据最终被以明文形式或者不安全 SSL 形式保存或者发送出去, 则认为存在敏感数据泄露隐患.

6.2 组件接口检测技术

Felt 等 [117] 开发了静态分析工具 Stowaway, 用于检测 Android 软件中是否存在额外申请的不必要的权限. 该工具首先分析了被软件调用的 API 函数, 然后分别判断调用相应 API 所需要的权限, 并将该权限集合与配置文件中声明的权限集合进行比较, 判断哪些申请的权限是不必要的. 该工具在 940 个应用上进行实验发现大约三分之一的应用具有权限提升现象.

为了检测权限提升问题, 现有工作主要是基于组件接口分析的检测技术, 即通过分析程序中组件接口的调用安全性, 判断是否存在漏洞可能被恶意软件攻击. Chin 等 [118] 提出了 ComDroid, 通过分析反编译后的 Dalvik 代码, 追踪组件间数据的传递情况, 从而可以有效解析数据组件之间发送或接收的系统变量, 包括数据类型、标志位以及是否包含动作和其他数据等. 在 100 个应用上的实验结果表明 57 个应用的接口可能被其他恶意软件所调用. Lu 等 [119] 提出了 CHEX, 用于检测 Android 应用中权限提升问题. 该方法首先识别应用中可能被调用的接口, 然后基于识别的接口执行数据流分析, 判断其可达性. 根据判定结果, 计算哪些接口可能被恶意软件利用, 从而达到权限提升的目的. 该方法在 5486 个软件上的实验发现, 其中 254 个软件存在权限提升问题.

为了减少组件接口的暴露, Kantola 等 [120] 分析了 Android 系统中导致权限提升问题的原因并提出了一种修改方案, 从而降低恶意软件提升权限的可能性. 主要思想是为组件间通讯添加更加严格的限制条件, 从而减少暴露给恶意软件的可利用接口数量. 然而该方法由于对 Android 系统的组件间通讯机制进行了改动, 使得一些第三方软件无法兼容.

6.3 自然语言处理分析技术

行为不一致问题往往是软件代码行为与其相关描述行为不一致所引起的. 现如今分析的描述对象主要为两类, 应用程序功能描述以及隐私保护政策描述. 应用程序功能描述即用户在下载应用时可以直接看到的描述该应用主要功能的文本信息, 而隐私保护政策描述指的是开发者针对相关应用撰写的针对隐私数据如何收集、使用、公开等相关操作的声明. 检测该问题除了前面提到的数据流分析等程序分析技术, 还需要借助自然语言处理技术对文本内容进行解析, 包括词性标注、类型依赖解析、命名实体抽取等.

针对应用程序功能描述行为一致性问题, Pandita 等 [121] 提出了 WHYPER, 首先抽取程序中对于 READ_CONTACTS, READ_CALENDAR 以及 RECORD_AUDIO 3 种权限的申请情况. 然后借助于自然语言处理技术分析应用程序的功能描述, 并判断描述中是否包含读取通讯录, 读取日程表以及录音功能. 如果不包含, 则存在行为不一致问题. 在 581 个样本上的实验结果表明 WHYPER 可以达

到 82.8% 的识别准确率以及 81.5% 的召回率. 然而 WHYPER 仅考虑 3 种常用权限, 并且无法处理没有 API 相关联的权限, 因此 Qu 等 [12] 提出了 AutoCog, 对 WHYPER 的功能进行了一定增强. 该方法采用了一种基于学习的方法关联 API 与权限, 并且在描述分析部分采用了 ESA (explicit semantic analysis) 实现基于语义的匹配, 从而大大提高了行为匹配的精度. 在 1785 个应用中 AutoCog 的准确率和召回率分别为 92.6% 以及 92%. Yu 等 [122] 则对 AutoCog 进行了改进, 提出了 TAPVerifier 检查与权限相关的 API 或者 url 是否在代码中被真实调用, 如果没有, 则不需要进行警告. 该方法能够有效减少已有工作 59.4% 的误报率.

针对隐私保护政策描述行为一致性问题, Yu 等 [123,124] 提出了 PPChecker, 结合程序分析技术以及自然语言处理技术检测应用程序相应的隐私保护政策的 3 个问题, 即完整性、准确性以及一致性. 该方法在 1197 个应用上发现, 282 (23.6%) 个应用程序至少存在一个问题. Slavin 等 [125] 则通过人工构建隐私保护政策的知识本体来检验行为的一致性, 该工作将行为不一致性划分为两个等级, 强不一致与弱不一致. 强不一致表示描述中完全不存在该行为而弱不一致则表示描述中存在相应行为的子行为. 该方法在 477 个应用中发现存在着 55 条强不一致行为以及 286 条弱不一致行为.

7 恶意软件检测未来研究方向

前文对目前主流的恶意软件检测方法进行了总结, 可以看出尽管现有的恶意软件检测相关研究工作已经取得了很大的进展, 但现有方法在性能上仍然存在很大的局限性, 不少技术还处于理论研究阶段, 难以在实际中得到推广应用. 除此之外, 移动平台上的新型威胁不断出现, 传统的恶意软件检测方法无法直接有效地对此类威胁进行检测. 本节对这些问题进行梳理, 并对未来的研究方向进行展望.

7.1 基于可解释深度学习的恶意软件检测

深度学习 (deep learning) 概念由 Hinton 等 [126] 于 2006 年提出, 它是机器学习中一种基于对数据进行表征学习的方法, 是一种能够模拟出人脑的神经结构的机器学习方法. 深度学习技术能够将特征学习融入到模型的建立过程中, 从而减少了人为设计特征所造成的不完备性, 它已经成熟地应用于计算机视觉、语音识别、记忆网络、自然语言处理等领域. 近两年, 不断有研究人员尝试将深度学习技术应用于恶意软件检测, 并取得了较好的检测结果.

在 Windows 平台上, Pascanu 等 [127] 抽取 API 事件当做特征并使用卷积神经网络 RNN 检测恶意软件. David 与 Netanyahu 提出了 Deepsign [128], 通过抽取 Windows 平台上可执行程序的动态 API 调用以及相关参数作为特征, 并结合深度信念网络检测其是否为恶意. Saxe 与 Berlin [129] 抽取可执行文件中的函数、字符串以及元数据等作为特征, 并使用深度神经网络 DNN 进行训练构建分类器来检测恶意软件.

在 Android 平台上, Yuan 等 [130] 提出了 DroidDetector, 首先抽取出 app 程序的 3 类特征, 即权限、敏感 API 以及动态执行行为. 然后, 使用深度信念网络对抽取的特征进行训练并用于恶意软件检测. McLaughlin 等 [131] 抽取 APK 文件反编译后的操作码作为特征, 结合 CNN 并用于恶意软件检测. Fereidooni 等 [132] 抽取了 Intent、权限、系统命令以及 API 调用等作为特征, 使用包括深度神经网络在内的 9 种分类算法进行模型构建并在大规模样本上实验验证, 结果表明深度学习算法效果优于传统算法. 相比于以上方法, Kim 等 [133] 则提出了一种多模式深度学习方法, 首先从 Android 配置文件、DEX 文件以及共享库中抽取出 7 类特征并构建 5 个向量, 然后针对每一个向量利用 DNN 构建相应的分类器, 最后将 5 个分类器的分类结果作为输入向量利用 DNN 构建最终的判别分类器.

尽管基于深度学习的恶意软件检测方法能够取得很好的检测率, 但是其无法真正在实际场景中应用, 主要原因是可解释性的缺失. 对于安全分析人员而言, 现如今的机器学习模型尤其是深度学习模型如同黑盒一样. 其主要功能就是给定一个样本输入, 模型反馈一个决策结果, 即该样本是恶意还是良性, 分析人员无法确切地知道模型背后的决策依据以及它所做出的决策依据是否可靠. 因此, 分析人员与决策模型之间存在着信任问题, 如何有效地提高深度学习模型的可解释性与透明性, 从而消除该模型在实际部署应用中的潜在威胁, 是未来恶意软件检测中的值得重点关注的方向之一.

近两年有不少深度学习解释模型被提出, 可参考 Ji 等 [41] 的关于模型可解释性相关综述论文. 现有的可解释性方法可分为全局解释和局部解释两大类. 全局解释主要是帮助人们从整体上理解模型背后的复杂逻辑和内部的工作机制, 而局部解释则是帮助人们理解学习模型针对每一个特定输入样本的决策过程和决策依据. 典型的全局解释方法有模型压缩, 通过将复杂模型学习的函数压缩成为更小, 更快, 解释性更强的模型. 例如 Tan 等 [134] 利用模型压缩的方法学习输入特征与负责模型预测之间的关系, 将其转化为可解释性更好的广义加模型. 典型的局部解释方法有局部近似, 在给定实例及其临域内利用可解释性模型对待解释模型的局部决策边界进行近似, 然后基于简单的可解释模型提供决策依据. 局部解释方法以 Ribeiro 等 [135] 提出的 LIME 为代表, 一种基于神经网络局部线性假设的模型无关可解释性方法. 对于每一个输入实例, LIME 首先通过该实例与其相近的实例训练出具有可解释性的线性回归模型, 基于该模型的权重系数对决策结果进行解释说明, 权重越大, 其相关特征对决策结果的重要性越高.

尽管不少可解释性方法被提出, 但是它们主要被应用在图像识别、文本分类等领域, 在恶意软件检测问题上具有怎样的效果还需要进一步探索. 例如, 对于全局解释方法而言, 如何保证压缩后的检测模型依旧具有较高的准确度. 对于局部解释方法而言, 给定一个恶意软件检测模型, 针对给定的输入样本, 不同的解释方法其解释结果是否一致, 如何去评估不同解释方法的有效性, 所提供的解释结果对于安全分析人员而言到底能够起到多大的帮助.

7.2 未知恶意软件检测问题

未知恶意软件主要可分为两类: 第 1 类是不在训练集当中的恶意样本. 基于机器学习的方法往往通过十折交叉验证方法评估其检测性能, 将恶意样本分为十份, 九份作为训练样本, 剩余一份作为测试样本, 而这一份测试样本则被认为是未知恶意软件. 但是, 这种情况下的未知恶意软件大部分为已知恶意软件的变种, 即训练集中已经包含其相同家族成员, 它们之间相似性较高, 因此针对此类未知恶意软件的检测效率较高. 但是我们认为这种情形下的检测率并不能真实反映现实恶意软件检测情况, 因为在这类检测场景中, 测试集的数据分布与训练集的数据分布高度相似, 训练好的分类器可能存在过拟合情形. 在真实场景中, 测试集的数据分布往往与训练集不同, 例如, 测试集中出现新型家族样本, 但是训练集中并不包含.

第 2 类则是市场上新出现的恶意软件, 即未知的恶意软件, 相比于训练集中的已知恶意软件而言, 该类恶意软件的出现时间较晚, 并完成新型的恶意行为. 针对该类未知恶意软件的检测存在概念漂移问题, 即通过已经训练好的模型去预测未出现的恶意软件, 值得注意的是恶意软件的行为会随着时间的推移发生改变. 针对恶意软件中的概念漂移问题已有部分研究成果, Mariconti 等 [17] 提出了 MaMaDroid, 通过构建马尔科夫 (Markov) 行为模型链抽取特征, 使用旧的恶意软件样本作为训练集预测新的恶意软件样本, 该方法针对出现时间间隔分别为 1 年与 2 年的未知恶意软件 F-measure 检测结果仅为 87% 与 73%. Jordaney 等 [40] 提出了 Transcend, 一种用于评估分类器性能以及过滤不可靠的分类结果的可调工具. 该工具可有效检测恶意软件检测模型中可能会出现的概念漂移现象.

总而言之, 现如今大部分方法主要是针对第 1 类未知恶意软件, 而第 2 类恶意软件的检测问题在实际应用中具有更重要的意义, 并且检测难度更大, 需要更深入的探索.

7.3 高对抗恶意软件检测问题

随着恶意软件检测技术的不断发展, 为了躲避现有技术的检测, 恶意软件开发者也对恶意软件对抗技术不断进行更新, 从而使得恶意软件存活时间延长, 实现利益最大化. 基于其实现原理, 我们将现有的对抗技术主要分为两类: 逆向对抗技术以及对抗样本生成技术.

逆向对抗技术又可分为静态对抗技术和动态对抗技术. 静态对抗技术主要是提高分析人员对 DEX 字节码逆向后的分析理解难度, 主要有代码混淆、程序加固、API 反射以及本地 native 代码执行等. 代码混淆技术主要是对程序中的符号信息进行混淆, 例如修改变量名、增加无效代码块等, 这样可以对基于 Dalvik 字节码的特征进行轻易修改, 从而增加检测难度. 程序加固通过修改 DEX 文件结构, 使得分析工具无法正常寻址, 程序代码段混乱, 并使得包括 smali, dex2jar 以及 androguard 等多种反汇编工具失效. API 反射采用基于反射的方法调用敏感 API, 这样使得基于 API 的静态行为特征不全从而影响检测结果. 本地 native 代码执行主要是通过将恶意代码隐藏在本地的 native 代码中, 使用 native 代码实现等价功能, 加大人工分析难度. 动态对抗技术主要是增加恶意行为在动态分析中的触发难度, 避免其行为被动态捕获, 主要有依赖条件的行为触发、模拟器检测等. 依赖条件的行为触发主要是选择特定的运行时间、用户行为、系统时间以及特定时间等各类条件进行组合才能触发恶意行为, 这样使得动态分析技术难以在规定时间内触发恶意行为. 模拟器检测是恶意软件在运行前判断其是否运行在模拟器中, 如果是, 则结束运行甚至卸载.

对于逆向对抗技术而言, 目前有一些特定的方法提出使得现如今的恶意软件检测技术具有一定的鲁棒性. 对于静态对抗技术, Li 等 [31] 提出的 DroidRA, 能够有效识别使用反射技术的敏感 API 调用. Garcia 等 [76] 提出的 RevalDroid 通过分析 native 代码将其相关函数调用作为特征. Xue 等 [32] 提出的 PackerGrind, 基于 Valgrind 分析工具[20] 检测恶意软件是否被加固, 如果是, 则进行脱壳处理. 对于动态对抗技术, Fratantonio 等 [136] 提出了 TriggerScope, 通过符号执行、路径预测等技术检测恶意软件中的特定触发器, 包含时间、位置等. 尽管这些技术能够在一定程度上增强恶意软件检测能力, 但是针对这些技术的新型逃避技术也会随之产生. 例如, 针对 PackerGrind 检测方法, 如果恶意样本检测其运行环境, 发现其具有 PackerGrind 相关特征, 则结束运行从而逃避检测.

对抗样本生成技术通过寻找机器学习模型的弱点, 产生能够欺骗模型的样本. 其最早应用是针对图像分类器, 通过在图像中注入噪声, 生成欺骗分类器的对抗样本, 可以使得分类器出错, 例如将女人识别为男人等. 后来, 恶意开发者通过将该项技术应用到恶意软件分类中, 通过修改部分特征, 在保证恶意行为不变的前提下欺骗已经构建好的分类器, 使得恶意样本被划分为良性样本, 从而绕过检测. 典型的工作有文献 [137~139], 其主要思想是基于前向梯度算法, 前向梯度是由神经网络的目标类别输出值对于每一个输入特征的偏导数组成的. 简单而言, 即通过迭代修改输入对象的特征向量 X, 直到达到扰动最大值或生成成功的对抗样本.

针对对抗样本的典型检测方法有模型蒸馏技术与重训练技术. 模型蒸馏技术通过从原来的 DNN 中蒸馏出来的类别概率知识, 加到一个小的 DNN 中, 从而在保持分类精度的同时提高模型泛化能力, 降低输入扰动对分类器模型的影响. 重训练技术即在模型训练中加入对抗样本, 从而使得训练出的模型具有更强的鲁棒性. 但是现有方法的缺点比较明显, 即只能对有限的对抗样本进行有效检测, 对于新生产的对抗样本精度较低.

20) Valgrind. 2019. http://www.valgrind.org/.

高对抗恶意软件的检测问题是典型的敌手环境问题, 它已经成为了现如今的研究热点. 但是现如今的检测方法主要是被动防御, 在攻击出现以后, 针对特定攻击问题提出相应的解决办法. 我们的检测方法不断地跟在安全威胁后面, 始终处于被动挨打局面. 在未来研究中, 我们如何变被动为主动, 控制攻击发生的根源, 是一个值得研究人员探索的问题.

7.4 新型威胁检测问题

移动恶意软件随着目的性的增强与攻击范围的扩大, 其技术手段与种类也更加的隐蔽和繁多. 移动平台上涌现出大量的新型安全威胁是未来可深入研究的热点问题, 例如广告内容安全、广告欺诈以及约会诈骗等.

广告内容安全问题主要指 app 内部与网络关联的链接相关内容可能具有恶意行为. 与传统恶意软件不同的是, 该类型 app 其本身功能可能为良性, 但是其内部代码中存在指向恶意页面的链接, 因此传统的恶意检测方法无法应对此类新型威胁. Shao 等 [140] 通过动态分析程序代码, 抽取相关链接并对动态页面进行分析, 将重定向链接以及动态页面上所提供的下载文件提交至 VirusTotal 进行恶意检测, 最终判断该程序是否具有广告内容安全问题.

广告欺诈主要指利用自动化脚本或者引诱用户等形式进行虚假的广告点击, 从而非法获得商业利润或其他利益. 广告欺诈不仅给用户带来不好的使用体验, 还会给广告主带来巨额的经济损失. 根据其欺诈形式, 可分为静态布局欺诈与动态交互欺诈. 静态布局欺诈指利用单个广告页面上的布局大小、位置、数量等参数来欺骗用户进行点击, 包括广告隐藏欺诈、广告大小欺诈、广告数量欺诈以及广告覆盖欺诈. 动态交互欺诈指利用用户交互行为在多个广告页面上欺骗用户进行点击, 包括交互广告欺诈、下载广告欺诈、界外广告欺诈、频繁广告欺诈以及无内容广告欺诈. 针对静态布局欺诈, Crussell 等 [141] 提出了 MAdFraud, 首先构建 HTTP 请求树模型, 然后基于机器学习方法识别广告请求页面, 最后基于启发式规则检测 HTTP 请求树上的广告点击. 但是该方法无法处理动态交互欺诈行为, 为了解决该问题, Dong 等 [142] 提出了 FraudDroid, 首先构建 UI 界面的状态转换图, 然后针对不同类型的动态交互欺诈方式提出相应的启发式检测方法.

约会诈骗指的是利用约会软件通过吸引眼球的话语或图片诱惑用户进行缴费聊天. 该类软件与以往的恶意软件性质不同, 由于此类软件没有恶意行为, 在欺诈过程中仅仅充当载体的作用, 所以依靠传统的代码分析以及行为检测等技术无法有效识别. Hu 等 [143] 首先通过关键字匹配以及静态程序分析技术在百万软件中筛选出包含消费服务的约会软件, 然后通过代码相似性检测将这些软件划分为不同家族, 最后基于人工分析并结合评论分析判断其是否具有欺诈行为.

除了上述安全威胁外, 还存在例如篡改剪切板内容等其他各式各样的新型威胁. 以上威胁归根结底还是人与人之间的较量, 这将成为未来移动平台诈骗的主流趋势. 由于该类型诈骗依托于具有正常行为的软件上, 因此基于传统的机器学习等恶意软件检测方法无法有效处理当前的新型威胁. 针对该类型威胁, 如何提出快速有效的防御措施值得我们在未来进行更深入的探索. 结合自身经验, 我们认为未来需要做到两点, 首先是如何有效提高用户的防范意识, 是否能够通过异常行为分析技术自动识别诈骗行为, 在用户被欺骗之前进行有效提醒; 其次, 针对可能出现的威胁, 如何基于现有知识设计预防措施, 变被动防御为主动防御.

8 结束语

Android 系统作为目前最流行的移动操作系统, 其已经成为了恶意软件的主要攻击目标. 恶意软

件的快速增长给移动智能手机用户带来了巨大的危害. 移动应用的安全问题已经得到越来越多研究人员、安全公司等学术界和工业界的普遍关注. 目前, 学术界已经提出了大量的恶意软件检测方法, 并已经形成了一些相对成熟的恶意软件检测工具与系统. 本文对该领域的研究成果进行了回顾, 首先介绍了恶意软件检测所面临的问题与挑战, 然后综述了近些年的恶意软件检测所使用的数据集信息以及相关方法, 将现有方法分为了基于特征码、基于机器学习以及基于行为 3 大类, 并针对各类方法所使用的技术进行了归纳总结, 全面比较和分析了不同技术的优缺点. 最后, 梳理了恶意软件检测的一些新技术以及面临的相应挑战并对未来的研究方法进行了展望.

总体而言, 目前的恶意软件检测技术还处于理论研究阶段, 在实际情况中有效应用还存在很大的局限性. 因此, 迫切需要学术界、工业界一起对恶意软件检测技术的深入探索, 这是帮助恶意软件检测技术得到更好的发展与成功应用的关键.

致谢 特别感谢 “雁栖湖大数据时代软件自动化的机遇和挑战会议”.

参考文献

1 Wang H Y, Liu Z, Liang J Y, et al. Beyond google play: a large-scale comparative study of chinese Android app markets. In: Proceedings of the Internet Measurement Conference (IMC), Boston, 2018. 293–307
2 Avdiienko V, Kuznetsov K, Gorla A, et al. Mining apps for abnormal usage of sensitive data. In: Proceedings of IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE), Florence, 2015. 426–436
3 Chen K, Liu P, Zhang Y J. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets. In: Proceedings of the IEEE/ACM 36th International Conference on Software Engineering (ICSE), Hyderabad, 2014. 175–186
4 Li M H, Wang W, Wang P, et al. Libd: scalable and precise third-party library detection in Android markets. In: Proceedings of the IEEE/ACM 39th International Conference on Software Engineering (ICSE), Buenos Aires, 2017. 335–346
5 Feng Y, Anand S, Dillig I, et al. Apposcopy: semantics-based detection of Android malware through static analysis. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), Hong Kong, 2014. 576–587
6 Liu J, Wu D Y, Xue J L. TDroid: exposing app switching attacks in Android with control flow specialization. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE), Montpellier, 2018. 236–247
7 Yan J W, Deng X, Wang P, et al. Characterizing and identifying misexposed activities in Android applications. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE), Montpellier, 2018. 691–701
8 Zhou Y J, Jiang X X. Dissecting Android malware: characterization and evolution. In: Proceedings of the IEEE Symposium on Security and Privacy, San Francisco, 2012. 95–109
9 Octeau D, McDaniel P, Jha S, et al. Effective inter-component communication mapping in Android with epicc: an essential step towards holistic security analysis. In: Proceedings of the 22nd USENIX Security Symposium, Washington, 2013. 543–558
10 Chen K, Wang P, Lee Y, et al. Finding unknown malice in 10 seconds: mass vetting for new threats at the google-play scale. In: Proceedings of the 24th USENIX Security Symposium, Washington, 2015. 659–674
11 Xue L, Zhou Y J, Chen T, et al. Malton: towards on-device non-invasive mobile malware analysis for ART. In: Proceedings of the 26th USENIX Security Symposium, Vancouver, 2017. 289–306
12 Qu Z Y, Rastogi V, Zhang X Y, et al. Autocog: measuring the description-to-permission fidelity in Android applications. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), Scottsdale, 2014. 1354–1365

13 Zhu Z Y, Dumitras T. FeatureSmith: automatically engineering features for malware detection by mining the security literature. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), Vienna, 2016. 767–778
14 Au K W, Zhou Y F, Huang Z, et al. Pscout: analyzing the Android permission specification. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), Raleigh, 2012. 217–228
15 Arp D, Spreitzenbarth M, Hubner M, et al. DREBIN: effective and explainable detection of Android malware in your pocket. In: Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS), San Diego, 2014
16 Feng Y, Bastani O, Martins R, et al. Automated synthesis of semantic malware signatures using maximum satisfiability. In: Proceedings of the 24th Annual Network and Distributed System Security Symposium (NDSS), San Diego, 2017
17 Mariconti E, Onwuzurike L, Andriotis P, et al. Mamadroid: detecting Android malware by building Markov chains of behavioral models. In: Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS), San Diego, 2016
18 Fan M, Liu J, Wang W, et al. DAPASA: detecting Android piggybacked apps through sensitive subgraph analysis. IEEE Trans Inform Forensic Secur, 2017, 12: 1772–1785
19 Wang W, Wang X, Feng D W, et al. Exploring permission-induced risk in Android applications for malicious application detection. IEEE Trans Inform Forensic Secur, 2014, 9: 1869–1882
20 Rastogi V, Chen Y, Jiang X X. Catch me if you can: evaluating Android anti-malware against transformation attacks. IEEE Trans Inform Forensic Secur, 2014, 9: 99–108
21 Liu J, Su P R, Yang M, et al. Software and cyber security — a survey. J Softw, 2018, 29: 42–68 [刘剑, 苏璞睿, 杨珉, 等. 软件与网络安全研究综述. 软件学报, 2018, 29: 42–68]
22 Qing S H. Research progress on Android security. J Softw, 2016, 27: 45–71 [卿斯汉. Android 安全研究进展. 软件学报, 2016, 27: 45–71]
23 Zhang Y Q, Wang K, Yang H, et al. Survey of Android OS security. J Comput Res Develop, 2014, 51: 1385–1396 [张玉清, 王凯, 杨欢, 等. Android 安全综述. 计算机研究与发展, 2014, 51: 1385–1396]
24 Nan Y Z, Yang M, Yang Z M, et al. UIPicker: user-input privacy identification in mobile applications. In: Proceedings of the 24th USENIX Security Symposium, Washington, 2015. 993–1008
25 Jiang X X. Security Alert: New Stealthy Android Spyware-Plankton-Found in Official Android Market. 2011. https://www.csc2.ncsu.edu/faculty/xjiang4/Plankton/
26 Fan M, Liu J, Luo X P, et al. Android malware familial classification and representative sample selection via frequent subgraph analysis. IEEE Trans Inform Forensic Secur, 2018, 13: 1890–1905
27 Fan M, Liu J, Luo X P, et al. Frequent subgraph based familial classification of Android malware. In: Proceedings of IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, 2016. 24–35
28 Zhang M, Duan Y, Yin H, et al. Semantics-aware Android malware classification using weighted contextual API dependency graphs. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), Scottsdale, 2014. 1105–1116
29 Tian Z Z, Liu T, Zheng Q H, et al. Exploiting thread-related system calls for plagiarism detection of multithreaded programs. J Syst Softw, 2016, 119: 136–148
30 Tian Z Z, Liu T, Zheng Q H, et al. Reviving sequential program birthmarking for multithreaded software plagiarism detection. IEEE Trans Softw Eng, 2018, 44: 491–511
31 Li L, Bissyande, T, Octeau D, et al. Droidra: taming reflection to support whole-program analysis of Android apps. In: Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA), Saarbrucken, 2016. 318–329
32 Xue L, Luo X P, Yu L, et al. Adaptive unpacking of Android apps. In: Proceedings of the IEEE/ACM 39th International Conference on Software Engineering (ICSE), Buenos Aires, 2017. 358–369
33 Kalysch A, Milisterfer O, Protsenko M, et al. Tackling Androids native library malware with robust, efficient and accurate similarity measures. In: Proceedings of the 13th International Conference on Availability, Reliability and

Security, Hamburg, 2018. 1–10
34 Qian C X, Luo X P, Shao Y R, et al. On tracking information flows through JNI in Android applications. In: Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Portland, 2014. 180–191
35 Xue L, Qian C X, Zhou H, et al. NDroid: toward tracking information flows across multiple Android contexts. IEEE Trans Inform Forensic Secur, 2019, 14: 814–828
36 Dong S K, Li M H, Diao W R, et al. Understanding Android obfuscation techniques: a large-scale investigation in the wild. In: Proceedings of the Security and Privacy in Communication Networks (SecureComm), Singapore, 2018. 172–192
37 Wang P, Bao Q K, Wang L, et al. Software protection on the Go: a large-scale empirical study on mobile app obfuscation. In: Proceedings of the 40th International Conference on Software Engineering (ICSE), Gothenburg, 2018. 26–36
38 Rastogi V, Chen Y, Jiang X X. Droidchameleon: evaluating Android anti-malware against transformation attacks. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), Berlin, 2013. 329–334
39 Son D. AVPASS-tool for leaking and bypassing Android malware detection system. 2017. https://www.kitploit.com/2017/08/avpass-tool-for-leaking-and-bypassing.html?m=1
40 Jordaney R, Sharad K, Dash S K, et al. Transcend: detecting concept drift in malware classification models. In: Proceedings of the 26th USENIX Security Symposium, Vancouver, 2017. 625–642
41 Ji S L, Li J F, Du T Y, et al. A survey on techniques, applications and security of machine learning interpretability. J Comput Res Develop, 2019, 56: 2071–2096
42 Guidotti R, Monreale A, Ruggieri S, et al. A survey of methods for explaining black box models. ACM Comput Surv, 2019, 51: 1–42
43 Wei F G, Li Y P, Roy S, et al. Deep ground truth analysis of current Android malware. In: Proceedings of International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Bonn, 2017. 252–276
44 Wang H Y, Si J J, Li H, et al. RmvDroid: towards a reliable Android malware dataset with app metadata. In: Proceedings of IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), Montreal, 2019. 404–408
45 Allix K, Bissyande T F, Klein J, et al. Androzoo: collecting millions of Android apps for the research community. In: Proceedings of IEEE/ACM 13rd International Conference on Mining Software Repositories (MSR), Austin, 2016. 468–471
46 Meng G Z, Xue Y X, Siow J K, et al. Androvault: constructing knowledge graph from millions of Android apps for automated analysis. 2017. ArXiv: 1711.07451
47 Sebastian M, Rivera R, Kotzias P, et al. Avclass: a tool for massive malware labeling. In: Proceedings of 19th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID), Paris, 2016. 230–253
48 Apktool. A tool for reverse engineering Android apk files. 2019. https://ibotpeaches.github.io/Apktool/
49 Xue L, Luo X P, Yu L, et al. Adaptive unpacking of Android apps. In: Proceedings of the 39th International Conference on Software Engineering (ICSE), Buenos Aires, 2017. 358–369
50 Zhang Y Q, Luo X P, Yin H Y. Dexhunter: toward extracting hidden code from packed Android applications. In: Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS), Vienna, 2015. 293–311
51 Duan Y, Zhang M, Bhaskar A V, et al. Things you may not know about Android (un)packers: a systematic study based on whole-system emulation. In: Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS), San Diego, 2018
52 Ristad E S, Yianilos P N. Learning string-edit distance. IEEE Trans Pattern Anal Machine Intell, 1998, 20: 522–532
53 Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification. In: Proceedings of the ACM Conference on Computer and Communications Security, Chicago, 2009. 235–245

54 Zhou Y J, Wang Z, Zhou W, et al. Hey, you, get off of my market: detecting malicious apps in official and alternative Android markets. In: Proceedings of the 19th Annual Network and Distributed System Security Symposium (NDSS), San Diego, 2012
55 Seo S H, Gupta A, Sallam A M, et al. Detecting mobile malware threats to homeland security through static analysis. J Netw Comput Appl, 2014, 38: 43–53
56 Zheng M, Sun M S, Lui J. Droid analytics: a signature based analytic system to collect, extract, analyze and associate Android malware. In: Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Melbourne, 2013. 163–171
57 Afonso V, Bianchi A, Fratantonio Y, et al. Going native: using a large-scale analysis of Android apps to create a practical native-code sandboxing policy. In: Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS), San Diego, 2016
58 Sun M T, Tan G. Nativeguard: protecting Android applications from third-party native libraries. In: Proceedings of the 7th ACM Conference on Security & Privacy in Wireless and Mobile Networks, Oxford, 2014. 165–176
59 Alam S, Qu Z Y, Riley R, et al. DroidNative: automating and optimizing detection of Android native code malware variants. Comput Secur, 2017, 65: 230–246
60 Alam S, Horspool R N, Traore I. MAIL: malware analysis intermediate language: a step towards automating and optimizing malware detection. In: Proceedings of the 6th International Conference on Security of Information and Networks, Aksaray, 2013. 233–240
61 Sanz B, Santos I, Laorden C, et al. Puma: permission usage to detect malware in Android. In: Proceedings of International Joint Conference CISIS, Ostrava, 2012. 289–298
62 Moonsamy V, Rong J, Liu S W. Mining permission patterns for contrasting clean and malicious Android applications. Future Generation Comput Syst, 2014, 36: 122–132
63 Aung Z, Zaw W. Permission-based Android malware detection. Int J Sci Technol Res, 2013, 2: 228–234
64 Liu X, Liu J Q. A two-layered permission-based Android malware detection scheme. In: Proceedings of the 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, Oxford, 2014. 142–148
65 Li J, Sun L C, Yan Q B, et al. Significant permission identification for machine-learning-based Android malware detection. IEEE Trans Ind Inf, 2018, 14: 3216–3225
66 Aafer Y, Du W L, Yin H. Droidapiminer: mining API-level features for robust malware detection in Android. In: Proceedings of the International Conference on Security and Privacy in Communication Networks, Sydney, 2013. 86–103
67 Zhao M, Ge F B, Zhang T, et al. AntiMalDroid: an efficient SVM-based malware detection framework for Android. In: Proceedings of the 2nd International Conference, Qinhuangdao, 2011. 158–166
68 Isohara T, Takemori K, Kubota A. Kernel-based behavior analysis for Android malware detection. In: Proceedings of the 7th International Conference on Computational Intelligence and Security (CIS), Sanya, 2011. 1011–1015
69 Peiravian N, Zhu X Q. Machine learning for Android malware detection using permission and api calls. In: Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence, Herndon, 2013. 300–305
70 Chan P P, Song W K. Static detection of Android malware by using permissions and API calls. In: Proceedings of the International Conference on Machine Learning and Cybernetics, LanZhou, 2014. 82–87
71 Wu D J, Mao C H, Wei T E, et al. Droidmat: Android malware detection through manifest and API calls tracing. In: Proceedings of the 7th Asia Joint Conference on Information Security, Kaohsiung, 2012. 62–69
72 Zhang L S, Niu Y, Wu X, et al. A3: automatic analysis of Android malware. In: Proceedings of the 1st International Workshop on Cloud Computing and Information Security, 2013
73 Sanz B, Santos I, Xabier U P, et al. Anomaly detection using string analysis for Android malware detection. In: Proceedings of the International Conference on Soft Computing Models in Industrial and Environmental Applications, Bilbao, 2014. 469–478
74 Wang X, Wang W, He Y, et al. Characterizing Android apps' behavior for effective detection of malapps at large scale. Future Generation Comput Syst, 2017, 75: 30–45
75 Tang A, Sethumadhavan S, Stolfo S J. Unsupervised anomaly-based malware detection using hardware features.

In: Proceedings of the International Workshop on Recent Advances in Intrusion Detection, Gothenburg, 2014. 109–129

76 Garcia J, Hammad M, Malek S. Lightweight, obfuscation-resilient detection and family identification of Android malware. ACM Trans Softw Eng Methodol, 2018, 26: 1–29

77 Tian Z Z, Zheng Q H, Liu T, et al. Software plagiarism detection with birthmarks based on dynamic key instruction sequences. IEEE Trans Softw Eng, 2015, 41: 1217–1235

78 Canfora G, De L A, Medvet E, et al. Effectiveness of opcode ngrams for detection of multi family Android malware. In: Proceedings of the 10th International Conference on Availability, Reliability and Security, Toulouse, 2015. 333–340

79 Zhang B, Xiao W T, Xiao X, et al. Ransomware classification using patch-based CNN and self-attention network on embedded N-grams of opcodes. Future Generation Comput Syst, 2019. doi: 10.1016/j.future.2019.09.025

80 Suarez-Tangil G, Tapiador J E, Peris-Lopez P, et al. Dendroid: a text mining approach to analyzing and classifying code structures in Android malware families. Expert Syst Appl, 2014, 41: 1104–1117

81 Teufl P, Ferk M, Fitzek A, et al. Malware detection by applying knowledge discovery processes to application metadata on the Android market (Google Play). Secur Comm Netw, 2016, 9: 389–419

82 Grampurohit V, Kumar V, Rawat S, et al. Category based malware detection for Android. In: Proceedings of the International Symposium on Security in Computing and Communication, Delhi, 2014. 239–249

83 Wang W, Li Y Y, Wang X, et al. Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers. Future Generation Comput Syst, 2018, 78: 987–994

84 Gorla A, Tavecchia I, Gross F, et al. Checking app behavior against app descriptions. In: Proceedings of the 36th International Conference on Software Engineering (ICSE), Hyderabad, 2014. 1025–1035

85 Fan M, Luo X P, Liu J, et al. CTDroid: leveraging a corpus of technical blogs for Android malware analysis. IEEE Trans Rel, 2020, 69: 124–138

86 Gascon H, Yamaguchi F, Arp D, et al. Structural detection of Android malware using embedded call graphs. In: Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security (AiSec), Berlin, 2013. 45–54

87 Hu W J, Tao J, Ma X B, et al. MIGDroid: detecting app-repackaging Android malware via method invocation graph. In: Proceedings of the 23rd International Conference on Computer Communication and Networks (ICCCN), Shanghai, 2014. 1–7

88 Marastoni N, Continella A, Quarta D, et al. GroupDroid: automatically grouping mobile malware by extracting code similarities. In: Proceedings of the 7th Software Security, Protection, and Reverse Engineering/Software Security and Protection Workshop, Orlando, 2017. 1–12

89 Sun X, Zhongyang Y B, Xin Z, et al. Detecting code reuse in Android applications using component-based control flow graph. In: Proceedings of the 23rd USENIX Security Symposium, San Diego, 2014. 142–155

90 Meng G Z, Xue Y X, Xu Z Z, et al. Semantic modelling of Android malware for effective malware comprehension, detection, and classification. In: Proceedings of the 25th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Saarbrucken, 2016. 306–317

91 Crussell J, Gibler C, Chen H. Attack of the clones: detecting cloned applications on Android markets. In: Proceedings of the 17th European Symposium on Research in Computer Security (ESORICS), Pisa, 2012. 37–54

92 Wolfe B, Elish K O, Yao D F. Comprehensive behavior profiling for proactive Android malware detection. In: Proceedings of the 17th International Conference Information Security and Cryptology, Seoul, 2014. 328–344

93 Zhang F F, Huang H Q, Zhu S C, et al. ViewDroid: towards obfuscation-resilient mobile application repackaging detection. In: Proceedings of the 7th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec), Oxford, 2014. 25–36

94 Shao Y R, Luo X P, Qian C X, et al. Towards a scalable resource-driven approach for detecting repackaged Android applications. In: Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC), New Orleans, 2014. 56–65

95 Zheng C, Zhu S X, Dai S F, et al. Smartdroid: an automatic system for revealing ui-based trigger conditions in Android applications. In: Proceedings of the 2nd ACM workshop on Security and Privacy in Smartphones and Mobile Devices, Raleigh, 2012. 93–104

96 Zhou W, Zhou Y J, Grace M, et al. Fast, scalable detection of piggybacked mobile applications. In: Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy, San Antonio, 2013. 185–196
97 Tian K, Yao D F, Ryder B G, et al. Analysis of code heterogeneity for high-precision classification of repackaged malware. In: Proceedings of the IEEE Security and Privacy Workshops, Austin, 2016. 262–271
98 Deshotels L, Notani V, Lakhotia A. Droidlegacy: automated familial classification of Android malware. In: Proceedings of the Program Protection and Reverse Engineering Workshop, New Orleans, 2014. 1–12
99 Hou S F, Ye Y F, Song Y Q, et al. Hindroid: an intelligent Android malware detection system based on structured heterogeneous information network. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD), Halifax, 2017. 1507–1515
100 Rasthofer S, Arzt S, Bodden E. A machine-learning approach for classifying and categorizing Android sources and sinks. In: Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS), San Diego, 2014
101 Hanna S, Huang L, Wu E, et al. Juxtapp: a scalable system for detecting code reuse among Android applications. In: Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Heraklion, 2012. 62–81
102 Zhou W, Zhou Y J, Jiang X X, et al. Detecting repackaged smartphone applications in third-party Android marketplaces. In: Proceedings of the Second ACM Conference on Data and Application Security and Privacy, San Antonio, 2012. 317–326
103 Narayanan A, Chandramohan M, Chen L H, et al. subgraph2vec: learning distributed representations of rooted sub-graphs from large graphs. 2016. ArXiv: 1606.08928
104 Fan M, Luo X P, Liu J, et al. Graph embedding based familial analysis of Android malware using unsupervised learning. In: Proceedings of the 41st International Conference on Software Engineering (ICSE), Montreal, 2019. 771–782
105 Ribeiro L F, Saverese P H, Figueiredo D R. struc2vec: learning node representations from structural identity. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, 2017. 385–394
106 Lin Y D, Lai Y C, Chen C H, et al. Identifying Android malicious repackaged applications by thread-grained system call sequences. Comput Secur, 2013, 39: 340–350
107 Kang H, Jang J, Mohaisen A, et al. Detecting and classifying Android malware using static analysis along with creator information. Int J Distrib Sens Netw, 2015, 11: 479174
108 Allix K, Bissyandé T F, Jérome Q, et al. Empirical assessment of machine learning-based malware detectors for Android. Empir Softw Eng, 2016, 21: 183–211
109 Zhang Y, Yang M, Xu B Q, et al. Vetting undesirable behaviors in Android apps with permission use analysis. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), Berlin, 2013. 611–622
110 Enck W, Gilbert P, Han S, et al. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans Comput Syst, 2014, 32: 5
111 Hornyack P, Han S, Jung J, et al. These aren't the droids you're looking for: retrofitting Android to protect data from imperious applications. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), Chicago, 2011. 639–652
112 Arzt S, Rasthofer S, Fritz C, et al. Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Edinburgh, 2014. 259–269
113 Klieber W, Flynn L, Bhosale A, et al. Android taint flow analysis for app sets. In: Proceedings of the ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis (SOAP), Edinburgh, 2014. 1–6
114 Li L, Bartel A, Bissyande T, et al. Iccta: detecting inter-component privacy leaks in Android apps. In: Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE), Florence, 2015. 280–291
115 Octeau D, Luchaup D, Dering M, et al. Composite constant propagation: application to Android inter-component

communication analysis. In: Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE), Florence, 2015. 77–88

116 Huang J J, Li Z C, Xiao X S, et al. SUPOR: precise and scalable sensitive user input detection for Android apps. In: Proceedings of the USENIX Security Symposium, Austin, 2015. 977–992

117 Felt A P, Chin E, Hanna S, et al. Android permissions demystified. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), Chicago, 2011. 627–638

118 Chin E, Felt A, Greenwood K, et al. Analyzing inter-application communication in Android. In: Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys), Bethesda, 2011. 239–252

119 Lu L, Li Z C, Wu Z Y, et al. Chex: statically vetting Android apps for component hijacking vulnerabilities. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), Raleigh, 2012. 229–240

120 Kantola D, Chin E, He W, et al. Reducing attack surfaces for intra-application communication in Android. In: Proceedings of the Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM), Raleigh, 2012. 69–80

121 Pandita R, Xiao X S, Yang W, et al. WHYPER: towards automating risk assessment of mobile applications. In: Proceedings of the USENIX Security Symposium, Washington, 2013. 527–542

122 Yu L, Luo X P, Qian C X, et al. Enhancing the description-to-behavior fidelity in Android apps with privacy policy. IIEEE Trans Softw Eng, 2018, 44: 834–854

123 Yu L, Luo X P, Liu X L, et al. Can we trust the privacy policies of Android apps? In: Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Toulouse, 2016. 538–549

124 Yu L, Luo X P, Chen J C, et al. PPChecker: towards accessing the trustworthiness of Android apps' privacy policies. IEEE Trans Softw Eng, 2018. doi: 10.1109/TSE.2018.2886875

125 Slavin R, Wang X Y, Hosseini M, et al. Toward a framework for detecting privacy policy violations in Android application code. In: Proceedings of the 38th International Conference on Software Engineering (ICSE), Austin, 2016. 25–36

126 Hinton G E, Osindero S, Teh Y W. A fast learning algorithm for deep belief nets. Neural Comput, 2006, 18: 1527–1554

127 Pascanu R, Stokes J W, Sanossian H, et al. Malware classification with recurrent networks. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Queensland, 2015. 1916–1920

128 David O E, Netanyahu N S. Deepsign: deep learning for automatic malware signature generation and classification. In: Proceedings of the International Joint Conference on Neural Networks, Killarney, 2015. 1–8

129 Saxe J, Berlin K. Deep neural network based malware detection using two dimensional binary program features. In: Proceedings of the 10th International Conference on Malicious and Unwanted Software, Fajardo, 2015. 11–20

130 Yuan Z L, Lu Y Q, Xue Y B. Droiddetector: Android malware characterization and detection using deep learning. Tinshhua Sci Technol, 2016, 21: 114–123

131 McLaughlin N, Martinez R J, Kang B, et al. Deep Android malware detection. In: Proceedings of the Conference on Data and Application Security and Privacy, Scottsdale, 2017. 301–308

132 Fereidooni H, Conti M, Yao D F, et al. ANASTASIA: Android malware detection using static analysis of applications. In: Proceedings of the 8th IFIP International Conference on New Technologies, Mobility and Security, Larnaca, 2016. 1–5

133 Kim T G, Kang B J, Rho M, et al. A multimodal deep learning method for Android malware detection using various features. IEEE Trans Inform Forensic Secur, 2019, 14: 773–788

134 Tan S, Caruana R, Hooker G, et al. Learning global additive explanations for neural nets using model distillation. 2018. ArXiv: 1801.08640

135 Ribeiro M T, Singh S, Guestrin C. Why should I trust you? Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), San Francisco, 2016. 1135–1144

136 Fratantonio Y, Bianchi A, Robertson W, et al. Triggerscope: towards detecting logic bombs in Android applications. In: Proceedings of the IEEE Symposium on Security and Privacy, San Jose, 2016. 377–396

137 Suciu O, Coull S E, Johns J. Exploring adversarial examples in malware detection. 2018. ArXiv: 1810.08280
138 Grosse K, Papernot N, Manoharan P, et al. Adversarial examples for malware detection. In: Proceedings of the European Symposium on Research in Computer Security, Oslo, 2017. 62–79
139 Al-Dujaili A, Huang A, Hemberg E, et al. Adversarial deep learning for robust detection of binary encoded malware. In: Proceedings of the IEEE Security and Privacy Workshops (SPW), Gothenburg, 2018. 76–82
140 Shao R, Rastogi V, Chen Y, et al. Understanding in-app ads and detecting hidden attacks through the mobile app-web interface. IEEE Trans Mobile Comput, 2018, 17: 2675–2688
141 Crussell J, Stevens R, Chen H. Madfraud: investigating ad fraud in Android applications. In: Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys), Bretton Woods, 2014. 123–134
142 Dong F, Wang H Y, Li L, et al. Frauddroid: automated ad fraud detection for Android apps. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE), Lake Buena Vista, 2018. 257–268
143 Hu Y Y, Wang H Y, Zhou Y J, et al. Dating with scambots: understanding the ecosystem of fraudulent dating applications. IEEE Trans Dependable Secure Comput, 2019. doi: 10.1109/TDSC.2019.2908939

Android malware detection: a survey

Ming FAN[1], Ting LIU[1*], Jun LIU[3], Xiapu LUO[2], Le YU[2] & Xiaohong GUAN[1]

1. *School of Cyber Science and Engineering, Xi'an Jiaotong University, Xi'an* 710049*, China*;
2. *Department of Computing, The Hong Kong Polytechnic University, Hong Kong* 999077*, China*;
3. *School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an* 710049*, China*
* Corresponding author. E-mail: tingliu@mail.xjtu.edu.cn

Abstract Android has become the most popular mobile operating system in the past ten years due to its three main advantages, namely, the openness of source code, richness of hardware selection, and millions of applications (apps). It is of no surprise that Android has become the major target of malware. The rapid increase in the number of Android malware poses big threats to smart phone users such as financial charges, information collection, and remote control. Thus, the in-depth study of the security issues of mobile apps is of great importance to the sound development of the smart phone ecosystem. We first introduce the existing problems and challenges of malware analysis, and then summarize the widely-used benchmark datasets. After that, we divide the existing malware analysis methods into three categories, including signature-based methods, machine learning-based methods, and behavior-based methods. We further summarize the techniques used in each method, and compare and analyze the advantages and disadvantages of different techniques. Finally, combined with our own research foundation in malware analysis, we explore and discuss future research directions and challenges.

Keywords Android, malware detection, familial identification, machine learning

Ming FAN was born in 1991. He received his B.S. and Ph.D. degrees in computer science and technology from Xi'an Jiaotong University, China, in 2013 and 2019, respectively. He received his Ph.D. degree in computing from The Hong Kong Polytechnic University in 2019. He is currently a lecturer in the School of Cyber Science and Engineering at Xi'an Jiaotong University, China. His research interests include trustworthy software and Android malware detection and familial identification.

Jun LIU was born in 1973. He received his B.S. and Ph.D. degrees in computer science and technology from Xi'an Jiaotong University in 1995 and 2004, respectively. He is currently a professor in the Department of Computer Science and Technology, Xi'an Jiaotong University, China. His current research focuses on data mining and text mining.

Ting LIU was born in 1981. He received his B.S. and Ph.D. degrees from Xi'an Jiaotong University, Xi'an, China, in 2003 and 2010, respectively. He was a visiting professor at Cornell University. He is currently a professor at the Systems Engineering Institute, Xi'an Jiaotong University. His research interests include software security and smart grids security.

Xiaohong GUAN was born in 1955. He received his B.S. and M.S. degrees in automatic control from Tsinghua University, Beijing, China, in 1982 and 1985, respectively, and a Ph.D. degree in electrical engineering from the University of Connecticut, Storrs, CT, USA, in 1993. Since 1995, he has been at the Systems Engineering Institute, Xi'an Jiaotong University, Xi'an, China. He became Dean of the School of Electronic and Information Engineering in 2008. His research interests include optimization of power and energy systems, electric power markets, and cyberphysical systems such as smart grids and sensor networks