软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(6):2683−2712 [doi: 10.13328/j.cnki.jos.007312] [CSTR: 32375.14.jos.007312]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

安卓恶意软件对抗样本攻击技术综述*

李 珩[1], 吴 棒[1], 龚 柱[1], 高翠莹[1], 袁 巍[1], 罗夏朴[2]

[1](华中科技大学 电子信息与通信学院, 湖北 武汉 430074)

[2](香港理工大学 计算机系, 香港 999077)

通信作者: 袁巍, E-mail: yuanwei@mail.hust.edu.cn

摘　要: 面对 Android 恶意软件带来的严重的安全风险, 如何有效检测 Android 恶意软件已成为工业界与学术界共同关注的焦点. 然而随着 Android 对抗样本技术的出现, 现有的恶意软件检测系统面临着前所未有的挑战. Android 恶意软件对抗样本攻击通过对恶意软件的源码或特征进行扰动, 使其在保持原始功能不受影响的条件下绕过恶意软件检测模型. 尽管目前已有大量针对恶意软件的对抗样本攻击研究, 但是现阶段仍缺乏针对 Android 系统对抗样本攻击的完备性综述, 且并未研究 Android 系统中对抗样本设计的独特要求, 因此首先介绍 Android 恶意软件检测的基本概念; 然后从不同角度对现有的 Android 对抗样本技术进行分类, 梳理 Android 对抗样本技术的发展脉络; 随后综述近年来的 Android 对抗样本技术, 介绍不同类别的代表性工作并分析其优缺点; 之后, 分类介绍常用的安卓对抗样本攻击所使用的代码扰动手段并分析其应用场景; 最后讨论 Android 恶意软件对抗样本技术面临的挑战, 展望该新兴领域的未来研究方向.

关键词: 安卓恶意软件检测; 对抗样本攻击; AI 安全; 软件安全

中图法分类号: TP311

中文引用格式: 李珩, 吴棒, 龚柱, 高翠莹, 袁巍, 罗夏朴. 安卓恶意软件对抗样本攻击技术综述. 软件学报, 2025, 36(6): 2683–2712. http://www.jos.org.cn/1000-9825/7312.htm

英文引用格式: Li H, Wu B, Gong Z, Gao CY, Yuan W, Luo XP. Survey on Android Malware Adversarial Example Attack Techniques. Ruan Jian Xue Bao/Journal of Software, 2025, 36(6): 2683–2712 (in Chinese). http://www.jos.org.cn/1000-9825/7312.htm

Survey on Android Malware Adversarial Example Attack Techniques

LI Heng[1], WU Bang[1], GONG Zhu[1], GAO Cui-Ying[1], YUAN Wei[1], LUO Xia-Pu[2]

[1](School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan 430074, China)

[2](Department of Computing, The Hong Kong Polytechnic University, Hong Kong 999077, China)

Abstract: In the face of the severe security risks posed by Android malware, effective Android malware detection has become the focus of common concern in both the industry and academia. However, with the emergence of Android adversarial example techniques, existing malware detection systems are facing unprecedented challenges. Android malware adversarial example attacks can bypass existing malware detection models by perturbing the source code or characteristics of malware while keeping its original functionality inact. Despite substantial research on adversarial example attacks against malware, there is still a lack of a comprehensive review specifically focusing on adversarial example attacks in the Android system at present, and the unique requirements for adversarial example design within the Android system are not studied. Therefore, this study begins by introducing the fundamental concepts of Android malware detection. It then classifies existing Android adversarial example techniques from various perspectives and provides an overview of the development sequence of Android adversarial example techniques. Subsequently, it reviews Android adversarial example techniques in recent years, introduces representative work in different categories and analyzes their pros and cons. Furthermore, it categorizes and introduces common

* 基金项目: 中国博士后科学基金 (2024M751010); 国家资助博士后研究人员计划 (GZB20240248); 网络空间安全教育部重点实验室及河南省网络密码技术重点实验室开放基金课题 (KLCS20240305); CCF-绿盟科技“鲲鹏”科研基金 (CCF-NSFOCUS 2023011)
收稿时间: 2023-11-02; 修改时间: 2024-01-28, 2024-06-03; 采用时间: 2024-10-30; jos 在线出版时间: 2025-03-12
CNKI 网络首发时间: 2025-03-13

means of code perturbation in Android adversarial example attacks, and analyzes their application scenarios. Finally, it discusses the challenges faced by Android malware adversarial example techniques, and envisions future research directions in this emerging field.

Key words: Android malware detection; adversarial example attack; AI security; software security

随着移动互联网的飞速发展, 手机等移动设备逐步成为人们上网的主要工具. 在这些移动设备中, Android (安卓) 占据了约 87% 的市场份额, 是最流行的移动操作系统[1]. 然而, 由于其巨大的市场份额和开放的生态系统, Android 不仅吸引了合法的 Android 应用程序 (Android application, APP) 开发者, 也吸引了传播恶意软件的攻击者, 攻击者通过恶意软件实现对智能手机用户的有害意图 (如恶意扣费、资源消耗、隐私窃取等)[2]. 在 2022 年, 卡巴斯基移动产品和技术检测到 1 661 743 个恶意安装程序, 其中包含了 196 476 个新的移动银行木马、10 543 个新的移动勒索软件木马[3].

由于 Android 内置的安全机制并不能完全防御恶意软件, 因此, 如何有效检测 Android 恶意软件迅速成为工业界与学术界共同关注的焦点. 早期的检测方法主要使用签名 (signature) 来甄别 Android 恶意软件[4−8], 但实际效果并不理想. 现行的检测方法主要基于机器学习, 它们将恶意软件检测视为一个分类问题, 先从 APP 中提取检测所需特征, 再输入到预先训练好的分类模型, 最终得到良性或恶性的检测结果. 现有的检测特征可大致分为语法 (syntactic) 特征[6,7,9−15]和语义 (semantic) 特征[16−28]两类, 而分类器可由传统分类模型 (如支持向量机 (support vector machine, SVM)[29]) 或基于深度学习的分类模型 (如卷积神经网络 (convolutional neural network, CNN)[30]) 实现.

随着各种基于机器学习的 Android 恶意软件检测方法不断涌现, 其检测性能也不断攀升, 然而这些工作在最近的研究中被证实[31−40]极易受到对抗样本 (adversarial example, AE) 的攻击. 2013 年, Szegedy 等人[40]首次提出了对抗样本的概念, 他们发现在自然样本中添加细微的干扰, 所形成的样本可以使得分类模型以高置信度给出一个错误的分类结果. 尽管对抗样本首先出现在计算机视觉领域[41−49], 但该现象在 Android 恶意软件检测领域也同样存在[50−63]. 为描述方便, 我们将 Android 恶意软件检测领域中的对抗样本简称为 APP 对抗样本. 图像对抗样本通过在像素点上施加扰动而生成, 而 Android 对抗样本则是在不影响软件原有功能的前提下, 通过对 Android 恶意软件的底层代码进行修改, 再重新打包而生成.

对抗样本技术在 Android 恶意软件中的应用对软件安全领域造成了巨大的冲击, 例如 Li 等人[50]证明了 Android 恶意软件对抗样本对俄罗斯公司开发的卡巴斯基, 美国公司开发的 McAfee、Comodo、Symantec, 德国公司开发的小红伞等商用 Android 恶意软件检测系统具有一定的攻击成功率. 更进一步地, He 等人[64]证明了, 即使对商用的恶意软件检测所使用的特征一无所知的场景下, 也可以对 McAfee、Symantec、MobileInsight、Microsoft、Avira 等知名恶意软件检测系统达到 50% 以上的攻击成功率. 对抗样本攻击一旦突破了 Android 恶意软件检测系统的防线, 将严重侵害用户的隐私和权益, 甚至还会给国家带来政治、经济、军事等多方面的威胁.

充分理解与研究恶意软件对抗样本攻击技术是鲁棒性恶意软件检测的前提与必要条件, 为后续鲁棒检测模型的构建提供更好的数据资源、评价指标, 对人工智能在智能手机生态圈的健全发展具有重要意义. 近些年, 对抗样本攻击在移动恶意软件检测领域的快速发展使其迅速成为学术界软件安全领域中的热点问题, 已有大量的相关研究成果发表于顶级的国际会议和期刊上, Android 对抗样本攻击算法期刊会议词云统计如图 1 所示.

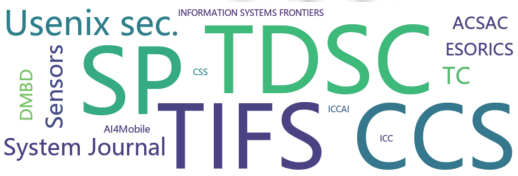

图 1 Android 对抗样本攻击算法期刊会议词云统计

随着 Android 恶意软件对抗样本的相关研究成果大量涌现, 对这些成果进行梳理和总结已成为必然. 目前已有多篇针对相关领域的综述论文发表[65−67]. 文献 [65] 由于发表时间较早, 因此缺乏近些年对于可实施的 Android 对抗样本攻击算法的讨论. 文献 [67] 由于并没有专门针对 Android 恶意软件, 因此缺乏 Android 领域对抗样本生

成过程中的问题空间实际操作的相关方法的分析. 文献 [66] 主要集中于 Android 恶意软件的防御研究综述, 对于攻击算法涉猎较少. 我们将现阶段 Android 恶意软件对抗样本综述性论文中亟待解决的问题总结如下.

(1) 缺乏对最新研究成果的总结. 现有的工作仅总结了 2021 年及以前的文献, 然而近两年大量开创性工作开始涌现.

(2) 缺乏对 Android 对抗样本的聚焦. 现有工作将 Windows、PDF、Android 恶意软件的对抗样本技术混合介绍, 忽略了 Android 系统对于对抗样本攻击技术的独特要求.

(3) 缺乏具体攻击实现手段的介绍. 暂无针对 Android 软件问题空间攻击手段的梳理、分析与展望.

因此, 本文聚焦于 Android 恶意软件检测领域, 从 Android 恶意软件开发者的视角梳理了现阶段关于 Android 恶意软件攻击的相关研究成果. 根据攻击的空间、攻击者掌握的信息以及被攻击模型使用的分类特征进行归纳总结, 全面比较和分析了不同技术的优缺点. 此外, 我们根据 Android 恶意软件攻击的实现手段进行归类分析. 最后, 结合现有的成果思考和讨论 Android 恶意软件对抗样本攻击的发展趋势.

本文第 1 节对 Android 恶意软件检测的相关定义以及所用到的特征进行介绍. 第 2 节介绍现有的 Android 恶意软件检测算法分类规则. 第 3 节依据攻击目标使用的特征对现有的 Android 对抗样本生成算法进行梳理. 第 4 节对现有的 Android 对抗扰动实现方法进行分析与总结. 第 5 节探讨 Android 领域对抗样本生成过程中遇到的挑战以及未来的研究方向. 第 6 节总结全文.

1 背景知识

1.1 APK 文件结构

安卓应用程序包 (Android application package, APK) 是一种基于 ZIP 格式的压缩文件, 其文件后缀被修改为 apk. Android 逆向工程主要通过拆包、反编译等手段将 APK 文件还原为可读的逻辑代码文件, 并加以分析和利用. 在拆包过程中, 可以利用 Apktool (反编译工具. https://ibotpeaches.github.io/Apktool/)、AndroidKiller (反编译工具. https://github.com/liaojack8/AndroidKiller) 等专业工具对 APK 进行拆分, 也可以直接将 APK 文件后缀改为 zip 或 rar, 并利用文件解压工具进行解压. 一个典型的 APK 文件通过拆包工具解压后的内容如表 1 所示.

表 1 APK 拆包后的内容说明

文件/文件夹	说明
classes.dex	字节码文件, 用以在Dalvik虚拟机上执行
AndroidManifest.xml	清单文件, 应用程序的版本号、所需权限、注册服务等信息
META-INF文件夹	用以存放应用程序的签名信息
res文件夹	编译后的资源文件
assets文件夹	不会被编译成二进制文件的APK资源文件
resources.arsc	资源索引文件, 帮助系统根据资源 ID 快速找到资源
lib文件夹	本地库文件

Android 应用的分析主要使用表 1 中的 classes.dex 字节码文件和 AndroidManifest.xml 清单文件. 其中, classes.dex 字节码文件包含了编译后的程序执行代码, 用以在 Android 虚拟机上执行. AndroidManifest.xml 清单文件则是描述了应用程序的版本号、所需权限、注册服务等信息.

拆包后得到的 classes.dex 和 AndroidManifest.xml 文件都不能直接进行阅读, 需要反编译后才能提取到有用信息. classes.dex 文件进行反编译可以得到后缀为 smali 的代码文件, smali 文件中的代码采用的是 Jasmin 语法, 语法上和汇编语言相似. 通过 smali 代码可以查看程序的变量值、应用程序编程接口 (application programming interface, API) 和返回类型等有用信息, 进而分析程序的运行过程, 常用反编译工具包括 baksmali (DEX 格式反编译器. https://github.com/JesusFreke/smali)、Apktool 等. 而对 AndroidManifest.xml 文件进行反编译后主要关注的信息包括 APK 所需的权限以及运行过程中的动作意图, 例如查看 APK 运行时是否需要开启摄像头、获取位置, 以

及运行过程中是否会监听网络连接状态等. 常利用 XMLPrinter2 (XML 格式转换器. https://github.com/digitalsleuth/AXMLPrinter2) 工具对 AndroidManifest.xml 文件进行反编译. 对一个 APK 文件进行处理的流程如图 2 所示.

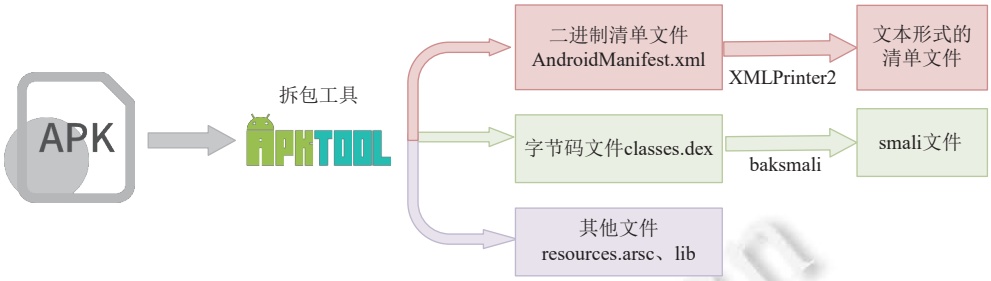

图 2 APK 处理流程图

1.2 常见恶意软件检测特征

目前的主流检测方法基于机器学习理论[68−74], 将恶意软件检测视为一个分类问题. 具体而言, 这些方法从 Android 软件中提取检测所需特征, 再利用现有的机器学习算法对提取到的特征进行分类. 这些特征大致可以分为静态特征[6,8,15]与动态特征[75−79]两类. 前者通过分析 APK 文件中的底层代码从而获得训练数据; 后者通过在沙盒中运行程序以获得训练数据. 由于后者需要对程序进行运行, 耗时耗力, 因此静态特征以低廉的检测成本被广泛使用. 根据从 Android 软件中提取到的静态特征类型不同, 基于静态特征的 Android 恶意软件检测模型可以大致分为 3 类: 基于向量特征 (vector-styled feature)、基于图特征 (graph-styled feature) 与基于图像特征 (image-styled feature) 的 Android 恶意软件检测算法. 接下来, 本文将详细介绍这 3 种特征.

(1) 向量特征

检测算法从 APK 文件中构造一组关于关键元素的向量. 这些关键元素包括从应用程序配置文件 AndroidManifest.xml 或者 smali 文件中提取的敏感 API[68,70,71,74]、权限 (permission)[71,80−82]、意图-动作 (intent action)[9,14]、组件间调用 (inter-component call, ICC)[7]等. 而关键元素对应的值, 通常可以用 0 或 1 表示. 具体而言, 1 表示当前 APK 调用某 API, 或要求某权限, 0 则相反. 一种典型的向量特征构建方式如图 3 所示, Yuan 等人[14]通过遍历 AndroidManifest 文件与 smali 文件得到某个 APK 是否含有特定的权限、动作、API, 从而为该 APK 构建出 0-1 的特征向量.

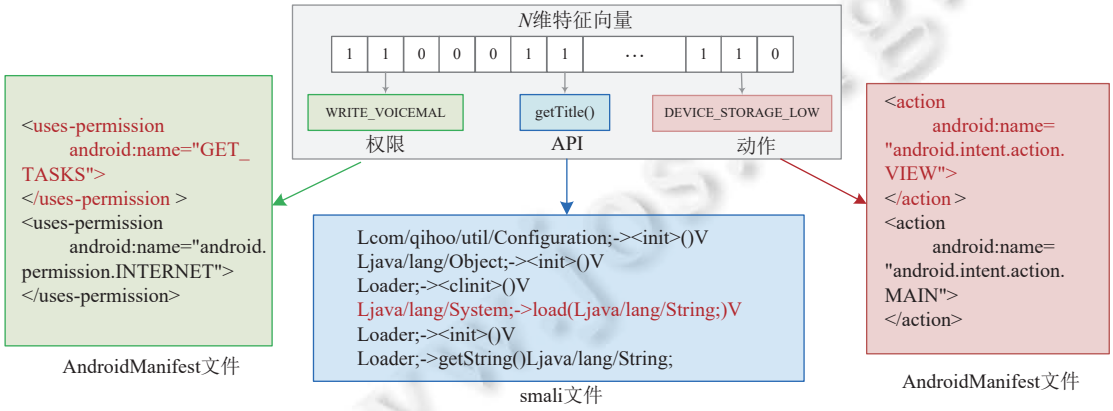

图 3 一种典型的 Android 向量特征提取方法

(2) 图特征

向量特征属于一种语法特征, 而图特征则属于语义特征, 其通过图数据的格式, 在特征提取的过程中保留更多的程序语义信息. 一般来说, 基于图数据的恶意软件检测模型由于更好地刻画了恶意软件的行为而具有更高的恶

意软件检测效率与鲁棒性. 常见的图特征有函数调用图 (function call graph, FCG)[17,20,24,25,83–85]、控制流图 (control flow graph, CFG)[69,73,86,87]、数据流图 (data flow graph, DFG)[88,89]和用户接口交互图 (user interface graph, UIG)[72,90–92]. 以函数调用图为例 (见图 4), 检测算法对 classes.dex 文件进行反编译获得 smali 文件, 然后分析 smali 文件中的代码逻辑获取函数调用图. 图数据中节点表示函数, 边表示函数之间的调用关系.

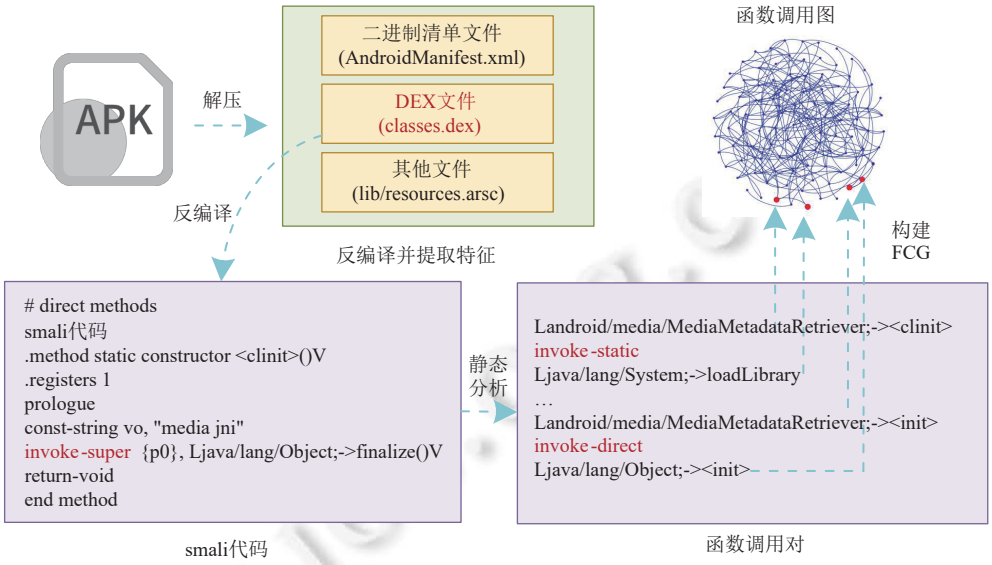

图 4 函数调用图特征提取方法

(3) 图像特征

由于近年来深度学习在图像领域的飞速发展, 有些工作[93–96]将 Android 恶意软件转换成图像数据, 进而利用已有图像分类技术对转换的图像数据进行恶意软件检测. DEX 可执行文件 (Dalvik executable file) 大致可以分为 3 个区域, 分别是文件头、索引区和数据区. 文件头包含有 DEX 文件的版本表示、文件大小、adler32 检验等信息. 索引区包含了类、方法、字段等各种符号的索引表, 其偏移量已经在文件头中定义, 而数据区则保存类的定义以及索引区中的数据. 如图 5 所示, 由于 DEX 文件是十六进制编码的文件, 因此可以每 8 bit 数据转换成图像数据中的一个像素值 (表示为 0–255 之间的数字), 以此将整个 DEX 文件转换成图像数据.

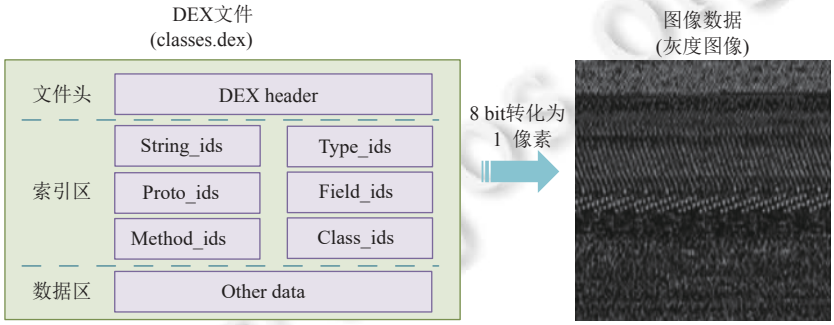

图 5 图像特征提取方法

1.3 Android 恶意软件对抗样本

1.3.1 基础知识

机器学习算法近年来广泛应用于 Android 恶意软件检测任务, 并凭借其优异的特征提取能力取得了很好的恶

意软件检测性能, 极大地促进了 Android 恶意软件检测领域的发展. 但是, 机器学习同样是一把双刃剑, 在提升恶意软件检测精度的同时, 也带来了对抗样本这一安全隐患. 对抗样本技术通过在正常样本上添加计算得到的噪声, 使得机器学习模型以高置信输出错误的结果[40]. 在 Android 恶意软件检测领域, 攻击者可以利用对抗样本算法构造 Android 恶意软件对抗样本, 使其能够躲避基于机器学习的检测模型, 这给 Android 用户和安全公司造成极大的威胁和挑战.

本节首先回顾现有的 Android 恶意软件检测流程. 如图 6 所示, 现有的基于静态分析的恶意软件检测系统大致包括问题空间中的操作与特征空间中的操作两个步骤. 在问题空间中, 首先对 APK 文件进行逆向工程操作, 利用 AXMLPrinter2、Apktool、baksmali 对 APK 文件进行拆包、反编译等, 将 APK 文件拆分成 smali、XML 等一系列可读文件. 在特征空间中, 检测系统首先从各可读文件中提取所需的权限、组件间调用、函数调用等各类特征. 随后利用卷积神经网络[30]、图卷积网络[97]等分类模型对所提取的特征进行检测判别.

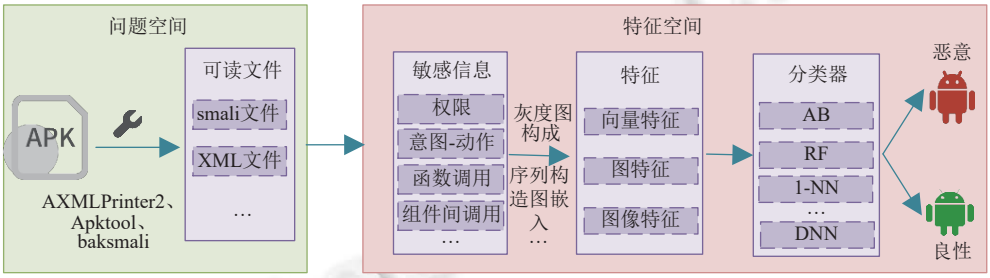

图 6 Android 恶意软件检测流程

Android 对抗样本攻击技术是信息安全领域中备受关注的一个重要议题. 它们代表了一种具有挑战性和威胁性的攻击方式, 旨在绕过 Android 恶意软件检测程序、模型或系统. Android 对抗样本的典型特征是它们的欺骗性和伪装性. 这些样本通常会对恶意软件的源码进行修改, 并伪装成合法的应用程序, 以躲避检测系统. 这些对抗样本一旦绕过了检测系统的检测, 就可能会窃取个人信息、访问敏感数据、监控用户活动, 甚至操纵 Android 设备的核心功能.

1.3.2 扰动要求

在图像领域中, 对抗样本的目标不仅是让目标模型错误分类, 也需要保证对图像的扰动尽可能小, 也就是说, 人类在视觉上无法区分对抗样本图像和正常图像. 然而由于 Android 领域的特殊性, 其对于对抗样本的生成提出了新的要求. 这些要求总结如下.

(1) 扰动目标: 计算机视觉领域中, 在测试对抗样本生成算法的效果时, 需对所有类别的图像进行攻击测试. 然而, 恶意软件即服务 (malware-as-a-service, MaaS) 开发商只关注扰动是否使恶意软件逃避检测系统. 因此在验证 Android 对抗样本攻击算法的效果时, 仅在恶意软件上进行测试.

(2) 功能留存: 在对 Android 恶意软件进行对抗扰动修改时, 不能影响原始软件的正常运行. 正是这一约束给 Android 恶意软件领域对抗样本生成带来了新的挑战. 例如, 在对基于图特征的 Android 恶意软件检测系统进行攻击时, 无法直接迁移图 (graph) 领域中已有的对抗样本攻击算法[98−107], 因为这些算法生成的扰动往往破坏了原始 APK 文件的完整性 (例如, 删除了关键的函数等). 而为了验证生成的 Android 恶意软件对抗样本功能未受影响, 往往需要对其进行功能一致性验证 (functional consistency verification)[54]. 常见的验证方法有: 静态分析法[54]与动态分析法[61]. 前者通过对修改的代码文件 (如 smali 文件或者 XML 文件) 进行检查, 分析修改后的文件是否能正常运行. 后者可以向 smali 文件中插入 log 函数, 在修改的代码前后打印相关的调用信息, 并以此为依据, 在 Android Studio 上运行并动态分析程序修改后执行了哪些调用, 从而判断恶意功能是否改变.

(3) 抗静态分析: 现有的程序分析工具 (如 Androguard (Android 逆向工具. https://github.com/androguard/androguard)) 可以对 Android 软件代码进行静态分析, 从而过滤掉无关的冗余代码段, 因此 Li 等人[61]提出, 在对恶意软件进行扰动时需要尽量避免此种分析工具将扰动滤除. 例如在对基于函数调用图的恶意软件检测系统攻击

时, 应尽量避免插入孤岛函数, 即没有被任意其他函数调用的函数.

(4) 扰动大小限制: 尽管恶意软件对抗样本无须与图像对抗样本一样具备视觉上的限制要求, 但是大多数的工作都会为了保证扰动引起的修改不会对应用产生太大的影响, 将生成的扰动量约束到一定的范围内.

上述的扰动要求同样适用于 Windows、Linux 等系统的恶意软件对抗样本攻击, 但是在实现手段上存在一定的区别. 例如, 在 Android 领域, 扰动的修改往往体现在 smali 和 XML 文件上, 同时, 扰动的实现过程往往配合着一套专用的 Android 逆向工具, 如 Apktool、baksmali、Android Studio 等.

1.3.3 Android 对抗样本发展脉络

Android 对抗样本技术经过很长时间的发展, 然而由于不同攻击方法针对的目标不同, 文献的侧重点不同, 因此对现有工作的发展脉络进行一个大致的梳理对理解领域内工作至关重要. 本文将 Android 对抗样本攻击的发展脉络及主要工作梳理在图 7 中.

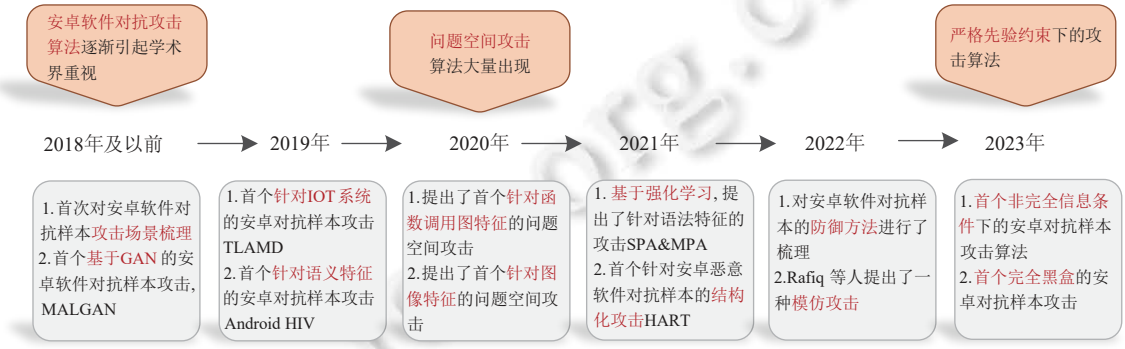

图 7 Android 对抗样本攻击的发展脉络及主要工作

在 Android 对抗样本技术发展的早期阶段, 大多攻击算法旨在明确 Android 恶意软件检测领域中对抗样本攻击算法的攻击场景与威胁模型, 且大多数攻击算法以较为简单的 0-1 向量特征为攻击目标. 例如, Demontis 等人[52]梳理了攻击者可能具备的先验知识, 即是否知道训练数据、是否知道恶意软件检测系统使用的特征、是否知道分类模型使用的算法等. Hu 等人[59]将对抗生成网络技术与 Android 对抗样本相结合, 提出了查询类的语法特征对抗样本攻击算法.

早期的工作主要聚焦于特征空间的攻击, 并未过多探究问题空间扰动的实现方式. Pierazzi 等人[58]的工作明确了 Android 对抗样本攻击中特征空间扰动与问题空间扰动之间的映射关系, 是 Android 恶意软件问题空间攻击的关键里程碑, 其提出使用非透明谓语作为代码扰动手段也为后续的问题空间攻击提供了思路. Chen 等人[108]从现有的图像领域中的 JSMA[109]与 CW[37]攻击算法获得灵感, 所提出的 Android HIV 攻击算法首次将攻击目标从简易的语法特征转向了复杂的语义特征. Zhao 等人[54]针对函数调用图特征提出了一种结构化攻击算法, 其提出了 4 种 smali 文件代码的扰动方式, 实现了高效的图特征对抗样本攻击算法. Gu 等人[60]受到单像素攻击的启发, 提出了第 1 个问题空间的针对图像特征的对抗样本攻击算法.

现阶段, 越来越多的工作关注于受限条件下的 Android 对抗样本攻击算法. 例如, Li 等人[61]在攻击函数调用图特征时, 研究如何在未知函数粒度的场景下提高攻击效率与攻击成功率. He 等人[64]更进一步将目光聚焦于零知识场景, 研究如何在未知特征的条件下实现 Android 恶意软件的对抗样本攻击.

2 Android 对抗样本攻击分类

现有工作提出了多种多样的 Android 对抗样本攻击算法, 因此需要从不同的角度对这些研究进行分类, 比如基于攻击所在的空间、攻击者所掌握的信息、被攻击模型使用的特征. 图 8 展示了 Android 对抗样本攻击的基本分类, 其中基于目标模型使用特征的分类方法最为直观且最符合 Android 软件研究的特色, 因此本文将其作为主要分类依据在第 3 节进行详细讲解.

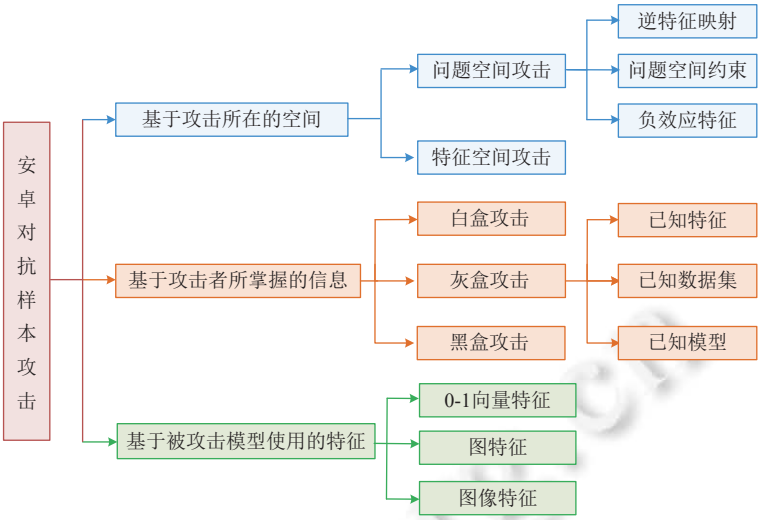

图 8 Android 对抗样本攻击的基本分类

2.1 基于攻击所在的空间

如图 9 所示, 根据攻击是否需要在代码层面上完成, Android 恶意软件对抗样本攻击可分为问题空间攻击与特征空间攻击[58]. 特征空间攻击仅对从 Android 恶意软件检测中提取到的特征进行修改, 而问题空间攻击往往需要将特征空间中的扰动映射到问题空间中的实际代码上, 完成对代码的修改. 问题空间攻击由于生成了实际的 Android 对抗样本软件, 更具有研究价值, 同时, 由于特征空间与问题空间的差异, 特征上的修改很难在问题空间中完美的复现, 这也导致问题空间的攻击更难实现.

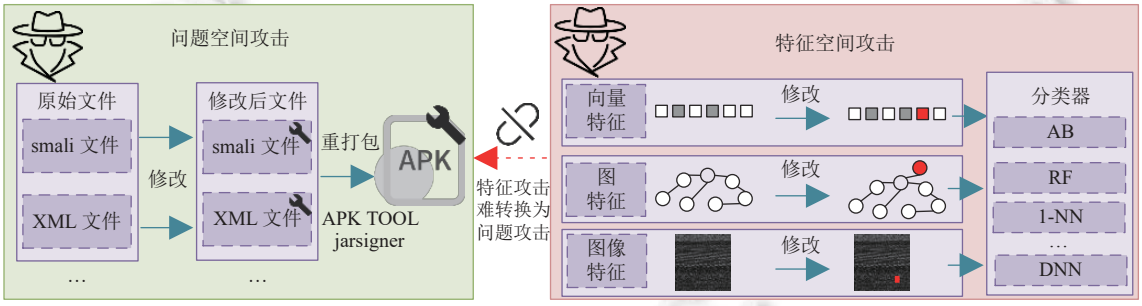

图 9 问题空间攻击与特征空间攻击

2.1.1 特征空间攻击

特征空间攻击的核心策略是修改从 Android 软件中提取的特征, 以欺骗或绕过恶意软件检测工具和机器学习模型. 这一攻击方法的重点在于特征的修改和处理, 攻击者试图通过变更这些特征, 使得 Android 恶意软件在恶意软件检测模型看来更像良性的应用, 从而逃避检测.

在特征空间攻击中, 攻击者的修改对象包括从 Android 应用程序文件中提取的各种特征, 这些特征可以涵盖不同领域的数据, 例如图数据[20]、0-1 向量数据[9]、图像数据[93]等. 对于特征空间攻击, 由于无须探究问题空间中复杂的代码逻辑、施加扰动的具体含义等, 攻击者通常可以借鉴图像处理领域中的工作[37,109], 将已有的图像对抗样本方法迁移到 Android 应用程序上. 尽管有部分攻击方法[59]将第 1.3.2 节提到的扰动约束以损失的形式加入对抗样本的训练过程中, 但是这些操作也仅仅在理论上保证了对抗扰动操作的可行性. 例如 Hu 等人[59]在针对语法

特征生成对抗样本的过程中, 限定只能向现有 APK 中增加函数调用/权限等操作, 但是并未深入讨论在问题空间中各个调用与权限之间的相互依存关系, 且并未检测对抗样本的恶意功能是否保持完整. Darwaish 等人[55]在攻击基于图像特征的 Android 恶意软件检测系统时, 仅仅从图像特征本身考虑对抗样本的产生方式, 并未在问题空间检测程序的功能完备性. 因此, 本工作仍然认为其属于特征空间的攻击.

2.1.2 问题空间攻击

问题空间攻击的核心策略是修改对抗样本原始代码, 生成真实的 Android 对抗样本. 有些攻击者通过特征攻击寻找到最优的特征扰动, 再通过各种 APK 逆向工程技术[110,111], 将特征上的扰动映射回问题空间上. 例如, Chen 等人[108]提出插入空函数或者无操作函数的方式在 smali 层面上实现扰动. Li 等人[61]在攻击函数调用图时, 通过 Try-Catch 陷阱的方式将扰动代码插入到 smali 代码中, 对修改后的源码进行重打包, 并对打包后的软件进行功能性校验. Pierazzi 等人[58]提出了非透明谓语的 smali 代码扰动方式, 实现了对函数调用图的修改. 有些攻击者利用已有的 smali 修改技术实现了问题空间中的攻击. 例如, Demontis 等人[52]使用代码混淆技术中的反射、加密等操作, Bostani 等人[112]通过程序移植技术, 同样产生了问题空间中的对抗样本. 还有些攻击者直接对问题空间进行建模, 利用基于优化的算法直接求解问题空间中的扰动. 例如, Zhao 等人[54]提出了结构化对抗样本生成方法, 提出了 4 种 smali 代码修改方式, 并使用深度强化学习技术找到最优操作方法. 本文将在第 4 节详细介绍各种问题空间的扰动实现方式.

现有的问题空间扰动技术 (例如代码混淆、程序移植等) 往往会引入与攻击无关的额外扰动, 导致攻击效果下降. 因此如何对齐特征空间上的扰动和问题空间上的修改是未来的研究趋势之一.

一般而言, 对于问题空间的攻击难点主要可以概括如下.

(1) 逆特征映射问题: 特征空间攻击方法难以直接应用于问题空间攻击的主要原因在于逆特征映射. 由于数据从问题空间映射到特征空间的过程是不可逆、不可微的, 难以真正将特征空间的扰动反映到问题空间中, 因此大多数情况下只能要求问题空间中的修改与特征空间上的扰动越接近越好.

(2) 问题空间约束: 为了保证修改后的恶意软件仍然能正常运行并执行恶意行为, 本文总结了 4 类针对问题空间攻击的约束. 1) 修改限制. 攻击者在问题空间中可执行的修改受到了严格的限制. 在针对语法特征的攻击方法中, 为了避免程序的崩溃, 常常规定只能向现有的程序中增加函数调用/权限/动作等[59,113]. 在针对函数调用图等语义特征进行攻击时, 也常常只允许向现有程序中增加函数调用关系[61]. 尽管 Zhao 等人[54]通过对某个函数源码的整段移植达到了删除某个函数的目的, 但是仍然保留了该函数的功能. 2) 语义保留. 在对问题空间进行修改时, 除了保证程序的正常运行外, 还需要让恶意软件的正常语义不受到破坏并保留下来. 3) 合理性. 在对问题空间进行攻击时需要满足修改的合理性. 程序的源代码必须符合正常的程序开发逻辑, 而不是人为刻意修改过的. 程序移植[71]技术通过从良性样本上截取部分代码段, 从而很好地解决了该问题. 4) 鲁棒性. 对数据预处理的鲁棒性. Android 恶意软件检测过程中往往会去除程序中的死代码或空函数, 而问题空间中的攻击需要能抵御这种预处理. 例如, Chen 等人[108]的工作中, 向现有 smali 文件中插入空函数, 虽然实现了对函数调用图的修改, 但是这些空函数却极易被检测并滤除.

(3) 副效应特征[58]. 由于问题空间攻击需要保证语法的完整性, 因此常常会引入很多额外的扰动, 导致特征空间与问题空间之间的映射更加复杂. 这些冗余的问题空间操作会在特征空间上产生若干具有副效应的特征. 这些特征不遵循任何特定的梯度方向, 因此可能对攻击效果产生负面的影响.

2.2 基于攻击者所掌握的信息

不同于图像分类场景, 在 Android 恶意软件检测系统中存在更多的先验知识可以提高攻击者的攻击效率, 因此可以根据攻击者对于 Android 恶意软件检测系统所拥有的知识和能力将攻击方法分类为: 白盒攻击、灰盒攻击和黑盒攻击.

2.2.1 白盒攻击

由图 6 可知, Android 恶意软件检测的流程包括问题空间中的 Android 特征提取以及在特征空间中的基于机器学习的样本分类. 在白盒攻击中, 攻击者可以获得目标检测系统使用的特征以及检测系统使用的模型的全部信

息, 包括模型结构、参数, 因此攻击者往往可以通过计算模型梯度来生成 Android 恶意软件的对抗样本.

2.2.2 灰盒攻击

在许多实际场景下, Android 恶意软件检测模型部署在云端时, 攻击者仅能获取到分类器的部分信息, 因此在本文中规定, 只要获取到了如下信息中的一部分, 都被定义为 Android 对抗样本的灰盒攻击.

(1) 特征

Android 恶意软件检测模型通常使用的特征包括应用程序的权限请求、API 调用序列、控制流特征、应用组件、用户接口交互图. 这些特征用于分析和识别应用程序的行为模式和潜在的恶意活动. 针对恶意软件使用的不同特征, 攻击者需要修改的文件与需要使用的攻击算法也不尽相同. 例如在攻击权限特征时, 需要对清单文件进行修改, 同时, 由于权限特征往往使用 0-1 向量表示, 需要在构造对抗样本攻击算法时考虑扰动的取整操作对攻击效果的影响. 现有的算法往往需要对目标模型使用的特征有较强的假设, 但是近年来有些工作旨在部分特征或者完全未知特征的场景下进行攻击. 例如, Li 等人[61]在对函数调用图特征的攻击过程中设定了未知函数调用图的粒度信息. 值得注意的是, 部分工作[50,52]通过代码混淆等手段也实现了未知特征场景下的攻击, 但是由于代码混淆技术本身的复杂性, 其与对抗样本产生的耦合性仍需进一步探索.

(2) 数据集

Android 恶意软件检测模型的效果往往在已经公开的著名数据集或在自建数据集上进行验证. 例如, 著名的 Android 恶意软件数据 Drebin[9], 它包含了数千个恶意和良性的 Android 应用程序样本, 用于研究和开发恶意软件检测模型. 这个数据集提供了详细的应用程序特征信息, 如权限、API 调用序列、应用程序行为等. 一旦攻击者知道了目标模型使用的数据集, 攻击者可以利用该数据训练一个与目标模型类似的替代模型, 利用对抗样本的迁移性, 通过在替代模型上产生对抗样本以产生能够逃脱目标检测模型的对抗软件. 例如, Chen 等人[108]在对攻击算法的性能进行测试时, 按照攻击者是否知道目标模型的训练集进行了设定.

(3) 模型信息

最后一个能够增加对抗样本攻击成功率的信息是攻击者是否能够获取目标模型的所有相关信息, 包括模型结构、超参数等, 比如隐藏层的维度、用户训练模型时使用的学习率等. 在 Android 对抗样本攻击中, 知道这些信息对于攻击者是非常有利的, 因为攻击者可以根据这些信息设计一个与目标模型更加接近的替代模型, 从而使得产生的对抗样本更具有迁移性.

2.2.3 黑盒攻击

黑盒攻击是指攻击者对目标模型或系统的内部结构和参数一无所知的攻击方式. 具体来说, 攻击者只能通过模型的输入和输出来进行攻击, 而无法获取模型的权重、结构或其他详细信息. 这种攻击方式模拟了现实世界中的情况, 攻击者只能观察到模型的行为 (即输入 APK, 输出检测结果), 但无法获得检测模型的具体细节. 例如, He 等人[64]在攻击过程中突破性地设定了攻击者对目标模型使用的特征信息完全未知. Li 等人[50]通过将代码混淆等各种攻击手段集成也实现了黑盒对抗样本的攻击.

2.3 基于被攻击模型使用的特征

由于目标模型使用特征不同, 攻击者产生对抗样本使用的手段也不同. 因此, 依据现阶段主流的 Android 恶意软件使用特征, 可以将 Android 对抗样本算法分为: 针对向量特征的攻击、针对图特征的攻击、针对图像特征的攻击, 如图 9 右侧框图所示.

2.3.1 向量特征

向量特征往往是一个由 0-1 值组成的离散数据, 因此针对图像这类连续数据的对抗样本攻击方法往往并不适用. 在对基于向量特征的 Android 恶意软件检测模型进行攻击时, 需要考虑的一大难点在于梯度的求解. 不同于连续数据, 离散数据的梯度往往难以估计, 同时, 离散数据在进行取整等操作时容易改变精心设计的扰动数值, 使得攻击成功率受到影响.

2.3.2 图特征

图数据是一种非欧几里得数据. 针对图数据的 Android 对抗样本攻击技术的难点主要如下. (1) 复杂性. 图数

据通常具有复杂的结构, 包括大量的节点和边以及多层次的连接关系. 例如, 在函数调用图特征中会有成千上万的节点与边, 攻击者需要考虑这种复杂性, 以生成有效的对抗样本. (2) 离散性. 图数据是离散的, 每个节点和边都具有离散的属性或标签.

2.3.3 图像特征

鉴于深度学习在图像领域中取得的优秀成果, 基于图像特征的 Android 恶意软件检测系统将 Android 软件的 DEX 等文件转化成图像数据, 随后利用深度学习算法对其进行判别. 然而, 图像领域中的大多数对抗样本技术无法迁移到 Android 领域中, 攻击基于图像特征的恶意软件检测系统. 这是因为 Android 对抗样本攻击需要维持 Android 软件的功能不受影响, 如果随意地对 DEX 等文件进行修改, 尽管扰动在人肉眼上无法看出, 但是却会对文件代码层面上造成灾难性的影响, 破坏恶意软件原有的功能.

由于针对不同特征的 Android 对抗样本攻击算法差异巨大, 因此本文将在第 3 节按照攻击者所针对的不同特征详细介绍不同的攻击算法.

3 Android 对抗样本攻击算法

本节根据被攻击模型使用的特征对现有工作进行介绍与对比. 表 2 对 Android 对抗样本攻击代表性工作进行总结, 概括不同攻击方法的特点, 包括攻击特征、拥有的先验知识、目标检测模型使用的特征、对抗样本产生思路、攻击空间以及使用的数据集. 表中的目标特征提取方法是指目标模型使用的特征类型, 数据集来源是指攻击算法与目标模型使用的训练与测试数据集合. 同时, 有些特征与数据集名称相同, 例如, Drebin 数据集与 Drebin 特征是由同一篇文献提出的, 因此名称一致.

表 2 典型 Android 对抗样本攻击算法比较

攻击方法	攻击特征	已有知识①	目标检测特征提取方法	对抗样本产生思路	攻击空间②	数据集来源
Demontis[52]	0-1向量特征	白盒攻击、灰盒攻击、黑盒攻击	Drebin[9]	梯度计算	特征空间/问题空间	Drebin、Contagio
Grosse[63]	0-1向量特征	白盒攻击	Drebin	梯度计算	特征空间	Drebin
Hou等人[53]	0-1向量特征、图特征	白盒攻击	Drebin, MaMaDroid[20]	梯度计算	特征空间	自建数据集
SPA&MPA[56]	0-1向量特征	白盒攻击、灰盒攻击	自建特征	启发式算法 (Q-learning[114])	特征空间	Drebin与自建数据集
TLAMD[62]	0-1向量特征	灰盒攻击	Drebin	启发式算法 (genetic algorithm[115])	特征空间	Drebin
OFEI[116]	0-1向量特征	灰盒攻击	Drebin	启发式算法 (simulated annealing algorithm[117])	特征空间	Virusshare, Contagio
TRPO-MalEAttack&PPO-MalEAttack[118]	0-1向量特征	灰盒攻击	自建特征	启发式算法 (TRPO[119]&PPO[120])	特征空间	Drebin
Pierazzi[58]	0-1向量特征	白盒攻击	Drebin	启发式算法 (greedy algorithm[121])	问题空间	自建数据集
UAP[122]	0-1向量特征	灰盒攻击	Drebin	启发式算法 (greedy algorithm)	问题空间	Drebin
MalGAN[59]	0-1向量特征	灰盒攻击	自建特征	生成对抗网络	特征空间	自建数据集
E-MalGAN[113]	0-1向量特征	灰盒攻击	BLS[14]	生成对抗网络	特征空间	自建数据集
Shahpasand[51]	0-1向量特征	灰盒攻击	Drebin	生成对抗网络	特征空间	Drebin
p-MalGAN[123]	0-1向量特征	黑盒攻击	自建特征	生成对抗网络	特征空间	DefenseDroid
MRV[57]	0-1向量特征	黑盒攻击	AppContext[124], Drebin	模仿攻击、混淆攻击	问题空间	Genome, Contagio, VirusShare, Drebin中随机挑选

表 2 典型 Android 对抗样本攻击算法比较 (续)

攻击方法	攻击特征	已有知识①	目标检测特征提取方法	对抗样本产生思路	攻击空间②	数据集来源
Rafiq[125]	0-1向量特征	灰盒攻击	自建特征	模仿攻击	特征空间	KronoDroid[126]
FSASG[127]	0-1向量特征	灰盒攻击	自建特征	模仿攻击	特征空间	自建数据集
i-bit[128]	0-1向量特征	灰盒攻击	自建特征	模仿攻击	特征空间	Drebin
Mixture of attacks[50]	0-1向量特征	白盒攻击、灰盒攻击、黑盒攻击	Drebin	集成学习	特征空间/问题空间	Drebin与自建数据集
Android HIV[108]	0-1向量特征、图特征	白盒攻击、灰盒攻击	Drebin, MaMaDroid	梯度计算	问题空间	MaMaDroid与自建数据集
HRAT[54]	图特征	白盒攻击	Malscan[85], MaMaDroid, APIGraph[24]	启发式算法 (deep Q-learning[129])	问题空间	Malscan
BagAmmo[61]	图特征	灰盒攻击	APIGraph, MaMaDroid, GCN[130]	启发式算法 (gnetic algorithm)	问题空间	Drebin、FalDroid和自建数据集
EvadeDroid[112]	0-1向量特征、图特征	黑盒攻击	Drebin, MaMaDroid	启发式算法 (greedy algorithm)	问题空间	Pierazzi[58]
AdvDroidZero[64]	0-1向量特征、图特征	黑盒攻击	Drebin, MaMaDroid, APIGraph	启发式算法	问题空间	Pierazzi[58]
Darwaish[55]	图像特征	白盒攻击	DroidDector[131], DeepClassifyDroid[132]	模仿攻击、生成对抗网络	特征空间	自建数据集
Gu等人[60]	图像特征	灰盒攻击	Gray image[95]	启发式算法 (differential evolution[133])	问题空间	Drebin

注: ① 和图像领域对抗样本不同, 在判断某一攻击是否属于黑盒时, 除了根据攻击者是否获知模型本身的信息 (如模型结构与参数) 还需要判断攻击者是否了解模型的输入特征. 因此, 尽管部分论文在其文中讨论时, 从传统图像领域对抗样本的角度认定其为黑盒攻击, 但是只要其获取了分类模型的任何信息 (例如, 使用的特征或者训练数据集) 在本文中都认定其为灰盒攻击.

② 尽管部分论文从理论上考虑了如何使攻击在问题空间可行, 但是只要文中没有在问题空间上实现APK的修改, 则认为其属于特征空间攻击.

3.1 面向向量特征的对抗样本攻击 (vector-based attack)

由于向量特征便于提取、训练方便, 因此被广泛用于 Android 恶意软件检测领域, 同时也是大多数 Android 对抗样本攻击算法的目标. 为了便于梳理, 本节根据攻击算法不同, 将向量特征的对抗样本攻击分为 3 类: 基于梯度的攻击、基于对抗生成网络的攻击、基于启发式算法的攻击.

3.1.1 基于梯度的攻击

基于梯度的攻击常见于白盒攻击或者灰盒攻击, 攻击者直接利用目标模型的梯度或者替代模型的梯度来计算扰动. 因此, 这一类算法的关键问题在于损失函数的设定.

Demontis 等人[52]将对抗攻击建模成如下优化问题:

$$Z^* = \text{argmin}_{Z'} f(\Phi(Z')) = \text{argmin}_{Z'} w^{\mathrm{T}} x',$$

其中, $x' = \Phi(Z')$ 是待修改的攻击样本 Z' 对应的特征向量, w 是分类模型 f 的权重向量. 然后根据攻击者的拥有的知识讨论了模仿攻击、有限知识攻击和完全知识攻击这 3 种不同的对抗攻击场景. 由于在计算机视觉领域中对抗样本技术的发展, 部分工作迁移了计算机视觉 (computer vision, CV) 领域中对样本的攻击算法. 例如 Grosse 等人[63]提出了一种白盒 Android 恶意软件对抗样本生成算法, 利用 JSMA[109]方法生成 Android 恶意软件对抗样本. 为了能将特征空间的修改映射回原始输入空间且保持修改前后软件功能一致性, 他们认为在原始 smali 文件中添加代码可能会引起一些无法预料的错误, 所以仅允许在 manifest.xml 文件中增加相关的特征, 这些限制导致对抗样本生成变得更加困难.

上述工作为不同 APK 样本生成不同的扰动. Hou 等人[53]首次在 Android 恶意软件对抗样本领域提出了通用

对抗扰动. 该通用扰动可以插入不同的恶意软件以生成对抗样本, 并使其中大多样本都能够躲避分类器的检测. 他们将通用扰动的求解转换为如下问题:

$$\|u\|_p \leqslant \varepsilon,$$

$$\rho_{x\sim\eta}(F(x+u)\neq F(x))\geqslant \alpha,$$

其中, ε 代表扰动 u 的尺度约束, x 是原始恶意样本的特征向量, η 代表恶意软件的分布, F 是恶意软件检测器, ρ 表示一个统计函数, α 代表扰动攻击成功率. 他们对单个对抗恶意软件扰动的迭代累积更新过程进行修改, 进而提出了通用扰动生成算法.

3.1.2 基于启发式算法的攻击

基于梯度的对抗样本攻击算法可以准确地找到特征空间中的扰动位置, 但是其需要准确的分类模型先验知识, 例如模型结构、参数等. 在这些先验未知的前提下, 攻击者可以使用基于启发式算法的攻击, 通过与目标模型的查询进行训练, 从而得到 Android 恶意软件对抗样本. 常见的启发式算法有以 Q-learning[114]为代表的强化学习、遗传算法[115]、模拟退火算法[117]、贪婪算法[121]等. 这类方法的关键在于对攻击过程的建模, 本文将启发式 Android 对抗样本攻击算法对比展示在表 3 中.

表 3 启发式 Android 对抗样本攻击算法对比

算法类型	群体/单体	新对抗样本的产生方式	核心思路	代表性工作
遗传算法	群体	对扰动进行交叉/变异/选择操作	选择阶段保留最优的扰动, 交叉变异阶段尝试新的扰动	文献[62]
模拟退火	单体	构造偏移量, 加到当前对抗样本上	搜索到更好的扰动一定采纳, 搜索到差的解以概率接受	文献[116]
Q-学习	单体	根据Q表选择/随机选择扰动	根据环境的反馈来获取奖励, 使攻击者能够在环境中取得最大的累积奖励的扰动	文献[56]
贪婪算法	单体	每次选择当前环境下的最优扰动	通过选取局部最有利的选择, 从而最大化收益	文献[58]

Rathore 等人[56]基于强化学习算法 Q-Table[114], 针对白盒和灰盒两种场景提出了单策略攻击和多策略攻击两种攻击算法. 他们将对抗样本生成过程建模为马尔可夫决策过程 (Markov decision processes, MDP)[134], 以 Android 恶意软件分类器作为强化学习中的环境, 软件提取的特征向量作为状态, 可修改的特征集合作为智能体的动作集合, 将扰动量与对目标模型的攻击效果作为奖励函. Xu 等人[116]提出了一种灰盒场景下恶意软件对抗样本生成算法 OFEI 攻击深度学习即服务 (deep learning as a service, DLaaS)[135,136]系统. 为了保持功能一致性, 他们在攻击时只进行特征添加操作. 他们使用模拟退火算法[117], 搜索 Android 恶意软件特征空间中的最佳扰动位置. Rathore 等人[118]基于强化学习中的 TRPO 算法[119]与 PPO 算法[120]提出了两种 Android 对抗样本攻击算法: TRPO-MalEAttack 和 PPO-MalEAttack, 利用对抗样本在目标模型上被分类为良性的概率作为奖励值, 概率越高则奖励值越高.

对抗样本技术不仅仅出现在针对手机端 Android 恶意软件检测的攻击中, 许多物联网 (Internet of Things, IoT) 系统或类似 Android 的系统[137−141]中也存在类似的攻击场景. 对于物联网设备, Liu 等人[62]提出了一种基于遗传算法[115]的 Android 恶意软件对抗样本生成技术.

上述启发式算法主要在特征空间中进行, Pierazzi 等人[58]提出了一种问题空间上的启发式攻击算法. 他们利用自动化程序移植技术[71,111], 从良性软件中提取多个字节码片段, 使用基于贪婪算法[121]的梯度驱动搜索策略, 将字节码片段注入恶意软件, 从而逃脱系统检测. Labaca-Castro 等人[122]对恶意软件特征空间和问题空间的通用对抗扰动以及迁移性进行分析, 并提出了一种针对恶意软件问题空间的通用对抗扰动生成算法. 他们使用贪婪搜索算法[121]与程序移植技术, 从候选集合中选择扰动操作, 直到算法生成的通用扰动使得模型达到指定的误分类率, 或者扰动达到最大长度.

3.1.3 基于生成对抗网络的攻击

上述大部分工作为不同样本产生对抗扰动时, 需要重复计算梯度或者重新训练启发式算法. 然而基于生成对抗网络的方法使得模型训练完成即可直接用于不同样本上, 无须针对新的样本重新训练. 基于生成对抗网络的攻击常见于白盒或灰盒攻击, 其需要知道模型的输入特征. 该类方法需要不断与目标检测模型进行交互, 通过查询的

结果指导训练生成模型. 一旦生成模型训练成功, 对于后续输入的样本则无须进一步地查询, 即可用过生成模型直接生成扰动或对抗样本. 主流的基于生成对抗网络的攻击方法模型如图 10 所示.

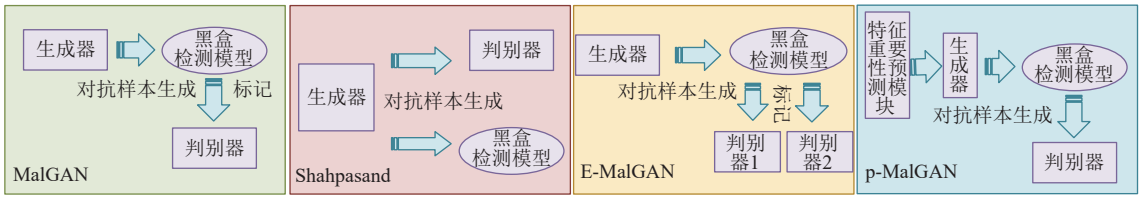

图 10 基于 GAN 模型的 Android 对抗样本生成方式对比

Hu 等人[59]首次提出了一种基于 GAN 的灰盒 Android 恶意软件对抗样本生成框架 MalGAN. 在其模型中, 一个用于拟合黑盒目标模型的替代模型作为判别器, 一个生成恶意软件对抗样本的网络作为生成器, 两者对抗训练以达到最小化目标系统分类准确率的目的. 本地替代模型的损失函数如下:

$$L_D = -E_{x\in BB_{\text{Benign}}}\log(1-D(x))-E_{x\in BB_{\text{Malware}}}\log(D(x)),$$

其中, D 表示判别模型, BB_{Benign} 表示被目标检测模型判别为良性的样本, BB_{Malware} 表示被目标检测模型判断为恶意的样本. 生成模型的损失函数为:

$$L_G = E_{m\in S_{\text{malware}},z\sim P_{uniform[0,1)}}\log D(G(m,z)),$$

其中, z 是噪声向量, $uniform[0,1)$ 表示一个从 0 至 1 的均匀分布, S_{malware} 表示恶意软件集合, G 表示生成模型.

与文献 [59] 类似, Shahpasand 等人[51]提出利用生成对抗网络去生成对抗扰动, 生成网络的输入为噪声序列, 输出为与样本特征相同维度的扰动. 为了让生成的对抗样本与良性样本更为相似, 判别器尽量区分干净良性样本和恶意对抗样本. 生成器的损失由两部分构成, 一部分损失使得产生的对抗样本尽量欺骗判别器, 另一部分损失使得生成的对抗样本让目标分类器的准确率尽可能降低. 不同于文献 [51,59], Li 等人[113]发现对抗样本与正常样本的分布存在差异, 可以通过部署一个对抗样本检测器来过滤对抗样本, 达到对抗防御的目的. 基于此, 他们在 MalGAN 的基础上进行修改, 通过新增一个判别模型, 指导生成器产生与正常样本更相近的对抗样本. Guo 等人[123]在 MalGAN 的基础上增加了一个特征重要性预测模块 (feature importance prediction, FIP) 并提出了 p-MalGAN 模型. 他们假设攻击者在不知道分类模型使用特征类型的情况下进行攻击. p-MalGAN 中的特征重要性预测模块使用随机森林作为替代模型, 通过测量特征与检测器标签之间的相关性来计算特征的重要性, 以预测检测器使用的特征.

3.1.4 其 他

除了上述的攻击算法类型, 还有一些算法利用混合策略、模仿攻击、代码混淆等手段实现对抗样本攻击.

将恶意软件的设计思路融入对抗样本的产生方法有助于产生更具迷惑性的对抗样本. Yang 等人[57]认为大多数恶意软件对抗样本生成算法都是从软件中提取特征, 从特征向量空间生成对应扰动, 这些生成算法可能缺乏实际可行性. 为了解决上述问题, 他们提出了恶意软件重组变种 (malware recomposition variation, MRV), 根据恶意软件的结构, 提出两种对应的变种策略, 恶意软件进化攻击和恶意软件混淆攻击. Li 等人[50]提出了一种集成多种攻击算法和修改操作集合的 Android 恶意软件对抗样本生成算法. 该攻击方法集成了基于梯度的攻击, 无梯度攻击和混淆攻击. 他们将对抗样本生成转换为求解如下问题:

$$h;M_x = \text{argmax}_{h;M_x}L(F(h(M_x;x)),y),$$

其中, M_x 表示对样本的扰动, h 表示一个攻击算法, F 表示目标模型或者替代模型, L 表示交叉熵损失. 攻击者最大化目标分类模型的损失函数, 迭代地在特征空间中优化扰动, 生成对抗样本. 为了保证特征空间的修改能转换到空间问题, 他们仅进行特征增量操作, 如仅在源文件中插入字符串和垃圾代码等.

模仿攻击是一种简单有效的攻击方式, 其剔除恶意软件中显著的敏感特征, 增加良性软件中常见的良性特征, 从而伪装成良性软件逃避检测. Rafiq 等人[125]针对 0-1 二进制特征提出了若干对抗样本攻击算法. 攻击者统计了良性软件与恶意软件中各特征的出现频率, 挑选了 30 个恶意样本中出现频率最高的特征以及 30 个良性样本中频

率最高的特征. 随后, 攻击者利用如下 3 种方法对恶意软件特征进行攻击: (1) 模仿攻击, 即向恶意样本逐个添加良性样本集合中频率最高的特征; (2) 移除攻击, 即恶意样本逐个移除恶意样本集合中频率最高的特征; (3) 混合攻击, 即混合上述两种攻击方法. Rathore 等人[128]提出了一种 Android 恶意软件对抗样本生成算法 i-bit, 其中的 i 代表允许最大修改的特征数目. 具体而言, 攻击者构造一个良性样本集合, 对于恶意样本, 依据余弦相似度, 找出与当前恶意样本最接近的良性样本, 逐位修改恶意样本特征使其靠近良性样本. Li 等人[127]提出了一种针对特征空间攻击的 Android 恶意软件对抗样本生成算法 FSASG. 具体来说, 他们利用卡方检验、信息增益和自己提出的特征频率这 3 种指标加权平均来对各类特征进行赋值排序, 根据排序挑选出良性样本典型特征集合列表. 在攻击时, 他们向恶意样本中按权重顺序逐个添加该集合列表中的特征.

3.2 面向图特征的对抗样本攻击 (graph-based attack)

与向量特征相比, 图特征往往具有更加复杂的结构, 因此更加难以被攻击. 具体来说, 向量特征通常是根据预先设定的某些函数、权限列表构造一个长度固定的向量值. 但是在图特征中, 不同样本根据其自身 APK 大小的不同, 构造的图特征大小不一. 针对此种特征进行攻击时, 若对图特征的图嵌入算法一无所知, 则难以使用基于梯度的攻击算法, 因此现有工作主要使用基于启发式的算法进行攻击.

Chen 等人[108]在攻击基于函数调用图的 Android 恶意软件检测系统时假设已知特征提取方式, 其利用图像领域中的 JSMA 算法[109]和 C&W 算法[37]. C&W 和 JSMA 是图像领域中的对抗样本攻击算法. 在 Android 领域, 他们将增加的函数调用的数量作为扰动, 重新定义二者的攻击算法.

传统的针对 FCG 的修改方式往往局限于插入函数调用, 极大地限制了攻击效果. Zhao 等人[54]提出了一种结构化的攻击算法 HART. 他们提出了 4 种新的函数调用图修改方法, 即增加函数调用、重写函数、插入函数和删除函数. 为了针对不同的 Android 恶意软件选择最优的扰动修改组合, 他们提出了一种基于深度强化学习的攻击策略. 该工作对函数调用图的扰动方法进行了扩充, 对于构造面向基于函数调用图的 Android 恶意软件系统对抗样本攻击具有重要意义.

Hamid 等人[112]提出了基于程序移植的对抗样本攻击算法 EvadeDroid. 他们假设攻击者仅仅拥有目标恶意软件检测器的黑盒访问权限, 即攻击者只能查询恶意软件检测器的输出结果, 以判断修改后的 Android 恶意软件是否成功地逃避了检测. 具体攻击流程如图 11 所示, 攻击者构造了一种基于 opcode[142−144]的软件相似性判别方法, 找到与待修改的恶意软件附近最相似的若干良性软件, 随后从相似的良性软件中切割代码片段作为扰动的候选. 随后攻击者通过随机搜索的优化方法和自动化程序移植技术, 通过反复与目标模型的交互, 找到这些代码片段中具有良性语义的代码段插入到恶意软件中, 达到生成对抗样本的目的.

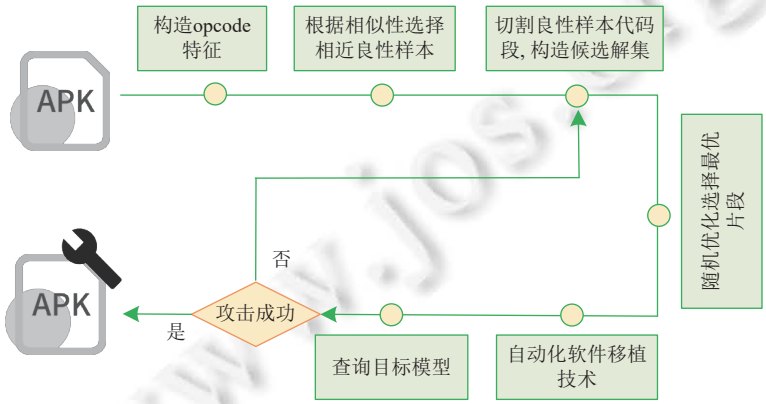

图 11 EvadeDroid 攻击流程

随着攻击技术的发展, 越来越多的工作着眼于有限知识场景下的攻击. Li 等人[61]在攻击基于 FCG 的 Android 恶意软件检测系统时, 假设攻击者并不知道所攻击的系统使用的 FCG 特征粒度. 基于此, 他们提出了一种多群协

同演化算法 BagAmmo, 攻击在非完全信息场景下的 Android 恶意软件检测系统. 具体而言, 如图 12 所示, 在构造 FCG 时, 一个函数可以被表示为不同的函数粒度 (如包粒度、类粒度、家族粒度), 导致构造的 FCG 大小不一. 当攻击者没有粒度信息时, 其设计的攻击算法往往无法收敛. 他们提出了一种多种群协同演化算法, 通过利用不同种群模拟不同粒度 FCG 的攻击过程, 并通过与目标模型之间的反复查询获得种群中不同个体的适应度函数, 执行优胜劣汰操作, 最终实现对任意粒度的函数调用图的对抗样本攻击.

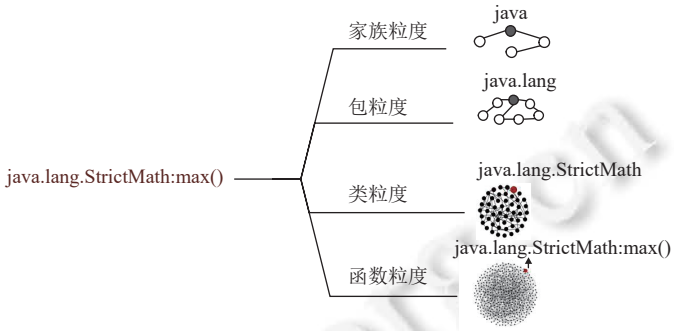

图 12 函数调用的多粒度表征

3.3 面向图像特征的对抗样本攻击 (image-based attack)

随着深度学习在图像领域的发展, 越来越多的工作将 Android 软件转换成图像来进行分类. 面向这类图像特征的 Android 恶意软件检测系统进行对抗样本攻击时, 其主要难点在于如何对问题空间进行修改.

Darwaish 等人[55]提出了两种针对图像特征的对抗样本生成算法. 在第 1 种方法中, 攻击者将已有的 Android 良性软件生成的图像特征以一定的比例加入恶意软件的图像特征中; 在第 2 种方法中, 攻击者训练了一个深度卷积生成对抗模型 (deep convolutional GAN)[145], 利用该模型不断生成伪造的良性软件图像特征, 并将其加到恶意软件的图像特征上. 该工作仅仅在特征空间中实现, 缺乏问题空间的可操作性.

Gu 等人[60]首次提出了问题空间中针对图像特征的 Android 对抗样本生成算法. 他们指出, 在对 Android 恶意软件的图像特征进行攻击时, 生成的对抗样本与原始图像的差异越小, 对其原始功能的影响就越小. 因此攻击者利用计算机领域的单像素攻击 (one-pixel attack)[39]对恶意软件的图像特征中可扰动的位置进行修改. 该算法对可改像素点的个数进行了限制, 利用差分进化算法[133]改变少数几个像素点的值, 即可完成对抗样本攻击. 如图 13 所示, 在对 Android 恶意软件的图像特征进行修改时, 攻击者先找到图像特征中可以扰动而尽可能不影响 APK 运行的位置 (即图 5 中的数据区域), 随后利用单像素攻击对图像特征进行扰动. 由于修改后的 DEX 文件像素不满足头部的校验和信息, 攻击者修复 DEX 文件的校验信息, 随后重签名并打包成为 APK 文件.

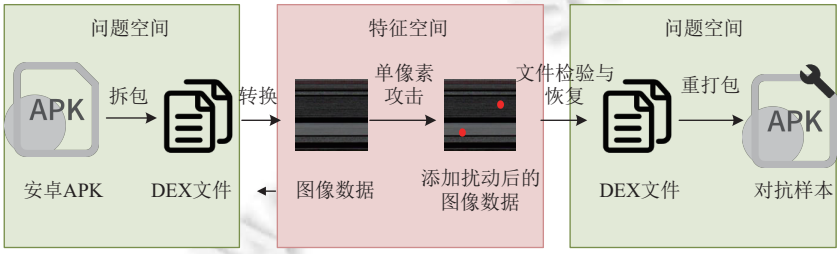

图 13 图像特征对抗样本攻击算法流程

3.4 未知特征攻击

除了针对某种特征提取固定的攻击方法, 有些工作着眼于更加困难的攻击场景, 即完全未知特征的对抗样本

攻击. 例如, Demontis 等人[52]提出使用 Android 混淆工具 DexGuard 来实施对抗样本攻击. Li 等人[50]提出的集成攻击中同样包含了代码混淆这一未知特征的攻击方法. 然而代码混淆这种攻击方式具有一定的盲目性, 导致攻击效果较低.

He 等人[64]提出了一种零知识的对抗样本生成算法 AdvDroidZero. 他们假设未知 Android 恶意软件检测模型使用的特征, 并从问题空间出发, 构造一系列的可扰动特征集合. 为了减少扰动搜索空间, 提高扰动生成效率, 他们将可行扰动通过语义相似性构造了扰动选择树. 同时, 他们提出了一种基于语义的调整策略调整扰动被选择的概率. 他们在多个 Android 恶意软件检测算法和检测引擎上进行评估, 实验结果表明了该方法的有效性.

3.5 总 结

现有工作的发展脉络归纳在图 14 中, 具体的分析如下.

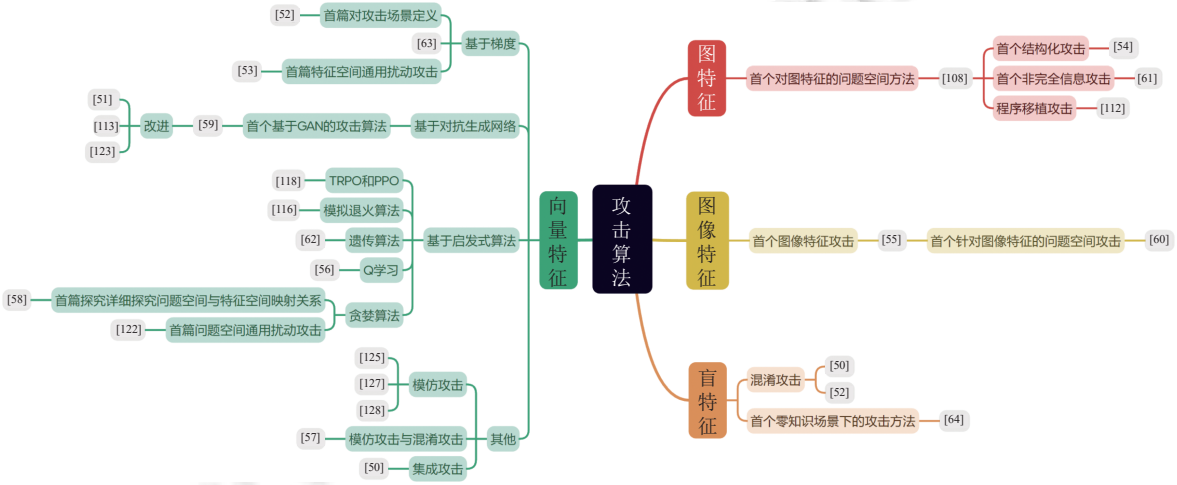

图 14 现有工作的对比

面向 0-1 向量特征的 Android 对抗样本攻击算法, 主要通过寻找特征中具有更高梯度的位置进行修改, 从而达到攻击的目的, 但是这种方法大多需要对不同的恶意软件重新进行查询. 而基于生成对抗网络的算法从另一种思想出发, 只需在训练阶段通过查询得到的数据训练生成模型. 一旦生成模型训练完成, 即可对未知的恶意软件执行无查询的攻击. 基于启发式的算法往往无需对模型细节的先验知识, 而是通过目标模型的返回值对最优扰动位置进行搜索求解, 然而这类方法受到搜索空间大小的限制, 一旦搜索空间过大, 算法的效率将大大降低.

在面向图特征数据进行对抗样本攻击时, 攻击者往往依赖于启发式的攻击算法, 通过迭代优化找到图数据中需要修改的位置 (或者需要插入的函数). 然而这种方法需要大量的查询次数, 容易引起防御者的警觉. 受此启发, 通用扰动算法的提出使得攻击者可以生成一个通用扰动, 对不同的恶意软件均能起到攻击的作用, 从而对未知的恶意软件无需大量的查询即可执行对抗攻击.

在面向图像特征的数据进行对抗样本攻击时, 对抗样本生成的难点在于如何将特征空间扰动映射到问题空间中, 尽管单像素攻击的提出使得攻击者在 DEX 文件上的修改尽可能小, 但是仍有可能使得修改后的对抗恶意软件在运行时崩溃, 因此面向图像特征的问题空间对抗样本技术是未来的研究重点.

随着研究的进行, 越来越多的攻击者研究如何在未知特征的零知识场景下进行攻击, 其难点在于如何通过查询快速确定目标模型使用的特征, 并使得本地的攻击算法快速收敛.

4 问题空间攻击实现手段

第 3 节主要分析了 Android 对抗样本生成算法技术上的内容. 本节将聚焦于如何将这些扰动通过逆向工程插入原始 APK 中. 不同于图像领域中直接对图像的像素值进行扰动, Android 恶意软件的问题空间攻击涉及代码层

面的修改, 属于 Android 领域中的独特要求. 在进行扰动求解算法时往往就需要考虑问题空间操作, 进而对算法的求解过程进行约束, 当扰动求解完成后再对扰动进行问题空间的实现. 由于在第 2.1.2 节中已经介绍了不同的映射方法, 因此本节着重介绍不同问题空间攻击方法的实现手段. 本文将问题域攻击实现手段按照修改粒度的不同分为粗粒度的修改和细粒度的修改. 粗粒度的修改的幅度较大, 往往以代码片段为最小粒度, 而细粒度的修改往往较为具体, 最小粒度可以精细化到具体的函数. 图 15 展示了问题空间攻击实现手段的分类.

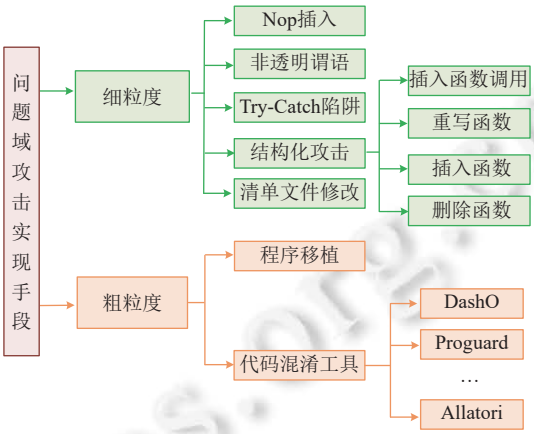

图 15 问题空间攻击实现手段分类

4.1 细粒度修改

细粒度的修改往往针对某一个具体函数, 其难点在于确保修改过程中函数的原始功能不受影响.

4.1.1 插入无操作调用 (non-operation call, Nop Call)

一个直观的可行操作即是向 smali 文件中插入无操作函数, 例如空函数、log 函数等[108,130]. 这些函数在运行过程中并不影响原始 APK 的运行, 但是却能向函数调用图增加新的函数调用, 从而修改图特征. 一个典型的空函数与 log 函数的例子如图 16 所示. 在图 16(a) 中, 通过构造空函数 *callee*, 即可实现 *caller* 函数对 *callee* 函数的调用. 在图 16(b) 中, 通过静态调用 (invoke-static) 实现 log 函数的插入.

```
package android.os.mypack

public class Myclass{
    public static void callee() {}
    public static void caller() {
        callee();
        callee();
}
}
```

(a) 插入空调用

```
const-string p0,""
const-string p1,""

.line 13
invoke-static{p0,p1},
Landroid/util/Log;->d(Ljava/lang/
String;
Ljava/lang/String;)I
```

(b) 插入log函数

图 16 插入无操作调用示例

4.1.2 非透明谓语 (opaque predicate)

由于上述操作仅插入某些特定的函数. 基于非透明谓语的函数调用修改方式解决了这一问题[58,112], 在实现了插入有意义函数的同时保证了函数不被调用. 为了保证插入的函数能够躲避静态分析的检测, 非透明谓语精心构造混淆条件, 这些条件在设计时就已知结果, 但在静态分析期间却使防御者难以确定实际真值. 为了方便理解, 给出一个简易的非透明谓语例子, 如图 17 所示. 图中代码实现了 100 个随机布尔值的或操作, 显然, 操作的结果仅有 $1/2^{100}$ 的概率为 false, 但是静态分析无法确定该操作的真值, 从而无法将 if 中插入的扰动语句过滤.

```
package com.example.mb                                    0
public class selfDefined {                                1
   public static void myfunc(){                          2
      Random random=new Random();                         3
      boolean result=false;                               4
      for (int i=0; i<100; i++){                          5
         boolean randomBool=random.nextBoolean();         6
         result=result || randomBool;                     7
      }                                                   8
      if(result==false){                                  9
          callee1();                                      10
          callee2();                                      11
          ....                                            12
   }                                                      13
}                                                         14
```

图 17 非透明谓语示例

4.1.3 Try-Catch 陷阱

非透明谓语引入了 *random* 函数这一额外的函数调用, 因此可能降低对抗样本的攻击效率, Try-Catch 陷阱[61]的提出很好地解决了这一问题. 具体来说, 攻击者将 Try-Catch 块插入到调用函数中, 并将扰动对应的被调函数语句放在 Try 模块中. 然后, 在被调函数前面添加若干异常语句. 这些异常语句用于触发运行时异常 (例如, *ArithmeticException*). 该方法的作用机理如下: 首先, 它在 smali 文件中插入了一个函数调用语句, 从而通过添加新的调用来改变 FCG; 其次, 该函数调用语句永远不会执行, 因此不会影响恶意软件的原始功能. 如图 18 给出了一个 Try-Catch 陷阱示例代码, 攻击者通过数组越界操作设置了一个运行时异常. 该异常无法被静态分析检测, 却会在运行过程中报错, 从而使得程序越过插入的函数, 保证原始恶意软件的功能不受影响.

```
package com.example.mb                                    0
public class selfDefined {                                1
   public static void myfunc() {                         2
      try{                                                3
           int[] array=new int[1];                        4
           int a=array[1];                                5
           callee1();                                     6
           callee2();                                     7
           ...                                            8
                }catch (Exception e){                     9
                                                          10
      }                                                   11
   }                                                      12
}                                                         13
```

图 18 Try-Catch 陷阱示例

4.1.4 结构化攻击

上述的所有对于函数调用的修改均是插入新的函数调用, 函数调用图修改手段有限, Zhao 等人[54]提出了一种结构化攻击算法, 实现了多种函数调用图的修改方式: 增加函数调用、重写函数、插入函数、删除函数.

(a) 增加函数调用. 假设原始的函数调用图中有两个函数 *caller* 与 *callee*, 这两个函数之间本没有调用关系, 增加函数调用则是在这两者之间增加 *caller* 到 *callee* 的调用. 具体的修改方式如图 19(a) 所示, 攻击者在 *callee* 中加入 FLAG 的条件表达式, 如果 FLAG 为 True 则进行一次空调用, 若 FLAG 为 False 则不影响原始 *callee* 函数的执行.

(b) 重写函数. 假设原来有函数调用 *caller* 到 *callee*, 重写函数通过增加中间函数 *imm*, 使得函数调用关系变为 *caller* 到 *imm*, *imm* 到 *callee*. 具体的修改方式如图 19(b) 所示, 对于 *imm* 函数, 在传参过程中加入 *callee* 函数需要的形参以及 FLAG 标识符, 如果 FLAG 为 True 则进行 *callee* 函数的调用, 反之则执行 *imm* 函数中原始的代码. 在 *caller* 函数中, 删除对于原始 *callee* 函数的调用, 并调用 *imm* 函数, 并设置 FLAG 为 True.

(c) 插入函数. 插入函数通过构造一个新的无意义函数 *callee*, 并在已有的函数 *caller* 中调用该函数来增加函数调用的同时保证函数功能不受影响. 具体的修改方式如图 19(c) 所示, 通过构造一个无实际意义的函数 *callee*

(图中该函数仅完成加法功能), 随后通过在主调函数 *caller* 中写入对 *callee* 的声明即可完成 *caller* 到 *callee* 的调用.

(a) 增加函数调用 (b) 重写函数 (c) 插入函数 (d) 删除函数

图 19 结构化修改方法示例

(d) 删除函数. 假设已有函数调用 *caller* 到 *callee*, 删除函数可以删除该调用. 具体的修改方式如图 19 (d) 所示. 攻击者从 *caller* 函数中移除目标函数 *callee* 的调用语句, 并将 *callee* 函数内的所有语句移植到 *caller* 函数中.

4.1.5 针对清单文件的修改

在 Android 应用程序开发中, Android 清单文件 (AndroidManifest.xml) 是一个重要的配置文件, 用于描述应用程序的基本信息、权限、组件和配置. 针对 Android 清单文件的修改主要包括权限的增加和组件声明的增加, 以下给出两者的具体定义.

权限声明: 清单文件列出了应用程序所需的权限, 例如访问互联网、读取手机状态、访问相机等. 这些权限决定了应用程序能够执行的操作.

组件声明: 清单文件定义了应用程序的各种组件, 包括活动 (activity)、服务 (service)、广播接收器 (BroadcastReceiver) 和内容提供程序 (ContentProvider). 这些组件是构成 Android 应用程序的核心部分, 清单文件指定了它们的名称、类型和属性.

在增加权限和组件声明时, 仅需遵循 Android 开发规则在 AndroidManifest.xml 文件中加入相应的权限和组件声明.

4.2 粗粒度修改

粗粒度的修改方式往往对整段的代码切片或者整个 DEX 文件进行修改. 该方法的难点在于修改过程中容易引入额外的副效应特征, 导致求解的对抗扰动无法精准地映射到问题空间中.

4.2.1 程序移植

程序移植[71,111,112]可以用来从良性程序 (又称为捐赠者) 中提取字节码片段, 并将它们注入到恶意宿主中, 以模仿良性应用程序的外观, 诱使学习算法错误地将恶意主机分类为良性. 程序移植过程包括两个步骤: (1) 代码段获取; (2) 代码段注入.

在代码段获取阶段, 攻击者试图获取良性软件中的一个良性功能, 以便后续移植到恶意软件中, 使得恶意软件看起来更类似于良性软件. 具体而言, 攻击者利用代码切片技术[146]从某个捐赠者中 (即某个良性应用中) 提取与某个良性功能对应的字节码片段, 获得包括该良性功能的入口点、前向切片和后向切片. 值得注意的是, 前向切片指从捐赠者中提取一组与特定功能相关的语句, 后向切片用于指构造入口点参数所需的所有语句.

在获取上述内容之后即可执行代码段注入. 攻击者在宿主 (即待修改的恶意软件本身) 中确定一个插入点, 该插入点的选择需要确保宿主程序的语法有效性; 其次, 它必须尽可能不被察觉, 以避免引起任何逻辑合理性方面的问题. 随后将上述获取的所有代码片段进行插入, 使得代码段注入后的程序在语法和逻辑上都能够融入宿主程序.

4.2.2 代码混淆工具

代码混淆技术[147−149]是一种旨在增加程序代码的复杂性和模糊性, 从而增强代码的安全性、降低逆向工程风

险以及保护知识产权的方法. 但是, Yang 等人[57]发现经过代码混淆后的恶意软件能够逃避恶意软件检测模型的检测. 因此, 代码混淆是一种极有应用前景的对抗样本扰动实现手段. 常见的代码混淆工具有 DashO (https://www.preemptive.com/products/dasho/overview)、Obfuscapk[150]、Allatori (https://allatori.com/) 和 Progud (https://www.guardsquare.com/proguard) 等, 常见的代码混淆技术包括重命名标识符、字符串加密等. 下面我们简要描述可以用于对抗样本生成的混淆技术.

(1) 标识符重命名. 标识符重命名将应用程序的包、类、方法和字段的名称更改为随机字符, 以增加代码的复杂性和模糊性. 标识符重命名还可以通过重命名一些不是入口点的应用程序组件来实现对清单文件进行修改.

(2) 字符串加密. 字符串加密可以加密应用程序中使用 const-string 指令定义的字符串, 为了保证这些字符串在解码时正常使用, 混淆过程中会添加一个额外的方法, 用于在运行时根据需要对这些字符串进行解密.

(3) 反射. 反射使用 Java Reflection API 来改变应用程序中的调用关系, 将原本的调用指令替换为 Java.lang.Reflect 类中的调用, 使得调用关系更加复杂和模糊.

(4) 类加密. 类加密将应用程序中除入口点类以外的所有类进行加密, 加密的类在运行时会被解密.

4.3 总 结

根据修改的粒度不同, 现有的扰动实现方法可以分为粗粒度扰动与细粒度扰动. 这些扰动方式与选择攻击的特征有一定的关系. 例如在攻击 0-1 特征时, 往往不需要对函数调用结构进行精细化的修改, 因此使用细粒度修改中插入无操作调用的方法或使用粗粒度修改方法会更加高效. 然而在攻击基于 graph 特征的恶意软件检测模型时, 细粒度攻击方法中的结构化攻击往往因为能更精细化地修改函数调用关系, 从而取得更好的攻击效果.

细粒度修改方式可以定位到某个函数进行细致化的修改: Nop 插入的修改方式简单便捷但是能够修改的函数有限. 非透明谓语和 Try-Catch 陷阱的方式虽然能够插入任意的函数, 但是只能向已有的函数调用图中增加调用. 而结构化攻击方法通过 FLAG 标志位实现了图数据的增删改, 面向清单文件进行修改较为简单, 可以直接向 XML 文件中增加需要的权限声明或者组件声明.

粗粒度的修改方式较为复杂, 可以实现大规模的文件修改, 但是容易引入副效应特征, 导致攻击的成功率下降. 其中, 程序移植实现了将良性软件中的功能移植到恶意代码中, 代码混淆的操作让恶意行为更加模糊, 两者都能在一定程度上增加恶意软件的隐蔽性, 但是都难以精准计算扰动.

所有的问题空间攻击实现手段、主要思想、优缺点以及代表性论文如表 4 所示.

表 4 问题空间攻击实现手段总结

问题空间攻击实现手段	主要思想	优点	缺点	代表论文
Nop插入	插入无意义函数	操作简单、便捷	可插入的函数有限, 且易被静态分析过滤	[108]
非透明谓语	构造静态分析无法解析的混淆条件, 实现函数的插入	抗静态分析	易引入额外扰动	[58,112]
Try-Catch陷阱	构造运行时异常, 并利用Try-Catch来实现函数的插入	抗静态分析、扰动精确	仅能增加函数调用	[61]
结构化攻击	利用FLAG形参实现多种函数调用图修改方法	实现了函数调用的增删改	操作复杂	[54]
程序移植	利用程序移植与代码切片技术, 实现良性软件代码片段的移植	良性样本代码的自动移植	扰动不可控, 易引入额外扰动	[57,112,122]
代码混淆	通过代码混淆技术, 增加恶意软件中恶意代码的模糊性	混淆手段多样、混淆工具完备, 实现了代码段的批量修改	扰动不可控, 易引入额外扰动	[50,57]

5 现存挑战与未来趋势

正所谓“知己知彼, 百战不殆”, 研究对抗样本攻击技术是设计鲁棒性 Android 恶意软件检测模型的基础, 为后续的鲁棒性恶意软件检测模型的构建提供更好的数据资源、评价指标. 目前工作虽然在 Android 对抗样本的攻击

研究上取得了显著进展, 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. 本节对这些问题进行梳理, 并对未来的研究方向进行展望.

5.1 对抗样本技术发展趋势

5.1.1 针对恶意软件对抗样本攻击的评估体系

现有的 Android 恶意软件攻击效果的评估体系并不统一. 不同的 Android 恶意软件攻击算法往往在不同的数据集上进行测试, 这些数据可能来自著名 Android 数据集, 例如 DREBIN、FalDroid 等, 也有可能是攻击者自己从 Virustotal、Google 应用市场等著名平台上爬取而来, 因此并没有一个权威的数据集对不同对抗样本攻击技术进行统一的评估. 此外对抗样本攻击算法的评估标准也不尽相同, 对于函数调用图的攻击往往是通过添加的函数调用数, 或者使用添加的函数调用数与整个 APK 所使用的函数调用数的比值进行衡量. 而针对权限特征的攻击则是根据所增加的权限个数进行衡量. 因此, 亟需一个健全而统一的评估体系以验证不同 Android 恶意软件攻击的有效性.

5.1.2 提高黑盒攻击效率

在攻击黑盒的 Android 恶意软件检测系统时, 攻击者往往通过构建本地替代模型或者通过对目标模型进行查询的方式构造对抗样本. 由于替代模型与目标恶意软件检测模型之间的差异, 导致前者的攻击成功率降低, 因此亟需研究缩小本地模型与目标模型之间差异的算法. 后者往往需要大量的与目标模型的通信, 这个过程涉及 APK 文件的反复重打包与上传操作, 攻击效率低下. 同时, 云端检测模型往往会监控用户本地上传的恶意软件数量, 反复上传修改的 APK 文件容易引起检测系统的警觉, 因此亟需研究减少攻击者对目标模型查询次数的攻击方法. 在未来, 结合使用模型窃取攻击技术与对抗样本攻击算法可能是一个重要研究方向, 模型窃取可以高效地窃取云端模型的模型参数, 借助模型窃取技术, 有望通过少量的试探与查询实现 Android 恶意软件检测模型使用的特征与模型.

5.1.3 优秀的代码操作方法

尽管现有的攻击算法针对函数调用特征的扰动已经提出了很多优秀的代码扰动操作方式, 例如非透明谓语、Try-Catch 陷阱、结构化攻击等. 但是针对图像特征的扰动操作方式仍然有限, 如何将特征中的扰动映射到问题空间中仍然是一个悬而未决的问题. 因此, 需研究优秀的代码扰动操作方式, 既不影响函数的原有功能, 又能保证扰动很难被防御者发现或消除. 代码混淆技术对 Android 恶意软件对抗样本攻击算法有一定的借鉴意义, 因此, 深入研究代码混淆与对抗样本问题空间的操作上的相似性对拓展问题空间的攻击手段有积极意义.

5.2 对抗样本技术对恶意软件检测领域的促进

5.2.1 Android 恶意软件对抗样本产生的本质原因

对于对抗样本的成因, 现阶段的学术界并未给出统一的解释, 各种假设性解释百花齐放. 从数据层面来说, 不同于计算机视觉领域中图像数据中每个像素的信息过于抽象, 在 Android 恶意软件检测领域中, 数据中每个数值或代表一个权限, 或代表一个敏感函数, 因此其更具有实际意义. 因此通过对 Android 恶意软件对抗样本中产生的扰动分析, 有助于研究者解释对抗样本存在的原因. 从模型层面来说, 由于 Android 领域中的模型较为简单实用, 研究者可以借助深度模型的可解释性研究与对抗样本研究, 分析不同模型对于不同 Android 特征、不同家族的 Android 软件分析识别机理, 对于 Android 恶意软件检测的可解释性、恶意代码定位等任务有一定的帮助.

5.2.2 对鲁棒性恶意软件检测模型设计的启发

了解现有的 Android 恶意软件对抗样本技术对鲁棒性恶意软件检测模型设计有极大的启发作用. 本文将从特征、模型、系统这 3 个方面论述这一问题.

在特征方面, 由于现有的问题空间对抗样本方法往往需要涉及对 Android 源码的修改, 因此在设计分类模型特征时, 除了分类精度, 还要考虑特征在问题空间中被篡改的难易程度. 例如, 0-1 向量特征、函数调用图特征、图像特征这三者中, 图像特征最难实现从特征空间到问题空间的映射, 因此最难被攻击. 此外, 对抗扰动可以成功扰

动分类器结果的关键在于其代码层面的扰动成功映射到了特征维度. 因此, 在映射过程中删减掉语义无关的扰动也是增强检测算法鲁棒性的研究方向之一. 现有的特征提取方法仅限于对代码进行字符串匹配, 然而大模型对于代码强大的理解能力能够有效过滤掉无关的扰动代码提取关键代码段, 因此在未来如何借助大语言模型提升Android 恶意软件鲁棒性也是重要的研究方向之一. 在模型方面, 对抗重训练被认为是现阶段最有效的对抗样本防御技术, 因此, 了解现有的对抗样本攻击算法, 可以为对抗重训练提供训练样本. 最后, 在系统设计方面, 现有的Android 恶意软件系统往往部署在云端, 而攻击算法为了保证攻击成功率需要对同一软件进行反复的修改并上传至云端模型进行查询, 因此, 云端模型需要额外的对抗样本检测模型来检查可疑的查询流以抵御对抗样本的攻击.

6 结 语

随着深度学习在 Android 恶意软件检测领域的应用越来越广泛, 与其安全密切相关的对抗样本攻击技术也需要不断跟进与总结. 本文针对现有研究综述存在的问题, 归纳总结了近年来的 Android 对抗样本攻击技术, 从对抗样本生成算法与对抗样本扰动实现手段两个角度同时对现有技术进行分类, 并讲解了不同方法的优缺点. 最后总结分析了 Android 对抗样本领域面临的挑战及未来的研究方向.

References:

[1] Fan M, Liu T, Liu J, Luo XP, Yu L, Guan XH. Android malware detection: A survey. Scientia Sinica Informationis, 2020, 50(8): 1148–1177 (in Chinese with English abstract). [doi: 10.1360/SSI-2019-0149]

[2] Ye YF, Hou SF, Chen LW, Lei JW, Wan WQ, Wang JB, Xiong Q, Shao FD. AiDroid: When heterogeneous information network marries deep neural network for real-time Android malware detection. arXiv:1811.01027, 2019.

[3] Tatyana S. The mobile malware threat landscape in 2022. 2023. https://securelist.com/mobile-threat-report-2022/108844

[4] Zheng M, Sun MS, Lui JCS. Droid analytics: A signature based analytic system to collect, extract, analyze and associate Android malware. In: Proc. of the 12th IEEE Int'l Conf. on Trust, Security and Privacy in Computing and Communications. Melbourne: IEEE, 2013. 163–171. [doi: 10.1109/TrustCom.2013.25]

[5] Zhou YY, Wang Z, Zhou W, Jiang XX. Hey, you, get off of my market: Detecting malicious APPs in official and alternative Android markets. In: Proc. of the 2012 NDSS Symp. San Diego, 2012.

[6] Seo SH, Gupta A, Sallam AM, Bertino E, Yim K. Detecting mobile malware threats to homeland security through static analysis. Journal of Network and Computer Applications, 2014, 38: 43–53. [doi: 10.1016/j.jnca.2013.05.008]

[7] Feng Y, Bastani O, Martins R, Dillig I, Anand S. Automated synthesis of semantic malware signatures using maximum satisfiability. arXiv:1608.06254, 2017.

[8] Feng Y, Anand S, Dillig I, Aiken A. Apposcopy: Semantics-based detection of Android malware through static analysis. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Hong Kong: ACM, 2014. 576–587. [doi: 10.1145/2635868.2635869]

[9] Arp D, Spreitzenbarth M, Hübner M, Gascon H, Rieck K. DREBIN: Effective and explainable detection of Android malware in your pocket. In: Proc. of the 2014 NDSS Symp. San Diego, 2014. [doi: 10.14722/ndss.2014.23247]

[10] Octeau D, McDaniel P, Jha S, Bartel A, Bodden E, Klein J, Le Traon Y. Effective inter-component communication mapping in Android with *Epicc*: An essential step towards holistic security analysis. In: Proc. of the 22nd USENIX Security Symp. Washington: USENIX Association, 2013. 543–558.

[11] Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification. In: Proc. of the 16th ACM Conf. on Computer and Communications Security. Chicago: ACM, 2009. 235–245. [doi: 10.1145/1653662.1653691]

[12] Li J, Sun LC, Yan QB, Li ZQ, Srisa-An W, Ye H. Significant permission identification for machine-learning-based Android malware detection. IEEE Trans. on Industrial Informatics, 2018, 14(7): 3216–3225. [doi: 10.1109/TII.2017.2789219]

[13] Bai YD, Xing ZC, Li XH, Feng ZY, Ma DY. Unsuccessful story about few shot malware family classification and siamese network to the rescue. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 1560–1571. [doi: 10.1145/3377811.3380354]

[14] Yuan W, Jiang Y, Li H, Cai MH. A lightweight on-device detection method for Android malware. IEEE Trans. on Systems, Man, and Cybernetics: Systems, 2021, 51(9): 5600–5611. [doi: 10.1109/TSMC.2019.2958382]

[15] Fereidooni H, Conti M, Yao DF, Sperduti A. ANASTASIA: Android malware detection using static analysis of applications. In: Proc. of

the 8th IFIP Int'l Conf. on New Technologies, Mobility and Security (NTMS). Larnaca: IEEE, 2016. 1–5. [doi: 10.1109/NTMS.2016.7792435]

[16] Xu K, Li YJ, Deng RH, Chen K. DeepRefiner: Multi-layer Android malware detection system applying deep neural networks. In: Proc. of the 2018 IEEE European Symp. on Security and Privacy (EuroS&P). London: IEEE, 2018. 473–487. [doi: 10.1109/EuroSP.2018.00040]

[17] Fan M, Liu J, Luo XP, Chen K, Tian ZZ, Zheng QH, Liu T. Android malware familial classification and representative sample selection via frequent subgraph analysis. IEEE Trans. on Information Forensics and Security, 2018, 13(8): 1890–1905. [doi: 10.1109/TIFS.2018.2806891]

[18] Gao TC, Peng W, Sisodia D, Saha TK, Li F, Al Hasan M. Android malware detection via graphlet sampling. IEEE Trans. on Mobile Computing, 2019, 18(12): 2754–2767. [doi: 10.1109/TMC.2018.2880731]

[19] Zhang M, Duan Y, Yin H, Zhao ZR. Semantics-aware Android malware classification using weighted contextual API dependency graphs. In: Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security. Scottsdale: ACM, 2014. 1105–1116. [doi: 10.1145/2660267.2660359]

[20] Onwuzurike L, Mariconti E, Andriotis P, De Cristofaro E, Ross G, Stringhini G. Mamadroid: Detecting Android malware by building Markov chains of behavioral models (extended version). ACM Trans. on Privacy and Security (TOPS), 2019, 22(2): 14. [doi: 10.1145/3313391]

[21] Wüchner T, Cisłak A, Ochoa M, Pretschner A. Leveraging compression-based graph mining for behavior-based malware detection. IEEE Trans. on Dependable and Secure Computing, 2019, 16(1): 99–112. [doi: 10.1109/TDSC.2017.2675881]

[22] Hou SF, Ye YF, Song YQ, Abdulhayoglu M. HinDroid: An intelligent Android malware detection system based on structured heterogeneous information network. In: Proc. of the 23rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Halifax: ACM, 2017. 1507–1515. [doi: 10.1145/3097983.3098026]

[23] Gong JC, Niu WN, Li S, Zhang MX, Zhang XS. Sensitive behavioral chain-focused Android malware detection fused with AST semantics. IEEE Trans. on Information Forensics and Security, 2024, 19: 9216–9229. [doi: 10.1109/TIFS.2024.3468891]

[24] Zhang XH, Zhang Y, Zhong M, Ding DZ, Cao YZ, Zhang YK, Zhang M, Yang M. Enhancing state-of-the-art classifiers with API semantics to detect evolved Android malware. In: Proc. of the 2020 ACM SIGSAC Conf. on Computer and Communications Security. Virtual Event: ACM, 2020. 757–770. [doi: 10.1145/3372297.3417291]

[25] Fan M, Liu J, Wang W, Li HF, Tian ZZ, Liu T. DAPASA: Detecting Android piggybacked APPs through sensitive subgraph analysis. IEEE Trans. on Information Forensics and Security, 2017, 12(8): 1772–1785. [doi: 10.1109/TIFS.2017.2687880]

[26] Wu YM, Qi M, Zou DQ, Jin H. Obfuscation-resilient Android malware detection based on graph convolutional networks. Ruan Jian Xue Bao/Journal of Software, 2023, 34(6): 2526–2542 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6848.htm [doi: 10.13328/j.cnki.jos.006848]

[27] Zhang XT, Wang JS, Sun M. GCN-based Android malware detection model. Software Guide, 2020, 19(7): 187–193 (in Chinese with English abstract). [doi: 10.11907/rjdk.192427]

[28] Yue ZW, Fang Y, Zhang L. Android malware detection based on graph attention networks. Journal of Sichuan University (Natural Science Edition), 2022, 59(5): 053002 (in Chinese with English abstract). [doi: 10.19907/j.0490-6756.2022.053002]

[29] Cortes C, Vapnik V. Support-vector networks. Machine Learning, 1995, 20(3): 273–297. [doi: 10.1023/A:1022627411411]

[30] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc. of the IEEE, 1998, 86(11): 2278–2324. [doi: 10.1109/5.726791]

[31] Goodfellow IJ, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. arXiv:1412.6572, 2015.

[32] Moosavi-Dezfooli SM, Fawzi A, Frossard P. DeepFool: A simple and accurate method to fool deep neural networks. In: Proc. of the 2016 IEEE Conf. on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016. 2574–2582. [doi: 10.1109/CVPR.2016.282]

[33] Wang JK. Adversarial examples in physical world. In: Proc. of the 30th Int'l Joint Conf. on Artificial Intelligence. Montreal, 2021. 4925–4926. [doi: 10.24963/ijcai.2021/694]

[34] Zhang CN, Benz P, Lin CG, Karjauv A, Wu J, Kweon IS. A survey on universal adversarial attack. In: Proc. of the 30th Int'l Joint Conf. on Artificial Intelligence. Montreal, 2021. 4687–4694. [doi: 10.24963/ijcai.2021/635]

[35] Chen PY, Zhang H, Sharma Y, Yi JF, Hsieh CJ. ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In: Proc. of the 10th ACM Workshop on Artificial Intelligence and Security. Dallas: ACM, 2017. 15–26. [doi: 10.1145/3128572.3140448]

[36] Moosavi-Dezfooli SM, Fawzi A, Fawzi O, Frossard P. Universal adversarial perturbations. In: Proc. of the 2017 IEEE Conf. on Computer Vision and Pattern Recognition. Honolulu: IEEE, 2017. 86–94. [doi: 10.1109/CVPR.2017.17]

[37] Carlini N, Wagner D. Towards evaluating the robustness of neural networks. In: Proc. of the 2017 IEEE Symp. on Security and Privacy (SP). San Jose: IEEE, 2017. 39–57. [doi: 10.1109/SP.2017.49]

[38] Madry A, Makelov A, Schmidt L, Tsipras D, Vladu A. Towards deep learning models resistant to adversarial attacks. arXiv:1706.06083, 2019.

[39] Su JW, Vargas DV, Sakurai K. One pixel attack for fooling deep neural networks. IEEE Trans. on Evolutionary Computation, 2019, 23(5): 828–841. [doi: 10.1109/TEVC.2019.2890858]

[40] Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R. Intriguing properties of neural networks. arXiv:1312.6199, 2014.

[41] Chow KH, Liu L, Loper M, Bae J, Gursoy ME, Truex S, Wei WQ, Wu YZ. Adversarial objectness gradient attacks in real-time object detection systems. In: Proc. of the 2nd IEEE Int'l Conf. on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA). Atlanta: IEEE, 2020. 263–272. [doi: 10.1109/TPS-ISA50397.2020.00042]

[42] Zhang HT, Zhou WG, Li HQ. Contextual adversarial attacks for object detection. In: Proc. of the 2020 IEEE Int'l Conf. on Multimedia and Expo (ICME). London: IEEE, 2020. 1–6. [doi: 10.1109/ICME46284.2020.9102805]

[43] Wang DR, Li CR, Wen S, Han QL, Nepal S, Zhang XY, Xiang Y. Daedalus: Breaking nonmaximum suppression in object detection via adversarial examples. IEEE Trans. on Cybernetics, 2022, 52(8): 7427–7440. [doi: 10.1109/TCYB.2020.3041481]

[44] Liu X, Yang HR, Liu ZW, Song LH, Li H, Chen YR. DPatch: An adversarial patch attack on object detectors. arXiv:180602299, 2019.

[45] Hu YCT, Chen JC, Kung BH, Hua KL, Tan DS. Naturalistic physical adversarial patch for object detectors. In: Proc. of the 2021 IEEE/CVF Int'l Conf. on Computer Vision. Montreal: IEEE, 2021. 7828–7837. [doi: 10.1109/ICCV48922.2021.00775]

[46] Lee M, Kolter Z. On physical adversarial patches for object detection. arXiv:1906.11897, 2019.

[47] Li YZ, Tian D, Chang MC, Bian X, Lyu SW. Robust adversarial perturbation on deep proposal-based models. arXiv:1809.05962, 2019.

[48] Xie CH, Wang JY, Zhang ZS, Zhou YY, Xie LX, Yuille A. Adversarial examples for semantic segmentation and object detection. In: Proc. of the 2017 IEEE Int'l Conf. on Computer Vision. Venice: IEEE, 2017. 1378–1387. [doi: 10.1109/ICCV.2017.153]

[49] Duan RJ, Mao XF, Qin AK, Chen YF, Ye SK, He Y, Yang Y. Adversarial laser beam: Effective physical-world attack to DNNs in a blink. In: Proc. of the 2021 IEEE/CVF Conf. on Computer Vision and Pattern Recognition. Nashville: IEEE, 2021. 16057–16066. [doi: 10.1109/CVPR46437.2021.01580]

[50] Li DQ, Li QM. Adversarial deep ensemble: Evasion attacks and defenses for malware detection. IEEE Trans. on Information Forensics and Security, 2020, 15: 3886–3900. [doi: 10.1109/TIFS.2020.3003571]

[51] Shahpasand M, Hamey L, Vatsalan D, Xue MH. Adversarial attacks on mobile malware detection. In: Proc. of the 1st IEEE Int'l Workshop on Artificial Intelligence for Mobile (AI4Mobile). Hangzhou: IEEE, 2019. 17–20. [doi: 10.1109/AI4Mobile.2019.8672711]

[52] Demontis A, Melis M, Biggio B, Maiorca D, Arp D, Rieck K, Corona I, Giacinto G, Roli F. Yes, machine learning can be more secure! A case study on Android malware detection. IEEE Trans. on Dependable and Secure Computing, 2019, 16(4): 711–724. [doi: 10.1109/TDSC.2017.2700270]

[53] Hou RT, Xiang XY, Zhang QX, Liu JB, Huang T. Universal adversarial perturbations of malware. In: Proc. of the 12th Int'l Symp. on Cyberspace Safety and Security. Haikou: Springer, 2020. 9–19. [doi: 10.1007/978-3-030-73671-2_2]

[54] Zhao KF, Zhou H, Zhu YL, Zhan X, Zhou K, Li JF, Yu L, Yuan W, Luo XP. Structural attack against graph based Android malware detection. In: Proc. of the 2021 ACM SIGSAC Conf. on Computer and Communications Security. Virtual Event: ACM, 2021. 3218–3235. [doi: 10.1145/3460120.3485387]

[55] Darwaish A, Naït-Abdesselam F, Titouna C, Sattar S. Robustness of image-based Android malware detection under adversarial attacks. In: Proc. of the 2021 IEEE Int'l Conf. on Communications. Montreal: IEEE, 2021. 1–6. [doi: 10.1109/ICC42927.2021.9500425]

[56] Rathore H, Sahay SK, Nikam P, Sewak M. Robust Android malware detection system against adversarial attacks using Q-learning. Information Systems Frontiers, 2021, 23(4): 867–882. [doi: 10.1007/s10796-020-10083-8]

[57] Yang W, Kong DG, Xie T, Gunter CA. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in Android APPs. In: Proc. of the 33rd Annual Computer Security Applications Conf. Orlando: ACM, 2017. 288–302. [doi: 10.1145/3134600.3134642]

[58] Pierazzi F, Pendlebury F, Cortellazzi J, Cavallaro L. Intriguing properties of adversarial ML attacks in the problem space. In: Proc. of the 2020 IEEE Symp. on Security and Privacy (SP). San Francisco: IEEE, 2020. 1332–1349. [doi: 10.1109/SP40000.2020.00073]

[59] Hu WW, Tan Y. Generating adversarial malware examples for black-box attacks based on GAN. In: Proc. of the 7th Int'l Conf. on Data Mining and Big Data. Beijing: Springer, 2022. 409–423. [doi: 10.1007/978-981-19-8991-9_29]

[60] Gu SY, Cheng SY, Zhang WM. From image to code: Executable adversarial examples of Android applications. In: Proc. of the 6th Int'l Conf. on Computing and Artificial Intelligence. Tianjin: ACM, 2020. 261–268. [doi: 10.1145/3404555.3404574]

[61] Li H, Cheng Z, Wu B, Yuan LH, Gao CY, Yuan W, Luo XP. Black-box adversarial example attack towards FCG based Android malware detection under incomplete feature information. In: Proc. of the 32nd USENIX Security Symp. Anaheim: USENIX Association, 2023. 1181–1198.

[62] Liu XL, Du XJ, Zhang XS, Zhu QX, Wang H, Guizani M. Adversarial samples on Android malware detection systems for IoT systems. Sensors, 2019, 19(4): 974. [doi: 10.3390/s19040974]

[63] Grosse K, Papernot N, Manoharan P, Backes M, McDaniel P. Adversarial examples for malware detection. In: Proc. of the 22nd European Symp. on Research in Computer Security. Oslo: Springer, 2017. 62–79. [doi: 10.1007/978-3-319-66399-9_4]

[64] He P, Xia YF, Zhang XH, Ji SL. Efficient query-based attack against ML-based Android malware detection under zero knowledge setting. In: Proc. of the 2023 ACM Conf. on Computer and Communications Security. Copenhagen: ACM, 2023. 90–104. [doi: 10.1145/3576915.3623117]

[65] Li JL, Wang YZ, Luo LG, Wang Y. A survey of adversarial attack techniques for Android malware detection. Journal of Cyber Security, 2021, 6(4): 28–43 (in Chinese with English abstract). [doi: 10.19363/J.cnki.cn10-1380/tn.2021.07.02]

[66] Liu Y, Tantithamthavorn C, Li L, Liu YP. Deep learning for Android malware defenses: A systematic literature review. ACM Computing Surveys, 2022, 55(8): 153. [doi: 10.1145/3544968]

[67] Yan SM, Ren J, Wang W, Sun LM, Zhang W, Yu Q. A survey of adversarial attack and defense methods for malware classification in cyber security. IEEE Communications Surveys & Tutorials, 2023, 25(1): 467–496.

[68] Aafer Y, Du WL, Yin H. DroidAPIMiner: Mining API-level features for robust malware detection in Android. In: Proc. of the 9th Int'l Conf. on Security and Privacy in Communication Networks. Sydney: Springer, 2013. 86–103. [doi: 10.1007/978-3-319-04283-1_6]

[69] Marastoni N, Continella A, Quarta D, Zanero S, Preda MD. GroupDroid: Automatically grouping mobile malware by extracting code similarities. In: Proc. of the 7th Software Security, Protection, and Reverse Engineering/Software Security and Protection Workshop. Orlando: ACM, 2017. 1. [doi: 10.1145/3151137.3151138]

[70] Isohara T, Takemori K, Kubota A. Kernel-based behavior analysis for Android malware detection. In: Proc. of the 7th Int'l Conf. on Computational Intelligence and Security. Sanya: IEEE, 2011. 1011–1015. [doi: 10.1109/CIS.2011.226]

[71] Au KWY, Zhou YF, Huang Z, Lie D. PScout: Analyzing the Android permission specification. In: Proc. of the 2012 ACM Conf. on Computer and Communications Security. Raleigh: ACM, 2012. 217–228. [doi: 10.1145/2382196.2382222]

[72] Shao YR, Luo XP, Qian CX, Zhu PF, Zhang L. Towards a scalable resource-driven approach for detecting repackaged Android applications. In: Proc. of the 30th Annual Computer Security Applications Conf. New Orleans: ACM, 2014. 56–65. [doi: 10.1145/2664243.2664275]

[73] Chen K, Liu P, Zhang YJ. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets. In: Proc. of the 36th Int'l Conf. on Software Engineering. Hyderabad: ACM, 2014. 175–186. [doi: 10.1145/2568225.2568286]

[74] Zhao M, Ge FB, Zhang T, Yuan ZJ. AntiMalDroid: An efficient SVM-based malware detection framework for Android. In: Proc. of the 2nd Int'l Conf. on Information Computing and Applications. Qinhuangdao: Springer, 2011. 158–166. [doi: 10.1007/978-3-642-27503-6_22]

[75] da Costa FH, Medeiros I, Menezes T, da Silva JV, da Silva IL, Bonifácio R, Narasimhan K, Ribeiro M. Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for Android malware identification. Journal of Systems and Software, 2022, 183: 111092. [doi: 10.1016/j.jss.2021.111092]

[76] Xue L, Zhou YJ, Chen T, Luo XP, Gu GF. Malton: Towards on-device non-invasive mobile malware analysis for ART. In: Proc. of the 26th USENIX Security Symp. Vancouver: USENIXAssociation, 2017. 289–306.

[77] Hasan H, Tork Ladani B, Zamani B. MEGDroid: A model-driven event generation framework for dynamic Android malware analysis. Information and Software Technology, 2021, 135: 106569. [doi: 10.1016/j.infsof.2021.106569]

[78] Mahdavifar S, Kadir AFA, Fatemi R, Alhadidi D, Ghorbani AA. Dynamic Android malware category classification using semi-supervised deep learning. In: Proc. of the 2020 IEEE Int'l Conf. on Dependable, Autonomic and Secure Computing, Int'l Conf. on Pervasive Intelligence and Computing, Int'l Conf. on Cloud and Big Data Computing, Int'l Conf. on Cyber Science and Technology Congress. Calgary: IEEE, 2020. 515–522. [doi: 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094]

[79] Khalid S, Hussain FB. Evaluating dynamic analysis features for Android malware categorization. In: Proc. of the 2022 Int'l Wireless Communications and Mobile Computing (IWCMC). Dubrovnik: IEEE, 2022. 401–406. [doi: 10.1109/IWCMC55113.2022.9824225]

[80] Zhu ZY, Dumitraş T. FeatureSmith: Automatically engineering features for malware detection by mining the security literature. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. Vienna: ACM, 2016. 767–778. [doi: 10.1145/2976749.2978304]

[81] Chan PPK, Song WK. Static detection of Android malware by using permissions and API calls. In: Proc. of the 2014 Int'l Conf. on

Machine Learning and Cybernetics. Lanzhou: IEEE, 2014. 82–87. [doi: 10.1109/ICMLC.2014.7009096]

[82] Peiravian N, Zhu XQ. Machine learning for Android malware detection using permission and API calls. In: Proc. of the 25th IEEE Int'l Conf. on Tools with Artificial Intelligence. Herndon: IEEE, 2013. 300–305. [doi: 10.1109/ICTAI.2013.53]

[83] Gascon H, Yamaguchi F, Arp D, Rieck K. Structural detection of Android malware using embedded call graphs. In: Proc. of the 2013 ACM Workshop on Artificial Intelligence and Security. Berlin: ACM, 2013. 45–54. [doi: 10.1145/2517312.2517315]

[84] Hu WJ, Tao J, Ma XB, Zhou WY, Zhao S, Han T. MIGDroid: Detecting APP-repackaging Android malware via method invocation graph. In: Proc. of the 23rd Int'l Conf. on Computer Communication and Networks (ICCCN). Shanghai: IEEE, 2014. 1–7. [doi: 10.1109/ICCCN.2014.6911805]

[85] Wu YM, Li XD, Zou DQ, Yang W, Zhang X, Jin H. MalScan: Fast market-wide mobile malware scanning by social-network centrality analysis. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 139–150. [doi: 10.1109/ASE.2019.00023]

[86] Sun X, Zhongyang YB, Xin Z, Mao B, Xie L. Detecting code reuse in Android applications using component-based control flow graph. In: Proc. of the 29th IFIP TC 11 Int'l Conf. on ICT Systems Security and Privacy Protection. Marrakech: Springer, 2014. 142–155. [doi: 10.1007/978-3-642-55415-5_12]

[87] Meng GZ, Xue YX, Xu ZZ, Liu Y, Zhang J, Narayanan A. Semantic modelling of Android malware for effective malware comprehension, detection, and classification. In: Proc. of the 25th Int'l Symp. on Software Testing and Analysis. Saarbrücken: ACM, 2016. 306–317. [doi: 10.1145/2931037.2931043]

[88] Crussell J, Gibler C, Chen H. Attack of the clones: Detecting cloned applications on Android markets. In: Proc. of the 17th European Symp. on Research in Computer Computer Security. Pisa: Springer, 2012. 37–54. [doi: 10.1007/978-3-642-33167-1_3]

[89] Wolfe B, Elish KO, Yao DF. Comprehensive behavior profiling for proactive Android malware detection. In: Proc. of the 17th Int'l Conf. on Information Security. Hong Kong: Springer, 2014. 328–344. [doi: 10.1007/978-3-319-13257-0_19]

[90] Chen K, Wang P, Lee Y, Wang XF, Zhang N, Huang HQ, Zou W, Liu P. Finding unknown malice in 10 seconds: Mass vetting for new threats at the Google-play scale. In: Proc. of the 24th USENIX Security Symp. Washington: USENIX Association, 2015. 659–674.

[91] Zhang FF, Huang HQ, Zhu SC, Wu DH, Liu P. ViewDroid: Towards obfuscation-resilient mobile application repackaging detection. In: Proc. of the 2014 ACM Conf. on Security and Privacy in Wireless & Mobile Networks. Oxford: ACM, 2014. 25–36. [doi: 10.1145/2627393.2627395]

[92] Zheng C, Zhu SX, Dai SF, Gu GF, Gong XR, Han XH, Zou W. SmartDroid: An automatic system for revealing UI-based trigger conditions in Android applications. In: Proc. of the 2nd ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. Raleigh: ACM, 2012. 93–104. [doi: 10.1145/2381934.2381950]

[93] Conti G, Bratus S, Shubina A, Sangster B, Ragsdale R, Supan M, Lichtenberg A, Perez-Alemany R. Automated mapping of large binary objects using primitive fragment type classification. Digital Investigation, 2010, 7: S3–S12. [doi: 10.1016/j.diin.2010.05.002]

[94] Yuan BG, Wang JF, Liu D, Guo W, Wu P, Bao XH. Byte-level malware classification based on Markov images and deep learning. Computers & Security, 2020, 92: 101740. [doi: 10.1016/j.cose.2020.101740]

[95] Nataraj L, Karthikeyan S, Jacob G, Manjunath BS. Malware images: Visualization and automatic classification. In: Proc. of the 8th Int'l Symp. on Visualization for Cyber Security. Pittsburgh: ACM, 2011. 4. [doi: 10.1145/2016904.2016908]

[96] Xiao X, Yang S. An image-inspired and CNN-based Android malware detection approach. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 1259–1261. [doi: 10.1109/ASE.2019.00155]

[97] Bruna J, Zaremba W, Szlam A, LeCun Y. Spectral networks and locally connected networks on graphs. arXiv:1312.6203, 2014.

[98] Li HY, Di SM, Li ZJ, Chen L, Cao JN. Black-box Adversarial attack and defense on graph neural networks. In: Proc. of the 2022 IEEE 38th Int'l Conf. on Data Engineering (ICDE). Kuala Lumpur: IEEE, 2022. 1017–1030. [doi: 10.1109/ICDE53745.2022.00081]

[99] Lin XX, Zhou C, Wu J, Yang H, Wang HB, Cao YN, Wang B. Exploratory adversarial attacks on graph neural networks for semi-supervised node classification. Pattern Recognition, 2023, 133: 109042. [doi: 10.1016/j.patcog.2022.109042]

[100] Chen JY, Zhang DJ, Ming ZY, Huang KJ, Jiang WR, Cui C. GraphAttacker: A general multi-task graph attack framework. IEEE Trans. on Network Science and Engineering, 2022, 9(2): 577–595. [doi: 10.1109/TNSE.2021.3127557]

[101] Fan WQ, Xu H, Jin W, Liu XR, Tang XF, Wang SH, Li Q, Tang JL, Wang JP, Aggarwal C. Jointly attacking graph neural network and its explanations. In: Proc. of the 39th IEEE Int'l Conf. on Data Engineering (ICDE). Anaheim: IEEE, 2023. 654–667. [doi: 10.1109/ICDE55515.2023.00056]

[102] Liu ZH, Luo Y, Zang ZL, Li SZ. Surrogate representation learning with isometric mapping for gray-box graph adversarial attacks. In: Proc. of the 15th ACM Int'l Conf. on Web Search and Data Mining. ACM, 2022. 591–598. [doi: 10.1145/3488560.3498481]

[103] Wang BH, Pang M, Dong Y. Turning Strengths into Weaknesses: A certified robustness inspired attack framework against graph neural

networks. In: Proc. of the 2023 IEEE/CVF Conf. on Computer Vision and Pattern Recognition. Vancouver: IEEE, 2023. 16394–16403. [doi: 10.1109/CVPR52729.2023.01573]

[104] Li JT, Xie T, Liang C, Xie FF, He XN, Zheng ZB. Adversarial attack on large scale graph. IEEE Trans. on Knowledge and Data Engineering, 2023, 35(1): 82–95. [doi: 10.1109/TKDE.2021.3078755]

[105] Sun YW, Wang SH, Tang XF, Hsieh TY, Honavar V. Adversarial attacks on graph neural networks via node injections: A hierarchical reinforcement learning approach. In: Proc. of the 2020 Web Conf. Taipei: ACM, 2020. 673–683. [doi: 10.1145/3366423.3380149]

[106] Tao SC, Cao Q, Shen HW, Wu YF, Hou L, Sun F, Cheng XQ. Adversarial camouflage for node injection attack on graphs. Information Sciences, 2023, 694: 119611. [doi: 10.1016/j.ins.2023.119611]

[107] Li JY, Zhang TY, Jin SM, Fardad M, Zafarani R. AdverSparse: An adversarial attack framework for deep spatial-temporal graph neural networks. In: Proc. of the 2022 IEEE Int'l Conf. on Acoustics, Speech and Signal Processing (ICASSP). Singapore: IEEE, 2022. 5857–5861. [doi: 10.1109/ICASSP43922.2022.9747850]

[108] Chen X, Li CR, Wang DR, Wen S, Zhang J, Nepal S, Xiang Y, Ren K. Android HIV: A study of repackaging malware for evading machine-learning detection. IEEE Trans. on Information Forensics and Security, 2020, 15: 987–1001. [doi: 10.1109/TIFS.2019.2932228]

[109] Papernot N, McDaniel P, Jha S, Fredrikson M, Celik ZB, Swami A. The limitations of deep learning in adversarial settings. In: Proc. of the 2016 IEEE European Symp. on Security and Privacy (EuroS&P). Saarbruecken: IEEE, 2016. 372–387. [doi: 10.1109/EuroSP.2016.36]

[110] Alghamdi R, Alfalqi K, Waqdan M. Android platform malware analysis. Int'l Journal of Advanced Computer Science and Applications, 2015, 6(1): 140–146.

[111] Barr ET, Harman M, Jia Y, Marginean A, Petke J. Automated software transplantation. In: Proc. of the 2015 Int'l Symp. on Software Testing and Analysis. Baltimore: ACM, 2015. 257–269. [doi: 10.1145/2771783.2771796]

[112] Bostani H, Moonsamy V. EvadeDroid: A practical evasion attack on machine learning for black-box Android malware detection. arXiv: 2110.03301, 2024.

[113] Li H, Zhou SY, Yuan W, Li JH, Leung H. Adversarial-example attacks toward Android malware detection system. IEEE Systems Journal, 2020, 14(1): 653–656. [doi: 10.1109/JSYST.2019.2906120]

[114] Watkins CJCH, Dayan P. Q-learning. Machine Learning, 1992, 8: 279–292. [doi: 10.1007/BF00992698]

[115] Holland JH. Genetic algorithms. Scientific American, 1992, 267(1): 66–72. [doi: 10.1038/scientificamerican0792-66]

[116] Xu GQ, Xin GH, Jiao LT, Liu J, Liu SY, Feng MQ, Zheng X. OFEI: A semi-black-box Android adversarial sample attack framework against DlaaS. IEEE Trans. on Computers, 2024, 73(4): 956–969. [doi: 10.1109/TC.2023.3236872]

[117] Bertsimas D, Tsitsiklis J. Simulated annealing. Statistical Science, 1993, 8(1): 10–15.

[118] Rathore H, Nandanwar A, Sahay SK, Sewak M. Adversarial superiority in Android malware detection: Lessons from reinforcement learning based evasion attacks and defenses. Forensic Science Int'l: Digital Investigation, 2023, 44(Supplement): 301511. [doi: 10.1016/j.fsidi.2023.301511]

[119] Schulman J, Levine S, Abbeel P, Jordan M, Moritz P. Trust region policy optimization. In: Proc. of the 32nd Int'l Conf. on Machine Learning. 2015. 1889–1897.

[120] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv:1707.06347, 2017.

[121] Bang-Jensen J, Gutin G, Yeo A. When the greedy algorithm fails. Discrete Optimization, 2004, 1(2): 121–127. [doi: 10.1016/j.disopt.2004.03.007]

[122] Labaca-Castro R, Muñoz-González L, Pendlebury F, Rodosek GD, Pierazzi F, Cavallaro L. Realizable universal adversarial perturbations for malware. arXiv:2102.06747, 2022.

[123] Guo YP, Yan Q. Android malware adversarial attacks based on feature importance prediction. Int'l Journal of Machine Learning and Cybernetics, 2023, 14(6): 2087–2097. [doi: 10.1007/s13042-022-01747-9]

[124] Yang W, Xiao XS, Andow B, Li SH, Xie T, Enck W. AppContext: Differentiating malicious and benign mobile APP behaviors using context. In: Proc. of the 37th IEEE/ACM IEEE Int'l Conf. on Software Engineering. Florence: IEEE, 2015. 303–313. [doi: 10.1109/ICSE.2015.50]

[125] Rafiq H, Aslam N, Issac B, Randhawa RH. On impact of adversarial evasion attacks on ML-based Android malware classifier trained on hybrid features. In: Proc. of the 14th Int'l Conf. on Software, Knowledge, Information Management and Applications (SKIMA). Phnom Penh: IEEE, 2022. 216–221. [doi: 10.1109/SKIMA57145.2022.10029504]

[126] Guerra-Manzanares A, Bahsi H, Nõmm S. KronoDroid: Time-based hybrid-featured dataset for effective Android malware detection and characterization. Computers & Security, 2021, 110: 102399. [doi: 10.1016/j.cose.2021.102399]

[127] Li XJ, Kong K, Xu S, Qin PT, He DJ. Feature selection-based Android malware adversarial sample generation and detection method. IET Information Security, 2021, 15(6): 401–416. [doi: 10.1049/ise2.12030]

[128] Rathore H, Sahay SK, Dhillon J, Sewak M. Designing adversarial attack and defence for robust Android malware detection models. In: Proc. of the 51st Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks-Supplemental Volume (DSN-S). Taipei: IEEE, 2021. 29–32. [doi: 10.1109/DSN-S52858.2021.00025]

[129] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing atari with deep reinforcement learning. arXiv:1312.5602, 2013.

[130] Zhang L, Liu P, Choi YH, Chen P. Semantics-preserving reinforcement learning attack against graph neural networks for malware detection. IEEE Trans. on Dependable and Secure Computing, 2023, 20(2): 1390–1402. [doi: 10.1109/TDSC.2022.3153844]

[131] Yuan ZL, Lu YQ, Xue YB. Droiddetector: Android malware characterization and detection using deep learning. Tsinghua Science and Technology, 2016, 21(1): 114–123. [doi: 10.1109/TST.2016.7399288]

[132] Zhang Y, Yang YX, Wang XL. A novel Android malware detection approach based on convolutional neural network. In: Proc. of the 2nd Int'l Conf. on Cryptography, Security and Privacy. Guiyang: ACM, 2018. 144–149. [doi: 10.1145/3199478.3199492]

[133] Abbass HA, Sarker R. The pareto differential evolution algorithm. Int'l Journal on Artificial Intelligence Tools, 2002, 11(4): 531–552. [doi: 10.1142/S0218213002001039]

[134] Puterman ML. Markov decision processes. In: Handbooks in Operations Research and Management Science, Vol. 2. North-Holland: Elsevier Science Publishers, 1990. 331–434.

[135] Watson MR, Shirazi NH, Marnerides AK, Mauthe A, Hutchison D. Malware detection in cloud computing infrastructures. IEEE Trans. on Dependable and Secure Computing, 2016, 13(2): 192–205. [doi: 10.1109/TDSC.2015.2457918]

[136] Hatem SS, Wafy MH, El-Khouly MM. Malware detection in cloud computing. Int'l Journal of Advanced Computer Science and Applications, 2014, 5(4): 187–192. [doi: 10.14569/IJACSA.2014.050427]

[137] Du XJ, Guizani M, Xiao Y, Chen HH. Transactions papers a routing-driven elliptic curve cryptography based key management scheme for heterogeneous sensor networks. IEEE Trans. on Wireless Communications, 2009, 8(3): 1223–1229. [doi: 10.1109/TWC.2009.060598]

[138] Hei X, Du XJ, Lin S, Lee I, Sokolsky O. Patient infusion pattern based access control schemes for wireless insulin pump system. IEEE Trans. on Parallel and Distributed Systems, 2015, 26(11): 3108–3121. [doi: 10.1109/TPDS.2014.2370045]

[139] Wu LF, Du XJ, Wu J. Effective defense schemes for phishing attacks on mobile computing platforms. IEEE Trans. on Vehicular Technology, 2016, 65(8): 6678–6691. [doi: 10.1109/TVT.2015.2472993]

[140] Du XJ, Xiao Y, Guizani M, Chen HH. An effective key management scheme for heterogeneous sensor networks. Ad Hoc Networks, 2007, 5(1): 24–34. [doi: 10.1016/j.adhoc.2006.05.012]

[141] Cheng YX, Fu X, Du XJ, Luo B, Guizani M. A lightweight live memory forensic approach based on hardware virtualization. Information Sciences, 2017, 379: 23–41. [doi: 10.1016/j.ins.2016.07.019]

[142] Ko J, Shim H, Kim D, Jeong YS, Cho SJ, Park M, Han S, Kim SB. Measuring similarity of Android applications via reversing and K-gram birthmarking. In: Proc. of the 2013 Research in Adaptive and Convergent Systems. Montreal: ACM, 2010. 336–341. [doi: 10.1145/2513228.2513308]

[143] Jerome Q, Allix K, State R, Engel T. Using opcode-sequences to detect malicious Android applications. In: Proc. of the 2014 IEEE Int'l Conf. on Communications (ICC). Sydney: IEEE, 2014. 914–919. [doi: 10.1109/ICC.2014.6883436]

[144] Kang B, Yerima SY, Sezer S, McLaughlin K. N-gram opcode analysis for Android malware detection. arXiv:1612.01445, 2016.

[145] Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y. Generative adversarial nets. In: Proc. of the 27th Int'l Conf. on Neural Information Processing Systems. Montreal: MIT Press, 2014. 2672–2680.

[146] Weiser M. Program slicing. IEEE Trans. on Software Engineering, 1984, SE-10(4): 352–357. [doi: 10.1109/TSE.1984.5010248]

[147] Dong SK, Li MH, Diao WR, Liu XY, Liu J, Li Z, Xu FH, Chen K, Wang XF, Zhang KH. Understanding Android obfuscation techniques: A large-scale investigation in the wild. In: Proc. of the 14th Int'l Conf. on Security and Privacy in Communication Networks. Singapore: Springer, 2018. 172–192. [doi: 10.1007/978-3-030-01701-9_10]

[148] Bacci A, Bartoli A, Martinelli F, Medvet E, Mercaldo F, Visaggio CA. Impact of code obfuscation on Android malware detection based on static and dynamic analysis. In: Proc. of the 4th Int'l Conf. on Information Systems Security and Privacy. Funchal-Madeira: Springer, 2018. 379–385. [doi: 10.5220/0006642503790385]

[149] Kovacheva A. Efficient code obfuscation for Android. In: Proc. of the 6th Int'l Conf. on Advances in Information Technology. Bangkok: Springer, 2013. 104–119. [doi: 10.1007/978-3-319-03783-7_10]

[150] Aonzo S, Georgiu GC, Verderame L, Merlo A. Obfuscapk: An open-source black-box obfuscation tool for Android APPs. SoftwareX,

2020, 11: 100403. [doi: 10.1016/j.softx.2020.100403]

附中文参考文献:

[1] 范铭, 刘烃, 刘均, 罗夏朴, 于乐, 管晓宏. 安卓恶意软件检测方法综述. 中国科学: 信息科学, 2020, 50(8): 1148–1177. [doi: 10.1360/SSI-2019-0149]

[26] 吴月明, 齐蒙, 邹德清, 金海. 图卷积网络的抗混淆安卓恶意软件检测. 软件学报, 2023, 34(6): 2526–2542. http://www.jos.org.cn/1000-9825/6848.htm [doi: 10.13328/j.cnki.jos.006848]

[27] 张雪涛, 王金双, 孙蒙. 基于 GCN 的安卓恶意软件检测模型. 软件导刊, 2020, 19(7): 187–193. [doi: 10.11907/rjdk.192427]

[28] 岳子巍, 方勇, 张磊. 基于图注意力网络的安卓恶意软件检测. 四川大学学报 (自然科学版), 2022, 59(5): 053002. [doi: 10.19907/j.0490-6756.2022.053002]

[65] 李佳琳, 王雅哲, 罗吕根, 王瑜. 面向安卓恶意软件检测的对抗攻击技术综述. 信息安全学报, 2021, 6(4): 28–43. [doi: 10.19363/J.cnki.cn10-1380/tn.2021.07.02]

李珩(1995－), 男, 博士, CCF 专业会员, 主要研究领域为恶意软件检测, 对抗样本攻防.

高翠莹(1997－), 女, 博士生, 主要研究领域为移动软件安全, AI 鲁棒性分析.

吴棒(2001－), 男, 硕士生, 主要研究领域为移动软件安全, 网站指纹攻击.

袁巍(1978－), 男, 博士, 教授, 博士生导师, 主要研究领域为机器学习, 信息安全.

龚柱(1997－), 男, 博士生, 主要研究领域为入侵检测, 流量对抗样本.

罗夏朴(1977－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为移动安全和隐私, 区块链/智能合约, 网络安全和隐私, 软件工程.