# 面向安卓恶意软件检测的对抗攻击技术综述

李佳琳[1,2], 王雅哲[1,2], 罗吕根[1,2], 王 瑜[1]

[1]中国科学院信息工程研究所 北京 中国 100093
[2]中国科学院大学 网络空间安全学院 北京 中国 100049

**摘要** 随着对 Android 恶意软件检测精度和性能要求的提高, 越来越多的 Android 恶意软件检测引擎使用人工智能算法。与此同时, 攻击者开始尝试对 Android 恶意软件进行一定的修改, 使得 Android 恶意软件可以在保留本身的功能的前提下绕过这些基于人工智能算法的检测。上述过程即是 Android 恶意软件检测领域的对抗攻击。本文梳理了目前存在的基于人工智能算法的 Android 恶意软件检测模型, 概述了针对 Android 恶意软件检测模型的对抗攻击方法, 并从特征和算法两方面总结了相应的增强模型安全性的防护手段, 最后提出了 Android 恶意软件检测模型和对抗攻击的发展趋势, 并分析了对抗攻击对 Android 恶意软件检测的影响。

**关键词** Android 安全; 恶意软件; 对抗攻击
**中图法分类号** TP309.5 **DOI 号** 10.19363/J.cnki.cn10-1380/tn.2021.07.02

# A Survey of Adversarial Attack Techniques for Android Malware Detection

LI Jialin[1,2], WANG Yazhe[1,2], LUO Lvgen[1,2], WANG Yu[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

**Abstract** With the accuracy and performance demand for Android malware detection, more and more Android malware detection engines integrate artificial intelligence algorithms. At the same time, attackers have begun to try to modify Android malware to bypass these artificial intelligence based algorithm while preserving their own functionality. It's called adversarial attack in the field of Android malware detection. This paper combs the existing Android malware detection model based on artificial intelligence algorithm, and summarizes the adversarial attack methods for Android malware detection model and the corresponding protection methods for enhancing model security from two aspects of features and algorithms. Finally, the development trend of Android malware detection model and confrontation attack is proposed, and the impact of adversarial attack on Android malware detection is analyzed.

**Key words** android security; malware detection; adversarial attack

## 1 引言

Android 是 Google 推出的一款面向智能手机的开源操作系统。根据知名市场分析咨询公司国际数据(IDC)的数据显示, 截至 2018 年, Android 占据了全球智能手机市场份额的 85%, 5 年的复合年增长率达到了 2.4%, 高于 IOS 的 2.1%[1]。并且随着万物互联时代的到来, Android 系统的可扩展性和开放性为物联网领域设备的生产提供了便利, 因此目前 Android 已逐渐渗透到其他制造业领域, 如车载系统、可穿戴设备、智能家电[2]等。

Android 设备的广泛应用在带来便利的同时也面临着安全风险, 其中最常见的就是 Android 恶意软件。由于国内无法使用 Google Play, Android 用户需要从第三方应用市场下载应用, 而市面上许多应用市场存在审核不严的问题, 这就使得恶意软件有了可乘之机。不仅仅是国内的第三方应用市场存在大量恶意软件, 使用了严格审查机制以及系统内置恶意软件检测工具的 Google Play 上仍然被曝光出包含恶意软件[2]。根据 Google 官方显示, Google Play 和其他应用市场中潜在有害应用(PHA)的比例为 0.06%。目前来看, Android 平台上主要存在的恶意软件有资

**通讯作者:** 王雅哲, 博士, 副研究员, Email: wangyazhe@iie.ac.cn。
本课题得到国家重点研发计划 (No. 2019YFB1706000)资助。
收稿日期: 2019-07-27; 修改日期: 2019-10-08; 定稿日期: 2021-06-24

费消耗、流氓行为、隐私窃取、恶意扣费、远程控制等几种类型。2018 年上半年 360 检测到了 238.3 万个活跃的盗版 app, 全年共截获新增恶意软件样本约 434.2 万个, 平均每天新增约 1.2 万个; 累计监测 Android 恶意软件感染量约为 1.1 亿人次, 平均每日恶意软件感染量约为 29.2 万人次[3]。

每年诞生的 Android 恶意软件数以百万计, 因此现在的安全公司或一些安全产品采用态势感知的方式发现并定位 Android 恶意软件。传统的检测方法一般通过提取 apk 配置文件以及源码或程序控制流[4], 使用确定的规则定位漏洞或恶意代码。这种方法大多数需要安全专家预先制定的规则, 建立庞大的数据库进行指纹匹配。而恶意软件开发者仅仅通过简单的混淆就可以绕过上述自动化检测方法[5-6]。随着人工智能的高速发展, 使用人工智能算法检测恶意软件具有较高的精确度和准确性, 越来越多的安全产品尝试使用人工智能算法来检测恶意软件。人工智能算法通过样本库的训练能够从静态、动态多角度挖掘恶意软件的抽象特征, 能够无视一些混淆操作精确定位到恶意代码。并且使用人工智能算法的检测模型具有良好的可扩展性, 一部分算法在添加特征后甚至不需要再重新学习模型[7], 流行的机器学习模型堆叠方法能够将多数据集多模型结合为精度更高的模型。因此, 当前人工智能算法在 Android 恶意软件检测方面具备很多传统分析方法不具备的优势, 越来越多的安全产品会采用人工智能算法。

值得注意的是, 最早在图像识别领域由 Goodfellow 提出的对抗样本的概念目前也被应用在了 Android 恶意软件检测当中。即通过添加一些不影响 Android 软件正常运行的扰动, 使得算法模型无法识别出该软件为恶意的。扰动在恶意软件检测算法的输入层添加, 对于输入为权限特征的算法, 扰动即应用申请的权限的修改; 对于输入为控制流图的算法, 扰动为控制流的修改。尽管有一些关于防御对抗攻击提高模型安全性的研究[8], 但根据对抗样本的生成原理, 只要使用梯度的模型, 无论是深度学习还是使用梯度更新的机器学习都会存在对抗样本, 同时对抗样本的生成是由梯度确定扰动的方向, 因此只要能够根据模型的结构或模型的更新方式确定扰动方向的人工智能算法模型都面临着对抗攻击的风险。因此, 增强基于人工智能算法的 Android 恶意软件检测模型的防对抗能力也是亟待研究的。

本文主要有如下三部分的工作和贡献:

(1) 总结了 Android 恶意软件检测算法的发展变化, 主要概述基于人工智能算法的 Android 恶意软件检测方法。

(2) 针对目前新兴的 Android 恶意软件对抗攻击进行了分析和概述, 并总结了目前存在的 Android 恶意软件领域的防对抗攻击算法研究。

(3) 讨论了 Android 恶意软件检测与对抗攻击领域的发展趋势, 指出对抗攻击对目前 Android 恶意软件检测发展的影响。

本文的组织结构如下: 第 2 节描述相关背景知识, 包括 Android 应用与传统 Android 恶意软件检测方法; 第 3 节指出目前利用人工智能算法进行 Android 恶意软件检测的相关研究; 第 4 节详细综述 Android 恶意软件的对抗攻击方法; 第 5 节从特征和算法两方面总结针对 Android 恶意软件对抗攻击的安全防护方法; 第 6 节思考和讨论 Android 恶意软件检测以及对抗攻击的发展趋势, 并指出对抗攻击对目前 Android 恶意软件检测发展的影响; 第 7 节总结全文。

## 2 背景知识

### 2.1 Android 平台应用程序简介

Android 应用通常由 Java 语言编写, 由 Android SDK 编译, 由虚拟机加载并解释执行。Android 平台上用于安装 Android 应用程序的文件是一个后缀名为.apk 的压缩文件, 其内包含 Android 应用所有的数据和资源文件。通过解压.apk 文件可以得到以下结构:

- assets: 用于存放需要打包到 apk 中的静态文件

- lib: 存放应用程序依赖的 native 库文件, 一般是用 C/C++语言编写

- res: 存放资源文件。如存放 Android 应用布局 xml 文件

- META-INF: 保存应用的签名信息, 签名信息可以验证 APK 文件的完整性

- AndroidManifest.xml: Android 应用的配置文件。声明应用所申请的权限信息、硬件信息、SDK 版本、包信息等; 描述应用的各个组件(activity、service、broadcast receiver 和 content provider), 向 Android 系统告知有关组件以及可以启动这些组件的条件的信息

- classes.dex: Android 应用程序的可执行文件。java 代码首先会被编译成.class 文件, 得到的类文件

被翻译成 Dalvik 字节码, 最终合并为一个或多个可执行 dex 文件

- resources.arsc: 记录资源文件和资源 ID 之间的映射关系, 用来根据资源 ID 寻找资源

由于 AndroidManifest.xml 含有 Android 应用程序的配置信息, classes.dex 文件是 Android 应用程序的可执行文件, 均与应用行为特征关系及恶意程度紧密相关。因此 Android 的恶意软件检测模型通常以上述两个文件为重点进行特征的分析和选择。需要注意的是, 由 apk 文件直接解压缩得到的 classes.dex 文件和 AndroidManifest.xml 文件都是二进制的形式, 分析人员往往需要进一步进行反编译得到可读的 smali 代码和配置文件。

### 2.2 传统的 Android 恶意软件检测方法

传统的 Android 恶意软件检测一般使用模式匹配或指纹匹配等方法, 在 Android 恶意软件爆发初期被广泛使用。

文献[9]依赖于手工选择的特征进行匹配和启发式过滤, 从而发现恶意软件以及恶意软件家族; 文献[10]利用权限和 API 的上下文和顺序区分 Android 良恶性软件, 建立权限事件图, 自动化发现以前通过手动分析发现的恶意行为; 文献[11]利用一些粗粒度的特征如 AndroidManifest.xml 中申请的权限信息进行 Android 恶意软件的检测; 另外一些研究[12]通过计算 API 调用频率来进行建模。

但这些方法都没有利用 Android 恶意软件的内部程序逻辑, 因此 Yang 等人[13]提出了 DroidMiner, 系统使用静态分析的方法, 从已知的 Android 恶意软件中自动挖掘恶意软件程序逻辑, 将此逻辑抽象为一系列威胁模式, 然后在其他软件中寻找这些威胁模态模式以此进行而恶意软件判别。

Grace 等人[14]提出的 RiskRanker 是一个给 Android 应用恶意程度评级的系统。对于利用 Android 平台漏洞的恶意软件, RiskRanker 会根据漏洞指纹进行识别并将其标记为高危; 对于恶意扣费等恶意行为系统会对代码进行静态分析和模式匹配, 根据其控制流图判断是否存在不提示用户就执行的敏感操作, 并将其标记为中危。另外, 系统会检查代码中是否含有加密方法、是否有 Runtime.exec()方法等特征, 若可疑恶意特征同时存在且共有同样的入口点, 那么系统也将这个 Android 应用标记为高危。中危和高危都会被标记为可疑恶意软件。RiskRanker 相对于其他传统 Android 恶意软件检测系统来说, 在发现 0-day 恶意软件方面具有一定的优势。

除了静态分析外, 也存在一些传统检测方案使用了动态分析。DroidScope[15]使用了污点分析的方法, 监视软件在 API 层的行为, 提取并分析内部组件之间的行为特征以及系统调用特征以检测恶意软件。

总的来说, 传统的 Android 恶意软件检测方法都需要专家建立相应的规则和模式库或指纹库, 需要手动构造和更新检测模式。并且随着时间的推移和恶意软件的不断丰富, 数据库的数据量也会逐渐庞大。但攻击者却可以轻而易举地通过代码混淆等方法绕过检测模型的检测[5-6]。随着恶意软件的不断变种和进化以及人工智能算法的发展, 研究者们开始探究新的检测方法。

## 3 基于人工智能算法的 Android 恶意软件检测方法

随着人工智能的兴起, 机器学习和深度学习算法不断提高自身的精确度, 在大量领域超过了传统算法。由于传统 Android 恶意软件检测算法存在诸多不足, 这就驱使许多研究者开始探究使用人工智能算法来检测 Android 恶意软件。从 Android 恶意软件检测算法中利用的特征类型进行划分, 可以分为{0, 1}型特征、序列型特征和端到端模式。

- {0, 1}型特征: {0, 1}型特征即每一个特征使用一个标志位, 每一个标志位非 1 即 0。1 代表应用存在该特征, 0 代表不存在。检测算法通常会从 AndroidManifest.xml 文件中提取应用申请的权限信息、硬件信息, intent-filter 组件等; 从反编译后的 dex 文件得到的 smali 代码中提取 API 特征、url 等信息。将得到的特征集表示为{0, 1}矩阵形式, 进而使用机器学习或深度学习算法进行训练。

- 序列型特征: 序列型特征主要基于代码中的 API 序列关系或字节码序列关系。一般通过反编译 dex 文件提取 API 调用顺序, 构建控制流图, 或直接提取应用的字节码。进而将控制流图或字节码序列使用能够直接处理多序列的算法或将序列转化为其他易于计算的特征向量形式, 输入到算法模型中进行下一步训练。

- 端到端模式: {0, 1}型特征和序列型特征通常要在解压缩及反编译原 apk 文件得到可读的 AndroidManifest.xml 代码和 smali 代码后, 通过进一步分析, 对原始数据进行构造得到复杂的标准化的特征, 才能进行下一步的操作。而端到端模式不需要构造复杂的特征, 直接使用原始 apk 的二进制码, 或只是进行简单的字节码截取或映射。此时一般会利

用图像领域的相关处理方法, 将应用程序表示为图像, 利用卷积神经网络(Convolutional Neural Network, CNN)等深度学习算法进行训练。

本小节从 Android 恶意软件检测算法中利用的特征类型出发, 分别阐述基于{0, 1}型特征、序列型特征和端到端模式进行恶意软件检测的方法。

### 3.1 0, 1 型特征

文献[16]提出了 DreBin, 这是一种基于静态分析的 Android 恶意软件检测方法。文章首先获取 APP 的 dex 文件反编译后得到的代码和 AndroidManifest. xml 配置文件, 进而利用静态分析提取应用需要的硬件、申请的权限、APP 组件、特殊 API 以及网络地址等信息作为特征嵌入矩阵, 使用 SVM(Support Vector Machine)算法进行训练, 最终得到线性模型以及每个特征所对应的权重。这项研究可以很好地解释模型的判别结果, 并且可以达到 94%的精确度。

但是由于{0, 1}型特征自身的局限性, 在特征过多的情况下样本矩阵变得过于稀疏。而 SVM 算法不能很好地处理稀疏矩阵的情况。除此之外, 不同特征之间的相关性也是揭露恶意软件很重要的一点, 而 Drebin 模型并没有将特征间的相关性考虑进去。基于上述问题, Li 等人[17]提出了利用 FM(Factorization Machine)算法来进行模型的构建。文章在 DreBin 的基础上, 对特征进行了精简, 从 dex 文件反编译得到的代码和 AndroidManifest.xml 配置文件中提取出了更加有效和简单的特征, 利用 FM 算法, 弥补了 SVM 算法在稀疏矩阵处理上的缺陷, 并且还可以表述出特征间的相关性。最终模型在多数据集的验证下精确度均达到了 99%以上。

上述两种方法都是仅仅使用了静态分析, 可能无法检测出来一些动态加载的恶意代码。文献[4]提出了利用静态分析和动态分析相结合的方式来检测 Android 恶意软件。文章利用静态分析提取出 APP 申请的权限和敏感 API 调用; 利用 DroidBox 进行动态分析并且实时监控 APP 行为, 得到 13 个 APP 行为特征。如 action_sendnet, 即把数据通过网络发送出去; action_phonecalls, 即拨打电话。文章将静态分析与动态分析得到的特征组合起来, 最终得到 192 个特征进行最后的训练。在模型的选择上, 考虑到机器学习算法结构层次较浅, 只有少于 3 层的计算单元, 因此选择了深度置信网络(Deep Belief Networks, DBN)。文章最后进行了静态分析、动态分析以及多种算法模型如 C4.5、SVM、朴素贝叶斯、logistic 回归和多层感知机、深度置信网络的组合对比, 最终发现利用静态分析和动态分析相结合并且采用深度置信网络所得到的分类结果精确度最高。

基于大量 Android 恶意软件研究成果的发布, Zhu 等人[18]提出了一种从文献中挖掘 Android 恶意软件特征的方法。文章从以往的相关科学文献中挖掘和提取与恶意软件有关的恶意行为, 并将恶意行为映射到具体的特征上, 这些特征包含 132 个权限、189 个 intent 和 11373 个 API 调用。进而建立语义网络, 计算每个特征的权重。最后选择出权重最高的一些特征作为进行训练的特征。文章选用了随机森林(Random Forest, RF)算法来训练模型并测试结果。此项研究从以往文献中挖掘出了 Android 恶意软件的一些较为关键的特征, 另外还发现了一些在手工提取的过程中容易被忽视的特征。

Xu 等人[19]提出的 DeepRefiner 采用了{0, 1}型特征与序列型特征相结合的两轮处理方式。第一轮处理仅使用了{0, 1}特征。文章认为 APP 里的 XML 文件中的值在良性和恶意软件中表现会有所不同, 所以首先对 AndroidManifest.xml 和/res/路径下 XML 文件进行预处理, 建立 XML 值的数据库, 将每个 APP 的 XML 信息值以 one-hot 矩阵的形式表示出来作为特征, 输入多层感知机(Multilayer Perceptron), 最后采用 Softmax 激活函数输出预测的此 APP 的良性概率和恶意概率。需要指出的是, 当良性概率和恶意概率差值很大的时候, 判别器可以确定这个输入 APP 的性质; 但当概率差值不大的时候, 判别器无法确定这个 APP 是良性的还是恶意的, 因此会进入第二轮计算。第二轮采用了序列型特征, 会在下一小节具体讨论。

### 3.2 序列型特征

DeepRefiner[19]的第二轮处理采用了序列型特征。文章首先对 dex 文件反编译得到的字节码进行简化, 然后将字节码与上下文进行组合得到语义序列, 利用自然语言处理领域 Word2Vec 的思想, 进行 Skip-Gram 建模, 把组合后的特征映射到新的空间。为了避免普通循环神经网络(Recurrent Neural Network, RNN)梯度消失问题, 文章选择了长短期记忆网络(Long Short Term Memory, LSTM)为最终的训练模型。

除了基于字节码序列, 早前就有研究者们提出了基于 API 序列来检测 Android 恶意软件的方法。以 API 序列构造的特征模式可以发现恶意软件家族共有的重要行为特征, 容忍小部分不同的代码实现。恶意软件家族中即使不同的软件代码实现不同, 但其

行为逻辑类似, 反映在 API 序列上也是基本相同的。因此使用 API 序列构造特征在检测变种恶意软件上具有一定的优势, 近年来广泛地被研究者们所使用。文献[20]通过建立基于上下文的带权 API 依赖图(weighted contexual API dependency Graph, WC-ADG), 分别建立良性软件与恶意软件 WC-ADG 的数据库。训练时, 首先从 APP 中提取出 WC-ADG, 并将此 WC-ADG 与数据库中的 WC-ADGs 进行相似度计算, 得到的相似度向量作为特征值, 利用朴素贝叶斯算法进行训练。

除此之外, 文献[21]提出了 MaMaDroid, 一种基于构造马尔可夫链, 以 API 调用序列转化概率作为特征训练机器学习模型的 Android 恶意软件检测方法。文章首先通过提取 APP 的控制流图, 构建 API 调用序列。由于直接处理原 API 调用序列需要消耗大量资源, 因此对原序列进行了简化处理, 例如 java.lang.Throwable:getMessage 使用 Package 名 java.lang 或 family 名 java 来替代。之后以 API 调用之间的转化关系建立马尔可夫链, 以转化概率作为特征向量, 输入到多种机器学习算法模型如 SVM、随机森林、1 近邻、3 近邻中训练, 最终得到判别模型。

Hou 等人[7]为了挖掘不同实体之间更加复杂关系、得到更高层次的语义信息, 提出利用异构信息网络 (Heterogeneous Information Network, HIN)来展现其中的复杂关系的方法 HinDroid。HinDroid 利用 Feature Extractor 模块提取完整的 Android API 调用列表, 并进一步分析提取的 API 调用之间的关系, 主要关注于 API 调用是否属于同一个包、同一个代码段以及是否含有同样的 invoke 方法等。文章基于先前提取的关系特征利用异构信息网络构造模块构造 HIN, 形成代表不同元路径的可交换矩阵。最后使用标准多核学习(Multiple Kernel Learning, MKL), 优化不同元路径的权重, 组合不同的可交换矩阵以形成用于 Android 恶意软件检测的更强大的核函数。对于每个新收集的未知的 Android 应用程序, 经过一系列预处理得到其 API 调用及相互关系后, 使用构建的分类模型, 判断该应用程序为良性或恶意。

有研究者考虑到静态分析无法检测到一些动态执行的恶意代码, 尝试利用动态分析的方式进行恶意软件检测。Hou 等人[22]提出的 Deep4MalDroid 通过分析 linux 系统调用识别 Android 恶意软件。文章通过提取 Android 组件列表, 动态执行组件并记录下系统调用, 将系统调用图用加权有向图表示出来。训练时先利用 SAEs(Stacked AutoEncoders)预训练权重, 再利用多种模型如 SVM、朴素贝叶斯、决策树、ANN 及深度学习等算法进行分类。

随着物联网产业的发展, 近年来越来越多使用 Android 系统的 IoT 设备出现在市场上, 一些研究者就提出了不仅可以运行在服务器上同时也可以运行在手机端和 IoT 设备端的 Android 恶意软件检测方法 MalDozer[23]。MalDozer 使用 API 方法调用的原始序列集合作为输入, 在训练过程中自动进行特征提取, 利用然语言处理领域 Word2Vec 的思想, 生成易于计算的稠密矩阵。由于特征提取过程与训练过程相互独立, 因此提高了方法的运行效率。同时由于方法的低开销, 在手机端甚至 IoT 设备上也能很好地运行。

### 3.3 端到端模式

卷积神经网络(Convolutional Neural Network, CNN)是深度学习领域中比较有代表性的一种方法, 主要被使用于图像识别问题上。CNN 能够自动提取感受野内的重要特征, 这意味着在 Android 恶意软件检测方面 CNN 也可能自动提取局部区域操作码之间的特征, 避免人工分析和提取特征的诸多问题。于是 McLaughlin 等人[24]开始利用 CNN 进行 Android 恶意软件检测。该方法受到基于 n-gram 的恶意软件检测的启发。先通过反编译 APP 的 dex 文件获取字节码的操作符, 将每个操作符映射成一个操作数。*n* 个操作数作为 CNN 的输入, CNN 会自动地识别恶意软件的特征。这种方法克服了传统的 n-gram 方法中 *n* 不能过大的限制。

以往的研究方法一般都需要比较大量的计算, 因此就有研究者提出一种基于 CNN 的轻量级 Android 恶意软件检测方法[25]。文章仅截取了 APK 文件的开头或结尾 *N* 个字节(实验中 *N* 分别取 512、1024、2048 或 4096), 采用 1D 卷积的 CNN 来进行训练。最后模型在不同的数据集上取得了 95%~96%的准确度。

Huang 等人[26]提出了 R2-D2,一种利用图像识别领域思想检测 Android 恶意软件的方法。文章通过反编译 APP 的 dex 文件获取字节码, 将字节码通过一定的映射规则表示成对应的 RGB 图片格式。最后将 APP 表示成的 RGB 图片输入到 CNN 中进行训练。模型在不同的数据集上取得了 92%~98%的准确度。

## 4 Android 恶意软件检测对抗攻击

随着越来越多的针对 Android 恶意软件检测的

**表 1 Android 恶意软件检测算法重要因素对比**

**Table 1 A comparison of the important factors in Android malware detection algorithm**

| 文献 | 分析方式 | 特征类型 | 特征 | 使用算法 | 数据集来源 | 样本数量 | 准确率(%) |
|---|---|---|---|---|---|---|---|
| Drebin[16] | 静态 | 0, 1 型 | 硬件组成, 申请的权限, App 组件, Filtered intents, 可疑 API 调用, 网络地址 | SVM | Google Play, 中国三方市场, 俄罗斯三方市场, 其他来源(Android 网站、安全博客等) | 131611 | 94 |
| Android 恶意软件检测 FM 算法[17] | 静态 | 0, 1 型 | APP 组件, 硬件组成, 申请的权限, Filtered intents, 限制 API, 可疑 API, 使用的权限 | FM | Drebin, AMD, Apkpure, 360.com, HKUST Wake Lock Mis-use Detection Project | 19100 | 99 |
| Droiddetector [44] | 静态+动态 | 0, 1 型 | 申请的权限, 敏感 API 调用, 动态收集的行为 | DBN | Google Play, Contagio community, Genome | 21760 | 96.76 |
| FeatureSmith [18] | 静态 | 0, 1 型 | 132 个权限, 189 个 Intents, 11373 个 API 调用 | RF | - | - | 92.5 |
| DeepRefine [19] | 静态 | 0, 1 型+序列型 | XML 值, 字节码及上下文 | MLP+LSTM | Google Play, VirulShare, Mass Vet | 110440 | 97.74 |
| WC-ADG 图法[20] | 静态 | 序列型 | 基于上下文的带权 API 依赖图 | NB | Google Play, McAfee, Genome | 15700 | 93 |
| MaMaDroid [21] | 静态 | 序列型 | 控制流图 | RF | PlayDrone, Google Play, Drebin, VirusShare | 43940 | 97~99 |
| Deep4MalDroid [22] | 动态 | 序列型 | Linux 系统调用图 | DL | Comodo | 3000 | 93.68 |
| HinDroid [7] | 静态 | 序列型 | API 调用及不同实体间的依赖关系图 | MKL | Comodo | 31834 | 98 |
| MalDozer [23] | 静态 | 序列型 | API 调用序列 | CNN | Genome, Drebin, VirusSjhare, Contagio Minidump | 33000 | 96~99 |
| 基于 CNN 的 Android 恶意软件检测[24] | 静态 | 端到端 | - | CNN | Genome, Google Play, McAfee | 26225 | 98 |

续表

| 文献 | 分析方式 | 特征类型 | 特征 | 使用算法 | 数据集来源 | 样本数量 | 准确率(%) |
|---|---|---|---|---|---|---|---|
| 1D 卷积法[25] | 静态 | 端到端 | - | CNN | AMD, Drebin, Appsapk, Apkpure | 7000 | 95~96 |
| R2-D2[26] | 静态 | 端到端 | - | CNN | Google Play, Contagio Minidump, VirulTotal | 1500000 | 92~98 |

人工智能算法模型的使用, 攻击者们开始考虑如何精巧地绕过这些检测算法。根据攻击者掌握机器学习模型信息的多少, 绕过 Android 恶意软件检测模型的攻击场景可以分为黑盒攻击和白盒攻击两种:

- 白盒攻击: 攻击者能够获知检测模型所使用的算法, 以及算法所使用的参数。攻击者在产生对抗性攻击数据的过程中能够与检测系统有所交互。此时攻击者可以直接对原有模型进行对抗攻击并生成对抗样本, 对抗样本可以直接用原有模型进行验证。

- 黑盒攻击: 攻击者不知道检测模型所使用的算法和参数, 但攻击者仍能与检测系统有所交互。如, 可以通过传入任意输入观察输出, 判断输出。此时攻击者通常需要构造原有模型的替代模型, 然后使用对抗学习的方法使得替代模型的功能与原有模型近似。在替代模型上产生的对抗样本往往对原有模型有效。

本小节从绕过 Android 恶意软件检测模型的攻击场景出发, 分别阐述针对不同检测模型的白盒攻击方法和黑盒攻击方法。

## 4.1 白盒攻击

### 4.1.1 0, 1 型特征

对于 Drebin[16]这种早期的 Android 恶意软件检测模型, 通过一些简单的混淆操作就可以绕过。文献[27]指出了一系列投毒攻击, 譬如在 AndroidManifest.xml 文件中添加许多与软件运行过程无关的且多在良性软件中出现的权限信息, 或者将代码中的一些敏感 API 藏在 png 图片中, 以及在代码中嵌入一些良性代码段或将代码通过 java 反射动态加载等。但随着检测模型特征所涵盖的信息逐渐复杂和算法鲁棒性的增强, 检测精度不断提高, 简单混淆逐渐难以绕过一些复杂模型的检测。一些研究者开始探讨将在图像领域“对抗样本”的概念和方法迁移到 Android 恶意软件检测领域中来。

基于雅各比显著性图的攻击(Jacobian-based Saliency Map Attack, JSMA)是最初应用于图像领域的一种对抗攻击方法[28]。该方法通过限制 $l_0$ 范数进行对抗攻击, 使得生成的对抗样本和原图像只在几个像素上有所差别。攻击过程通过监控输出的梯度计算一个显著性图, 修改显著性最强即最重要的像素, 直至达到最大允许修改的像素数量或欺骗模型成功。但是图像领域存在这些较为成熟的对抗攻击方法往往针对连续型特征值, 恶意软件检测领域很多方法都是利用了离散型的{0, 1}特征。因此就有研究者在继承图像领域对抗攻击思想的基础上改进算法使之更适应恶意软件检测领域的对抗攻击。

Grosse 等人[29]提出的产生 Android 恶意软件对抗样本的思想与 JSMA 类似, 文章关注离散型的{0, 1}特征。由于恶意软件的特征代表自身的配置信息或功能, 单纯地修改恶意软件的特征可能会导致软件崩溃或原有功能的丢失, 因此文章在修改特征时仅仅会添加特征。文章通过如下公式计算雅各比矩阵, 得到模型 $F$ 对于样本 $X$ 的梯度来获得扰动的方向,

$$J_F = \frac{\partial F(X)}{\partial F} = \left[ \frac{\partial F_i(X)}{\partial X_j} \right]_{i \in 0,1, j \in [1,m]}$$

算法会添加分类结果显著性最高的特征, 然后继续迭代直至达到最大允许添加的特征数量或成功欺骗模型。

有研究者提出了专门针对{0, 1}型特征的对抗攻击方法[30]。文章提出了一种基于梯度生成对抗样本的方式——BCA$^k$(multi-step Bit Coordinate Ascent)。该方法需要多轮迭代, 每轮从未拥有的特征中添加梯度分量最大的。具体原理如图 1。

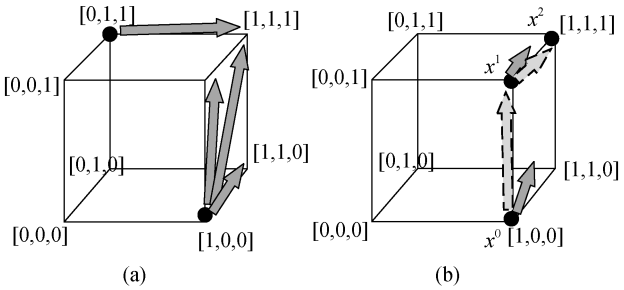



**图 1　BCA$^k$ 算法三维向量空间示意图**

**Figure 1　BCA$^k$ algorithm in the 3-dimensional vector space**

图 1 表示样本的特征空间, 每个点表示一个样本。图 1(a)中, 箭头方向表示允许添加的扰动方向。由于扰动不能使样本软件崩溃或丢失原有的功能, 因此特征只能添加不能删除。以[1,0,0]点的恶意软件为例, 其可以修改为的样本集合为: S([1,0,0]) = {[1,0,0], [1,1,0],[1,0,1],[1,1,1]}; [0,1,1]点的恶意软件可以修改为的样本集合为: S([0,1,1]) = {[0,1,1], [1,1,1]}。图 1(b)为 BCA$^k$ 算法的流程, 实线箭头表示梯度方向, 虚线为扰动的方向。根据 BCA$^k$ 算法, 样本 $x^0$ 处最大的梯度分量指向 $X^1$ 方向, 因此将 $X^0$ 的第三个特征位修改为 1 到达 $X^1$, 同样地, 再通过将 $X^1$ 的第二个特征位修改为 1 到达 $X^2$, 最终将样本软件的损失沿梯度上升至目标值。

#### 4.1.2　序列型特征

文献[27]除了指出对 Drebin 的攻击, 也指出了对序列型检测模型 MaMaDroid[21]的攻击。文章将那些频繁在良性软件中出现的 API 序列添加到恶意软件中, 从而达到混淆特征序列的目的, 因此可以绕过模型的检测。但该种方法较为传统, 无法应对复杂的检测模型, 且没有通过自动化的方式生成大量对抗样本。

与上一小节 Grosse 等人类似, 利用 JSMA 思想进行对抗攻击的还有文献[31]。Chen 等人提出的 AndroidHIV 中也利用 JSMA 的思想产生对抗样本。与 Grosse 等人不同的是, AndroidHIV 除了{0, 1}型特征也关注了序列型特征。通过计算模型输出和输入特征 $X$ 之间的雅各比矩阵来进一步计算显著性图。以攻击 MaMaDroid 的模型为例, 通过计算 API 调用数目 $A$ 和替代模型 $F$ 输出之间的雅各比矩阵得到显著性图, 根据显著性图, 修改显著性最高的一个 API 调用。此过程不断迭代, 直至达到最大允许修改的 API 调用数量或欺骗模型成功。

同样是在图像识别领域, Carlini 和 Wagner 提出了三种针对图像识别模型的对抗攻击算法[32]。该算法通过限制 $l_2$、$l\infty$、$l_0$ 范数限制对图像中的扰动, 进而使扰动无法被人眼识别, 图像识别模型却能将图像错分。AndroidHIV[31]攻击中的另一种方法就是利用的这种思想。文章在 C&W 基础上进行了一些修改, 对于特征添加一定的扰动。通过以下限制优化下列函数:

$$\min_\delta \| \delta \|_2^2 + c \cdot f(X+\delta)$$
$$s.t. X+\delta \in [0,1]^n,$$
$$\text{and} \| X_g + \delta_g \|_1 = 1, g \in 1 \cdots k$$

其中, $\delta$ 表示要被优化的添加到样本中的扰动, $c$ 是平衡两项的置信度常数。第一项 $\min_\delta \| \delta \|_2^2$ 代表通过 $l_2$ 范数的限制来最小化需要修改的特征, 第二项 $f(X+\delta)$ 表示损失函数。损失函数定义如下:

$$f(X) = \max(Z(X)_t - \max\{Z(X)_i : i \neq t\}, -\kappa)$$

其中, $t$ 是当前样本 $X$ 的正确的标注, $Z(X)$是替代模型的 pre-softmax 输出, $\kappa$ 是可以调节置信度的超参数。损失函数 $f$ 的作用是计算目前模型产生的误分类结果和正确结果间的差异。在攻击不同的检测模型时, 将扰动对应到不同的特征类型的修改上, 如对 MaMaDroid 模型添加扰动就是指对 API 调用数量进行修改进而扰乱原来的 API 调用转化概率矩阵。最后通过梯度下降法找到最小扰动并生成攻击样本。

#### 4.1.3　端到端模式

对于其他平台的恶意软件(如, windows 平台下的 PE 类型文件), 已有研究者提出了各种对抗检测模型的方法。针对恶意软件的对抗攻击其中一个难点在于在修改样本的过程中如何保留原样本软件的功能。所以除了小心地增加或移除 API 调用或其他特征, 文献[33]提出了一种白盒攻击方法, 这种方法不同于其他直接对特征进行修改的方式, 而是采用了只在可执行文件的文件尾进行填充少量字节来躲避检测的方法。由于填充字节并不破坏原软件的代码和数据, 因此可以很好地避免软件原有功能遭到破坏。文章攻击的对象 MalConv[34]模型输出的是某一样本为恶意软件的概率, 因此文章计算添加填充位后的样本为恶意的概率, 最小化这个输出值, 同时限制添加的字节数不能过大, 利用梯度下降法找到合适的对抗样本。

由于 Android 软件中的 dex 文件与 windows 平台下的 PE 文件类似, 都属于某一平台下的可执行文件, 也都存在为了对齐字节产生的填充位, 修改此类填充位对可执行文件本身无影响。因为此类在文件末

尾添加填充位来逃避检测的方法也适用于 Android 恶意软件检测模型对抗样本的生成。

## 4.2 黑盒攻击

### 4.2.1 0, 1 型特征

在黑盒条件下, 有些高效的 Android 恶意软件检测模型的性能也很容易受到干扰。文献[35]针对关注{0, 1}型特征的 Drebin 模型指出了一些潜在的攻击场景, 其中包括工具混淆攻击、模仿攻击、替代模型攻击等。文章针对工具混淆攻击做了较为详细的阐述。工具混淆攻击利用了 DexGuard, 这是一个 Android APP 代码混淆和加密工具, 其中包含了字符串加密、反射、类加密、混合混淆等混淆加密手段。文章发现, 仅仅使用 DexGuard 就能对 Drebin 产生一定干扰。

Yang 等人[36]提出了两种针对 Android 恶意软件检测模型黑盒攻击方案: 进化攻击和特征模糊攻击。进化攻击模仿了恶意软件自身迭代和变种的过程, 并使其自动化。文章选择了四种重要类型特征(资源、触发时间、触发位置、依赖)和上下文特征, 通过建立进化树, 理解恶意软件家族的迭代过程。最后根据不同类型特征的可行度和统计学频率计算评分函数, 用评分最高的一些特征进行更新迭代。每次迭代后进行测试验证, 将没有成功逃避检测的样本继续迭代, 直至生成成功逃过检测的样本或样本程序无法执行。

特征模糊攻击是把一些特征转化为良性软件和恶意软件中都有的模糊特征, 从而逃避检测。文章从上述提到的四种重要特征和上下文特征中找出良性软件和恶意软件共有的部分, 通过良性软件的计数来计算不同特征的权重。生成对抗样本时, 根据输入软件每个特征找到模糊特征集中最接近的模糊特征, 将此特征修改为对应模糊特征或修改其对应的上下文特征, 从而达到混淆检测模型的目的。但需要指出的是, 由于混淆攻击修改的特征会很多, 导致很多软件在被修改后不能运行。最后 Yang 等人提出的对抗样本生成模型利用以上两种攻击方法, 结构如图 2 所示。文章最后将修改后的攻击样本输入到多种 Android 恶意软件检测工具如 VirusTotal、AppContext 和 Drebin 中, 均取得了不同程度的成功。

### 4.2.2 序列型特征

近年来很多研究者利用基于序列型 API 特征的人工智能算法来检测恶意软件, 因此有研究者思考黑盒场景下通过对 API 序列进行修改来绕过模型检测。Chen 等人[37]通过分析 Comodo 云安全中心提供

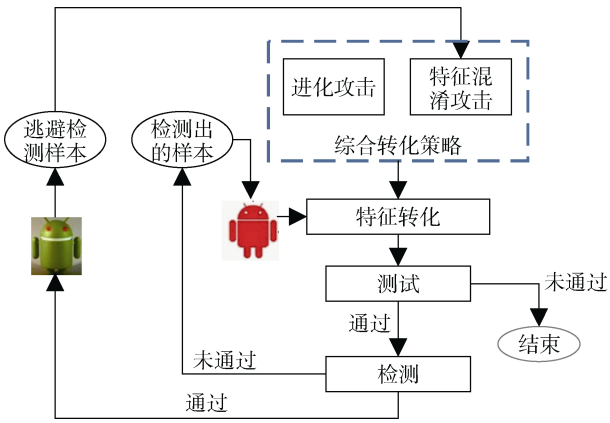



**图 2 对抗样本生成模型示意图**

**Figure 2 Illustration of adversarial examples generation model**

的 10000 个带标注的软件样本, 计算出不同 API 在良性软件与恶意软件中的相关性得分, 通过添加与良性软件相关性高的 API 以及删除与恶意软件相关性高的 API 来逃避模型检测。

在恶意软件检测领域, 存在许多以 RNN 为代表的序列型人工智能算法检测模型[19]。Hu 等人[38]就提出了一种针对 RNN 模型的黑盒攻击方式。文章训练了一个用于近似原黑盒 RNN 模型的替代模型, 另外提出了一个生成 RNN 模型用来根据原始输入的恶意软件样本生成对抗样本。其模型结构如图 2 所示。训练过程旨在使得替代模型学习原模型更多的信息, 让生成 RNN 模型生成的对抗样本特征更不容易被原模型检测出来。

### 4.2.3 通用方法

目前已经出现一些 Android 恶意软件采用抗逆向分析技术, 如代码加壳等手段来逃避恶意软件检测模型的检测[39]。此类方法主要针对一些静态分析模型, 使得模型无法正常反编译 apk 得到代码逻辑, 难以提取出样本的静态特征, 从而无法正确判定样本恶意与否。

Hu 等人[40]提出了 MalGAN, 一种利用生成对抗网络(Generative Adversarial Networks, GAN)的思想生成对抗样本的方法。MalGAN 使用一个替代鉴别器来匹配原有的黑盒检测模型, 生成器用来生成恶意样本。训练过程中, 替代鉴别器通过最小化如下损失函数来使自身更接近原黑盒检测模型:

$$L_D = -E_{x\in BB_{Benign}} \log\left(1 - D_{\theta_d}(x)\right) - E_{x\in BB_{Malware}} \log D_{\theta_d}(x).$$

定义预测的样本 $x$ 的恶意概率为 $D_{\theta_d}(x)$, $\mathrm{BB}_{\mathrm{benign}}$ 是原黑盒检测模型识别出的良性软件, $\mathrm{BB}_{\mathrm{malware}}$ 是原

黑盒检测模型识别出的恶意软件。生成器通过最小化如下损失函数来降低替代鉴别器检测出的恶意软件概率:

$$L_G = E_{m\in S_{Malware}, z\sim p_{uniform}[0,1)} \log D_{\theta_d}(G_{\theta_g}(m,z)).$$

其中 $S_{\text{malware}}$ 表示真实的恶意软件集合。

整体过程如图 3 表示。

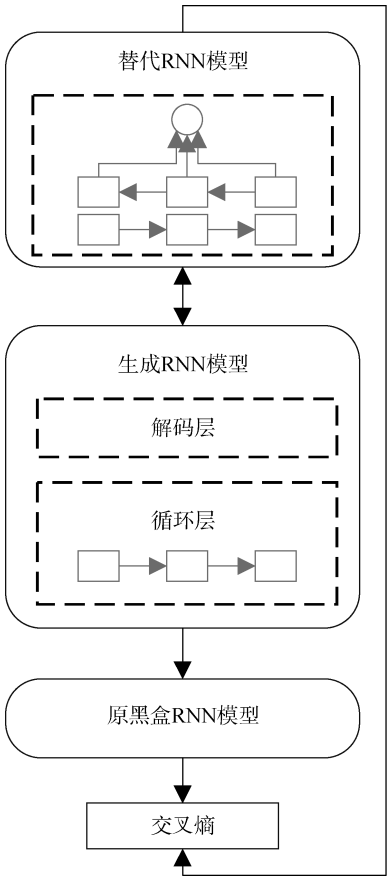



**图 3　针对基于 RNN 算法的对抗攻击模型**

**Figure 3　Adversarial attack model in the RNN-based algorithm**

利用 MalGAN 生成的对抗样本取得了非常好的效果，甚至能够使某些模型对这些对抗样本的检出率降低到几乎为 0。

## 5　Android 恶意软件对抗攻击的防护手段

随着恶意软件对抗样本的发展，研究者们也开始考虑如何提高 Android 恶意软件检测模型的安全性。本小节从针对特征的防护和针对算法的防护两方面出发，概述当前防止对抗攻击和增强 Android 恶意软件检测模型安全性的方法。

### 5.1　针对特征的防护

文献[35]在提出针对 Drebin 攻击之后又提出了针对这类问题的一些增强安全性的手段。Drebin 的可解释性暴露了特征的权重，使得攻击者可以通过修改权重最大的少量特征来绕过模型的检测，因此文章指出，可以在模型训练过程中添加框式约束(box constraint)将权重限制在一定范围之内，从而使特征权重分布更平均，进而提高对抗攻击的难度。

Chen 等人[41]提出了 SecureDroid 方法，用来提高 Android 恶意软件检测模型面临对抗攻击时的鲁棒性。以往 Android 恶意软件检测模型都是基于不同的特征具有不同权重的思想(例如，Drebin)；攻击者在生成对抗样本时对每个特征进行修改所花费的代价是不同的，比如修改配置文件容易而修改 dex 文件更困难，因此文章在增强模型鲁棒性和安全性方面时考虑到了这两方面。文章提出了一种新的特征选择方法 SecCLS，这种方法提取特征时综合考虑了特征权重和修改代价。文章选取了权限、Filtered Intents、API 调用和 New-Instance 4 种类型的特征，首先使用机器学习方法获得每个特征的权重，并且计算出针对{0, 1}型特征进行特征修改所花费的代价。最后选择那些权重低且修改代价大的特征进行训练。文章在训练过程提出了 SecENS 方法。SecENS 集成了多种不同分类器，并且尽最大可能覆盖整个特征空间，使最后生成的模型鲁棒性得到了增强。SecureDroid 的模型架构如图 5 所示。

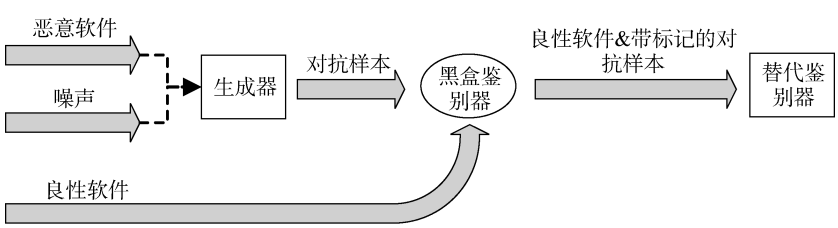



**图 4　MalGAN 模型架构图**

**Figure 4　MalGAN model architecture diagram**

**表 2　恶意软件对抗攻击重要因素对比**

**Table 2　A comparison of the important factors in malware adversarial attack**

| 文献 | 攻击场景 | 特征类型 | 对抗模型 | 主要思想 | 样本数量 | 效果 |
|---|---|---|---|---|---|---|
| 投毒攻击[27] | 白盒 | 0, 1 型, 序列型 | Drebin, MaMaDroid 等 | 特征混淆 | 1000 | 错误率 75.2%, 错误率 68.95% |
| 基于 JSMA 的攻击[29] | 白盒 | 0, 1 型 | Drebin | 基于梯度进行修改 | 129013 | 错误率 85% |
| BCA$^k$法[30] | 白盒 | 0, 1 型 | SLEIPNIR | 基于梯度进行修改 | 38000 | 错误率 99.7% |
| Android HIV[31] | 白盒 | 0, 1 型, 序列型 | Drebin, MaMaDroid | 基于梯度进行修改 | 5879 | 精确度由 96%降低至 1%, 精确度由 97%降低至 1% |
| 端到端攻击[33] | 白盒 | 端到端 | Malconv | 在文件尾进行特征添加 | 13195 | 错误率 60% |
| 混淆攻击[35] | 黑盒 | 0, 1 型 | Drebin | 代码混淆 | 10500 | 精确度由 97%降低至约 80% |
| 特征进化和特征模糊攻击[43] | 黑盒 | 0, 1 型 | VirusTotal, AppContex, Drebin | 特征进化, 特征模糊 | 3852 | 错误率 12%~97% |
| 针对 RNN 的黑盒攻击[38] | 黑盒 | 序列型 | 基于 RNN 的恶意软件检测模型 | 替代模型 | 180 | 错误率 90% |
| MalGAN[40] | 黑盒 | - | VirusTotal | GAN | 38000 | 最高错误率 100% |

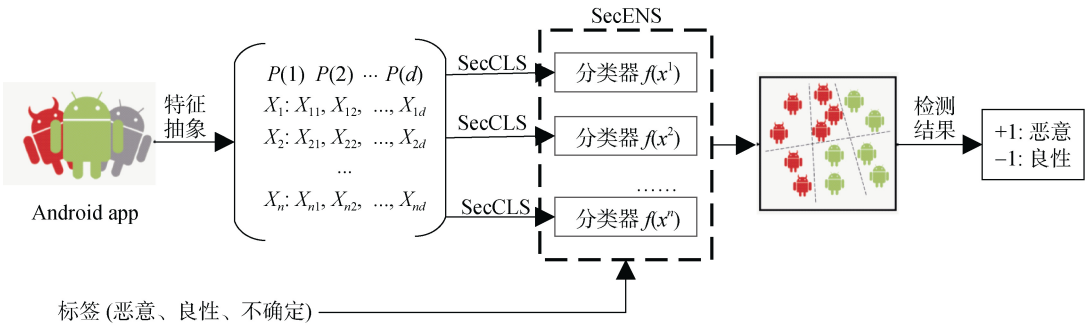



**图 5　SecureDroid 模型架构图**

**Figure 5　SecureDroid model architecture diagram**

## 5.2　针对算法的防护

文献[27]针对投毒攻击提出了识别 Android 恶意软件检测模型输出结果中假阴性软件的方法。文章首先通过手工筛选, 筛选出最良性的和最恶意的两类 Android 应用, 基于语法和语义特征, 通过计算训练集中的 Android 应用和手工筛选出来的两类应用的 Jaccard 相似度和余弦相似度, 利用相似度来判断某个 Android 应用更像良性软件还是恶意软件, 从而识别出那些伪装过的恶意软件。Grosse 等人在文献[21]中提出利用 JSMA 思想对离散型{0, 1}特征进行对抗攻击的方法后, 指出了两种可行的增强模型安全性的防护方案。

第一种是蒸馏法。蒸馏法引入了一个和原神经网络 $F$ 相同的神经网络 $F'$, 通过训练 $F'$使模型达到更好的分类效果。与原网络 $F$ 的训练过程不同的是, $F'$的训练集标签不是简单的 0 或 1, 而是原网络 $F$ 输出的概率分布(如, 使用 softmax 作为输出层得到的概率分布结果)。比起利用简单的{0, 1}标签, 概率分布标签包含了训练集中每个成员对应每个类别的更多信息。文章对神经网络输出层 softmax 函数进行了修改, 使用下列函数:

$$F_i(X)=\frac{e^{z_i(x)/T}}{\sum_{l=1}^{|y|}e^{z_l(x)/T}}$$

$T$ 是蒸馏参数。$T$ 越大, 输出的概率分布越统一。在训练原网络 $F$ 和新网络 $F'$的过程中都使用了相同的较高的参数 $T$。最终实验发现新网络 $F'$确实可以不同程度地提升在对抗样本干扰情况下检测的准确度, 但是提升的效果比较轻微。

第二种是再训练法。再训练法在训练神经网络的同时会使用前文对抗攻击的方法产生对抗样本, 并将对抗样本不断地加入到训练过程中, 以此提高最终模型的鲁棒性。经实验发现在训练过程中添加一定数量的对抗样本可以提高模型的抗对抗攻击能力, 但如果训练中添加的对抗样本大于一定数量则会导致模型误分率明显提高, 使得模型的准确度和抗对抗攻击能力降低。因此使用此类方法

增强模型安全性的时候需要谨慎地选择添加到训练过程中的的对抗样本的数量，否则可能对模型产生相反的效果。再训练法可以与其他增强安全性方法相结合[37]。如可以通过在原模型的损失函数上增加一个安全规范化项来加强模型安全性。攻击代价表示攻击者针对此模型进行对抗攻击时对原恶意样本进行修改的产生修改代价，安全规范化项取代价的倒数，因此训练过程在最小化损失函数的同时可以最大化攻击者产生对抗样本的攻击代价，从而提高模型安全性。

虽然在训练过程中加入对抗样本的方式能够取得比较明显的效果，但是利用一些已有方法如FGSM[42]，可以有效地快速生成大量对抗样本，而若想抵御这些对抗样本就需要在每发现一个对抗样本后就将对抗样本加入训练集重复训练过程。这就造成了增强后的模型不能很好地应对对抗样本的变化，导致安全性无法得到彻底的提升。文献[45]提出了一种针对深度神经网络(Deep Neural Networks, DNN)模型提高对抗样本生成难度的方法。文章通过在输入层与第一个隐藏层之间增加一个中间层，这个中间层的作用是随机地隐藏一些特征。正是这些随机被隐藏的特征，会给攻击者在计算梯度的时候带来困难，因此可以防止攻击者通过简单地计算梯度生成对抗样本。最后作者在大量真实存在的恶意软件和良性软件中做测试，结果证实了方法的有效性。

## 6 思考与讨论

### 6.1 Android 恶意软件检测模型发展趋势

目前大部分 Android 恶意软件检测系统都是基于静态分析的方法，很多恶意软件为了躲避检测，尝试利用各种方式隐藏自己的恶意特征。许多恶意软件能够通过动态加载的方式逃避这些模型的检测，包括恶意代码动态加载、隐藏进图片或其他资源文件中等。另外还存在一些恶意软件采用了抗逆向分析技术，如 apk 加壳等手段。因此需要在检测模型中引入动态分析方法，采用动静结合的分析方式来全面地发现恶意软件。

传统的 Android 恶意软件检测方法通常从应用的配置文件、代码文件中提取特征作为检测模型的训练样本，然而提取特征的过程中，通过人工的筛选抽象，最终得到的特征损失了原有应用的大量信息，这限制了模型的准确率，因此 Android 恶意软件检测模型会考虑更多更加复杂的特征，例如使用图神经网络生成控制流图各个节点的嵌入，在应用与 API 等构成的异构图中提取表征应用内部信息的嵌入等。为增加检测模型的准确率与鲁棒性，通常将多种恶意软件检测模型融合为堆叠模型。在堆叠模型中集合了多种分类器，这些分类器并行计算后得到各自预测的概率，然后将这些概率作为特征输入，经过堆叠分类器得到更准确的结果。模型架构如图 6 所示。

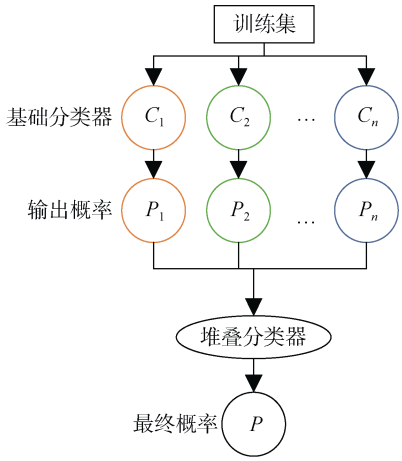



**图 6 Android 恶意软件堆叠检测模型**

**Figure 6 Android malware stacked detection model**

随着 IoT 产业的发展，许多智能设备采用了 Android 系统。但绝大多数基于 Android 的智能设备如智能音箱、智能网联车的车机系统等，都没有采用较新版本的 Android 系统。使用旧版本的 Android 系统导致智能设备无法享受到新 Android 版本的安全特性和安全补丁，从而导致智能设备暴露在低版本 Android 系统漏洞的风险中。在这种情况下，Android 恶意软件可发挥的空间更大，甚至可以进一步提权获取 root 权限。另外，由于智能设备本身资源受限的情况，普通的高开销、高计算方法并不适用于智能设备上的 Android 恶意软件检测。因此随着市场上出现越来越多的基于 Android 系统的智能设备，针对智能设备的轻量级 Android 恶意软件检测方案在未来会得到很大的发展。

Android 恶意软件检测技术不仅需要在移动设备中实时运行，还需要在安全厂商进行安全审计、态势感知时发挥功效，这些场景下对模型的运行速度、准确性的要求不同，因此需要不同的 Android 恶意软件检测方法，例如在智能设备上使用更加轻量级的模型，安全厂商为平衡计算资源在云端使用多层检测模型，越深层的模型使用越复杂的特征和模型以获得更准确的结果。模型架构如图 7 所示。

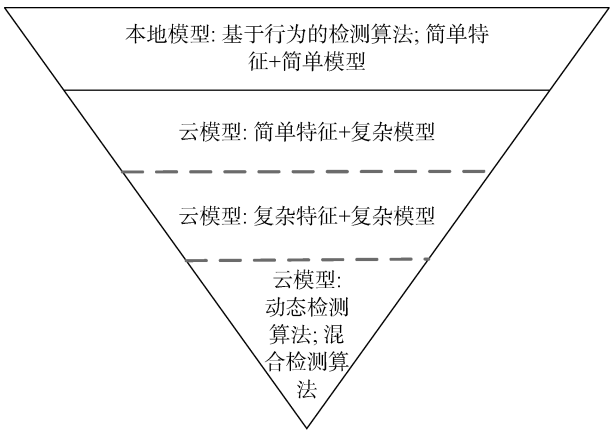



**图 7 Android 恶意软件分层检测模型**

**Figure 7 Android malware layered detection model**

### 6.2 Android 对抗样本发展趋势

目前针对图像领域的对抗攻击方法较为成熟, 以上提到的基于 FGSM[42]、C&W[32]、JSMA[28]等算法的 Android 对抗样本生成方法都是利用了图像领域对抗攻击的思路。对图像像素的修改可以对应到对 Android 恶意软件某一特征(API 调用、权限等)的修改, 需要研究者考虑的主要就是如何对原算法进行一定的改进, 使得在修改过程中不破坏原软件的功能。因此在未来越来越多图像领域的对抗样本生成方法会迁移到 Android 恶意软件对抗样本的生成上。如对 FGSM 方法改进的扰动更小的 DeepFool 方法[46], 通过迭代计算的方法生成最小规范对抗扰动, 将位于分类边界内的图像逐步推到边界外, 直到出现错误分类。相对于已有的基于 FGSM 算法生成 Android 对抗样本的方法, DeepFool 可以在迭代过程进行优化, 解决FGSM中扰动系数的选择问题, 并且可以通过多次线性逼近实现对一般非线性决策函数的攻击。

但白盒攻击应用场景受限, 实际的场景中更容易使用黑盒攻击。目前最广泛使用的黑盒攻击方法为使用替代模型学习黑盒模型足够多的信息, 对替代模型利用白盒攻击方法生成的对抗样本在原模型中有效。同时也可以基于混淆的思想, 使用遗传算法或强化学习算法通过不断试错确定生成对抗样本时应该添加的扰动方向。

### 6.3 对抗攻击对 Android 恶意软件检测的影响

由于对抗攻击对人工智能算法的威胁性直接关系到使用人工智能算法的 Android 恶意软件检测算法的可用性, 因此目前的 Android 恶意软件检测算法必须考虑针对对抗攻击的防御, 在设计模型时需要使其对于输入有较大的鲁棒性, 并且恶意软件检测在不同的应用场景下, 需要有不同的防御方法。

根据用户对模型的访问场景, 可以将其分为能够不限次数访问模型的情况以及在一定时间内对模型的访问次数有一定限制的情况, 前者对于防御的需求较高, 后者可以考虑较弱的防御方式。在获取 Android 恶意软件检测模型的特征提取、模型训练等各个流程中, 都能够使用防御对抗攻击的方法, 不同的方法可能对模型结构、模型准确率有一定的影响。在用户能够大量访问的场景下, 防御者可以使用组合模型、增加模型输入的复杂度等对原有检测模型修改较大的方法, 或者利用一些典型防御对抗的手段, 如对网络进行修改, 包括添加更多层或子网络、修改损失函数或激活函数等; 使用外部模型作为附加网络以及在学习过程中修改训练过程或样本等。对于安全性要求较低的场景, 例如用户无法大量访问的模型的情况, 攻击者进行黑盒攻击时很难通过替代模型在短时间内学习原有模型足够的信息, 攻击难度较大, 因此只需要采用在训练时修改损失函数或将对抗样本加入训练集等基本的防御方法。对于拥有多层恶意软件检测模型的系统, 深层的模型通常使用了较为复杂的特征, 并且可能使用了多种分类器混合, 对抗攻击的难度较大, 同时攻击者无法得知是否访问了该深层的模型, 因此该层的模型并不需要考虑对抗攻击, 只需要确保检测的准确率。

## 7 总结

目前越来越多的 Android 恶意软件检测引擎使用了人工智能算法, 因此就有攻击者开始尝试对 Android 恶意软件进行一定的修改, 使得 Android 恶意软件可以在保留本身的功能的前提下绕过这些基于人工智能算法的检测模型。这就是 Android 恶意软件检测领域的对抗攻击。

本文梳理了目前存在的基于人工智能算法的 Android 恶意软件检测模型, 概述了针对 Android 恶意软件检测模型的对抗攻击方法, 并从特征和算法两方面总结了相应的增强模型安全性的防护手段。最后提出了 Android 恶意软件检测模型和对抗攻击的发展趋势, 并分析了对抗攻击对 Android 恶意软件检测的影响。

## 参考文献

[1] Android Phones to Have 85IDC. RAY S. https://www.news18.

com/news/tech/android-phones-to-have-85-global-market-share-in-2018-idc-1862965.html. 2018.

[2] Android security: Malicious apps sneak back into Google Play after tweaks. ZDNet. https://www.zdnet.com/article/android-security-malicious-apps-sneak-back-into-google-play-after-tweaks/. 2018.

[3] 2018 年 Android 恶意软件专题报告. 360. http://blogs.360.cn/post/review_android_malware_of_2018.html. Feb. 2018.

[4] Bonett R, Kafle K, Moran K, et al. Discovering Flaws in Security-Focused Static Analysis Tools for Android Using Systematic Mutation[C]. *SEC'18: Proceedings of the 27th USENIX Conference on Security Symposium*, 2018: 1263-1280.

[5] Rastogi V, Chen Y, Jiang X X. DroidChameleon: Evaluating Android Anti-Malware Against Transformation Attacks[C]. *The 8th ACM SIGSAC symposium on Information, computer and communications security - ASIA CCS '13*, 2013: 329-334.

[6] Android 逃逸技术汇编, DroidSec,“ http://www.droidsec.cn/ android 逃逸技术汇编/”, 2016.

[7] Hou S F, Ye Y F, Song Y Q, et al. HinDroid: An Intelligent Android Malware Detection System Based on Structured Heterogeneous Information Network[C]. *The 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017: 1507-1515.

[8] Papernot N, McDaniel P, Goodfellow I, et al. Practical Black-Box Attacks Against Machine Learning[C]. *The 2017 ACM on Asia Conference on Computer and Communications Security*, 2017: 506-519.

[9] Zhou Y, Wang Z, Zhou W, et al. Hey, You, Get off of my Market: Detecting Malicious Apps in Official and Alternative Android Markets [C]. *Network and Distributed System Security Symposium*, 2012, 25(4): 50-52.

[10] Chen K Z, Johnson N M, D'Silva V, et al. Contextual policy enforcement in android applications with permission event graphs[C]. *Network and distributed system security symposium*, 2013: 234.

[11] Peng H, Gates C, Sarma B, et al. Using Probabilistic Generative Models for Ranking Risks of Android Apps[C]. *The 2012 ACM conference on Computer and communications security - CCS '12*, 2012: 241-252.

[12] Wu D J, Mao C H, Wei T E, et al. DroidMat: Android Malware Detection through Manifest and API Calls Tracing[C]. *2012 Seventh Asia Joint Conference on Information Security*, 2012: 62-69.

[13] Yang C, Xu Z Y, Gu G F, et al. Droid*miner:* Automated Mining and Characterization of Fine-Grained Malicious Behaviors in Android Applications [C]. *Computer Security - ESORICS 2014,* 2014: 163-182.

[14] Grace M, Zhou Y J, Zhang Q, et al. RiskRanker: Scalable and Accurate Zero-Day Android Malware Detection[C]. *The 10th international conference on Mobile systems, applications, and services - MobiSys '12*, 2012: 281-294.

[15] L.K. Yan and H. Yin. DroidScope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis[C] *Usenix Security Symposium*, 2012.

[16] Arp D, Spreitzenbarth M, Hübner M, et al. Drebin: Effective and Explainable Detection of Android Malware in your Pocket[C]. *2014 Network and Distributed System Security Symposium*, 2014.

[17] Li C L, Mills K, Niu D, et al. Android Malware Detection Based on Factorization Machine[J]. *IEEE Access*, 2019, 7: 184008-184019.

[18] Zhu Z Y, Dumitraş T. FeatureSmith: Automatically Engineering Features for Malware Detection by Mining the Security Literature[C]. *The 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016: 767-778.

[19] Xu K, Li Y J, Deng R H, et al. DeepRefiner: Multi-Layer Android Malware Detection System Applying Deep Neural Networks[C]. *2018 IEEE European Symposium on Security and Privacy*, 2018: 473-487.

[20] Zhang M, Duan Y, Yin H, et al. Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs[C]. *The 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014: 1105-1116.

[21] Mariconti E, Onwuzurike L, Andriotis P, et al. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models [J]. *Network and Distributed System Security Symposium*, 2016, 7(1): 1-8.

[22] Hou S F, Saas A, Chen L F, et al. Deep4MalDroid: A Deep Learning Framework for Android Malware Detection Based on Linux Kernel System Call Graphs[C]. *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops*, 2016: 104-111.

[23] Karbab E B, Debbabi M, Derhab A, et al. Android Malware Detection Using Deep Learning on API Method Sequences[EB/OL]. 2017.

[24] McLaughlin N, Martinez del Rincon J, Kang B, et al. Deep Android Malware Detection[C]. *The Seventh ACM on Conference on Data and Application Security and Privacy*, 2017: 301-308.

[25] Hasegawa C, Iyatomi H. One-Dimensional Convolutional Neural Networks for Android Malware Detection[C]. *2018 IEEE 14th* International Colloquium on Signal Processing & Its Applications, 2018: 99-102.

[26] Huang T H D, Kao H Y. R2-D2: ColoR-Inspired Convolutional NeuRal Network (CNN)-Based AndroiD Malware Detections[C]. *2018 IEEE International Conference on Big Data*, 2018: 2633-2642.

[27] Chen S, Xue M H, Fan L L, et al. Automated Poisoning Attacks and Defenses in Malware Detection Systems: An Adversarial Machine Learning Approach[J]. *Computers & Security*, 2018, 73: 326-344.

[28] Papernot N, McDaniel P, Jha S, et al. The Limitations of Deep Learning in Adversarial Settings[C]. *2016 IEEE European Symposium on Security and Privacy*, 2016: 372-387.

[29] Grosse K, Papernot N, Manoharan P, et al. Adversarial Perturbations Against Deep Neural Networks for Malware Classification[EB/OL]. 2016

[30] Al-Dujaili A, Huang A, Hemberg E, et al. Adversarial Deep Learning for Robust Detection of Binary Encoded Malware[C]. *2018 IEEE Security and Privacy Workshops*, 2018: 76-82.

[31] Chen X, Li C R, Wang D R, et al. Android HIV: A Study of Repackaging Malware for Evading Machine-Learning Detection[J]. *IEEE Transactions on Information Forensics and Security*, 2020, 15: 987-1001.

[32] Carlini N, Wagner D. Towards Evaluating the Robustness of Neural Networks[C]. *2017 IEEE Symposium on Security and Privacy,* 2017: 39-57.

[33] Kolosnjaji B, Demontis A, Biggio B, et al. Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables[C]. *2018 26th European Signal Processing Conference*, 2018: 533-537.

[34] Raff E, Barker J, Sylvester J, et al., Malware Detection by Eating a Whole EXE[C]. *National Conference on Artificial Intelligence*, 2017:268-276.

[35] Demontis A, Melis M, Biggio B, et al. Yes, Machine Learning can be more Secure! a Case Study on Android Malware Detection[J]. *IEEE Transactions on Dependable and Secure Computing*, 2019, 16(4): 711-724.

[36] Sankaranarayanan S, Jain A, Chellappa R, et al. Regularizing Deep Networks Using Efficient Layerwise Adversarial Training [C], *National Conference on Artificial Intelligence,* 2018:4008-4015.

[37] Chen L W, Ye Y F, Bourlai T. Adversarial Machine Learning in Malware Detection: Arms Race between Evasion Attack and Defense[C]. *2017 European Intelligence and Security Informatics Conference*, 2017: 99-106.

[38] W. Hu and Y. Tan, Black-Box Attacks Against RNN Based Malware Detection Algorithms[C]. *National Conference on Artificial Intelligence*, 2017:245-251.

[39] 如何使用 FRIDA 搞定 Android 加壳应用, 安全客,“https://www.anquanke.com/post/id/163390”, 2018.

[40] Hu W W, Tan Y. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN[EB/OL]. 2017

[41] Chen L W, Hou S F, Ye Y F. SecureDroid: Enhancing Security of Machine Learning-Based Detection Against Adversarial Android Malware Attacks[C]. *The 33rd Annual Computer Security Applications Conference*, 2017: 362-372.

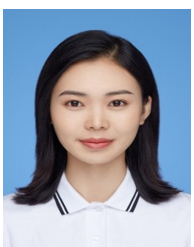
[42] Goodfellow I J, Shlens J, Szegedy C. Explaining and harnessing adversarial examples[J]. *International conference on learning representations*, 2014, 4(11): 287-293.

[43] Yang W, Kong D G, Xie T, et al. Malware Detection in Adversarial Settings: Exploiting Feature Evolutions and Confusions in Android Apps[C]. *The 33rd Annual Computer Security Applications Conference*, 2017: 288-302.

[44] Yuan Z L, Lu Y Q, Xue Y B. Droiddetector: Android Malware Characterization and Detection Using Deep Learning[J]. *Tsinghua Science and Technology*, 2016, 21(1): 114-123.

[45] Wang Q L, Guo W B, Zhang K X, et al. Adversary Resistant Deep Neural Networks with an Application to Malware Detection[C]. *The 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017: 1145-1153.

[46] Moosavi-Dezfooli S M, Fawzi A, Frossard P. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks[C]. *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016: 2574-2582.



**李佳琳** 于 2017 年在青岛大学信息安全专业获得学士学位, 现在中国科学院信息工程研究所第五研究室攻读硕士学位。研究领域为网络与系统安全, 研究兴趣包括: 移动安全、智能设备安全。Email: lijalin@iie.ac.on

**王雅哲** 于 2010 年在中国科学院软件研究所信息安全专业获得博士学位, 现任中国科学院信息工程研究所副研究员。研究领域为网络与系统安全, 研究兴趣包括: 移动安全、智能设备安全。Email: wangyazhe@iie.ac.cn

**罗吕根** 于 2016 年在复旦大学微电子专业获得学士学位, 现在中国科学院信息工程研究所第五研究室攻读硕士学位。研究领域为网络与系统安全, 研究兴趣包括: 移动安全、区块链。Email: luolvgen@iie.ac.cn

**王瑜** 于 2010 年在北京科技大学计算机与科学专业获得硕士学位，现任中国科学院信息工程研究所第五研究室助理研究员。研究领域为网络与系统安全，研究兴趣包括：移动安全、智能设备安全。Email：wangyu@iie.ac.cn