计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(12):29−40 [doi: 10.15888/j.cnki.csa.008800]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于可解释性的 Android 恶意软件检测①

黄海彬[1], 万　良[2], 褚　堃[1]

[1](贵州大学 计算机科学与技术学院, 贵阳 550025)
[2](贵州大学 计算机软件与理论研究所, 贵阳 550025)
通信作者: 万　良, E-mail: lwan@gzu.edu.cn

摘　要: 针对 Android 恶意软件检测, 通常仅有检测结果缺乏对其检测结果的可解释性. 基于此, 从可解释性的角度分析 Android 恶意软件检测, 综合利用多层感知机和注意力机制提出一种可解释性的 Android 恶意软件检测方法 (multilayer perceptron attention-method, MLP_At). 通过提取 Android 恶意软件的应用权限和应用程序接口 (application programming interface, API) 特征来进行数据预处理生成特征信息, 采用多层感知机对特征学习. 最后, 利用 BP 算法对学习到的数据进行分类识别. 在多层感知机中引入注意力机制, 以捕获敏感特征, 根据敏感特征生成描述来解释应用的核心恶意行为. 实验结果表明所提方法能有效检测恶意软件, 与 SVM、RF、XGBoost 相比准确率分别提高了 3.65%、3.70% 和 2.93%, 并能准确地揭示软件的恶意行为. 此外, 该方法还可以解释样本被错误分类的原因.

关键词: Android; 恶意软件检测; 多层感知机; 可解释性; 注意力机制; BP 算法

引用格式: 黄海彬,万良,褚堃.基于可解释性的 Android 恶意软件检测.计算机系统应用,2022,31(12):29–40. http://www.c-s-a.org.cn/1003-3254/8800.html

Interpretability-based Android Malware Detection

HUANG Hai-Bin[1], WAN Liang[2], CHU Kun[1]

[1](College of Computer Science and Technology, Guizhou University, Guiyang 550025, China)
[2](Institute of Computer Software and Theory, Guizhou University, Guiyang 550025, China)

Abstract: As the detection result lacks interpretability, the Android malware detection is analyzed in terms of interpretability. This study proposes an interpretable Android malware detection method (multilayer perceptron attention method, MLP_At) comprehensively using the multilayer perceptron and attention mechanism. By extracting permissions and application programming interface (API) features from Android malware, it performs data preprocessing on the proposed features to generate feature information, and multilayer perceptrons are utilized for learning features. Finally, the learned data is classified by the BP algorithm. The attention mechanism is introduced in the multilayer perceptron to capture sensitive features and generate descriptions based on the sensitive features to explain the core malicious behavior of the application. The experimental results show that the proposed method can effectively detect malware and the accuracy is improved by 3.65%, 3.70%, and 2.93% compared with that of SVM, RF and XGBoost, respectively. The method can accurately reveal the malicious behavior of the software and can also explain the reasons why samples are misclassified.

Key words: Android; malware detection; multilayer perceptron; interpretability; attention mechanism; BP algorithm

① 基金项目: 国家自然科学基金 (62062020)
收稿时间: 2022-03-06; 修改时间: 2022-04-02; 采用时间: 2022-04-13; csa 在线出版时间: 2022-07-15

移动设备用户正在急剧增加, 针对 Android 系统的恶意软件随之滋生[1,2], 越来越多的应用程序会将用户银行交易信息、社交私人信息存储在移动设备上[3,4], 用户的信息隐私和资产安全都存在着严重的威胁. 近年来提出许多恶意软件分析方法, 基于特征码[5–7] 的方法虽能有效检测, 但是需要频繁的更换特征码数据库, 并且对于 0-Day 恶意代码是无效的. 基于行为的方法[8–11] 也会依赖预定义的恶意行为, 受限制于现有的恶意样本的分析. 基于数据流的方法[12–14] 则是注重于数据泄露相关的恶意行为, 缺乏对其他恶意行为的检测. 虽然此类方法对恶意软件检测均有效. 然而, 所构建的分类器模型就像一个黑盒子, 不能解释为什么给定的样本被识别为恶意样本. 由于训练数据的大小和分析模型的复杂性太大, 分析师很难手动分析黑箱模型并推断. 此外, 现有的方法表明, 学习到的模型很容易被对抗性样本攻击, 这甚至会增加恶意软件分析的难度. 因此, 分析人员无法确定是否可以相信这些决策. 为了对检测结果提供可解释性, 本文提出 MLP_At 检测算法, 将注意力机制应用于恶意软件检测中, 确定与其检测结果显著性相关的特征, 进一步分析恶意软件行为. 实验验证, 该方法能对恶意软件检测具有良好的检测性能, 并具有可靠的可解释性.

1 相关工作

为有效对 Android 恶意软件检测, 保障用户的信息安全. 相关学者将机器学习算法应用于 Android 恶意软件检测方法. 如 KNN[15], SVM[16], RF[17] 和 XGBoost[18] 来对恶意软件进行分类, 在此类方法中, Android 的权限和应用程序接口 (API) 调用是常用的特征类型[19,20], 并取得较高准确率的检测. Taheri 等[21] 人报告一项研究, 该研究基于应用程序的权限、目的和 API 调用等属性的提取, 以及使用 K 近邻 (KNN) 派生算法对这些提取进行分类. 这是一种基于相似性的方法. 深度学习作为机器学习的重要分支, 其网络对复杂的数据能较好处理. 业内学者也将 CNN 和 RNN 等深度神经网络模型应用于检测 Android 恶意软件[22–24], 并且取得良好的性能. Naeem 等[24] 设计了检测工业物联网恶意软件攻击的架构 (MD-IIOT). 深入分析恶意软件, 构建基于 RGB 图像可视化和 CNN 的方法.

但是, 这些方法通常仅呈现检测结果 (良性、恶性), 并没有完全解决恶意软件检测的问题, 缺少对其行为的解释分析. 分析大规模的 Android 恶意软件是一个极其困难和耗费时间的任务. 此外, 机器学习的鲁棒性面临着对抗样本的安全威胁[24–27]. 因此, 解释机器学习模型已经学到的内容, 以及该模型如何进行预测也是极其重要的. 为了让用户信任基于机器学习的方法, 近年开发了各种解释方法通过提供重要特征来解释预测. 这些解释方法中最突出的包括局部的、模型不可知的方法[25–29], 它们专注于解释给定黑盒分类器的个体预测. 这些解释方法的基本思想是使用线性模型来近似局部决策边界, 以推断当前输入实例的重要特征. 分析师可以利用这种解释特征来分析结果, 调试预先训练的模型[30], 并检测对抗性输入[31]. 可解释的方法似乎是打开黑盒模型和推断决策机制的关键. 但是, 在现有的解释方法发现所提供的解释结果不能在总体上达成共识为了提供恶意软件检测的可解释性, Adebayo 等[32] 提出一种基于随机化测试的可行方法来评估解释方法的充分性. 他们的结果发现, 一些广泛部署的显着性方法独立于模型训练的数据和模型参数, 主要侧重于评估为图像分类领域的 CNN 设计的解释方法. Yeh 等[33] 提出了两个客观评估指标用于机器学习解释, 但是需要人类标记的解释结果的真实性. Camburu 等[34] 在特征选择的角度下, 介绍了一种对事后解释方法的现成评估测试. Arp 等[16] 在检测 Android 恶意软件中解释恶意行为的方法, 但该方法仅在训练时对其恶意行为进行定位. 由于注意力机制机器翻译和 CV 中均有良好性能, 并在可解释性中取得很大成功. 基于此, 本文循着此路线研究并提出一种基于注意力机制 Android 恶意软件检测方法. 该方法能确定与预测结果显著性相关的特征, 解释预测结果是如何做出的, 并生成软件行为的描述.

2 MLP_At 检测方法

为有效检测 Android 恶意软件, 并提供检测结果的可解释性, 本文提出 MLP_At 的检测方法. 该方法如图 1 所示, MLP_At 包含 3 个阶段, 分别为数据预处理、分类器和解释器. 实验中数据预处理是将数据去除冗余、清理缺失值和处理异常值. 分类器则是将处理后的数据作为输入, 目的是准确预测应用程序是否为恶意软件, 且捕获到与预测结果最相关的输入特征. 解释器就是生成行为描述, 根据捕获到特征信息, 解释该恶意软件的行为, 以及可能会造成的损害.

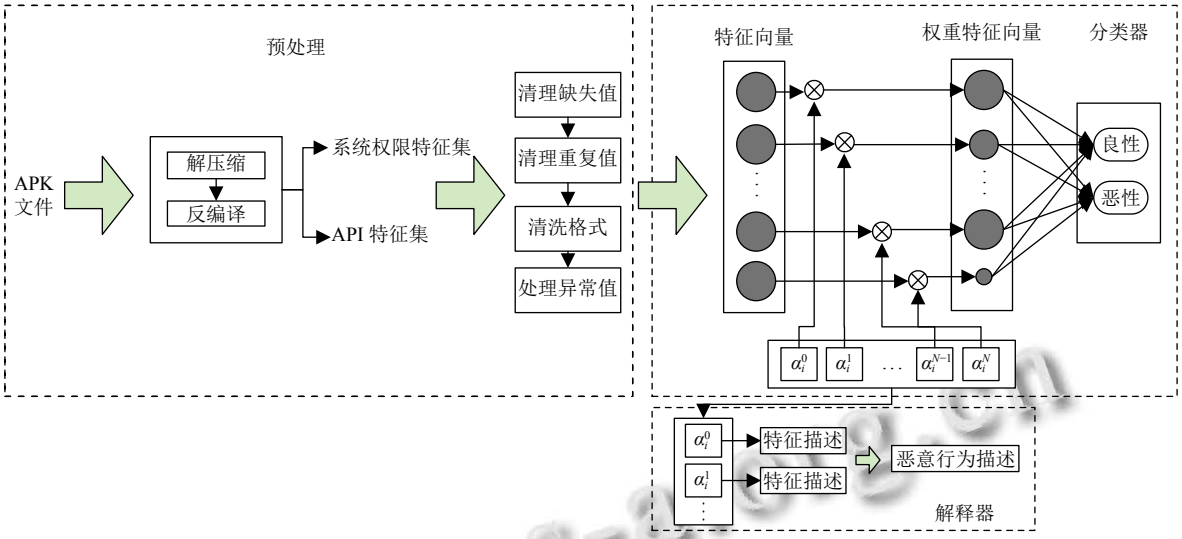


图 1 MLP_At 原理框架图

2.1 数据预处理

2.1.1 特征提取

Android 应用程序的行为需要授予必要的权限并调用相应的 API. 权限和 API 调用是 Android 恶意软件检测和分析[35] 最重要和最常用的特征. 因此, 本文亦使用 API 调用和权限作为特征来训练检测模型.

为了提取 API 调用和权限, 本文利用 Androguard[36] 从 APK 文件中提取 API 调用和权限, 将其用来构造特征向量. 下面的方法表示一组样本$\{\{(x_i, y_j)\} \mid x_i \in X, y_i \in y, 1 \leqslant i \leqslant M\}$, 其中, X是x_i的集合, Y是y_i的集合, $x_i = \left(x_i^{(1)}, x_i^{(2)}, x_i^{(3)}, \cdots, x_i^{(N)}\right)$是第 i 个样本的特征向量, N是特征的总数, $y_i \in \{0,1\}$是第 i 个样本的标签 (0 表示良性软件, 1 表示恶性软件), M是样本的总量, $x_i^{(j)}$是第 i 个样本的第 j 个特征, 如果第 j 个特征存在则$x_i^{(j)} = 1$, 反之$x_i^{(n)} = 0$.

2.1.2 剪枝算法

在 Android 系统中, 有数百个权限, API 的数量也有上千个. 但并不是所有的特征都有助于区分恶意软件. Li 等[37] 利用 3 个级别的修剪, 发现只有 22 个权限对于检测恶意软件是重要的. 因此, 本文需要使用剪枝来保存那些可以用来有效地识别恶意软件的特征. 从训练样本集中提取的原始特征中, 使用文献 [38] 中的手动统计剪枝方法, 选择 154 个特征进行研究. 所选特征对恶意软件分类具有较高的识别率, 有利于提高分类的准确性和可解释性. 同时, 由于 API 调用和权限具有更多的语义, 能说明它们在应用程序中的角色行为. 将它们作为特征使用可以帮助进一步解释本文的模型.

特征剪枝操作——卡方检验 (Chi-square test) 是计算出真实值和理论值的偏离程度, 提出假设, 判断在变量之间是否存在关系, 故本文使用卡方检验对权限、API 特征进行筛选. 卡方检测是将实际值和理论值的偏离程度作为依据, 根据偏离大小来判断之前的假设是否成立. 偏离小, 则认为这是一种正常反应, 反之, 认为这不是自然偏差, 二者存在着某些关系. 卡方检验使用的差值衡量式 (1) 如下:

$$x^2 = \sum_{i=1}^{n} \frac{(x_i - E)^2}{E} \tag{1}$$

其中, x_i 是实验值, i 为第 i 次实验, E 为理论值. 要将卡方检验作为剪枝算法, 则需假设某条特征出现与否与软件是否良性无关, 二者相互独立. 将样本总数设为 N 值, 其中 A、B、C、D 是从总样本 N 中取出的实际值. A、B、C、D 参数如表 1 所示, 分别代表良性或恶意软件是否包含某条特征的样本数量.

表 1 包含某条特征的样本实际值表

是否包含某条特征	恶意样本数量	良性样本数量
包含	A	B
不包含	C	D

根据假设条件, 是否包含某条特征与是否是恶意软件无关, 因此, 软件中包含某条特征的概率 P 如式 (2) 所示:

$$P = \frac{A + B}{N} \tag{2}$$

进一步可得出软件中有多少恶意软件包含此条特征. 其数量见式 (3):

$$E=\frac{A+B}{N}(A+C) \tag{3}$$

由式 (3) 可推导出剩余的理论值, 详见表 2.

表 2 包含某条特征的样本理论值表

是否包含某条特征	恶意样本数量	良性样本数量
包含	$\frac{A+B}{N}(A+C)$	$\frac{A+B}{N}(B+D)$
不包含	$\frac{C+D}{N}(A+C)$	$\frac{C+D}{N}(B+D)$

最后将 N 个样本的实际值和理论值代入式 (1) 中将得到差值衡量x^2, x^2如式 (4) 所示:

$$x^2=\frac{\left(A-\frac{A+B}{N}(A+C)\right)^2}{\frac{A+B}{N}(A+C)}+\frac{\left(B-\frac{A+B}{N}(B+D)\right)^2}{\frac{A+B}{N}(B+D)}+\frac{\left(C-\frac{C+D}{N}(A+C)\right)^2}{\frac{C+D}{N}(A+C)}+\frac{\left(D-\frac{C+D}{N}(B+D)\right)^2}{\frac{C+D}{N}(B+D)} \tag{4}$$

将式 (4) 化简, 得出x^2的最终结果为:

$$x^2=\frac{(AD-BC)^2N}{(A+C)(A+B)(C+D)(B+D)} \tag{5}$$

至此, 可将每个特征的卡方值x^2计算出来, 卡方检验的x^2值越小说明假设的结论成立, 二者相互独立. 但在本次实验中, 需要选择良性或恶意软件是否包含某条软件的相关性. 故此次实验需要选择x^2值比较大的, 当x^2值越大, 说明假设不成立, 二者存在关联性越紧密. 反之越小, 二者相关性越小, 假设成立.

2.2 Andriod 恶意代码检测方法

在本阶段将基于剪枝算法所获取特征向量构建一个 Android 恶意软件检测模型, 用于区分出良性和恶性软件. 如图 1 所示分类器主要由两个部分组成: 多层感知机 (MLP) 和注意力层 (如图 2). 注意力层的目的是学习特征的权重, 可以看作特征与分类结果的相关性.

注意力机制是通过对输入特征和输出特征的匹配程度获取特征的权重, 表述为:

$$e_{ij}=score\left(h_{i-1},h_j\right) \tag{6}$$

e_{ij} 表示捕获编码器隐藏状态h_j和解码器隐藏状态h_{i-1}的相关性, 特征权重α_{ij}表示为:

$$\alpha_{ij}=Softmax\left(e_{ij}\right) \tag{7}$$

其中, α_{ij}表示第 j 个输入对第 t 个输出的权重分数, $\alpha_{ij}\in(0,1)$, α_{ij}越趋近于 1 相关度越高, 反之趋近于 0 相关度越低.

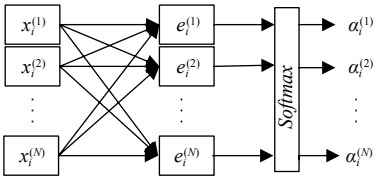


图 2 MLP_At 注意力层

$$e_i^{(j)}=\sum_{k=1}^{N}x_i^{(k)}w_{kj} \tag{8}$$

其中, $e_i^{(j)}$ 表示输入对 j 个位置输出, $e_i^{(j)}$ 是所有输入$x_i^{(k)}$与其当前输出的权重w_{kj}的点乘的方式计算的. 它可以看作在第 j 个位置与输入特征有不同相关性的一组特征组合.

至此, 将网络的输出$e_i^{(j)}$再执行一个 *Softmax* 函数, 获取不同位置的输入特征的权值. $\alpha_1^{(j)}$表示为获取的注意力向量, 其表示在第 i 个样本中第 j 个的特征的权重. 计算公式为:

$$a_i^{(j)}=\frac{\exp(e_i^{(j)})}{\sum_{k=1}^{n}\exp(e_i^{(k)})} \tag{9}$$

通过注意力层生成注意力向量后, 结合 MLP 将加权后的特征向量映射到 MLP 中的特征向量使之成为加权特征向量. 将z_i定义为第 i 个样本的加权特征向量. 则z_i的计算公式为:

$$z_i=a_ix_i^{\mathrm{T}} \tag{10}$$

最后将z_i作为分类函数的输入, 分类结果为:

$$y_i=f(z_i) \tag{11}$$

其中, f为 MLP 的分类函数, 预测输入的特征向量z_i在分类函数中的结果. 注意力机制学习算法如算法 1 所示.

算法 1. 注意力机制学习算法

输入: 剪枝后的 154 维特征向量 *Feature*, 样本标签 Y
输出: 注意力特征

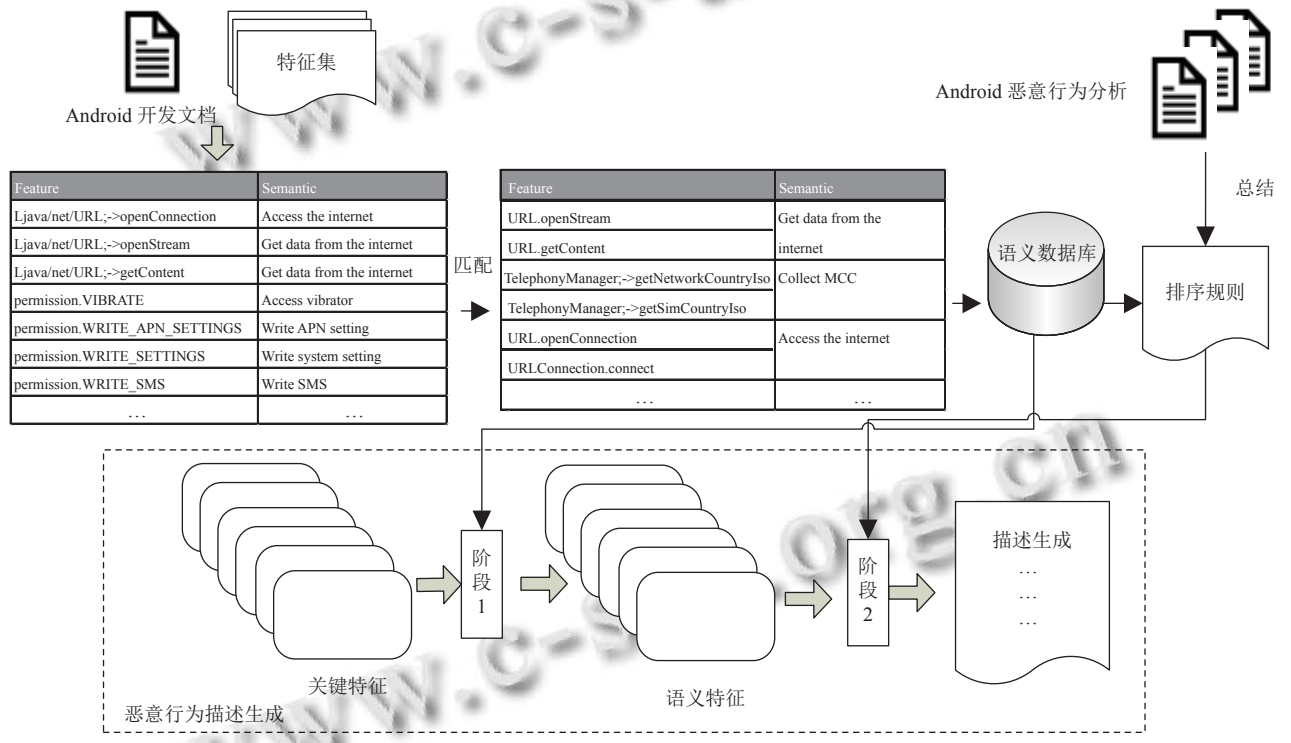
1. 初始化特征 $Feature=\{f_1,f_2,\cdots,f_n\}$, $Y=\{y_1,y_2\}$
2. 初始化隐藏状态H_0、权重系数$\{w_1,w_2,\cdots,w_n\}$偏执值 $b=\{b_1,b_2,\cdots,b_n\}$
3. for i in range(len(Y)):

4. for j in range(n):
5. 计算 f_i与y_i的权重$a_i^{(j)}$
6. 计算加权特征向量z_i
7. end for
8. end for
9. 返回z

当输入训练数据对分类器进行训练时, 注意力层根据对应的特征与分类结果的相关性为其分配不同的权重. 与分类相关度较高的特征被赋予较大的权重, 而对分类影响较小的特征被赋予较小的权重. 当一个样本输入到分类器中, 得到分类结果和一个不同权重的特征列表. 剔除了样本中不存在的特征, 并根据其权重对剩下的特征进行排序, 选出关键特征为 Android 恶意程序的行为做分析.

2.3 生成恶意行为描述

为了生成针对 Android 恶意软件的恶意行为描述, 首先将恶意软件的关键特征与它们对应的语义进行匹配. 本文根据剪枝操作选择 154 维的特征向量作为输入来训练分类器. 依据 Android 开发人员文档[35], 根据每个特性的名称寻找每个特性的语义. Android 开发者文档对每个 API 和权限都有详细的功能描述. 通过截取和泛化关键谓词、对象和补语, 将它们简化并一般化为简单的语义. 然后, 利用该特征和相应的语义建立一个语义数据库 (如图 3 所示). 根据观察, 一些特征具有相同的语义. 此外, 有些特性具有类似的功能, 可以组合使用变成一个语义特征. 因此, 制定如下两条规则.



Feature	Semantic
Ljava/net/URL;->openConnection	Access the internet
Ljava/net/URL;->openStream	Get data from the internet
Ljava/net/URL;->getContent	Get data from the internet
permission.VIBRATE	Access vibrator
permission.WRITE_APN_SETTINGS	Write APN setting
permission.WRITE_SETTINGS	Write system setting
permission.WRITE_SMS	Write SMS
…	…

Feature	Semantic
URL.openStream	Get data from the internet
URL.getContent	
TelephonyManager;->getNetworkCountryIso	Collect MCC
TelephonyManager;->getSimCountryIso	
URL.openConnection	Access the internet
URLConnection.connect	
…	…

图 3 恶意软件描述生成

规则 1: 如果特征属于相同的功能, 则会为它们分配相同的语义.

规则 2: 如果特征具有相似的功能, 那么它们将被分配给相似的语义, 并将这两个相似的语义组合为一个.

这样就可以根据功能描述将特征与语义进行匹配, 从而获得简单有用的特征语义. 然后将语义转换为恶意软件的描述, 以便用户更容易理解. 为了对 Android 恶意软件进行合理的描述, 总结 10 种基本的恶意行为, 并建立了恶意行为与其对应的语义之间的映射关系.

恶意行为描述生成如下, 首先得到一组关键特征 U, 其中$k_i \in U$是第i个关键特征. 然后在图 3 所示的阶段 1 中将k_i转换为s_i, 其中s_i是第 i 个语义. 根据规则 1, 如果关键特征属于相同的功能, 则它们被分配到相同的语义. 因此, S中存在的那些语义不会再次添加到S中. 根据规则 2, 如果关键特征表现出相似的功能, 则将它们相似的语义合并为一个. 因此, 当S中出现与语义 s 相似的语义时, 我们将其与 s 合并, 然后在S中更新. 之后, 在图 3 所示的阶段 2 中将语义一一转换为描

述. 图 3 显示了解释器如何逐步生成恶意软件描述以及如何建立语义数据库和排序规则.

2.4 模型训练

神经网络模型训练分为两个阶段, 第一, 数据前向传播阶段, 第二, 根据损失函数计算, 将误差反传回去, 优化权重, 此为反向传播阶段, 训练过程如图 4 所示.

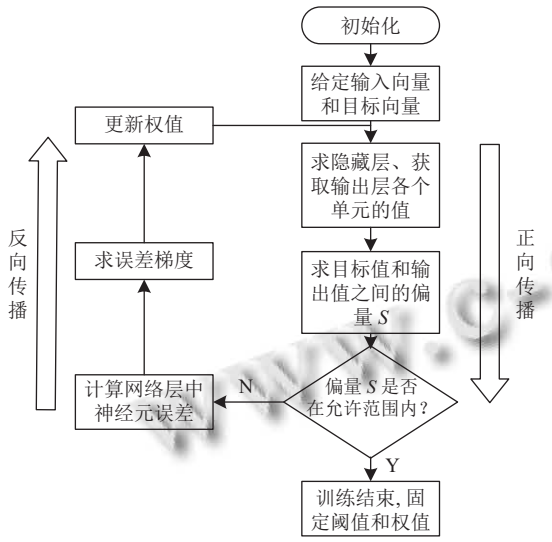


图 4 模型训练流程

Step 1. 模型权重的初始化.

Step 2. 图像数据经过 MLP_At 的向前传递.

Step 3. 根据设定的损失函数, 计算误差.

Step 4. 误差若大于期望值, 则将误差反向传递, 计算各层之间的误差, 根据优化函数更新权重与偏置值, 反之结束训练.

3 实验及结果

3.1 实验环境及样本

实验的硬件环境为 Intel Corei7-10700F CPU 2.90 GHz, 16 GB RAM; 实验平台 Window 10; 主要软件和开发工具包为 Python 3.6, Androguard 3.3.5, TensorFlow 1.14, Keras 2.1. 实验主要涉及 Drebin[16]、FalDroid[39] 两个公开的数据集, Drebin 包含 179 个恶意家族 5 560 个恶意样本; FalDroid 包含 36 个恶意家族 8 407 个恶意样本. 随机选择 4076 个恶意样本用于实验, 在 Google Play 上通过网络爬虫工具爬取了 4 038 个软件.

3.2 评价指标

为了客观地评价本文检测方法, 本文采用以下几个评价指标, 从多个角度对模型进行评估, 准确率 (accuracy rate, ACC)、精确率 (precision rate, PR)、召回率 (recall, RE) 和 $F1$ 值 ($F1$ score, $F1$) 作为实验评估指标[40].

3.3 损失函数

本文使用交叉熵“categorical_crossentropy”作为模型的损失函数; 数学公式如式 (12) 所示:

$$loss(y,\hat{y}) = -\sum_{i=1}^{C} y_i \cdot \log(\hat{y}_i) \tag{12}$$

其中, y 代表着分类的真实标签, $\hat{y}$ 代表着模型预测标签, C 为分类数.

3.4 实验参数

首先将数据集中的 50% 数据作为训练集, 30% 数据作为验证集, 训练集 20% 的数据作为测试集. 为了为 MLP_At 选择最佳的超参数, 从数据集中提取包含 94 个 API 调用和 60 个权限的 154 维特征向量对 MLP_At 进行训练, 并利用测试集对 MLP_At 的检测精度进行评估. 然后对 MLP_At 的不同超参数进行测试, 最终确定获得最佳检测性能的超参数. 为了获得更好的检测性能, 首先对 MLP_At 的最佳超参数 (即学习率、优化器、激活函数、epoch 和 batch_size) 进行设置. 将学习率设置为 0.000 1、0.001、0. 01 和 0.1, 实验表明检测性能略有不同, 但是 0.001 的学习率对于模型最优, 结果如图 5 所示. 因此, 我们在实验中选择了 0.001 作为学习率.

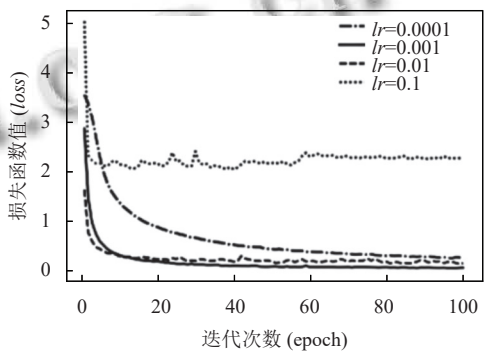


图 5 学习率与模型检测损失函数值变化的关系

图 6 为数据集在不同优化器和激活函数的训练结果. 根据实验结果“Adam+Softmax”整体表现最佳. 故将其作为此次实验的优化器和激活函数.

进一步研究 epoch 和 batch_size 的影响. 根据实验结果, 发现 batch_size 较小时, 训练准确率较低, 且损失函数曲线波动较大, 反之, batch_size 较大时, 损失函数曲线波动较小, 但是准确率也降低. 如图 7 所示, 100

个 epoch 和 100 个 batch_size 的配置效果最好. 故选其作为训练的 epoch 和 batch_size.

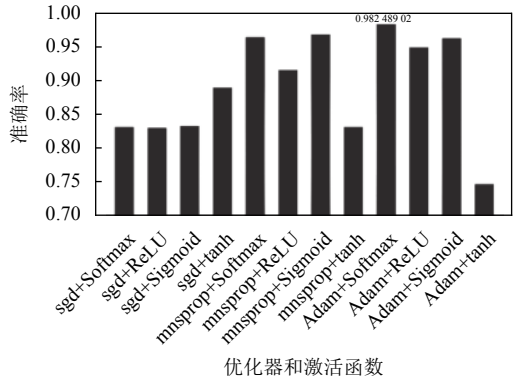


图 6 不同优化器和激活函数的训练准确率

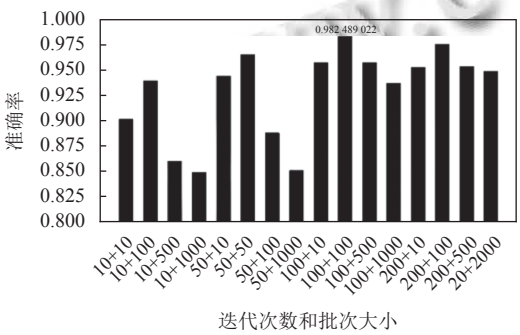


图 7 不同迭代次数和批次大小的训练准确率

为了验证模型的有效性, 对构建模型的进行多次实验, 将实验结果均值作为最终结果. 模型的损失函数和准确率曲线如图 8 所示, 准确率曲线随着 epoch 增加呈现稳定上升趋势, 同时损失函数曲线随着 epoch 增加呈现稳定下降趋势. 准确率曲线和损失函数曲线最后平稳趋于一个稳定的值, 可以得出模型具良好的检测效能.

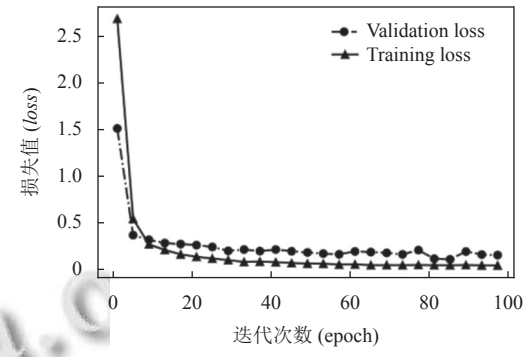
3.5 实验结果

3.5.1 特征选择实验结果

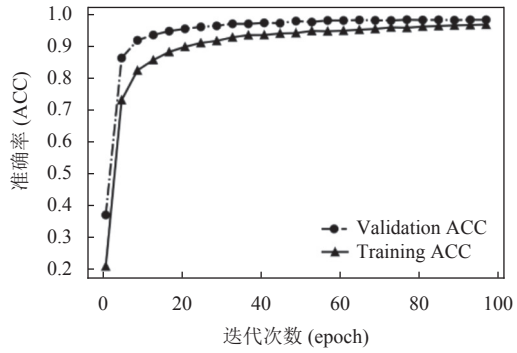
以 FalDroid 数据集为例, 使用卡方剪枝算法对原始特征剪枝, 最终选择 154 维的特征子集, 如表 4 所示.

完成特征选择后, 对其检测效能进行验证, 本文使用机器学习算法 KNN[15], SVM[16], RF[17] 和 XGBoost[18] 对实验所收集的数据集进行训练和测试. 实验结果如表 5 所示.

与现有的特征选择算法对对比, 本文在训练集下, 分别利用文献 [37–40] 中提到的方差选择、递归特征消除、遗传算法和关联规则特征选择方法和本文方法对原始特征集进行特征选择进行对比. 对比结果如表 6 所示.



(a) 损失函数



(b) 准确率

图 8 损失函数和准确率曲线

表 4 FalDroid 最优特征子集

类型	数目	特征 (FalDroid数据集)
系统权限	60	android.permission.ACCESS_COARSE_LOCATION android.permission.ACCESS_FINE_LOCATION android.permission.ACCESS_LOCATION_EXTRA_COMMA …
API	94	Landroid/app/ActivityManager;->getRunningAppProcesses Landroid/app/ActivityManager;->isLowRamDevice Landroid/app/ActivityManager;->killBackgroundProcesses …

表 5 特征选择实验结果

算法	ACC	PR	RE	*F*1
KNN	0.988 53	0.978 35	0.983 41	0.983 35
SVM	0.955 72	0.890 68	0.922 06	0.938 44
RF	0.953 25	0.993 20	0.972 82	0.978 35
XGBoost	0.962 94	0.963 00	0.962 96	0.963 01
本文算法	**0.992 21**	**0.982 47**	**0.987 31**	**0.991 68**

表 6 特征选择方法对比实验结果

方法	特征数量	ACC	PR	RE	*F*1
文献[37]	394	0.964	0.964	0.969	0.967
文献[38]	400	0.97	0.968	0.973	0.971
文献[40]	22	0.961	0.97	0.974	0.962
文献[39]	314	0.966	0.964	0.964	0.968
本文方法	154	**0.991**	**0.992**	**0.982**	**0.987**

3.5.2 生成行为描述

从常见的 Android 恶意软件家族中选取样本, 对 MLP_At 方法进行验证, 如表 7、表 8 所示. 为了验证模型的有效性, 本文从恶意家族中选择两个家族进行说明, 可解释性结果如表 8. 为了说明实验结果如何解释为什么一个应用程序被归类为恶意软件, 本文以 Adrd 和 Geinimi 家族为例. Adrd 恶意软件家族是一个木马在窃取信息从 Android 设备. 如表 7, MLP_At 输出 5 个关键特性 (即 READ_PHONE_STATE; RECEIVE_BOOT_COMPLETED; LocationManager.requestLocation-Updates; HttpURLConnection.getRespo-nseCode; TelephonyManager.getSubscriberId), 并生成相应的语义 (1. Collect IMEI/IMSI/location; 2. Activate by BOOT; 3.Collect phone status; 4.Query URL data (colle-ct SMS); 5.Write external storage) 和恶意行为描述 (Start with system startup, collect info on the device and send it to remote server over the Internet). MLP_At 生成的恶意行为描述可以清楚地解释 Adrd 样本被归类为恶意软件. 此外, 专家分析显示 Adrd 的恶意行为是: 当移动设备启动时, 它会重新执行自己的行为, 窃取信息并发送到远程服务器. 这个结果与本文所生成描述一致, 可以认为本文的恶意行为描述生成具有有效性.

Geinimi 家族在系统启动时启动, 向收费号码发送 SMS 消息, 在设备上收集信息并通过 Internet 将其发送到远程服务器. 从表 8 可以看出, MLP_At 输出下面几个个关键特性 (URL.openConnection; SEND_SMS; READ_PHONE_STATE; RECEIVE_BOOT_COMPLE-TED). Geinimi 样本, 并生成相应的语义 (1. Access the Internet; 2. Send SMS message; 3. Collect IMEI; 4. Activate by BOOT) 和恶意行为描述 (Start with system startup, send SMS message to premium-rate numbers, collect info on the deviceand Send it to remote server over the Internet). MLP_At 生成的恶意行为描述也与表 7 中 Geinimi 家族的的恶意行为一致. 这也解释了 Geinimi 为恶性软件的原因.

两个的应用程序被错误分类, 试图根据 MLP_At 的可解释结果分析它们为什么被错误分类. 被误分类的两个应用程序是 Airpush 和 AntiVirus. 可以从表 8 中看到, AntiVirus 应用程序确实使用了一些可疑的权限和 API, 导致它被归类为恶意软件. 实际上, 它只是使用敏感 API 和权限的内置系统应用程序. 在这种情况下, MLP_At 很难正确区分恶意软件, 因为内置的系统应用程序具有与恶意软件相同的特性和行为. 对于被错误分类的 Airpush 样本, MLP_At 没有为它所有输出任何关键特性, 这意味着 MLP_At 没有识别这些样本的任何关键特性来将它们分类为恶意软件, 导致恶意样本被错误分类为良性. 进一步手动分析这个样本, 发现 Airpush 中恶意的 Apk 文件安装在虚拟环境下呈现是广告软件, 导致 App 进行恶意行为不足以分为恶意软件, 从而被识别为良性.

在恶意样本中重新选择 60 个恶意样本, 这些样本没有训练和测试过. 分别从 Adrd、GingerMaster、Plankton 家族中各选 20 个进行验证本文的所提方法 MLP_At 的有效性. 此外还将 MLP_At 与 Drebin 进行对比, 可解释结果如表 9 所示.

表 9 显示 Drebin、本文方法和真实分析的特征语义描述, Drebin 生成一些关键特性, 在 GingerMaster 中提取的关键特征 NotificationManager.cancel 不匹配任何语义, 在 FakeInstaller 中 Intent.action.MAIN 同样没有匹配语义. 而 MLP_At 生成语义描述贴切真实软件分析结果.

4 结论

本文提出一种名为 MLP_At 的新方法来解释 Android 应用程序的恶意行为. MLP_At 在 Android 恶意软件检测中实现了较高的准确性, 并利用分类阶段确定的关键特性, 输出合理的自然语言描述来解释恶意行为. 在解释一个应用程序被归类为恶意软件方面存在许多公开的挑战. 其中之一是复杂的场景. 一些危险的 API 和权限可能被用于良性应用程序中以达到好的目的, 比如内部系统应用程序. 基于特征的预测和解释方法是一个巨大的挑战. 不同场景中使用的特性可能有不同的用途. 另一个挑战是当前恶意软件的恶意行为变得更加复杂. 恶意软件可以通过代码混淆隐藏其行为, 并在安装后下载有效负载以逃避恶意软件检测. 恶意软件作者修改其 DEX 文件的一个模糊版本的重新编译

代码, 并使用反逆向技术, 以避免动态分析, 并防止恶意软件运行在模拟器. 即使使用人工分析, 也很难完全理解某些复杂系统的所有恶意行为.

未来将研究图卷积神经网络, 在提取 Android 函数调用图的前提下, 实现伪动态的分析, 做出更加全面的可解释的模型.

表 7 部分恶意样本可解释性结果

Sample	Key Features	Semantic matching	Description generated
Adrd	READ_PHONE_STATE RECEIVE_BOOT_COMPLETED LocationManager.requestLocationUpdates HttpURLConnection.getResponseCode TelephonyManager.getSubscriberId	(1) Collect IMEI/IMSI/location (2) Activate by BOOT (3) Collect phone status (4) Query URL data (collect SMS) (5) Write external storage	Start with system startup, collect info on the device and send it to remote server over the Internet
DroidDream	URL.openConnection READ_PHONE_STATE RECEIVE_BOOT_COMPLETED HttpURLConnection.getResponseCode READ_SMS WRITE_SMS	(1) Access the Internet (2) Collect IMEI/SMS/location (3) Activate by BOOT	Start with system startup, collect info on the device and send it to remote server over the Internet
FakeDoc	URL.openConnection SEND_SMS READ_PHONE_STATE RECEIVE_BOOT_COMPLETED HttpURLConnection.getResponseCode WAKE_LOCK	(1) Access the Internet (2) Send SMS message (3) Collect IMEI (4) Activate by BOOT (5) Unlock phone	Start with system startup, send SMS message to premium-rate numbers, collect info on the device and send it to remote server over the Internet, keep running in the background
Geinimi	URL.openConnection SEND_SMS READ_PHONE_STATE RECEIVE_BOOT_COMPLETED	(1) Access the Internet (2) Send SMS message (3) Collect IMEI (4) Activate by BOOT	Start with system startup, send SMS message to premium-rate numbers, collect info on the deviceand Send it to remote server over the Internet
GinMaster	URL.openConnection READ_PHONE_STATE RECEIVE_BOOT_COMPLETED HttpURLConnection.getResponseCode telephony/TelephonyManager.getSubscriberId	(1) Access the Internet (2) Collect IMEI (3) Activate by BOOT	Start with system startup, collect info on the deviceand send it to remote server over the Internet
Kmin	URL.openConnection SEND_SMS READ_PHONE_STATE RECEIVE_SMS READ_SMS WRITE_SMS	(1) Access the Internet (2) Send SMS message (3) Collect IMEI/SMS	Send SMS message to premium-rate numbers, collect info on the device, and send it to a remote server over the Internet
Plankton	URL.openConnection READ_PHONE_STATE HttpURLConnection.getResponseCode LocationManager.requestLocationUpdates WAKE_LOCK ContentResolver.query	(1) Access the Internet (2) Collect IMEI/SMS/location (3) Unlock phone	Collect info on the device, and send it to a remote server over the Internet, keep running in the background
SMSreg	URL.openConnection READ_PHONE_STATE HttpURLConnection.getResponseCode LocationManager.requestLocationUpdates WAKE_LOCK NotificationManager.notify	(1) Access the Internet (2) Collect IMEI/SMS/location (3) Unlock phone (4) Notify the info	Collect info on the device, and send it to a remote server over the Internet, Send a notification as a system
SendPay	URL.openConnection READ_PHONE_STATE URLConnection.connect HttpURLConnection.getResponseCode	(1) Access the Internet (2) Collect IMEI	Collect info on the device, and send it to a remote server over the Internet
Iconosys	SEND_SMS READ_PHONE_STATE READ_SMS WRITE_SMS WAKE_LOCK	(1) Send SMS message (2) Collect SMS (3) Unlock phone	Send SMS message to premium-rate numbers, collect info on the device, and send it to a remote server over the Internet, keep running in the background

表 8 被错误分类的样本

Sample	Key features	Semantic matching
Airpush (Malicious)	Landroid/content/ContentResolver;->query	(1) Query URL data (collect SMS)
	android.permission.INTERNET	(2) Access the Internet
	Landroid/app/NotificationManager;->cancel	(3) Cancel notification
	Landroid/telephony/TelephonyManager;->getNetworkOperatorName	(4) Collect network operator name
AntiVirus (Benign)	android.permission.INTERNET	(1) Access the Internet
	Landroid/app/NotificationManager;->cancel	(2) Cancel notification
	Landroid/content/ContentResolver;->query	(3) Query URL data (collect SMS)
	android.permission.SEND_SMS	(4) Send SMS message
	Landroid/telephony/TelephonyManager;->getNetworkOperatorName	(5) Collect network operator name
	Landroid/app/ActivityManager;->getLargeMemoryClass	(6) Get per-application memory

表 9 MLP_At 有效性分析

Sample		Drebin	MLP_At (本文方法)	真实分析
Adrd	Key feature	INSTALL_PACKAGES URL.getContent TelephonyManager.getDeviceId URL.openConnection NotificationManager.cancel	READ_PHONE_STATE RECEIVE_BOOT_COMPLETED LocationManager.requestLocationUpdates HttpURLConnection.getResponseCode TelephonyManager.getSubscriberId	(1) Activate when the mobile device is booted up (2) Access the Internet and download components (3) Steal some info and send to remote server
	Semantic matching	(1) Install packege (2) Get data from the Internet (3) Collect device ID (4) Access the Internet (5) Cancel notification	(1) Collect IMEI/IMSI/location (2) Activate by BOOT (3) Collect phone status (4) Query URL data (collect SMS) (5) Write external storage	
GingerMaster	Key feature	RECEIVE_BOOT_COMPLETED TelephonyManager.getDeviceId TelephonyManager.getSimSerialNumber URL.openConnection NotificationManager.cancel	URL.openConnection READ_PHONE_STATE RECEIVE_BOOT_COMPLETED HttpURLConnection.getResponseCode telephony/TelephonyManager.getSubscriberId	(1) Steal info from the device (2) Send info to remote server (3) The malicious service is triggered when the device finishes a boot
	Semantic matching	(1) Activited by BOOT (2) Collect device ID(IMEI) (3) Collect ICCID (4) Access the Internet (5) Cancel notification	(1) Access the Internet (2) Collect IMEI (3) Activate by BOOT	
FakeInstaller	Key feature	Intent.action.MAIN Send SMS SEND_ SMS android. hardware . telephony send Text Message . READ_ PHONE_ STATE	SEND_SMS READ_PHONE_STATE RECEIVE_SMS READ_SMS TelephonyManager.getNetworkOperator WAKE_LOCK	(1) Send the premium SMS (2) Receive commands from a remote server
	Semantic matching	(1) Send SMS to toll number (2) Get phone information (3) Send text messages to toll numbers and delete text messages without the user's knowledge (4) collect info on the device	(1) Send SMS messages (2) Collect IMEI/SMS (3) Unlock phone	

参考文献

1 中国互联网络信息中心. 第 47 次中国互联网络发展状况统计报告. 北京: 中国互联网络信息中心, 2021.

2 IDC. Smartphone market share. https://www.idc.com/promo/smartphone-market-share. (2021-10-28).

3 Chen S, Fan LL, Meng GZ, *et al*. An empirical assessment of security risks of global Android banking apps. Proceedings of the 2020 IEEE/ACM 42nd International Conference on Software Engineering. Seoul: IEEE, 2020. 1310–1322.

4 Chen S, Su T, Fan LL, *et al*. Are mobile banking apps

secure? What can be improved? Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Lake Buena Vista: ACM, 2018. 797–802.

5 Schlegel R, Zhang KH, Zhou XY, *et al*. Soundcomber: A stealthy and context-aware sound trojan for smartphones. Proceedings of Network and Distributed System Security Symposium. San Diego: The Internet Society, 2011. 17–33.

6 Zhou W, Zhou YJ, Grace M, *et al*. Fast, scalable detection of "piggybacked" mobile applications. Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy. San Antonio: ACM, 2013. 185–196.

7 Zhou YJ, Wang Z, Zhou W, *et al*. Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets. Proceedings of the 19th Annual Network and Distributed System Security Symposium. San Diego: The Internet Society, 2012. 50–52.

8 Graziano M, Canali D, Bilge L, *et al*. Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence. Proceedings of the 24th USENIX Security Symposium. Washington: USENIX Association, 2015. 1057–1072.

9 Tam K, Khan SJ, Fattoriy A, *et al*. CopperDroid: Automatic reconstruction of Android malware behaviors. Proceedings of the 22nd Annual Network and Distributed System Security Symposium. San Diego: The Internet Society, 2015. 1–15

10 Wu C, Zhou YJ, Patel K, *et al*. AirBag: Boosting smartphone resistance to malware infection. Proceedings of the 21st Annual Network and Distributed System Security Symposium. San Diego: The Internet Society, 2014.

11 Yan LK, Yin H. DroidScope: Seamlessly reconstructing the OS and dalvik semantic views for dynamic Android malware analysis. Proceedings of the 21th USENIX Security Symposium. Bellevue: USENIX Association, 2012. 569–584.

12 Arzt S, Rasthofer S, Fritz C, *et al*. FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. Edinburgh: ACM, 2014. 259–269.

13 Gordon MI, Kim D, Perkins J, *et al*. Information-flow analysis of Android applications in droidsafe. Proceedings of the 22nd Annual Network and Distributed System Security Symposium. San Diego: The Internet Society, 2015. 110.

14 Li L, Bartel A, Bissyandé TF, *et al*. IccTA: Detecting inter-component privacy leaks in Android apps. Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. Florence: IEEE, 2015. 280–291.

15 Aafer Y, Du WL, Yin H. Droidapiminer: Mining api-level features for robust malware detection in Android. Proceedings of the 9th International Conference on Security and Privacy in Communication Systems. Sydney: Springer, 2013. 86–103.

16 Arp D, Spreitzenbarth M, Hubner M, *et al*. Drebin: Effective and explainable detection of Android malware in your pocket. Proceedings of the 21st Annual Network and Distributed System Security Symposium. San Diego: The Internet Society, 2014. 23–26.

17 Rastogi V, Chen Y, Jiang XX. Droidchameleon: Evaluating Android anti-malware against transformation attacks. Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security. Hangzhou: ACM, 2013. 329–334.

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

19 Chen S, Xue MH, Fan LL, *et al*. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. Computers & Security, 2018, 73: 326–344.

20 Chen S, Xue MH, Tang ZS, *et al*. StormDroid: A streaminglized machine learning-based system for detecting Android malware. Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. Xi'an: ACM, 2016. 377–388.

21 Taheri R, Ghahramani M, Javidan R, *et al*. Similarity-based Android malware detection using Hamming distance of static binary features. Future Generation Computer Systems, 2020, 105: 230–247. [doi: 10.1016/j.future.2019.11.034]

22 Feng RT, Chen S, Xie GZ, *et al*. A performance-sensitive malware detection system using deep learning on mobile devices. IEEE Transactions on Information Forensics and Security, 2020, 16: 1563–1578.

23 Feng RT, Lim JQ, Chen S, *et al*. SeqMobile: An efficient sequence-based malware detection system using RNN on mobile devices. Proceedings of the 2020 25th International Conference on Engineering of Complex Computer Systems. Singapore: IEEE, 2020. 63–72

24 Naeem H, Ullah F, Naeem MR, *et al*. Malware detection in industrial Internet of Things based on hybrid image

visualization and deep learning model. Ad Hoc Networks, 2020, 105: 102154. [doi: 10.1016/j.adhoc.2020.102154]

25 Ribeiro MT, Singh S, Guestrin C. "Why should I trust you?": Explaining the predictions of any classifier. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco: ACM, 2016. 1135–1144.

26 Ribeiro MT, Singh S, Guestrin C. Anchors: High-precision model-agnostic explanations. Proceedings of the AAAI Conference on Artificial Intelligence, 2018, 32(1): 1527–1535.

27 Guidotti R, Monreale A, Ruggieri S, *et al*. Local rule-based explanations of black box decision systems. 2018, arXiv: 1805.10820.

28 Lundberg SM, Lee SI. A unified approach to interpreting model predictions. Proceedings of the 31st International Conference on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 4768–4777.

29 Guo WB, Mu DL, Xu J, *et al*. LEMNA: Explaining deep learning based security applications. Proceedings of 2018 ACM SIGSAC Conference on Computer and Communications Security. Toronto: ACM, 2018. 364–379.

30 Krause J, Perer A, Ng K. Interacting with predictions: Visual inspection of black-box machine learning models. Proceedings of 2016 CHI Conference on Human Factors in Computing Systems. San Jose: ACM, 2016. 5686–5697.

31 Liu NH, Yang HX, Hu X. Adversarial detection with model interpretation. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. London: ACM, 2018. 1803–1811.

32 Adebayo J, Gilmer J, Muelly M, *et al*. Sanity checks for saliency maps. Proceedings of the 32nd International Conference on Neural Information Processing Systems. Montréal: Curran Associates Inc., 2018. 9525–9536.

33 Yeh CK, Hsieh CY, Suggala AS, *et al*. On the (in) fidelity and sensitivity of explanations. Proceedings of the 33rd International Conference on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 10967–10978.

34 Camburu OM, Giunchiglia E, Foerster J, *et al*. Can I trust the explainer? Verifying post-hoc explanatory methods. arXiv: 1910.02065.

35 Peiravian N, Zhu XQ. Machine learning for Android malware detection using permission and API calls. Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence. Herndon: IEEE, 2013. 300–305.

36 Desnos A, Gueguen G, Bachmann S. Androguard package. https://androguard.readthedocs.io/en/latest/api/androguard.html. (2020-04-30).

37 Li J, Sun LC, Yan QB, *et al*. Significant permission identification for machine-learning-based Android malware detection. IEEE Transactions on Industrial Informatics, 2018, 14(7): 3216–3225. [doi: 10.1109/TII.2017.2789219]

38 刘亚姝, 王志海, 李经纬, 等. 基于卡方检验的 Android 恶意应用检测方法. 北京理工大学学报, 2019, 39(3): 290–294.

39 Fan M, Liu J, Luo XP, *et al*. Android malware familial classification and representative sample selection via frequent subgraph analysis. IEEE Transactions on Information Forensics and Security, 2018, 13(8): 1890–1905. [doi: 10.1109/TIFS.2018.2806891]

40 Sharma S, Singh S. Texture-based automated classification of ransomware. Journal of the Institution of Engineers (India):Series B, 2021, 102(1): 131–142. [doi: 10.1007/s40031-020-00499-w]

(校对责编: 牛欣悦)