1 动态行为采集方法

任何 Android 程序所产生的行为实际上都具有特定的行为模式, 即这些行为可以分解为由原子行为组成的时间序列. 而这些原子行为往往需要调用系统提供的函数接口来实现, 因此可以采用基于系统函数调用的动态行为分析方法从程序对 Android 框架层 API 和 Native 层函数的调用情况中提取行为特征, 再通过对比恶意程序的行为来判断程序的合法性. 基于这个原理, 本文提出了一种以监测系统函数调用为基础的动态行为采集方法, 该方法主要包含两个模块: 行为触发模块和行为采集模块. 行为触发模块的作用是采用多种触发机制尽可能多地触发程序的行为, 使得采集模块可以尽可能全面地采集到软件的所有行为数据. 行为采集模块的作用是通过 hook 技术来监控并记录程序对 Android 系统函数的调用情况以形成行为日志.

1.1 行为触发模块

已知动态行为检测方法是在沙箱或隔离环境中运行程序, 其中第一个关键的步骤就是要能够触发程序的行为. 倘若无法触发恶意程序的全部行为, 则后续的行为采集模块很可能就无法提取到有效的恶意行为数据, 继而一定会影响检测模型最后的检测效率. 因此, 为了提高程序行为触发的覆盖率, 首先需要明确的是被触发的行为所发生在 Android 系统中的层面, 主要包含以下 3 类: (1) 系统层面, 当系统层面的事件 (例如开机、连接网络等) 执行后会立即触发恶意程序的恶意行为, 比如当系统接入互联网后, 恶意程序会开始传输系统中隐私数据. (2) Service 层面, 许多恶意程序会将其恶意行为隐藏在后台服务中, 这样就可以在用户未知的情况下执行操作. (3) UI 及 Activity 层面, 当用户对恶意程序的 UI 进行操作时, 往往就会触发恶意的 Activity 活动. 针对不同系统层面的行为需要采取不同的触发机制, 具体的流程与框架如图 1 所示.

本模块通过使用 Android 模拟器模拟各种系统事件发生的方式来触发程序系统层面的恶意行为. 已知在 Android 系统中, 软件若要监听系统事件就必须在 Manifest 文件中注册广播组件. 因此本模块通过解析目标程序的 Manifest 文件来获取已注册的广播组件信息, 然后利用 ADB 调试桥与 telnet 远程控制命令在模拟器中模拟广播组件相对应的各种事件, 从而触发程序系统层面相对应的行为.

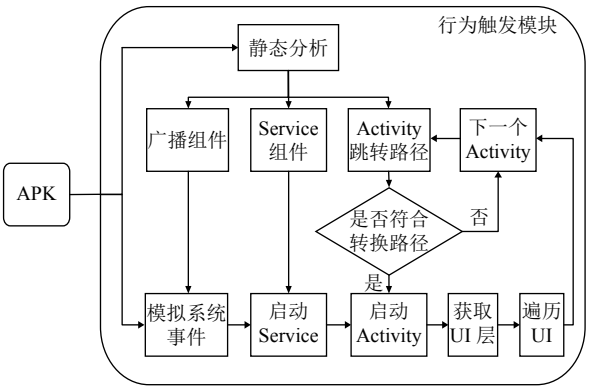

图 1 行为触发模块框架流程图

针对程序在 Service 层面的恶意行为, 本模块采用启动所有 Service 事务的方式来触发潜在该层面的恶意行为. 因为任何应用想要在 Android 系统中使用 Service 事务就必须在 Manifest 的文件中注册 Service 组件, 所以本模块通过解析目标程序的 Manifest 文件来获取已注册的 Service 组件, 然后利用 ADB 工具启动这些 Service 事务, 进而触发执行在这些 Service 中的恶意行为.

针对 UI 及 Activity 层面的恶意行为, 本模块所采用的方法相对复杂一些. 首先, 通过静态分析目标程序来获取软件中 Activity 之间的跳转路径信息; 接着, 针对每一个 Activity, 在启动之后利用 UI 获取部件提取当前 Activity 的 UI 层次信息; 然后, 本模块会利用系统自动化测试工具 MonkeyRunner[9] 针对这些 UI 信息来模拟各种 UI 事件触发程序的行为; 当遍历完当前 Activity 的所有 UI 元素后, 即可按照 Activity 的跳转路径信息继续下一个 Activity 的遍历.

1.2 行为采集模块

考虑到目前 Android 平台的应用软件都会使用混淆、加壳技术来保护自身的程序代码, 因此本模块采用了一种基于动态分析的多层次的行为数据提取方法以应对该问题. 本模块所采用的方法主要以 Android 逆向技术和 Android 安全框架为理论基础, 分别对 Android 的应用框架层和 Native 层的程序行为进行采集.

针对 Android 应用框架层的行为数据, 本模块主要通过 Android 注入技术以及 Java hook 技术进行采集的. 已知 Android 系统一般采用虚拟机对应用层的软件程序进行隔离和管理, 其中 4.4 版本之前采用的是 Dalvik 虚拟机, 而 4.4 版本之后采用的是 Android Runtime. 在 Dalvik 虚拟机进行 Hook 的原理是: 首先

将虚拟机里面的 Java 方法的 Method 标识字段改为 nativeMethod; 接着自己编写一个 Native 方法, 在这个 Native 方法中, 再自定义一个 Java 方法, 并在这个方法里调用了原方法; 然后在调用前后分别通过注入技术插入钩子, 从而可以记录原方法的调用情况了; 最后把原 Java 方法的 nativeFunc 字段指向这个 Native 方法. 当原方法被调用时, 首先会调用到自定义的 Native 方法, 在调用过程中, 原函数的调用数据就被采集. 对于 Android Runtime 进行 hook 的原理实际上是类似的, 已知 Java 层的每一个方法在 Android Runtime 实现中都对应一个 ArtMethod 结构体, 只要把原方法的结构体内容通过注入方式替换为新的结构体内容, 当然原方法被调用的时候, 真正执行的就是注入的新方法的指令. 为了兼容不同版本的 Android 虚拟机, 本模块采用了同时支持 Dalvik 虚拟机和 ART 虚拟机的 Xposed[10] 框架对 Android 应用框架层 API 进行 hook 工作.

对于 Android Native 层的行为数据, 本模块则采用基于 Inline hook[11] 的动态注入技术进行采集. 对比基于全局偏移表的 hook 方法, Inline hook 的优势在于不会受到表的限制, 能够修改重写内存中任意一处的指令, 从而实现对系统函数的挂钩. Inline Hook 的实现原理是首先获得被 hook 系统函数的入口地址, 然后在入口地址处插入强制跳转指令, 使进程跳转到特定函数中, 在该函数中执行监听并记录程序的行为数据, 从而就实现了对 Android 底层接口函数调用情况的监测.

当目标程序的一系列操作行为被成功触发之后, 各种操作行为必然依靠应用框架层或系统 Native 层的函数接口来实现, 此时行为采集模块就可以利用上述方法对函数接口的调用情况进行监测与记录, 根据记录结果即可生成程序的行为日志. 本实验所生成的行为日志的条目及相关含义如表 1 所示.

表 1 目标程序行为日志条目说明

条目名称	说明
Method Category	被调用函数所属的类型
Method Name	被调用函数的名称及所属包的包名
Arguments	调用函数是传递的参数

2 基于 textCNN 的检测框架

卷积神经网络 CNN 曾被广泛应用于计算机视觉领域, 随着深度学习研究的不断深入, 不少学者也开始使用 CNN 模型来处理自然语言处理问题. textCNN[12] 即是由 Yoon Kim 提出的一种用于处理文本分类问题的卷积神经网络模型. textCNN 的主要思想是使用多个通道以及多个不同大小的卷积核, 并通过一维卷积的方式提取词向量矩阵的特征, 然后使用最大池化层从特征矩阵选出每个通道中的最大值, 与其他通道的最大值进行拼接, 组合成最终的特征向量, 最后通过全连接层计算分类的概率. 经过动态行为触发模块和采集模块后, 程序的行为数据实际上转换为了文本数据, 因此可以采用 textCNN 模型来进行处理.

2.1 嵌入层预处理文本数据

对于文本类数据, 首先需要将自然语言数值化, 以方便后续处理. 对于动态行为日志文本, 其中每一行的函数调用记录代表一个动态行为, 本文将每一行视为一个行为词汇, 然后分别对每个词汇构建相应的词向量. 将文本中的词汇表征成词向量最简单的方式就是采用 one-hot 编码方法, 但是这种方法也存在比较明显的缺点, 比如生成的向量长度过大以及无法准确表达词汇之间的相似关系. 而 textCNN 模型提供一个隐藏的嵌入层, 可以将 one-hot 向量投影到低维空间里, 在指定维度中编码语义特征. 本文在嵌入层采用的向量表示方法为 fastText[13]. fastText 实际上是 Word2Vec 中跳字模型[14] 的一种改进, 使用子词嵌入的方法将构词信息引入到了模型中.

在 fastText 中, 往往使用子词集合来表示中心词, 假设一个中心词 w, 将其长度在一定范围内的子词及特殊子词的并集记为 G_w. 已知 fastText 中词典由所有词的子词集合的并集构成, 假设 w 的子词 y 在词典中的向量表示为z_y, 则中心词的向量 v_w 就表示成:

$$v_w = \sum_{y \in G_w} z_y \tag{1}$$

对比跳字模型, fastText 训练的词向量可以更加准确地描述动态行为词汇之间相关性. 当在检测过程中即便遇到词典中未曾出现的行为词汇, fastText 模型也可以从与其结构类似的其他词汇中获得对该词汇更好的向量表示.

2.2 使用 textCNN 对数据进行分类

textCNN 模型的核心结构是卷积神经网络, 其基本结构包括输入层、卷积层、池化层、全连接层和输出层. 卷积神经网络用于行为分类的模型结构图[15] 如图 2 所示.

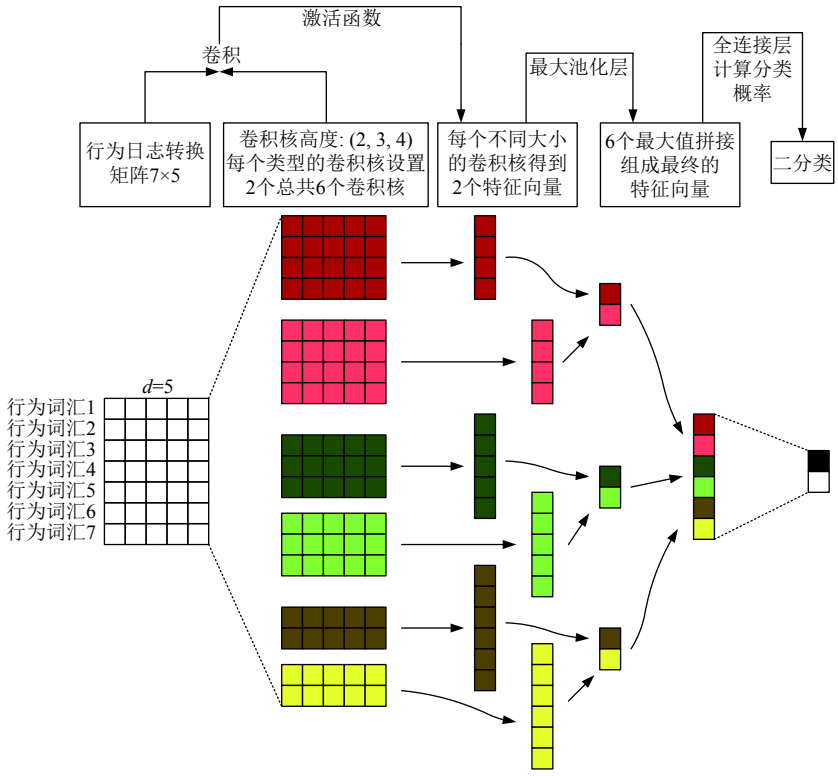

图 2 textCNN 行为分类模型图

从图 2 中可以看到, 在 textCNN 模型的卷积层, 卷积核的宽度与词向量的维度是一致的, 表明模型在提取特征的时候, 将行为词汇作为日志文本的最小粒度. 在 Android 动态行为日志中, 往往恶意行为是由多个连续的动态行为词汇组成, 因此可以通过设置卷积核的高度来提取日志中相邻行为词汇的关联性, 不仅考虑了动态行为文本中的词义而且兼顾了词序以及上下文. 图 2 中卷积核的高度分别设置为 2、3、4, 对应提取文本中的 2-gram、3-gram、4-gram 特征. 对比传统机器学习中的 n-gram 模型, 使用卷积核的优势在于, 只考虑连续词汇的组成, 不会使训练集词表爆炸式增长.

在卷积层中模型使用了不同大小的卷积核, 卷积得到的 feature maps 会具有不同的向量维度. 因此在池化层中, 可以使用 1-max-pooling 方法通过提取每个 feature map 中的最大值来表征该特征向量. 池化层的主要作用是下采样, 通过不断降低数据维度来减少网络中的参数和计算次数. 最后模型将每个特征向量经过 1-max-pooling 池化得到的值拼接起来, 即为池化层最终的输出向量.

textCNN 模型最后一层为全连接层, 池化层的输出结果在进入全连接层之前, 需要进行 Dropout 操作来避免过拟合. 全连接层设置可以参照传统卷积神经网络, 第 1 层采用 ReLU 作为激活函数, 第 2 层则使用 Softmax 分类函数来进行分类.

3 实验分析

3.1 实验环境与数据

本文的实验环境与相关配置如表 2 所示.

表 2 实验环境与配置

实验环境	相关配置
移动端操作系统	Android 6.0
APK分析框架	MobSF v3.0.0
Android Hook框架	Xposed v90-beta3
PC端操作系统	Window 10
编程语言	Python 3.6
深度学习框架	PyTorch 1.4.0

本文实验所需的 Android 恶意软件样本均来自于 VirusShare 网站[16], 其中样本的数量为 1500 个. 同时, 实验所需的 1000 个正常样本均下载于 Android 官方应用市场 Google Play. 对于 1500 个恶意样本和 1000 个正常样本, 分别使用其中的 70% 作为本文模型的训练样本, 剩下的 30% 作为测试样本.

对于每一个样本, 经过动态行为采集框架处理后会得到一个行为日志. 原始的行为日志是 json 数据格式, 为了便于后面 textCNN 模型的处理, 实验需要通过 Python 爬取日志中关键信息, 将其转换为 txt 文档表示. 文档中的每一行即为一个行为 API, 每一行分为 3 个部分, 由空格分隔, 每个部分的说明如表 1 所示.

3.2 实验评价指标

为了充分评估模型的性能, 本文在验证模型训练结果的时候采用了 k 折交叉验证技术. 在本实验中, k 的取值设定为 10, 即将训练集划分为 10 个规模相等且无交集的子集, 在每次训练过程中依次选择 1 个子集作为测试集, 剩余 9 个子集作为训练集, 最终以 10 次计算结果的平均值作为模型评价指标的最终结果.

本实验使用 3 个通用的评价指标即准确率$P_{\text{Precision}}$、召回率P_{Recall}以及 F_1 值来对实验结果进评价. 本实验采用的样本分类混淆矩阵如表 3 所示.

表 3 样本分类结果混淆矩阵

真实类别	预测类别	
	正常样本	恶意样本
正常样本	*TP*	*FN*
恶意样本	*FP*	*TN*

表 3 中, *TP* 表示实际为正常样本, 被预测为正常样本的样本数目; *FP* 表示实际为恶意样本, 被预测为正常样本的样本数目; *FN* 表示实际为正常样本, 被预测为恶意样本的样本数目; *TN* 表示实际为恶意样本, 被预测为恶意样本的数目. 实验评价指标的计算公式如下所示:

$$\text{准确率}: P_{\text{Precision}} = \frac{TP}{TP+FP} \tag{2}$$

$$\text{召回率}: P_{\text{Recall}} = \frac{TP}{TP+FN} \tag{3}$$

$$F_1\text{值}: F_1 = \frac{2P_{\text{Precision}} \times P_{\text{Recall}}}{P_{\text{Precision}} + P_{\text{Recall}}} \tag{4}$$

3.3 实验参数设置

由于实验结果在很大程度上会受到实验参数的影响, 因此在实验中将会对模型的参数进行设置. 首先实验将行为日志转换后的文本最大长度设置为 400, 超过长度的部分将直接进行截断操作. 本实验使用 fastText 模型训练词向量时所设置的参数值如表 4 所示.

表 4 fastText 模型训练参数设置

参数名	参数值
word_ngrams	2
Window上下文窗口	5
最小字符长度	2
最大字符长度	5
迭代周期	5000

在 textCNN 模型的训练过程中, 本实验将卷积核的高度固定设置为 3、4、5. 对于其他参数, 分别设置了几组对照实验, 其中卷积核数量分别对比了 64、128、256 三组值, Dropout 参数分别对比了 0.3、0.4、0.5 三组值, 激活函数对比了 ReLU 和 tanh 两种函数. 通过对比上述几组参数值对实验结果的影响, 最终确定模型的参数如表 5 所示.

表 5 textCNN 模型训练参数设置

参数名	参数值
卷积核高度	3, 4, 5
卷积核数量	128
Dropout	0.5
激活函数	ReLU
Pooling	1-max-pooling
迭代周期	2000

3.4 实验结果与分析

为了验证本文提出的对于 Android 应用程序的动态行为采集框架的有效性, 本文设置了 3 组对照实验, 分别采用不同的方法来提取 Android 软件的行为特征. 第 1 组采用传统的静态行为分析法来提取程序的静态行为文本数据, 实验中具体参考文献 [3] 提出的方法; 第 2 组采用基于污点跟踪技术的分析方法来提取程序的行为文本数据, 实验中具体参考文献 [5] 中方法; 第 3 组就采用本文的基于系统函数调用的动态行为采集方法来提取实验数据. 3 组实验均使用相同的 Android 样本数据集, 然后将 3 组实验得到的 3 组文本数据集都经过 fastText 模型预处理以及 textCNN 模型进行训练, 最后相同模型在不同数据集上的实验结果如表 6 所示.

表 6 不同数据集实验结果对比

数据集	$P_{\text{Precision}}$	P_{Recall}	F_1
第1组	0.7732	0.7693	0.7713
第2组	0.8756	0.8861	0.8810
第3组	0.9264	0.9342	0.9305

从表 6 中数据可以看出, 本文提出的动态行为采集框架可以更有效的提取出 Android 程序中行为特征.

第一组数据集的检测准确率相对较低, 原因在于现在的 Android 应用程序都添加了代码混淆和代码保护壳等对抗反编译的措施, 对比而言本文方法几乎没有受到这些措施的影响. 第 2 组与第 3 组实验结果的对比证实了基于污点跟踪技术分析方法存在的局限性, 即无法检测恶意程序在 Android Native 层的恶意行为.

为了对比 textCNN 模型与传统机器学习模型及传统深度学习模型在 Android 恶意程序上的检测能力, 本文设置了 4 组对照实验, 将 textCNN 模型与 SVM 模型、RandomForest 模型以及循环神经网络 LSTM 模型进行了对比. SVM 与 RandomForest 均是机器学习中比较成熟的分类模型, LSTM 也是一种可以高效处理文本问题的神经网络模型. 对于 SVM、RandomForest 和 LSTM 模型中使用的词向量, 本实验均采用传统 Word2Vec 模型预处理行为数据集来得到. 不同模型在相同测试集上的实验结果对比如表 7 所示.

表 7 不同模型实验结果对比

模型	$P_{\text{Precision}}$	P_{Recall}	F_1
SVM	0.8713	0.8863	0.8789
RandomForest	0.9047	0.8907	0.8977
LSTM	0.9268	0.9212	0.9240
textCNN	0.9386	0.9411	0.9397

从表 7 中可以看出, 对比 SVM 与 RandomForest 这两种机器学习模型, textCNN 模型的准确率、召回率和 F_1 值分别高出了 6.73%、5.48%、6.08% 和 3.39%、5.04%、4.2%, 同时 LSTM 模型的实验结果较两种机器学习模型而言也有明显的提升. 这说明采用神经网络模型较传统机器学习方法而言确实可以提高 Android 恶意程序的检测准确率. 对比 textCNN 与 LSTM, 可以看到在相同测试集上 textCNN 模型的检测准确率还是有所提升的, 虽然这种提升是有限的, 但是应该考虑到 textCNN 具有模型复杂度更小且训练速度更快的先天优势. 图 3 展示的是 4 种模型在相同数据集上损失函数值随迭代次数变化而变化的情况. 从图中可以看出, textCNN 模型的损失值相对于其他 3 种模型不仅下降速度较快, 而且最终收敛到一个更低的稳定值.

综合两部分的实验结果可以表明本文提出的基于 textCNN 模型的利用动态行为分析方法的 Android 恶意程序检测模型是切实可行的方案, 能够有效提升 Android 恶意程序检测的准确性和有效性.

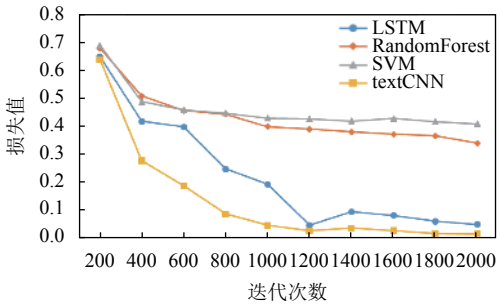

图 3 不同模型损失值变化图

4 结论与展望

本文提出了基于 textCNN 模型的 Android 恶意程序检测方法. 该方法实现了包含多种触发机制的行为触发系统, 有效提高了程序动态行为的检测覆盖率; 通过 hook 技术动态采集程序系统函数接口和应用层 API 的调用情况来生成动态行为日志; 通过 fastText 算法对行为日志进行词嵌入处理, 将文本数据转换为词向量数据; 使用 textCNN 模型对程序的行为数据进行检测与分类. 在实验环节, 本文使用 1500 个恶意样本和 1000 个正常样本, 分别对动态行为采集模块和基于 textCNN 的检测模块设置了对照实验, 实验结果证实了本文提出的这两个模块是切实可行的且能够有效提高 Android 恶意软件检测的准确性.

在下一步工作中, 针对动态行为采集模块, 将研究更加智能的动态行为触发机制, 以提高行为检测的覆盖率; 针对动态行为检测与分类模块, 将继续研究分类模型的优化问题, 并尝试将捕捉行为词汇前后依赖关系的机制以及注意力机制引入到模型中, 使模型能够更高效地检测 Android 程序是否存在恶意性.

参考文献

1 360 互联网安全中心. 2019 年手机安全状况报告. 2020. 1–9.
2 吴震雄. Android 恶意软件静态检测方案研究 [硕士学位论文]. 南京: 南京邮电大学, 2015.
3 Ma Z, Ge HR, Liu Y, *et al*. A combination method for Android malware detection based on control flow graphs and machine learning algorithms. IEEE Access, 2019, 7: 21235–21245. [doi: 10.1109/ACCESS.2019.2896003]
4 Enck W, Gilbert P, Han S, *et al*. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems,

2014, 32(2): 5. [doi: 10.1145/2619091]

5 Sun MS, Wei T, Lui JCS. TaintART: A practical multi-level information-flow tracking system for android RunTime. Proceedings of ACM SIGSAC Conference on Computer and Communications Security. Vienna, Austria. 2016. 331–342. [doi: 10.1145/2976749.2978343]

6 Sun YS, Chen CC, Hsiao SW, *et al*. ANTSdroid: Automatic malware family behaviour generation and analysis for Android apps. Proceedings of the 23rd Australasian Conference on Information Security and Privacy. Wollongong, Australia. 2018. 796–804. [doi: 10.1007/978-3-319-93638-3_48]

7 孙润康, 彭国军, 李晶雯, 等. 基于行为的 Android 恶意软件判定方法及其有效性. 计算机应用, 2016, 36(4): 973–978. [doi: 10.11772/j.issn.1001-9081.2016.04.0973]

8 Vinod P, Zemmari A, Conti M. A machine learning based approach to detect malicious android apps using discriminant system calls. Future Generation Computer Systems, 2019, 94: 333–350. [doi: 10.1016/j.future.2018.11.021]

9 Hu YJ, Azim T, Neamtiu I. Versatile yet lightweight record-and-replay for Android. Proceedings of 2015 ACM SIGPLAN International Conference on Object-oriented Programming, Systems, Languages, and Applications. Pittsburgh, PA, USA. 2015. 349–366. [doi: 10.1145/2814270.2814320]

10 柯懂湘, 潘丽敏, 罗森林, 等. 基于随机森林算法的 Android 恶意行为识别与分类方法. 浙江大学学报 (工学版), 2019, 53(10): 2013–2023. [doi: 10.3785/j.issn.1008-973X.2019.10.019]

11 Gu J, Xian M, Chen T, *et al*. A Linux rootkit improvement based on inline hook. Proceedings of the 2nd International Conference on Advances in Mechanical Engineering and Industrial Informatics (AMEII 2016). Hangzhou, China. 2016. 800–805. [doi: 10.2991/ameii-16.2016.155]

12 Kim Y. Convolutional neural networks for sentence classification. Proceedings of 2014 Conference on Empirical Methods in Natural Language Processing. Doha, Qatar. 2014. 1746–1751. [doi: 10.3115/v1/D14-1181]

13 Bojanowski P, Grave E, Joulin A, *et al*. Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 2017, 5: 135–146. [doi: 10.1162/tacl_a_00051]

14 Mikolov T, Sutskever I, Chen K, *et al*. Distributed representations of words and phrases and their compositionality. Proceedings of the 26th International Conference on Neural Information Processing Systems. Lake Tahoe, CA, USA. 2013. 3111–3119.

15 Zhang Y, Wallace B. A sensitivity analysis of (and Practitioners' Guide to) convolutional neural networks for sentence classification. arXiv: 1510.03820, 2015.

16 侯勤胜. 基于网络行为分析的 Android 恶意软件动态检测 [硕士学位论文]. 徐州: 中国矿业大学, 2017.