DeepRD：基于 Siamese LSTM 网络的 Android 重打包应用检测方法

汪润[1,2]，唐奔宵[1,2]，王丽娜[1,2]

（1. 空天信息安全与可信计算教育部重点实验室，湖北 武汉 430072；2. 武汉大学国家网络安全学院，湖北 武汉 430072）

摘 要：目前，Android 平台重打包应用检测方法依赖于专家定义特征，不但耗时耗力，而且其特征容易被攻击者猜测。另外，现有的应用特征表示难以在常见的重打包应用类型检测中取得良好的效果，导致在实际检测中存在漏报率较高的现象。针对以上 2 个问题，提出了一种基于深度学习的重打包应用检测方法，自动地学习程序的语义特征表示。首先，对应用程序进行控制流与数据流分析形成序列特征表示；然后，根据词向量嵌入模型将序列特征转变为特征向量表示，输入孪生网络长短期记忆（LSTM, long short term memory）网络中进行程序特征自学习；最后，将学习到的程序特征通过相似性度量实现重打包应用的检测。在公开数据集 AndroZoo 上测试发现，重打包应用检测的精准率达到 95.7%，漏报率低于 6.2%。

关键词：重打包；深度学习；孪生网络；长短期记忆；安全与隐私

中图分类号：TP309.1

文献标识码：A

doi: 10.11959/j.issn.1000−436x.2018148

DeepRD: LSTM-based Siamese network for Android repackaged applications detection

WANG Run[1,2], TANG Benxiao[1,2], WANG Li'na[1,2]

1. Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education, Wuhan University, Wuhan 430072, China
2. School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China

Abstract: The state-of-art techniques in Android repackaging detection relied on experts to define features, however, these techniques were not only labor-intensive and time-consuming, but also the features were easily guessed by attackers. Moreover, the feature representation of applications which defined by experts cannot perform well to the common types of repackaging detection, which caused a high false negative rate in the real detection scenario. A deep learning-based repackaged applications detection approach was proposed to learn the program semantic features automatically for addressing the above two issues. Firstly, control and data flow analysis were taken for applications to form a sequence feature representation. Secondly, the sequence features were transformed into vectors based on word embedding model to train a Siamese LSTM network for automatically program feature learning. Finally, repackaged applications were detected based on the similarity measurement of learned program features. Experimental results show that the proposed approach achieves a precision of 95.7% and false negative rate of 6.2% in an open sourced dataset AndroZoo.

Key words: repackaging, deep learning, Siamese network, LSTM, security and privacy

收稿日期：2018−04−03；**修回日期**：2018−06−28

通信作者：王丽娜，lnwang@whu.edu.cn

基金项目：国家自然科学基金资助项目（No.U1536204）；中央高校基本科研业务费专项资金资助项目（No.2042018kf1028）；国家高技术研究发展计划（“863”计划）基金资助项目（No.2015AA016004）

Foundation Items: The National Natural Science Foundation of China (No.U1536204), The Central University Basic Business Expenses Special Funding for Scientific Research Project (No.2042018kf1028), The National High Technology Research and Development Program of China (863 Program) (No.2015AA016004)

1 引言

Android 和 iOS 作为移动应用市场的两大主流平台，截至 2017 年底，它们的官方应用商店 Google Play 和 Apple Store 上的移动应用数量已经超过了 300 多万。近年来，移动平台特别是 Android 平台已经逐渐成为恶意软件泛滥的重灾区，给用户的安全和隐私带来了严重的威胁。

报告显示，超过 98%的移动恶意软件来自 Android 平台，在这些恶意软件中，重打包应用超过了 86%[1]。Android 重打包应用示例如图 1 所示。重打包应用是指恶意开发者在保持正常合法应用功能不变的前提下，通过添加、修改应用程序代码或资源文件生成的非法应用。这类应用通过各种渠道让用户下载安装使用，达到窃取用户隐私、攻击用户设备等目的。获取非法收益和传播恶意软件是开发者制造重打包应用的两大动机。

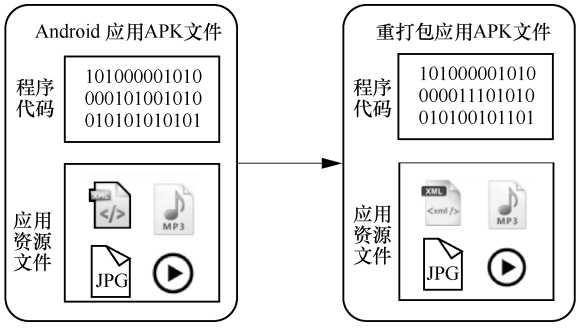


图 1 Android 重打包应用示例

研究分析发现，Android 平台上重打包应用传播广泛的主要原因如下：①Android 应用容易被反编译修改，应用程序代码的混淆程度低，反编译后可读性强；②Android 平台开放程度高，第三方应用市场没有严格的审查机制，导致用户可以随意地安装非官方应用商店的应用；③Android 市场份额高，传播恶意软件可以获取大量的非法收益。

为了构建健康的移动应用市场，国内外研究人员针对 Android 平台的重打包应用展开了一系列的相关研究[2-9]。具体来说，分为以下 2 类：第一类是基于应用相似性比较的方法；第二类是基于重打包应用行为检测的方法[10-14]。其中，第一类方法又可以分为基于代码克隆的检测和基于应用资源文件的检测这 2 种方法，分别使用程序代码特征如控制流图（CFG, control flow graph）、程序依赖图（PDG, program dependency graph）以及应用资源文件作为应用的特征表示。通过度量应用特征表示之间的相似性实现重打包应用的检测。有研究发现，部分重打包应用中恶意的代码片段与正常的代码片段逻辑上独立，但与正常的功能代码片段交互性差。基于上述发现，研究人员提出了第二类基于重打包应用行为的检测方法，提取可以表示正常代码片段与恶意代码片段行为差异的特征，然后训练一个机器学习的分类模型实现重打包应用的检测。

目前，重打包应用检测方法共同存在的不足在于，这些方法都强烈地依赖专家的特征定义。第一类基于应用相似性比较的检测方法，需要分析应用的程序代码或资源文件来定义应用的特征表示；第二类基于重打包应用行为的检测方法，也需要定义程序代码的特征表示。然而，针对这类依赖专家特征定义的检测方法，攻击者可以通过试探的方法猜测特征表示，精心地构造恶意应用来绕过检测[15-17]。此外，不同类型的重打包应用，能够实现最佳检测效果的应用特征表示也不同。例如，针对汉化等重打包应用，适用于基于应用资源文件特征表示的检测方法；针对 Piggy-packing 应用[18]的检测，则适用于基于重打包应用行为的检测方法，分析 Piggy-packing 应用中注入的恶意组件和正常功能代码在功能上的差异，利用代码功能的差异抽取重打包应用检测的特征表示。综上，现有的依赖专家定义特征的检测方法，难以定义一种通用的应用特征表示可以适用于所有的重打包应用类型检测，这导致在实际检测中存在漏报率高的问题。

针对现有的重打包应用检测方法依赖于专家定义特征存在耗时耗力，且容易被攻击者猜测出检测的特征表示，通过精心构造恶意应用绕过检测的问题以及现有方法在实际检测中漏报率高的现象，本文提出一种基于深度学习的 Android 重打包应用检测方法，借助于深度学习强大的特征学习能力[19]，自动地学习应用的程序代码特征，通过程序代码的相似性度量实现重打包应用的检测。深度学习在图像识别[20]、语音识别[21]以及自然语言处理[22]等领域已经取得了突破性进展。相关学者利用深度学习研究安全领域中的一些重要问题并取得了非常显著的效果[23-32]，例如，漏洞检测[23]、缺陷预测[25]、二进制分析[27-28]以及恶意软件检测[24,30-32]等。受这类工作的启发，本文利用深度学习解决目前 Android 重打包应用检测中面临的依赖于专家定义特征表示和漏报率高这 2 个问题。然而，目前应用深度学

习解决图像识别、语音识别等方法并不能直接应用于重打包应用的检测。在应用深度学习检测重打包应用时，需要解决以下 3 个基本问题。

1) 应用分析与检测的基本单元。在正常的合法应用基础上添加完整组件以及部分代码重用是构造重打包应用的 2 种主要方式。在 Android 重打包应用检测时，不仅需要判断一对应用是否为重打包应用，还需要识别出被重用的代码片段。

2) 应用的向量表示。在深度学习模型的训练及预测阶段，模型的输入都是向量。因此，需要将应用表示成向量的形式。本文利用深度学习检测 Android 平台的重打包应用，不能将应用表示成任意的向量形式，在应用向量表示时需要保留与应用相似性度量有关的程序语义信息。

3) 深度学习模型的选择。深度学习在其他领域取得了显著的效果，但是不同的深度学习模型擅长解决的问题不尽相同。本文中，通过程序代码的相似性度量实现重打包应用的检测，这类问题类似于自然语言处理中的文本相似性检测。因此，选择的深度学习模型应擅长序列特征数据的表示。

针对应用深度学习检测 Android 平台重打包应用中面临的 3 个基本问题，本文提出将应用函数作为分析与检测的基本单元，应用函数被表示成序列特征形式并进行向量化处理，然后输入具有 LSTM 的 Siamese 网络中学习程序代码的语义特征，最后通过应用程序代码的相似性度量判断是否为重打包应用。本文中应用函数的序列特征及特征向量表示方法如下：首先，获取应用函数中 API 的调用序列以及与 API 存在数据依赖关系的上下文信息形成应用函数的序列特征；然后，将应用函数的序列特征表示成抽象字符串形式；最后，使用词向量嵌入模型将抽象字符串表示成向量形式输入深度学习模型 Siamese LSTM 网络中训练。本文的主要贡献如下。

1) 提出基于深度学习的 Android 重打包应用检测方法，并基于该方法设计并实现一套原型系统 DeepRD。利用 Siamese LSTM 网络学习程序的语义特征表示，实现重打包应用的检测。

2) 提出一种基于词向量嵌入的应用函数向量化表示方法。利用应用函数的 API 调用序列及与 API 存在数据依赖的上下文信息来保留程序语义信息，使用词向量嵌入模型生成应用的向量表示。

3) 在程序代码的相似性度量中，本文使用 2 个 LSTM 网络自动地学习程序代码的序列特征表示，利用 Siamese 网络度量程序代码的相似性。

4) 在公开数据集 AndroZoo[33]上验证本文方法的有效性，实验结果发现，本文方法可以达到检测精准率为 95.7%，漏报率低于 6.2%。

2　相关工作

2.1　基于特征的重打包应用检测

目前，重打包应用检测方法强烈地依赖于专家定义应用的特征表示，主要分为以下 2 类方法：基于应用相似性比较的检测方法和基于重打包应用行为的检测方法。

基于应用相似性比较的检测方法又可以分为基于代码克隆的检测方法[1,34-39]和基于应用资源文件的相似性检测方法[40-43]。这 2 种方法分别将程序代码的特征和应用资源文件的特征作为应用的特征表示，然后通过度量特征之间的相似性来判断是否为重打包应用。DNADroid[34]和 Andarwin[35]定义程序依赖图作为应用的特征表示，程序依赖图的顶点之间通过数据依赖建立关联，挖掘应用之间的同构子图来判定是否为重打包应用。ViewDroid[40]定义应用的用户界面（UI, user interface）作为应用的特征表示，使用有向图表示应用的用户界面，图的顶点表示应用的视图，边表示视图之间可以通过事件调用进行切换。

基于重打包应用行为的检测方法通过分析应用中正常的功能代码片段与恶意插入的代码片段在功能上的差异，定义代码特征表示，然后训练分类模型完成检测。文献[13]发现重打包应用中加载的恶意组件不是应用的核心代码，使用基于 PDG 的模块解耦技术将应用程序代码分为核心模块和非核心代码，设计了一种特征指纹技术抽取核心模块的语义特征，实现重打包应用的检测。

以上基于特征的重打包应用检测方法强烈地依赖于专家定义各种巧妙的特征，研究发现，攻击者可以猜测检测系统所使用的特征，然后通过精心地构造应用绕过检测[15-17]。此外，现有的应用特征表示方法无法应用于所有的重打包应用类型的检测中，导致在真实的检测中产生报率高的问题。本文方法能够有效地克服现有方法存在的以上不足，

不但可以保证较高的检测精准率，而且漏报率远低于现有的检测方法。

2.2 深度学习在网络安全中的应用

近年来，有学者利用深度学习研究网络安全中一些重要的问题，例如，漏洞检测、缺陷预测、二进制程序分析、恶意软件检测等，特别是在恶意软件检测方面取得了一系列重要的研究成果。Droid-Sec 利用静态分析和动态分析技术提取了 200 多个移动应用特征，通过训练一个深信念网络（DBN, deep belief network）检测 Android 恶意应用[24]。DeepSign 提出一种基于 DBN 的恶意软件特征自动生成及分类检测方法，捕获程序在沙箱中运行的行为数据作为 DBN 的输入，来识别未知的恶意软件[31]。Deep4MalDroid 通过动态地运行 Android 应用，获取应用在执行过程中的动态行为特征（Linux 内核层的系统调用），然后将 Linux 内核层的系统调用表示成带有权重的有向图输入栈式自动编码器（SAE，stacked auto encoder）中，自动地学习 Android 恶意应用的特征来检测未知的恶意应用[32]。

以上这些方法均面向通用的恶意软件检测，并将获取应用的恶意行为特征作为深度学习模型的输入。这些方法并不适用于本文的重打包应用检测，主要体现在以下 2 点：① 这些方法均将恶意软件的检测看作一个分类问题，而本文研究的是应用的相似性判定问题；② 重打包应用的行为与正常合法应用的行为相似，因此上述基于恶意行为的建模方法并不适用于重打包应用的检测。

本文受到以上工作的启发，应用深度学习解决 Android 平台的重打包应用检测问题，丰富了深度学习在安全领域中的应用。

3 相关背景知识

3.1 Android 重打包应用

重打包应用检测的 2 类典型方法如图 2 所示。一类方法是基于应用相似性的比较，通过定义应用特征的相似性计算函数，设置阈值来判定是否为重打包应用；另一类方法是抽取重打包应用的行为特征，然后使用大量的重打包应用和原应用的标记样本训练分类模型，最后利用训练好的分类模型预测一对应用是否为重打包应用。本文方法属于第一类，通过度量应用的相似性来判定是否为一对重打包应用，但是本文中用于应用相似性度量的特征是通过自动学习得到的。

3.2 循环神经网络

循环神经网络（RNN, recurrent neural network）是一种擅长处理序列数据的人工神经网络。近年来，RNN 也被应用于程序分析的相关领域[27-28]。RNN 与传统神经网络的不同在于，隐藏层的节点之间有连接，隐藏层的输入包括输入层的输出和上一时刻隐藏层的输出。隐藏层第 t 步状态 h_t 的计算式为

$$h_t = f(Ux_t + Wh_{t-1}) \tag{1}$$

其中，x_t 为第 t 步的输入，f 为非线性激活函数，如 tanch 或 ReLU 等，U、W 为训练中共享的权重。由于 RNN 模型在训练过程中依赖前面步骤的数据，因此，RNN 会随着序列长度的增加存在梯度消失或梯度爆炸等问题。因此，有学者提出 LSTM 和 GRU（gated recurrent unit）模型。相比于 GRU 模型，LSTM 模型更适用于序列数据的建模。LSTM 的长时间记忆网络特性，使其能够有效地应用于程序分析等相关领域，本文使用

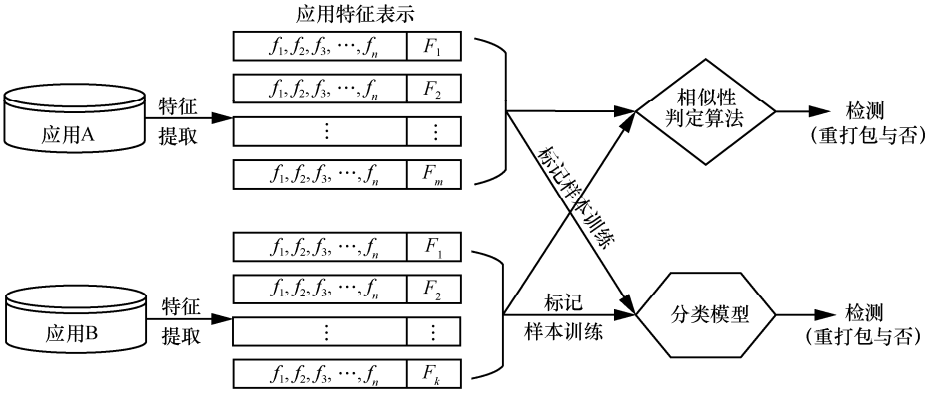


图 2 重打包应用检测方法示意

LSTM 学习应用程序的序列特征表示。

4　总体设计

本文旨在研究一种不依赖于专家特征定义的重打包应用检测方法，降低现有方法在实际检测中存在的高漏报率现象。本节具体描述文中方法 DeepRD 的总体设计，首先讨论在 DeepRD 设计与实现时面临的 3 个基本问题。

1) 应用分析与检测的基本单元

针对本文重打包应用检测的两大目标：判断一对应用是否为重打包应用；识别被重用的代码片段，将应用整体作为分析与检测的单元存在无法识别出被重用的代码片段的问题。研究发现，超过 86.2%的重打包应用是通过修改应用函数的程序代码生成的。实验验证发现，将应用整体作为基本分析与检测的单元，会存在漏报率高以及检测精准率低等问题。因此，本文使用应用函数作为分析与检测的基本单元可以满足文中细粒度分析的需要。

2) 应用的向量表示

本文将应用函数表示成序列特征的形式，使用词向量嵌入模型将序列特征进行向量化表示。抽取平台 API 及其存在数据依赖的上下文信息，根据 API 的调用序列形成序列特征表示，可以保留程序的语义信息。此外，基于数据依赖的程序代码特征表示，可以有效地抵御攻击者在应用程序中插入代码、修改代码顺序等攻击操作。

3) 深度学习模型的选择

在应用的向量表示中，本文将应用函数表示成序列特征形式。RNN 被广泛地应用于自然语言处理的相关任务中，擅长序列数据的建模。此外，研究显示 RNN 在程序分析方面也取得了不错的效果。3.2 节提到在序列长度增加时，RNN 会存在梯度消失或爆炸的问题。因此，本文使用 LSTM 网络学习程序的序列特征表示，利用 Siamese 网络度量特征的相似性。

DeepRD 主要包括模型训练和重打包应用检测这 2 个阶段，分别如图 3 和图 4 所示。图 3 为 DeepRD 的训练阶段，输入大量的第三方重打包应用对以及第三方库的白名单，输出训练好的应用函数相似性度量模型。训练阶段主要包括 Android 应用的预处理、特征抽取、应用向量化表示以及 MaLSTM 模型训练这 4 个步骤。MaLSTM 模型是一种用于处理变长序列数据的 Siamese LSTM 网络。图 4 为 DeepRD 的检测阶段，输入为 2 个待检测的应用，与训练阶段相似，也包括预处理、特征抽取和应用向量化表示等步骤，然后利用训练好的 MaLSTM 模型度量应用函数的相似性来判断是否为重打包应用。各步骤具体描述的内容如第 5 节所示。

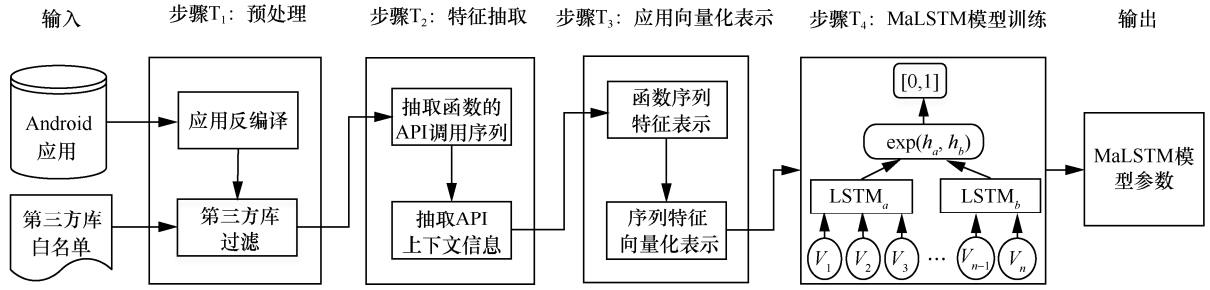


图 3　DeepRD 框架—训练阶段

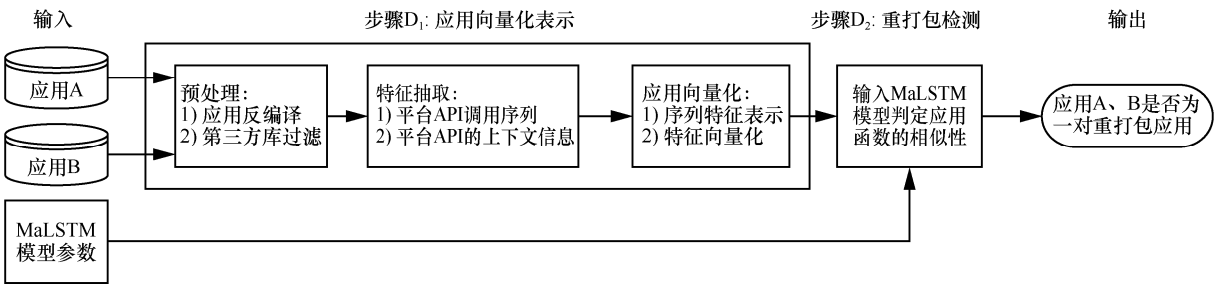


图 4　DeepRD 框架—检测阶段

5 DeepRD 方法

5.1 预处理

Android 应用安装（APK, Android package）文件主要由程序代码 dex 文件和应用资源文件构成。在预处理阶段的应用反编译是为了获取 APK 文件的程序代码，提取应用函数的特征，生成应用的向量表示。为了提高开发效率或实现应用的某些功能要求，程序开发人员在编写 Android 应用程序时，会大量地调用第三方库，如广告库和功能库等。

由于第三方库在应用中占用了一定比例的程序代码片段，因此在本文基于程序代码相似性度量的重打包应用检测方法中，第三方库会严重地影响检测的效果，并在某种程度导致误报。为了降低检测中由于第三方库带来的误报，本文使用白名单过滤应用中的第三方库。

5.2 特征抽取

深度学习模型的输入都是向量，在应用向量化表示时，需要保留应用程序的语义信息用于程序代码片段的相似性度量。本文重打包应用分析与检测的基本单元是应用函数，因此本文抽取应用函数的特征形成应用的特征表示。

本文抽取的特征用于判定应用的相似性，抽取的程序特征应满足以下基本条件：①特征应包含有应用的语义信息且可分辨性强，相似的程序片段特征相似，不同的程序片段特征差异明显；②特征易抽取，不依赖于复杂的静态分析方法，且可抵御常见的代码混淆攻击。基于以上程序特征表示中需要满足的 2 个基本条件，本文提出了一种基于平台 API 调用序列的特征表示方法，抽取平台 API 及其存在数据依赖的上下文信息作为程序的特征表示。

Android 应用的恶意行为总是与平台 API 的不合理使用有关，特别是与应用权限申请有关的敏感 API 有密切的联系，它们在恶意应用中被开发者频繁地调用以达到某些非法的目的。一些研究中将 API 的调用序列作为恶意应用检测的特征[44-45]。另外，有研究发现，恶意开发者在修改或插入恶意代码生成重打包应用时，也会频繁地调用平台 API，收集用户的隐私数据或执行其他敏感的恶意操作等[11]。基于以上 2 点启发，本文提出了基于 API 调用序列的特征表示方法。该特征表示方法有如下优点：①平台 API 在程序分析中不易受到代码混淆的影响；②平台 API 调用序列提取简单，不依赖于复杂的静态分析方法；③API 的上下文信息保留了程序的语义信息，且基于数据依赖的平台 API 上下文信息可以防御代码插入等攻击[34-35]；④定义 API 的上下文信息作为应用的特征表示，还可以抵御攻击者通过修改 API 的调用顺序，实施针对深度学习模型的对抗攻击。

特征抽取阶段主要包括抽取函数的 API 调用序列和抽取 API 的上下文信息这 2 个步骤，如图 3 中步骤 T_2 特征抽取所示。下面详细描述这 2 个步骤。

1)抽取函数的 API 调用序列

Android 应用开发包括基于 SDK（software development kit）的 Java 语言开发和基于 NDK（native development kit）的 C/C++语言开发这 2 种，NDK 开发主要用于在 APK 文件中可以调用由 C/C++语言编写生成的 so 文件。本文仅考虑修改 Java 语言生成的重打包应用，且目前尚未有研究发现通过修改 C/C++语言生成的重打包应用。因此，本文特征抽取阶段分析的 API 包括 Android 平台 SDK 中的库和 API 以及 Java 平台 JDK 中的库和 API。本文通过静态的控制流分析，获取应用函数中 API 的调用序列。

2)抽取 API 的上下文信息

在应用程序中，语句之间的关系主要分为数据依赖和控制依赖这 2 种形式。数据依赖是指如果语句 U'数据依赖于 U，则 U'中有变量的值取决于语句 U。本文 API 的上下文信息是指与 API 在函数体中存在数据依赖的语句，这里的数据依赖包括前向数据依赖和后向数据依赖这 2 种，如图 5 所示。

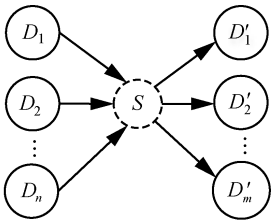


图 5 平台 API 的上下文信息示意

在图 5 中，S 为 API 调用序列中的元素，语句 $D_1,D_2,\cdots,D_n$ 均数据依赖于 S，S 数据依赖于 $D'_1,D'_2,\cdots,D'_m$。本文用 *S_prior* 表示 S 的前向数据依赖语句集合，$S\_prior=\{D_1, D_2,\cdots, D_n\}$；*S_post* 表示 S 的后向数据依赖语句集合，$S\_post=\{D'_1, D'_2,\cdots, D'_m\}$。本文平台 API 及其上下文信息表示为一个三元组的形式，即 $V=<S\_prior, S, S\_post>$。

在特征抽取阶段，API 及其上下文信息特征表示 V 为本文深度学习模型输入的基本单元。本文抽取的程序特征包含程序的语义信息，还可以抵御针对深度学习模型的对抗攻击。接下来，具体介绍如何将该阶段抽取的特征 V 形成应用函数的序列特征并进行向量化表示。

5.3　应用向量化表示

在应用向量化表示中，根据 5.2 节特征抽取方法形成应用函数的序列特征，然后使用词向量嵌入模型对应用函数的序列特征进行向量化表示。具体分为函数序列特征表示和序列特征向量化表示这 2 个部分的内容。接下来，对其进行详细描述。

1) 函数序列特征表示

在 Android 应用中，开发人员调用大量的平台 API 实现应用的功能需求。应用函数中 API 的调用顺序可用于形成函数的序列特征表示，一些研究将 API 的调用顺序作为恶意软件检测的特征。本文则通过静态的控制流分析获取应用函数中 API 的调用序列，然后根据 5.2 节 API 上下文信息的提取方法，形成应用函数的序列特征表示。函数序列特征表示如算法 1 所示。

算法 1　函数序列特征表示算法

输入　应用函数的程序代码 *func_code*

输出　应用函数的序列特征表示 *F*

1) *api_Call*=*getCallSequence*(*func_code*);
/*获取应用函数的 API 调用序列*/

2) for *S* in *api_Call*

3)　*S_prior*=*getPriorDep*(*S*); /*获取 *S* 的前向数据依赖语句*/

4)　*S_post*=*getPostDep*(*S*); /*获取 *S* 的后向数据依赖语句*/

5)　*V*=<*S_prior*, *S*, *S_post*>; /*平台 API 及其上下文信息的特征表示*/

6)　*F*←*V*; /*将 *V* 放入序列特征表示 *F* 中*/

7) end for

8) return *F*

本文使用 F 表示应用函数的序列特征表示，$F=(V_1,V_2,\cdots,V_i,\cdots,V_t)$，其中，$V_i$ 表示应用函数 API 调用序列中 API 的特征表示，V_i=<*S_prior*, *S*, *S_post*>，S 表示函数中调用平台 API 的语句，*S_prior* 和 *S_post* 表示与 S 存在数据依赖关系的语句。

2)序列特征向量化表示

在获取应用函数的序列特征表示后，生成序列特征的向量化表示作为本文深度学习模型的输入。序列特征向量化表示主要包括以下几个步骤：首先，将函数序列特征进行符号化表示；然后，对符号化表示的特征进行抽象化字符串表示；最后，利用词向量嵌入模型对其进行向量化表示，如图 6 所示。接下来，具体描述这 3 个步骤的细节内容。

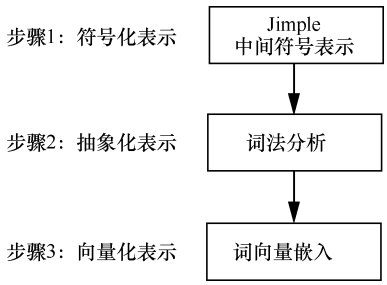


图 6　应用向量化表示步骤

在符号化表示中，将应用函数的序列特征用 Jimple 中间符号表示。Jimple[46]是 Java 字节码的一种抽象表现形式，相比于 Java 字节码和 Dalvik 字节码均超过 200 条指令，Jimple 只有 15 条指令，常用于简化程序的静态分析。在本文的特征抽取中，将 Dalvik 字节码转化为 Jimple 中间符号表示，然后对 Jimple 中间符号表示的应用函数进行静态的程序分析，抽取应用函数中 API 调用序列和数据依赖关系。本文选择 Jimple 作为序列特征的符号表示是为了简化应用的向量表示。

在抽象化表示中，通过词法分析将应用函数的 Jimple 中间符号表示转化成抽象字符串表示。在词法分析中，针对自定义的对象、变量以及函数的处理方式如下：①针对用户自定义的数据结构或对象，字符串抽象表示为常量 STR；②针对用户自定义的变量，字符串抽象表示为常量 VAR；③针对用户自定义的函数调用，字符串抽象表示为常量 FUNC。

在向量化表示中，将应用函数的字符串抽象表式利用词向量嵌入模型转化为向量形式，输入深度学习模型中进行训练与特征学习。下面介绍应用函数中 API 调用的一个抽象字符串描述简化案例。

VAR=staticinvoke <parseInt>(VAR);

上述案例中，该抽象字符串分割为“VAR”“=”“staticinvoke”“<”“parseInt”“>”“(”“VAR”“)”“;”这 10 个字符串形式。然后，使用 Word2Vec 词向量嵌入模型将分割的 10 个字符串生成相应的向量表示。

5.4 模型训练

本文使用基于 LSTM 的 Siamese 网络度量应用函数的相似性。在 Siamese LSTM 模型训练时，一对应用函数通过预处理、特征抽取及向量化表示等步骤后，将生成的特征向量及其对应的标签作为模型的输入。其中，标签是指一对特征向量的相似度 $\pi(F_1, F_2)\in[0, 1]$，$\pi(F_1, F_2)$越接近 1 表示 2 个特征向量越相似，$\pi(F_1, F_2)$越接近 0 则表示 2 个特征向量的差别越大，越不相似。

在应用函数的特征抽取和向量表示中，每个函数的特征向量长度不同，本文使用 MaLSTM 模型处理变长的序列特征数据，如图 7 所示。MaLSTM 是具有 LSTM 的 Siamese 网络，它由 2 个平行的 LSTM 网络构成[47]。Siamese 网络用于度量 2 个输入特征的相似性，MaLSTM 将 Siamese 中用于特征表示的网络替换成 LSTM 网络，使其适用于序列数据的学习，更加切合本文的研究问题。接下来，详细介绍 Siamese 网络和 MaLSTM 模型的工作原理。

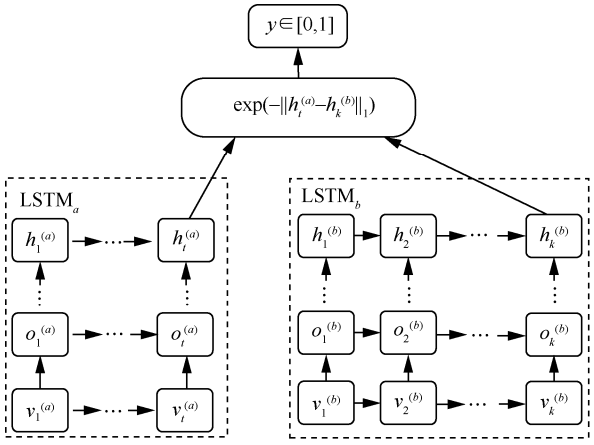


图 7 MaLSTM 结构

1) Siamese 网络

Siamese 网络[48]是一种应用广泛的特征相似性度量方法，它由 2 个结构相同的网络组成，这 2 个网络共享相同的权值 w，如图 8 所示。Siamese 网络常用于解决标签样本缺乏情况下的模型训练问题，它通过从数据中学习样本的相似性度量，然后与未知的标签样本进行比较。Siamese 网络使用一个函数将特征输入映射到目标空间，然后使用距离函数进行相似性对比。在图 8 中，N_a 和 N_b 为结构相同的网络，输入 V_1 和 V_2 分别被映射为 $N_a(V_1)$和 $N_b(V_2)$，V_1 和 V_2 相似性度量为 $E_w(N_a(V_1), N_b(V_2))$。在 Siamese 网络训练阶段，采用的优化策略是最小化相同类别样本的损失函数值和最大化不同类别样本的损失函数值。

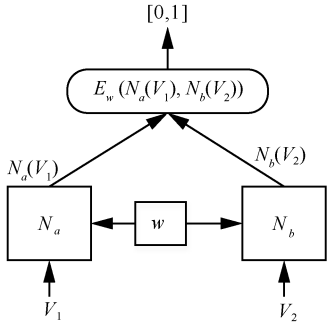


图 8 Siamese 网络结构

2) MaLSTM 模型

MaLSTM 是由 2 个结构相同的 LSTM 网络组成的 Siamese 网络，其中，LSTM$_a$ 和 LSTM$_b$ 的权重参数相同，如图 7 所示。MaLSTM 的相似性度量函数定义为一阶范数的指数函数，如式(2)所示。

$$g(h_{T_a}^{(a)}, h_{T_b}^{(b)}) = \exp(-\left\|h_{T_a}^{(a)} - h_{T_b}^{(b)}\right\|_1) \in [0,1] \quad (2)$$

其中，$h_{T_a}^{(a)}$ 和 $h_{T_b}^{(b)}$ 分别表示输入 LSTM$_a$ 和 LSTM$_b$ 网络中最后学习到的特征表示。LSTM$_a$ 和 LSTM$_b$ 网络的输入分别为 $F_1=\{v_1^{(a)},\cdots,v_{T_a}^{(a)}\}$ 和 $F_2=\{v_1^{(b)},\cdots,v_{T_b}^{(b)}\}$，其中，$T_a\neq T_b$。LSTM 网络中的记忆单元组成如图 9 所示。LSTM 记忆单元通过输入门 i_t、输出门 o_t、遗忘门 f_t 等控制信息的传递，最终输出的 h_t 由式(3)~式(8)计算得出。

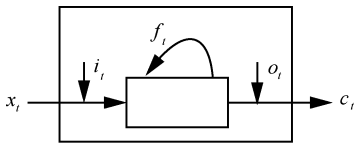


图 9 LSTM 记忆单元结构

$$i_t = \text{sigmoid}(W_i x_t + U_i h_{t-1} + b_i) \quad (3)$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad (4)$$

$$f_t = \text{sigmoid}(W_f x_t + U_f h_{t-1} + b_f) \quad (5)$$

$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1} \quad (6)$$

$$o_t = \text{sigmoid}(W_o x_t + U_o h_{t-1} + b_o) \quad (7)$$

$$h_t = o_t \odot \tanh(c_t) \quad (8)$$

其中，x_t 表示 t 时刻记忆单元的输入，W_i、W_f、W_c、W_o、U_i、U_f、U_c 和 U_o 表示权重系数，b_i、b_f、b_c 和 b_o 表示偏置项。本文 MaLSTM 模型的输出 $h_{T_a}^{(a)}$ 和 $h_{T_b}^{(b)}$ 也是通过式(3)~式(8)计算得到的。为了训练以

上的模型参数，本文的优化函数定义为

$$\min_{W,U,b}\sum_{i=1}^{|L|}(g(F_i,F_i')-1)^2+\min_{W,U,b}\sum_{j=1}^{|K|}g(G_j,G_j') \quad (9)$$

其中，F_i和F_i'表示一对相似应用函数，来自集合 L，L 中包含$|L|$个相似应用函数对；G_j和G_j'表示一对不同的应用函数，来自集合 K，K 中包含$|K|$个不同的应用函数对。集合 L 和 K 分别是一对相似应用 A 和 A'中相似应用函数和非代码重用函数的集合。

本文使用 BPTT 算法训练 MaLSTM 模型的 Siamese 网络，使用均方差（MSE, mean squared error）作为损失函数。通过输入大量的标记样本训练模型参数，将训练好的模型用于应用函数的相似性度量，通过应用函数的相似性比较判定是否为重打包应用。

5.5　重打包应用检测

本文方法的训练阶段是为了训练 Siamese LSTM 网络，用于度量应用函数的相似性。测试阶段是利用训练好的模型，通过度量应用函数的相似性，来判断一对应用是否为重打包应用。

本文方法 DeepRD 检测阶段（如图 4 所示）的步骤 D_1“应用向量化表示”与训练阶段（如图 3 所示）的前 3 个步骤类似，包括预处理、特征抽取和应用向量化表示。首先，将输入的 3 个应用进行预处理，反编译获得应用的程序代码并过滤第三方库；然后，抽取应用函数的程序特征表示；最后，将应用函数表示成序列特征形式，利用词向量嵌入模型进行特征向量化表示。检测阶段的步骤 D_2“检测”是利用已经训练好的 MaLSTM 模型度量应用函数的相似性，然后判断输入的一对应用是否为重打包应用。

本文使用应用函数作为分析与检测的基本单元，通过细粒度的代码相似性度量来判断应用的相似性，检测重打包应用以及识别被重用的代码片段。本文基于应用函数相似性度量的重打包应用检测中，需要考虑以下 2 个因素：①应用中相似函数的个数或比例；②相似应用函数的规模。应用 A_1 和应用 A_2 的相似性度量如式(10)~式(14)所示。

$$P=(o,r) \quad (10)$$

$$R=\{P_1,\cdots,P_n\} \quad (11)$$

$$\alpha=\frac{|R|}{\max(|A_1|,|A_2|)} \quad (12)$$

$$\beta=\frac{\max(|o_i|,|r_i|)}{\sum_{j=1}^{|R|}\max(|o_j|,|r_j|)},i=\{1,\cdots,|R|\} \quad (13)$$

$$sim(A_1,A_2)=\alpha\sum_{i=1}^{|R|}\beta sim(o_i,r_i) \quad (14)$$

式(10)中，o 和 r 表示应用函数，P 表示 o 和 r 为一对相似函数，P 中函数 o 和 r 的相似度 $sim(o, r)$ 大于阈值 0.87，通过大量的测试样本调节发现，阈值 0.87 是合理的。式(11)中，R 表示应用 A_1 和 A_2 的所有相似应用函数的集合，n 表示 R 中元素的个数。式(12)中，$|A_1|$和$|A_2|$分别表示应用 A_1 和 A_2 中函数的个数，$|R|$表示集合 R 中元素的数量，系数 α 表示应用 A_1 和 A_2 中相似应用函数所占的比例。式(13)中，$|o|$、$|r|$表示应用函数进行特征向量表示后的向量维度，系数 β 表示基于应用函数代码规模计算的权重系数。式(14)中，$sim(A_1, A_2)$表示应用 A_1 和 A_2 的相似性度量值，如果 $sim(A_1, A_2)$大于阈值 0.81，则判定应用 A_1 和 A_2 为一对重打包应用。通过大量的测试样本调节发现，阈值 0.81 是合理的。

6　实验与结果分析

6.1　方法实现

本文基于深度学习的 Android 重打包应用检测的实验部分主要包括以下 4 个部分。

1) Android 应用 APK 文件反编译。Android 平台的每个应用都有唯一的签名，恶意开发者在修改合法应用重新编译生成新应用发布时，其应用的签名信息一定会改变。本文使用 Keytool 获取应用的签名信息，作为唯一的标识区分应用。Apktool 用于反编译应用的 APK 文件，获取应用的程序代码。

2) 应用程序第三方库过滤。第三方库主要包括广告库和功能库这 2 种，本文使用文献[49]提供的 1 113 个功能库和 240 个广告库建立白名单，用于过滤应用中的第三方库。

3) 特征抽取及应用向量化表示。本文使用 soot 将 Dalvik 字节码转化为 Jimple 中间符号表示，然后通过控制流与数据流分析，获取应用函数的 API 调用序列及与 API 存在数据依赖关系的上下文信息。

4) MaLSTM 模型实现与训练。本文基于 TensorFlow 和 Keras 实现 Siamese LSTM 网络，用于度量应用函数的相似性。

6.2　实验环境与数据来源

本文实现了基于深度学习的 Android 重打包

应用检测原型系统 DeepRD，具体的实验环境为 CPU Intel(R) Core(TM) i7-6700K 4 GHz，64 GB 内存，2 块 GPU 显卡 NVIDA Titan X Pascal，每块显卡的显存为 12 GB，1 TB SSD，操作系统为 Ubuntu16.04。

本文的实验数据来自公开的实验数据集 AndroZoo[33]。该数据集中提供了大量重打包应用的标记样本数据，在本文模型的训练及预测阶段使用 AndroZoo 提供的 15 297 对重打包应用，其中，原始的正常合法应用共计 2 776 个，重打包应用共计 15 297 个。

实验数据集中 APK 文件的大小分布如图 10 所示。实验中 APK 文件最小为 50 KB，最大为 123.5 MB。超过 88%的 APK 文件大小在 20 MB 以内，超过 79%的 APK 文件大小超过 1 MB。实验样本数据总量达到 147 GB，应用函数的个数超过 40 万个。

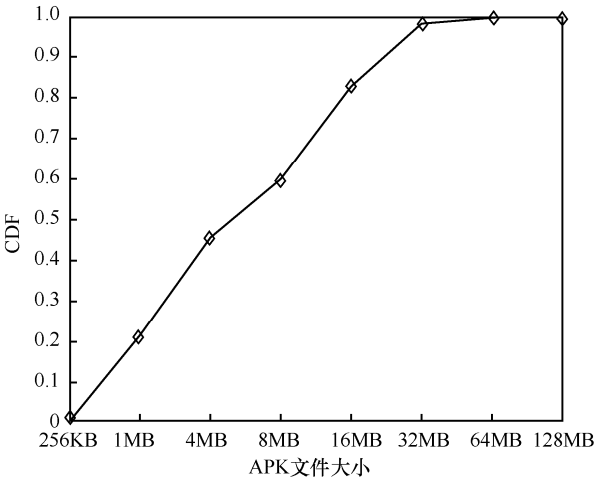


图 10 实验数据集中 APK 文件的大小分布

6.3 实验结果分析

本文从重打包应用检测的效果以及特征表示的有效性这 2 个方面评价本文方法。因此，本文实验评价中回答以下 2 个问题：①本文方法的精准率和漏报率分别是多少，是否优于现有方法？②本文提出的应用函数序列特征表示方法是否有效？

在实验效果评价中，本文使用漏报率 *FNR*、召回率 *recall*、精准率 *precision* 以及精准率和召回率的调和平均值 *F-score* 这 4 个指标来评价本文方法 DeepRD 的有效性，其中，*recall*= 1−*FNR*。本文 4 个评价指标的具体计算方法为

$$FNR = \frac{FN}{TP + FN} \tag{15}$$

$$recall = \frac{TP}{TP + FN} \tag{16}$$

$$precision = \frac{TP}{TP + FP} \tag{17}$$

$$F\text{-}score = \frac{2precision \cdot recall}{precision + recall} \tag{18}$$

其中，各参数的具体含义为：真阳性（TP, true positive）、假阳性（FP, false positive）、真阴性（TN, true negative）、假阴性（FN, false negative）。

在本文重打包应用检测中，漏报率越接近 0，精准率越接近 1，表明本文方法的检测效果越好。

1) 本文检测方法的精准率与漏报率分析

本文主要从检测的精准率和漏报率这 2 个方面来说明本文方法的有效性。在实验结果评价中，在公开数据集 AndroZoo 上将 DeepRD 与 SPRD[50]进行对比，分别从 *FNR*、*recall*、*precision* 和 *F-score* 这 4 项指标来评价 2 种方法的有效性。

SPRD 是一种用于检测第三方 Android 应用市场重打包应用的方法，该方法针对现有的检测方法在实际应用中存在难以有效均衡检测速度、精准率以及资源开销这 3 个方面的问题，提出了一种基于应用 UI 和程序依赖图的重打包应用快速检测方法。并利用重打包应用 UI 结构相同或相似的特点，设计了一种应用 UI 抽象表示的散列快速相似性检测方法，识别出可疑的重打包应用，然后将应用程序依赖图作为应用的特征表示，实现应用的细粒度相似性比较。但是，该方法并不适用于基于部分代码复用方式生成的重打包应用。

实验中，根据以往的经验，随机地选择了 70%的数据作为训练数据，30%的数据作为测试数据。本文模型训练时隐藏层数 *hidden_layer*=4。图 11 和图 12 分别表示漏报率及 *F-score* 与隐藏层数的关系。从图 11 和图 12 可以看出，当层数为 4 时，漏报率最低；当层数为 5 时，*F-score* 最高。其中，*F-score* 综合考虑了 *recall* 和 *precision*。本文目标是为了保持低漏报率，当隐藏层数设置为 4 和 5 时，*F-score* 只相差 1%左右。因此，本文中隐藏层数设置为 4 是合理的。

实验中，根据上述的模型参数设置，通过 10 折交叉验证得到本文方法 DeepRD 与 SPRD 在数据集 AndroZoo 的最终对比结果如表 1 所示。

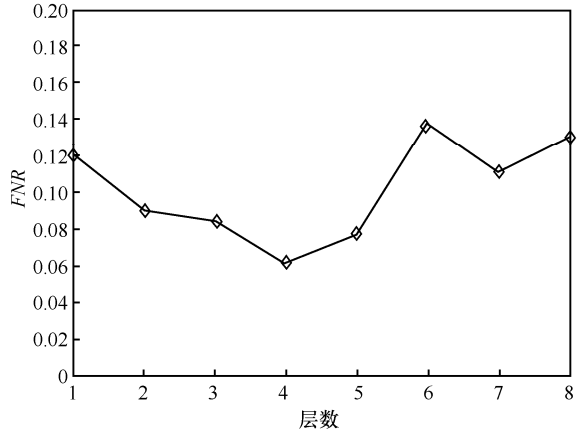


图 11　漏报率与隐藏层数关系

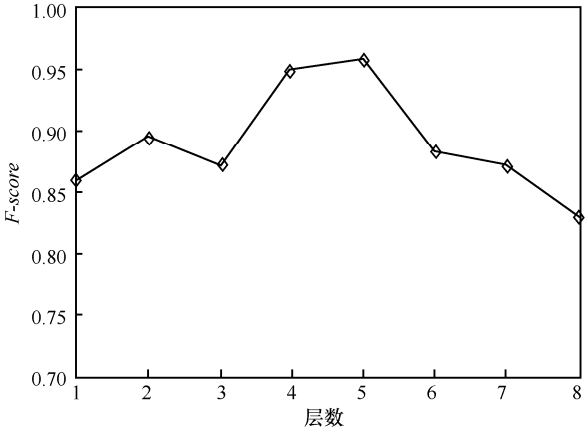


图 12　*F-score* 与隐藏层数关系

表 1　　**实验对比结果**

方法	*FNR*	*recall*	*precision*	*F-score*
DeepRD	6.2%	93.8%	95.7%	94.7%
SPRD	12.8%	87.2%	93.3%	90.1%

从表 1 可以看出，本文方法 DeepRD 在真实数据集上测试的漏报率低于 SPRD。实验结果显示，DeepRD 的检测精准率达到 95.7%，漏报率低于 6.2%，可以有效地应用于 Android 应用市场的重打包应用检测，还可以有效地应对现有重打包应用检测中漏报率高的问题。

实验中，通过人工分析 SPRD 的漏报样本发现，一些通过程序代码复用生成的重打包应用的 UI 不同，SPRD 方法无法有效识别，造成了漏报的现象。

2) 本文函数序列特征表示的有效性分析

本文抽取应用函数的序列特征，利用词向量模型对序列特征进行向量化表示，输入模型中进行训练和学习。最后，通过应用函数的相似性度量来检测重打包应用。本节主要分析本文特征提取方法对于降低漏报率和提高 *F-score* 是否有效。

在讨论函数序列特征表示的有效性问题时，本文将应用函数表示成任意的特征表示。在实验中，本文采取 2 种处理方式。第一种方式将应用函数当成一个完整的字符串来处理，不考虑任何的程序语义信息。Y 是应用函数的特征表示，$Y=(y_1,y_2,\cdots,y_i,\cdots,y_n)$，其中，$y_i$ 是将应用函数的完整字符串按照空格进行分割得到的元素，然后利用词向量嵌入模型对 Y 进行特征向量化表示，输入深度学习模型中进行训练和学习。第二种方式是仅抽取应用函数调用的 API 序列，没有获得 API 的上下文信息，向量化表示等与本文方法相同。

实验中基于相同的训练与测试数据集，根据以上 3 种特征表示对模型进行训练与参数学习。将 DeepRD 的序列语义特征表示与其他 2 种特征表示进行比较，实验结果如表 2 所示。

表 2　　**特征对比实验结果**

特征表示	*FNR*	*recall*	*precision*	*F-score*
语义序列特征	6.2%	93.8%	95.7%	94.7%
原始特征表示	21.1%	78.9%	98.3%	87.5%
API 调用特征	16.5%	83.5%	95.5%	89.1%

在表 2 中，语义序列特征是指本文提出的特征表示方法，原始特征表示是指将应用函数当成完整字符串生成的特征，API 调用特征是指获取应用函数的 API 调用形成的序列特征。从表 2 可以看出，如果对应用函数进行任意的特征表示，再利用深度学习模型进行程序特征学习，并不能取得比较好的检测效果。本文的特征表示方法在漏报率和 *F-score* 这 2 项指标上均优于其他 2 种特征表示方法。因此，本文的函数序列特征表示方法能够有效地应用于重打包应用检测中，相比于任意的程序特征表示方法，可以明显地降低漏报率和提高 *F-score*。

利用本文基于深度学习的重打包应用检测系统 DeepRD，分析目前国内外主流 Android 应用市场的重打包应用情况，特别是国内的应用市场。实验中，我们针对每一个应用市场随机地下载 1 万个应用进行检测，这 5 个应用市场的重打包应用分布情况如表 3 所示。

通过表 3 可以看出，Android 官方应用商店 Google Play 以及各大手机厂商的应用商店如小米、华为等，由于严格的审查机制，重打包应用的比例

少。但是，Android 应用市场上却存在大量的重打包应用，通过分析发现，在 Android 应用市场上，有超过 7.8%属于汉化破解类的重打包应用。

表 3 国内外主流应用市场重打包应用的比例

应用市场	重打包应用比例	国家
Google Play	4.71%	美国
百度	3.2%	中国
小米	5.3%	中国
华为	4.1%	中国
安智	12.6%	中国

6.4 相关讨论

本文方法能够有效地应用于 Android 平台的重打包应用检测，相比于传统的检测方法可以明显地降低漏报率，但依然存在一些不足或需要改进的地方。主要体现在以下 4 个方面。

1) 在本文方法的预处理阶段，使用白名单过滤第三方库在应用相似性比较中会带来干扰。但是由于收集的第三方库由文献[49]提供，因此会存在不完备的问题，在检测中会导致一些误报。针对第三方库的完备性问题，可以通过网络或其他渠道收集更多的第三方库来扩充白名单。

2) 本文中程序分析对象应用的是 Java 代码，没有分析由 C/C++语言编写的本地代码。因此，针对修改本地代码生成的重打包应用，本文方法无效。但是，目前尚未发现由修改本地代码生成的重打包应用样本。

3) 本文的模型训练阶段需要标记样本的支撑。然而，安全领域的标记样本是极其宝贵的资源。因此，在未来的工作中研究不依赖于标记样本的学习模型，或在只有少量标记样本的情况下，也能取得非常好的效果。

4) 在应用向量表示及深度学习模型实现方面，本文抽取应用函数的序列特征并进行向量化表示，使用 Siamese LSTM 网络学习应用程序的语义特征。在未来的工作中，研究其他深度学习模型在重打包应用检测方面的应用，例如深信念网络等。

7 结束语

针对目前 Android 重打包应用检测方法存在依赖于专家定义特征和漏报率高的问题，本文提出了一种基于深度学习的 Android 重打包应用检测方法。该方法以应用函数为基本的分析与检测单元，通过程序的控制流与数据流分析，获取应用函数的序列特征表示，利用词向量模型生成应用的向量表示，然后输入 Siamese LSTM 网络中自动地学习应用程序的语义特征，通过程序代码的相似性度量实现重打包应用的检测。实验结果表明，本文方法不依赖于专家定义应用的特征表示，并且能够有效地检测 Android 重打包应用，相比于传统的基于特征定义的检测方法，本文方法可以明显地降低检测的漏报率。

本文工作表明，深度学习可以有效地应用于 Android 重打包应用检测以及程序分析等相关领域。在未来工作中，准备将深度学习方法应用于其他类型的 Android 恶意软件检测中，如隐私泄露、权限滥用等，目前，这类恶意软件的检测依然依赖于复杂的静态分析或动态分析等方法。

参考文献：

[1] ZHOU W, ZHOU Y J, JIANG X X, et al. Detecting repackaged smartphone applications in third-party Android marketplaces[C]//The Second ACM Conference on Data and Application Security and Privacy. 2012: 317-326.

[2] 卿斯汉. Android 安全研究进展[J]. 软件学报, 2016, 27(1): 45-71.
QING S H. Research progress on Android security[J]. Journal of Software, 2016, 27(1): 45-71.

[3] 文伟平, 梅瑞, 宁戈, 等. Android 恶意软件检测技术分析和应用研究[J]. 通信学报, 2014, 35(8): 78-86.
WEN W P, MEI R, NING G, et al. Malware detection technology analysis and applied research of Android platform[J]. Journal on Communications, 2014, 35(8): 78-86.

[4] 张玉清, 王凯, 杨欢, 等. Android 安全综述[J]. 计算机研究与发展, 2014, 51(7): 1385-1396.
ZHANG Y Q, WANG K, YANG H, et al. Survey of Android OS security[J]. Journal of Computer Research and Development, 2014, 51(7): 1385-1396.

[5] 张玉清, 方喆君, 王凯, 等. Android 安全漏洞挖掘技术综述[J]. 计算机研究与发展, 2015, 52(10): 2167-2177.
ZHANG Y Q, FANG Z J, WANG K, et al. Survey of Android vulnerability detection[J]. Journal of Computer Research and Development, 2015, 52(10): 2167-2177.

[6] 杨威, 肖旭生, 李邓锋, 等. 移动应用安全解析学: 成果与挑战[J]. 信息安全学报, 2016, 1(2): 1-14.
YANG W, XIAO X S, LI D F, et al. Security analytics for mobile apps: achievements and challenges[J]. Journal of Cyber Security, 2016, 1(2): 1-14.

[7] 刘新宇, 翁健, 张悦, 等. 基于 APK 签名信息反馈的 Android 恶意应用检测[J]. 通信学报, 2017, 38(5): 190-198.
LIU X Y, WENG J, ZHANG Y, et al. Android malware detection based on APK signature information feedback[J]. Journal on Communications, 2017, 38(5): 190-198.

[8] 杨欢, 张玉清, 胡予濮, 等. 基于多类特征的 Android 应用恶意行为检测系统[J]. 计算机学报, 2014, 37(1): 15-27.
YANG H, ZHANG Y Q, HU Y P, et al. A malware behavior detection system of Android applications based on multi-class features[J]. Chinese Journal of Computers, 2014, 37(1): 15-27.

[9] SADEGHI A, BAGHERI H, GARCIA J, et al. A taxonomy and qualitative comparison of program analysis techniques for security assessment of Android software[J]. IEEE Transactions on Software Engineering, 2017, 43(6): 492-530.

[10] TIAN K, YAO D D, RYDER B G, et al. Detection of repackaged Android malware with code-heterogeneity features[J]. IEEE Transactions on Dependable and Secure Computing, 2017, PP(99): 1.

[11] FAN M, LIU J, WANG W, et al. DAPASA: detecting Android piggybacked apps through sensitive subgraph analysis[J]. IEEE Transactions on Information Forensics and Security, 2017, 12(8): 1772-1785.

[12] LI L, LI D, BISSYANDÉ T F, et al. Understanding Android app piggybacking: a systematic study of malicious code grafting[J]. IEEE Transactions on Information Forensics and Security, 2017, 12(6): 1269-1284.

[13] ZHOU W, ZHOU Y J, GRACE M, et al. Fast, scalable detection of piggybacked mobile applications[C]//The Third ACM Conference on Data and Application Security and Privacy. 2013: 185-196.

[14] LI L, LI D, BISSYANDÉ T F, et al. Automatically locating malicious packages in piggybacked Android apps[C]//The 4th International Conference on Mobile Software Engineering and Systems. 2017: 170-174.

[15] ANDERSON H S, KHARKAR A, FILAR B, et al. Evading machine learning malware detection[C]//Black Hat USA. 2017.

[16] 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, 2017, doi: 10.1109/TDSC.2017.2700270.

[17] YANG W, KONG D, 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.

[18] 刘剑, 苏璞睿, 杨珉, 等. 软件与网络安全研究综述[J].软件学报, 2018, 29(1): 42-68.
LIU J, SU P R, YANG M, et al. Software and cyber security-a survey[J]. Journal of Software, 2018, 29(1): 42-68.

[19] LECUN Y, BENGIO Y, HINTON G. Deep learning[J]. Nature, 2015, 521(7553): 436-444.

[20] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]//The IEEE Conference on Computer Vision and Pattern Recognition. 2016: 770-778.

[21] HINTON G, DENG L, YU D, et al. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups[J]. IEEE Signal Processing Magazine, 2012, 29(6): 82-97.

[22] PENNINGTON J, SOCHER R, MANNING C. Glove: global vectors for word representation[C]//The 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014: 1532-1543.

[23] LI Z, ZOU D Q, XU S H, et al. VulDeePecker: a deep learning-based system for vulnerability detection[C]//NDSS. 2018.

[24] YUAN Z, LU Y, WANG Z, et al. Droid-Sec: deep learning in Android malware detection[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(4): 371-372.

[25] WANG S, LIU T, TAN L. Automatically learning semantic features for defect prediction[C]//The 38th International Conference on Software Engineering. 2016: 297-308.

[26] PRADEL M, SEN K. Deep learning to find bugs. technical report[R]. TU Darmstadt, Department of Computer Science. 2017.

[27] SHIN E, SONG D, MOAZZEZI R. Recognizing functions in binaries with neural networks[C]//The 24th USENIX Conference on Security Symposium. 2015: 611-626.

[28] CHUA Z L, SHEN S, SAXENA P, et al. Neural nets can learn function type signatures from binaries[C]//The 26th USENIX Conference on Security Symposium. 2017: 99-116.

[29] XU X, LIU C, FENG Q, et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]//The 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 363-376.

[30] KOLOSNJAJI B, ZARRAS A, WEBSTER G, et al. Deep learning for classification of malware system call sequences[C]//Australasian Joint Conference on Artificial Intelligence. 2016: 137-149.

[31] DAVID O E, NETANYAHU N S. Deepsign: deep learning for automatic malware signature generation and classification[C]//2015 International Joint Conference on Neural Networks (IJCNN). 2015: 1-8.

[32] HOU S, SAAS A, CHEN L, et al. Deep4maldroid: a deep learning framework for Android malware detection based on linux kernel system call graphs[C]//Web Intelligence Workshops (WIW). 2016: 104-111.

[33] ALLIX K, BISSYANDE T F, KLEIN J, et al. AndroZoo: collecting millions of Android apps for the research community[C]//The 13th Working Conference on Mining Software Repositories. 2016: 468-471.

[34] CRUSSELL J, GIBLER C, CHEN H. Attack of the clones: detecting cloned applications on Android markets[C]//European Symposium on Research in Computer Security. 2012: 37-54.

[35] CRUSSELL J, GIBLER C, CHEN H. Andarwin: scalable detection of semantically similar Android applications[C]//European Symposium on Research in Computer Security. 2013: 182-199.

[36] CHEN K, LIU P, ZHANG Y. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets[C]//The 36th International Conference on Software Engineering. 2014: 175-186.

[37] CHEN K, WANG P, LEE Y, et al. Finding unknown malice in 10 seconds: mass vetting for new threats at the google-play scale[C]//The 24th USENIX Conference on Security Symposium. 2015: 659-674.

[38] WANG H Y, GUO Y, MA Z, et al. Wukong: a scalable and accurate two-phase approach to Android app clone detection[C]//The 2015 International Symposium on Software Testing and Analysis. 2015: 71-82.

[39] 王浩宇, 王仲禹, 郭耀, 等. 基于代码克隆检测技术的 Android 应用重打包检测[J]. 中国科学:信息科学, 2014, 44: 142-157.
WANG H Y, WANG Z Y, GUO Y, et al. Detecting repackaged Android applications based on code clone detection technique[J]. Science China Information Sciences, 2014, 44(1): 142-157.

[40] ZHANG F, HUANG H, ZHU S, et al. ViewDroid: towards obfuscation-resilient mobile application repackaging detection[C]//The 2014 ACM conference on Security and Privacy in Wireless & Mobile Networks. 2014: 25-36.

[41] SUN M, LI M, LUI J. Droideagle: seamless detection of visually

similar Android apps[C]//The 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks. 2015: 1-9.

[42] SHAO Y, LUO X, QIAN C, et al. Towards a scalable resource-driven approach for detecting repackaged Android applications[C]//The 30th Annual Computer Security Applications Conference. 2014: 56-65.

[43] SOH C, TAN H B K, ARNATOVICH Y L, et al. Detecting clones in Android applications through analyzing user interfaces[C]//The 2015 IEEE 23rd International Conference on Program Comprehension. 2015: 163-173.

[44] 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.

[45] AAFER Y, DU W, YIN H. Droidapiminer: mining api-level features for robust malware detection in Android[C]//International Conference on Security and Privacy in Communication Systems. 2013: 86-103.

[46] VALLEE-RAI R, HENDREN L J. Jimple: simplifying Java bytecode for analyses and transformations[R]. Technical Report, Sable Group, McGill University, Montreal, Canada, 1998.

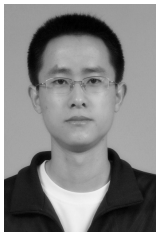
[47] MUELLER J, THYAGARAJAN A. Siamese recurrent architectures for learning sentence similarity[C]//AAAI. 2016: 2786-2792.

[48] BROMLEY J, GUYON I, LECUN Y, et al. Signature verification using a “siamese” time delay neural network[C]//Advances in Neural Information Processing Systems. 1994: 737-744.

[49] LI L, BISSYANDÉ T F, KLEIN J, et al. An investigation into the use of common libraries in Android apps[C]//The 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). 2016: 403-414.

[50] 汪润, 王丽娜, 唐奔宵, 等. SPRD:基于应用 UI 和程序依赖图的 Android 重打包应用快速检测方法[J]. 通信学报, 2018, 39(3): 159-171.
WANG R, WANG L N, TANG B X, et al. SPRD: fast application repackaging detection approach in Android based on application’s UI and program dependency graph[J]. Journal on Communications, 2018, 39(3): 159-171.

[作者简介]



汪润（1991−），男，安徽安庆人，武汉大学博士生，主要研究方向为 Android 安全与隐私、AI 安全等。

唐奔宵（1991−），男，湖北黄石人，武汉大学博士生，主要研究方向为移动安全与隐私、系统安全等。

王丽娜（1964−），女，辽宁营口人，博士，武汉大学教授、博士生导师，主要研究方向为网络安全、信息隐藏、AI 安全等。