软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(11):3404−3420 [doi: 10.13328/j.cnki.jos.006061]

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

基于代码属性图及注意力双向 LSTM 的漏洞挖掘方法*

段 旭[1,2], 吴敬征[1,3], 罗天悦[1], 杨牧天[1], 武延军[1,3]

[1](智能软件研究中心(中国科学院 软件研究所),北京 100190)

[2](中国科学院大学,北京 100049)

[3](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

通讯作者: 吴敬征, E-mail: jingzheng08@iscas.ac.cn

摘 要: 随着信息安全愈发严峻的趋势,软件漏洞已成为计算机安全的主要威胁之一.如何准确地挖掘程序中存在的漏洞,是信息安全领域的关键问题.然而,现有的静态漏洞挖掘方法在挖掘漏洞特征不明显的漏洞时准确率明显下降.一方面,基于规则的方法通过在目标源程序中匹配专家预先定义的漏洞模式挖掘漏洞,其预定义的漏洞模式较为刻板单一,无法覆盖到细节特征,导致其存在准确率低、误报率高等问题;另一方面,基于学习的方法无法充分地对程序源代码的特征信息进行建模,并且无法有效地捕捉关键特征信息,导致其在面对漏洞特征不明显的漏洞时,无法准确地进行挖掘.针对上述问题,提出了一种基于代码属性图及注意力双向 LSTM 的源码级漏洞挖掘方法.该方法首先将程序源代码转换为包含语义特征信息的代码属性图,并对其进行切片以剔除与敏感操作无关的冗余信息;其次,使用编码算法将代码属性图编码为特征张量;然后,利用大规模特征数据集训练基于双向 LSTM 和注意力机制的神经网络;最后,使用训练完毕的神经网络实现对目标程序中的漏洞进行挖掘.实验结果显示,在 SARD 缓冲区错误数据集、SARD 资源管理错误数据集及它们两个 C 语言程序构成的子集上,该方法的 *F*1 分数分别达到了 82.8%,77.4%,82.5%和 78.0%,与基于规则的静态挖掘工具 Flawfinder 和 RATS 以及基于学习的程序分析模型 TBCNN 相比,有显著的提高.

关键词: 漏洞挖掘;深度学习;静态分析;注意力机制;代码属性图

中图法分类号: TP311

中文引用格式: 段旭,吴敬征,罗天悦,杨牧天,武延军.基于代码属性图及注意力双向 LSTM 的漏洞挖掘方法.软件学报,2020,31(11):3404−3420. http://www.jos.org.cn/1000-9825/6061.htm

英文引用格式: Duan X, Wu JZ, Luo TY, Yang MT, Wu YJ. Vulnerability mining method based on code property graph and attention BiLSTM. Ruan Jian Xue Bao/Journal of Software, 2020,31(11):3404−3420 (in Chinese). http://www.jos.org.cn/1000-9825/6061.htm

Vulnerability Mining Method Based on Code Property Graph and Attention BiLSTM

DUAN Xu[1,2], WU Jing-Zheng[1,3], LUO Tian-Yue[1], YANG Mu-Tian[1], WU Yan-Jun[1,3]

[1](Intelligent Software Research Center (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

[3](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

* 基金项目: 国家重点研发计划(2018YFB0803600); 国家自然科学基金(61772507); 北京市科委产业技术创新战略联盟促进专项(Z181100000518032)

Foundation item: National Key Research and Development Program of China (2018YFB0803600); National Natural Science Foundation of China (61772507); Special Promotion of Industrial Technology Innovation Strategic Alliance of Beijing Municipal Science and Technology Commission (Z181100000518032)

收稿时间: 2019-07-08; 修改时间: 2019-11-30, 2020-04-11; 采用时间: 2020-04-26

Abstract: With the increasingly serious trend of information security, software vulnerability has become one of the main threats to computer security. How to accurately mine vulnerabilities in the program is a key issue in the field of information security. However, existing static vulnerability mining methods have low accuracy when mining vulnerabilities with unobvious vulnerability features. On the one hand, rule-based methods by matching expert-defined code vulnerability patterns in target programs. Its predefined vulnerability pattern is rigid and single, which is unable to cover detailed features and result in problems of low accuracy and high false positives. On the other hand, learning-based methods cannot adequately model the features of the source code and cannot effectively capture the key feature, which makes it fail to accurately mine vulnerabilities with unobvious vulnerability features. To solve this issue, a source code level vulnerability mining method based on code property graph and attention BiLSTM is proposed. It firstly transforms the program source code to code property graph which contains semantic features, and performs program slicing to remove redundant information that is not related to sensitive operations. Then, it encodes the code property graph into the feature tensor with encoding algorithm. After that, a neural network based on BiLSTM and attention mechanism is trained using large-scale feature datasets. Finally, the trained neural network model is used to mine the vulnerabilities in the target program. Experimental results show that the *F*1 scores of the method reach 82.8%, 77.4%, 82.5%, and 78.0% respectively on the SARD buffer error dataset, SARD resource management error dataset, and their two subsets composed of C programs, which is significantly higher than the rule-based static mining tools Flawfinder and RATS and the learning-based program analysis model TBCNN.

Key words: vulnerability mining; deep learning; static analysis; attention mechanism; code property graph

随着信息技术的发展,计算机软件已成为生活不可或缺的一部分,各行各业都在广泛地应用着计算机系统,包括医疗、教育、军事、政治和新零售等领域.在计算机系统的迅速发展和普及之下,软件漏洞已然成为威胁计算机系统安全的主要威胁之一.一旦漏洞被攻击者利用,就会导致严重的后果.据美国 MITRE 公司发布的数据,截止 2019 年 4 月 15 日,CVE(common vulnerabilities and exposures,公共漏洞和暴露数据库)中已存在 121 078 个条目,漏洞数量呈爆发式增长.探索并实现一种准确度高的漏洞挖掘方法,成为了一个亟待解决的问题.

然而,现有的静态漏洞挖掘方法在挖掘漏洞特征不明显的漏洞时准确率普遍下降.一方面,基于规则的方法通过在目标源程序中匹配专家预先定义的漏洞模式挖掘漏洞,其大量依赖专家知识,并且其预定义的漏洞模式较为刻板单一,无法覆盖到细节特征,导致其存在准确率低、误报率高等问题.例如,Flawfinder[1]和 RATS[2]是两个被广泛应用的基于规则的源代码安全性自动挖掘工具,它们使用词法分析的方式在目标程序汇中匹配该数据库中预先定义的漏洞模式,从而实现漏洞挖掘.然而,其预定义的漏洞模式难以对真实程序中复杂多变的漏洞进行有效地覆盖,导致其挖掘漏洞的误报率很高.另一方面,基于学习的方法无法充分地对程序源代码的特征信息进行建模,并且无法有效捕捉关键特征信息.例如,TBCNN[3]将程序源代码建模为抽象语法树,然后根据抽象语法树的结构和节点类型进行基于树的卷积操作,进行特征抽象,然后对程序进行分类.由于其仅使用抽象语法树对程序进行建模,且仅考察了节点类型信息,导致其无法有效地区分具有细微差异的漏洞代码和非漏洞代码.

针对上述问题,本文将代码属性图、注意力机制以及漏洞挖掘相结合,提出了一种基于代码属性图(code property graph,简称 CPG)及注意力双向 LSTM 的源码级漏洞挖掘方法.该方法首先将程序源代码生成代码属性图对代码的语义特征进行表示,并对其进行切片以剔除与敏感操作无关的冗余信息;其次,使用编码算法将代码属性图编码为特征张量;最后搭建基于双向 LSTM 和注意力机制的神经网络模型,使用特征张量对神经网络模型进行训练,并使用训练完毕的模型实现对目标程序中是否存在漏洞进行预测.本文针对语义特征建模难和关键特征捕获难的两点问题,利用代码属性图对程序源代码的语义特征进行抽象,设计编码算法将其编码为特征张量,有效地避免了语义信息的丢失.并且针对如图 1 中实线框所示的微小差异以及虚线框所示的冗余信息,分别引入深度学习中的注意力机制和程序切片,以捕获微小而关键的漏洞特征并剔除冗余信息.

为了验证本方法的有效性,本文分别选取 Flawfinder、RATS 作为基于规则的方法基线,在 SARD 缓冲区错误数据集和 SARD 资源管理错误数据集上进行实验.同时,选取 TBCNN 作为基于学习的方法基线,由于 TBCNN 所使用的静态分析工具仅支持对 C 语言源程序解析,因此在 SARD 缓冲区错误数据集和 SARD 资源管理错误数据集中 C 语言源程序构成的子集上进行实验.实验结果表明,本方法在上述 4 个数据集上的 *F*1 分数分别达到了 82.8%、77.4%、82.5%以及 78.0%,相比各基线提升 10%以上.此外,对于注意力机制的可视化实验,证明其可以有效地捕获到漏洞的关键特征,证明了本方法的有效性.

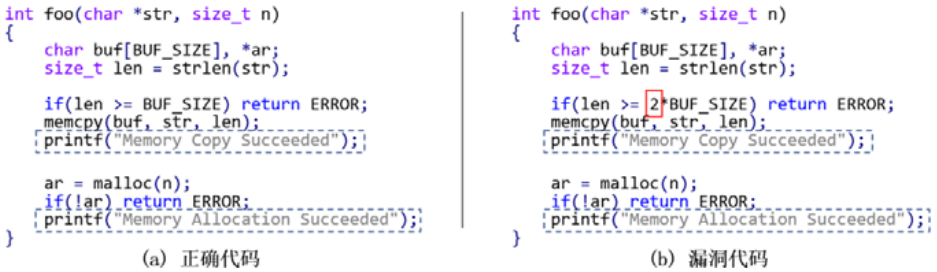

Fig.1 An example of the buffer error vulnerability.

图 1 缓冲区错误漏洞代码示例

本文的主要贡献总结如下.

(1) 提出了一种基于代码属性图对程序源代码进行建模的方法,其首先将程序表示为代码属性图,然后以敏感操作作为切片准则对其进行切片,最后基于编码规则将代码属性图编码为张量形式的数据,其可以有效地避免程序语义信息的丢失,并为后续的机器学习任务提供支持.

(2) 提出了一种面向源代码漏洞挖掘的神经网络模型,该模型利用双向 LSTM 感知源代码的上下文信息,并利用注意力机制捕获关键特征,在实现漏洞挖掘的同时,有效地避免了漏洞与非漏洞代码类间差异小所导致的准确率降低.

(3) 通过将本文所提出的方法与 Flawfinder、RATS 和 TBCNN 在 SARD 缓冲区错误数据集、SARD 资源管理错误数据集和由它们 C 语言程序构成的子集上进行对比实验,实验结果表明,本方法相比各对比工具均具有显著提升.

本文第 1 节对背景知识与相关技术进行介绍.第 2 节介绍本文所提出的基于代码属性图及注意力双向 LSTM 的漏洞挖掘方法的整体方案以及实现细节.第 3 节对本文中所进行的实验进行阐述.第 4 节对本文工作进行了总结,并对本方法的不足以及改进空间进行讨论.

1 相关技术概述

本节将对本文中所涉及的背景知识与相关技术进行介绍.主要包括源代码漏洞挖掘技术、程序抽象图结构表示以及注意力机制这 3 个方面.

1.1 源代码漏洞挖掘技术

近年来,随着信息化的不断发展,计算机软件对人们生活的影响也越来越大.如何有效地保证软件质量,已然成为学术界和工业界共同关注的重要问题.在此背景下,程序分析技术应运而生,程序分析指对计算机程序进行自动化的处理,从而确认或发现其某种特征,例如性能、正确性和安全性等[4].其中,漏洞挖掘就是对于程序的安全性进行分析,及时发现软件中存在的安全漏洞,从而提高软件的安全性.

漏洞(vulnerability)指在软件的需求、设计、实现、配置、运行等过程中被引入的代码缺陷,这些缺陷具有被攻击者利用的可能性,并且其存在将导致系统对某一特定类型的攻击具有一定的敏感性.漏洞的产生主要源于软件生命周期中的逻辑错误、编码缺陷等问题.例如,在系统设计阶段没有考虑到对用户输入数据或者用户权限进行检查,或者在开发阶段以错误的方式调用敏感的库函数.针对不同类型的漏洞,漏洞的起因、可利用性和影响性不尽相同,但其存在均会在不同程度上影响信息系统的安全性,并且被攻击者利用后,会导致受影响实体的机密性、完整性或可用性带来一定程度的损害.

面向源代码的漏洞挖掘是静态漏洞挖掘中的一个分支[5],其根据程序源代码等静态信息对程序中的漏洞进行挖掘,该类方法可以在程序开发的早期阶段快速地发现程序中存在的漏洞,进而避免后续进一步的损失.本文根据是否引入机器学习技术,将源代码漏洞挖掘方法分为基于规则的源代码漏洞挖掘方法和基于学习的源代码漏洞挖掘方法,在下文中将对两类源代码漏洞挖掘方法进行展开介绍.

1.1.1 基于规则的源代码漏洞挖掘技术

基于规则的源代码漏洞挖掘方法具有较为悠久的发展历史,在 20 世纪 70 年代,SteveJohnson 就开发了 Lint 工具[6],其通过规则检查对 C 语言代码中存在的错误进行挖掘.在 2000 年,出现了通过简单的词法分析对代码漏洞进行挖掘的工具 ITS4[7],但是其准确率较低.随后,Flawfinder[1]和 RATS[2]对词法分析方式进行了改进,它们对每种类漏洞维护内建的特征库,然后通过词法分析算法对其中条目进行匹配,从而挖掘代码中的漏洞,其可以有效地挖掘 API 误用[8]等问题导致的漏洞.在同期,也先后出现了例如 MOPS[9],BLAST[10]等基于逻辑推理的漏洞挖掘工具,它们通过使用状态机对程序的结构进行表示,然后利用安全属性对状态机进行遍历,从而挖掘漏洞.

1.1.2 基于学习的源代码漏洞挖掘技术

近年来,机器学习和深度学习技术在计算机视觉、自然语言处理和推荐系统等领域被广泛应用,并且取得了较好的成效.与此同时,由于机器学习和深度学习模型强大的特征学习和数据拟合能力,也逐渐被应用在漏洞挖掘中,直接或间接地为漏洞挖掘提供服务[5,11].Yamaguchi 等人[12]使用从已知漏洞中获取 API 使用模式,然后使用 TFIDF 和主成分分析等机器学习技术将 API 使用模式映射为向量,从而根据 API 使用模式之间的相似度来推测漏洞,即与具有漏洞的 API 使用模式相似度高的 API 使用模式很可能也具有漏洞.Feng 等人[13]提出 Genius,其对已知漏洞的控制流图进行无监督聚类生成码本,然后根据该码本对程序构造特征向量,通过比较目标程序和已知漏洞的特征向量的相似度,从而判定目标程序中是否存在漏洞.但是该方法由于涉及图匹配,效率较低,并且相似性判定依赖聚类生成的码本,难以对新任务进行适应.为此,Xu 等人[14]在 Genius 的基础上提出 Gemini,其通过 Structure2vec 算法将控制流图转化为数字向量,然后训练孪生神经网络,判定目标程序与已知漏洞是否相似,进而挖掘目标程序是否存在漏洞.该方法实现了比 Genius 更高的准确率和效率,并且可以通过迁移学习快速地适应新任务.Li 等人[15]提出了 VulDeePecker,其根据程序中的 API 调用对程序代码进行切片,然后将切片后的代码视作纯文本,使用词嵌入技术将其映射为向量,最后输入到双向 LSTM 中,判断被测程序是否具有漏洞.Duan 等人[16]提出了 VulSniper,其使用代码属性图对代码进行建模,然后使用基于规则的方法将其编码为特征张量,最后构建带有注意力机制的全连接网络,判断被测程序是否具有漏洞.该方法虽然能够在一定程度上检测细粒度漏洞,但是其网络结构较为简单,仅由全连接网络构成,特征学习能力有待提高.同时,由于其没有对程序进行切片,导致其难以对包含大量与敏感操作无关代码的程序进行处理.Grieco 等人[17]提出分别用静态分析和动态分析的方法对程序进行静态特征和动态特征的提取,然后综合动态特征和静态特征,使用词嵌入技术将其映射到含有语义信息的固定长度的向量中,并使用多层感知机进行训练和分类.Kim 等人[18]使用语法、语义和控制流对程序源代码进行建模,并使用基于注意力机制的双向 LSTM 推断漏洞代码模式,并将进行漏洞定位.Russell 等人[19]同时利用卷积神经网络和随机森林进行漏洞检测,其首先使用卷积和池化操作对程序源代码进行表示学习,然后将习得的向量表示输入到随机森林中进行分类.

1.2 程序抽象图结构表示

在程序分析任务中,将程序抽象成图结构是一种常见的处理方式,其通过使用图数据结构对程序的语义属性进行表示,从而实现更加有效地对程序进行分析.目前已有很多不同的图结构被提出,包括控制流图(control flow graph,简称 CFG)、数据流图(data flow graph,简称 DFG)、抽象语法树(abstract syntax tree,简称 AST)、程序依赖图(program dependence graph,简称 PDG)等.不同的图结构可以视为从程序的不同特征视角来描述程序,例如,AST 描述了代码的语法结构,CFG 描述了程序的执行路径和控制依赖,而 PDG 描述了程序内的控制依赖和数据依赖关系.

此外,由于程序的抽象图结构中包含充足的语义信息,利用抽象图结构进行程序分析是目前较为流行的趋势.例如,TBCNN[3]将程序源代码建模为抽象语法树,然后根据抽象语法树的结构和节点类型进行基于树的卷积和池化操作,实现对程序进行分类.Yu 等人[20]在抽象语法树的基础上加入细粒度的 Token 信息,并提出基于位置的 Token 向量嵌入方法,用于代码克隆检测.SecureSync[21]对程序生成抽象语法树和程序依赖图,并通过子图同构判定程序代码中是否存在漏洞.针对子图同构时间复杂度较大的问题.Li 等人[22]提出了 4 种对搜索范围剪枝的方法,有效地提高了搜索同构子图的效率.另外,有很多方法在已有图结构的基础上提出新的图结构,以适应

特定的任务.例如:Chang 等人[23]提出了增强程序依赖图(enhanced procedure dependence graph,简称 EPDG),其根据共享数据的依赖性,对 PDG 中的控制和数据依赖边进行增强,并通过启发式最大频繁子图挖掘算法从 EPDG 中挖掘规则,从而发现软件中由于条件被忽视而引起的软件缺陷;Yamaguchi 等人[24]提出了代码属性图(CPG),其是一种综合了 CFG、AST 和 PDG 的联合数据结构,被用来解决不同类型漏洞的关键特征在单一的图结构中无法充分体现的问题.例如,“缺少输入验证”类型漏洞在 CFG 中体现为缺少分支判断,但该特征在 PDG 中无法体现;“除零错误”在 AST 中体现为除法的分母元素为零,但该特征在 CFG 中无法体现.该方法通过对不同类型的漏洞定义漏洞模式,并根据该模式在代码属性图上执行图遍历,从而检查程序是否存在该类漏洞.由于其极大地包含了程序不同视角的信息,因此其对不同类型漏洞的挖掘均具有较好的适应性.

1.3 注意力机制

近年来,在深度学习领域中提出的注意力机制(attention mechanism)极为流行,其在面临一些困难任务时(例如细粒度图像分类),能有效地提高深度学习算法的准确率.注意力机制的灵感最初源自于人脑的注意力机制,当人脑在接收外部输入的信息时,往往不会对全部信息进行处理,而是将注意力集中在部分关键的信息上,过滤不重要的信息,从而提升信息处理的效率.目前,注意力机制在深度学习中的不同领域有着不同的实现,但总体上都可以概括成对注意力权重向量的学习.该权重向量的作用可以是反映了当前元素与其他元素的关联程度,也可以是反映了输入的每个元素在当前任务中的重要程度,其视具体的任务和注意力模型的实现方式而定[25].

在自然语言处理领域中,常使用注意力机制作为输出的目标序列和输入的源序列的对齐模型.如果将源序列中的元素看做键和值构成的二元组⟨*Key*,*Value*⟩,那么对给定目标序列中的某个元素 *Query*,可以根据 *Query* 和 *Key* 的相关性计算得到权重系数,权重系数越大,说明 *Key* 所对应的信息 *Value* 对 *Query* 的重要性越强,从而指导元素的对齐.在该模型中,如果使源序列与目标序列相同,就形成了自注意力(self attention),其可以反映序列中某个元素与其他元素之间的关联,从而有效地理解序列中存在的句法结构.Vaswani 等人[26]提出的 Transformer 模型是自注意力机制最具代表性的实现之一,其完全摒弃了先前机器翻译中广泛使用的 RNN 模型,而仅依赖注意力机制挖掘输入与输出之间的关系,从而避免了基于 RNN 的模型无法并行计算的问题,在保证准确率的同时,极大地提高了训练的效率.

在计算机视觉领域中,同样具有很多注意力机制的应用.注意力机制在该领域中的实现主要可以分为两类,分别为柔性注意力(soft attention)和刚性注意力(hard attention)[27].其中,

- 柔性注意力对整个输入计算注意力权重分布,该注意力权重反映了不同区域元素对当前任务的重要程度.该模型通常是可微的,可以通过网络端到端的训练获得.柔性注意力具有很多具体的实现[28−31],其可以有效地捕获图片中的关系特征信息,从而提高图片分类的准确率,或对图片进行更加细粒度的分类.例如,Wang 等人[30]提出的残差注意力网络(residual attention network)是柔性注意力的实现之一,其在神经网络的主分支外引入了一个注意力分支,在该分支中,通过自下而上和自上而下的网络结构对注意力权重进行学习,该权重反映图片中不同区域的重要程度,并指导神经网络对图片进行分类.
- 与柔性注意力不同的是,刚性注意力每次仅对输入的局部进行分析,其是一个随机预测过程,其重点在于预测注意力在不同区域的变化并强调其动态性.该模型通常是不可微的,需要通过强化学习进行训练.Mnih 等人[32]将注意力问题看作目标引导的序列决策过程,在每一个时间点,其智能体只能在局部区域进行信息提取,并不断以试错的方式进行学习.通过与环境进行交互获得到奖赏对行为进行指导,从而使注意力集中在具有更高奖赏的重要区域.

此外,由于图结构可以很好地表征数据之间的关联,在诸如材料分子工程、图像分析、软件工程等领域应用广泛,并且很多基于图结构的神经网络模型被提出[33].目前,很多研究将注意力机制应用在与图结构相关的任务中[34−37],试图在庞大而复杂的图结构中更加有效地挖掘信息,从而针对具体任务做出更好的决策[38].例如,Velickovic 等人[39]提出了图注意力网络(graph attention networks,简称 GAT),该网络模型由多个图注意力层组成,在每个图注意力层中,对图节点的邻接节点的注意力权重进行学习,并根据该权重对图节点的向量表示进行更新.该图注意力网络模型通过堆叠多个图注意力层实现对节点向量表示的迭代更新,得到节点的最终向量表

示,从而为后续的节点分类、图分类等任务提供支持.Ryu 等人[35]利用 GAT 模型学习节点的向量表示,并在模型的末端添加全连接层,实现对特征的整合,进而得到图的向量表示,并基于该向量表示实现对分子性质的预测,为材料分子工程和药物发现提供帮助.

2 基于代码属性图及注意力双向 LSTM 的漏洞挖掘方法

为了解决现有的静态漏洞挖掘方法无法有效地对代码特征进行建模,并且在挖掘漏洞特征不明显的漏洞时准确率普遍下降的问题,研究并实现了一种基于代码属性图及注意力双向 LSTM 的漏洞挖掘方法.如图 2 所示为本方法的整体框架示意图,本方法通过以下 4 个步骤对目标程序源代码进行漏洞挖掘.

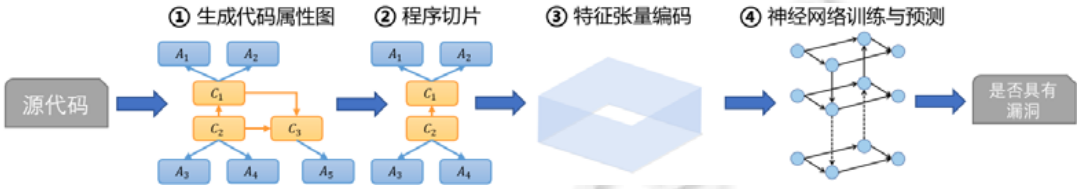

Fig.2 General framework of vulnerability mining method based on code property graph and attention BiLSTM

图 2 基于代码属性图及注意力双向 LSTM 的漏洞挖掘方法整体框架

2.1 代码属性图生成

程序源代码的语义信息在代码特征表示中尤为重要,直接决定了模型进行特征学习的难易程度.在程序分析任务中,代码的抽象图结构是对程序语义信息进行表示的一种有效方法,其将顺序的代码文本按照语义结构转换为图,从而直观地表示代码中不同元素之间的语义关系.代码属性图[24]是一种综合了抽象语法树、控制流图和程序依赖图的联合数据结构,其包含了代码的控制依赖、数据依赖以及语法结构等语义信息,是目前语义信息最为全面的抽象图结构之一,可以有效地表征多种类型的漏洞.本文利用代码属性图中的抽象语法树和控制流图构成的子结构对程序的语义进行表示,利用程序依赖图对程序进行切片.其中,使用抽象语法树和控制流图构成的子结构对程序进行表示的原因在于,在切片后程序依赖图信息已经被包含在切片后的代码属性图中,并且在后续的编码步骤中,程序依赖图中的依赖关系可以通过在编码时对前两者的组合间接表示.因此,本文使用该子结构对程序进行表示.为了更好理解,本文对图 1(b)中的代码生成代码属性图,并提取出抽象语法树和控制流图构成的子结构,其结果如图 3 所示,将其称为简化后的代码属性图.

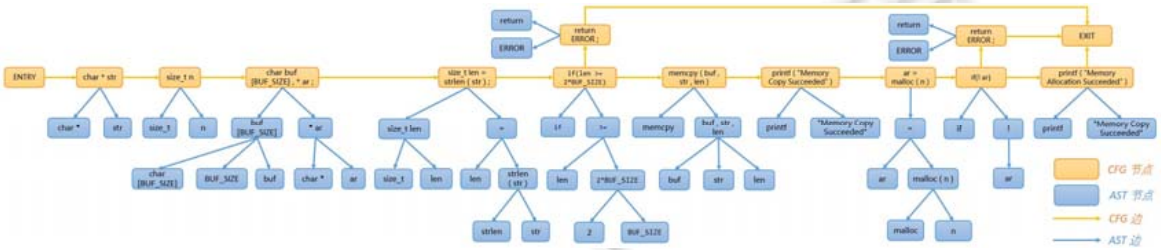

Fig.3 Simplified code property graph

图 3 简化后的代码属性图

本文利用开源的静态分析工具 Joern[40]对 C/C++源代码进行分析,并生成代码属性图,将其存储在 Neo4J 图数据库中,以便后续通过图中包含的信息进行查询,从而编码成为张量形式的数据.需要注意的是,Joern 使用模糊解析器对代码进行解析,即使无法提供有效地构建环境,或者源代码无法通过编译,也能够使用 Joern 进行解析,其大大地降低了待测试代码的门槛.相对地,其无法进行过程间分析,也无法对函数之间的调用关系进行建模;同时,其无法识别指针所指向的信息,而是直接将指针变量作为抽象语法树节点存储在图中.

2.2 程序切片

本文以函数为单位进行过程内分析(intra-procedural analysis),从而检测程序中存在的漏洞.在对函数作为输入之前,需要对其进行预处理,裁剪掉与敏感操作无关的语句.对函数语句进行裁剪的原因有两点.

- 第一,敏感操作是漏洞存在的必要条件,例如缓冲区错误漏洞的必要条件是存在对内存进行分配的操作,因此对于该类漏洞,内存分配语句就是其存在的必要条件;反之,与敏感语句不具有依赖关系的语句很大程度上是与漏洞无关的,这些语句会在代码属性图中引入冗余的节点,使真正有价值的特征信息变得稀疏,进而导致模型难以训练.
- 第二,神经网络模型要求输入的张量具有固定的形状,而真实环境下函数的代码行数是不可控的.如果函数的代码行数过多,那么在调整特征张量形状时会导致大量有价值的信息丢失,因此需要在特征张量编码之前对节点集进行裁剪,裁减掉与敏感操作不具有依赖关系的节点,可以很大程度上避免有价值的信息的丢失.

具体地,裁剪与敏感操作无关的语句的实现方式如下.

(1) 查找敏感语句

通过正则匹配等词法分析技术查找函数中的敏感操作语句,其在代码属性图中体现为控制流图节点.将该节点进行记录,并作为后续对程序进行切片的切片准则(slicing criterion).此外,由于敏感操作与漏洞具有密切关联,本方法会输出敏感操作所在的位置,辅助开发人员进行漏洞定位.

(2) 对函数代码进行静态切片

静态切片以程序的静态信息为依据,通过语句之间的数据和控制依赖对代码进行裁剪,从而保留与特定语句具有依赖关系的代码,并剔除无关代码.目前,程序静态切片方法可分为两类[41]:一类是基于数据流方程的程序切片技术,另一类是基于图可达性分析的程序切片技术.本文所使用的是基于程序依赖图进行图可达性分析的程序切片技术[42],其在程序依赖图上使用图可达性算法,计算可能影响切片准则处语句的语句和谓词,进而获得程序切片.由于代码属性图是抽象语法树、控制流图和程序依赖图组成的联合数据结构,因此本文根据代码属性图中的程序依赖图进行切片,在上一步获得与敏感操作语句对应的节点后,根据程序依赖图中的控制依赖边和数据依赖边,找到与该敏感语句节点具有依赖关系的节点集,并删除该节点集之外的节点,将剩余节点所构成的生成子图作为切片后的代码属性图.经过切片之后,图 1 虚线框中的冗余语句在图 3 中所对应的节点被剔除,进而降低无关语句对特征建模的负面影响.

2.3 特征张量编码

深度学习中的神经网络模型要求输入为固定形状的张量形式数据,因此需要实现一种对代码进行表示及编码的算法作为程序抽象图结构和神经网络之间的桥梁.为此,本文提出了代码属性图的特征张量的概念,并实现了一种基于代码属性图的特征张量编码技术,极大地保留了代码属性图中的语义信息.

2.3.1 特征张量设计

本文根据设计好的编码规则对函数的切片并简化后的代码属性图进行编码,生成三阶特征张量,以便输入到神经网络中进行处理.该张量与图的邻接矩阵类似,行和列均代表节点,但不同点在于其衍生出第三维,即使用一个固定长度的向量描述节点之间的关系,而不是邻接矩阵中使用 0 或 1 描述是否临接.在输出的张量中,前两维对应简化的代码属性图中的节点,第三维为描述节点之间关系的 144 维向量,其为对节点关系的编码.其中,代码属性图的特征张量的定义如下:

定义 1(特征张量). 设 $G=(V,E,\lambda,\mu)$为一个代码属性图,它包含 n 个节点 $V=\{v_1,v_2,\ldots,v_n\}$,则定义三阶张量 $T(G)=(t_{ijk})$称为 G 的特征张量,其中,$T(G)\in R^{n\times n\times 144}$,且满足:

$$t_{ijk}=\begin{cases}1, & v_i和v_j的关系具有k所描述的特征\\ 0, & v_i和v_j的关系不具有k所描述的特征\end{cases} \tag{1}$$

为了更加便于描述 144 维向量中所描述的特征,将 144 维向量称为关系特征向量,并给出如下定义.

定义 2(关系特征张量). 设 $G=(V,E,\lambda,\mu)$为代码属性图,它有 n 个节点 $V=\{v_1,v_2,\ldots,v_n\}$,$T(G)=(t_{ijk})$为 G 的特征张量,则定义向量 $K_{ij}=(t_{ij1},t_{ij2},\ldots,t_{ij144})$为 v_i 和 v_j 的关系特征向量.

本文将关系特征向量 K_{ij} 拆分为 5 个字段,分别为 $TYPE_{var}(i)$,$TYPE_{var}(j)$,$OP(i,j)$,$TYPE_{ast}(i)$和 $TYPE_{cfg}(i)$,每个字段都对不同特征进行编码.据此,可以得到关系特征向量 K_{ij} 的另一种表达形式:

$$K_{ij}=\{TYPE_{var}(i),TYPE_{var}(j),OP(i,j),TYPE_{ast}(i),TYPE_{cfg}(i)\} \tag{2}$$

其中,{·}代表向量拼接.K_{ij} 的每个字段是一个长度为$|S|$的向量,其中,S 是该字段编码的目标特征的集合.由于不同字段对不同特征进行编码,因此不同字段的目标特征的集合 S 不同.

(1) $TYPE_{var}(i)$和 $TYPE_{var}(j)$

$TYPE_{var}(i)$和 $TYPE_{var}(j)$分别对 v_i 和 v_j 的数据类型进行编码,其均为长度为 19 的向量,并且目标特征集为 $S=\{short,int,long,char,float,double,bool,const,static,*,void,unsigned,signed,struct,union,enum,function,constant,else\}$.在 $TYPE_{var}(i)$字段中的 t_{ijk} 可表示成如下形式.

$$t_{ijk}=\begin{cases}1, & v_i\text{所述变量的数据类型包含}f(k)\\ 0, & v_i\text{所述变量的数据类型不包含}f(k)\end{cases} \tag{3}$$

其中,$0\leqslant k\leqslant 18$,$f(k)$为 K_{ij} 中索引为 k 所对应的特征,$f(k)\in S$.

同理,可将在 $TYPE_{var}(j)$字段中的 t_{ijk} 表示成如下形式.

$$t_{ijk}=\begin{cases}1, & v_j\text{所述变量的数据类型包含}f(k)\\ 0, & v_j\text{所述变量的数据类型不包含}f(k)\end{cases} \tag{4}$$

其中,$19\leqslant k\leqslant 37$,$f(k)$为 K_{ij} 中索引为 k 所对应的特征,$f(k)\in S$.

(2) $OP(i,j)$

$OP(i,j)$对 v_i 和 v_j 之间的运算符进行编码,其是长度为 29 的向量,并且目标特征集为 $S=\{+,-,*,/,\%,\hat{},=,|,\&,||,\&\&,<,>,<=,>=,==,!=,+=,-=,\square=,/=,\%=,\hat{}=,|=,\&=,<<,>>,<<=,>>=\}$.在 $OP(i,j)$字段中的 t_{ijk} 可以表示成如下形式.

$$t_{ijk}=\begin{cases}1, & v_i\text{和}v_j\text{所述表达式之间的运算符为}f(k)\\ 0, & v_i\text{和}v_j\text{所述表达式之间的运算符不为}f(k)\end{cases} \tag{5}$$

其中,$38\leqslant k\leqslant 66$,$f(k)$为 K_{ij} 中索引为 k 所对应的特征,$f(k)\in S$.

(3) $TYPE_{ast}(i)$

$TYPE_{ast}(i)$对 v_i 的父 *AST* 节点的所述表达式类型进行编码,其是长度为 57 的向量,其目标特征集 S 对应 CPG 中的 57 种 AST 节点的 *type* 属性值,例如 *CallExpression* 和 *ConditionExpression*.在 $TYPE_{ast}(i)$字段中的 t_{ijk} 可以表示成如下形式.

$$t_{ijk}=\begin{cases}1, & v_j\text{是}v_i\text{的父}AST\text{节点且}v_j\text{的}type\text{属性为}f(k)\\ 0, & v_j\text{不是}v_i\text{的父}AST\text{节点且}v_j\text{的}type\text{属性不为}f(k)\end{cases} \tag{6}$$

其中,$67\leqslant k\leqslant 123$,$f(k)$为 K_{ij} 中索引为 k 所对应的特征,$f(k)\in S$.

(4) $TYPE_{cfg}(i)$

$TYPE_{cfg}(i)$对 v_i 所在语句的 CFG 节点的所述语句类型进行编码,其是长度为 20 的向量,其目标特征集 S 对应 CPG 中的 20 种 CFG 节点的 *type* 属性值,例如 *ReturnStatement*.需要注意的是,在 CPG 中,CFG 节点是 AST 节点的子集,因此,此处 CFG 节点的 20 种 *type* 属性值同样是 AST 节点 57 种 *type* 属性值的子集.在 $TYPE_{cfg}(i)$字段中的 t_{ijk} 可以表示成如下形式.

$$t_{ijk}=\begin{cases}1, & v_j\text{是}v_i\text{所在语句的}CFG\text{节点且}v_j\text{的}type\text{属性为}f(k)\\ 0, & v_j\text{不是}v_i\text{所在语句的}CFG\text{节点且}v_j\text{的}type\text{属性不为}f(k)\end{cases} \tag{7}$$

其中,$124\leqslant k\leqslant 144$,$f(k)$为 K_{ij} 中索引为 k 所对应的特征,$f(k)\in S$.

图 4 为图 3 所示代码属性图切片后的特征张量.

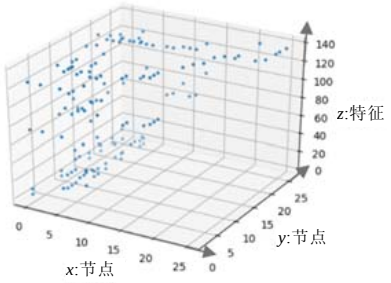

Fig.4 Feature tensor of code property graph

图 4 代码属性图的特征张量

2.3.2 特征张量形状校正

由于静态神经网络要求输入为固定形状的张量,但是由于代码长短不一,代码属性图节点数量不定,进而导致上述生成的特征张量形状不同.因此需要将特征张量进行处理,将其调整为统一的形状.由于特征张量中 z 轴代表的关系特征向量的长度是固定的,因此只需要对 x 轴和 y 轴的进行调整即可.具体方式为:预先为 x 轴和 y 轴设定一个固定长度,记为 *fixed_size*,然后将 x 轴和 y 轴长度均调整到 *fixed_size*.

当 x 轴和 y 轴长度小于 *fixed_size* 时,处理方案较为简单,即直接在其后端扩充零元素,直到达到 *fixed_size* 即可.当 x 轴和 y 轴长度大于 *fixed_size* 时,处理方案略为复杂,因为随意对 x 轴和 y 轴信息进行裁切,可能会导致重要信息的丢失.根据定义 1 可以发现,特征张量的 x 轴和 y 轴均对应 CPG 中的节点,因此,可以通过对 CPG 节点进行有选择的删减,从而缩小 x 轴和 y 轴长度.具体方式为:在 CPG 中找到敏感语句所在的 CFG 节点,然后计算其余节点到达该节点的最短路径长度,并删减与该节点最短路径长度较远的节点,直到 CPG 节点数量小于等于 *fixed_size*.这样做的依据是:在一定程度上可以认为在代码属性图中,距离敏感语句较远的语句与漏洞的关联和依赖较弱,其重要性较低,因此对其进行删减不会损失过多语义信息.

需要注意的是,在代码属性图上进行最短路径长度计算,时间开销较大,由于代码属性图节点和特征张量的 x 轴和 y 轴之间存在对应关系,因此可以近似地删减距离敏感语句索引较远的元素,这不失为一种可行的简化方法,其可以直接在特征张量上进行操作,在一定程度上提高形状调整的效率.

2.4 神经网络训练与预测

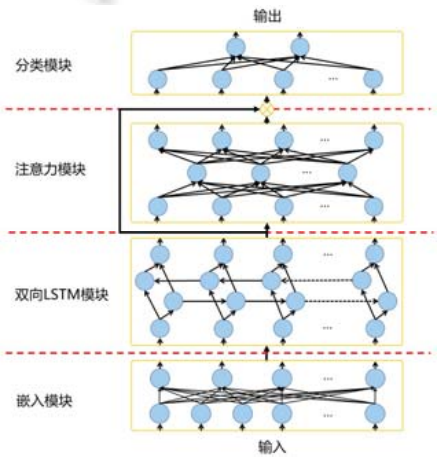

Fig.5 Structure of the neural network model

图 5 神经网络模型结构

本文采用深度学习的方法对神经网络模型进行训练,从而自动对漏洞代码模式进行学习;并使用训练完毕的神经网络模型对目标程序的特征张量进行分类,从而预测该程序是否具有漏洞.如图 5 所示,本文的神经网络模型由 4 个模块组成,分别为嵌入模块、双向 LSTM 模块、注意力模块和分类模块.

(1) 嵌入模块

从公式(3)~公式(7)可以发现,v_i 和 v_j 的关系特征张量 K_{ij} 在对 v_i 和 v_j 的关系进行编码的同时,会更加倾向于对节点 v_i 中的特征信息进行体现.因此,K_{i*} 在本质上可以看作是节点 v_i 的特征向量,其描述了 v_i 节点本身的特征,同时包含了 v_i 与其他所有节点之间的关系.

如图 6 所示,在嵌入模块中,首先将形状为(*fixed_size*,*fixed_size*,144)的输入中的 144 维向量通过线性变换映射到一个更低的维度,记为 1st_*ebd_dim*,此时,特征张量形状为(*fixed_size*,*fixed_size*,1st_*ebd_dim*);然后,将其形状调整为(*fixed_size*,*fixed_size*×1st_*ebd_dim*),并再次通过

两个全连接层对其降维,最终得到特征张量的形状为(*fixed_size*,2nd_*ebd_dim*)的张量.可以发现,最终得到的特征张量中的第一维与原始特征向量中的 *x* 轴相对应,其代表 CPG 中的节点.经过上述操作,将每个节点的特征都压缩到了长度为 2nd_*ebd_dim* 的向量中进行表示,以便后续的双向 LSTM 模块进行操作.在本文中,1st_*ebd_dim* 和 2nd_*ebd_dim* 分别被设置为 6 和 64.

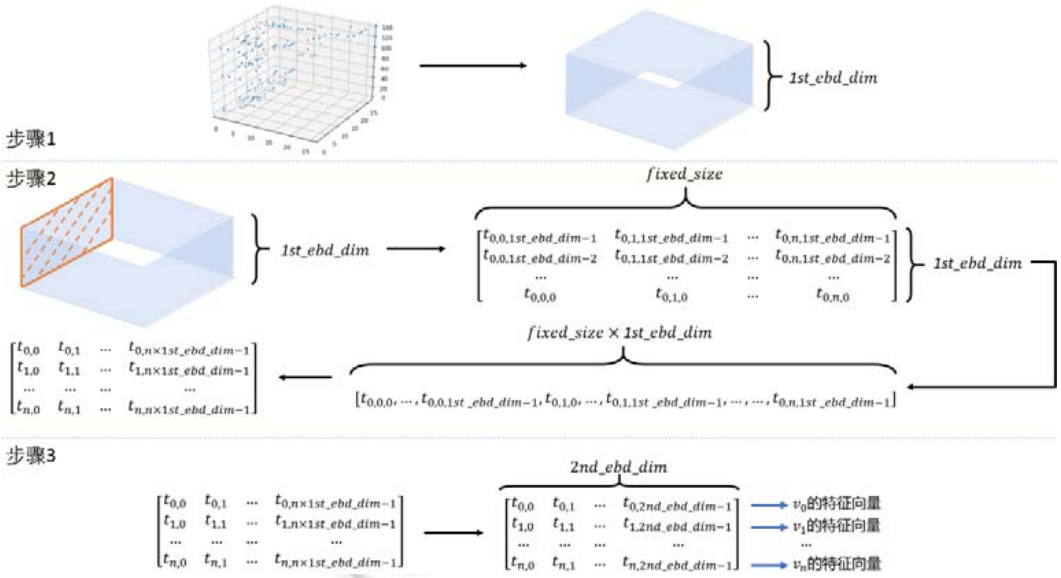

Fig.6 Feature tensor embedding process

图 6 特征张量嵌入过程

(2) 双向 LSTM 模块

虽然程序源代码的语义结构比自然语言更为复杂,但是其仍然存在一定的顺序性,其体现为代码中存在向前和向后的依赖.因此,本文采用双向 LSTM 对代码的上下文信息进行捕获.双向 LSTM 的结构由向前和向后的 LSTM 组合而成,其中,向前的 LSTM 可以根据前序信息影响后序信息,而向后的 LSTM 可以通过后序信息影响前序信息.该结构可以有效地处理存在双向上下文依赖的任务.

在本文中,使用节点的特征向量序列作为序列输入.对于节点特征向量在序列中的顺序,由于在编码时按照节点所述代码在文件中的位置将其编码进特征张量中,因此节点特征向量序列中节点的顺序由代码在源文件中的物理位置决定.为此,结合图 3 中的代码属性图进行阐述,假设用⟨·⟩代表节点,⟨2⟩在⟨*BUF_SIZE*⟩之前,因此通过前向的 LSTM 可以使网络得知“2”作用在 *BUF_SIZE* 上.同理,⟨*memcpy*⟩在⟨*buf*,*str*,*len*⟩之前,因此通过后向的 LSTM 可以使网络得知 *buf*、*str* 和 *len* 是 *memcpy* 的参数,进而影响 *memcpy* 调用.此外,双向 LSTM 还可以在一定程度上处理 AST 节点之间的嵌套关系,前向的 LSTM 可以使网络得知表达式由哪些元素组成,而后向的 LSTM 可以使网络得知元素组合成了什么表达式.在双向 LSTM 的末端,我们将前向 LSTM 和后向 LSTM 的输出结果拼接在一起,并输出给后续的网络结构进行处理.

(3) 注意力模块

为了处理漏洞挖掘类间差异小的问题,本文借鉴细粒度图片分类任务中的注意力模型.细粒度图片分类是指对图片进行更加细粒度的分类,例如对不同类别的鸟进行分类.该任务同样面临类间差异小的问题,其通常借助注意力机制加以解决.通过在漏洞挖掘问题中引入注意力模型,可以使神经网络将更多的注意力放在关键的代码语句及其依赖的变量上,例如敏感 API 调用及其依赖的变量等.通过在关键特征上赋予更多的注意力权重,使其对分类结果产生更重要的影响,从而指导神经网络根据关键的细微差异进行分类.

在本文中,通过对双向 LSTM 输出的不同节点特征向量所组成的特征张量赋予不同的注意力权重,从而将

不同的关注度作用在不同的代码属性图节点上.本文中的注意力模块结构如图 7 所示,其可以分为两个子分支,分别为掩模分支和短路分支,其具体作用如下.

1) 掩模分支:掩模分支实现对注意力权重的学习,其由自底向上和自顶向下的结构构成.其中:自底向上的结构使用多次一维卷积逐步获取高维特征,扩大感受野,从而捕获跨越不同节点特征向量的全局信息;自顶向下的结构使用多次一维反卷积,将尺寸缩小后的高维特征图恢复为与输入同样大小,从而使得注意力权重能够作用在输入上.在掩模分支的末端,使用 Sigmiod 函数对注意力权重矩阵进行归一化并对其加 1,防止梯度的消失[30].

2) 短路分支:短路分支将注意力模块的输入作为输出,没有经过任何额外处理.该分支的作用是保留注意力模块的原输入,从而将掩模模块输出的注意力权重以逐元素相乘的方式作用在注意力模块的输入上.假设注意力模块的输入为 $\boldsymbol{x}$,掩模分支的输出为 $A(\boldsymbol{x})$,短路分支的输出为 $S(\boldsymbol{x})$,则注意力模块的输出 $O(\boldsymbol{x})$可以用下式表示.

$$O(\boldsymbol{x})=(\mathbf{1}+A(\boldsymbol{x}))\odot S(\boldsymbol{x}) \tag{8}$$

其中,⊙代表逐元素相乘,$\mathbf{1}$ 代表元素全为 1 的矩阵.

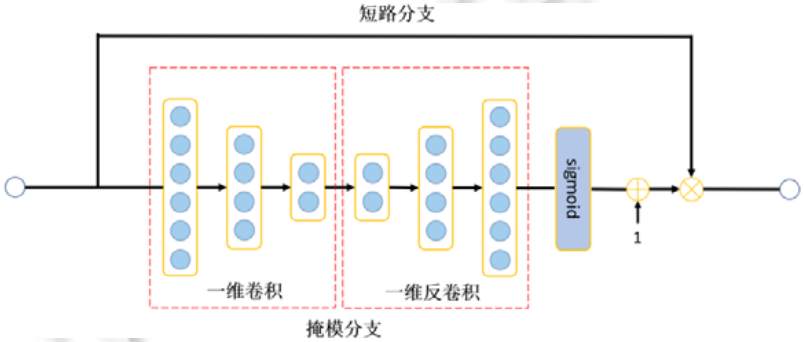

Fig.7 Structure of the attention module

图 7 注意力模块结构

(4) 分类模块

分类模块实现对特征信息的整合与最终的二分类,其输入为注意力模块输出的附加了注意力权重的特征张量,该模块由 3 个全连接层组成,实现对特征的整合.最后一层使用 *Softmax* 输出二分类的结果:输出为 0 代表目标程序不存在漏洞,或 1 代表目标程序存在漏洞.

3 实验评估

本节将对实验环节进行阐述.首先将对实验准备工作进行说明,包括实验环境、评价指标和实验数据集;其次,分别对本文中进行的实验内容进行阐述;最后对实验结果进行分析.

3.1 实验准备

(1) 实验环境

本文中涉及的实验在配有 Tesla P100 PCIE 12GB GPU,Intel Xeon Silver 4116 CPU@2.10GHz CPU 和 64GB 内存的设备上进行.

(2) 评价指标

误报率(false positive rate,简称 FPR)、漏报率(false negative rate,简称 FNR)、查全率(true positive rate,简称 TPR)、查准率(precision,简称 *P*)和 *F*1 分数(*F*1 score)是 5 个常见的评价模型检测准确程度的指标.假设 FP 为本身不存在漏洞但被判定为存在漏洞的样本数量,FN 为本身存在漏洞但被判定为不存在漏洞的样本数量,TP 为本身存在漏洞同时也被判定为存在漏洞的样本数量,TN 为本身不存在漏洞同时也被判定为不存在漏洞的样本

数量,则误报率(FPR)、漏报率(FNR)、查全率(TPR)、查准率(P)和 $F1$ 分数($F1$)的计算方法如下:

$$FPR=\frac{FP}{FP+TN} \tag{9}$$

$$FNR=\frac{FN}{TP+FN} \tag{10}$$

$$TPR=\frac{TP}{TP+FN} \tag{11}$$

$$P=\frac{TP}{TP+FP} \tag{12}$$

$$F1=\frac{2\times P\times TPR}{P+TPR} \tag{13}$$

从公式(9)可见,误报率表示所有不存在漏洞的样本中被错误判定为存在漏洞的样本的比例,误报率越低,则模型检测效果越理想.从公式(10)可见,漏报率表示所有存在漏洞的样本中被错误判定为不存在漏洞的样本的比例,漏报率越低,则模型检测效果越理想.从公式(11)可见,查全率表示所有存在漏洞的样本中被正确判定为存在漏洞的样本的比例,查全率越高,则模型检测效果越理想.从公式(12)可见,查准率表示所有被判定为存在漏洞的样本中,确实存在漏洞的样本的比例.从公式(13)可见,$F1$ 分数表示查全率和查准率的调和平均值,其综合考虑了查全率和查准率,$F1$ 分数越高,则模型检测效果越理想.

(3) 数据准备

软件保障参考数据集(software assurance reference dataset,简称 SARD)是由美国国家标准与技术研究院维护的漏洞数据集,该数据集中包含了不同代码形式(源代码或二进制代码)、不同语言(C、Java、Python 等)、不同漏洞类型(缓冲区错误、资源管理错误、注入等)的漏洞数据,每条数据以源文件或二进制文件的形式进行存储,供研究人员和软件安全保障工具开发人员对测试工具进行评估.在 SARD 的每个源文件中包含一个坏函数(bad function)和多个好函数(good function):坏函数中存在特定的漏洞;而好函数以不同方式修复了坏函数中存在的漏洞,其可以视作打完补丁的代码.由于好函数与坏函数的代码差异较少,因此该数据集可以很好地评估模型对类间差异小问题的适应能力以及对漏洞特征不明显的漏洞的检测能力.

由于本方法的目的是挖掘代码中是否存在漏洞,但同样具有漏洞的代码会根据漏洞类型的不同具有较大的差异,导致神经网络预测准确率下降,因此,本文选择在 SARD 中收集特定类型的漏洞数据.使用该特定类型漏洞数据独立地对神经网络进行训练,并实现对该类型的漏洞进行挖掘.对于不同类型的漏洞,则需使用相应类型的漏洞数据训练不同的神经网络,从而实现对不同类型漏洞的挖掘.具体地,本文在 SARD 中收集 C/C++语言的、源代码形式的缓冲区错误(buffer error)和资源管理错误(resource management error)类型的漏洞数据进行实验.这两类漏洞是 C\C++中较为常见的两类漏洞,并且在 SARD 中对于这两类漏洞有足够的数据以支持模型训练.本方法以函数为单位进行漏洞检测,因此对上述数据集中的源代码文件提取出源文件中的好函数和坏函数,将好函数作为负样本标注为“0”,代表该函数不存在漏洞;将坏函数作为正样本标注为“1”,代表该函数存在漏洞.以函数为单位构建“SARD 缓冲区错误数据集”和“SARD 资源管理错误数据集”.

3.2 实验结果与分析

首先,本文选取 Flawfinder 和 RATS 作为基于规则的静态漏洞挖掘方法基线,与本方法一起作为实验对象进行对比实验.Flawfinder 和 RATS 是两个基于规则的源代码漏洞挖掘工具,它们维护一个内建的 C/C++常见漏洞数据库,该数据库中记录漏洞代码模式,Flawfinder 和 RATS 通过词法分析在目标程序中对漏洞代码模式进行匹配,从而挖掘目标程序中的漏洞.Flawfinder 和 RATS 既可以对 C 语言源程序进行检测,也可以对 C++语言源程序进行检测,因此对于它们的实验,在完整的 SARD 缓冲区错误数据集和 SARD 资源管理错误数据集上进行.其中,SARD 缓冲区错误数据集和 SARD 资源管理错误数据集的源文件数量以及编码后的坏函数和好函数数量见表 1.由于静态分析工具 Joern 无法对其中一些源文件进行正确解析,导致编码后的函数数量略微少于数据集中的源文件数量.需要注意的是,由于 Flawfinder 和 RATS 能够直接通过对程序源代码进行扫描从而检测漏洞,其

不依赖 Joern 对图结构的解析,因此本文直接使用它们在源文件上进行实验.此外,本文对第 2.2 节中所述的敏感语句切片裁剪掉的节点数量进行了统计.对于 SARD 缓冲区错误数据集,切片后比切片前的控制流图节点数量平局减少 11.0 个;对于 SARD 资源管理错误数据集,切片后比切片前的控制流图节点数量平局减少 6.30 个.

Table 1 Amount of data in the dataset

表 1 数据集中的数据量

数据集	源文件数量	坏函数数量	好函数数量
SARD缓冲区错误	7 273	7 232	10 612
SARD资源管理错误	4 124	4 033	8 349

为对神经网络进行训练并评价预测效果,本文以 6:2:2 的比例分别将 SARD 缓冲区错误数据集和 SARD 资源管理错误数据集划分为训练集、验证集和测试集,使用训练集对神经网络模型进行训练,并使用测试集对训练完毕的模型的预测效果进行测试.表 2 为 Flawfinder、RATS 和本方法在 SARD 缓冲区错误数据集和 SARD 资源管理错误数据集的测试集上的对比实验结果.从对比实验数据中可见,本方法在 SARD 缓冲区错误数据集和 SARD 资源管理错误数据集上的表现明显优于 Flawfinder 和 RATS,其在两个数据集上的 *F*1 分数分别达到了 82.8%和 77.4%,并且漏报率明显低于另外两个对比工具.

Table 2 Result of comparative experiment with Flawfinder and RATS

表 2 与 Flawfinder 和 RATS 进行对比实验的结果

数据集	工具或方法	误报率(%)	漏报率(%)	查全率(%)	查准率(%)	*F*1分数(%)
SARD缓冲区错误	Flawfinder	56.6	44.8	55.2	39.9	46.3
	RATS	68.7	31.3	68.7	40.5	51.0
	本方法	14.3	14.6	85.4	80.4	82.8
SARD资源管理错误	Flawfinder	40.7	58.4	41.6	34.1	37.4
	RATS	33.9	63.8	36.2	35.0	35.6
	本方法	14.1	18.5	81.5	73.7	77.4

本文基于实验数据进行了分析,Flawfinder 和 RATS 通过专家定义代码漏洞模式,并通过词法分析在目标程序代码中对漏洞代码模式进行匹配从而挖掘漏洞,但预定义的模式较为单一,其难以覆盖复杂多变的真实环境,导致挖掘的准确率降低.例如,*memcpy* 是 C/C++中一个较为常见的敏感库函数,其实现从源内存地址的起始位置开始拷贝若干个字节到目标内存地址中,该函数具有 3 个参数,分别为目标地址、源地址和拷贝数据长度.在 Flawfinder 中,对 *memcpy* 调用是否构成漏洞的逻辑判断较为简单,当 *memcpy* 的参数少于 3 个、目标地址中多次使用&运算符取地址或者拷贝数据长度未使用 sizeof 获取时,Flawfinder 将判定该 *memcpy* 调用构成漏洞.可以发现,Flawfinder 仅通过正则表达式对库函数的参数进行检查,未考察程序中的数据依赖或控制依赖等语义信息,如果拷贝数据长度虽未使用 sizeof 获取,但在上下文进行了无害检查,那么其将不构成漏洞.上述原因直接导致了 Flawfinder 和 RATS 等基于规则的源代码漏洞挖掘工具的准确率降低.

反观本方法,在利用代码属性图考察程序语义信息的同时,构建了更为有效的神经网络模型,利用双向 LSTM 捕获代码中向前和向后的依赖,并引入注意力机制解决漏洞和非漏洞代码类间差异小的问题.基于上述原因,本方法成功地提高了源代码漏洞挖掘的准确率,证明了其有效性.

其次,本文选取 TBCNN 作为基于学习的程序分析方法基线,与本方法一起作为实验对象进行对比实验.TBCNN 是一种基于深度学习的程序分析模型,其将程序源代码建模为抽象语法树,然后根据抽象语法树的结构和节点类型进行基于树的卷积和池化操作,从而进行特征抽象,实现对程序进行分类.

由于 TBCNN 中用于解析源代码构造抽象语法树的工具 PyCParser 仅支持 C 语言源程序,因此对于 TBCNN 的实验在由 SARD 缓冲区错误数据集和 SARD 资源管理错误数据集中 C 语言源程序构成的子集上进行.其子集的源文件数量以及编码后的坏函数和好函数数量见表 3.由于静态分析工具 Joern 无法对其中一些源文件进行正确解析,导致编码后的函数数量略微少于数据集中的源文件数量.

Table 3 Amount of data in the sub-dataset composed of C programs

表 3 由 C 语言程序构成的子数据集中的数据量

数据集	源文件数量	坏函数数量	好函数数量
SARD缓冲区错误C语言源程序子集	4 264	4 242	6 669
SARD资源管理错误C语言源程序子集	3 116	3 033	6 550

同样地,本文以 6:2:2 的比例分别将子数据集划分为训练集、验证集和测试集.表 4 为 TBCNN 和本方法在 SARD 缓冲区错误数据集和 SARD 资源管理错误数据集中,由 C 语言源程序构成的子集的测试集上进行对比实验的结果.从对比实验数据中可见,本方法在两个数据集上的表现均明显优于 TBCNN,其在两个数据集上的 *F*1 分数分别达到了 82.5%和 78.0%.

Table 4 Result of comparative experiment with TBCNN

表 4 与 TBCNN 进行对比实验的结果

数据集	工具或方法	误报率(%)	漏报率(%)	查全率(%)	查准率(%)	*F*1分数(%)
SARD缓冲区错误C语言源程序子集	TBCNN	25.2	41.4	58.6	60.9	59.7
	本方法	7.06	22.3	77.7	88.1	82.5
SARD资源管理错误C语言源程序子集	TBCNN	3.02	55.5	44.5	87.7	59.0
	本方法	12.8	19.1	80.9	75.4	78.0

本文针对 TBCNN 的实验进行了分析.TBCNN 使用抽象语法树对程序进行建模,但是这种方式仅考察了程序中的语法结构,没有将控制流、数据流等对于漏洞密切相关的因素进行考量.此外,TBCNN 中仅将抽象语法树的节点类型作为节点的特征信息,其不足以涵盖程序中细粒度的信息,导致其在面对类间差异小的分类任务时表现不佳,例如对仅有细微差别的漏洞代码与非漏洞代码进行分类.

反观本方法,其使用代码属性图对程序进行建模,其中同时包括了程序的语法结构、控制流和数据流这 3 种与漏洞密切相关的信息;并且本方法利用精心设计的编码方法,在将代码属性图装换为特征张量时,极大地保留了代码属性图中的信息,有利于分类器进行学习.此外,本方法中引入的注意力机制有效地捕获了特征张量中的关键信息,使其在面对类间差异小的分类问题时仍具有良好的表现.

最后,为研究注意力机制在本方法中的作用,证明注意力机制的有效性,本文将图 1(b)中程序的特征张量输入到在 SARD 缓冲区错误数据集上训练完毕的神经网络中,并获得了注意力层输出的注意力权重分布,将每个节点的注意力向量取均值后得到标量并归一化,从而直观地表示该节点上的注意力,最终使用热力图对其进行可视化的结果如图 8 所示,颜色越亮注意力权重越高.

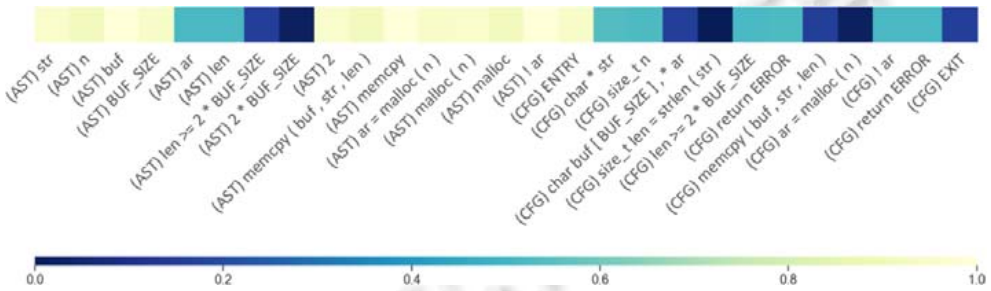

Fig.8 Visualization of attention weights distribution

图 8 注意力权重分布可视化

该程序调用 *memcpy* 库函数对内存数据进行拷贝,如果目标地址空间不足以接收指定长度的数据,则会发生缓冲区错误.该漏洞可以被攻击者利用,使得返回地址指向预先设定的恶意代码处,从而实现对恶意代码的执行.该漏洞是由 *memcpy* 敏感库函数的调用以及长度判断语句中额外的“2”所导致的,从图 8 中可以发现,注意力主要分布在 *memcpy* 和 *malloc* 等敏感库函数附近,并且在 *buf*、*BUF_SIZE* 和“2”处具有较高的注意力权重.由于

双向 LSTM 能够充分感知到节点周围上下文,因此注意力权重高的节点呈现出一定的聚集性,体现为图中连续多个节点均具有较高的注意力权重.此外,CFG 节点的注意力权重普遍比 AST 节点低的原因在于,本文中所使用的数据集中多数数据的漏洞成因与控制流不相关,而更多地体现在 AST 节点所代表的细粒度特征中,其使得网络更加关注 AST 节点中的细粒度特征,以准确识别特征不明显的漏洞.由于上述注意力权重分布体现了不同节点与漏洞的密切程度,因此其可被用以进一步指导开发人员进行漏洞修复.

4 结论与未来的工作

本文针对现有的静态漏洞挖掘方法在挖掘漏洞特征不明显的漏洞时准确率普遍下降的问题,研究并实现了一种基于代码属性图及注意力双向 LSTM 的漏洞挖掘方法.该方法将程序源代码表示成代码属性图,并按照独创的编码规则将代码属性图编码为特征张量,使用特征张量训练神经网络模型,自动地学习代码的漏洞模式,并利用训练完毕的模型预测目标程序中是否具有漏洞.本文在 SARD 缓冲区溢出数据集、SARD 资源管理错误数据集以及它们 C 语言程序构成的子集上进行了对比实验,其中,本方法在上述 4 个数据集上的 $F1$ 分数分别达到了 82.8%和 77.4%以及 82.5%和 78.0%,相比基线具有显著的提升,证明了本方法可以有效地提高了检测漏洞特征不明显的漏洞时的准确率.

本文所提出的基于代码属性图及注意力双向 LSTM 的漏洞挖掘方法能够有效地识别单个函数中是否存在漏洞,其仍具有一定的改进空间:第一,本方法以单个函数为单位进行漏洞检测,暂未考察调用的上下文,如果待检测函数本身没有漏洞,而其调用的函数中存在漏洞,则需对该被调用函数进行检测才能发现该漏洞;第二,本方法利用深度学习模型在分类问题上的优势,将漏洞挖掘问题视作一个二分类问题,从而有效地预测漏洞的存在性,但是其只能预测函数中是否具有漏洞,不论单个漏洞或多个漏洞,其都将报告为具有漏洞,因此对于函数中存在多个漏洞时的检测能力有待提高.未来会对针对以上两个问题进行研究.

References:

[1] Dwheeler. Flawfinder software official website. 2001. https://dwheeler.com/flawfinder/

[2] Secure software solutions. RATS software website. 2001. https://code.google.com/archive/p/rough-auditing-tool-for-security/

[3] Mou L, Li G, Zhang L, Wang T, Jin Z. Convolutional neural networks over tree structures for programming language processing. In: Proc. of the 30th AAAI Conf. on Artificial Intelligence. 2016. 1287−1293.

[4] Zhang J, Zhang C, Xuan JF, Xiong YF, Wang QX, Liang B, Li L, Dou WS, Chen ZB, Chen LQ, Cai Y. Recent progress in program analysis. Ruan Jian Xue Bao/Journal of Software, 2019,30(1):80−109 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5651.htm [doi: 10.13328/j.cnki.jos.005651]

[5] Zou QC, Zhang T, Wu RP, Ma JX, Li MC, Chen C, Hou CY. From automation to intelligence: Survey of research on vulnerability discovery techniques. Journal of Tsinghua University (Science & Technology), 2018,58(12):1079−1094. (in Chinese with English abstract).

[6] Darwin IF. Checking C Programs with Lint. O'Reilly & Associates, Inc., 1986.

[7] Viega J, Bloch JT, Kohno Y, Mcgraw G. ITS4: A static vulnerability scanner for C and C++ code. In: Proc. of the 16th Annual Computer Security Applications Conf. (ACSAC 2000). 2000. 257−267.

[8] Li Z, Wu JZ, Li MS. Study on key issues in API usage. Ruan Jian Xue Bao/Journal of Software, 2018,29(6):1716−1738 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5541.htm [doi: 10.13328/j.cnki.jos.005541]

[9] Chen H, Wagner D. MOPS: An infrastructure for examining security properties of software. In: Proc. of the 9th ACM Conf. on Computer and Communications Security. 2002. 235−244.

[10] Henzinger TA, Jhala R, Majumdar R, Sutre G. Software verification with BLAST. In: Proc. of the Int'l SPIN Workshop on Model Checking of Software. 2003. 235−239.

[11] Sun HY, He Y, Wang JC, Dong Y, Zhu LP, Wang H, Yang YQ. Application of artificial intelligence technology in the field of security vulnerability. Journal on Communications, 2018,39(8):1−17 (in Chinese with English abstract).

[12] Yamaguchi F, Lindner F, Rieck K. Vulnerability extrapolation: Assisted discovery of vulnerabilities using machine learning. In: Proc. of the 5th USENIX Conf. on Offensive Technologies. 2011. 118−127.

[13] Feng Q, Zhou R, Xu C, Cheng Y, Testa B, Yin H. Scalable graph-based bug search for firmware images. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. 2016. 480−491.

[14] Xu X, Liu C, Feng Q, Yin H, Song L, Song D. Neural network-based graph embedding for cross-platform binary code similarity detection. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security (CCS 2017). 2017. 363−376.

[15] Li Z, Zou D, Xu S, Ou X, Jin H, Wang S, Deng Z, Zhong Y. VulDeePecker: A deep learning-based system for vulnerability detection. In: Proc. of the 2018 Network and Distributed System Security Symp. 2018.

[16] Duan X, Wu J, Ji S, Rui Z, Luo T, Yang M, Wu Y. VulSniper: Focus your attention to shoot fine-grained vulnerabilities. In: Proc. of the 28th Int'l Joint Conf. on Artificial Intelligence (IJCAI 2019). 2019. 4665−4671.

[17] Grieco G, Grinblat GL, Uzal L, Rawat S, Feist J, Mounier L. Toward large-scale vulnerability discovery using machine learning. In: Proc. of the ACM Conf. on Data and Application Security and Privacy. 2016. 85−96.

[18] Kim J, Hubczenko D, Montague P. Towards attention based vulnerability discovery using source code representation. In: Proc. of the Int'l Conf. on Artificial Neural Networks. 2019. 731−746.

[19] Russell R, Kim L, Hamilton L, Lazovich T, Harer J, Ozdemir O, Ellingwood P, Mcconley M. Automated vulnerability detection in source code using deep representation learning. In: Proc. of the 17th IEEE Int'l Conf. on Machine Learning and Applications (ICMLA). 2018. 757−762.

[20] Yu H, Lam W, Chen L, Li G, Xie T, Wang Q. Neural detection of semantic code clones via tree-based convolution. In: Proc. of the 27th Int'l Conf. on Program Comprehension. 2019. 70−80.

[21] Pham NH, Nguyen TT, Nguyen HA, Nguyen TN. Detection of recurring software vulnerabilities. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2010. 447−456.

[22] Li J, Ernst MD. CBCD: Cloned buggy code detector. In: Proc. of the Int'l Conf. on Software Engineering. 2012. 310−320.

[23] Chang RY, Podgurski A, Yang J. Discovering neglected conditions in software by mining dependence graphs. Trans. on Software Engineering, 2008,34(5):579−596.

[24] Yamaguchi F, Golde N, Arp D, Rieck K. Modeling and discovering vulnerabilities with code property graphs. In: Proc. of the 2014 IEEE Symp. on Security and Privacy. 2014. 590−604.

[25] Chaudhari S, Polatkan G, Ramanath R, Mithal V. An attentive survey of attention models. arXiv preprint, arXiv: 1904.02874, 2019.

[26] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I. Attention is all you need. In: Proc. of the Advances in Neural Information Processing Systems. 2017. 5998−6008.

[27] Xu K, Ba JL, Kiros R, Cho K, Courville A, Salakhutdinov R, Zemel RS, Bengio Y. Show, attend and tell: Neural image caption generation with visual attention. In: Proc. of the 32nd Int'l Conf. on Machine Learning, Vol.37. 2015. 2048−2057.

[28] Xiao T, Xu Y, Yang K, Zhang J, Peng Y, Zhang Z. The application of two-level attention models in deep convolutional neural network for fine-grained image classification. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition. 2015. 842−850.

[29] Jaderberg M, Simonyan K, Zisserman A, Kavukcuoglu K. Spatial transformer networks. In: Proc. of the Advances in Neural Information Processing Systems. 2015. 2017−2025.

[30] Wang F, Jiang M, Qian C, Yang S, Li C, Zhang H, Wang X, Tang X. Residual attention network for image classification. In: Proc. of the Computer Vision and Pattern Recognition. 2017. 6450−6458.

[31] Zhao B, Wu X, Feng J, Peng Q, Yan S. Diversified visual attention networks for fine-grained object classification. IEEE Trans. on Multimedia, 2017,19(6):1245−1256.

[32] Mnih V, Heess N, Graves A. Recurrent models of visual attention. In: Proc. of the Advances in Neural Information Processing Systems. 2014. 2204−2212.

[33] Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. The graph neural network model. IEEE Trans. on Neural Networks, 2009,20(1): 61−80.

[34] Choi E, Bahadori MT, Song L, Stewart WF, Sun J. GRAM: Graph-based attention model for healthcare representation learning. In: Proc. of the 23rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2017. 787−795.

[35] Ryu S, Lim J, Hong SH, Kim WY. Deeply learning molecular structure-property relationships using attention-and gate-augmented graph convolutional network. arXiv preprint, arXiv: 1805.10988, 2018.

[36] Lee JB, Rossi R, Kong X. Graph classification using structural attention. In: Proc. of the 24th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining (KDD 2018). 2018. 1666−1674.

[37] Shang C, Liu Q, Chen KS, Sun J, Lu J, Yi J, Bi J. Edge attention-based multi-relational graph convolutional networks. arXiv preprint, arXiv: 1802.04944, 2018.

[38] Lee JB, Rossi RA, Kim S, Ahmed NK, Koh E. Attention models in graphs: A survey. ACM Trans. on Knowledge Discovery from Data, 2019,13(6):62:1−62:25.

[39] Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y. Graph attention networks. In: Proc. of the 6th Int'l Conf. on Learning Representations. 2017.

[40] Joern. 2014. https://joern.readthedocs.io/en/latest/

[41] Li BX, Zheng GL, Wang YF, Li XD. An approach to analyzing and understanding program—Program slicing. Journal of Computer Research and Development, 2000,37(3):284−291 (in Chinese with English abstract).

[42] Ottenstein KJ, Ottenstein LM. The program dependence graph in a software development environment. ACM SIGPLAN Notices, 1984,19(5):177−184.

附中文参考文献:

[4] 张健,张超,玄跻峰,熊英飞,王千祥,梁彬,李炼,窦文生,陈振邦,陈立前,蔡彦.程序分析研究进展.软件学报,2019,30(1):80−109. http://www.jos.org.cn/1000-9825/5651.htm [doi: 10.13328/j.cnki.jos.005651]

[5] 邹权臣,张涛,吴润浦,马金鑫,李美聪,陈晨,侯长玉.从自动化到智能化:软件漏洞挖掘技术进展.清华大学学报(自然科学版),2018,58(12):1079−1094.

[8] 李正,吴敬征,李明树.API 使用的关键问题研究.软件学报,2018,29(6):1716−1738. http://www.jos.org.cn/1000-9825/5541.htm [doi: 10.13328/j.cnki.jos.005541]

[11] 孙鸿宇,何远,王基策,董颖,朱立鹏,王鹤,张玉清.人工智能技术在安全漏洞领域的应用.通信学报,2018,39(8):1−17.

[41] 李必信,郑国梁,王云峰,李宣东.一种分析和理解程序的方法——程序切片.计算机研究与发展,2000,37(3):284−291.

段旭(1997−),男,硕士生,主要研究领域为漏洞挖掘,智能安全.

杨牧天(1990−),男,工程师,主要研究领域为开源软件安全,安全漏洞挖掘检测,人工智能安全.

吴敬征(1982−),男,博士,副研究员,主要研究领域为系统安全,漏洞挖掘,移动安全.

武延军(1979−),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为操作系统,机器学习系统软件,系统安全.

罗天悦(1990−),男,工程师,主要研究领域为操作系统安全分析,代码漏洞挖掘,人工智能安全.