软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006055]

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

基于机器学习的软件漏洞挖掘方法综述*

李 韵[1], 黄辰林[1], 王中锋[2], 袁 露[1], 王晓川[1]

[1](国防科技大学 计算机学院,湖南 长沙 410073)

[2](中国人民解放军 61302 部队,北京 100016)

通讯作者: 黄辰林, E-mail: clhuang@nudt.edu.cn

摘 要: 软件复杂性的增加给软件安全性带来极大的挑战.随着软件规模不断增大以及漏洞形态多样化,传统漏洞挖掘方法由于存在高误报率和高漏报率的问题,已无法满足复杂软件的安全性分析需求.近年来,随着人工智能产业的兴起,大量机器学习方法被尝试用于解决软件漏洞挖掘问题.首先,本文通过梳理基于机器学习的软件漏洞挖掘的现有研究工作,归纳了其技术特征与工作流程.接着,从其中核心的原始数据特征提取切入,以代码表征形式作为分类依据对现有研究工作进行分类阐述,并系统地进行了对比分析.最后依据对现有研究工作的整理总结,探讨了基于机器学习的软件漏洞挖掘领域面临的挑战,并展望了该领域的发展趋势.

关键词: 机器学习;漏洞挖掘;代码表征;软件质量;深度学习

中图法分类号: TP311

中文引用格式: 李韵,黄辰林,王中锋,袁露,王晓川.基于机器学习的软件漏洞挖掘方法综述.软件学报,2020. http://www.jos.org.cn/1000-9825/6055.htm

英文引用格式: Li Y, Huang CL, Wang ZF, Yuan L, Wang XC. Survey of software vulnerability mining methods based on machine learning. Ruan Jian Xue Bao/Journal of Software, 2020 (in Chinese). http://www.jos.org.cn/1000-9825/6055.htm

Survey of Software Vulnerability Mining methods Based on Machine Learning

LI Yun[1], HUANG Chen-Lin[1], WANG Zhong-Feng[2], YUAN Lu[1], WANG Xiao-Chuan[1]

[1](College of Computer, National University of Defense Technology, Changsha 410073, China)

[2](61302 PLA Troops, Beijing 100016, China)

Abstract: The increasing complexity of software application brings great challenges to software security. Due to the increase of software scale and diversity of vulnerability forms, the high false positives and false negatives of traditional vulnerability mining methods cannot meet the requirements of software security analysis. In recent years, with the rise of artificial intelligence industry, a large number of machine learning methods have been tried to solve the problem of software vulnerability mining. Firstly, the latest research results of applying machine learning method to the research of vulnerability mining are summarized in recent years, and the technical characteristics and workflow are proposed. Then, starting from the core original data features extraction, the existing research is classified according to the code representation form, and the existing research is systematically compared. Finally, based on the summary of the existing research, the challenges in the field of software vulnerability mining based on machine learning are discussed, and the development trends of this field are proposed.

Key words: machine learning; vulnerability mining; code representation; software quality; deep learning

* 基金项目: 国家重点研发计划(2018YFB0803501)
Foundation item: National Key Technologies Research and Development Program (China)(2018YFB0803501)
收稿时间: 2019-11-8; 修改时间: 2020-02-07, 2020-03-25; 采用时间: 2020-04-13; jos 在线出版时间: 2020-05-26

1 引言

计算机的迅速发展极大地便利了人们的生活,计算机软件渗透进生活的方方面面.以移动端软件为例,工信部发布的数据[1]显示,2018 年我国市场监测到的移动应用程序净增 42 万款,总量达到 449 万款,涵盖生活服务、日常工具及互联网金融等诸多类别.随着计算机软件的目标客户越来越广泛,需求也越来越庞杂,使软件的复杂性大幅增加.如 Linux 内核[2]2018 年共有 74 974 次提交,增加了 3 385 121 行代码,移除了 2 512 040 行代码,内核树的总代码行数为 26 132 637 行.与此同时,被披露的安全漏洞数量也呈现逐年增长的趋势.据安全数据库网站 CVE Details[3]发布的报告显示,2016 年共发布 6 447 个安全漏洞,2017 年共发布了 14 714 个安全漏洞,2018 年共发布了 16 556 个安全漏洞,而 2019 年截止到十月已发布了 12 174 个安全漏洞,其中 8.7%的漏洞 CVSS 得分超过 9.0.除了数量的增长,安全漏洞的形态也呈现复杂性和多样性,给计算机系统的正常安全运行造成极大的威胁.

目前对于软件漏洞的定义尚未形成广泛的共识,本文参照吴世忠等人[4,5]对软件漏洞的定义,即软件漏洞是指在软件系统或产品的软件生命周期中,由于操作实体有意或无意的疏忽而产生的设计错误、编码缺陷、运行故障,它们以不同形式存在于软件系统的各个层次与环节之中.一旦被恶意主体利用,比如获得更高级别权限、泄露软件中用户隐私数据等,将危害软件系统安全,并可能影响构建于软件系统之上的服务的正常运行.由于安全漏洞具有隐蔽性和可利用性,因此软件提供方通常需要在软件生命周期的各个阶段,利用各种技术手段尽早发现软件漏洞,降低漏洞的可利用性,从而降低软件系统被恶意损害的风险.

软件漏洞挖掘是检查并发现软件系统中存在安全漏洞的主要手段之一,通过利用各种工具对软件的代码进行审计,或者分析软件的执行过程来查找软件的设计错误、编码缺陷、运行故障.早期漏洞挖掘技术依据是否依赖程序运行划分为静态分析方法与动态分析方法,其中静态分析方法一般应用于软件的开发编码阶段,无需运行软件,通过扫描源代码分析词法、语法、控制流和数据流等信息来发现漏洞.动态分析方法则一般应用于软件的测试运行阶段,在软件程序运行过程中,通过分析动态调试器中程序的状态、执行路径等信息来发现漏洞.典型漏洞挖掘工具如 Coverity[6]、Klocwork[7]以及 Cobot[8]等通过对程序源代码进行静态分析来检测系统漏洞;KLEE[9]、S2E[10]、Mayhem[11]等采用符号执行这种动态分析方法进行漏洞挖掘;libFuzzer[12]、Radamsa[13]以及 AFL[14]等采用模糊测试这种动态分析方法排查错误.同时也有相关研究尝试通过动静态相结合的分析方法来提高检测效率和准确率,如 angr[15]集成了静态分析和动态符号执行,能够实现自动化分析二进制文件;SAGE[16]结合使用了模糊测试和符号执行,并将动态符号执行应用在 x86 架构的程序分析中.近年来,为了提高漏洞挖掘的效率,也出现了一些自动化或半自动化的漏洞挖掘工具,比较有代表性的如 Bochspwn[17]对内核层采用污点追踪,从而检测用户层泄露数据的行为;Digtool[18]针对 Windows 系统,可以自动化捕获程序执行过程中触发的漏洞;Syzkaller[19]是谷歌团队推出的一款针对 Linux 内核的无监督、覆盖引导的模糊测试工具;RapidScan[20]漏洞扫描器则通过自动化执行 nmap、dnsrecon 和 wafw00f 等多个安全扫描工具,尝试通过相互关联扫描结果来发现漏洞.

上述漏洞挖掘方法一般适用于小型规模软件,已经在各类小型规模软件的漏洞挖掘中取得了许多成果,但是在应对大型复杂软件系统以及变化多样的新型漏洞时,通常无法满足需求.如传统的静态分析方法往往依赖于人工专家构造漏洞模式,随着软件复杂性增加,人工构造成本过高,且人的主观性会影响误报率和漏报率.动态分析方法中,监测目标程序的崩溃是模糊测试发现漏洞的重要依据之一,因此测试效果依赖于输入种子的质量,存在测试冗余、测试攻击面模糊、难以发现访问控制漏洞和设计逻辑错误等问题[21].动态分析方法中的符号执行方法虽然能以较少的测试用例覆盖更多的程序路径,从而挖掘复杂软件更深层次的漏洞,但仍存在路径爆炸、约束求解难、内存建模与并行处理复杂等问题[22],单独处理大型软件系统时仍存在较大困难[23].

针对复杂软件系统的漏洞挖掘问题,学术界一方面尝试通过优化和改造现有方法来突破传统漏洞挖掘方法的瓶颈,另一方面尝试探索新的智能化软件漏洞挖掘方法.由于漏洞是代码缺陷被利用形成的,直观上认为分析代码可以获取最多的漏洞相关信息,即挖掘软件漏洞的能力与分析代码数据的能力紧密相关,而机器学习方法非常适合于从海量数据中发现和学习规律.理论研究中 Hindle 等人[24]利用统计学的方法将编程语言和自然

语言进行比较,研究结果表明两者具有非常相似的统计学特性,甚至编程语言更加规整,从而提出了代码的“自然说”假设.受到这个假设的启发,研究人员逐渐意识到运用统计学习方法对代码中蕴含的规律进行分析和泛化的可行性.

随着人工智能产业的兴起,大量基于机器学习的漏洞挖掘研究应运而生,覆盖了静态分析、动态分析以及动静态分析结合等典型场景.因为静态分析仍然是当前分析软件漏洞的主要手段,所以,基于静态分析的漏洞挖掘方法和动静态分析相结合的漏洞挖掘方法是近年来研究的重点和热点.而面向动态分析场景的研究主要针对缓解模糊测试、符号执行等动态分析方法存在的程序路径覆盖率低、路径爆炸以及约束求解难等问题,将机器学习方法用于测试输入生成与筛选、路径约束求解以及模糊测试参数配置预测等方面[23].本文主要侧重于基于静态分析的漏洞挖掘方法和动静态分析相结合的漏洞挖掘方法两类.其中,基于静态分析的漏洞挖掘方法面向静态分析场景,通过机器学习方法对代码的词法、语法、控制流和数据流等静态特征进行分析和学习,从而发现代码漏洞.动静态分析相结合的漏洞挖掘方法则通过动态分析对静态分析的检测结果进行校正,或综合分析代码分析得到的静态特征以及动态执行得到的动态特征,从而解决静态分析的高漏报率与动态分析的低代码覆盖率等问题[25].近年来相关研究[26]中的对比实验表明,与传统漏洞挖掘方法相比,基于机器学习的漏洞检测方法具有更高的准确率和完备性,能满足实际生产中对大型复杂软件系统进行漏洞挖掘的需求,是解决传统漏洞挖掘方法中瓶颈问题的新思路.

本文广泛收集了基于机器学习的软件漏洞挖掘领域的相关研究,文献选取标准为该文献对基于机器学习的软件漏洞挖掘方法提出新技术、新理论,或者该文献对基于机器学习的软件漏洞挖掘领域的相关理论和技术提供实证研究支持.本文的收录对象为 2007 年以来的符合定义的文献选取标准的相关研究,收集范围涵盖 SCI(Web of Science)、IEEE Xplore、SpringerLink、ACM Digital Library 以及中国知网 CNKI 等计算机领域常用的国内外全文数据库.按照文献选取标准在收集范围中通过关键字、标题、摘要进行手工检索,通过对收集的文献逐一阅读进行筛选,并通过各文献的参考文献进行查漏补缺,最终选择出 73 篇文献用于综述分析.自 2007 年以来,基于机器学习的漏洞挖掘方法的相关文献分布情况如图 1 所示,可以发现整体呈现逐年上升的趋势,其中 2018 年和 2019 年的相关文献最多.

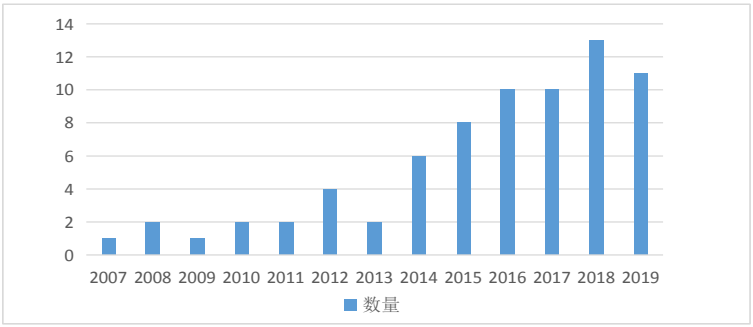

Fig.1 Literature distribution of software vulnerability mining based on machine learning

图 1 基于机器学习的软件漏洞挖掘文献分布情况

本文对近年来基于机器学习的软件漏洞挖掘研究进行综合分析,归纳了基于机器学习的软件漏洞挖掘的技术特征与工作流程.接着从提取原始数据特征切入,以代码的表征形式作为分类依据对现有研究工作进行分类论述.最后讨论了该领域存在的挑战和未来的发展趋势.

2 基于机器学习的软件漏洞挖掘流程

不同应用场景的机器学习训练和检测模型都具有相似的流程.具体应用于漏洞挖掘场景时,基于机器学习

的软件漏洞挖掘工作流程如图 2 所示.首先数据获取模块收集大量用于训练的软件程序相关数据(如已知是否包含漏洞的二进制代码、源代码、Github 提交信息等)以及用于评估的目标程序相关数据,分别应用于训练阶段和检测阶段.

上方虚线框表示的训练阶段中,由于获取到的数据集中通常漏洞较少且稀疏,会出现不平衡数据集问题.文献[27]通过实验表明在基于机器学习的软件缺陷预测方法中,数据预处理比分类器的选择更为重要.由于漏洞可以被理解为可利用的缺陷,预测缺陷和漏洞的研究对于数据的预处理方式相似,因此在训练阶段,首先需要通过随机欠采样、随机过采样和合成少数类过采样等方法对训练数据集进行平衡预处理.接着由于源代码、Github 提交信息等软件程序相关数据通常是文本表示形式,需要转换成向量表示形式才可用于模型计算,因此将预处理后的训练数据集通过数据表征模块进行一系列处理.在数据表征模块中,首先从训练数据集中按照代码度量、Token 序列、抽象语法树和图等代码的表征形式(由于训练数据集多为代码数据,因此本文将这类数据表征形式统称为代码的表征形式)抽取相应的代码表征.其次,通过编码模型将抽取得到的代码表征映射为向量表示形式.最后,将从数据表征模块得到的向量送到预先设计的机器学习模型中进行训练.在模型训练模块中,将从数据表征模块得到的向量作为输入,经过模型提取到的特征表达作为输出.根据计算得到的特征表达与原始标签的差异构建损失函数,通过优化方法来最小化损失函数,从而不断调整模型的参数.经过若干次训练得到一个用于进行漏洞分类或预测的分类器模型,供下方虚线框表示的检测阶段使用.

检测阶段中,首先将目标代码数据集按照同样的方法进行数据表征.接着将表征得到的向量送入到由训练阶段得到的分类器模型,得到目标代码的分类或者预测结果.最后,结合测试标签集计算模型算法的精确率(precision)和召回率(recall)等,进行模型评估和调优.

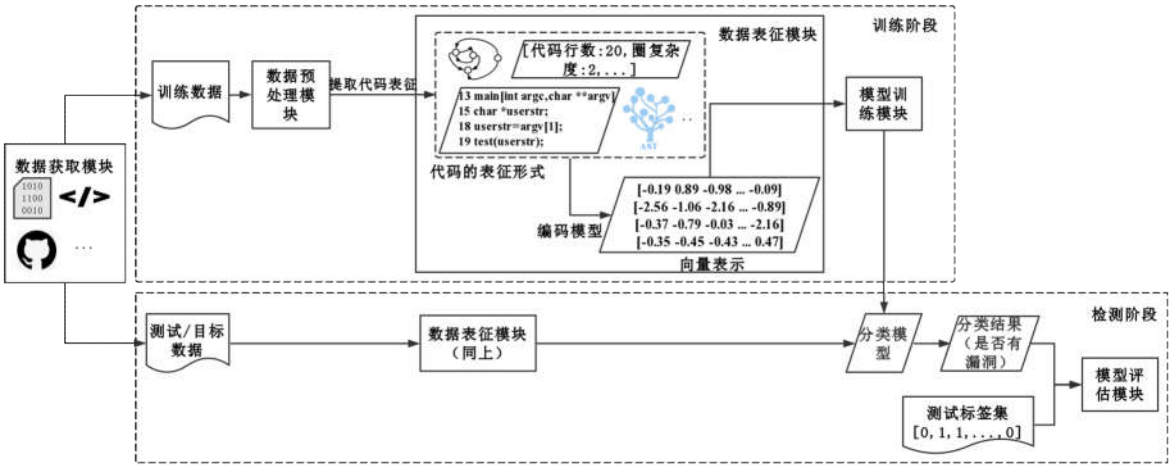

Fig.2 Software vulnerability mining process based on machine learning

图 2 基于机器学习的软件漏洞挖掘工作流程

通过上述的工作流程可以看出,基于机器学习的软件漏洞挖掘模型主要对数据表征模块中的代码表征形式与编码模型、模型训练模块中的机器学习模型进行改进,以优化漏洞挖掘的准确率和效率.

其中数据表征模块主要实现代码的特征选择过程,即从源数据(数据获取模块收集到的软件程序相关数据)中根据代码表征形式提取最具代表性的信息,并转化成可供后续机器学习模型训练的数值数据.具体而言,若根据表征形式选择的特征可以直接统计量化,如基于代码度量的表征形式提取的代码行数、圈复杂度、“for”计数等.那么在各代码块中通过统计量化得到各度量指标对应的数值,将统计结果组合形成固定长度的向量形式,用该特征向量来表示对应代码块.若根据表征形式选择的特征不可以直接统计量化,则将程序代码视为一种特殊的文本形式,并通过自然语言处理(Natural Language Processing,简称 NLP)领域中常用的技术来处理.如基

于 Token 的表征形式提取的代码段、函数调用序列以及程序执行路径等文本序列,通常使用词袋模型、TF-IDF 算法和 N-gram 模型将源数据视为与频率相关的术语集合,从而将源数据映射到向量空间.而树和图的表征形式,则通常从代码解析得到的树和图各结点中提取代码的语法信息,通过一定的方式组织成 NLP 领域中的语料形式,再通过词向量模型训练成对应的分布式向量表示.其中常用的词向量模型有 word2vec,word2vec 通过构建多层神经网络处理词单元,通过大量的语料训练,将词映射到一个高维向量中.word2vec 包含根据周围词来预测中心词的连续词袋模型(Continuous Bag-of-Words Model,简称 CBOW),以及根据中心词来预测周围词的 Skip-gram 模型这两种训练模型.

模型训练模块则需要选择对当前问题最优的机器学习模型,通过从数据表征模块得到的向量表示形式的数据对问题建模.对于机器学习模型的选择,机器学习模型依据训练数据集是否有标记分为有监督、半监督和无监督三类.除了少部分研究[28,29,30]通过聚类、LP 等算法对无监督和半监督模型在漏洞挖掘领域的应用进行探索,漏洞挖掘领域中的相关研究大多使用逻辑回归、随机森林和支持向量机(Support Vector Machine,简称 SVM)等有监督的机器学习模型.由于深度学习在图像处理、NLP 等领域的迅速发展,并且能取得比传统“浅层”机器学习模型更好的检测效果,深度学习也被逐渐引入到漏洞挖掘领域.常用的深度模型有多层感知机(Multi-Layer Perception,简称 MLP)、卷积神经网络(Convolutional Neural Networks,简称 CNN)、循环神经网络(Recurrent Neural Network,简称 RNN)以及长短期记忆网络(Long Short-Term Memory,简称 LSTM)等,其中 MLP 模型需要大量的训练参数来实现拟合,因此不适合处理较长的输入序列;CNN 模型的卷积操作在处理树和图等二维结构以及长文本序列时非常适用,但不适用于处理函数名等单个单词;RNN 模型在输入序列的长度过大时,存在梯度消失问题,可能导致训练无效甚至失败;LSTM 模型则在 RNN 模型的基础上引入了记忆单元和遗忘门,使 LSTM 模型在处理序列数据时具有更好的性能.由于代码是具有逻辑和语义的结构,具有紧密耦合的性质,因此易受攻击的代码片段的出现通常与其上下文代码都相关,选用的神经网络应该能够对输入序列进行正向和反向处理.近期研究常采用双向长短期记忆网络(Bidirectional Long Short-Term Memory,简称 BLSTM),以获得更深层的表示,达到更强的鲁棒性.

通过以上分析以及综合梳理近年来的相关研究,本文发现用到的编码模型集中于统计量化、word2vec、词袋模型、TF-IDF 算法和 N-gram 模型.机器模型的选择方面,除了少部分研究[31,32]引入了 LSTM、BLSTM 等在 NLP 领域中常用的深度学习模型,大多研究[26,33]使用 SVM、随机森林和朴素贝叶斯等传统的机器学习模型.暂时没有针对漏洞挖掘场景而提出的全新编码模型和机器学习模型,只是针对不同的表征方式,选取合适的编码模型和机器学习模型.即各项研究中的编码模型和机器学习模型的选择通常较为传统和集中,研究贡献点主要在于改进代码的表征形式,从而更大程度地从原始数据中提取特征供模型使用.这也正符合特征工程领域中的“数据和特征决定了机器学习的上限,而模型和算法只是逼近这个上限而已”这一通识.基于以上思考,本文认为把代码的表征形式作为对相关研究进行分类探讨的标准是合理且有意义的.

3 代码的表征形式

向量表示是机器学习模型的实际输入,所以需要将源数据转换为能够表现漏洞特性的向量形式.由于需要保留源数据的语义信息(例如数据依赖和控制依赖),通常引入中间表示作为源数据与其对应的向量表示之间的“桥梁”[31].中间表示即本文提到的代码表征,好的表征形式可以最大程度地从原始数据中提取特征,具有更丰富的漏洞特性表达能力.常用的代码表征形式有代码度量、Token 序列、抽象语法树和图,目前相关研究主要基于这四种表征形式进行优化和变种,四种形式的特征来源、检测速度及检测效果的对比关系如表 1 所示.本部分将具体介绍每种表征形式,并阐述每种表征形式中具有代表性的研究工作,最后将本部分所有提及的研究工作进行可视化汇总,如图 3 所示.

Table 1 Representation of code

表 1 代码的表征形式

代码表征形式	特征来源	检测速度	检测效果
代码度量	程序属性	快	较差
Token	词法信息、程序运行信息	较快	较好
抽象语法树	词法、语法信息	较慢	好
图	语法、语义信息	最慢	好

3.1 基于软件代码的度量

代码度量是一组用于衡量软件质量的软件度量值,可供开发者确定项目的潜在风险、当前状态以及跟踪开发进度,也可以作为表示代码特性的度量指标,常用的代码度量指标有代码行数、圈复杂度、继承深度和类耦合等.基于代码度量的表征方式通过选取若干个代码度量指标,得到能够概括代码整体信息的度量序列,使用该度量序列表示相应代码.

早期对于代码漏洞的检测大多基于“越复杂的代码中漏洞越多”这一假设,而 Shin 等人[34,35]通过实验证明代码复杂度指标与漏洞之间的相关性很弱.软件设计中的相关研究[36,37]表明,与复杂性、耦合性和内聚性(简称 CCC 度量)相关的结构度量是软件体系结构质量的重要指标,而软件体系结构质量是影响软件系统最终质量的重要早期设计决策之一.基于这种认识,Chowdhury 等人[38,39]引入了耦合性、内聚性指标,通过实验证明 CCC 度量与代码漏洞在统计上显著相关,并表明高复杂性、高耦合且低内聚的软件组件更容易产生漏洞.同时提出了一个基于 CCC 度量的漏洞自动预测框架,对 Mozilla Firefox 的 52 个版本进行了大规模的实证研究,相较于 Shin 等人的研究[34,35],该框架在性能上有所提高.并比较了 C4.5 决策树、随机森林、逻辑回归和朴素贝叶斯的预测性能,结果表明在正确识别易受攻击文件和总体预测精度的评估中,C4.5 决策树和随机森林的效果显著优于逻辑回归和朴素贝叶斯.Zimmermann 等人[40]在 Windows Vista 上提出了第一个大规模的实证研究,评估如复杂度、流失量、覆盖率和依赖性指标等度量指标的功效,同时评估了这些度量指标与漏洞的相关程度.结果表明,没有一种通用的度量指标可以有效地预测漏洞,因此在预测漏洞时,可以使用度量的组合来获得更合理的精确度和召回率.该研究用到的度量指标是进行缺陷预测时常用的指标,但是由于漏洞的出现还取决于组件的使用,因此漏洞预测比缺陷预测更加复杂,应当充分利用如缓冲区溢位、算术错误、欺骗攻击错误和拒绝服务攻击错误等软件安全相关的特定指标.

Younis 等人[33]选取了代码行数、圈复杂度、嵌套程度、信息流和调用函数等八个代码度量指标描述代码特征,通过机器学习方法预测漏洞的可利用性.首先使用 Welch’s t-test 检验了单个度量的漏洞检测能力,然后使用基于相关性的特征选择法、包装法和主成分分析这三种特征选择算法对八个代码度量的组合选择特征子集,最后使用逻辑回归、朴素贝叶斯、随机森林和 SVM 四种分类器测试所选择的度量指标子集的预测能力.实验部分从美国国家漏洞资料库(National Vulnerability Database,简称 NVD)中收集了来自 Linux 内核和 Apache HTTP 服务器中的 183 个漏洞,其中包括 82 个利用代码已经在漏洞利用库 Exploit-DB 中公开发布的可利用漏洞,通过这八个代码度量对漏洞进行描述.实验结果表明,所选择的代码度量可以对漏洞是否被利用进行有效的特征表达,并且当使用包装法进行特征选择以及随机森林作为分类器时,F 值(F-Measure)达到 84%,漏洞检测能够达到最佳性能.这项研究给我们带来一些启发,由于不被利用的漏洞可能不会造成威胁,因此如果使用考虑安全领域知识的度量[41],将漏洞检测的范围缩小至易被利用的漏洞检测,可以使得检测更具有针对性,检测结果也更具有实际应用价值.

传统的代码度量指标只关注代码本身,而全面描述一个漏洞还应该包含漏洞的工作方式以及成因等业务逻辑信息.Bozorgi 等人[42]从现有漏洞披露报告的文本字段、时间戳、交叉引用和其他条目中提取每个漏洞的高维向量(维数=93 578),使用 SVM 模型进行脆弱性分类,并预测漏洞是否会在特定的时间段内被利用.虽然提取的特征中大多数最终被证明是不相关的或多余的,但是该研究中特征提取的目标是提取尽可能多的信息用于后续的统计分析.该研究旨在用大规模的统计法取代传统小规模的启发式方法,实验结果表明,与基于专家知识和行业标准的启发式算法相比,该模型能更准确地预测单个漏洞是否会被利用,以及多久会被利用.

除此之外,考虑到代码开发人员作为项目的建造者,对代码的质量也会造成一定的影响,因此代码存储库中包含的能反映开发人员情况的元数据可能与代码质量高度相关.基于这种认识,Perl 等人[26]提出了VCCFinder 模型,基于代码度量和从 Github 代码库中收集的元数据,采用 SVM 模型识别漏洞代码.为了评估该模型,文章构建了一个包含 66 个 C 和 C++项目的 170 860 次代码提交的评估数据库,其中包含 640 次含漏洞的提交.与具有代表性的静态漏洞挖掘工具 Flawfinder 进行对比实验,实验结果表明,当固定召回率为 0.24 时,Flawfinder 的精确度只有 0.01,而 VCCFinder 的精确度达到 0.6.但是评估数据库中的项目都至少创建了一个CVE,文章并没有提到 VCCFinder 如何在迄今尚未收到任何 CVE 的项目上执行.王飞雪等人[43]则基于文本报告和代码修复定义了一组特征,使用 WEKA 工具包中的随机森林、C4.5 决策树、逻辑回归和朴素贝叶斯四种机器学习方法实现漏洞分类.其中文本报告主要以修复时间、开发人员、安全漏洞位置和严重性为分析维度,代码修复则以更改文件数、更改代码行数以及漏洞修复熵为分析维度.该研究依据失效再现性条件,将 bug 类型分为在测试阶段易再现的波尔 bug 和难以再现的曼德博 bug,最终实现的分类器以能够有效区分具有波尔 bug 的安全漏洞 BV 和具有曼德博 bug 的安全漏洞 MV 为目标.实验结果表明,具有 67%准确率(accuracy)的 C4.5 决策树可以识别 69%的 MV,比其他方法在准确率和 F 值上取得更好的效果.

以上几项研究扩大了代码度量的范围,除了代码本身还考虑到了开发者的因素,取得了很好的效果.但是这些代码度量与漏洞之间的相关性依然较弱,一定程度上制约了代码度量的有效性.并且 Rice 定理[44]指出任何关于程序的非平凡问题都是不可判定的,而判断程序是否存在漏洞是非平凡问题,因此是不可判定的.为了解决这个问题,Meng 等人[45]通过静态分析与符号执行相结合的方法检测缓冲区溢出漏洞.首先,基于数组索引操作的复杂度度量,训练 SVM 分类器用于进行漏洞检测.然后,将分类结果中易受攻击的函数反馈给函数调用图引导的符号执行,确定其是否真正存在漏洞.

基于代码度量的表征方式通过若干个代码度量指标得到对应代码的度量序列,即这种度量序列通过量化程序属性来表示代码的各种信息,并且量化表示的序列非常适合统计分析,因此检测速度快.虽然相关研究不断地扩大代码度量的范围,但是由于代码度量是基于程序的整体属性,因此与漏洞代码本身关联性不强,检测能力较差.

3.2 基于Token的表征

基于 Token 的表征形式通过对源代码进行词法分析得到.即在编译过程中扫描源代码的字符流,依据程序语言的词法规则标记源代码的标识符、关键字、函数名和运算符等重要信息,从而将源代码的字符流转换成等价 Token 序列,映射到向量空间作为机器学习模型的输入.除此之外,由于函数调用序列、执行轨迹等序列是由程序代码执行得到的,也属于代码级别的统计信息,因此本文将这类程序执行信息也归为 Token.

Yamaguchi 等人[46]2012 年通过词法分析将源代码解析为各个函数,对于每个函数提取引用类型和函数名称,将提取出来的信息称为 API 符号.使用提取的 API 符号将每个函数都映射到向量空间,并使用主成分分析法自动确定脆弱函数的 API 使用模式.预测阶段,从已知漏洞的 API 使用模式开始,利用这些模式指导代码审计,识别具有类似特征的潜在漏洞代码,将此过程称为漏洞外推.这种方法只考虑到 API 使用模式,并没有考虑源代码的结构信息,因此不能发现与代码结构相关的漏洞.Li 等人[47]同样将函数名作为特征,提出了基于深度神经网络的轻量级漏洞发现方法 LAVDNN.首先对每个函数提取其函数名,并将其分为脆弱函数数据集和良性函数数据集.由于实际编程中,函数的命名是一项主观性任务,当函数名倒过来也可能是相同的含义,因此该研究利用 BLSTM 神经网络对函数进行分类,对每个函数的脆弱概率进行预测.实验结果表明,LAVDNN 在 C/C++以及 python 源代码中的 F2 分数(F2-score)分别达到了 0.91 和 0.915.作为一个轻量级辅助工具,可以帮助研究人员把分析重点放在脆弱概率较高的函数上,从而降低假阳性率.但是函数名能够提供的信息毕竟有限,且命名可能存在不规范、不准确的问题,所以会造成该方法的结果可信度低,只能作为初筛工具.

Scandariato 等人[48]2014 年通过对源代码进行文本挖掘来预测软件组件是否脆弱,每个组件的特征由对应源代码中的一系列术语以及出现频率来表示.研究选择了决策树、k 近邻、朴素贝叶斯、随机森林以及 SVM 五种机器学习方法,通过分析发现朴素贝叶斯和随机森林取到了最好效果.研究结果表明,该方法用于项目内预

测时,具有良好的精确度或召回率.

Padmanabhuni 等人[49]2014 年提出了一种基于静态分析的缓冲区溢出漏洞预测方法.在静态分析工具 CodeSurfer 的基础上实现了 BOMiner 工具,提取潜在的易受攻击语句的控制和数据依赖信息,进而形成轻量级静态代码属性.再使用 WEKA 工具包中的朴素贝叶斯、J48 决策树、简单逻辑回归、MLP 和 SVM 五种机器学习方法建立预测模型,其中 J48 决策树效果最佳,实现了 95%的召回率与 80.9%的精确度.

考虑到 Token 间存在关联,为了减少特征量和搜索空间,Pang 等人[50]在 2017 年选取源代码文件中的连续 Token 序列作为特征,使用 N-gram 分析构建 Token 之间的关联,再利用统计特征选择算法来减少特征量和搜索空间.该研究采用深度神经网络技术,用随机梯度下降法和批量归一化方法训练整流线性单元,建立用于预测脆弱软件组件的二分类器.通过分析 Java Android 程序进行评估,使用的 Token 包括类、对象、方法、参数和变量,强调了在分析和计算中考虑连续 Token 序列的可行性.

虽然直观上我们认为提取的特征越复杂,包含的信息越丰富,检测效果越佳,但 Chernis 等人[51]在 2018 年通过实验证明了简单的特性也包含许多判断函数是否脆弱的相关信息.该研究从 C 源代码的函数中提取相对简单的特征(字符数、字符多样性、熵、最大嵌套深度、箭头数、“if”计数、“if”复杂度、“while”计数和“for”计数),以及复杂的特征(字符 N-gram 模型、单词 N-gram 模型和后缀树),选用了朴素贝叶斯、k 近邻和 k 均值等分类器对样本进行分类.实验结果表明,与复杂特性相比,简单特性的性能出乎意料的更好(使用朴素贝叶斯作为分类器时,准确率分别为 74%与 69%).同时也对不同分类器效果进行了测试,发现不同分类器得到的准确率结果相近.相对而言,使用简单特性时,朴素贝叶斯与 k 均值分类器效果最好.而使用复杂特性时,朴素贝叶斯与 SVM 效果最好.

Russell 等人[52]在 2018 年提出一种基于深度表征学习的快速、可伸缩的漏洞检测工具.首先,创建一个自定义的 C/C++词法分析器来构建简单、通用的函数表示,将词法分析得到的函数 Tokens 映射到固定的 k 维表示中.该 k 维表示由函数 Tokens 经过 one-hot 编码,然后在模型训练的反向传播时进行一次线性变换学习得到.接着,利用类似于使用 CNN 和 RNN 进行句子情感分类时的特征提取方法,从 k 维表示中提取神经特征.最后,将构建的神经特征作为随机森林分类器的输入,以提升在完整数据集上的分类性能.

Li 等人[31]在 2018 年提出基于深度学习的漏洞检测系统,以解决专家手工定义特征带来的极大人力消耗以及高漏报率.VulDeePecker 采用 code gadget 来表示程序,将 code gadget 编码为向量作为 BLSTM 的输入来自动学习,从而生成漏洞模式.其中 code gadget 是语义相关的多行代码(可以不连续),其生成过程需要先提取库/API 函数调用,接着提取库/API 函数调用的每个参数的一个或多个程序切片,最后将针对同一个库/API 函数调用的多个切片组合形成一个 code gadget[53].为了验证有效性,将 VulDeePecker 与人工定义规则的静态分析工具(开源工具和商业工具)进行对比,结果表明 VulDeePecker 具有更低的漏报率,并且 VulDeePecker 在 Xen、Seamonkey 和 Libav 这 3 个开源软件产品中检测到 4 个在 NVD 漏洞库中未公布的漏洞.但是在 VulDeePecker 解决方案中,通过启发式方法将 code gadget 转换成向量的形式时可能会造成一些信息的丢失,一定程度上影响检测结果的有效性.虽然 code gadget 可以检测一段代码是否包含漏洞,但是无法提供具体的漏洞信息.对此,Zou 等人[54]利用词法分析技术分析 code gadget 的语句属性,再根据语句属性和代码文本来匹配漏洞语法规则,进而从 code gadget 中提取 code attention.提取出的 code attention 是符合漏洞语法规则的代码语句集合,可以辅助模型识别漏洞的具体类型.基于这一概念,该研究提出了由全局特征学习模型、局部特征学习模型和特征融合模型三个 BLSTM 网络构成的 μVulDeePecker 系统.其中全局特性由 code gadget 学习得到,并且包含一些程序语句间更广泛的语义关系.局部特性则由 code attention 学习得到,并且特定于程序中单条语句.这样的设计使得 μVulDeePecker 系统可以检测多种类型漏洞,并且也可以辅助开发者确定漏洞位置.

实际生产中,由于代码的重用和共享,获取源代码并非易事.为解决这一问题,Zuo 等人[55] 在 2019 年将 NLP 中的方法和思想应用到汇编语言中,以解决跨体系结构的二进制分析问题.该文章主要针对两个研究问题:给定一对不同指令集体系结构(instruction set architectures,简称 ISA)的二进制基本块,确定其语义是否相似;给定一段关键代码,确定它是否包含在另一段不同 ISA 的代码段中.针对第一个研究问题,借鉴神经网络机器翻译

(Neural Machine Translation,简称 NMT)的思想,将指令当作单词,将基本块当作句子.再使用神经网络处理汇编语言,利用 LSTM 模型将基本块编码成一个可以捕获指令序列间语义关系的高维数值向量,通过度量高维数值向量间的距离来检测基本块的相似度.针对第二个研究问题,将相关代码的控制流图分解为多个路径,每个路径都可以看作是基本块的序列,应用最长公共子序列(Long Common Sequence,简称 LCS)算法来比较这些路径的相似性,从而定量计算两段代码的相似度.

仅使用通过词法分析得到的静态特征存在误报率高的问题,针对这一问题,可以结合动态分析得到的动态特征进行修正,从而提高预测精度.Wu 等人[56]以函数调用序列为特征,提出了 CNN、LSTM 和 CNN-LSTM 这三种用于漏洞检测的深度学习模型.该研究从 9 872 个二进制程序中提取动态特征以训练检测模型,实验结果表明,三种深度学习模型的预测精度相近,均远高于传统的 MLP 方法,其中 CNN-LSTM 模型的预测准确度达到 83.6%.VDiscover[57]通过二进制文件静态特征和执行分析动态特征来量化程序,以解决检测操作系统漏洞时的时间开销大及准确率低的问题.在有限的时间内执行一个测试用例和连接程序事件,将源代码结构“近似”为一组对标准 C 库的函数调用序列,将其作为静态特征;根据程序对 C 标准库的具体调用顺序捕获程序行为,将其作为动态特征.使用 N-gram 模型和 word2vec 模型对提取的动静态特征进行向量化,利用逻辑回归、MLP 和随机森林三种有监督机器学习方法预测是否存在内存破坏漏洞.实验结果表明,将机器学习应用于大规模的二进制单漏洞检测可以显著增加在操作系统中发现的漏洞数量,其中使用动态特征训练的随机森林分类器能达到最佳效果,平均测试误差(error)为 31%.但是该研究的实验结果表明,结合动静态特征的测试误差和仅使用静态特征的测试误差相似,说明并没有找到一种有效结合动静态特征的方法.Tireias[58]以从终端收集的安全事件序列为输入,按时间戳排序重构为安全事件时间序列,利用 RNN 模型预测攻击者将采用的确切漏洞攻击操作,而不仅仅预测攻击是否会发生(即漏洞是否会被利用).

代码度量是对整个程序项目中各种属性的概括统计,相比之下,Token 序列则由直接对代码级别进行统计得到,因此基于 Token 的表征形式与漏洞的相关性更强,检测能力也更强.但是由于基于 Token 的表征形式实现代码级别的抽象,提取出来的 Token 序列需要经过 one-hot、word2vec 等模型进行向量化处理后,才能得到可供机器学习模型训练的特征集,因此检测速度一般慢于基于代码度量的表征形式.

3.3 基于抽象语法树的表征

抽象语法树(Abstract syntax tree,简称 AST)是编译过程中的一种中间表示形式,树节点上存储的是代码的语法结构信息[59].基于 AST 的表征形式通常先使用 lex/yacc、flex/bison、ANTLR 等开源的词法和语法分析工具进行预处理,生成词法分析器和语法分析器.接着对代码进行词法分析和语法分析,将分析的结果构建成 AST,对语法树节点进行编码作为代码的特征.

为了在代码审计过程中帮助安全分析人员发现漏洞,Yamaguchi 等人[60]在 2012 年基于 island grammars 的概念,使用解析器对代码库中的所有函数提取 AST,并确定这些 AST 的结构模式,因此代码中的每个函数都可以描述为这些结构模式的组合.然后将函数的 AST 映射到向量空间,利用潜在语义分析技术在代码中识别结构模式,并将其外推至一个代码库,就可以向分析人员建议可能存在相同缺陷的函数.在 LibTIFF、FFmpeg、Pidgin 和 Asterisk 这四个开源项目的源代码上评估了该方法,对于其中三个项目,能够通过检查一小部分代码库来识别零日漏洞.Lin 等人[32,61]参考 Yamaguchi 等人在 2012 年的工作[60],使用 ANTLR 语法分析工具获得源代码的 AST,通过深度优先遍历获得序列化形式的 AST.输出端使用 word2vec 工具中的 CBOW 模型,将序列中的每个元素映射到语义空间中的一个 100 维向量,在这个语义空间中,相似的元素非常接近.再将得到的向量作为 BLSTM 模型的输入进行训练,从而获得代码更深层次的脆弱性特征表示.选取 23 个传统的代码度量指标做对比实验,将提取的特征都用于训练一个随机森林分类器,发现无论在项目内还是跨项目场景中,通过该方法提取的源代码特征在预测漏洞函数上明显更有效.通过迁移学习得到相似项目的丰富特性,也解决了机器学习中的冷启动问题.但是该方法以单个源代码函数作为输入,无法检测出涉及多个函数或多个文件的漏洞.

Medeiros 等人[62]在 2014 年提出一种用于检测和修正 web 应用程序中漏洞的方法.首先,通过解析源代码生成 AST,再基于 AST 进行污点分析,生成描述候选漏洞控制流路径的树,从候选漏洞控制流路径中获取属性.

接着,通过评估随机森林、朴素贝叶斯、MLP 和逻辑回归等十种机器学习分类器的效果,发现逻辑回归为当前问题的最佳分类器.因此使用逻辑回归分类器对每个候选漏洞是否为假阳性进行预测.对预测不为假阳性的漏洞,将确定漏洞、修复程序以及修复程序插入的位置,同时评估漏洞的误报率.

Meng 等人[28]在 2016 年提出一种基于半监督学习的缓冲区溢出漏洞的预测方法.首先,利用 ANTLR 从 C/C++源文件中提取 AST.然后,按照 Kratkiewicz 等人[63]通过总结缓冲区溢出漏洞的各种表现形式得到的 22 个属性,从 AST 中的每个函数中提取一个 22 维向量.最后,通过 LP 半监督学习方法训练分类器,从而预测缓冲区溢出漏洞.该研究的研究范围只局限于缓冲区溢出漏洞,所以在设计过程中忽略了许多与缓冲区溢出无关的属性,这样确实极大地简化了问题,但也使该方法很难泛化到其他类型漏洞的预测问题上,通用性较差.

Mou 等人[64]2016 年提出了一种基于树的卷积神经网络(TBCNN)用于编程语言处理,在程序代码的 AST 上设计一个卷积核来捕获结构信息,并引入连续二叉树和动态池概念使模型能够处理不同大小和不同形状的树.通过根据功能对程序进行分类,以及检测特定模式的代码片段这两个任务对 TBCNN 模型进行评估.实验结果表明,在两个任务中 TBCNN 模型在性能上都优于 RNN 模型、基于 RBF 核的 SVM 模型和基于词袋模型的深度神经网络(Deep Neural Networks,简称 DNN)等基线方法,且具有更强的鲁棒性.

对于 PHP web 应用的漏洞检测问题,Anbiya 等人[65]在 2018 年提出一种基于机器学习进行漏洞检测的技术.将通过宽度优先遍历算法遍历 AST 得到的 AST Token,以及 PHP 提供的 token_get_all 函数得到的 PHP Token 分别作为特征.接着使用剪枝或过滤的方法处理 AST 以及 PHP Token 中一些不可用的节点或子树,再通过 TF-IDF 算法将特征映射到向量空间.最后使用高斯朴素贝叶斯、决策树和 SVM 分类器进行分类.结果表明,当 PHP Token 作为特征且高斯朴素贝叶斯作为分类器时,能取得最高的召回率.

Peng 等人[66]提出了将 UNIX diff 工具与 AST 相结合的松散完整性验证方法,面向 Python 脚本进行漏洞检测.该方法被嵌入到 Python 解释器中,当用户运行 Python 脚本时,解释器自动查找备份文件并检查脚本的完整性.如果脚本的完整性被破坏,带有污点模块的 Bandit 将检测漏洞.如果没有漏洞,程序正常运行,否则程序停止运行,得到差异文件和漏洞检测报告.其中完整性验证的目的不仅是检测当前文件是否完整,更重要的是获得一个可用于漏洞检测的差异文件.Bandit 由 OpenStack 安全组开发,是一个针对 Python 的静态安全分析框架,可以发现 Python 中很多常见的安全问题.实验结果表明,该 Python 安全分析框架具有良好的性能,带有污点模块的 Bandit 可以降低误报率.

虽然基于 Token 的表征形式紧密结合了代码信息,逐行实现对代码的抽象,但是只进行词法分析会忽略代码的结构化信息,从而导致代码抽象级别较低.AST 综合考虑了代码的词法分析与语法分析,兼顾代码语义和结构信息,它还包含函数的组件和功能级控制流之间的关系,因此相比于前两种表征形式会更加全面[67].但是 AST 是树形结构,其生成和处理过程相较于 Token 序列更为复杂,因此检测速度较慢.

3.4 基于图的表征

图是一种比树更为抽象复杂的表示形式,图的顶点之间是多对多的关系,并且不具有树结构中明确的父子节点层次划分.因此基于图的表征形式能反映代码中更丰富的语法和语义特征,有很好的检测能力.如常用的程序依赖图表示软件程序的控制依赖和数据依赖关系,控制流图(Control Flow Graph,简称 CFG)则是由编译器在内部维护的一种程序的抽象表现,代表了程序执行过程中会遍历到的所有路径.

软件行为图是由函数调用和基本块执行关系构成的有向图,其中一个软件行为图中的每个节点对应一个方法或基本块,每条边对应各自节点间的执行关系,包括调用、返回或转换,能有效表示程序执行路径.Cheng 等人[68]通过软件行为图来对程序执行路径进行建模.给出了一组正确执行和错误执行的行为图,提取最具判别性的子图来比较正确执行和错误执行的程序流,所提取的子图不仅可以精确定位错误,而且还为理解和修复错误提供上下文信息.因为生成多个漏洞签名的排序列表比生成单个最佳签名包含更多信息,所以将图挖掘算法 LEAP 从挖掘最具判别性子图扩展到挖掘 top-k 判别子图,从而得到 top-k 判别子图的排序列表,这些子图中可能包含漏洞的位置.实验结果表明,该方法能够有效地挖掘出用于漏洞定位和上下文识别的判别子图.但是为了限制图挖掘的搜索空间,该方法只挖掘连通子图,因此连通性约束以及非可疑边的去除可能会导致一些有意义签

名的丢失.

Nguyen 等人[69]在 2013 年基于组件依赖图预测脆弱性组件,其中组件由数据项和可执行代码组成,组件依赖图描述软件组件之间的关系,反映了一个组件依靠另外一些元素来支持某个特定功能.组件依赖图从成员依赖图中生成,成员依赖图是由软件系统的数据项和方法构成的有向图,通过从代码的详细设计规范中提取信息与 Doxygen 静态分析器获得.在成员依赖图中,将一个组件上的所有成员节点聚集成一个组件节点,并且将属于两个组件且方向相同的成员间的边分解为这两个组件间的单条边,以此构成组件依赖图.实验部分以组件依赖图为代码表征形式,使用 WEKA 工具包中的贝叶斯网络、朴素贝叶斯、神经网络、随机森林和 SVM 五种分类器,对 Mozilla Firefox 1.5 和 2.0 版本中使用的 JavaScript 引擎的脆弱性组件进行分类和脆弱程度排序.实验结果表明,将组件依赖图作为代码表征来预测脆弱性组件的模型具有较高的准确率(84%)和召回率(60%),并且具有比 Shin 等人 2008 年提出的模型[35]更低的假阴性率.

由于许多漏洞只有通过同时考虑代码的结构、控制流和依赖关系才能充分发现,Yamaguchi 等人[70]2014 年提出了一种新的代码表示方法,即将经典程序分析中 AST、CFG 和程序依赖图的概念合并成一个联合数据结构,称为代码属性图,从而提升对漏洞上下文的表征能力.其中属性图是带属性和边标记的多重图,图中所有节点和边都可以指定属性,并依据属性进行赋值.AST 表示程序结构如何嵌套,CFG 表示语句的执行顺序,程序依赖图则明确数据流和控制依赖.系统首先通过对给定代码库中的每个函数进行解析和分析,生成 AST、CFG 和程序依赖图,并将这三种图表示为属性图.再将三种属性图以语句节点为融合点进行融合,得到代码属性图.最后基于明确的调用与被调用关系将所有函数的图链接在一起,从而将整个代码库表示为一个大型代码属性图.通过图遍历为常见类型的漏洞创建有效的检测模板来发现漏洞,实验结果表明,几乎 2012 年报告的所有 Linux 内核漏洞都可以使用代码属性图来描述,同时该方法发现 18 个 Linux 内核中以前未知的漏洞.由于程序中可用于创建漏洞的检测模板的信息有限,且通常分布在不同函数中,而代码属性图没有考虑到过程间分析.因此,Yamaguchi 等人 2015 年[30]通过在函数调用点的实参和被调用函数的形参之间,以及在调用点和被调用函数的返回语句之间添加边,得到一个表示函数之间调用关系的图.再使用后序支配树来检测参数修改,从而对这个图进行改进,得到改进后的代码属性图.以此实现通过增加语句优先级信息来扩展代码属性图,并提出一种使用聚类算法来自动推断 Taint-Style 漏洞搜索模式的方法.即给定一个安全敏感的 Sink,使用聚类算法提取调用模式.再将调用模式转换为能审计大型代码库的图遍历形式的搜索模式,从而自动识别出与之相关的 Sink-Source 系统,并构建出模拟这些 Sink-Source 系统中数据流和输入检查的判断模式,用于实现漏洞检测.实验结果显示,要审计的代码量平均减少了 94.9%,即自动生成的搜索模式可以精确地对污染型漏洞进行建模.

为了提高匹配精度,受到基本块上具有不同属性的思路启发,2016 年 Qian 等人[71]利用具有不同基本块属性的控制流图作为表征形式,并将其称为属性控制流图(Attributed Control Flow Graph,简称 ACFG).具体而言,ACFG 从二进制函数中抽取生成,每个顶点都是带有一组属性的基本块,且主要关注统计和结构这两类属性.其中统计属性用于描述基本块内的部分统计信息,包括字符串常量、数值常量、转移指令数量等,而结构属性则用于描述 CFG 中基本块的位置属性,包括子结点数量、结点的中心性.确定了代码的表征形式后,利用 ACFG 对数据集中的函数进行建模,再对数据集中的 ACFG 使用聚类算法,进而实现了漏洞搜索引擎 Genius.Xu 等人[72]对 Genius 进行改进,实现了基于 ACFG 的跨平台相似度检测原型 Gemini,并设计了使用 Structure2vec 图嵌入网络[73]的 Siamese 架构.Siamese 架构首先以 ACFG 的形式提取原始特征并作为每一个图嵌入网络的输入,然后获取其对应的高维向量,最终输出两个高维向量的 cosine 距离来衡量向量间的相似度.在由 OpenSSL 构建的测试集中,Gemini 可以实现比其他基于图形匹配的方法更高的 AUC 值,并且将现有技术的 Embedding 生成时间加快 3 到 4 个数量级,将所需的训练时间从 1 周以上缩短到 30 分钟至 10 小时.以上几项研究主要考虑控制流,但是正常行为和有漏洞的异常行为可以执行相同的控制流,它们可能只会在沿着控制流流入的数据上产生差别,因此仅考虑控制流并不完善.

由以上几种图表征形式,可以发现图能够提供代码的结构信息,作为直接从代码文本中读取信息的补充.Harer 等人[74]考虑了两种互补的方法,一种方法首先从编译和构建过程中生成程序中间表示,然后使用从程

序中间表示中提取的特性,另一种方法直接对源代码进行操作.其中基于构建的模型首先提取函数的 CFG,CFG 的每个节点都是一个基本块,即没有分支的代码单元,连接基本块的边表示基本块间的执行关系.在 CFG 中,提取每个基本块中执行操作的特征,以及变量的定义和使用.从提取的特征中手工定义了一个 116 维的特征向量,输入到随机森林分类器中实现分类.基于源代码的模型则实现了一个自定义的 C/C++词法分析器,解析代码并将元素分为注释、操作符和字符串文本等类型.实验部分测试了三个模型,第一个模型是极端随机树分类器,以由词袋模型得到的向量为输入.第二个模型则将由 word2vec 模型得到的序列送入 TextCNN 模型中,通过卷积层与全连接层直接得到分类结果.第三个模型也是极端随机树分类器,以 TextCNN 的卷积层学习到的特征为输入,其中第二个模型获得最佳的性能.除了比较基于源代码和基于构建的模型的结果外,该研究工作还生成了一个使用基于构建的向量和基于源代码的词袋向量的组合模型.结果表明,组合模型的性能比单独的模型更好,即基于构建的特性确实提供了源代码本身没有提供的附加信息.并且基于构建的模型虽然比基于源代码的模型性能差,但是基于构建的模型性能在两个不同的数据集上非常相似.这表明基于构建的模型可以学习不同类型的代码,未来的工作将测试该模型跨项目的泛化效果.

由于基于图的表征形式能反映代码中更丰富的语法和语义特征,相较于其他三种表征形式更加抽象,能蕴含代码中更深层次的特征.但是由于图的生成过程复杂、开销大,且处理基于图的表征形式需要使用到图匹配算法,这类算法的时间复杂度和空间复杂度往往非常高,甚至会达到 NP 难.因此基于图的表征形式的研究检测速度慢,不适用于大型软件程序的检测.

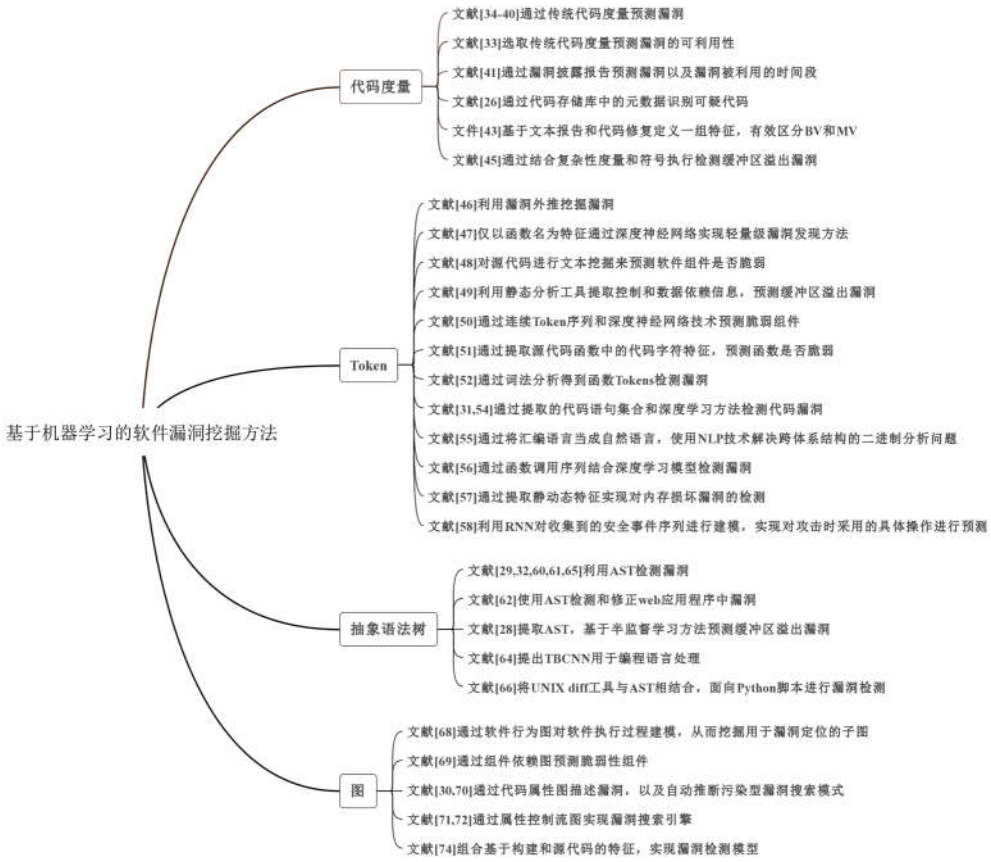

Fig.3 Summary of research on vulnerability mining based on code presentation

图 3 以代码表征为分类标准的漏洞挖掘研究工作汇总

4 分析与讨论

4.1 总述与对比

对基于机器学习的漏洞挖掘模型的评估方式主要有两种:一种是根据机器学习算法中常用的评估指标,如准确率、召回率、F1 分数(F1-score)以及 F 值等;另一种是将模型应用于构建的数据集中,依据其能否在实际项目中检测到漏洞来评估模型.由于各项研究的主要贡献以及评估方式不一致,且实验部分并不建立在统一的数据集和实验平台上,对比实验结果无法说明模型的优劣,因此本文以技术特点为切入点对各项研究工作进行对比分析.

Table 2 Summary of existing research work

表 2 现有研究工作小结

表征形式	文献	被分析对象	特征的选取	模型构造	编码方式(若有,特征选择)	是否跨项目	漏洞类型
软件度量	文献[33]	脆弱性函数	代码行数、圈复杂度、嵌套程度、信息流、调用函数等八个软件度量	逻辑回归、朴素贝叶斯、随机森林、SVM	统计量化(特征选择法、包装法和主成分分析)	否	不限
	文献[26]	C/C++源代码、Github 元数据	代码度量与 GitHub 元数据特性	SVM	词袋模型	是	不限
	文献[45]	源代码	数组索引操作的复杂度	SVM	统计量化	否	缓冲区溢出类型
Token	文献[51]	C 源代码	简单特征(字符数、熵、最大嵌套深度、“if”复杂度、“for”计数等),以及其复杂特征(字符 N-gram 模型、单词 N-gram 模型和后缀树)	朴素贝叶斯、k 近邻、k 均值、神经网络、SVM	统计量化	否	不限
	文献[31]	C/C++源代码	"code gadget"(语义相关的可以不连续的多行代码)	BLSTM	word2vec	否	缓冲区溢出类型、资源管理异常类型
	文献[57]	二进制代码	函数调用序列、程序执行路径	逻辑回归、MLP 和随机森林	N-gram 模型、word2vec 模型	否	内存损坏类型
AST	文献[32]	C/C++源代码	AST	BLSTM	CBOW 模型	是	不限
	文献[65]	PHP 源代码	PHP 原生 token 和 AST	高斯朴素贝叶斯、决策树、SVM	TF-IDF	否	不限
	文献[28]	C/C++源代码	AST	LP 半监督学习方法	统计量化	否	缓冲区溢出类型
图	文献[69]	源代码	组件依赖图	朴素贝叶斯、随机森林、神经网络、贝叶斯网络、SVM	根据公式量化	否	不限
	文献[30]	C/C++源代码	代码属性图	聚类、图遍历	词袋模型	否	污染类型
	文献[72]	二进制代码	属性控制流图	Siamese 架构	Structure2vec 网络	是	不限

对于每种表征形式,本文从第三章节梳理的研究工作中挑选了三项具有代表性的研究,按照技术特点进行总结.如表 2 所示,表中的每一行都代表一项研究工作,第一列代表了该项研究的代码表征形式.第三列为被分析对象的粒度,包括源代码、二进制代码以及函数块,其中源代码又分别由 C/C++、Java、PHP 等编程语言编写.如果不指明编程语言的源代码,则表示该方法不限定编程语言.第四列为该项研究提取的特征,即依据表征形式

从被分析对象中抽取哪些特征来表示源数据.第五列为该研究采用的机器学习方法,主要包括 SVM、逻辑回归、朴素贝叶斯与 BLSTM 等.第六列为将代码表征转换成向量形式的方法,包括词袋模型、TF-IDF、统计量化以及 word2vec 等.如果有特征选择的过程,则第六列的括号内为特征选择所用的方法,包括基于相关性的特征选择法、包装法、主成分分析三种方法.第七列代表该项研究是否实现跨项目,即是否能通过有效迁移源项目的相关知识为目标项目构建预测模型[32].第八列代表该项研究的研究目标针对何种类型的漏洞,若该项为不限,则表示该项研究的目标为构建通用漏洞检测模型,不局限于某种特定漏洞类型.通过梳理以上具有代表性的研究工作,我们可以做出以下总结与分析:

(1) 对于研究目标而言,基于机器学习的漏洞挖掘方法的研究目标大体分两类,一类是构建通用漏洞检测模型,这种模型不会限制检测到的漏洞的类型,是漏洞类型无关的.但是这类模型的检测结果往往只能提供潜在的可能性,一般可以认定为软件缺陷,但并不保证检测到的缺陷一定是安全漏洞;另一类则针对如缓冲区溢出、内存破坏等特定类型的漏洞,这类模型在漏洞模式的分析及特征选取时更有针对性.但是由于只关注一类或几类漏洞,构建模型时会过滤掉无关的属性,导致模型泛化性较差.若需要挖掘其余类型的漏洞,还需要再次对漏洞模式加以分析,并对模型进行重建.

(2) 对于被分析对象而言,现有研究集中于通过对代码的分析,得到含漏洞代码和不含漏洞代码在指定特征表达上的差别,进而得到含漏洞代码的模式.由表 2 可以看出,在代码分析中对源代码的分析占比更大,其中大多数模型以 C/C++语言[26,31]为分析对象,也有对 PHP、Java 以及 Python 等编程语言[65]进行分析的.虽然在恶意攻击中二进制代码才是真正执行的部分,理论上直接分析二进制代码会得到更有说服力的结果.但是直接分析二进制代码的研究[57,75]占比重小可能是由于二进制代码的可读性差,研究者难以从中发现含漏洞代码的一般规律并据此选取合适的特征,分析面临更大的挑战.除了对代码的分析,少数研究[26]对提交和变更等非直接相关的信息进行尝试,也取得不错的检测效果.

(3) 对于提取程序特征的方式而言,虽然直观上代码文本自身能够带来最多的程序相关信息,但是由表 2 可以看出,基于代码度量的研究也占很大比重.文献[76]将基于代码度量的漏洞挖掘模型与基于文本挖掘的模型进行了比较,实验结果也表明基于文本挖掘的模型比基于代码度量的模型具有更高的召回率.文献[77]则认为上述实验得出结论时没有考虑单个组件的大小,影响了结论的正确性.也通过实验说明了实际应用中基于文本和代码度量的模型能取得相同的效果,甚至基于代码度量的模型所需成本更低,是实际生产中的更优选择.通过分析,本文也发现基于文本挖掘和代码度量的漏洞检测研究都在不断地发展,面向各自针对的研究问题取得有价值的研究成果.

(4) 对于机器学习模型的选择,表 2 中使用传统机器学习模型[26,33,45]的比例约占 80%,如逻辑回归、朴素贝叶斯、随机森林、SVM 等,近几年的研究逐渐引入了 CNN、LSTM、BLSTM 等在 NLP 领域取得较好效果的模型[31,32,64].漏洞挖掘需要依赖上下文环境,这与 NLP 的需求相同,所以借鉴在 NLP 领域较为成功的模型能挖掘更深层次的代码特征,从而更加细粒度地发现漏洞.关于分类模型对于预测性能的影响,Lessmann 等人[78]在 NASA 语料库中的 10 个公共数据集上,对 22 种分类模型的性能进行比较,研究表明分类模型的选择对于软件缺陷预测模型性能的影响差别不大.而 2015 年 Ghotra 等人[79]通过在 PROMISE 语料库的研究中得到不一样的观点,发现分类模型的选择会对性能产生一定的影响.但是鉴于实证研究都存在一定的局限性,因此建议研究人员使用各种可用的技术进行实验,而不是依赖特定的技术.漏洞预测问题比缺陷预测问题更加复杂,因此本文认为在漏洞预测问题中分类模型对于预测模型性能的影响应该更加复杂,更难得出一个概括性的结论.通过分析相关研究本文也发现,2015 年以及之前的研究基本都只选用一种分类器,2015 年之后的相关研究大多使用多种分类器,实验部分评估并选取当前应用场景下效果最佳的分类器.

(5) 对于实际应用场景而言,大多数现有研究[31,33,45,50]局限于项目内的漏洞检测,少数[32]可以实现跨项目检测.文献[80]提出了一个实证研究,说明跨项目检测技术在软件漏洞挖掘应用场景中的重要性.2012 年 Ma 等人[81]提出了迁移朴素贝叶斯模型,用于解决跨公司的软件缺陷检测问题,也为之后跨项目漏

洞检测问题提供了极大参考价值.之后许多研究也对跨项目检测进行尝试,如 2014 年文献[48,76]在实验设计中也对跨项目的效果进行了检测,但是检测效果并不理想.文献[82]比较了复杂度和耦合代码度量在跨项目脆弱性检测中的能力,提出单元复杂度和包含脆弱头文件(Included Vulnerable Header,简称 IVH)度量指标是跨项目漏洞检测场景的最佳指标集.文献[32]首次将迁移学习引入基于机器学习的漏洞检测问题中,在跨项目场景中取得较好的效果.但是由于跨项目检测时训练集和测试集的数据分布不同,影响传统机器学习方法的性能,所以现有的研究多数都面向项目内检测.

通过对各项具有代表性的研究工作的总结与对比,可以发现近年来基于机器学习的漏洞挖掘研究成为安全领域的研究热点.安全研究人员通过对漏洞的形式、产生原理的深入分析,根据研究目标构造出合适的代码表征形式,从而对漏洞的模式进行构造,接着利用机器学习算法构建漏洞挖掘模型进行漏洞挖掘,取得了一系列突破性的成果.其主要体现在漏洞检测更加细粒度,从早期的文件级[83]和函数级[45,47,50],到如今已有研究实现以代码段为粒度[31,54],从而实现漏洞定位;代码表征更加有效,引入深度学习模型使得特征中能蕴含代码更深层次的信息[35,40],从而扩展特征的深度.与此同时,考虑除代码本身外的开发文档、开发者情况等,不断扩大特征的选取范围[37,43],从而扩展特征的广度;适用场景更具普适性,从仅实现项目内检测[31,33,45,51],到引入迁移学习实现跨项目检测[32];机器学习模型的选择更具针对性,从如 SVM、逻辑回归等经典的分类模型[26,33,45],到更符合漏洞挖掘场景的 LSTM、BLSTM 等深度学习模型[31,32].这一系列在漏洞挖掘研究中提出的新理论、新技术不断扩展漏洞挖掘的能力.

4.2 问题与展望

智能化漏洞挖掘能够更好应对层出不穷的新型漏洞,以及大型复杂软件系统的应用场景,并且可以减少人力的投入以及由于专家主观性、分析不完备带来的误报和漏报,因此本文认为采用智能的方法解决漏洞挖掘问题是一项非常有意义的工作.

通过分析现有研究成果,本文发现在基于机器学习的漏洞挖掘领域中,虽然已经取得许多突破性进展,但是现阶段发展得并不成熟,还面临着以下挑战:

- **准确数据集的获取.**机器学习方法非常依赖数据,但是目前各项研究都依赖于各自构建的数据集,尚未建立可以作为基准的开源数据集.2019 年的一项研究[84]通过案例证实 NVD/CVE 数据库中信息不一致的现象非常普遍,即存在声明漏洞版本过高或过低的问题.Jimenez 等人[85]对 1 898 个真实漏洞进行了全面的实证研究,结果表明当使用足够且准确的标记数据进行训练时,脆弱性预测确实具有良好的性能.然而不真实的标签会严重误导科学结论,当只考虑到实际的标记时,高度有效的预测结果将失效.
- **罕见漏洞的挖掘.**由于部分类型的漏洞并不是频繁被利用,因此可用于训练的数据可能较少.根据“长尾效应”[86],这类漏洞不被挖掘出来始终会对软件的安全使用造成影响.如何在训练数据稀缺的情况下有效挖掘该漏洞也是一个需要探讨的话题.
- **程序判定是否有漏洞问题的不可判定性.**由 Rice 定理可知判断程序是否存在漏洞是不可判定的,现有的解决方案是在静态分析检测后,再通过动态分析验证结果.但这种方法显然不适用于大规模软件场景,如何才能确保这类不可判定性问题的结果具有可信度也需要进一步研究.
- **逻辑漏洞的挖掘方式.**逻辑漏洞是由于程序逻辑不严谨或业务流程中逻辑太过复杂,导致一些逻辑分支不能正常处理而出现的.现有研究中关于代码特征的构建极少考虑到业务流程,从而很难挖掘到逻辑漏洞.2019 年的一项研究[87]通过 NLP 技术处理程序的开发文档来对逻辑漏洞进行预测,而这项研究还处于初期,只关注于缺少安全检查的支付服务,并且预测效果受开发文档写作质量的影响.本文认为如何构建能精确表达业务逻辑的代码特征,从而检测业务逻辑漏洞也需要进一步研究.
- **编程语言的多样性.**编程语言的类型庞杂且更新迭代较快,不同语言具有不同的语法规则.如何能在出现新兴语言的时候快速适配现有模型,也存在一定的挑战.
- **跨项目和类不平衡场景下的检测.**2018 年 Ban 等人[88]针对每个数据集的单类漏洞和多类漏洞场景、

跨项目检测场景,以及不同不平衡比例的类不平衡场景,对基于机器学习的软件漏洞检测解决方案进行性能评估.实验结果表明,基于机器学习的软件漏洞检测是可行的,但是应用在跨项目和类不平衡场景中,性能普遍较差.因此本文认为跨项目开发与类不平衡场景中的检测仍存在极大挑战,需要进一步研究.

虽然基于机器学习的漏洞挖掘领域现阶段的发展仍存在许多挑战,但是新的挑战也产生了新的机遇.结合近几年的研究热点,我们认为基于机器学习的漏洞挖掘领域未来的重点研究方向如下:

- **构建统一且规范的开源漏洞数据集.**首先是需要构建统一的、涵盖各种类型漏洞的开源漏洞数据集,基于规范的数据集,才能科学地对现有研究进行全方位评估,从而更好地推进该领域的研究.
- **构建更符合漏洞检测场景的机器学习模型.**现有研究工作大多使用有监督学习方法,但是数据集的标记需要很大的工作量,无监督学习将相似的样本聚类,又会浪费已标记的样本数据.文献[28]提出的研究结果表明,半监督学习利用少量的已标记数据和大量的未标记数据集,非常符合漏洞检测的应用场景.本文认为未来在机器学习模型的选择方面,可以对半监督学习和强化学习、以及表征学习[89]模型进行更多尝试.并且现有研究工作大多使用成熟机器学习方法,Votipka 等人[90]深入调研了软件测试人员与黑客在漏洞识别任务中构成的生态,认为具有不同经验和安全概念基础的两个群体可以互相作用,实现可能更好的漏洞识别效果.受到该研究调研结论以及近年来生成对抗网络(Generative Adversarial Network,简称 GAN)在计算机视觉领域取得令人惊艳的效果的启发,本文认为结合 GAN 的思想改进应用于漏洞挖掘领域的机器学习模型,或许将取得不错的效果.
- **基于机器学习的动静态综合分析模型**.现有的动静态分析结合的模型通常将静态分析检测与动态分析验证结合形成一个完整的模型,即静态分析检测的结果输出后不直接起作用,而是需要先通过符号验证、模糊测试等动态分析方法进行验证,确保结果的准确性.但是现有模型[45]通常只是在静态分析检测阶段引入机器学习模型,为了应对大规模软件应用场景的需求,本文认为未来两个阶段都应该基于机器学习方法进行实现,并可以考虑建立在分析过程中即实现动静态综合分析的机器学习模型.
- **扩大被分析对象的范围.**随着开源生态的建立,软件开发已经处于一个十分开放的环境,因此本文认为检测范围不应只局限在代码本身能包含的信息,未来应充分利用开源生态中丰富的漏洞相关信息来评估漏洞.如 Chen 等人[91]通过 Twitter 讨论数据来预测一个漏洞何时将被利用的框架,而不使用由美国国家标准与技术研究院(National Institute of Standards and Technology,简称 NIST)发布的 CVSS 评分信息.
- **充分挖掘图表征形式的优势.**图的表征形式包含丰富的语法及语义信息,但是局限于图匹配算法的时间和空间复杂性过高,近年来基于图表征形式的相关研究并未取得很大突破.本文认为可以与图匹配和图挖掘领域[92]的最新研究成果进行深度结合,对基于图的表征形式进行尝试,也许可以取得不错的效果.
- **关注跨项目场景.**由于现在收集的漏洞数据集数量有限,但是机器学习初期训练的时候需要大量数据,使用迁移学习可以实现跨项目检测,并且解决机器学习的冷启动问题.现在已经有一些研究[32]对跨项目漏洞挖掘进行尝试,由于跨项目检测才真正符合实际应用的需求,本文认为跨语言和跨项目的漏洞挖掘也将是未来的研究热点.由于缺陷预测关注于预测可能存在缺陷的软件实体,其被分析对象和研究思路与漏洞挖掘存在共性,本文认为可以借鉴缺陷预测领域中跨项目的相关研究[93,94]的解决思路.

5 结语

机器学习方法已经成功应用在许多实际场景中,在漏洞挖掘场景中,利用机器学习可以缓解传统的漏洞挖掘方法耗费大量的人力以及误报率和漏报率较高的问题,同时适用于大规模复杂软件的应用场景.本文通过梳理近十年来国内外将机器学习技术应用到漏洞挖掘的相关研究,归纳了基于机器学习的漏洞挖掘研究的技术特征与工作流程.接着以代码的表征形式为分类依据对现有研究进行分类阐述,并做出系统性的对比分析,最后

总结并探讨了现有研究中存在的挑战和未来的发展展望.

通过总结相关研究,可以看出基于机器学习的漏洞挖掘技术在近十年来发展迅速,取得了相当具有竞争力的性能和效果,也成为网络安全领域的研究热点之一.随着近年来深度学习的广泛应用,也出现一些将深度学习应用到漏洞挖掘的研究,在检测粒度和检测效果方面得到了一定提升,并且在尝试解决跨项目的检测问题,成为漏洞挖掘研究的发展方向之一.同时随着迁移学习、强化学习、知识图谱等技术的成熟,也可以逐步克服现有基于机器学习的漏洞挖掘技术的局限性,进一步提升模型的效果.

References:

[1] Ministry of industry and information technology of people's republic of china. 2018 (in Chinese with English title). http://www.miit.gov.cn/

[2] Linux. 2018. https://github.com/torvalds/linux

[3] CVE details. 2019. https://www.cvedetails.com/

[4] Wu SZ. Review and outlook of information security vulnerability analysis. Journal of Tsinghua Univ (Science and Technology), 2009, 49(S2): 2065–2072 (in Chinese with English abstract). [doi: 10.16511/j.cnki. qhdxxb.2009.s2.005]

[5] Wu SZ, Guo T, Dong GW, Wang JJ. Software vulnerability analyses: a road map. Journal of Tsinghua University Science and Technology, 2012, 52(10): 1309–1319 (in Chinese with English abstract). [doi: 10.16511/j.cnki.qhdxxb.2012.10.001]

[6] Coverity: Coverity Scan Static Analysis. 2020. https://scan.coverity.com/

[7] KlockWork: Static Code Analysis For C, C++, C#, and Java. 2020. https://www.perforce.com/products/klocwork

[8] Gao Q, Zhang S, Chen X, Ma S, Shao S, Sui Y, Zhao G, Ma L, Ma X, Duan F, Deng X. CoBOT: static c/c++ bug detection in the presence of incomplete code. In Proceedings of the 26th Conference on Program Comprehension (ICPC'18). Association for Computing Machinery, New York, NY, USA, 385-388. [doi: 10.1145/3196321.3196367]

[9] Cadar C, Dunbar D, Engler D R. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. OSDI. 2008, 8: 209-224.

[10] Chipounov V, Kuznetsov V, Candea G. S2E: a platform for in-vivo multi-path analysis of software systems. Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems. ACM, 2011.. [doi: 10.1145/2248487.1950396]

[11] Cha SK, Avgerinos T, Rebert A, Brumley D. Unleashing mayhem on binary code. 2012 IEEE Symposium on Security and Privacy. IEEE, 2012: 380–394. [doi: 10.1109/SP.2012.31]

[12] LibFuzzer: A library for coverage-guided fuzz testing. 2020. http://llvm.org/docs/LibFuzzer.html.

[13] Vimpari M. An evaluation of free fuzzing tools. Master's Thesis, University of Oulu, 2015.

[14] AFL: american fuzzy lop. 2020. https://lcamtuf.coredump.cx/afl/

[15] Song CX, Wang X, Zhang WZ. Anlysis and optimization of angr in dynamic software test application. Computer Engineering&Science, 2018, 40(S1): 167–172 (in Chinese with English abstract). [doi: 10.3969/j.issn.1007-130X.2018.Suppl (1).030]

[16] Godefroid P, Levin MY, Molnar D. SAGE: whitebox fuzzing for security testing. Communications of the ACM, 2012, 55(3): 40–44. [doi: 10.1145/2093548.2093564]

[17] Bochspwn. 2020. https://github.com/googleprojectzero/bochspwn

[18] Pan J, Yan G, Fan X. Digtool: A Virtualization-Based Framework for Detecting Kernel Vulnerabilities. 26th USENIX Security Symposium (USENIX Security 17). 2017: 149-165. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/spresentation/pan

[19] Syzkaller. 2020. https://github.com/google/syzkaller

[20] Rapidscan. 2020. https://github.com/skavngr/rapidscan

[21] Zhang X, Li ZJ. Survey of fuzz testing technology. Computer Science, 2016, 43(5): 1-8 (in Chinese with English abstract). [doi:

10.11896/j.issn.1002-137X.2016.5.001]

[22] Ye ZB, Yan B. Survey of symbolic execution. Computer Science, 2018, 45(s1): 28-35 (in Chinese with English abstract). [doi: CNKI:SUN:JSJA.0.2018-S1-006]

[23] 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 and Technology), 2018, 58(12): 45-60 (in Chinese with English abstract). [doi: 10.16511/j.cnki.qhdxxb.2018.21.025]

[24] Hindle A, Barr ET, Su Z, Gabel M, Devanbu P. On the naturalness of software. 2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012: 837–847. [doi: 10.1109/ICSE.2012.6227135]

[25] SUN HY, HE Y, WANG JC, DONG Y, ZHU LP, WANG H, ZHANG 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). [doi: 10.11959/j.issn.1000–436x.2018137]

[26] Perl H, Dechand S, Smith M, Arp D, Yamaguchi F, Rieck K, Fahl S, Acar Y. VCCFinder: finding potential vulnerabilities in open-source projects to assist code audits. Acm Sigsac Conference on Computer & Communications Security. 2015. [doi: http://dx.doi.org/10.1145/2810103.2813604]

[27] Agrawal A, Menzies T. Is "better data" better than "better data miners"?: on the benefits of tuning smote for defect prediction. Proceedings of the 40th International Conference on Software engineering. ACM, 2018: 1050–1061. [doi: https://doi.org/10.1145/3180155.3180197]

[28] Meng Q, Shameng W, Chao F, Chaojing T. Predicting buffer overflow using semi-supervised learning. 2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI). IEEE, 2016: 1959–1963. [doi: 10.1109/CISP-BMEI.2016.7853039]

[29] Dam HK, Tran T, Pham T, Ng SW, Grundy J, Ghose A. Automatic feature learning for vulnerability prediction. arXiv preprint arXiv:1708.02368, 2017.

[30] Yamaguchi F, Maier A, Gascon H, Rieck K. Automatic inference of search patterns for taint-style vulnerabilities. IEEE Symposium on Security & Privacy. 2015. [doi: 10.1109/SP.2015.54]

[31] Zhen L, Zou D, Xu S, Ou X, Hai J, Wang S, Deng Z, Zhong Y. VulDeePecker: a deep learning-based system for vulnerability detection. The 25th Annual Network and Distributed System Security Symposium(NDSS). 2018. [doi: http://dx.doi.org/10.14722/ndss.2018.23158]

[32] Lin G, Zhang J, Luo W, Pan L, Xiang Y, Vel OD, Montague P. Cross-project transfer representation learning for vulnerable function discovery. IEEE Transactions on Industrial Informatics, 2018, 14(7): 1–1. [doi: 10.1109/TII.2018.2821768]

[33] Younis A, Malaiya YK, Anderson C, Ray I. To fear or not to fear that is the question: code characteristics of a vulnerable function with an existing exploit. Conference on Data & Applications Security & Privacy. 2016. [doi: 10.1145/2857705.2857750]

[34] Shin Y, Williams L. Is complexity really the enemy of software security? Acm Workshop on Quality of Protection. 2008. [doi: 10.1145/1456362.1456372]

[35] Shin Y, Williams L. An empirical model to predict security vulnerabilities using code complexity metrics. International Symposium on Empirical Software Engineering & Measurement. 2008. [doi: 10.1145/1414004.1414065]

[36] Bass L, Clements P, Kazman R. Software architecture in practice. United States: Addison-Wesley, 2003.

[37] Fenton N, Bieman J. Software metrics: a rigorous and practical approach, Third Edition. CRC press, 2014.

[38] Chowdhury I, Zulkernine M. Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities? Proceedings of the 2010 ACM Symposium on Applied Computing. New York, NY, USA: ACM, 2010: 1963–1969. [doi: 10.1145/1774088.1774504]

[39] Chowdhury I, Zulkernine M. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. Journal of Systems Architecture, 2011, 57(3): 294–313. [doi: 10.1016/j.sysarc.2010.06.003]

[40] Zimmermann T, Nagappan N, Williams LA. Searching for a needle in a haystack: predicting security vulnerabilities for windows vista. International Conference on Software Testing. 2010. [doi: 10.1109/ICST.2010.32]

[41] Morrison P, Herzig K, Murphy B, Williams L. Challenges with applying vulnerability prediction models. Proceedings of the 2015 Symposium and Bootcamp on the Science of Security. ACM, 2015: 4. [doi: http://dx.doi.org/10.1145/2746194.2746198]

[42] Bozorgi M, Saul LK, Savage S, Voelker GM. Beyond heuristics: learning to classify vulnerabilities and predict exploits. Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2010: 105–114. [doi: 10.1145/1835804.1835821]

[43] Wang FX, Li F. Software vulnerability automatic classification framework based on activation vulnerability conditions. Journal of Chongqing University of Technology(Natural Science), 2019(5): 154-160 (in Chinese with English abstract). [doi: 10.3969/j.issn. 1674-8425(z).2019.05.025]

[44] Rice HG. Classes of recursively enumerable sets and their decision problems. Transactions of the American Mathematical Society, 1953, 74(2): 358–366. [doi: 10.2307/1990888]

[45] Meng Q, Zhang B, Feng C, Tang C. Detecting buffer boundary violations based on svm. 2016 3rd International Conference on Information Science and Control Engineering (ICISCE). IEEE, 2016: 313–316. [doi: 10.1109/ICISCE.2016.76]

[46] Yamaguchi F, Lindner F, Rieck K. Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning. In Proceedings of the 5th USENIX Workshop on Offensive Technologies (WOOT). USENIX Association, 2011, pp. 13-13.

[47] Li R, Feng C, Zhang X, Tang C. A lightweight assisted vulnerability discovery method using deep neural networks. IEEE Access, 2019, 7: 80079–80092. [doi: 10.1109/ACCESS.2019.2923227]

[48] Scandariato R, Walden J, Hovsepyan A, Joosen W. Predicting vulnerable software components via text mining. IEEE Transactions on Software Engineering, 2014, 40(10): 993–1006. [doi: 10.1109/TSE.2014.2340398]

[49] Bindu Madhavi Padmanabhuni HBKT. Predicting buffer overflow vulnerabilities through mining light-weight static code attributes. 2014 IEEE International Symposium on Software Reliability Engineering Workshops. IEEE, 2014: 317–322. [doi: 10.1109/ISSREW.2014.26]

[50] Pang Y, Xue X, Wang H. Predicting vulnerable software components through deep neural network. Proceedings of the 2017 International Conference on Deep Learning Technologies. ACM, 2017: 6–10. [doi: 10.1145/3094243.3094245]

[51] Chernis B, Verma R. Machine learning methods for software vulnerability detection. Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics. ACM, 2018: 31–39. [doi: https://doi.org/10.1145/3180445.3180453]

[52] 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. 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA). 2018: 757–762. [doi: 10.1109/ICMLA.2018.00120]

[53] LI Z, ZOU DQ, WANG ZL, JIN H. Survey on static software vulnerability detection for source code. Chinese Journal of Network and Information Security, 2019, 5(01): 5-18 (in Chinese with English abstract). [doi: 10.119590.issn.2096-109x.2019001]

[54] Zou D, Wang S, Xu S, Li Z, Jin H. MVulDeePecker: a deep learning-based system for multiclass vulnerability detection. IEEE Transactions on Dependable and Secure Computing, 2019: 1–1. [doi: 10.1109/TDSC.2019.2942930]

[55] Zuo F, Li X, Young P, Luo L, Zeng Q, Zhang Z. Neural machine translation inspired binary code similarity comparison beyond function pairs. Network and Distributed Systems Security (NDSS) Symposium 2019, 2019. [doi: https://dx.doi.org/10.14722/ndss.2019.23492]

[56] Wu F, Wang J, Liu J, Wang W. Vulnerability detection with deep learning. 2017 3rd IEEE International Conference on Computer and Communications (ICCC). 2017: 1298–1302.

[57] Grieco G, Grinblat GL, Uzal L, Rawat S, Feist J, Mounier L. Toward large-scale vulnerability discovery using machine learning. Acm Conference on Data & Application Security & Privacy. 2016. [doi:http://dx.doi.org/10.1145/2857705.2857720]

[58] Shen Y, Mariconti E, Vervier PA, Stringhini G. Tiresias: predicting security events through deep learning. Proceedings of the 2018

ACM SIGSAC Conference on Computer and Communications Security. ACM, 2018: 592–605. [doi: https://doi.org/10.1145/3243734.3243811]

[59] Zhu B, Zheng H, Sun LL, Yang YX. Research on similarity measure for ast-based program codes. Journal of Jilin University, 2015, 33(1): 99-104 (in Chinese with English abstract). [doi: 10.3969/j.issn.1671-5896.2015.01.016]

[60] Yamaguchi F, Lottmann M, Rieck K. Generalized vulnerability extrapolation using abstract syntax trees. Computer Security Applications Conference. 2012. [doi: 10.1145/2420950.2421003]

[61] Lin G, Zhang J, Wei L, Lei P, Yang X. POSTER: vulnerability discovery with function representation learning from unlabeled projects. Acm Sigsac Conference. 2017. [doi: https://doi.org/http://dx.doi.org/10.1145/3133956.3138840]

[62] Medeiros I, Neves NF, Correia M. Automatic detection and correction of web application vulnerabilities using data mining to predict false positives. International Conference on World Wide Web. 2014. [doi: http://dx.doi.org/10.1145/2566486.2568024]

[63] Kratkiewicz KJ. Evaluating static analysis tools for detecting buffer overflows in c code. HARVARD UNIV CAMBRIDGE MA, 2005.

[64] Mou L, Ge L, Zhi J, Lu Z, Tao W. Convolutional neural network over tree structures for programming language processing. Thirtieth Aaai Conference on Artificial Intelligence. 2016. [doi: 10.13140/RG.2.1.2912.2966]

[65] Anbiya DR, Purwarianti A, Asnar Y. Vulnerability detection in php web application using lexical analysis approach with machine learning. 2018 5th International Conference on Data and Software Engineering (ICoDSE). IEEE, 2018: 1–6.

[66] Peng S, Liu P, Han J. A python security analysis framework in integrity verification and vulnerability detection. Wuhan University Journal of Natural Sciences, 2019, 24(2): 141–148. [doi: https://doi.org/10.1007/s11859-019-1379-5]

[67] Liu YL. Token-based structured code matching homology detection technology. Application Research of Computers, 2014, 31(6): 1841-1845 (in Chinese with English abstract). [doi: 10.3969/j.issn.1001-3695.2014.06.057]

[68] Cheng H, Lo D, Zhou Y, Wang X, Yan X. Identifying bug signatures using discriminative graph mining. Eighteenth International Symposium on Software Testing & Analysis. 2009. [doi: 10.1145/1572272.1572290]

[69] Nguyen H, Tran LMS. Predicting vulnerable software components with dependency graphs. 2013. [doi: 10.1145/1853919.1853923]

[70] Yamaguchi F, Golde N, Arp D, Rieck K. Modeling and discovering vulnerabilities with code property graphs. Security & Privacy. 2014. [doi: 10.1109/SP.2014.44]

[71] Qian F, Zhou R, Xu C, Yao C, Testa B, Yin H. Scalable graph-based bug search for firmware images. Acm Sigsac Conference on Computer & Communications Security. 2016. [doi: http://dx.doi.org/10.1145/2976749.2978370]

[72] Xu X, Liu C, Feng Q, Yin H, Song L, Song D. Neural network-based graph embedding for cross-platform binary code similarity detection. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017: 363–376.[doi: http://dx.doi.org/10.1145/3133956.3134018]

[73] Dai H, Dai B, Song L. Discriminative embeddings of latent variable models for structured data. International conference on machine learning. 2016: 2702–2711. [doi: 1603.05629]

[74] Harer JA, Kim LY, Russell RL, Ozdemir O, Kosta LR, Rangamani A, Hamilton LH, Centeno GI, Key JR, Ellingwood PM, Antelman E, Mackay A, McConley MW, Opper JM, Chin P, Lazovich T. Automated software vulnerability detection with machine learning. CoRR, vol.abs/1803.04497, 2018.

[75] Liu DJ, Li Y, Tang Y, Wang BS, Xie W. VMPBL: Identifying Vulnerable Functions based on Machine Learning Combining Patched Information and Binary Comparison Technique by LCS. 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications. 2018. [doi: 10.1109/TrustCom/BigDataSE.2018.00114]

[76] Walden J, Stuckman J, Scandariato R. Predicting vulnerable components: software metrics vs text mining. 2014 IEEE 25th international symposium on software reliability engineering. IEEE, 2014: 23–33. [doi: 10.1109/ISSRE.2014.32]

[77] Tang Y, Fei Z, Yang Y, Lu H, Xu B. Predicting vulnerable components via text mining or software metrics? an effort-aware

perspective. IEEE International Conference on Software Quality. 2015. [doi: 10.1109/QRS.2015.15]

[78] Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE Transactions on Software Engineering, 2008, 34(4): 485–496. [doi: 10.1109/TSE.2008.35]

[79] Ghotra B, McIntosh S, Hassan AE. Revisiting the impact of classification techniques on the performance of defect prediction models. Proceedings of the 37th International Conference on Software Engineering - Volume 1. Piscataway, NJ, USA: IEEE Press, 2015: 789–800. [doi: 10.1109/ICSE.2015.91]

[80] Abunadi I, Alenezi M. Towards cross project vulnerability prediction in open source web applications. International Conference on Engineering & Mis. 2015. [doi: http://dx.doi.org/10.1145/2832987.2833051]

[81] Ying M. Transfer learning for cross-company software defect prediction. Information & Software Technology, 2012, 54(3): 248. [doi: 10.1016/j.infsof.2011.09.007]

[82] Moshtari S, Sami A. Evaluating and comparing complexity, coupling and a new proposed set of coupling metrics in cross-project vulnerability prediction. Acm Symposium on Applied Computing. 2016. [doi: http://dx.doi.org/10.1145/2851613.2851777]

[83] Shin Y, Meneely A, Williams L, Osborne JA. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Transactions on Software Engineering, 2011, 37(6): 772–787. [doi: 10.1109/TSE.2010.81]

[84] Dong Y, Guo W, Chen Y, Xing X, Zhang Y, Wang G. Towards the detection of inconsistencies in public security vulnerability reports. 28th USENIX Security Symposium (USENIX Security 19). 2019: 869–885. https://www.usenix.org/conference/usenixsecurity19/presentation/dong

[85] Jimenez M, Rwemalika R, Papadakis M, Sarro F, Le Traon Y, Harman M. The importance of accounting for real-world labelling when predicting software vulnerabilities. Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE). 2019. [doi: https://doi.org/10.1145/3338906.3338941]

[86] Anderson C. The long tail: Why the future of business is selling less of more. Hachette Books, 2006.

[87] Chen Y, Xing L, Qin Y, Liao X, Wang X, Chen K, Zou W. Devils in the guidance: predicting logic vulnerabilities in payment syndication services through automated documentation analysis. 28th USENIX Security Symposium (USENIX Security 19). 2019: 747–764. https://www.usenix.org/conference/usenixsecurity19/presentation/chen-yi

[88] Ban X, Liu S, Chen C, Chua C. A performance evaluation of deep-learnt features for software vulnerability detection. Concurrency and Computation: Practice and Experience, 2019: e5103. [doi: 10.1002/cpe.5103]

[89] Jian S, Pang G, Cao L, Lu K, Gao H. Cure: Flexible categorical data representation by hierarchical coupling learning. IEEE Transactions on Knowledge and Data Engineering. 2018 Jun 25;31(5):853-66. [doi: 10.1109/TKDE.2018.2848902]

[90] Votipka D, Stevens R, Redmiles E, Hu J, Mazurek ML. Hackers vs. testers: a comparison of software vulnerability discovery processes. 2018 IEEE Symposium on Security and Privacy (SP). 2018. [doi: 10.1109/SP.2018.00003]

[91] Chen H, Liu R, Park N, Subrahmanian V. Using twitter to predict when vulnerabilities will be exploited. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, 2019: 3143–3152. [doi: https://doi.org/10.1145/3292500.3330742]

[92] Xu J, Zhang QZ, Zhao X, Lv P, Li TS. Survey on dynamic graph pattern matching technologies. Ruan Jian Xue Bao/Journal of Software, 2018, 29(3): 663–688 (in Chinese with English abstract). [doi: 10.13328/j.cnki.jos.005444]

[93] Chen X, Gu Q, Liu WS, Liu SL, Ni C. Survey of static software defect prediction. Journal of Software, 2016, 27(10): 1–25. (in Chinese with English abstract). [doi: 10.13328/j.cnki.jos.004923]

[94] Chen X, Wang LP, Gu Q, Wang Z, Ni C, Liu WS, Wang QP. A Survey on Cross-Project Software Defect Prediction Methods. chinese journal of computers, 2018, 041(001):254-274. (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2018.00254]

附中文参考文献:

[1] 中华人民共和国工业和信息化部. 2018. http://www.miit.gov.cn/

[4] 吴世忠. 信息安全漏洞分析回顾与展望. 清华大学学报(自然科学版), 2009, S2: 2065–2072.

[doi: 10.16511/j.cnki. qhdxxb.2009.s2.005]

[5] 吴世忠, 郭涛, 董国伟, 王嘉捷. 软件漏洞分析技术进展. 清华大学学报(自然科学版), 2012, (10): 1309–1319. [doi: 10.16511/j.cnki.qhdxxb.2012.10.001]

[15] 宋丛溪, 王辛, 张文喆. Angr 动态软件测试应用分析与优化. 计算机工程与科学, 2018, 40(z1): 163–168. [doi: 10.3969/j.issn.1007-130X.2018.Suppl(1).030]

[21] 张雄, 李舟军. 模糊测试技术研究综述. 计算机科学, 2016, 43(5): 1–8. [doi: 10.11896/j.issn.1002-137X.2016.5.001]

[22] 叶志斌, 严波. 符号执行研究综述. 计算机科学, 2018, 45(s1): 28–35. [doi: CNKI:SUN:JSJA.0.2018-S1-006]

[23] 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. 2018, 58(12): 45-60. [doi: 10.16511/j.cnki.qhdxxb.2018.21.025]

[25] 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. [doi: 10.11959/j.issn.1000−436x.2018137]

[43] 王飞雪, 李芳. 基于激活漏洞能力条件的软件漏洞自动分类框架. 重庆理工大学学报(自然科学), 2019(5): 154–160. [doi: 10.3969/j.issn.1674-8425(z).2019.05.025]

[53] 李珍, 邹德清, 王泽丽, 金海. 面向源代码的软件漏洞静态检测综述. 网络与信息安全学报, 2019, 5(01): 5-18. [doi: 10.119590.issn.2096-109x.2019001]

[59] 朱波, 郑虹, 孙琳琳, 杨友星. 基于 AST 的程序代码相似性度量研究. 吉林大学学报(信息科学版), 2015, 33(1): 99–104. [doi: 10.3969/j.issn.1671-5896.2015.01.016]

[67] 刘云龙. 基于 Token 的结构化匹配同源性代码检测技术研究. 计算机应用研究, 2014, 31(6): 1841–1845. [doi: 10.3969/j.issn.1001-3695.2014.06.057]

[92] 许嘉, 张千桢, 赵翔, 吕品, 李陶深. 动态图模式匹配技术综述. 软件学报, 2018, 29(3): 663-688. [doi: 10.13328/ j.cnki.jos.005444]

[93] 陈翔, 顾庆, 刘望舒, 刘树龙, 倪超. 静态软件缺陷预测方法研究. 软件学报, 2016, 27(1): 1–25. [doi: 10.13328/ j.cnki.jos.004923]

[94] 陈翔, 王莉萍, 顾庆, 王赞, 倪超, 刘望舒, 王秋萍. 跨项目软件缺陷预测方法研究综述.计算机学报,2018,41(1): 254-274. [doi: 10.11897/SP.J.1016.2018.00254]