引用格式:陆余良,于璐,赵家振. 软件漏洞智能化挖掘技术研究进展[J]. 信息对抗技术,2023,2(2):1-19. [LU Yuliang, YU Lu, ZHAO Jiazhen. Survey of software vulnerability mining methods based on machine learning[J]. Information Countermeasure Technology, 2023, 2(2):1-19. (in Chinese)]

软件漏洞智能化挖掘技术研究进展

陆余良[1,2*],于 璐[1,2],赵家振[1,2]

(1. 国防科技大学电子对抗学院,安徽合肥 230037;
2. 网络空间安全态势感知与评估安徽省重点实验室,安徽合肥 230037)

摘 要 软件规模的不断扩大和新技术平台的发展对软件漏洞挖掘方法提出了新的挑战。在突破漏洞挖掘技术瓶颈的过程中,研究人员将机器学习方法应用于漏洞挖掘,利用机器学习模型自动学习代码的深层语法和语义规律,以提高漏洞挖掘的智能化水平和有效性,软件漏洞智能化挖掘技术已成为当前研究的热点。围绕软件漏洞智能化挖掘技术的研究展开分析,从静态挖掘和动态挖掘 2 个方面,对机器学习与漏洞挖掘技术结合的研究进行了深入分析。在漏洞智能化静态挖掘方面,从基于代码度量、基于代码模式和基于代码相似性 3 个方面梳理了现有研究工作;在漏洞智能化动态挖掘方面,则分类阐述了机器学习方法与动态挖掘方法结合的相关研究。依据对现有工作的总结,对未来漏洞智能化挖掘的发展趋势进行了展望。

关键词 机器学习;漏洞挖掘;代码特征;静态分析

中图分类号 TP 393 **文章编号** 2097-163X(2023)02-0001-19

文献标志码 A **DOI** 10.12399/j.issn.2097-163x.2023.02.001

Survey of software vulnerability mining methods based on machine learning

LU Yuliang[1,2*], YU Lu[1,2], ZHAO Jiazhen[1,2]

(1. College of Electronic Engineering, National University of Defense Technology, Hefei 230037, China;
2. Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, Hefei 230037, China)

Abstract Software vulnerability is the main cause of various network security events, and it has received continuous and extensive attention from security research institutions, academic groups and enterprises. With the expansion of software scale and the development of new technology, researchers in software vulnerability mining fields are facing new challenges. However, it has been found that applying machine learning model to vulnerability mining can automatically learn the deep syntax and semantic rules of code. This method has been proved to effectively improve the intelligence level and effectiveness of vulnerability mining. In this review, we conducted an extensive and in-depth investigation and analysis of vulnerability mining technology combined with machine learning methods, especially deep learning methods. First, the static vulnerability mining methods based on machine learning were analyzed from three aspects: code metrics, code patterns, and code similarity. Then, the application

收稿日期:2023-01-10 **修回日期**:2023-02-20
通信作者:陆余良,E-mail:luyuliang@nudt.edu.cn
基金项目:国家重点研发计划项目(2016QY07X1501)

of machine learning in the dynamic vulnerability mining was summarized and discussed. Finally, based on the summary of existing research, the challenges of machine learning based vulnerability mining were proposed, and future trends were presented.

Keywords machine learning; vulnerability mining; code characteristics; static analysis

0 引言

随着信息技术的飞速发展,网络空间安全在国家安全中的地位日益提升,软件漏洞作为影响网络空间安全的重要因素,得到了学术界和工业界的广泛关注和研究。根据互联网工作任务组(Internet engineering task force, IETF)的定义[1],漏洞是指在系统设计、完成、操作、管理中出现的缺陷或者弱点,能够被利用以违反系统的安全策略。目前对于软件漏洞的定义尚未形成广泛的共识[2],本文认为软件漏洞是软件系统或者产品的软件生命周期中,在需求、设计、编码、配置、运行等阶段产生的缺陷或者错误。软件漏洞挖掘技术是检查并发现软件中存在的漏洞,提高软件安全性的有效手段之一。

软件漏洞挖掘技术自 20 世纪 70 年代开始引发了研究人员的关注,之后不断发展成熟,由单一化方法向多样化方法演进,挖掘手段由手工挖掘发展到自动化挖掘,并形成了静态分析、动态分析和动静结合的漏洞挖掘方法。其中,静态分析方法通过广义的抽象分析程序属性,无需执行程序,不依赖程序的特定运行环境,且能够对代码进行全覆盖分析,但是该方法存在误报率高的问题,典型的静态分析方法包括基于规则的分析[3]、代码相似性检测[4-5]以及静态符号执行等[6-8]。动态分析方法使用特定的输入数据引导目标程序执行,并监视其运行时的行为发现可能存在的漏洞,该方法能够对程序的属性进行分析和精确定位漏洞代码,误报率低,但需要构建程序执行的依赖环境,且代码覆盖率较低,资源开销大,典型的动态分析方法包括模糊测试[9-10]、动态污点分析[11-12]、动态符号执行[13]等。将动态分析和静态分析方法结合来进行漏洞挖掘,可以利用动态分析方法对静态分析方法中的误报进行甄别,或者利用静态分析方法来辅助动态分析方法的测试用例的生成和分析过程。

传统软件漏洞挖掘方法在对特定类型或者规模较小的程序分析中取得了一定成果,但是在分析大规模复杂软件时,以及随着新技术而产生的变化多样的新类型漏洞时,通常无法满足需求。因此,对漏洞挖掘方法进行优化,突破其技术瓶颈,使其适应于分析更大规模、更加复杂的程序,是漏洞挖掘技术的必然要求。近年来,随着机器学习方法在各个领域的广泛应用,为解决传统漏洞挖掘方法存在的不足,研究人员尝试将机器学习方法应用于漏洞挖掘中,利用机器学习模型从海量数据中学习代码的深层语法和语义规律,可以提高软件漏洞挖掘方法的智能化水平和有效性,当前该研究方向取得了显著成果,已经成为软件漏洞挖掘领域研究的热点之一。

本文重点关注近年来将机器学习,特别是深度学习应用于漏洞挖掘领域,从而提高软件漏洞挖掘智能化水平的相关研究。在广泛调研的基础上,梳理并分析当前软件漏洞智能化静态挖掘和智能化动态挖掘方面的研究进展。其中,漏洞智能化静态挖掘方法研究侧重于对漏洞代码语法语义特征的提取和表征,利用模型对漏洞代码进行分析和检测;漏洞的智能化动态挖掘方法则关注传统动态挖掘方法中存在的技术瓶颈,缓解动态挖掘方法存在的固有问题,如模糊测试的覆盖率低,以及符号执行的约束求解困难等问题,为漏洞动态挖掘方法提供有效的辅助。最后,在总结漏洞智能化挖掘技术的基础上,对漏洞智能化挖掘领域的发展趋势进行了展望。

1 软件漏洞智能化静态挖掘方法

在软件漏洞智能化静态挖掘方法研究中,研究人员将机器学习方法与软件漏洞静态挖掘技术结合,基于代码的静态特征,收集大量代码数据,利用机器学习模型学习代码语法和语义特征,将训练好的机器学习模型应用于漏洞代码的挖掘和检测中。本文根据机器学习模型学习和表征的不同代码特征,将漏洞智能化静态挖掘的研究分为 3 个方向:基于代码度量的漏洞挖掘方法、基于代码模式的漏洞挖掘方法和基于代码相似性的漏洞挖掘方法。

1.1　基于代码度量的漏洞挖掘方法

代码度量是在开发测试程序的过程中用于衡量软件质量的指标，常用于开发人员对项目的状态和进度进行实时跟踪，以识别可能存在的风险。基于代码度量的方法主要针对源码已知的目标程序，利用数据挖掘、机器学习和统计分析技术，对漏洞相关的度量特征进行提取和表征，实现对漏洞代码的检测。基于代码度量的漏洞智能化挖掘方法的一般工作流程如图 1 所示。由图 1 可见，在训练阶段，基于训练数据样本提取代码的度量特征，包括代码行数、函数数目、圈复杂度、最大嵌入复杂度、代码流失等，将代码的度量特征作为机器学习模型的输入，使用监督学习或者非监督学习的方法得到不同代码度量特征在判断漏洞时的不同作用；在检测阶段，在给定目标程序的情况下，基于提取代码度量特征，使用训练好的机器学习模型对目标程序中可能存在的漏洞进行挖掘。

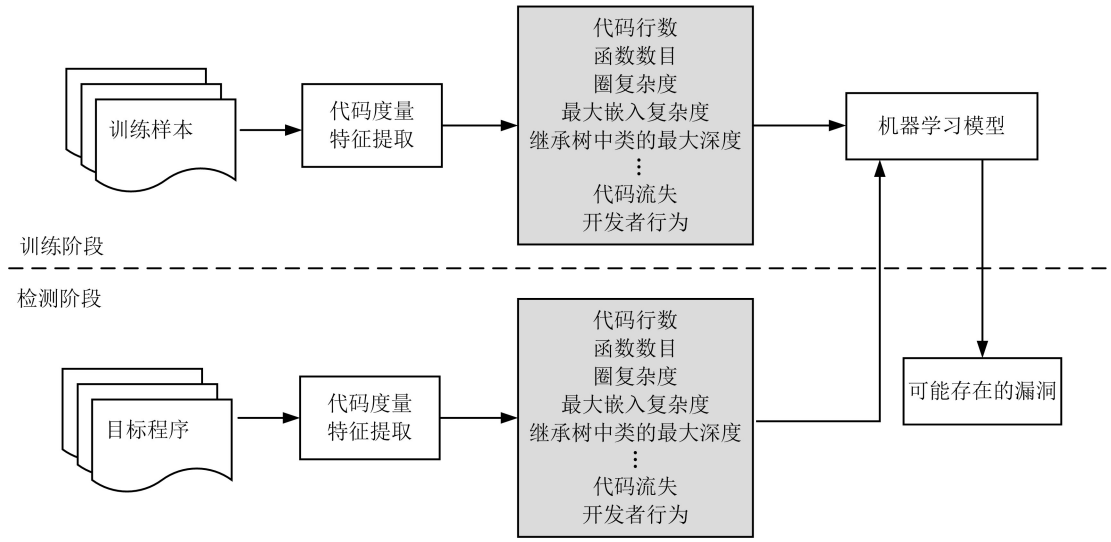

图 1　基于代码度量的漏洞智能化挖掘一般工作流程

Fig. 1　The workflow of intelligent vulnerability mining based on code metrics

SHIN 等[14]使用复杂度、代码流失（code churn）和开发者行为 3 个指标对目标程序进行分析，使用逻辑回归的方法，衡量了 3 个指标与软件漏洞之间的关系。ZIMMERMANN 等[15]对 Windows Vista 进行了大规模的实证研究，表明代码流失、代码复杂性和组织度量等指标可以在低召回率下高精度地挖掘漏洞。

KALOUPTSOGLOU 等[16]针对 PHP 应用软件进行跨项目的漏洞检测。使用随机森林算法对包括代码行数 LOC、非 HTML 代码行数、函数数目、圈复杂度等共 12 种度量特征的重要程度进行衡量，并采用机器学习分类器如支持向量机（SVM）、XGBoost 和深度学习中多层感知机（MLP）来进行漏洞分类和检测。

ZAGANE 等[17]则基于深度学习和软件度量的方法来进行漏洞检测。软件度量方法包括 LOC 度量（代码总行、空行数、注释行数等）、McCabe 度量（圈复杂度）和 Halstead 度量（程序中不同的操作符个数、操作数个数、操作符总数及操作数总数），基于特定漏洞类型进行切片，并收集对应的代码度量特征，最后使用多层感知机来进行度量特征嵌入。

GUPTA 等[18]选择了 3 类常见漏洞，将程序中的代码平均行数、圈复杂度、继承树中类的最大深度作为度量来表征代码。对 32 个监督学习算法在漏洞代码检测的有效性进行评估，得到针对每一种类型漏洞对应的表现最好的学习模型（J48、AdaBoostM1 和 LWL），证明了基于软件度量标准的监督学习方法在发现漏洞中的有效性。

基于代码度量的漏洞挖掘方法虽然能够在特定环境下完成漏洞挖掘，但这种方法的粒度较粗，代码度量指标与代码本身的语法或者语义信息并不相关，且度量标准与项目密切相关，如代码丢失指标与项目成熟度有关；部分开源项目可能只有一个维护者，导致开发者行为可能成为无用指标。因此，这种方法在发现漏洞上存在局限性。但是这些度量指标往往易于获得，构建对应的漏洞预测模型并不需要额外的专业知识，且在

特定场景下的漏洞挖掘表现较好，基于代码度量的漏洞挖掘方法依然是研究者关注的方向之一。

1.2 基于代码模式的漏洞挖掘方法

代码模式是指与代码语法和语义相关的特征和行为，基于代码模式的漏洞检测方法使用预先定义漏洞代码模型对漏洞的特征、表现行为进行描述，然后使用模式匹配的方法对漏洞代码进行定位。将机器学习方法应用于基于代码模式的漏洞挖掘中，借助一定规模漏洞代码数据集，使用机器学习方法来进行漏洞代码模式的提取和学习；根据学习到的漏洞模式，检测目标程序中的漏洞。图 2 给出了基于机器学习模型进行漏洞模式学习，从而挖掘漏洞的一般过程，该过程包含训练和检测 2 个阶段。在训练阶段，基于大量的训练样本对代码属性相关特征进行提取和学习，得到包括抽象语法树(abstract syntax tree，AST)、控制流图(control flow graph，CFG)、数据流图(data flow graph，DFG)等代码语法和语义特征，使用机器学习模型将特征进行向量化并映射到向量空间，通过分类模型对漏洞代码和非漏洞代码进行分类；在检测阶段，给定目标程序，对其代码进行特征提取和向量表征后，使用已经训练好的分类模型判定目标程序中是否存在漏洞。

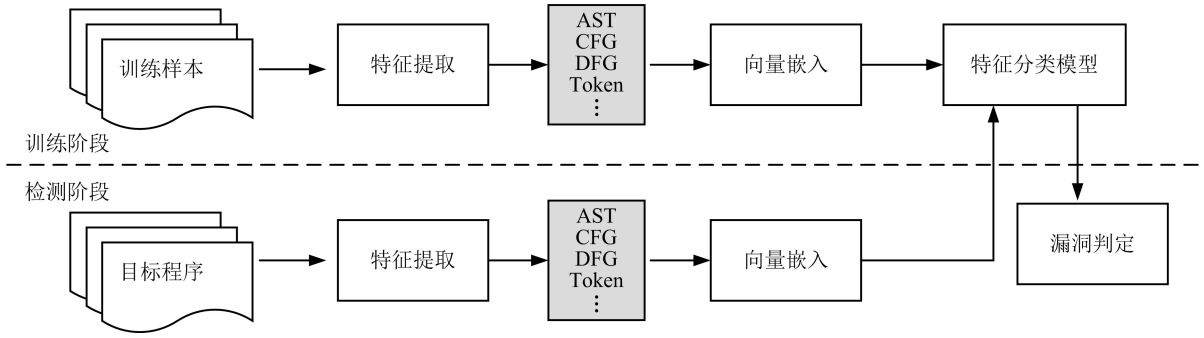

图 2 基于代码模式的漏洞智能化挖掘一般工作流程

Fig. 2 The workflow of intelligent vulnerability mining based on code pattern

从 2011 年开始，研究人员开始关注将机器学习模型应用于代码模式的学习，从而发现漏洞的研究。YAMAGUCHI 等[21-22]提出了一种辅助发现漏洞的方法，引入了“漏洞外推”的概念，旨在根据已知安全漏洞中观察到的编程模式识别未知漏洞。文献[21-22]的工作是针对源码已知的目标程序，之后的研究不仅针对源码已知的程序[23-30]，还有针对源码未知的二进制程序[31-32]进行漏洞模式的学习。根据基于代码模式的漏洞智能化挖掘方法的一般工作流程，本文从代码特征提取、向量嵌入和特征分类模型 3 个方面对现有工作进行梳理。

在代码特征提取上，针对源码漏洞检测的方法多是在抽象语法树的基础上，添加代码的其他序列化特征或者结构特征，用于后续的特征嵌入和表示。PANG 等[23]将源码中的类、对象、方法、参数和变量等作为代码特征，并使用统计假设检验方法对特征进行排序并排除大量不太重要的特征；LI 等[25,30]的研究则是基于数据依赖和控制依赖关系，对库函数/API 函数进行程序切片，提出一种叫作“代码小片段”(code gadget)的结构来表征程序；KRONJEE 等[27]将数据流分析和机器学习结合，使用到达-定值分析、污点分析和到达变量分析来提取代码特征，基于抽象语法树构建控制流图(CFG)作为代码的结构特征；ZHOU 等[28]重点关注代码图的结构，将抽象语法树、控制流图和数据流图以及在抽象语法树上的叶子节点之间添加边后的图结合后生成新图，使用生成的图结构特征来表征代码特征；WANG 等[29]则使用程序的数据流图、控制流图和函数调用图作为代码结构特征。针对二进制程序的漏洞检测方法无法得到富含语义的代码特征的问题；GRIECO 等[31]收集程序中标准 C 库函数的调用序列，并结合程序执行得到的相关跟踪信息作为二进制程序特征；LE 等[32]则直接分别对汇编代码中的操作数和操作码进行特征的提取和处理。

在向量嵌入方法上，当前的研究一般借鉴文本挖掘的方法来进行序列化特征的嵌入，文献[23-24,31]使用 N-gram 文本挖掘技术，而文献[25-26,28-30]使用 word2vec 进行嵌入，文献[32]则使用独热编码(one-hot)方法得到嵌入向量。在表征结构化特征上，ZHOU 等[28]和

WANG 等[29]使用了门控图神经网络（gated graph neural networks，GGNN）来对各自使用的多个图结构进行表征。

在使用机器学习模型对特征进行分类上，研究人员使用一个分类器或者对多个分类器进行比较，如 PANG 等[23]选择 SVM 分类算法来构建预测模型；GRIECO 等[31]使用逻辑回归、MLP 和随机森林进行了比较；KRONJEE 等[27]则使用决策树、随机森林、逻辑回归、朴素贝叶斯和树增强朴素贝叶斯（tree augmented naive Bayes，TAN）5 种分类器来进行分类；ZHOU 等[28]在进行图嵌入后，使用 ReLu 函数和全池化层进行处理后，再使用一层卷积层进行漏洞判断；WANG 等[29]使用 SVM、随机森林、k-近邻、逻辑回归、梯度提升 5 个分类器来进行分类。

可以看出，将机器学习应用于基于代码模式方法的研究主要关注 2 个方面：一是在特征的提取方法上；二是在使用不同的机器学习模型对漏洞代码模式的学习上。在特征提取方面，针对源码已知的程序和源码未知的二进制程序的代码特征提取方法不同，包括常规代码解析、静态数据流和控制流分析、程序源码文本挖掘等，代码的表示也有所不同，如有表示代码结构的图特征、树特征，也有扁平化的代码序列特征。除了代码序列化语义特征，代码的结构特征同样能够表征代码的语义，因此，近期的研究都将代码结构特征作为代码语义的重要组成进行研究。在学习模型选择上，研究人员也开始使用多个模型进行训练学习和比较，找到特定条件下表现最优的模型。基于模式匹配的漏洞检测方法对代码语义特征的依赖性更强，而源码已知的程序包含了更加丰富的语义，对其代码特征进行学习后得到的模型可以更加有效地检测漏洞代码，因此相对源码未知的二进制程序，基于代码模式的方法在源码已知程序的研究更为成熟。

1.3　基于代码相似性的漏洞挖掘方法

代码相似性比较技术借助传统程序分析技术，对代码的特征进行提取，分析和判断 2 个程序中的代码片段是否相似。传统基于代码相似性比较的漏洞挖掘方法提取目标代码的关键特征，包含程序的控制流图、数据流图或者执行序列，并使用图同构[33]、树编辑距离[9]、执行迹距离[34]、最长公共子序列[35]、图编辑距离[36]来判断代码是否相似，基于目标代码与漏洞代码的相似程度检测目标代码中是否存在漏洞，该方法虽然可以对代码进行相似性度量，但是存在效率不高的问题。将代码相似性比较方法与机器学习模型相结合，将代码特征转为向量空间的表示，使用向量空间的距离计算要比传统的图同构、图编辑距离等方法的效率高，近年来受到了研究人员的广泛关注。将机器学习方法应用于代码相似性比较技术从而检测漏洞的一般工作流程如图 3 所示，使用机器学习进行相似性漏洞检测同样包括特征提取、向量嵌入和模型学习 3 个主要阶段，但是与基于代码模式的方法有以下区别：1）训练数据库中包含的不是单个代码样本，而是根据相似程度构建相似函数对和不相似函数对；2）训练阶段在进行特征学习时，学习的不是漏洞代码的模式，而是样本对代码之间的相似程度，基于相似性的漏洞检测模型多使用基于孪生网络的架构，并使用向量空间距离等作为评判代码相似性的度量标准，在对样本对中的 2 个样本代码进行向量表征时，相似样本的距离小，而不相似样本之间的距离大；3）在检测阶段，将测试程序的每一个函数与漏洞函数构建一个待测函数对，对其进行特征提取和向量嵌入，并将嵌入的向量映射到向量空间中，测试函数与漏洞函数的相似性分值与其在向量空间中的距离相关，基于相似性分值判断实现对目标程序的漏洞检测。

Genius[37]和 Gemini[38]是将机器学习与代码相似性比较结合进行漏洞检测的 2 个最具代表性的工具。它们分别利用传统的机器学习和深度学习，将函数的代码特征转化为向量进行相似性比较。2016 年 FENG 等[37]引入了一种解决方案 Genius，给定一个固件二进制函数，Genius 首先以属性化控制流图（attributed control flow graph，ACFG）的形式提取原始特征，之后基于代码本（codebook）将属性化控制流图转换为特征向量，代码本的生成过程较为昂贵，且受到训练数据集的规模限制。为了解决这一问题，XU 等[38]使用深度神经网络模型来进行属性化控制流图的嵌入，在准确性、性能和灵活性方面均优于 Genius。后续的研究多受 Genius[37]和 Gemini[38]的研究内容启发，使用不同的代码特征和机器学习模型来进行代码的向量化表示，最后使用 Siamese 孪生网络对得到的特征向量进行距离计算。

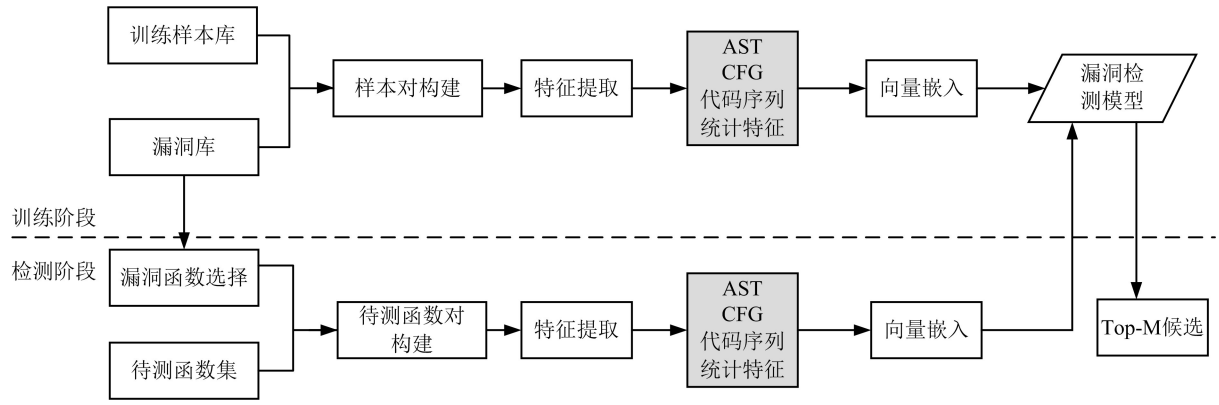

图 3 基于相似性检测的漏洞智能化挖掘一般过程

Fig. 3 The workflow of intelligent vulnerability mining based on code similarity comparison

在代码特征提取方面，GAO 等[39]在 Gemini[38]工作的基础上，将数据流图和控制流图融合为语义流图，将语义流图的结构特征和统计特征作为二进制程序函数的代码特征；TUFANO 等[40]探索将识别符、字节码、抽象语法树、控制流图和数据流图进行不同组合来进行代码的不同表征，探索其在代码表示和相似性计算方面的作用；文献[41-42]直接使用二进制原始字节作为代码特征，LIU 等[42]还添加了函数调用图相关的结构特征；文献[43-46]则将汇编代码中的指令作为代码特征；WANG 等[47]基于控制流图和函数调用图，得到统计信息、结构信息和函数调用信息，收集包含节点数、边数、基本块深度、控制流图宽度等 50 个特征；YANG 等[48]基于中间语言构建抽象语法树来表征代码结构特征。

在使用机器学习模型进行代码表示和相似性比较中，多数研究借鉴自然语言处理的方法表征代码上下文语义，使用图嵌入或者图神经网络模型实现对代码结构特征的表征。TUFANO 等[40]使用循环神经网络(recurrent neural network，RNN)学习代码序列特征，用高阶邻近保持嵌入(high-order proximity preserved embedding，HOPE)进行表示；MARASTONI 等[41]将二进制原始字节文件转为图的像素，之后使用图分类的方法(卷积神经网络 CNN)来进行相似性比较；ZUO 等[43]对指令使用 word2vec 进行向量表示，对基本块使用长短期记忆(LSTM)进行向量嵌入和表示；WANG 等[47]使用残差网络(residual network)对筛选出的特征进行函数向量的嵌入；MASSARELLI 等[44]则使用自注意力网络来进行函数级别的向量表示(包含双向的 RNN 网络)；DING 等[45]使用基于 word2vec 的 PV-DM 模型，将汇编代码的操作码和操作数分别进行向量化表示，并对操作数对应的向量进行平均求和后与操作码对应的向量进行连接，使用连接后的向量来表示每一条汇编指令，在得到汇编指令的嵌入向量后，对函数进行比较则基于指令的向量，使用随机游走(random walk)和边取样(edge sampling)的方法在扩展后的控制流图中进行汇编指令序列获取，汇编指令序列的向量可以用于进行函数的表征；YANG 等[48]对生成的基于中间语言的抽象语法树，使用树-LSTM(Tree-LSTM)网络来对树结构进行嵌入；TIAN 等[46]根据不同代码场景(同架构，同优化选项和跨架构、跨优化选项和跨编译器 3 种场景)来选择不同的 CNN＋LSTM 模型进行训练；SUN 等[49]在进行特征表示时，构建基于 BLSTM 和注意力机制的孪生网络，以提高漏洞检测方法的精确性；WU 等[50]基于控制依赖关系和数据依赖关系构建程序依赖图(program dependency graph)，使用社交网络的中心性分析，将程序依赖图中的节点向量与节点的 3 个中心性指标相乘后添加到节点代码向量中，将表征代码的图作为输入，使用卷积神经网络模型进行分类，完成相似性检测。同时，借鉴神经网络模型完成代码结构特征的向量表征，也是近年来研究的重点之一。XBA 将代码的表示学习转为图对齐问题，完成对不同架构二进制代码的对齐，并使用图卷积网络对代码语义进行学习[53]；QBinDiff 将代码的二进制相似性比较看作调用图上的图编辑问题，将其等价为网络对齐问题，提出一种基于最大乘积置信传播(max-product belief propagation)的求解策略解决该问

题[54]。ASTERIA 将代码的抽象语法树作为结构特征，使用 Tree-LSTM 学习代码丰富的结构化语义特征[55]。GMN 则使用跨图的注意力机制匹配方法来对代码进行结构化表征和相似性比较[56]。

笔者所在的研究团队在基于代码相似性的方法进行漏洞智能化静态分析方面也做了相应研究[57-59]，利用自然语言处理模型和图神经网络模型对代码的语义特征进行表征，其工作流程如图 4 所示。其中，使用自然语言处理的 skip-thoughts 模型，对代码在指令粒度的语义特征进行了提取和表征，实现了更细粒度的代码语义特征的表示；此外，针对当前结构化特征表征不足的情况，提出了一种基于图编码-图解码的结构化表征方法，利用图卷积网络对代码的全局结构特征进行向量化表征。最后对经过指令编码器得到的代码语义特征和经过图编码器得到的结构化语义特征向量，使用超参数 W_1 和 W_2 进行合并后，使用双曲正切函数 tanh 得到包含图结构特征和代码指令特征的向量并进行池化，得到的向量进行相似性比较，判断 2 个函数之间的相似程度。

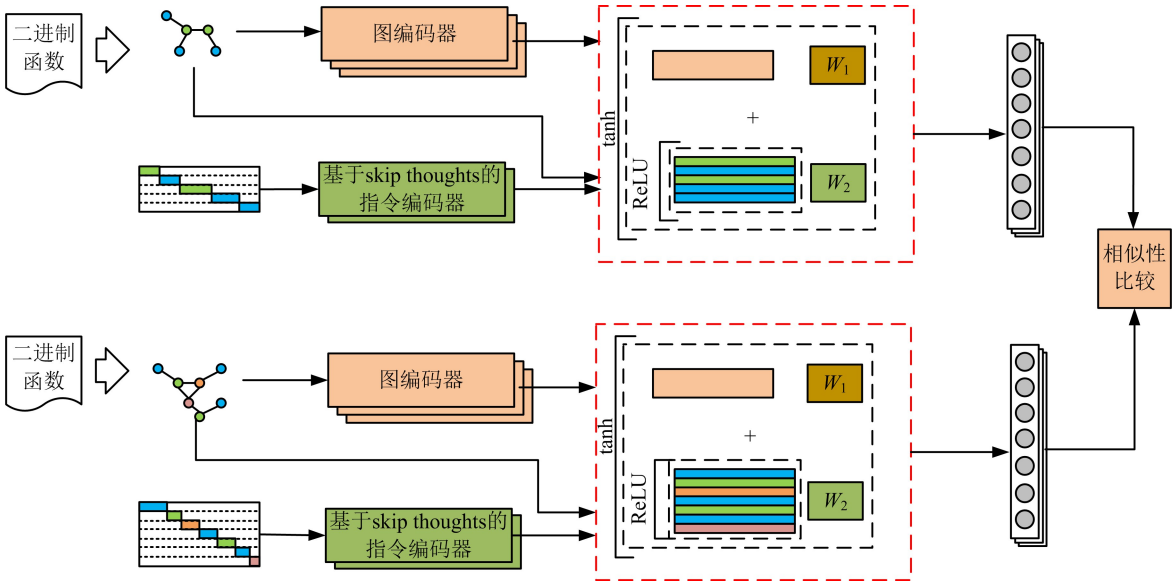

图 4　基于图神经网络和自然语言处理模型的代码相似性比较过程

Fig. 4　Code similarity comparison based on graph neural network and natural language process

将机器学习模型与代码相似性比较技术结合来进行漏洞挖掘主要通过使用大量的训练样本对相似代码的语法、语义信息进行学习，得到能够表征代码特征并进行代码相似度匹配的机器学习模型，基于目标程序代码与漏洞代码的相似程度完成检测。与基于模式的方法不同的是，基于代码相似性的方法在给定特定漏洞的情况下，对目标程序中是否存在该漏洞进行检测和判断，而并非对漏洞模式进行学习。基于相似性比较的方法更加适应于源码未知的二进制程序的漏洞检测中，特定的场景包含跨平台、跨编译优化选项和跨版本的比较等，基于机器学习的代码相似性比较方法不仅可以有效应用于漏洞检测中，还可以应用于软件供应链安全分析中的恶意代码识别以及补丁存在性判断中。

2　软件漏洞智能化动态挖掘方法

将机器学习方法应用于软件漏洞动态挖掘时，研究人员则致力于以动态程序分析方法为主，应用机器学习算法来解决动态程序分析方法中存在的技术瓶颈。软件漏洞的智能化动态挖掘方法涉及使用机器学习模型对模糊测试、符号执行和污点分析等动态分析方法的智能化辅助技术研究。

2.1　智能化模糊测试方法

模糊测试是目前最流行的漏洞动态挖掘技术之一。从概念上讲，模糊测试从向目标应用程序生成大量正常和异常输入开始，并尝试通过将生成的输入提供给目标应用程序并监视执行状态来检测异常，其一般工作流程如图 5 所示。与其他技术相比，模糊测试易于部署，具有良好的

可扩展性和适用性，并且可以在使用或不使用源代码的情况下执行；同时，模糊测试只需要很少的目标应用程序知识，并且可以轻松扩展到大规模应用程序。

传统的模糊测试技术面临许多挑战，比如如何变异输入种子文件，如何绕过格式验证等。机器学习算法能够应用在模糊测试的 3 个场景中：

1）在种子文件生成中，机器学习技术主要用于学习输入种子文件的语法和语义信息，输入样本与执行路径之间的映射关系，或现有种子调度策略的经验。

2）在测试用例生成中，有 2 种类型的测试用例生成：基于突变和基于生成。在基于突变的方法中，机器学习技术的运用主要是为了解决种子文件应该在哪里发生突变的问题；在基于生成的方法中，机器学习技术主要学习输入样本的语法和语义信息，以生成符合输入语法规则的样本。

3）在测试用例筛选中，机器学习可以应用于 2 个方面。一是用于构建分类器，以便它可以区分哪些测试用例更容易执行更多代码并更容易触发崩溃；二是使用强化学习指导模糊化过程中突变算子的选择。

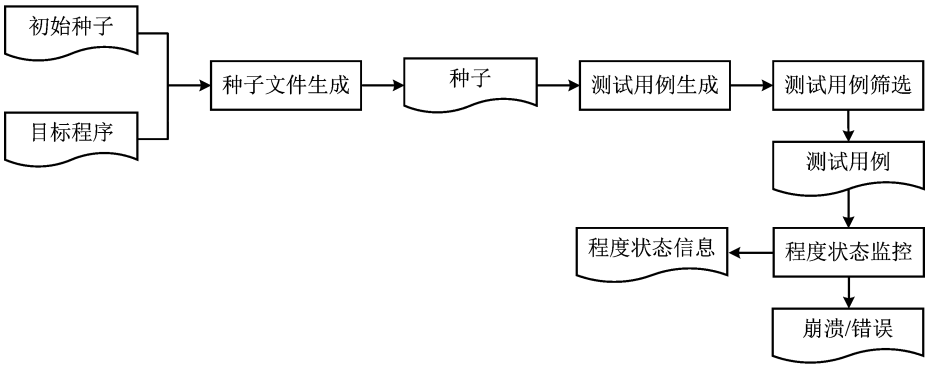

图 5 模糊测试方法的一般工作流程

Fig. 5 The workflow of fuzzing testing

2.1.1 种子文件生成

种子文件经过突变操作后形成测试用例，高质量的种子文件更有可能生成能够到达更深、更难到达的路径的测试用例。因此，种子文件是测试用例是否良好的先决条件，是影响测试有效性的重要因素。

通常，种子文件生成的策略主要有 3 种：使用基准种子文件集，使用现有的 POC 和使用随机生成的种子文件。然而，这些种子选择策略都难以满足目标程序对输入的结构化要求，或是能够满足结构化要求的种子数量少，导致模糊测试难以探测程序深层，影响测试有效性。

因此，研究人员开始关注如何利用现有知识结合机器学习方法，生成足量且能够满足输入结构化要求的种子文件。WANG 等[60]提出了一种名为 Skyfire 的新型数据驱动的种子生成方法，利用大量现有样本中的知识为处理高度结构化输入的模糊测试程序生成分布良好的种子输入，利用概率上下文相关语法（PCSG）指定语法特征和语义规则，进而生成种子输入。使用该方法在实验数据集上路径覆盖率提高了 20％和函数覆盖率提高了 15％。

CHEN 等[61]提出了一种机器学习增强的混合型模糊测试系统 MEUZZ。该系统采用监督学习进行自适应和可扩展的种子调度，根据从过去在相同或类似程序上做出的种子调度决策中学到的知识，提取一系列静态和动态的程序特征，通过代码可访问性和动态分析进行学习，最终确定哪些新种子有望产生更好的模糊测试效果。

2.1.2 测试用例生成

测试用例可以通过对种子文件执行突变来生成（基于突变的测试用例），也可以基于已知的输入文件格式构建（基于生成的测试用例）。测试用例作为模糊测试器的输入，不同的测试用例会影响程序控制流，直接影响模糊测试触发 crash 的概率，构建具有高代码覆盖率或漏洞导向的测试用例，可以有效提高模糊测试的漏洞挖掘效率。

基于突变的测试用例生成方法通过随机改变格式良好的种子文件或使用预定义的突变策略来生成测试输入，这些策略可以根据运行时收集的面向目标的程序信息进行调整。如何对种子文件进行突变，使生成的测试用例能够覆盖更

多的代码是基于突变的测试用例生成方法研究中的关键问题。

SHE 等[62]针对基于遗传算法的模糊测试容易陷入无效随机突变序列中的问题，提出了一种使用替代神经网络模型的新的程序平滑技术，能够解决普通梯度引导优化技术中难以解决的程序不连续性问题，可以有效识别目标关键突变位置，对种子文件进行突变以生成能够触发更多 crash 的测试用例。

LI 等[63]构建了一个基于神经网络的漏洞预测模型，并形成了面向漏洞的模糊测试器 V-Fuzz。首先，V-Fuzz 使用数据处理模块对二进制文件进行反汇编，得到每个函数的 ACFG，这是漏洞预测模型的输入。然后，预测模型将为每个二进制程序函数提供漏洞预测。根据漏洞预测结果，程序中的每个基本块都会被赋予一个静态易受攻击分数（SVS），稍后将用于评估执行的测试用例。根据 SVS 和执行信息，V-Fuzz 将为每个执行的测试用例计算一个适应度分数。具有高适应度分数的测试用例被视为高质量输入，并被发送到测试种子库中。

当前给定格式的测试用例生成研究所面临的挑战在于，如何智能化、全面地确定输入格式规范。输入格式规范包括字段类型、数目、长度及字段间的层级嵌套关系、数据约束等信息，早期的输入格式规范定义依赖专家经验，耗时长且存在误操作，属于劳动力密集型工作。当前已有研究将机器学习与输入格式规范确定相结合，智能化地从大量符合格式规范的输入中识别格式特征，确定格式规范。

GODEFROID 等[64]研究了使用机器学习技术和示例输入为基于语法的模糊测试自动生成输入语法的问题，并首次提出使用基于神经网络的统计学习方法 Learn&Fuzz。具体来说，使用递归神经网络来学习一个统计输入模型（seq2seq 模型），根据学习模型的概率分布生成新的输入，但该方法只能针对文本信息进行学习。

为了解决 GODEFROID 等[64]工作中只能生成文本信息的问题，LIU 等[65]将存储在文件中的纯数据与描述文件格式的元数据区分开来，没有非文本数据的测试用例则使用 LSTM 进行训练。最后，将生成的文本数据与突变的非文本数据组合在一起生成新的测试用例。

JITSUNARI 等[66]从生成指令序列的角度改进了 Learn&Fuzz 方法的低覆盖率，其基本思想是根据基于混合字符/标记级别的指令序列训练递归神经网络模型。测试用例的训练模型生成了新的 PDF 页面流指令，以诱导指令解析代码的正面覆盖。

2.1.3　测试用例筛选

测试用例筛选是在生成的大量测试用例中进行筛选，以期排除掉不太可能触发新路径或漏洞的测试输入，提高模糊测试的执行效率。

GONG 等[67]使用代码覆盖率作为测试用例过滤器，基于 AFL 生成的样本训练了一个深度学习模型，这些样本带有能改变程序状态的标签，训练的模型可以预测新一轮 AFL 生成的样本是否可以改变程序状态，能够改变程序状态的测试用例会被标记为高质量测试用例。

ZONG 等[68]以可能存在漏洞的区域作为导向，提出了一种深度学习方法 FuzzGuard，在执行目标程序之前预测测试用例对目标区域的可访问性，帮助导向式灰盒模糊测试器提前过滤掉无法到达目标区域的测试用例，以提高模糊测试的执行效率。

SUN 等[69]将马尔可夫链和 PCFG 模型相结合，从互联网上获取大量 JS 代码作为语料库进行学习。利用学习到的信息通过计算代码与通用代码的偏差来判定代码的不常见性，结果表明，代码偏差较大的脚本，即更不常见的代码更有可能触发 crash。

KARAMCHETI 等[70]提出了一种基于汤普森抽样的优化方法，该方法可以在模糊测试单个程序的过程中自适应地调整突变体分布。通过了解每个突变运算符对代码覆盖率的影响来确定应选择哪个突变运算符。

2.2　智能化符号执行方法

符号执行是一种广泛使用的程序分析技术，它收集并求解路径条件以指导程序遍历。但是由于当前约束求解器的限制，很难对具有复杂路径条件的程序（如非线性约束和函数调用）应用符号执行。

LI 等[71]提出了一种结合机器学习的符号执行工具 MLB，首先将路径条件的可行性问题转化为优化问题，不依赖于经典的约束求解。然后通过机器学习中的优化技术 RACOS 引导采样和验

证来处理优化问题。MLB 基于符号路径查找器实现,线性路径条件编号和非线性算数运算以及库方法的黑盒函数调用都能够被编码到符号路径条件中。

CHEN 等[72]提出了一种将被测程序转换为语义不变但更易于分析的程序来提高符号执行的效率的方法 LEO。该方法重点在于学习如何调整编译器优化选项以加速符号执行。首先,将程序拆分为多个文件,使得每个文件仅包含一个源代码部分。然后,基于机器学习算法,对拆分开的每个文件进行针对性地编译器优化选项学习。最后,使用学习到的编译器优化选项对拆分开的文件进行优化并最终合并到一个经过了精细优化的程序中。实验结果表明,LEO 对符号执行的速度有显著的提升。

CHEN 等[73]提出了一种新的基于突变的模糊测试器 Angora,通过污点分析跟踪输入中影响条件分支的字节,然后使用梯度下降的方式对变异后生成的路径约束进行求解。能够有效避免符号执行调用 SMT 求解器可能带来的开销以及复杂约束不可解的问题。但是梯度下降方法面对目标函数不可导或存在不可导点时,仍然会出现求解困难的问题。

SHI 等[74]提出了一种补充动态符号执行的新方法 NEUEX,被称为神经符号执行,其符号执行引擎除了标准符号约束之外,还为被分析的程序的某些部分积累神经约束,神经约束捕获传统符号执行引擎无法快速恢复求解约束的程序值之间的关系。在神经符号执行过程中,通过分支此类程序逻辑的实施训练过程切换到归纳学习模式,将目标逻辑视为黑匣子,并学习尽可能近似它的神经网络表示。

WEN 等[75]聚焦于研究不同 SMT 求解器对不同的路径约束的求解效果。首先,定义了符号执行中求解器选择问题和评估指标,用于衡量通过求解器选择提高性能的可能性。然后,提出了路径约束分类器(PCC)的设计和实现,PCC 是一种基于机器学习的元求解器,用于预测哪个求解器在接收路径约束时能够表现良好,通过动态选择每个查询的求解器来减少整体约束的求解耗时,并通过使用约束特征缓存表利用符号执行的结构特征来大幅缩短特征提取时间。

BU 等[76]提出了一个基于机器学习的符号执行框架 MLBSE。首先,将所有非线性数值运算、数学方法、浮点表达式和库方法调用编码为路径条件中的符号约束,即将路径条件的可行性问题转换为搜索问题。然后,采用机器学习中的无导数优化方法来解决搜索问题。相对于其他方法,无导数优化方法具备理论基础和经验性能上的优势,其优化性能是有界的,当它无法找到可行的解决方案时,可以估计问题可满足性的置信度,让用户了解无法求解的原因,从而决定继续分析还是停止执行。

HE 等[77]提出了一种基于学习的符号执行求解策略 Learch,通过选择有希望的状态进行符号执行,以解决路径爆炸问题。Learch 直接估计每个状态对在时间预算内最大化覆盖范围目标的贡献,而不是基于简单的统计数据的启发式方法进行计算。另外,Learch 在训练数据生成和特征提取中利用了现有的启发式方法,因此具备扩展融合任何新的启发式方法的优势,这是通过迭代学习过程构建的。在每次迭代中,首先在一组训练程序上运行符号执行,利用不同的状态选择策略生成一组不同的测试,然后,对于生成的测试中每个探索的状态,提取一组高级特征,并根据整体覆盖率的提高和探索状态所花费的时间计算奖励。同时,生成一个受监督的数据集,该数据集捕获上一步中使用的策略及对应行为。最后,通过训练回归模型来构建一个学习策略,以便在监督数据集上实现小的损失,使得模型能够对奖励做出准确的估计。在当前迭代中学到的策略用于在下一次迭代中添加新的监督数据,以创建其他学习的策略。

RUARO 等[78]提出了一种基于模式识别和机器学习的路径优先排序方法。通过提取表征执行路径、函数、基本块和连接组件的特征,利用受监督的机器学习技术,发现并学习易受攻击路径的模式,利用它们的预测在新程序中发现有趣的执行路径。这种方法的原理是,具有类似功能失效的执行状态的程序,其特征一般是一组特定的功能,如 API 调用序列,内存取消引用,非常复杂的函数或者循环行为。

3 软件漏洞智能化挖掘研究总结与展望

3.1 软件漏洞智能化挖掘研究总结

通过对软件漏洞智能化挖掘的研究及梳理,

可以看出，在漏洞静态智能化挖掘技术研究中，研究人员发挥了机器学习对数据学习的优势，将静态分析得到的代码特征使用机器学习模型进行表征和特征学习，能够有效提高基于静态方法对漏洞分析和挖掘的自动化和智能化程度；在漏洞动态智能化挖掘技术研究中，则将机器学习方法作为动态分析的辅助，以提高漏洞动态挖掘的效率。

3.1.1　软件漏洞静态挖掘方法具备较高的智能化水平

将静态程序分析与机器学习技术结合，实现漏洞智能化静态挖掘的研究主要是利用机器学习方法能够对数据规律和特征进行学习的优势，将相应的机器学习模型应用于代码特征的表征和漏洞模式的学习中。软件漏洞智能化静态挖掘方法实现了程序静态分析与机器学习方法的深度融合。在软件漏洞智能化静态挖掘研究中，研究人员可从以下 3 个方面开展研究：

1）目标程序方面，基于机器学习的静态挖掘方法不仅可以针对目标源码程序，还可以针对源码未知的二进制程序。其中，基于代码度量的漏洞挖掘方法和基于模式的方法需要得到代码的更多语义信息，这两类研究中以源码程序作为目标进行漏洞挖掘更加充分和成熟，而基于代码相似性的漏洞挖掘方法的研究更多聚焦于二进制程序，基于提取的代码语义特征，对代码之间的相似程度进行学习，实现对漏洞的检测。

2）提取的代码特征方面，虽然代码本身能够表征更多的语义信息，但是代码的其他信息，如抽象语法树、控制流图、数据流图、函数调用图等结构以及源码的 Token 等，也可以表征代码的特征。同时在特定的场景下，针对不同的代码特征，使用对应的模型来对代码特征进行表征，从而挖掘漏洞的研究，都取得了较好的效果。

3）选择的机器学习模型方面，早期的研究多使用传统机器学习模型，如逻辑回归、随机森林等，近年来则引入了循环神经网络、卷积神经网络、图神经网络等模型，并添加如注意力机制等到模型中，以提高特征提取和表示的有效性。

本文将基于代码度量、基于代码模式和基于代码相似性 3 类漏洞智能化静态挖掘方法的代表性工作，按照目标程序是否开源、提取的代码特征和选择的机器学习模型进行总结，如表 1 所列。

表 1　漏洞智能化静态挖掘研究总结

Tab. 1　Summary of the researches on intelligent static vulnerability mining technology

研究工作	研究方向	针对源码/二进制	代码特征	使用学习模型/分类器
文献[14]	基于代码度量	源码	复杂度、代码流失（code churn）和开发者行为	逻辑回归
文献[15]	基于代码度量	源码	代码流失、代码复杂性、代码覆盖、依赖和组织度量	逻辑回归
文献[51]	基于代码度量	源码	代码流失、开发者行为的代码度量和 GitHub 数据库中提取的元数据	SVM 分类器
文献[16]	基于代码度量	源码	代码行数 LOC，非 HTML 代码行数，函数数目等 12 种度量	SVM，XGBoost 分类器
文献[17]	基于代码度量	源码	LOC 度量（代码总行等），McCabe 度量和 Halstead 度量	MLP
文献[18]	基于代码度量	源码	代码平均行数、圈复杂度、继承树中类的最大深度	包括 random forest，naive Bayes 的 32 个监督学习方法
文献[21-22]	基于代码模式	源码 C/C++	AST/CFG/PDG	结构比较
文献[23]	基于代码模式	源码	Token＋统计特征	SVM
文献[31]	基于代码模式	二进制	标准 C 库的有限序列调用的集合＋动态运行调用序列	逻辑回归，MLP 和随机森林

续表

研究工作	研究方向	针对源码/二进制	代码特征	使用学习模型/分类器
文献[25]	基于代码模式	源码	符号化 Token	BLSTM
文献[26]	基于代码模式	源码	Token+控制流	改进的 BLSTM
文献[27]	基于代码模式	源码	AST、CFG	决策树、随机森林、逻辑回归、朴素贝叶斯和树增广朴素贝叶斯 5 种分类器
文献[32]	基于代码模式	二进制	汇编代码	VAE(变分自编码器)
文献[28]	基于代码模式	源码	AST,CFG,DFG+NCS+Token	GGNN+Conv layer
文献[29]	基于代码模式	源码	数据流图、控制流图和函数调用图+Token	GGNN+SVM,随机森林,k-近邻,逻辑回归 LR,gradient boosting GB
文献[30]	基于代码模式	源码	基于中间语言的程序切片	双向 RNN(BRNN)模型
文献[52]	基于代码相似性	二进制	数字化统计信息	k-最近邻
文献[37]	基于代码相似性	二进制	属性化控制流图	代码本(codebook)
文献[38]	基于代码相似性	二进制	属性化控制流图	structure2vec
文献[39]	基于代码相似性	二进制	CFG+DFG	structure2vec
文献[40]	基于代码相似性	源码	识别符、字节码、AST、CFG 和 DFG	HOPE+RNN
文献[41]	基于代码相似性	二进制	原始字节	CNN
文献[42]	基于代码相似性	二进制	原始字节	CNN
文献[43]	基于代码相似性	二进制	指令	LSTM
文献[47]	基于代码相似性	二进制	CFG,函数调用图+筛选后的统计信息	残差网络 residual network+梯度下降
文献[44]	基于代码相似性	二进制	汇编指令	带有注意力机制的双向的 RNN 网络
文献[48]	基于代码相似性	二进制	基于中间语言的 AST	Tree-LSTM
文献[46]	基于代码相似性	二进制	汇编指令	CNN+LSTM

3.1.2 软件漏洞动态挖掘方法亟待突破技术瓶颈

在基于机器学习的模糊测试研究中,其创新之处在于利用机器学习的文本处理能力提取更加准确的程序特征,以及利用机器学习的分类能力筛选更加优异的测试用例和变异操作;基于机器学习的符号执行方面研究较少,但也取得了一定成果。需要在软件漏洞智能化动态挖掘方法的进一步探索中,深度融合机器学习模型,对动态挖掘方法的技术瓶颈进行突破。软件漏洞智能化动态挖掘方法主要包括以下研究重点。

1) 模糊测试的目标阶段方面,大多数研究集中在测试用例生成环节。一方面是因为作为模糊测试器的输入,测试用例优劣将直接影响触发 crash 的概率,构建具有高代码覆盖率或漏洞导向的测试用例可以有效提高模糊测试器中的漏洞挖掘与检测效率。利用机器学习算法在测试用例生成环节进行研究,能够较为有效地提高模糊测试器的工作效率,容易获得优异的效果。另一方面是因为机器学习算法在文本处理能力、结构化信息提取能力以及提取程序语义方面具备优势,利用机器学习算法对测试用例的生成进行改进,使得生成的测试用例在结构信息、程序语义上更符合实际要求。

2) 模糊测试选择的模型方面,在模糊测试的不同阶段由于目的不同,导致使用的算法模型不同,但就当前研究趋势来看,研究人员暂未在算

法和模型上面进行细致的研究,应用到模糊测试上的机器学习算法十分有限。

3) 符号执行收集并求解路径条件以指导程序遍历方面。由于当前约束求解器的限制,对于复杂条件的求解以及路径爆炸等问题依然存在难点,是当前研究的重点。结合机器学习进行符号执行主要关注约束求解的效率提升、路径爆炸问题解决以及路径优先排序问题。

表 2 总结了当前漏洞智能化动态挖掘方法的代表性工作。

表 2　漏洞智能化动态挖掘研究总结

Tab. 2　Summary of the researches on intelligent dynamic vulnerability mining technology

研究工作	针对问题	特征学习模型/算法	优势
文献[60]	种子文件生成	PCSG	利用概率上下文相关语法(PCSG)指定语法特征和语义规则,进而生成种子输入
文献[61]	种子文件生成	supervised machine learning	采用监督机器学习进行自适应和可推广的种子调度,根据从过去在相同或类似程序上做出的种子调度决策中学到的知识
文献[62]	测试用例生成	CNN	使用平滑近似梯度(即 NN 模型)来解决程序行为包含许多不连续性,平台和脊的问题。计算结果是实现更高覆盖范围的最佳解决方案
文献[63]	测试用例生成	struc2vec	面向漏洞的模糊测试器,能够有效提高模糊测试效率
文献[64]	测试用例生成	LSTM,seq2seq	首次提出使用基于神经网络的统计学习方法 Learn&Fuzz。具体来说,使用递归神经网络来学习一个统计输入模型(seq2seq 模型),可以用来根据学习模型的概率分布生成新的输入
文献[65]	测试用例生成	seq2seq	减轻文献[58]工作中只能生成文本信息的问题
文献[66]	测试用例生成	LSTM	从生成指令序列的角度改进 Learn&Fuzz[58] 的低覆盖率
文献[67]	测试用例筛选	BLSTM,MLP	基于 AFL 生成的样本训练了一个深度学习模型,这些样本带有将改变程序状态的标签。训练的模型可以预测新一轮 AFL 生成的样本是否可以改变程序状态
文献[68]	测试用例筛选	CNN	在执行目标程序之前预测输入的可访问性(即错过目标与否),帮助定向灰盒模糊化过滤掉无法访问的输入
文献[69]	测试用例筛选	PCFG	将马尔可夫链和 PCFG 模型结合起来,从程序员开发的普通脚本语料库中学习共性,并利用学习到的信息通过测量脚本与通用脚本的偏差来计算脚本的不常见性
文献[70]	测试用例筛选	Thompson sampling	可以在模糊测试单个程序的过程中自适应地调整突变体分布
文献[71]	约束求解效率	RACOS	将路径条件的可行性问题转化为优化问题,通过机器学习中的优化技术 RACOS 引导采样和验证来处理优化问题
文献[72]	符号执行效率	SMO、SVM	将程序拆分为多个文件,针对每个文件使用 SVM 进行针对性的编译器优化选项训练。通过对程序的精细优化达到提高符号执行速度的效果
文献[73]	复杂约束不可解	gradient descent	使用梯度下降的方式对变异后生成的路径约束进行求解
文献[74]	约束求解效率	MLP	使用神经约束捕获传统符号执行引擎无法快速恢复约束的程序值之间的关系
文献[75]	约束求解效率	DNN	提出路径约束分类器,基于机器学习预测在接收路径约束时表现良好的 SMT 求解器,通过动态选择优异的求解器降低整体约束的求解耗时

续表

研究工作	针对问题	特征学习模型/算法	优势
文献[76]	约束求解效率	RACOS	将所有非线性数值运算、数学方法、浮点表达式和库方法调用编码为路径条件中的符号约束，将其转换为搜索问题，然后采用 RACOS 方法解决搜索问题
文献[77]	路径爆炸问题	regression	通过机器学习中的回归算法，直接估计每个状态对在时间预算内最大化覆盖范围目标的贡献，并通过迭代学习的过程引入启发式方法的策略
文献[78]	路径优先排序	supervised machine learning	通过提取表征执行路径、函数、基本块和其他连接组件的特征，利用受监督的机器学习技术，发现并学习易受攻击路径的模式，以此预测在新程序中的更容易触发漏洞的路径

3.2 软件漏洞智能化挖掘研究展望

虽然将机器学习应用于漏洞挖掘领域，进行漏洞智能化挖掘研究得到了广泛关注，并取得了显著成果，但是漏洞成因复杂，对其进行智能化挖掘依然是一个开放性问题，面临着诸多挑战，漏洞智能化挖掘技术的研究将是漏洞挖掘领域研究人员持续关注的研究点。结合近几年的研究热点，漏洞智能化挖掘未来的重点研究方向包括以下几个方面。

3.2.1 机器学习模型自适应选择

虽然机器学习模型对于漏洞代码的特点进行学习，并取得了一定的成果，但是不同漏洞类型对应的代码特征不同，当前部分研究也证明在不同场景下，不同的模型表现也有所差异。因此，使用特定的模型来对一般漏洞代码进行学习训练得到的结果并不尽如人意。根据不同场景、漏洞的类型或者代码行为自适应选择机器学习模型，从而有针对性地检测漏洞，是提高漏洞挖掘智能化水平的一个更为精细且较为重要的研究内容。同时根据不同的场景，使用机器学习模型对代码不同特征的感知和自适应学习，进一步应用图神经网络等最新模型，深化对代码特征的学习与表征，实现对程序语义特征的精确刻画，是研究人员需重点关注的研究方向之一。

3.2.2 人工智能与动态挖掘深度融合

当前漏洞智能化动态挖掘技术的研究多是将机器学习方法作为动态漏洞挖掘的辅助，如针对模糊测试和符号执行中存在的路径状态空间探索有限和约束求解计算量大的问题，将机器学习方法与程序分析结合，可以提高模糊测试的路径探索效率，以及辅助约束求解的复杂计算。将机器学习应用于动态程序分析技术，让其对于漏洞动态挖掘中起的辅助作用逐渐上升为主导作用，使机器学习方法更加深入地融合于漏洞动态挖掘过程。如在模糊测试过程中，以模糊测试不同阶段的特征、相应数据的大小以及不同算法的优缺点作为机器学习算法选择的基础；研究更复杂和合适的神经网络模型并深度应用于模糊测试中，使用图卷积网络、融合神经网络和可解释的深度学习模型以提高生成的样本的质量；在符号执行与机器学习结合方面，将进一步探索应用机器学习算法如何解决符号执行中的难题，包括但不仅限于路径排序选择[72]、约束求解器选择[69]、编译优化选项选择[66]等。

3.2.3 智能动静态挖掘模型探索

现有的漏洞智能化挖掘方法分别在静态智能化挖掘和动态智能化挖掘方法上进行研究，将智能化技术应用于动静结合的研究相对较少。因此，需要研究基于机器学习的动静化结合方法，对静态智能化挖掘的结果进行动态验证；或者基于静态挖掘方法的结果，提升动态挖掘方法的智能化水平。进一步，未来的研究方向还包括将机器学习模型同时作用于 2 个阶段，考虑构建基于机器学习的动静态分析结合模型。

3.2.4 漏洞分析能力跨平台迁移

随着物联网、云计算、大数据等技术的发展，对漏洞的研究已经由传统计算机向新技术设施平台扩展，特别是当前的物联网设备的漏洞挖掘已经成为研究热点。目前，对漏洞挖掘的研究目标并不局限于传统架构（如 X86）的程序，还包括非传统架构（如 MIPS、ARM）中的程序。而随着新技术的发展，对非传统架构的二进制程序漏洞智能化挖掘方法将获得研究人员进一步的关注。当前机器学习与非传统架构漏洞挖掘技术结合

的主要研究方向是基于代码相似性的漏洞智能化挖掘研究，提取代码特征来进行相似性比较，完成跨平台、跨优化选项的二进制程序漏洞代码检测，实现对物联网固件等非传统架构的二进制程序在具体应用场景的漏洞检测。

3.2.5　规范化漏洞数据集建设

对于基于机器学习的静态漏洞挖掘方法而言，无论是使用传统机器学习模型还是深度学习模型，都是基于代码训练数据集来构建训练数据，而当前的研究使用的数据集或者是研究人员自行收集漏洞信息构建，或者是基于手工插入的缺陷(LAVA-M)构建，或者是使用比赛用的漏洞程序(DARPA CGC)。需要构建统一的漏洞数据集，其中包含各种类型的漏洞对应代码，从而方便研究人员进行相应的训练数据集的构建，进一步实现对机器学习模型的比较。

3.2.6　大规模预训练语言模型在漏洞挖掘中的深度应用

当前，大规模预训练语言模型如 ChatGPT、GPT-4 已经呈现了强大的应用能力，引发研究人员的极大关注。ChatGPT 基于 Transformer 神经网络架构，利用大量的语料库来完成模型训练，具有强大的语言理解和文本生成的能力。目前将 ChatGPT 等模型应用于漏洞挖掘领域的一个具有研究前景的方向是将其应用于代码复用漏洞的检测中，即通过学习代码的语义判断代码之间的相似程度，从而基于已有的漏洞代码，对目标程序中是否存在漏洞进行智能化判断。

3.2.7　针对智能学习模型本身的漏洞挖掘

基于机器学习，特别是基于深度学习的智能学习模型在包括漏洞挖掘领域等多个领域进行了广泛的应用，同时智能学习模型本身同样存在脆弱性，研究智能学习模型的脆弱性，实现对模型的漏洞挖掘也是后续研究的重要方向[79]。当前的研究主要是对于模型的算法存在的脆弱性进行的研究，如使用生成对抗样本验证模型的鲁棒性[80-83]，对于模型实现的代码本身的漏洞挖掘研究相对较少，如使用模糊测试等方法进行模型代码的漏洞挖掘研究，但针对模型代码的漏洞挖掘也是后续研究的重点。

4　结束语

将机器学习方法与传统漏洞挖掘方法结合，已经被证明能够有效提高漏洞挖掘的自动化和智能化水平，是当前漏洞挖掘的重要研究方向。本文对近年来将机器学习，特别是深度学习与漏洞挖掘结合的研究进行了分析和梳理，从漏洞代码智能化静态挖掘方法和智能化动态挖掘方法 2 个方面对相关工作进行了分类阐述。基于对已有工作的比较和总结，对当前的漏洞智能化挖掘技术的发展方向进行了展望。

可以看出，结合机器学习方法提高漏洞挖掘智能化水平的研究取得了显著成果。在未来的研究工作中，对漏洞分类特征的精细化学习，对代码语义特征的精确刻画以及自适应模型选择等方向将引发更多的关注。随着机器学习、特别是深度学习的广泛研究，相关日益成熟的技术可以进一步克服当前研究的局限性，基于机器学习的软件漏洞智能化挖掘方法是网络安全领域持续研究的热点。

参 考 文 献

[1] SHIREY R. Internet security glossary, version 2[R]. 2007.

[2] 吴世忠. 软件漏洞分析技术[M]. 北京：科学出版社，2014.
WU Shizhong. Sofware vulnerability analysis technology[M]. Beijing: Science Press, 2014. (in Chinese)

[3] ENGLER D, CHEN D Y, HALLEM S, et al. Bugs as deviant behavior: a general approach to inferring errors in systems code[J]. ACM SIGOPS Operating Systems Review, 2001, 35(5): 57-72.

[4] JANG J, AGRAWAL A, BRUMLEY D. ReDeBug: finding unpatched code clones in entire OS distributions[C]//Proceedings of 2012 IEEE Symposium on Security and Privacy. [S. l.]: IEEE, 2012: 48-62.

[5] KIM S, WOO S, LEE H, et al. Vuddy: a scalable approach for vulnerable code clone discovery[C]//Proceedings of 2017 IEEE Symposium on Security and Privacy. [S. l.]: IEEE, 2017: 595-614.

[6] CADAR C, DUNBAR D, ENGLER D R. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs[C]//Proceedings of Usenix Conference on Operating Systems Design & Implementation. [S. l.: s. n.], 2008: 209-224.

[7] RAMOS D A, ENGLER D. Under-constrained symbolic execution: correctness checking for real code [C]//Proceedings of the 24th USENIX Security Symposium. [S. l.: s. n.], 2015: 49-64.

[8] AVGERINOS T, CHA S K, REBERT A, et al. Au-

tomatic exploit generation[J]. Communications of the ACM, 2014, 57(2): 74-84.

[9] PEWNY J, SCHUSTER F, BERNHARD L, et al. Leveraging semantic signatures for bug search in binary programs[C]//Proceedings of the 30th Annual Computer Security Applications Conference. [S. l.: s. n.], 2014: 406-415.

[10] SUTTON M, GREENE A, AMINI P. Fuzzing: brute force vulnerability discovery[M]. USA: Pearson Education, 2007.

[11] NEWSOME J, SONG D X. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software[C]//Proceedings of the 12th Annual Network and Distributed System Security Symposium. [S. l.: s. n.], 2005: 3-4.

[12] PORTOKALIDIS G, SLOWINSKA A, BOS H. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation[J]. ACM SIGOPS Operating Systems Review, 2006, 40(4): 15-27.

[13] GODEFROID P, KLARLUND N, SEN K. DART: directed automated random testing[C]//Proceedings of the 5th International Haifa Verification Conference on Hardware and Software: Verification and Testing. [S. l.: s. n.], 2005: 213-223.

[14] SHIN Y, MENEELY A, WILLIAMS L, et al. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities[J]. IEEE Transactions on Software Engineering, 2010, 37(6): 772-787.

[15] ZIMMERMANN T, NAGAPPAN N, WILLIAMS L. Searching for a needle in a haystack: predicting security vulnerabilities for windows vista[C]//Proceedings of the 3rd International Conference on Software Testing, Verification and Validation. [S. l.]: IEEE, 2010: 421-428.

[16] KALOUPTSOGLOU I, SIAVVAS M, TSOUKALAS D, et al. Cross-project vulnerability prediction based on software metrics and deep learning[C]//Proceedings of International Conference on Computational Science and its Applications. [S. l.: s. n.], 2020: 877-893.

[17] ZAGANE M, ABDI M K, ALENEZI M. Deep learning for software vulnerabilities detection using code metrics[J]. IEEE Access, 2020, 8: 74562-74570.

[18] GUPTA A, SURI B, KUMAR V, et al. Extracting rules for vulnerabilities detection with static metrics using machine learning[J]. International Journal of System Assurance Engineering and Management, 2021, 12(1): 65-76.

[19] LIANG H, WANG L, WU D, et al. MLSA: a static bugs analysis tool based on LLVM IR[C]//Proceedings of the 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing(SNPD). [S. l.: s. n.], IEEE, 2016: 407-412.

[20] FANG Z, LIU Q, ZHANG Y, et al. A static technique for detecting input validation vulnerabilities in Android apps[J]. Science China Information Sciences, 2017, 60(5): 1-16.

[21] YAMAGUCHI F, LINDNER F, RIECK K. Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning[C]//Proceedings of the 5th USENIX Conference on Offensive Technologies. [S. l.: s. n.], 2011: 13-23.

[22] YAMAGUCHI F, LOTTMANN M, RIECK K. Generalized vulnerability extrapolation using abstract syntax trees[C]//Proceedings of the 28th Annual Computer Security Applications Conference. [S. l.: s. n.], 2012: 359-368.

[23] PANG Y, XUE X, NAMIN A S. Predicting vulnerable software components through N-Gram analysis and statistical feature selection[C]//Proceedings of the 14th International Conference on Machine Learning and Applications(ICMLA). [S. l.]: IEEE, 2015: 543-548.

[24] SCANDARIATO R, WALDEN J, HOVSEPYAN A, et al. Predicting vulnerable software components via text mining[J]. IEEE Transactions on Software Engineering, 2014, 41(10): 993-1006.

[25] LI Z, ZOU D, TANG J, et al. A comparative study of deep learning-based vulnerability detection system[J]. IEEE Access, 2019, 7: 103184-103197.

[26] ZOU D, WANG S, XU S, et al. VulDeePecker: a deep learning-based system for multiclass vulnerability detection[J]. IEEE Transactions on Dependable and Secure Computing, 2019, 18(5): 2224-2236.

[27] KRONJEE J, HOMMERSOM A, VRANKEN H. Discovering software vulnerabilities using data-flow analysis and machine learning[C]//Proceedings of the 13th International Conference on Availability, Reliability and Security. [S. l.: s. n.], 2018: 1-10.

[28] ZHOU Y, LIU S, SIOW J, et al. Devign: effective vulnerability identification by learning comprehensive program semantics via graph neural networks[C]//Proceedings of Neural Information Processing Systems. [S. l.: s. n.], 2019: 10197-10207

[29] WANG H, YE G, TANG Z, et al. Combining graph-

based learning with automated data collection for code vulnerability detection[J]. IEEE Transactions on Information Forensics and Security, 2020, 16: 1943-1958.

[30] LI Z, ZOU D, XU S, et al. VulDeeLocator: a deep learning-based fine-grained vulnerability detector[J]. IEEE Transactions on Dependable and Secure Computing, 2021, 19(4): 2821-2837.

[31] GRIECO G, DINABURG A. Toward smarter vulnerability discovery using machine learning[C]//Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security. [S. l.: s. n.], 2018: 48-56.

[32] LE T, NGUYEN T, LE T, et al. Maximal divergence sequential autoencoder for binary software vulnerability detection[C]//Proceedings of 2019 International Conference on Learning Representations. [S. l.: s. n.], 2019: 20-25.

[33] DULLIEN T, ROLLES R. Graph-based comparison of executable objects[J]. SSTIC, 2005, 5(1): 1-3.

[34] NG B H, PRAKASH A. Expose: discovering potential binary code re-use[C]//Proceedings of the 37th Annual Computer Software and Applications Conference. [S. l.]: IEEE, 2013: 492-501.

[35] LUO L, MING J, WU D, et al. Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection [C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. [S. l.: s. n.], 2014: 389-400.

[36] ALRABAEE S, WANG L, DEBBABI M. BinGold: towards robust binary analysis by extracting the semantics of binary code as semantic flow graphs(SFGs) [J]. Digital Investigation, 2016, 18: 11-22.

[37] FENG Q, ZHOU R, XU C, et al. Scalable graph-based bug search for firmware images[C]//Proceedings of 2016 ACM SIGSAC Conference on Computer and Communications Security. [S. l.: s. n.], 2016: 480-491.

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

[39] GAO J, YANG X, FU Y, et al. VulSeeker: a semantic learning based vulnerability seeker for cross-platform binary[C]//Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering(ASE). [S. l.]: IEEE, 2018: 896-899.

[40] TUFANO M, WATSON C, BAVOTA G, et al. Deep learning similarities from different representations of source code[C]//Proceedings of the 15th International Conference on Mining Software Repositories(MSR). [S. l.]: IEEE, 2018: 542-553.

[41] MARASTONI N, GIACOBAZZI R, DALLA PREDA M. A deep learning approach to program similarity [C]//Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis. [S. l.: s. n.], 2018: 26-35.

[42] LIU B, HUO W, ZHANG C, et al. αDiff: cross-version binary code similarity detection with DNN[C]// Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering. [S. l.: s. n.], 2018: 667-678.

[43] ZUO F, LI X, YOUNG P, et al. Neural machine translation inspired binary code similarity comparison beyond function pairs[C]//Proceedings of Network and Distributed Systems Security Symposium 2019. [S. l.: s. n.], 2019: 242-260.

[44] MASSARELLI L, LUNA G A D, PETRONI F, et al. Safe: self-attentive function embeddings for binary similarity[C]//Proceedings of International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. [S. l.]: Springer, Cham, 2019: 309-329.

[45] DING S H H, FUNG B C M, CHARLAND P. Asm2vec: boosting static representation robustness for binary clone search against code obfuscation and compiler optimization[C]//Proceedings of 2019 IEEE Symposium on Security and Privacy(SP). [S. l.]: IEEE, 2019: 472-489.

[46] TIAN D, JIA X, MA R, et al. BinDeep: a deep learning approach to binary code similarity detection[J]. Expert Systems with Applications, 2021, 168: 114348.

[47] WANG Y, SHEN J, LIN J, et al. Staged method of code similarity analysis for firmware vulnerability detection[J]. IEEE Access, 2019, 7: 14171-14185.

[48] YANG S, CHENG L, ZENG Y, et al. Asteria: deep learning-based AST-encoding for cross-platform binary code similarity detection[C]//Proceedings of the 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). [S. l.]: IEEE, 2021: 224-236.

[49] SUN H, CUI L, LI L, et al. VDSimilar: vulnerability detection based on code similarity of vulnerabilities and patches[J]. Computers & Security, 2021, 110: 102417.

[50] WU Y M, ZOU D Q, DOU S H, et al. VulCNN: an

image-inspired scalable vulnerability detection system [C]//Proceedings of the 44th International Conference on Software Engineering. [S. l.]: IEEE, 2022: 2365-2376.

[51] PERL H, DECHAND S, SMITH M, et al. Vccfinder: finding potential vulnerabilities in open-source projects to assist code audits[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. [S. l.: s. n.], 2015: 426-437.

[52] ESCHWEILER S, YAKDAN K, GERHARDS-PADILLA E. DiscovRE: efficient cross-architecture identification of bugs in binary code[C]//Proceedings of 2016 Network and Distributed System Security Symposium. [S. l.: s. n.], 2016: 58-79.

[53] GEUNWOO K, SANGHYUN H, MICHAEL F, et al. Improving cross-platform binary analysis using representation learning via graph alignment[C]//Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. [S. l.: s. n.], 2022: 151-163.

[54] MENGIN E, ROSSI F. Binary diffing as a network alignment problem via belief propagation[C]//Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). [S. l.]: IEEE, 2021: 967-978.

[55] YANG S G, CHENG L, ZENG Y C, et al. Asteria: deep learning-based ast-encoding for cross-platform binary code similarity detection[C]//Proceedings of the 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks. [S. l.]: IEEE, 2021: 224-236.

[56] LI Y, GU C, DULLIEN T, et al. Graph matching networks for learning the similarity of graph structured objects[C]//Proceedings of International Conference on Machine Learning. [S. l.: s. n.], 2019: 3835-3845.

[57] YU L, LU Y L, SHEN Y, et al. PBDiff: neural network based program-wide diffing method for binaries [J]. Mathematical Biosciences and Engineering, 2022, 19(3): 2774-2799.

[58] YU L, LU Y L, SHEN Y, et al. A heuristic local-sensitive program-wide diffing method for IoT binary files[J]. Arabian Journal for Science and Engineering, 2022, 47(8): 9713-9725.

[59] YU L, LU Y L, SHEN Y, et al. BEDetector: a two-channel encoding method to detect vulnerabilities based on binary similarity [J]. IEEE Access, 2021, 9: 51631-51645.

[60] WANG J, CHEN B, WEI L, et al. Skyfire: data-driven seed generation for fuzzing[C]//Proceedings of 2017 IEEE Symposium on Security and Privacy(SP). [S. l.]: IEEE, 2017: 579-594.

[61] CHEN Y, AHMADI M, WANG B, et al. MEUZZ: smart seed scheduling for hybrid fuzzing[C]//Proceedings of the 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020). [S. l.: s. n.], 2020: 77-92.

[62] SHE D, PEI K, EPSTEIN D, et al. NEUZZ: efficient fuzzing with neural program smoothing[C]//Proceedings of 2019 IEEE Symposium on Security and Privacy(SP). [S. l.]: IEEE, 2019: 803-817.

[63] LI Y, JI S, LYU C, et al. V-fuzz: vulnerability prediction-assisted evolutionary fuzzing for binary programs[J]. IEEE Transactions on Cybernetics, 2020, 52(5): 3745-3756

[64] GODEFROID P, PELEG H, SINGH R. Learn &fuzz: machine learning for input fuzzing[C]//Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). [S. l.]: IEEE, 2017: 50-59.

[65] LIU X, LI X, PRAJAPATI R, et al. Deepfuzz: automatic generation of syntax valid C programs for fuzz testing[C]//Proceedings of AAAI Conference on Artificial Intelligence. [S. l.: s. n.], 2019: 1044-1051.

[66] JITSUNARI Y, ARAHORI Y. Coverage-guided learning-assisted grammar-based fuzzing [C]//Proceedings of 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops(ICSTW). [S. l.]: IEEE, 2019: 275-280.

[67] GONG W, ZHANG G, ZHOU X. Learn to accelerate identifying new test cases in fuzzing[C]//Proceedings of International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage. [S. l.]: Springer, Cham, 2017: 298-307.

[68] ZONG P, LV T, WANG D, et al. FuzzGuard: filtering out unreachable inputs in directed grey-box fuzzing through deep learning [C]//Proceedings of the 29th USENIX Security Symposium. [S. l.: s. n.], 2020: 2255-2269.

[69] SUN X, FU Y, DONG Y, et al. Improving fitness function for language fuzzing with PCFG model[C]//Proceedings of the 42nd Annual Computer Software and Applications Conference (COMPSAC). [S. l.]: IEEE, 2018: 655-660.

[70] KARAMCHETI S, MANN G, ROSENBERG D. Adaptive grey-box fuzz-testing with thompson sampling [C]//Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security. [S. l.: s. n.], 2018: 37-47.

[71] LI X, LIANG Y, QIAN H, et al. Symbolic execution of complex program driven by machine learning based constraint solving[C]//Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering(ASE). [S. l.]: IEEE, 2016: 554-559.

[72] CHEN J, HU W, ZHANG L, et al. Learning to accelerate symbolic execution via code transformation [C]//Proceedings of the 32nd European Conference on Object-Oriented Programming(ECOOP 2018). [S. l. : s. n.], 2018: 1-27.

[73] CHEN P, CHEN H. Angora: efficient fuzzing by principled search[C]//Proceedings of 2018 IEEE Symposium on Security and Privacy(SP). [S. l.]: IEEE, 2018: 711-725.

[74] SHI S, SHINDE S, RAMESH S, et al. Neuro-symbolic execution: augmenting symbolic execution with neural constraints[C]//Proceedings of 2019 Network and Distributed System Symposium. [S. l. : s. n.]: 2019: 217-230.

[75] WEN S H, MOW W L, CHEN W N, et al. Enhancing symbolic execution by machine learning based solver selection[C]//Proceedings of the NDSS Workshop on Binary Analysis Research. [S. l. : s. n.], 2019: 125-140.

[76] BU L, LIANG Y, XIE Z, et al. Machine learning steered symbolic execution framework for complex software code [J]. Formal Aspects of Computing, 2021, 33(3): 301-323.

[77] HE J, SIVANRUPAN G, TSANKOV P, et al. Learning to explore paths for symbolic execution[C]//Proceedings of 2021 ACM SIGSAC Conference on Computer and Communications Security. [S. l. : s. n.], 2021: 2526-2540.

[78] RUARO N, ZENG K, DRESEL L, et al. SyML: guiding symbolic execution toward vulnerable states through pattern learning[C]//Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses. [S. l. : s. n.], 2021: 456-468.

[79] 鲁溟峰，陶然. 基于深度神经网络的对抗样本生成算法及其应用研究综述[J]. 信息对抗技术，2022，1(1)：11-23.
LU Mingfeng, TAO Ran. Survey of adversarial examples algorithms and applications based on deep neural network[J]. Information Countermeasures Technology, 2022, 1(1): 11-23. (in Chinese)

[80] 邵堃，杨俊安. 一种基于篡改训练数据的词级文本后门攻击方法[J]. 信息对抗技术，2022，1(1)：81-89.
SHAO Kun, YANG Jun'an. A word-level textual backdoor attack method based on tampering with training data[J]. Information Countermeasures Technology, 2022, 1(1): 81-89. (in Chinese)

[81] GU T, LIU K, DOLAN-GAVITT B, et al. Badnets: evaluating backdooring attacks on deep neural networks[J]. IEEE Access, 2019, 7: 47230-47244.

[82] GROSSE K, LEE T, BIGGIO B, et al. Backdoor smoothing: demystifying backdoor attacks on deep neural networks[J]. Computers & Security, 2022, 120: 102814.

[83] ZHANG J, C D D, HUANG Q D, et al. Poison ink: robust and invisible backdoor attack[J]. IEEE Transactions on Image Processing, 2022, 31: 5691-5705.

作 者 简 介

陆余良

男，1964 年生，教授，博士研究生导师，研究方向为软件与系统安全、网络态势感知、大数据分析

E-mail：luyuliang@nudt.edu.cn

于　璐

女，1985 年生，博士，讲师，研究方向为软件与系统安全

E-mail：yulu@nudt.edu.cn

赵家振

男，1996 年生，博士研究生，研究方向为软件安全、程序分析

E-mail：jiazhenzhao@nudt.edu.cn

责任编辑　董　莉