基于图深度学习的漏洞检测

董继平[1,2,3],郭启全[1,2],高春东[2],郝蒙蒙[1,2,3],江东[1,2,3*]

1. 中国科学院地理科学与资源研究所,北京 100101
2. 中国科学院公安部网络空间地理学实验室,北京 100101
3. 中国科学院大学资源与环境学院,北京 100190

摘要　图深度学习技术在处理非欧氏结构数据中显示了巨大潜力,大量研究工作尝试将图嵌入或图神经网络应用到漏洞检测中。梳理了基于图深度学习的漏洞检测方法,按其一般流程,归纳了数据集、图数据、图深度学习模型构建及结果评估4个主要阶段;从图深度学习漏洞检测的有效性出发,阐述了基于代码模式和基于相似性及具体应用场景中的研究成果;分析了该领域面临的挑战和未来的趋势。

关键词　网络安全;漏洞检测;图深度学习;图嵌入;图神经网络

随着数字化技术的高速发展,信息技术基础设施、计算机网络和数字设备在社会、经济、军事等各方面发挥着越来越重要的作用,互联网与各行各业的深度融合,变革了产业格局、提高了生产效率。与此同时,计算机软件(如web应用、计算机程序、操作系统等)中潜在的安全问题正成为一个新兴的全球性挑战,其中安全漏洞是问题的根源之一[1-5]。

近年来,软件漏洞不仅在数量上呈上升趋势,其形态也表现出复杂性和多样性的特点[6]。美国国家漏洞数据库(National Vulnerability Database, NVD)中记录的每年新增漏洞数量从2016年的6447条记录增长到2021年的20158条记录,已经连续5年创下新增数量纪录[7]。利用漏洞进行网络犯罪是攻击者常用的手段,给个人、企业乃至政府带来巨大损失。2021年11月由阿里云安全团队向Apache软件基金会披露的Log4Shell(CVE-2021-44228)是开源日志框架Log4j中的一个零日漏洞,它广泛用于Java应用程序,尤其是企业软件。黑客可以利用该漏洞操控被攻击者设备挖掘加密货币、创建僵尸网络、发送垃圾邮件、建立后门,以及进行

收稿日期:2022-10-31;修回日期:2022-11-19
基金项目:中国科学院重点部署项目(ZDRW-XH-2021-3)
作者简介:董继平,博士研究生,研究方向为地理大数据与智能认知,电子信箱:dongjiping2017@igsnrr.ac.cn;江东(通信作者),研究员,研究方向为地理大数据与智能认知,电子信箱:jiangd@igsnrr.ac.cn
引用格式:董继平,郭启全,高春东, 等. 基于图深度学习的漏洞检测[J]. 科技导报, 2023, 41(13): 41-59; doi: 10.3981/j.issn.1000-7857.2023.13.005

其他非法活动,预计会影响数亿台设备,造成数十亿美元的损失[8]。

缓解网络安全威胁的有效方法是尽快找到并修复漏洞。传统的基于机器学习或深度学习的漏洞检测通常将代码函数体、抽象语法树或代码属性图等特征信息转为非结构化的序列流,这忽略了数据流和控制流中丰富的语义信息,无法有效利用程序在执行过程中的执行逻辑和调用关系等结构信息,从而导致大量误报[9]。图深度学习,尤其是图神经网络的发展极大促进了图表示学习在现实场景中的广泛应用,例如蛋白质预测、医学诊断、交通预测、推荐系统等[10-11]。在网络安全领域,研究人员也尝试使用图深度学习技术(如图嵌入或图神经网络)建立基于图结构的漏洞检测模型。通过将源代码表示为控制流程图、程序依赖图等中间图结构,可以与图学习完美匹配,因而能更好地捕获程序的控制、数据调用和依赖等代码间的关系,显著提高漏洞检测准确率和检测效率[12-13]。但是,目前大多数漏洞检测调研主要集中在传统机器学习或深度学习领域[5,14-16],尽管有部分文献提及了基于图深度学习的漏洞检测,但并没有对此进行详细的介绍[6,13,17-20]。

基于图深度学习的漏洞检测的文章逐年增加,越来越多的研究人员对此开展了广泛研究,利用图嵌入或图神经网络进行漏洞检测已经成为当前该领域的研究热点之一。因此,本研究重点关注基于图深度学习的漏洞检测方法,在调研了漏洞检测流程的基础上,总结了数据集、图数据和图深度学习模型构建,以及结果评估4个主要阶段的研究进展,重点对比了基于代码模式和基于相似性的漏洞检测方法,并从数据集、被分析对象、代码表征生成、图模型构建等方面进行了总结、分析与展望。

1 基于图深度学习的漏洞检测流程

利用图深度学习进行漏洞检测包括数据集构建、图数据构建、图深度学习模型构建和训练评估4个阶段(图1)。其中,图数据构建一般包括中间图表示和特征向量生成2个过程:首先,提取程序源代码中的语法和语义信息,解析程序中的数据流、控制流等信息,得到代码的中间图表示;然后,对此图结构数据进行嵌入(embedding)操作,生成图的初始特征向量作为图神经网络的输入。训练评估涉及训练和检测2个阶段:训练阶段将图向量数据作为输入,使用图神经网络学习节点特征或图特征;检测阶段对待检测代码进行相同的数据处理,将其图向量数据作为输入,经过图神经网络模型输出预测结果,完成对待检测代码漏洞的预测。

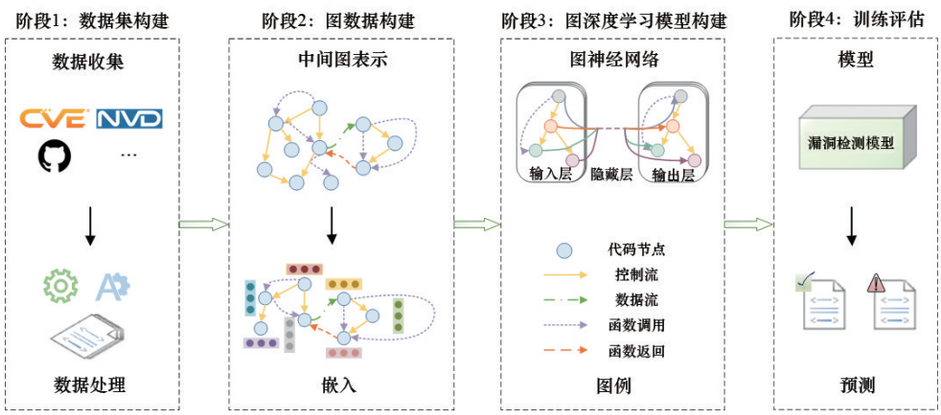

图1 基于图深度学习的漏洞检测工作流程

1.1 数据集构建

数据收集和整理是漏洞检测研究的初步任务,数据集的数量和质量将直接影响图深度学习模型的有效性,是训练和评估模型的关键。本节将数据集分为合成数据、半合成数据、真实数据3类,并对漏洞检测中常用的公开数据集进行了总结(表1)。

现有的用于漏洞检测的公共数据集主要源自SATE(static analysis tool exposition) IV Juliet[21]、SARD(software assurance reference dataset)[22]和NVD[23]。其中,SATE IV Juliet数据集根据CVE数据集[24]和大约6万合成测试用例构建而成;SARD数据集包含近20万测试程序,涵盖150多个漏洞类型;NVD数据集与CVE数据集完全同步,并提供了漏洞类型等其他信息。科研人员利用这3种数据集构建了如SeVC、Draper MUSE等漏洞检测数据集。Li等[25]从NVD和SARD收集了15591个C/C++程序,构建了SeVC漏洞数据集,共包括126种类型的漏洞,其中每种类型都由CWE ID唯一标识。Russell等[26]从SATE IV Juliet、Debian Linux和GitHub公开代码仓库收集了数百万个C++代码,通过词法分析、数据清洗和标签标注一系列过程构建了Draper MUSE漏洞数据集。

表1　漏洞检测公开数据集

类型	数据集	数据组成	源码语言	年份	文献
通用漏洞披露	CVE[24]				
合成数据集	SATEIV Juliet[21]		C/C++, Java, PHP	2013	
	SARD[22]		C, C++, Java, PHP, C#		
半合成数据集	NVD[23]				
	SeVC[25]	SARD, NVD	C/C++	2022	[37]
	DraperMUSE[26]	SATEIV Juliet, Debian Linux, GitHub	C/C++	2018	[38-39]
	DeepTective[27]	SARD, GitHub	PHP	2021	[27]
	DeepWukong[28]	SARD, lua, redis	C/C++	2021	[28]
	MVD[29]	SARD, CVE	C/C++	2022	[29]
真实数据集	OpenSSL[30]		C		[31,40-42]
	Gemini[31]	OpenSSL	C	2017	[31,43-44]
	Devign[32]	Linux Kernel, QEMU, Wireshark, FFmpeg	C	2019	[12, 34, 37, 39, 45-48]
	Big-Vul[33]	CVE, GitHub	C/C++	2020	[48-49]
	ReVeal[34]	Linux Debian Kernel, Chromium	C/C++	2021	[12,37,48]
	BGNN4VD[35]	Linux Kernel, FFmpeg, Wireshark, Libav	C/C++	2021	[12,35]
	[36]	Wireshark	C	2022	[36]

但是,SATE IV Juliet和SARD数据集是合成数据,通过已知漏洞模式人为合成,并没有真实性,而NVD、SeVC和Draper MUSE数据集虽是半合成数据,但经过了修改和简化,也无法完全体现真实世界漏洞的复杂性。因此,为了确保实验数据的真实性和确定性,科研人员选择从真实的项目中创建漏洞数据集。Xu等[31]针对跨平台二进制相似性检测问题收集并构建了4种数据集,分别进行神经网络模型的训练和评估、特定任务的性能评估、效率评估以及漏洞案例研究,并在GitHub上公开了数据集和代码——Gemini。Zhou等[32]从4个大型C语言开源项目(Linux Kernel、QEMU、Wireshark、FFmpeg)中收集了48687条与安全漏洞有关的提交数据,由4名专业人员进行了2轮标注和交叉验证(大约600工时),共获得大约2.3万条漏洞数据,并公开了部分数据集——FFmpeg和QEMU。Fan等[33]根据CVE数据库的漏洞信息及对应的GitHub项目链接,从348个开源GitHub项目收集了3754个代码漏洞,涉及91种不同的漏洞类型,发布了一个大型C/C++代码漏洞数据集——Big-Vul。Chakraborty等[34]详细分析了现有数据集VulDeePecker、SySeVR、FFmpeg和QEMU的局限性,根据2个开源项目Linux Debian Kernel和Chromium,分别从Debian安全漏洞追踪器和Bugzilla收集数据,构建了一个更

全面的真实数据集——ReVeal。

1.2 图数据构建

1.2.1 中间图表示

计算机程序有各种经典的中间图表示形式,使用图数据结构对程序的特定属性进行表示,可以对应用程序进行更加有效的分析[50]。在这些图结构中间表示中,图的节点代表表达式或语句,边代表控制流、控制依赖或数据依赖,不同的图结构可以视为从程序的不同特征角度来描述程序[51]。抽象语法树、控制流程图、程序依赖图、代码属性图、属性控制流程图等已广泛应用于图深度学习漏洞检测,表2对各种程序代码中间图表示进行了汇总。

抽象语法树(abstract syntax tree,AST)[32,52]:源代码的抽象语法结构的树表示。树中的每个节点表示源代码中出现的语句或代码行,语法是抽象的,不代表真实语法中出现的每个细节,只是结构上的与内容相关的细节。

控制流程图(control flow graph,CFG)[29,53]:用图的形式表示一个程序执行过程中会遍历到的所有路径,也能反映一个过程的实时执行过程。图中的每个节点表示代码的基本语句或代码行,通过有向边来表示从一个节点到另一个节点的控制转移。

程序依赖图(program dependence graph,PDG)[29,52]:由控制依赖图(control dependence graph,CDG)和数据依赖图(data dependence graph,DDG)2个子图组成,同时结合了程序中每个操作的数据依赖项和控制依赖项。CDG是一个有向树,节点是程序语句,每个节点对其父节点具有控制依赖性。DDG是一个多向图,节点是程序语句,边表示从源语句到目标语句的数据流。

代码属性图(code property graph,CPG)[38,45]:是一种综合了抽象语法树、控制流程图和程序依赖图的联合图数据结构,包含了程序的语法、控制、数据流等信息,可以同时编码语法和语义,是最为全面的抽象图结构之一。

属性控制流程图(attributed control flow graph,ACFG)[40,54]:是CFG的属性集合,包含了统计和结构2种类型的特征。统计特征描述了基本块内的局部特征,如字符串常量、指令数量等,结构特征捕获了基本块在CFG中的位置特征,如子节点数量、节点中心性等。

表2 代码中间图表示

中间图	全称	描述
AST	Abstract Syntax Tree(抽象语法树)	源代码的抽象语法结构的树表示
CFG	Control Flow Graph(控制流程图)	程序执行过程中会遍历到的所有路径
CDG	Control Dependence Graph(控制依赖图)	表示程序中的控制依赖关系
DFG	Data Flow Graph(数据流图)	程序执行过程中变量的访问或修改
DDG	Data Dependence Graph(数据依赖图)	表示程序中的数据依赖关系
CDFG	Control Data Flow Graph(控制数据流图)	表示程序中的控制流程和数据依赖关系
FCG	Function Call Graph(函数调用图)	表示程序中子程序之间的调用关系
ACFG	Attributed Control Flow Graph(属性控制流程图)	CFG的属性集合,包含了统计特征和结构特征
NCS	Natural Code Sequence(自然代码序列)	源代码词符(token)序列
PDG	Program Dependence Graph(程序依赖图)	组合CDG, DDG
CPG	Code Property Graph(代码属性图)	组合AST, CFG, PDG
	Composite Code Semantics(复合代码语义图)	组合AST, CFG, DFG, NCS
SPG	Slice Property Graph(切片属性图)	组合DDG, CDG, FCG
CCG	Code Composite Graph(代码复合图)	组合AST, CFG, DFG

除了上述经典的图结构,针对不同的漏洞类型、漏洞模式等特点,研究人员也尝试利用自然代码序列(natural code sequence,NCS)、数据流图(data flow graph,DFG)、调用图(call graph)等表示不同的结构化或非结构化信息,并将多种信息进行融合,构建了一系列新颖的图结构表示形式。Wu等[55]将代码函数表示为简化代码属性图(simplified code property graphs,SCPG),既可以保留源代码的

句法和语义信息,也保证了较小的计算量。Zhou等[32]提出了复合代码语义图(composite code semantics),整合了抽象语法树、控制流程图、数据流图、自然代码序列,将函数源代码编码为具有综合语义信息的联合图结构。Ye等[9]利用多个图来表示程序的句法和语义结构,从程序源代码构建抽象语法树和控制数据流图(control data flow graph,CDFG),并使用附加边对AST进行扩展,使其同时携带数据流和控制流信息。类似地,Wang等[56]对抽象语法树和程序控制依赖图(program control dependence graph,PCDG)进行组合,构建了程序图(program graph)。Zheng等[37]将漏洞代码中的数据依赖图、控制依赖图和函数调用依赖图编码为图数据结构,设计了切片属性图(slice property graph,SPG),可以同时保留与漏洞相关的语义和结构信息。Cao等[35]将抽象语法树、控制流程图和数据流图进行组合,提出了代码复合图(code composite graph,CCG),可以同时捕获源代码的各种语法和语义信息,反映漏洞语句和非漏洞语句之间的语法和语义差异。Ji等[41]设计了二进制代码的图表示——属性函数调用图(attributed function call graph,AFCG),涵盖了指令级别、函数级别,以及二进制级别3种类型的代码特征。

1.2.2　特征向量生成

在获得中间图之后,可以使用解析器程序提取图数据,如邻接矩阵、节点属性、边属性、图标签等[52]。其中,节点属性是代码的表达式或语句,一般为文本格式,因此需要进行编码或嵌入以将文本转为向量,作为神经网络的实际输入。常见的编码模型有One-Hot、TF-IDF、Word2Vec、Doc2Vec、Transformer等。

One-Hot[57]:又称一位有效编码,主要采用N维向量对N个文本单词进行一一对应编码,任何时候只有词语所在下标位置的值为1,所有其他位置的值均为0。因此,在文本单词过多时,会造成矩阵稀疏和维度灾难,且无法保持词语间语义上的相似性。

TF-IDF[52]:即词频—逆向文本频率(term frequency-inverse document frequency),是一种统计方法,可以评估一个字词对于一个文本集的重要性,其重要性与该字词在文本中出现的次数成正比,与其在整个文本集中出现的次数成反比。但是,该方法没有考虑词与词之间的相互关系,也无法体现词的位置信息。

Word2Vec[57]:从大量文本语料中以无监督方式学习语义知识,是一种静态词嵌入方法,包括连续词袋模型(continuous bag of words,CBOW)和跳字模型(Skip-gram)2种算法。CBOW算法适用于小样本数据集,以周围词作为输入预测中心词,Skip-gram算法适用于数据量较大的情况下,根据中心词预测其对应的周围词。Word2Vec可以为语义相似的单词分配距离相近的向量,但是仍没有解决一词多义问题。

Doc2Vec[58]:基于Word2Vec提出,可以通过无监督方式从变长的文本(如句子、段落和文档)中学习到固定长度的特征表示。类似于Word2Vec中的CBOW算法和Skip-gram算法,Doc2Vec也具有PV-DM(distributed memory model of paragraph vectors)和PV-DBOW(distributed bag of words of paragraph vector)2种训练方法。

Transformer[57]:其架构采用自注意力(self-attention)机制,通过关联文本序列的不同位置学习每个字词的表示。GPT、Bert、XLNet等预训练模型以Transformer为核心,采用动态表征,解决了一词多义问题,具有更强的上下文词表示能力。相比RNN等时序类模型,基于Transformer架构的编码模型能并行化计算且能捕捉更长距离的依赖关系。

1.3　图深度学习模型构建

对于图数据而言,图嵌入(graph embedding,network embedding)和图神经网络(graph neural networks,GNN)是2个类似的研究领域。图嵌入旨在将图中节点表示为低维向量,并同时保留图的拓扑结构信息和节点内容信息,以便使用机器学习算法进行后续的图分析任务(如分类、聚类、推荐等);而图神经网络是一种深度学习模型,旨在以端到端方式解决与图相关的任务[59]。许多图嵌入算法通常是无监督算法,可以大致划分为矩阵分解、随机游走和深度学习3个类别[60],其中使用深度学习技术的图嵌入也属于图神经网络方法,比如基于图自

动编码器的算法、图卷积网络等[61]。

基于随机游走的图嵌入方法利用拓扑相似性(或邻近性)来学习节点的向量表征,通常采用不同的游走策略生成节点序列,获取图的局部结构和全局结构信息,再利用Skip-gram算法完成图中节点的降维嵌入,例如DeepWalk算法[62]、Node2Vec算法[63]、Metapath2Vec算法[64]等。但是,这些基于游走的图嵌入模型只考虑了图的结构特征,并没有考虑节点的属性、文字等信息,而且这些方法只能生成训练阶段中已存在节点的嵌入,这限制了此类模型的泛化能力[65]。

为解决这些问题,科研人员提出了图神经网络和一些变体,这些基于GNN架构的模型使用了更复杂的编码器,可以将节点特征输入到模型,同时学习图结构信息和节点属性信息。具体而言,图神经网络利用消息传递的形式进行节点间的消息交换和更新,在计算或更新目标节点$v\in V$表示$h_v^{(l)}$时,需要根据网络的层数L(邻居的跳数)确定局部网络图,再通过迭代转换并聚合相邻节点$u\in N(v)$的表示来更新该节点的表示[66]。通过设计不同的消息函数、聚合函数、更新函数、读出函数可以实现不同的图神经网络模型,例如图卷积网络、图注意力网络、门控图神经网络等。表3总结了漏洞检测任务中常用的图深度学习模型架构。图卷积神经网络(graph convolutional network,GCN)[66-68]:可分为基于谱域的方法和基于空域的方法,是最简单的GNN形式。基于谱域的GCN近似图拉普拉斯算子的一阶特征分解,通过迭代聚合来自邻居的信息;基于空域的GCN由卷积神经网络延伸而来,它将中心节点的表示与其相邻节点的表示进行卷积,得到中心节点的更新表示,其本质是沿边传播节点信息。

GraphSAGE(graph sample and aggregate)[66,68-69]:对基于空域的GCN进行了改进,将GCN的全图采样优化为对目标节点部分邻居的采样,通过训练一系列不同深度的聚合函数来学习节点邻居的聚合特征。这样,对于新加入的节点,可以通过对其邻居进行聚合来给出该节点的表示,而不必对整个网络重新迭代学习。此外,GraphSAGE提出了3个聚合函数,分别是平均(mean)聚合、LSTM聚合及池化(pooling)聚合。

图注意力网络(graph attention network,GAT)[68,70]:对于目标节点而言,每个邻居节点对该节点的影响可能并不相同,直接将邻居的信息简单相加并不是一种理想的聚合操作,可以采用注意力机制给不同邻居赋予不同权重来判断不同邻居的重要性。此外,应用多个独立的注意力机制(即多头注意力机制)计算隐含状态,通过拼接或平均得到输出表示以达到稳定学习过程的目的。

门控图神经网络(gated graph neural network,GGNN)[68,71]:加入了门控循环单元(gated recurrent unit,GRU),把邻居节点的信息作为输入,节点本身的状态作为隐藏状态,可以使模型选择性记忆节点及其邻居节点的隐藏信息,提高图结构信息的长期(long-term)传播能力。

表3 漏洞检测任务中图深度学习模型架构

模型架构	全称	文献
GNN	Graph Neural Network(图神经网络)	[28-29,72]
GCN	Graph Convolutional Network(图卷积网络)	[27-28,36,39,47-49,50,52-53,55,73-76]
GAT	Graph Attention Network(图注意力网络)	[28,41,43,45-46,49-50,52-53,55,77-79]
GGNN	Gated Graph Neural Network(门控图神经网络)	[32,34-35,38,47,53,55-56,80]
GEN	Graph Embedding Network(图嵌入网络)	[31,40,42,44,77,81-83]
RGCN	Relational Graph Convolutional Network(关系图卷积网络)	[9,37,84]
GIN	Graph Isomorphism Network(图同构神经网络)	[85]

1.4 评估指标

一般情况下,漏洞检测是二分类任务,共有4种可能的结果,分别是TP(true positives,真阳性)、FP(false positives,假阳性)、TN(true negative,真阴性)、FN(false negative,假阴性),可以用混淆矩阵描述(表4)。其中,TP表示是漏洞且被检测为漏洞

表4　混淆矩阵

预测值	真实值	
	Positive	Negative
Positive	TP(true positives 真阳性)	FP(false positives 假阳性)
Negative	FN(false negatives 假阴性)	TN(true negatives 真阴性)

的样本数量,FP表示非漏洞但被检测为漏洞的样本数量,TN表示非漏洞且被检测为非漏洞的样本数量,FN表示是漏洞但被检测为非漏洞的样本数量。常用的评估指标包括准确率、精确率、召回率、F1-score、ROC-AUC等[35,42-43,46,77]。

此外,上述指标虽然使用广泛,但对于类别不平衡问题并不能很好地处理。当数据失衡时(负例多于正例),各模型间ROC-AUC的差异较小,PR-AUC更能体现模型之间的差异。该指标是指精确率-召回率曲线与横轴围成的面积,其面积越大,模型分类效果越好[86]。Informedness(IFN)和Markedness(MKN)分别是召回率和精确率的无偏评估指标(取值范围为-1~1),更适用于类别不平衡的数据集[87]。马修斯相关系数(Matthews correlation coefficient,MCC)同时考虑了混淆矩阵中的真阳性、假阳性、真阴性和假阴性,描述了实际分类与预测分类之间的相关系数(取值范围为-1~1),是一个比较均衡的指标[87]。上述各评估指标的公式及总结如表5所示。

表5　常用评估指标

评估指标	描述	计算公式
准确率(Accuracy)	正确预测的样本数量与样本总数的比例	$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}$
精确率(Precision)	在所有被预测为正的测试数据中,真正类所占的比例	$Precision = \frac{TP}{TP + FP}$
召回率(Recall)	在所有实际为正的测试数据中,真正类所占的比例	$Recall = \frac{TP}{TP + FN}$
F1-score	一个综合指标,用于平衡准确率和召回率的影响	$F1 = 2 \times \frac{Precition \times Recal}{Precision + Recal}$
AUC-ROC	假阳性率-真阳性率曲线与横轴围成的面积	
PR-AUC	精确率-召回率曲线与横轴围成的面积	
Informedness(IFN)	召回率的无偏评估指标,同时考虑实际为正和实际为负的样本	$Inverse\,Recall = \frac{TN}{TN + FP}$ $IFN = Recall + Inverse\,Recall - 1$
Markedness(MKN)	精确率的无偏评估指标,同时考虑预测为正和预测为负的样本	$Inverse\,Precision = \frac{TN}{TN + FN}$ $MKN = Precision + Inverse\,Precision - 1$
马修斯相关系数(Matthews correlation coefficient,MCC)	描述了实际分类与预测分类之间的相关系数	$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$

2　图深度学习在漏洞检测中的应用

目前,科研人员已开展了诸多将代码表示为图结构并利用图深度学习方法进行漏洞检测的研究。我们对近年来的工作进行了调研,把基于图深度学习的漏洞检测方法分成了基于代码模式的漏洞检测和基于代码相似性的漏洞检测2大类,本节将对这2部分内容进行详细地阐述。此外,为了表明图深度学习在漏洞检测上的有效性以及其广阔的适用场景,我们在2.1节和2.4节分别对其进行介绍。

2.1 基于图深度学习的漏洞检测方法

为验证图神经网络在漏洞检测中的有效性,Ghaffarian等[50]通过诸多实验详细讨论了图神经网络在不同情景(如图表示形式、代码属性嵌入方式、跨项目检测等)下的表现。笔者从OWASP基准测试套件构建了漏洞数据集,开发了PROGEX工具来提取程序的图形表示(AST、CFG和PDG等),然后采用TF-IDF和Doc2Vec将属性图节点转为向量表示,并利用GCN和GAT进行训练。结果表明,基于图神经网络的漏洞分析系统大大优于基于传统深度神经网络的基线方法,所需的训练样本数量更少,而且能有效进行跨项目漏洞分析。

Chakraborty等[34]对基于深度学习的漏洞检测进行了系统性研究,发现已有数据集过于简单且无法代表现实世界的漏洞,而且现有模型并不能完全解决漏洞检测中的代码语义表征和数据不平衡问题,这导致深度学习模型在真实漏洞数据集上的预测性能下降了50%以上。为此,笔者提出了一个用于收集真实世界漏洞数据集的框架,发布了一个更全面漏洞数据集ReVeal;对于模型训练,利用图神经网络进行特征提取,通过重采样平衡训练数据集,并在平衡数据上训练表示学习模型(多层感知机),从而实现了对漏洞数据和非漏洞数据更好的区分。结果表明,通过上述改进可以将模型的精确率和召回率分别提高33.57%和128.38%。

尽管图深度学习在漏洞检测中是一个有效工具,但由于图神经网络是黑盒模型,其决策过程对安全专家完全不透明,这阻碍了在实践中的进一步应用。针对此问题,Ganz等[12]提出了一个评估GNN解释方法的框架,开发了一套评估标准(如图稀疏性、结构鲁棒性、稳定性等),对9种常规解释方法和3种图解释方法进行了实验研究。研究表明,解释GNN是一项艰巨的任务,所有评估标准都有着或大或小的作用,但是进行图解释可以更好地了解代码语义,为安全专家提供更多信息。

2.2 基于代码模式的漏洞检测模型

利用代码模式进行漏洞检测可基于词法分析或语法分析进行。前者采用文本挖掘技术,对源代码的标识符、函数名和运算符等进行标记,利用编码模型进行向量化获得代码特征;后者利用静态分析技术,依据抽象语法树、控制流程图、程序依赖图、代码属性图等提取特征,对源代码的控制流、数据流等进行更深层次的表示[5-6,20]。表6对基于代码模式的漏洞检测模型进行了汇总。

基于图神经网络的漏洞检测,一般可视为节点分类或图分类问题。前者通过学习节点特征,综合考虑节点及邻居特征进行节点分类,可用于语句级的漏洞检测,后者从图级学习代码的图结构数据表示进行图分类,可用于切片级的漏洞检测。Cheng等[74]提出了VGDetector,用于高级控制流相关(high-level control-flow related,CFR)的漏洞检测。该模型将代码程序转为抽象语法树,并为每个函数构造控制流程图(CFG),然后使用Doc2Vec技术将CFG上的每个代码块编码为向量,最后使用图卷积网络完成训练。结果表明,对于所有3个CFR漏洞,VGDetector的平均检测准确率超过91%。Rabheru等[27]提出了DeepTective,用于检测PHP源代码中的漏洞。该模型将GRU和GCN结合,利用GRU分析词符(token)序列学习源代码的句法和结构信息,使用GCN分析控制流程图学习源代码的语义信息。实验证明,DeepTective在合成数据集和真实数据集上达到最先进的分类性能,并从WordPress插件中检测到4个新型安全漏洞。Cao等[29]针对与内存有关的漏洞提出了语句级漏洞检测方法——MVD,采用FS-GNN(flow-sensitive graph neural network),利用程序依赖图同时嵌入代码程序的非结构化信息(源代码)和结构化信息(控制流和数据流),可以保留更高层的程序语义信息,学习隐式漏洞模式。实验结果表明,MVD表现优于深度学习方法和静态分析方法,实现了更好的漏洞检测准确率。但是,上述漏洞检测模型并没有在连续时间变化上进行代码细粒度的实时分析,为此Şahin等[36]提出了一种漏洞检测方法,通过GCN和GraphSAGE进行预测,可以自动检查代码更改是否会引起漏洞,并可以方便地集成到版本控制系统中。

在此基础上采用注意力机制,可以对输入的每个部分分配不同的权重使网络关注更多关键信息,从而使模型的预测更准确,也可避免过高的模型计

表6　基于代码模式的漏洞检测模型

模型	年份	数据集	分析对象	跨项目	中间图表示	编码方式	模型构造
VGDetector[74]	2019	SARD		否	CFG	Doc2Vec	GCN
DeepTective[27]	2021	SARD, GitHub PHP项目	PHP	否	CFG	phply	GRU, GCN
MVD[29]	2022	SARD, CVE	C/C++	是	PDG	Doc2Vec	Flow-Sensitive GNN
[36]	2022	Wireshark	C	否	AST	Word2Vec	GCN, GraphSAGE
GraphEye[78]	2021	SARD	C/C++	否	CPG	One-Hot	GAT, SAGpool
DeepWukong[28]	2021	SARD, lua, redis	C/C++	否	PDG	Doc2Vec	GCN, GAT, k-GNN
Vulmg[79]	2021	SARD		否	CPG, FCG	One-Hot	Multi-Head GAT
LineVD[49]	2022	CVE, GitHub项目	C/C++	是	PDG	Transformer	GCN, GAT
SAR-GIN[85]	2021	[56]	C, C++, Java, PHP, Swift	否	AST	Word2Vec	GIN, SAGPool
IVDetect[48]	2021	[32-33]	C/C++	是	PDG	GloVe, GRU, LSTM	Feature-Attention GCN
Devign[32]	2019	Linux Kernel, QEMU, Wireshark, Ffmpeg	C	否	AST, CFG, DFG, NCS	Word2Vec	GGNN
BGNN4VD[35]	2021	Linux Kernel, FFmpeg, Wireshark, Libav	C/C++	否	AST, CFG, DFG	Word2Vec	GGNN
Poem[9]	2020	CVE, NVD, Github项目	C, Java, Swift	否	AST, CDFG	Word2Vec	Multi-Rational GNN
Funded[56]	2021	CVE, NVD, SARD, Github项目	C, C++, Java, PHP, Swift	是(跨语言)	AST, PCDG	Word2Vec	GGNN
HGVul[53]	2022	[24,33,88]		是	AST, CPG, NCS	Word2Vec	GCN, GAT, GGNN
3GNN[39]	2021	[26,32]	C/C++	否	AST, CFG, DFG	Transformer	Crystal GCN, SAGPool
ACGVD[46]	2021	[32]	C	否	CFG, DFP	Word2Vec	GAT, MLP
VulSPG[37]	2021	[25,32]	C/C++	否	DDG, CDG, FCDG	Word2Vec	Relational GCN, Triple Attention Mechanism

算和存储。此外,用多头注意机制拼接多个输出表示,可以有效稳定模型的训练过程。Zhou等[78]提出了GraphEye,主要由VecCPG和GcGAT 2部分组成。其中,VecCPG对代码属性图进行向量表示,可以同时获取源代码的关键语法和语义特征,GcGAT利用图注意力网络模型进行图分类,通过多个实验验证了GraphEye的有效性。Zhang等[79]提出了Vulmg,基于代码属性图,将源代码的词法、语法和语义信息提取为特征矩阵,并将结构、控制和依赖等信息提取为3个邻接矩阵。基于多头图注意力网络,Vulmg可以接受多个邻接矩阵作为输入来更新特征向量。此外,Vulmg使用函数调用图将调用函数与被调用函数相关联,以便进行跨函数漏洞检测。Hin等[49]提出了LineVD,使用Transformer模型同时从函数级和语句级进行嵌入,利用图注意力网络学习程序依赖图的结构化表征,通过共享全连接层解决函数级和语句级输出时产生的相互冲突。结果表明,F1-score比当前最先进模型更高,同时

在跨项目漏洞预测中也具有一定的有效性。Xia等[85]首次尝试使用图同构网络(graph isomorphism network,GIN)进行漏洞检测,提出了SAR-GIN。该模型引入标准离散傅立叶变换,以关注更多的有效信息,利用注意力机制为每一层网络分配不同的注意力权重,以解决节点表示过平滑以及图级特征不佳的问题,能有效提取和利用所有层的全局信息。但是,上述几种漏洞检测模型仅限于确定有关代码是否存在漏洞,而没有给出与漏洞有关的详细信息。针对此问题,Li等[48]提出了IVDetect,该模型包括基于图的漏洞检测模块(采用特征注意图卷积网络——FAGCN)和基于图的解释模块(采用GNNExplainer)2部分,除了可以检测代码是否存在漏洞,也可以给出程序依赖图的子图以及与漏洞有关的关键性描述信息。结果表明,有67%的测试结果在top-5中。

进一步地,为了使图神经网络学习更全面、更深层的代码特征,科研人员尝试将多种特征(如抽象语法树、控制流程图等图特征及自然代码序列特征等)进行融合,以捕捉更广泛的漏洞类型。Zhou等[32]提出了Devign,主要包括以下3个顺序组件:复合代码语义图嵌入层,将函数源代码编码为具有综合语义信息的联合图结构(组合AST、CFG、DFG、NCS);门控图循环层,通过汇总和传递图中相邻节点的信息来学习节点特征;Conv模块,可以更有效地提取与当前图级分类任务相关的有意义的节点和特征。Cao等[35]提出了基于双向图神经网络的漏洞检测方法——BGNN4VD。利用代码复合图(组合AST、CFG、DFG)捕获源代码的语法和语义信息,将图神经网络引入后向边,构建了双向图神经网络,通过前向边和后向边将信息传递给邻居以捕获更多信息。最后使用卷积层提取输出、使用分类器检测漏洞,该方法比最新方法具有更高的精度和准确性。Ye等[9]提出了Poem,该模型利用多个图(AST、CDFG)来表示程序的句法和语义结构,并提出了一个新颖的图神经网络POEM-GNN,可以同时对图的多种边类型进行建模,以捕捉更多的程序内关系。在此基础上,Wang等[56]提出了Funded,进一步扩展了使用多关系图(AST、PCDG)进行程序结构建模的工作。Funded采用了新的GGNN模型,使用门控循环单元(GRU)和高速门(highway gated)对长期依赖性进行建模,获得了更好的性能。

最后,部分研究工作同时对多特征融合方法和注意力机制进行了改进,取得了比基准更好的结果。Li等[46]提出了ACGVD,是一种通过双层注意力机制构建图神经网络的漏洞检测方法。该模型将CFG和DFG进行整合以形成更全面的代码图,使用节点级和路径级的注意力机制更新图节点信息,以更多地关注与漏洞相关的节点,并采用卷积层提取与漏洞更相关的高级特征,最后利用全连接层进行二分类。实验结果表明,与最先进的方法相比,ACGVD获得了更高准确率、召回率和F1-score。Zhuang等[39]提出了3GNN,分别在AST、CFG和DFG上执行,每个图都可以提取各自的结构信息、省略无关信息,从而检测特定的漏洞,同时采用卷积层、自注意力池化层的层次结构,降低网络的复杂度。结果表明,该模型在2个现实数据集上优于基于BiLSTM、CNN和GGNN的方法。Zheng等[37]提出了VulSPG,组合DDG、CDG和FCDG构建了切片属性图,可以同时保留与漏洞相关的语义和结构信息,然后使用具有三重(token-level、node-level、subgraph-level)注意力机制的关系图卷积网络识别代码中的潜在漏洞。在切片级别和函数级别上进行了对比实验,结果表明,VulSPG模型可以准确地表征与漏洞相关的特征,实现漏洞检测任务。最近的工作,Song等[53]提出了一种基于源代码异构中间表示的漏洞检测方法——HGVul,分别将抽象语法树和代码属性图与自然代码序列相结合,根据各种语义关系将中间表示图划分为不同的子图,用于获取不同类型边所传达的语义信息;然后在每个子图上使用具有注意力机制的图神经网络学习代码表示,最后利用学习到的代码特征表示进行漏洞检测。在多个数据集上的实验表明,HGVul的F1-score在平衡数据集上达到96.1%,在不平衡数据集上达到88.3%。

2.3 基于代码相似性的漏洞检测模型

二进制代码相似性分析技术旨在检测2个二

进制函数的编译形式是否相似,该技术是网络安全领域的基本技术,可以进一步应用到漏洞检测、恶意软件分析、版权纠纷等下游任务[31,45]。对于漏洞检测任务,可以采用图嵌入或图神经网络技术学习代码的分布式表示,再比较已知漏洞和未知代码的向量相似度,从而判断未知代码是否存在漏洞。表7对基于代码相似性的漏洞检测模型进行了汇总。

表7　基于代码相似性的漏洞检测模型

模型	年份	数据集	中间图表示	是否跨架构	模型构造
Genius[54]	2016	OpenSSL	ACFG	是	—
Gemini[31]	2017	[54]	ACFG	是	Structure2Vec
Bin2Vec[73]	2021	Juliet C/C++ test suite	CFG	否	GCN
[40]	2021	OpenSSL	ACFG	是	Graph Embedding Network
DVul-WLG[76]	2021	OpenSSL以及D-Link, TP-Link等固件	ACFG	是	Word2Vec, LSTM, Directed GCN
BugGraph[43]	2021	[31]	ACFG	是	Graph Triplet-Loss Network
Vestige[41]	2021	OpenSSL	AFCG	是	GAT
HBinSim[77]	2021	[89]等	ACFG	是	Word2Vec, Hierarchical Attention Graph Embedding Network

Feng等[54]首次使用图嵌入的概念提出了Genius,从控制流程图中学习高级特征表示,用于在物联网设备中搜索固件漏洞。但是,为了计算二进制函数的嵌入,Genius需要利用图匹配算法计算目标函数和二进制函数之间的相似性。在此基础上,Xu等[31]提出了Gemini,将二进制函数表示为属性控制流程图,利用Structure2Vec算法计算图嵌入,采用孪生网络架构(siamese architecture)计算相似度得分。结果表明,在相似性检测上,Gemini比Genius获得了更高的准确率,可以显著加快嵌入生成时间和训练时间,在漏洞检测上可以识别更多的漏洞。

之后,越来越多的研究工作开始使用图嵌入和图神经网络进行二进制代码相似性的漏洞检测。Arakelyan等[73]提出了Bin2Vec,该模型将二进制可执行文件转换为控制流程图,用以捕获程序的语义信息,利用图卷积网络学习局部控制流和数据流信息,获得图的分布式表示。在代码函数分类和漏洞检测2个语义不同的任务上,取得了比基准更好的结果。Sun等[40]提出了一个大规模固件漏洞检测框架,首先将二进制函数转为属性控制流程图(ACFG),然后通过训练图嵌入网络提取ACFG的特征,生成每个函数的嵌入向量,最后通过计算漏洞函数与目标函数向量间的相似性,以确定目标函数是否包含漏洞。

但是,在进行二进制固件代码相似性检测时,会面临以下2个难点:一是由于架构、编译器版本、优化级别等不同,会造成目标代码与源代码格式不同;二是存在不同程度的代码相似度,其中较少相似的代码对于漏洞检测任务来说更加重要[45]。Sun等[76]提出了DVul-WLG,将二进制函数转为ACFG,使用Word2Vec、LSTM和GCN提取ACFG的特征,通过比较漏洞函数与固件函数特征之间的相似性,确定固件是否包含漏洞。此外,该模型通过典型相关分析(canonical correlation analysis,CCA)将不同体系结构的指令嵌入到相同空间中,并以中间向量的形式进行表达,进而用于跨架构固件漏洞检测。Ji等[43]提出了BugGraph,尝试同时解决上述2个问题。首先确定目标二进制代码的编译出处,生成具有相同出处的二进制代码,这可以大大减少编译过程的负面影响。然后设计了一个新颖的图三重损失网络(graph triplet-loss network,GTN),将三重ACFG(锚、正、负3种函数)作为输入,其学习目标是确保锚函数和正函数之间的相似性高于锚函数

和负函数,通过这种方式产生代码相似度排名,从而辨别相似代码之间的差异。

最后,为了让漏洞检测模型学习到更全面的代码表征,科研人员尝试融合多个代码级别(如指令级别、函数级别、代码块级别等)的特征。Ji等[41]提出了Vestige,首先构建二进制代码的属性函数调用图(AFCG),涵盖了指令级别、函数级别、二进制级别3种类型的代码特征,然后在AFCG上应用图注意力网络生成代码表征。结果表明,相比Gemini,Vestige可以将漏洞检测的top-1命中率提高27%。Wang等[77]提出了HBinSim,通过构建句法属性控制流程图和语义属性控制流程图,获得每个代码块的句法和语义特征。然后使用层次注意力图嵌入网络学习每个图的嵌入表示,该网络模型具有层次结构,可在指令、基本代码块和函数级别应用注意力机制,从而在构建函数特征时,使其以不同方式关注更多的关键信息。最后,计算2个嵌入向量的余弦距离,确定2个二进制函数是否相似。结果表明,在公开漏洞数据集中,Hbinsim的召回率高于基线模型。

2.4 具体应用场景中的漏洞检测模型

在实际应用场景中,科研人员也尝试利用图深度学习技术进行漏洞检测,但是目前而言,该方面的研究仍在初期阶段。

美国国家标准技术研究所(National Institute of Standards and Technology,NIST)通过对漏洞评分来衡量漏洞的严重程度,但这是一个非常耗时的过程。为此,Chen等[75]提出了漏洞分析和评分引擎——VASE,根据Twitter数据的相似性构建漏洞图(节点是CVE,边通过对应Twitter数据的相似性得到),采取注意力机制提取Twitter数据的潜在特征,并利用图卷积网络捕捉漏洞节点间的相关性预测得分。经实际数据测试,VASE的平均绝对误差(MAE)为1.255,可以比NIST提前至少1周对漏洞评分。

为了防止科学网络基础设施社区(如GitHub)的滥用和攻击,Lazarine等[82]使用社交网络分析并构建了用户和存储库之间的关系图,利用无监督图嵌入算法生成图嵌入来捕获存储库-用户图的网络属性和节点特征,对具有相似属性和漏洞的存储库和用户进行聚类,以识别存在漏洞的存储库和用户群体。Ullman等[83]提出了一个新颖的无监督图嵌入框架,通过将镜像(由各种开源应用程序组成的定制化虚拟机镜像)与相似的应用程序和漏洞聚集,能够捕捉应用程序之间的关系以及在GitHub中已经确定的漏洞,可以帮助进一步评估和缓解漏洞。

针对Android操作系统的漏洞检测,Renjith等[52]收集了Android 11、Android 12版本的应用层和框架层中存在的Java漏洞代码,并将其转为中间图表示(AST、CFG、PDG、CPG),使用图神经网络(GCN和GAT)进行漏洞检测,F1-score达到了0.92。

针对智能合约的漏洞检测,Huang等[81]首先解决了字节码多样性的问题,在数据和指令级别进行规范化,统一了由不同编译器生成的字节码,并通过跟踪字节码执行时产生的数据流和控制流进行特定合约的切片,以尽可能减少噪声代码。然后,对字节码进行预处理并构建控制流程图,采用无监督图嵌入算法将其编码为定长向量。最后,通过对比该向量和已知漏洞的智能合约识别潜在漏洞。结果表明,该方法优于最先进的检测方法,可以有效、准确地确定大量漏洞合约和数十个蜜罐合约。

此外,无源代码(如商业软件)的静默(程序无崩溃、错误等现象)漏洞分析仍然是一个开放问题。Wang等[84]发现,静默漏洞程序执行时,数据会流向被无声缓冲区溢出破坏的变量,而且这些受破坏变量的内存空间与未受影响变量的内存空间存在本质上的差异。为了检测程序执行时的静默缓冲区溢出漏洞,笔者设计了一种新的图数据结构DFG+来表示程序执行时的数据流、变量的空间信息及其他隐式信息,利用双向传播关系图卷积网络(BRGCN)学习这些重要特征,并执行节点的分类任务。实验表明,该模型在测试数据集上精确率为94.39%、F1-score为94.18%,并成功从30个漏洞中找到了29个漏洞。

3　分析与讨论

3.1　总结与分析

本研究首先对数据集、图数据、图深度学习模型和训练评估进行了总结,分别见表1、表2、表3及表5;然后介绍了基于代码模式和基于代码相似度的漏洞检测模型的研究进展,并给出了详尽的阐述和对比,分别见表6和表7。通过梳理以上具有代表性的研究工作,做出以下总结与分析。

1)对于数据集,本文将其分为合成数据、半合成数据及真实数据3类。其中,合成数据和半合成数据无法完全体现真实世界漏洞的复杂性,在这些数据集上建立的漏洞检测模型并不能很好地应用到真实漏洞数据集上。部分研究工作选择从真实的项目(如GitHub项目,Linux Kernel、FFmpeg、QEMU等开源项目)中构建数据集,并进行了公开。Devign、Gemini等数据集在后续的研究工作中获得了广泛应用,这给漏洞检测领域作出了极大的贡献(表1)。

2)对于被分析对象,基于代码模式的漏洞检测模型以C/C++语言为主,也有部分以PHP、Java和Swift为分析对象,而基于代码相似性的漏洞检测模型主要是二进制代码,如表6和表7中分析对象字段。例如,顾绵雪等[6]、李韵等[17]对传统机器学习和深度学习的漏洞检测进行了调研,发现二进制代码的研究占比较小,这是由于二进制代码的可读性差,对研究人员选取漏洞代码特征造成了更大的挑战。但是,本研究的调查显示,利用图深度学习技术进行二进制代码漏洞检测的研究在数量上有了增加,且取得了不错的效果。这表明在图结构数据上构建图深度学习模型可以更好地学习二进制代码的表征,提高二进制代码漏洞检测能力。

3)对于代码表征生成,基于代码模式的漏洞检测模型尝试了多种中间图表示,例如抽象语法树、控制流程图、函数调用图等,也有不少研究工作将多个特征表示进行融合,构建了一系列新颖的图结构表示形式(表3)。与单特征表示相比,多个特征融合可以同时考虑结构化信息(控制流和数据流)和非结构化信息(源代码),综合代码的词法、语法、语义和结构信息,从而获得更强的检测能力,如表6中图表示字段。而基于代码相似性的漏洞检测模型大多使用单特征,如属性控制流程图、属性函数调用图等,在多特征融合方面尝试较少,如表7中图表示字段。

4)对于图深度学习模型构建,基于代码模式的漏洞检测模型专注于漏洞检测一项任务,端到端训练并输出漏洞预测结果,图卷积神经网络和图注意力网络应用较多,关系图卷积网络、图同构网络等应用较少(表3)。而基于代码相似性的漏洞检测模型分为2个步骤,先学习代码的分布式表示,再比较已知漏洞和未知代码的向量相似度,进而判断未知代码是否存在漏洞,主要利用图嵌入网络构建模型。同时,研究人员偏向于在模型中应用注意力机制,如多头注意力机制、自注意力机制等,并根据漏洞代码的特点对注意力机制进行了改进,如表6和表7中模型构造字段。

5)对于跨项目、跨语言漏洞检测,训练集和测试集数据分布的不同会直接影响模型性能,目前大多数的研究工作更关注项目内漏洞检测,如表6中跨项目字段,但也有部分工作在尝试跨项目、跨语言的漏洞检测。例如,Li等[48]、Hin等[49]、Ghaffarian等[50]等工作证明基于图神经网络的漏洞检测模型能有效进行跨项目漏洞分析,Wang等[56]采用迁移学习进行跨语言漏洞检测,使用少量目标语言数据集即可完成模型训练,取得了不错的效果。

3.2　未来的研究方向与挑战

基于中间图数据结构并利用图深度学习技术学习源代码表征,能够捕捉深层次的代码特征信息,在漏洞检测任务中的表现比基于传统深度神经网络(如CNN、RNN等)的方法更优异。通过对现有研究成果的归纳和梳理,发现基于图深度学习的漏洞检测虽然已经取得许多突破性进展,但是在漏洞数据集构建、代码表征生成、图神经网络设计、跨语言检测等方面仍然存在或多或少的问题,具体内容如下。

1)真实世界的漏洞数据更具复杂性。现有的大部分数据集来自SATE IV Juliet、SARD和NVD等合成或半合成数据集,无法完全体现真实世界漏

洞的复杂性。同时,用于模型训练的数据往往存在数据类型不平衡的问题,这进一步限制了模型的性能。

2)建立统一规范的公开数据集。通过对已有公开数据集的总结发现,目前的各项研究大多依赖于各自收集和建立的数据集,尽管存在部分公开数据集,但仍是研究人员自发建立并公开的。因此,以统一规范建立一个能代表现实世界漏洞,且可以作为基准的公开数据集是利用图深度学习模型进行漏洞检测的首要挑战。

3)扩大被分析对象的范围。目前,有部分研究利用Twitter、开源社区(如GitHub)等开放数据进行漏洞分析,而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,漏洞检测不能只局限在代码本身,也可以充分利用开源生态中丰富的与漏洞相关的信息。

4)探索新的代码表征方式。在利用图神经网络进行漏洞检测时,通常将程序源代码表示为AST、CFG、PDG等中间图,而为了使模型学习到更全面、更深层的代码特征,科研人员也尝试将多种特征(如序列、文本、中间图)进行融合,这样做可以获得比单特征更好的效果。因此,如何构建新特征、生成高质量的代码表示是提高模型学习能力的重要途径。

5)使用更有效的词嵌入技术。程序代码的表达式或语句一般为文本格式,因此需要进行编码或嵌入以将文本转为向量,作为神经网络的实际输入。词嵌入技术会影响模型性能,更有效的词嵌入可以为模型学习提供更丰富的语义信息,生成的词向量质量越高,模型的训练效果越好。

6)图神经网络的设计会直接影响模型的训练效果。GCN、GAT、GGNN、GEN是最常用的图神经网络模型架构,除此之外,研究人员也尝试将图神经网络模型与其他网络模型(如LSTM、GRU、self-attention)结合以捕捉更全面的代码特征。但是,已有的图神经网络模型并不能完全解决漏洞检测中的深层次代码表征、数据不平衡、细粒度可解释等问题,如何设计并优化神经网络架构是一个值得探讨的问题。

7)跨项目、跨语言漏洞检测仍需研究人员进一步探索。对于更细粒度的漏洞位置检测,在构建数据集时,需要专家手动标注漏洞的确切位置,这无疑耗费了大量的人力物力财力,使用迁移学习进行跨项目漏洞检测可以很好地应对这个问题。此外,当前大多数的研究对象集中在C/C++,对多种编程语言构成的数据集进行漏洞检测需要进一步探索。同时,编程语言种类繁多且更新迭代较快,不同语言具有不同的语法规则,跨语言的漏洞检测依旧是一个值得探索的研究领域。

4 结论

图深度学习技术已经成功应用到诸多现实场景中,在网络安全领域,研究人员尝试利用图嵌入或图神经网络建立基于图数据结构的漏洞检测模型。基于图深度学习的漏洞检测方法在检测粒度和检测能力上有了一定提升,提高了漏洞检测准确率和检测效率,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。

通过对基于图深度学习的漏洞检测方法的梳理,归纳了其工作流程、技术特征及相关的研究成果,重点阐述了基于代码模式和基于相似性的漏洞检测方法,并展望了该领域所面临的挑战和机遇。通过总结现有研究工作,认为在未来的研究中,以统一规范建立一个能体现真实世界漏洞且可以作为基准的公开数据集是首要挑战,融合多种代码表征、获得高质量的词向量是提高模型性能的重要研究方向,设计并优化神经网络架构进行跨项目、跨语言的细粒度漏洞检测是一个值得进一步探索的研究领域。

参考文献(References)

[1] Araba V-P M, Addai P, Isteefanos S, et al. Survey on types of cyber attacks on operating system vulnerabilities since 2018 onwards[C]//2022 IEEE World AI IoT Congress (AIIoT). Seattle, WA, USA: IEEE, 2022: 1-7.

[2] Eceiza M, Flores J L, Iturbe M. Fuzzing the internet of things: A review on the techniques and challenges for eff

icient vulnerability discovery in embedded systems[J]. IEEE Internet of Things Journal, 2021, 8(13): 10390-10411.

[3] Aydos M, Aldan C, Coşkun E, et al. Security testing of web applications: A systematic mapping of the literature [J]. Journal of King Saud University-Computer and Information Sciences, 2022, 34(9): 6775-6792.

[4] Hanif H, Md Nasir M H N, Ab Razak M F, et al. The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches[J]. Journal of Network and Computer Applications, 2021, 179: 103009.

[5] Shen Z, Chen S. A survey of automatic software vulnerability detection program repair and defect prediction techniques[J]. Security and Communication Networks, 2020, 2020: 1-16.

[6] 顾绵雪, 孙鸿宇, 韩丹, 等. 基于深度学习的软件安全漏洞挖掘[J]. 计算机研究与发展, 2021, 58(10): 2140-2162.

[7] National vulnerability of database. CVSS severity distribution over time[EB/OL]. [2022-07-23]. https://nvd.nist.gov/general/ visualizations/vulnerability-visualizations.

[8] Wikipedia. Log4Shell[EB/OL]. (2023-01-21) [2023-05-23]. https://en.wikipedia.org/wiki/Log4Shell.

[9] Ye G, Tang Z, Wang H, et al. Deep program structure modeling through multi-relational graph-based learning [C]//The 29th ACM/IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT). New York, USA: Association for Computing Machinery, 2020: 111-123.

[10] Tian X, Ku W S. Geometric graph representation learning on protein structure prediction[C]//Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'21). New York, USA: Association for Computing Machinery, 2021: 1873-1883.

[11] Ahmedt-Aristizabal D, Armin M A, Denman S, et al. Graph-based deep learning for medical diagnosis and analysis: Past present and future[J]. Sensors, 2021, 21 (14): 4758.

[12] Ganz T, Härterich M, Warnecke A, et al. Explaining graph neural networks for vulnerability discovery[C]//Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security. New York, USA: Association for Computing Machinery, 2021: 145-156.

[13] Li G. Source code vulnerability mining method based on graph neural network[J]. International Journal of Frontiers in Engineering Technology, 2022, 4(4): 21-32.

[14] Jie G, Xiao H K, Qiang L. Survey on software vulnerability analysis method based on machine learning[C]//2016 IEEE First International Conference on Data Science in Cyberspace(DSC). Changsha, China: IEEE, 2016: 642-647.

[15] Bahaa Farid A, Kamal A, Ghoneim A. A systematic literature review on software vulnerability detection using machine learning approaches[J]. Informatics Bulletin, Faculty of Computers and Artificial Intelligence, 2022, 4 (1): 1-9.

[16] Alaoui R L, Nfaoui E H. Deep learning for vulnerability and attack detection on web applications: A systematic literature review[J]. Future Internet, 2022, 14(4): 118.

[17] 李韵, 黄辰林, 王中锋, 等. 基于机器学习的软件漏洞挖掘方法综述[J]. 软件学报, 2020, 31(7): 2040-2061.

[18] Semasaba A O A, Zheng W, Wu X, et al. Literature survey of deep learning-based vulnerability analysis on source code[J]. IET Software, 2020, 14(6): 654-664.

[19] Sonnekalb T, Heinze T S, Mäder P. Deep security analysis of program code[J]. Empirical Software Engineering, 2022, 27(1): 2.

[20] Lin G, Wen S, Han Q L, et al. Software vulnerability detection using deep neural networks: A survey[J]. Proceedings of the IEEE, 2020, 108(10): 1825-1848.

[21] Okun V, Delaitre A M, Black P E. Report on the static analysis tool exposition (SATE) IV. Special Publication, National Institute of Standards and Technology, Gaithersburg, MD[EB/OL]. [2022-07-23]. https://doi.org/10.6028/NIST.SP.500-297.

[22] NIST software assurance reference dataset. National Institute of Standards and Technology[EB/OL]. [2022-07-23]. https://samate.nist.gov/SARD.

[23] Booth H, Rike D, Witte G A. The national vulnerability database (NVD): Overview. ITL Bulletin, National Institute of Standards and Technology, Gaithersburg, MD[EB/OL]. [2022-07-23]. https://tsapps. nist. gov/publication/get_pdf.cfm?pub_id=915172.

[24] CVE Details. The ultimate security vulnerability data source[EB/OL]. [2022-07-23]. https://www. cvedetails.com/.

[25] Li Z, Zou D, Xu S, et al. SySeVR: A framework for using deep learning to detect software vulnerabilities[J]. IEEE Transactions on Dependable and Secure Computing, 2022, 19(4): 2244-2258.

[26] Russell R, Kim L, Hamilton L, et al. Automated vulnerability detection in source code using deep representation

learning[C]//2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA). Orlando, FL, USA: IEEE, 2018: 757-762.

[27] Rabheru R, Hanif H, Maffeis S. DeepTective: Detection of PHP vulnerabilities using hybrid graph neural networks[C]//Proceedings of the 36th Annual ACM Symposium on Applied Computing. New York, USA: Association for Computing Machinery, 2021: 1687-1690.

[28] Cheng X, Wang H, Hua J, et al. DeepWukong: Statically detecting software vulnerabilities using deep graph neural network[J]. ACM Transactions on Software Engineering and Methodology, 2021, 30(3): 1-33.

[29] Cao S, Sun X, Bo L, et al. MVD: Memory-related vulnerability detection based on flow-sensitive graph neural networks[C]//Proceedings of the 44th International Conference on Software Engineering (ICSE'22). New York, USA: Association for Computing Machinery, 2022: 1456-1468.

[30] OpenSSL. Cryptography and SSL/TLS Toolkit[EB/OL]. [2022-07-23]. https://www.openssl.org/.

[31] Xu X, Liu C, Feng Q, et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. New York, USA: Association for Computing Machinery, 2017: 363-376.

[32] Zhou Y, Liu S, Siow J, et al. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks[J]. arXiv preprint, 2019, arXiv:1909.03496.

[33] Fan J, Li Y, Wang S, et al. A C/C++ code vulnerability dataset with code changes and CVE summaries[C]//Proceedings of the 17th International Conference on Mining Software Repositories. New York, USA: Association for Computing Machinery, 2020: 508-512.

[34] Chakraborty S, Krishna R, Ding Y, et al. Deep learning based vulnerability detection: Are we there yet[J]. IEEE Transactions on Software Engineering, 2021, 48(9): 3280-3296.

[35] Cao S, Sun X, Bo L, et al. BGNN4VD: Constructing bidirectional graph neural-network for vulnerability detection[J]. Information and Software Technology, 2021, 136 (C): 106576.

[36] Şahin S E, Özyedierler E M, Tosun A. Predicting vulnerability inducing function versions using node embeddings and graph neural networks[J]. Information and Software Technology, 2022, 145(C): 16.

[37] Zheng W, Jiang Y, Su X. VulSPG: Vulnerability detection based on slice property graph representation learning[J]. arXiv preprint, 2021, arXiv:2109.02527.

[38] Suneja S, Zheng Y, Zhuang Y, et al. Learning to map source code to software vulnerability using code-as-a-graph[J]. arXiv preprint, 2020, arXiv:2006.08614.

[39] Zhuang Y, Suneja S, Thost V, et al. Software vulnerability detection via deep learning over disaggregated code graph representation[J]. arXiv preprint, 2021, arXiv: 2109.03341.

[40] Sun H N, Xie J T, Lin B, et al. Large-scale firmware vulnerability analysis based on code similarity[C]//2021 IEEE International Conference on Power Intelligent Computing and Systems (ICPICS). Shenyang, China: IEEE, 2021: 184-189.

[41] Ji Y, Cui L, Huang H H. Vestige: Identifying binary code provenance for vulnerability detection[C]//Applied Cryptography and Network Security: 19th International Conference, ACNS 2021. Kamakura, Japan: Springer-Verlag, 2021: 287-310.

[42] Baldoni R, Luna G A D, Massarelli L, et al. Unsupervised features extraction for binary similarity using graph embedding neural networks[J]. arXiv preprint, 2018, arXiv:1810.09683.

[43] Ji Y D, Cui L, Huang H H. BugGraph: Differentiating source-binary code similarity with graph triplet-loss network[C]//Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS'21). New York, USA: Association for Computing Machinery, 2021: 702-715.

[44] Wang S, Jiang X, Yu X, et al. Cross-platform binary code homology analysis based on GRU graph embedding [J]. Security and Communication Networks, 2021, 2021: 1-8.

[45] Liu S. A unified framework to learn program semantics with graph neural networks[C]//Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. New York, USA: Association for Computing Machinery, 2020: 1364-1366.

[46] Li M, Li C, Li S, et al. ACGVD: Vulnerability detection based on comprehensive graph via graph neural network with attention[C]//Information and Communications Security: 23rd International Conference, ICICS 2021. Chongqing, China: Springer-Verlag, 2021: 243-259.

[47] Nguyen V A, Nguyen D Q, Nguyen V, et al. ReGVD: Re-

visiting graph neural networks for vulnerability detection [J]. arXiv preprint, 2022, arXiv:2110.07317.

[48] Li Y, Wang S, Nguyen T N. Vulnerability detection with fine-grained interpretations[C]//Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021). New York, USA: Association for Computing Machinery, 2021: 292-303.

[49] Hin D, Kan A, Chen H, et al. LineVD: Statement-level vulnerability detection using graph neural networks[C]// 2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR). Pittsburgh, PA, USA: Association for Computing Machinery, 2022: 596-607.

[50] Ghaffarian S M, Shahriari H R. Neural software vulnerability analysis using rich intermediate graph representations of programs[J]. Information Sciences, 2021, 553: 189-207.

[51] 段旭, 吴敬征, 罗天悦, 等. 基于代码属性图及注意力双向 LSTM 的漏洞挖掘方法[J]. 软件学报, 2020, 31(11): 3404-3420.

[52] Renjith G, Aji S. Vulnerability analysis and detection using graph neural networks for android operating system [C]//Information Systems Security: 17th International Conference, ICISS 2021. Patna, India: Springer-Verlag, 2021: 57-72.

[53] Song Z, Wang J, Liu S, et al. HGVul: A code vulnerability detection method based on heterogeneous source-level intermediate representation[J]. Security and Communication Networks, 2022: 1-13.

[54] Feng Q, Zhou R, Xu C, et al. Scalable graph-based bug search for firmware images[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16). New York, USA: Association for Computing Machinery, 2016: 480-491.

[55] Wu Y, Lu J, Zhang Y, et al. Vulnerability detection in C/C++ source code with graph representation learning[C]// 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC). New York, USA: Association for Computing Machinery, 2021: 1519-1524.

[56] 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, 2021, 16: 1943-1958.

[57] Wang Y, Hou Y, Che W, et al. From static to dynamic word representations: A survey[J]. International Journal of Machine Learning and Cybernetics, 2020, 11(7): 1611-1630.

[58] Le Q V, Mikolov T. Distributed representations of sentences and documents[J]. arXiv preprint, 2014, arXiv: 1405.4053.

[59] Wu Z, Pan S, Chen F, et al. A comprehensive survey on graph neural networks[J]. IEEE Transactions on Neural Networks and Learning Systems, 2021, 32(1): 4-24.

[60] Goyal P, Ferrara E. Graph embedding techniques applications and performance: A survey[J]. Knowledge-Based Systems, 2018, 151: 78-94.

[61] Cai H, Zheng V W, Chang K C C. A comprehensive survey of graph embedding: Problems techniques and applications[J]. IEEE Transactions on Knowledge and Data Engineering, 2018, 30(9): 1616-1637.

[62] Perozzi B, Al-Rfou R, Skiena S. DeepWalk: Online learning of social representations[C]//Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14). New York, USA: Association for Computing Machinery, 2014: 701-710.

[63] Grover A, Leskovec J. Node2vec: Scalable feature learning for networks[C]//Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16). New York, USA: Association for Computing Machinery, 2016: 855-864.

[64] Dong Y, Chawla N V, Swami A. Metapath2vec: Scalable representation learning for heterogeneous networks[C]// The 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'17). New York, USA: Association for Computing Machinery, 2017: 135-144.

[65] Hamilton W L. Graph representation learning[J]. Synthesis Lectures on Artificial Intelligence and Machine Learning, 2020, 14(3): 1-159.

[66] Patil S S. Automated vulnerability detection in java source code using J-CPG and graph neural network[D]. Netherlands: University of Twente, 2021: 21-28.

[67] Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks[J]. arXiv preprint, 2017, arXiv:1609.02907.

[68] Wu S, Sun F, Zhang W, et al. Graph neural networks in recommender systems: A survey[J]. ACM Computing Surveys, 2022, 55(5): 1-37.

[69] Hamilton W L, Ying R, Leskovec J. Inductive representation learning on large graphs[C]//Proceedings of the 31st International Conference on Neural Information Pro-

cessing Systems (NIPS′17). New York, USA: Curran Associates, 2017: 1025-1035.

[70] Veličković P, Cucurull G, Casanova A, et al. Graph attention networks[J]. arXiv preprint, 2018, arXiv: 1710.10903.

[71] Li Y, Tarlow D, Brockschmidt M, et al. Gated graph sequence neural networks[J]. arXiv preprint, 2017, arXiv: 1511.05493.

[72] Feng Q, Feng C, Hong W, et al. Graph neural network-based vulnerability predication[C]//2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). Adelaide, SA, Australia: IEEE, 2020: 800-801.

[73] Arakelyan S, Arasteh S, Hauser C, et al. Bin2vec: Learning representations of binary executable programs for security tasks[J]. Cybersecurity, 2021, 4(1): 1-14.

[74] Cheng X, Wang H, Hua J, et al. Static detection of control-flow-related vulnerabilities using graph embedding [C]//2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS). Guangzhou, China: IEEE, 2019: 41-50.

[75] Chen H, Liu J, Liu R, et al. VASE: A twitter-based vulnerability analysis and score engine[C]//2019 IEEE International Conference on Data Mining (ICDM). Beijing, China: IEEE, 2019: 976-981.

[76] Sun H, Tong Y, Zhao J, et al. DVul-WLG: Graph embedding network based on code similarity for cross-architecture firmware vulnerability detection[C]//Information Security 24th International Conference ISC 2021. Cham: Springer, 2021: 320-337.

[77] Wang Y, Jia P, Huang C, et al. Hierarchical attention graph embedding networks for binary code similarity against compilation diversity[J]. Security and Communication Networks, 2021: 1-19.

[78] Zhou L, Huang M, Li Y, et al. GraphEye: A novel solution for detecting vulnerable functions based on graph attention network[C]//2021 IEEE Sixth International Conference on Data Science in Cyberspace (DSC). Shenzhen, China: IEEE, 2021: 381-388.

[79] Zhang H J, Li Y J, Liu Y W, et al. Vulmg: A static detection solution for source code vulnerabilities based on code property graph and graph attention network[C]// 2021 18th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP). Chengdu, China: IEEE, 2021: 250-255.

[80] Wu T, Chen L, Du G, et al. Inductive vulnerability detection via gated graph neural network[C]//2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD). Hangzhou, China: IEEE, 2022: 519-524.

[81] Huang J, Han S, You W, et al. Hunting vulnerable smart contracts via graph embedding based bytecode matching[J]. IEEE Transactions on Information Forensics and Security, 2021, 16: 2144-2156.

[82] Lazarine B, Samtani S, Patton M, et al. Identifying vulnerable github repositories and users in scientific cyberinfrastructure: An unsupervised graph embedding approach[C]//2020 IEEE International Conference on Intelligence and Security Informatics (ISI). Arlington, VA, USA: IEEE, 2020: 1-6.

[83] Ullman S, Samtani S, Lazarine B, et al. Smart vulnerability assessment for scientific cyberinfrastructure: An unsupervised graph embedding approach[C]//2020 IEEE International Conference on Intelligence and Security Informatics (ISI). Arlington, VA, USA: IEEE, 2020: 1-6.

[84] Wang Z, Yu L, Wang S, et al. Spotting silent buffer overflows in execution trace through graph neural network assisted data flow analysis[J]. arXiv preprint, 2021, arXiv:2102.10452.

[85] Xia X, Wang Y, Yang Y. Source code vulnerability detection based on SAR-GIN[C]//2021 2nd International Conference on Electronics Communications and Information Technology (CECIT). Sanya, China: IEEE, 2021: 1144-1149.

[86] Davis J, Goadrich M. The relationship between precision-recall and ROC curves[C]//The 23rd International Conference on Machine Learning (ICML′06). New York, USA: Association for Computing Machinery, 2006: 233-240.

[87] Powers D. Evaluation: From precision recall and F-factor to ROC informedness, markedness & correlation[J]. Journal of Machine Learning Technologies, 2011, 2: 37-63.

[88] Zheng Y, Pujar S, Lewis B, et al. D2A: A dataset built for AI-based vulnerability detection methods using differential analysis[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering Software Engineering in Practice (ICSE-SEIP). Madrid, ES: IEEE, 2021, 111-120.

[89] David Y, Partush N, Yahav E. Statistical similarity of binaries[C]//The 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI′16). New York, USA: Association for Computing Machinery, 2016: 266-280.

Survey of vulnerability detection based on graph deep learning

DONG Jiping[1,2,3], GUO Qiquan[1,2], GAO Chundong[2], HAO Mengmeng[1,2,3], JIANG Dong[1,2,3*]

1. Institute of Geographic Sciences and Nature Resources Research, Chinese Academy of Sciences, Beijing 100101, China
2. Laboratory of Cyberspace Geography, Chinese Academy of Sciences and The Ministry of Public Security of the People's Republic of China, Beijing 100101, China
3. College of Resources and Environment, University of Chinese Academy of Sciences, Beijing 100190, China

Abstract The recent advances made by graph-based deep learning have demonstrated its great potential in processing non-Euclidean structured data, and a large number of research efforts have attempted to apply graph embeddings or graph neural networks to vulnerability detection. This survey systematically investigates the vulnerability detection based on graph deep learning. Firstly, we summarize the four main stages of the vulnerability detection process, including data set, graph data preparation, graph deep learning model construction, and result evaluation. Then, starting from the effectiveness of graph-based deep learning vulnerability detection, we respectively expound the research results based on code patterns, code similarity and specific application scenarios. Finally, by sorting out and summarizing the existing research works, we analyze the challenges and foresee the trends in this research field.

Keywords cybersecurity; vulnerability detection; graph-based deep learning; graph embedding; graph neural networks

（责任编辑　傅雪）