SCG-Detector:基于图注意力网络的智能合约漏洞检测方法

顾锡国[1],王志伟[2],陈 翔[3],何启帆[1],崔展齐[1*]
(1. 北京信息科技大学计算机学院,北京 100101;2. 国家网信办数据与技术保障中心,北京 100048;
3. 南通大学信息科学技术学院,江苏南通 226019)

摘 要: 随着智能合约被广泛使用,其处理的业务逻辑更加复杂,代码复杂度越来越高,引发了大量安全漏洞.为避免潜在安全漏洞造成的危害,研究人员提出了一系列智能合约漏洞检测方法.但现有方法对合约特征表征不完整,未将合约的语义及结构特征进行统一表征,难以准确、全面地检测和识别智能合约中的潜在漏洞和安全风险.为此,本文提出了基于图注意力网络的智能合约漏洞检测方法SCG-Detector(Smart Contract Graph Detector).首先,通过解析合约源代码构建抽象语法树(Abstract Syntax Tree,AST)以表征合约语法结构信息,并在AST上添加表示语义信息的数据依赖关系和控制依赖关系,以构建合约图(Smart Contract Graph,SCG)同时表征合约的语法结构及语义信息;然后,将SCG输入到图注意网络模型中进行训练,利用注意力机制学习合约中漏洞的特征;最后,利用训练好的图注意力网络模型检测合约中是否存在漏洞及所存在漏洞的类型.SCG-Detector在12 616个智能合约上进行的实验结果表明,相比于sFuzz、Conkas、ConFuzzius、Mythril、Osiris、Slither、Oyente、MANDO-GURU等8种广泛使用的方法,SCG-Detector的Precision最高提升了26.46%,Recall最高提升了69.64%,F_1最高提升了59.57%.

关键词: 智能合约;图注意力网络;合约图;抽象语法树;数据依赖关系;控制依赖关系

基金项目: 江苏省前沿引领技术基础研究专项(No.BK20202001);国家自然科学基金(No.61702041);北京控制工程研究所高可信嵌入式软件工程技术实验室开放基金(No.LHCESET202307);北京信息科技大学"勤信人才"培育计划(No.QXTCP B202406)

中图分类号: TP311.5 **文献标识码:** A **文章编号:** 0372-2112(2024)12-4101-12

电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20231187

SCG-Detector: A Smart Contract Vulnerability Detection Method Based on Graph Attention Networks

GU Xi-guo[1], WANG Zhi-wei[2], CHEN Xiang[3], HE Qi-fan[1], CUI Zhan-qi[1*]
(1. *School of Computer Science*, *Beijing Information Science and Technology University*, *Beijing* 100101, *China*;
2. *Data and Technical Support Center*, *Cyberspace Administration of China*, *Beijing* 100048, *China*;
3. *School of Information Science and Technology*, *Nantong University*, *Nantong*, *Jiangsu* 226019, *China*)

Abstract: With the widespread of smart contracts, the business logic has become more complex, causing a large number of security vulnerabilities. In order to avoid huge losses caused by potential vulnerabilities, a series of smart contract vulnerability detection methods were proposed. However, existing methods cannot comprehensively represent semantic and structural features of the contract, making it difficult to accurately detect potential vulnerabilities and security risks in smart contracts. To address this issue, this paper proposes a smart contract vulnerability detection method based on graph attention networks, named SCG-Detector (Smart Contract Graph Detector). Firstly, an abstract syntax tree (AST) is constructed by parsing the contract source code to represent the contract's syntactic structure information. Data dependency relationships and control dependency relationships, which represent semantic information, are added to the AST to construct a smart contract graph (SCG) that characterizes the contract's syntactic structure and semantic information. Secondly, using the SCG as input, the graph attention network model is trained with an attention mechanism to learn the features of vulnerabilities in the contract. Finally, the trained graph attention network model is used to detect whether there are vulnerabilities in the contract

收稿日期:2023-12-21;修回日期:2024-08-04;责任编辑:李勇锋
*通讯作者:崔展齐

and the types of vulnerabilities present. Experiments are conducted on 12 616 smart contracts to compare with 8 widely used methods, including sFuzz, Conkas, ConFuzzius, Mythril, Osiris, Slither, Oyente, and MANDO-GURU. The experimental results shows that the Precision of SCG-Detector is improved by up to 26.46%,recall is improved by up to 69.64%, and F_1 is improved by up to 59.57%.

Key words: smart contracts; graph attention network; contract graph; abstract syntax tree; data dependencies; control dependencies

Foundation Item(s): Jiangsu Province Frontier Leading Technology Basic Research Project (No.BK20202001); National Natural Science Foundation of China (No.61702041); Open Fund of High Trust Embedded Software Engineering Technology Laboratory of Beijing Institute of Control Engineering (No.LHCESET202307); Beijing Information Science and Technology University "Qin-Xin Talent" Cultivation Project (No.QXTCP B202406)

1 引言

区块链是一种由矿工按照规定的协议进行维护的分布式交易账本,具有去中心化和不可篡改等特征[1].智能合约是运行在区块链分布式节点上的一种基于状态、采用事件驱动的图灵完备程序,具有公开透明、可追溯和去信任等特点[2],近年来被广泛应用于金融服务、物联网、基础设施、医疗保健等领域.随着智能合约的广泛使用,其处理的业务逻辑越来越复杂,代码复杂度越来越高,也引入了大量潜在漏洞.这些潜在漏洞一旦被利用,将可能导致巨大经济损失.如2016年DAO合约的重入漏洞导致价值6 000万美元的以太币丢失[3],2018年BEC代币的整数溢出漏洞导致9亿多美元瞬间蒸发[4].因此,为避免由于智能合约中的潜在漏洞造成巨大损失,在智能合约部署前对其中可能存在的漏洞进行充分检测是十分必要的.

为了在合约部署前检测合约中是否存在漏洞,研究者提前将智能合约的漏洞规则整合到集合中,通过与漏洞规则集合进行匹配以检测智能合约中可能存在的漏洞.但随着智能合约功能越来越丰富,新的漏洞模式不断出现,预定义的漏洞规则需要不断更新,这需要消耗大量时间和人力开销.为了降低开销,研究者提取合约的抽象语法树(Abstract Syntax Tree,AST)来表征合约词法信息和语法结构信息,然后使用Transformer等机器学习技术来检测合约漏洞.如SRCL[2]通过中序遍历AST分别获取合约的值序列和类型序列,并从中分别学习局部和全局语法结构信息,以检测合约中存在的漏洞.然而,AST序列能够表征的智能合约信息不够完整,缺少合约的语义信息,这可能会因语义特征表征不完整而导致难以检测合约漏洞.因此,DR-GCN[5]方法根据合约的DFG和CFG构建合约语义图,并利用图卷积神经网络(Graph Convolutional Networks,GCN)检测合约漏洞.然而,这类方法大多只学习了智能合约的语法结构信息或语义信息,所使用的合约特征不完整,导致检测过程中的漏报率或误报率较高.若将AST以及数据依赖和控制依赖关系进行有效融合,以更完整地表征合约的语法结构信息和语义信息,将能进一步提高智能合约的漏洞检测性能.

为此,本文提出了基于图注意力网络的智能合约漏洞检测方法SCG-Detector(Smart Contract Graph Detector).首先,通过合约源代码构建AST,并在AST上添加数据依赖关系和控制依赖关系,以构建合约图(Smart Contract Graph,SCG);然后,使用SCG训练图注意力网络(Graph Attention Network,GAN)模型;最后,利用训练好的模型检测合约中是否存在漏洞及所存在漏洞的类型.将SCG-Detector在包含12 616个合约的数据集上进行实验,实验结果表明,相比于sFuzz[6]、Conkas[7]、ConFuzzius[8]、Mythril[9]、Osiris[10]、Slither[11]、Oyente[12]、MANDO-GURU[13]等8种广泛使用的方法,SCG-Detector的Precision最高提升了26.46%,Recall最高提升了69.64%,F_1值最高提升了59.57%.

本文的贡献主要包括:(1)提出了一种更全面的智能合约特征表征方法,通过分析智能合约源代码,将数据依赖和控制依赖关系融合到AST中构建SCG,以更全面地反映智能合约的特征,并为进一步分析智能合约提供丰富信息;(2)提出了一种基于图注意力网络的智能合约漏洞检测方法,利用图注意力网络学习SCG特征,从而有效地学习合约的语法结构信息和语义信息,以更准确地检测合约漏洞;(3)基于所提出的方法SCG-Detector实现了原型工具,在广泛使用的大规模真实合约数据集上进行实验,并与sFuzz、Conkas、ConFuzzius等8种方法进行比较,以验证SCG-Detector的有效性.

2 研究动机示例

由于智能合约一旦部署就无法修改的特性,在部署合约前对其进行全面检测至关重要.然而,现有的研究主要利用AST、DFG或CFG来表征合约特征,这些方法对合约特征的表征并不全面,可能导致漏洞检测的准确性不足.图1为智能合约PoCGame(https://github.com/smartbugs/smartbugs-wild/blob/master/contracts/0x7d09edb07d23acb532a82be3da5c17d9d85806b4.sol)中的donateToWhale函数和RandomNumberGenerator(https://swcregist

ry.io/docs/SWC-120/#random_number_generatorsol)合约中的random函数,以及两个函数对应的AST类型序列和值序列.

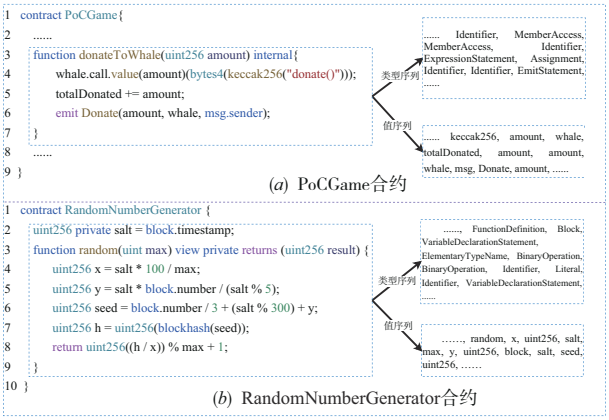


图1　动机示例

图1(*a*)中的PoCGame合约是以太坊上的博弈合约,该合约中的donateToWhale函数存在未检查外部调用漏洞.在执行过程中,第4行调用了donate()函数,但是由于未对该函数的执行结果进行检查,即使donate()函数执行失败,合约仍会继续执行.这可能出现合约状态不一致,增加交易结果的不确定性,最终可能会导致经济损失.若此漏洞被恶意利用,非法的外部调用将有可能破坏这个博弈合约的公平性.而图1(*b*)中的RandomNumberGenerator合约是以太坊上的随机数产生合约,该合约中的random函数存在时间戳依赖漏洞.第2行变量salt的值是通过获取所在块的时间戳赋值的,而矿工可以通过操纵时间戳来影响salt的值.因此,使用时间戳来产生随机数是不安全的,可能会导致经济损失.

对于基于专家规则的智能合约漏洞检测方法,以Oyente和Mythril为例,这两种方法通常只能有效地检测已知漏洞,对于没有定义相应专家规则的漏洞类型,会因缺乏专家规则的指导而无法提供准确的检测结果.在检测图1(*a*)的合约时,由于Oyente缺乏针对未检查外部调用漏洞的专家规则,未能识别出该漏洞.同样,Mythril在检测图1(*b*)的合约时,由于没有设定针对时间戳依赖漏洞的专家规则,也未能发现对应漏洞.

对于基于合约AST序列检测合约漏洞的方法,以SRCL为例,在检测图1中的合约时,会将AST转换为图1中右侧的类型序列和值序列.由于上述漏洞能否被成功检测分别和控制依赖关系和数据依赖关系相关,而类型序列和值序列中缺乏数据依赖关系和控制依赖关系,因此SRCL无法准确检测这两个合约中的漏洞,更不能识别出具体漏洞类型.

为进一步分析外部调用的安全性,可根据控制依赖关系判断外部调用是否已经过检查.具体来说,如果一个外部调用已经被检查过,在AST中将会有一条从Block节点到BinaryOperation节点的边来表示这种检查.反之,如果不存在这样的边,则表示这个外部调用没有得到检查.对于图1(*a*)所示的PoCGame合约,如果能有效利用合约的AST结构信息和控制依赖关系,将能检测出其存在的未检查外部调用漏洞.

类似地,为进一步分析时间戳的安全性,可对数据依赖关系进行分析.对于图1(*b*)中的RandomNumberGenerator合约,当使用所在块的时间戳来产生随机数时,变量salt与时间戳之间存在数据依赖关系,如果能有效利用合约的AST结构信息和数据依赖关系,将能检测出其中存在的时间戳依赖漏洞.

3　基于图注意力网络的智能合约漏洞检测方法

现有的智能合约漏洞检测方法未能完整地使用合约的语法结构信息和语义信息,导致漏洞检测的漏报率和误报率偏高.为此,本文提出了一种基于图注意力网络的智能合约漏洞检测方法SCG-Detector,其框架如图2所示.首先,将训练集进行预处理去除重复及不能编译的合约,为预处理后的合约源代码构建AST,并提取数据依赖关系和控制依赖关系以构建训练集的SCG.然后,使用训练集的SCG训练图注意力网络模型.最后,使用训练好的模型来检测合约中是否存在漏洞及具体的漏洞类型.通过将合约的语法结构信息和语义信息融合,SCG-Detector能够更全面地分析合约特征,并利用图注意力网络提高漏洞检测的准确性.

3.1　合约图构建

在基于机器学习技术的智能合约漏洞检测方法中,准确全面地表征合约特征信息对于提高性能至关重要.现有方法主要依赖于对智能合约的AST进行分析.然而,AST只能直接表征合约的语法结构信息,不能显式地表征合约的语义信息[14],导致一些较为复杂的漏洞(如未检查外部调用漏洞和时间戳依赖漏洞)不能被成功检测.

以图3中的智能合约为例,对于语句"if(a>10){d=a}",可通过中序遍历AST得到节点序列<IfStatement, BinaryOperation, Block, Identifier d, Identifier a>,该序列能够表征该语句的部分语法结构信息,但是该序列表征的语法结构关系并不完整,且无法表征出该语句的数据依赖关系和控制依赖关系等语义信息.

为解决该问题,本文提出了一种融合AST、数据依赖关系和控制依赖关系的SCG,通过更全面地表征智能合约的不同类型信息,提高图注意力网络模型对代码结构和行为的全面理解.结合文献[15]对语法结构

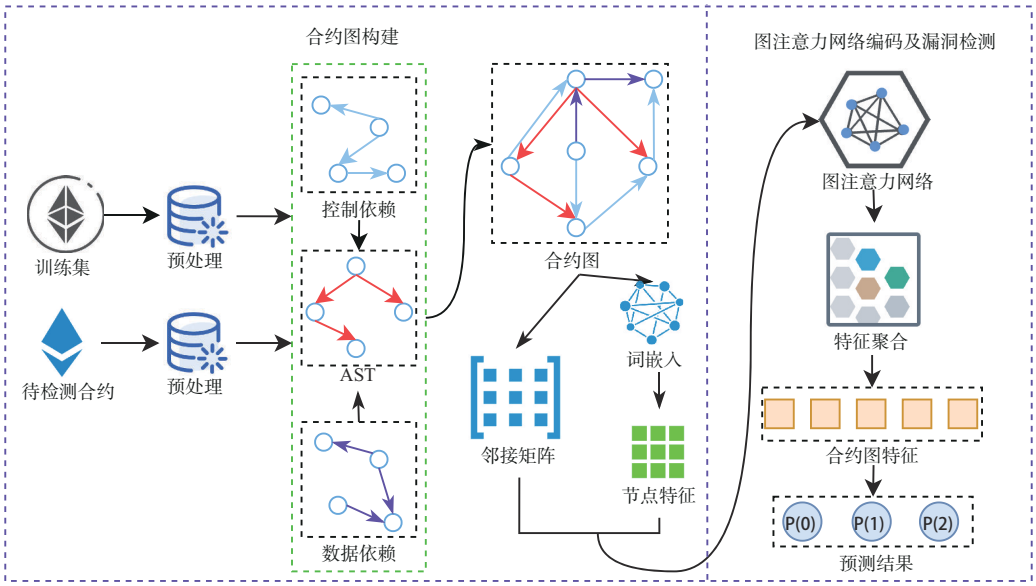


图2 SCG-Detector框架图

关系的定义，以及文献[16]对数据依赖和控制依赖关系的定义，对SCG定义如下.

定义1 SCG为包含智能合约语法结构信息和语义信息的四元组 $G=(V,E_{\mathrm{AST}},E_d,E_c)$.

V是合约源代码经过解析后得到的AST中的节点集合；E_{AST}是AST的边的集合，表示AST节点之间的语法结构关系. 语法结构关系是程序中两个基本语法标识节点之间的结构关系. 对于合约中的两个基本语法标识节点 V_i,V_j，若在AST中节点 V_i 是 V_j 的父节点，则认为 V_j 在语法结构上依赖于 V_i，记为 $V_i\rightarrow V_j$.

E_d 是数据依赖关系边的集合，表示节点之间的数据依赖关系. 数据依赖关系是程序中任意两个变量标识节点之间存在的数据定义和使用关系. 对于智能合约中的两个变量标识节点 V_i,V_j，若变量 V_j 使用了 V_i，则认为 V_j 数据依赖于 V_i，记为 $V_i\rightarrow V_j$.

E_c 是控制依赖关系边的集合，表示节点之间的控制依赖关系. 控制依赖关系是程序中表达式标识节点及代码块标识节点之间的执行关系，用于判断表达式标识节点及代码块标识节点在执行过程中是否相互影响. 对于智能合约中的表达式标识节点 V_i 和代码块标识节点 V_j，若 V_i 控制决定 V_j，则 V_j 控制依赖于 V_i，记为 $V_i\rightarrow V_j$.

对于 $e_{i,j}^{\mathrm{AST}}\in E_{\mathrm{AST}}$，$\mathrm{From}\left(e_{i,j}^{\mathrm{AST}}\right)=s_i\in V$，$\mathrm{To}\left(e_{i,j}^{\mathrm{AST}}\right)=s_j\in V$，表示 G 中存在 s_i 指向 s_j 的语法结构关系；对于 $e_{i,j}^{d}\in E_d$，$\mathrm{From}\left(e_{i,j}^{d}\right)=s_i\in V$，$\mathrm{To}\left(e_{i,j}^{d}\right)=s_j\in V$，表示 G 中 s_i 和 s_j 中存在数据依赖关系；对于 $e_{i,j}^{c}\in E_c$，$\mathrm{From}\left(e_{i,j}^{c}\right)=s_i\in V$，$\mathrm{To}\left(e_{i,j}^{c}\right)=s_j\in V$，表示 G 中 s_i 和 s_j 中存在控制依赖关系.

为在AST上添加数据依赖关系，可通过分析源代码中的数据依赖关系，并将其添加到 G 的数据依赖边集合 E_d 中. 接下来，以图3中的表达式 $a=b+c$ 和 $d=a$ 为例，阐述这一过程.

上述表达式包含4个变量，它们在AST中对应的节点分别是：Identifier a、Identifier b、Identifier c 和 Identifier d. AST可直接表征出节点之间的语法结构关系，但是不能显式地表征出合约语义信息. 通过识别变量的使用和赋值逻辑，可以建立这些节点之间的数据依赖关系. 具体来说，变量 a 的值是使用变量 b 和 c 的值定义的，因此变量 a 与变量 b 和 c 之间存在数据依赖关系，所以将(Identifier a, Identifier b)和(Identifier a, Identifier c)添加到了 G 的 E_d 集合中. 相应地，变量 d 是由变量 a 定义的，因此变量 d 的值依赖于变量 a 的值，所以将(Identifier d, Identifier a)添加到 G 的 E_d 集合中. 通过这种方式，可以捕捉到数据在程序中的传递和依赖情况.

为在AST上添加控制依赖关系，可通过分析AST获取基本控制结构节点，并将节点间的控制依赖关系

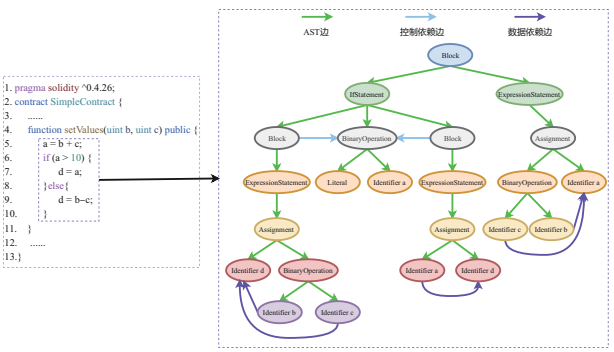


图3 合约图构建示例

添加到G的控制依赖边集合E_c中. 本文主要关注4种基本控制结构,分别是if结构、for结构、while结构和do while结构,对应的节点分别为IfStatement、ForStatement和WhileStatement(其中while和do while都对应WhileStatement). 以下将详细描述如何将每一种控制结构的控制依赖关系添加进G的E_c集合中.

对于IfStatement节点,其子节点包括BinaryOperation和Block. 其中,BinaryOperation代表if语句的条件,Block代表条件为真或假时执行的代码块. 在AST中,节点BinaryOperation和Block之间并无直接关系,但在判断结构中,Block节点的执行是依赖于BinaryOperation节点的. 为表示判断结构的控制依赖关系,将(BinaryOperation,Block)添加到G的E_c集合中.

对于ForStatement和WhileStatement节点,其子节点同样包括BinaryOperation和Block. 其中,BinaryOperation表示执行循环的条件,Block表示循环中的代码块. 与IfStatement节点类似,BinaryOperation和Block之间在AST中并无直接关系,但在循环结构中,Block节点的执行依赖于BinaryOperation节点. 为表示循环结构的控制依赖关系,同样将(BinaryOperation,Block)添加到G的E_c集合中.

通过在AST上添加这些控制依赖关系边,能够捕获程序中的执行顺序信息和控制依赖关系. 这样构建的SCG能够更好地反映程序的控制流程和条件依赖关系.

3.2　图注意力网络编码及漏洞检测

SCG融合了AST的语法结构关系、数据依赖关系和控制依赖关系,包含更为丰富的智能合约特征. 传统的GCN通过计算所有邻居节点特征信息的平均值来得到每一个节点的特征信息,所有邻居节点对目标节点具有相同的影响力,而GAT能够根据节点的重要性来学习节点特征信息. 因此,SCG-Detector采用GAT,以更有效地从SCG中学习合约的语法结构和语义信息.

GAT模型的关键优势在于其引入的注意力机制,该机制能够在学习SCG节点特征时为不同节点赋予不同的重要性,从而更好地捕捉节点间的特征关系. 这种机制使得GAT模型能够更有效地捕捉到SCG的特征. SCG-Detector中的GAT模型由两个图注意力卷积层(Graphic Attention Convolution)组成,每一层中均使用dropout正则化技术来防止过拟合. 其中,第一个图注意力卷积层接收输入特征,并通过注意力机制将其转换为隐藏层特征,在经过ReLU激活函数后进入第二个图注意力卷积层;第二个图注意力卷积层则通过注意力机制将隐藏层特征转化为输出特征. 经过两层图注意力卷积层后,SCG-Detector对输出特征使用Softmax函数进行标准化,得到每个节点的最终特征表示. 通过这种方式,SCG-Detector能够充分利用有限的训练数据集,获得更准确、更具代表性的特征表示,从而提高漏洞检测的准确性.

为了使用GAT检测智能合约中的漏洞,首先,将训练集$\mathrm{TD}=\{C_1, C_2, \cdots, C_l, \cdots\}$中的合约$C_l$转化为合约图$G_l$,获取对应的合约图集合$\mathrm{TG}=\{G_1, G_2, \cdots, G_l, \cdots\}$. 然后,将TG中的合约图$G_l$逐一转化为表示AST语法结构关系、控制依赖关系及数据依赖关系的邻接矩阵$\boldsymbol{G}_l^M$和表示节点特征的节点特征矩阵$\boldsymbol{G}_l^U$,并将$M_l=\{\boldsymbol{G}_l^M, \boldsymbol{G}_l^U\}$加入训练集特征集合$\mathrm{TM}=\{M_1, M_2, \cdots, M_l, \cdots\}$. 最后,将TM作为输入训练GAT模型.

在训练GAT过程中,GAT注意力层的计算过程如式(1)~(3)所示.

$$e_{ij}=a\left([W\cdot\boldsymbol{h}_i \| W\cdot\boldsymbol{h}_j]\right),\ j\in N_i \tag{1}$$

$$\alpha_{ij}=\mathrm{Softmax}_j\left(e_{ij}\right)=\frac{\exp\left(\mathrm{LeakyReLU}(e_{ij})\right)}{\sum\limits_{k\in N_i}\exp\left(\mathrm{LeakyReLU}(e_{ik})\right)} \tag{2}$$

$$\boldsymbol{h}_i'=\sigma\left(\sum_{j\in N_i}\alpha_{ij}\cdot\boldsymbol{h}_j\right) \tag{3}$$

式(1)用于计算节点i与邻居节点j的注意力系数. 其中,$\boldsymbol{W}$为TG中合约图G_l中节点集合V的权重矩阵,j为G_l对应邻接矩阵$\boldsymbol{G}_l^M$中节点i的邻居节点,$\boldsymbol{h}_i$、$\boldsymbol{h}_j$分别为G_l对应节点特征矩阵G_l^U中节点i、j的特征向量,$\boldsymbol{W}\cdot\boldsymbol{h}_i$和$\boldsymbol{W}\cdot\boldsymbol{h}_j$表示对$G_l$中的节点$i$和节点$j$分别进行线性变换,$\|$表示对线性变换后的结果进行垂直拼接,$a$表示把拼接后的特征映射为一个实数($a$可以选择多层感知器、单层前馈神经网络等). 式(2)通过Softmax对G_l中节点i与邻居节点j的注意力系数进行归一化处理,N_i表示G_l中节点i的邻居节点集合,e_{ik}表示G_l中节点i与邻居节点k的注意力系数,LeakyReLU为激活函数,α_{ij}表示归一化后的注意力系数. 式(3)为节点特征聚合(即加权求和),σ为sigmoid函数,$\boldsymbol{h}_i'$表示G_l中节点i经过加权求和后的特征向量.

在得到加权求和后的特征$\boldsymbol{h}_i'$后,使用式(4)聚合合约图的特征向量:

$$\boldsymbol{h}_G=\frac{1}{n}\sum_{i\in V}\boldsymbol{h}_i' \tag{4}$$

其中,n为G_l中的节点数量,V为G_l中所有节点的集合,$\boldsymbol{h}_i'$为G_l中的节点i经过加权求和后的特征向量,$\boldsymbol{h}_G$为合约图G_l的特征向量.

GAT模型在完成训练后,可用于检测待检测合约的漏洞. 首先,将待检测合约根据合约图构建过程转换为待检测合约图S. 然后,将S转化为表示节点间语法结构关系、控制依赖关系及数据依赖关系的邻接矩阵$\boldsymbol{S}^M$和表示每个节点特征的节点特征矩阵$\boldsymbol{S}^U$. 最后,将

$\boldsymbol{S}^M$和$\boldsymbol{S}^U$输入GAT,得到S的特征向量$\boldsymbol{h}_S$,输入到全连接网络,并通过分析特征向量$\boldsymbol{h}_S$来推断待检测合约中可能存在的漏洞,如溢出、重入和时间戳依赖等漏洞.

4 实验设计

为评估SCG-Detector的有效性,我们设计了以下三个研究问题:

RQ1:相比于其他智能合约漏洞检测方法,SCG-Detector的性能如何?

为验证SCG-Detector在检测智能合约漏洞的有效性,将SCG-Detector与sFuzz、Conkas、ConFuzzius等其他8种方法检测漏洞的情况进行比较.

RQ2:使用数据依赖关系和控制依赖关系对SCG-Detector性能的影响如何?

为验证数据依赖关系和控制依赖关系对SCG-Detector的影响,分别去除SCG中的数据依赖关系和控制依赖关系,并与SCG-Detector进行比较.

RQ3:相比于其他图神经网络模型,GAT对SCG-Detector的性能影响如何?

为验证GAT对SCG-Detector的有效性,将SCG-Detector中的GAT模型替换为GCN、GGNN等其他图神经网络模型进行实验,并与SCG-Detector进行比较.

4.1 基线

为评估SCG-Detector的有效性,选择Zhang等人[17]和Liu等人[18]及其他相关研究中广泛使用的智能合约漏洞检测工具sFuzz(https://github.com/duytai/sFuzz)、Conkas(https://github.com/smartbugs/conkas)、ConFuzzius(https://github.com/ConsenSys/mythril)、Mythril(https://github.com/ConsenSys/mythril)、Osiris(https://github.com/christoftorres/Osiris)、Slither(https://github.com/crytic/slither)、MANDO-GURU(https://github.com/MANDO-Project/ge-sc)和Oyente(https://github.com/smartbugs/oyente)作为基线方法进行比较.上述方法可分为基于模糊测试的检测方法、基于静态分析的检测方法和基于符号执行的检测方法三种类型.

基于模糊测试的智能合约漏洞检测方法包括sFuzz和ConFuzzius.其中,sFuzz是一种针对智能合约漏洞检测的灰盒模糊测试方法,该方法采用AFL(https://lcamtuf.coredump.cx/afl/)(American Fuzzy Lop)中的交叉、变异方法及多目标优化策略生成测试用例,通过执行智能合约并使用测试用例来探索合约分支,进而检测合约中的漏洞.ConFuzzius采用模糊测试和符号执行混合的方法,使用模糊测试来执行智能合约的浅层部分(Shallow Parts),并利用约束求解来生成满足复杂条件的输入,以探索深层部分(Deeper Parts).此外,ConFuzzius还利用动态数据依赖性分析来生成交易序列,这些序列更有可能触发潜在的合约漏洞.

基于静态分析的智能合约漏洞检测方法包括Mythril、Slither、Conkas和MANDO-GURU.其中,MANDO-GURU方法将合约表征为异构控制流图和异构函数调用图,在对异构控制流图和异构函数调用图进行合并后使用MANDO-HGT GNN(Graph Neural Network)模型学习异构图特征,以检测合约漏洞.

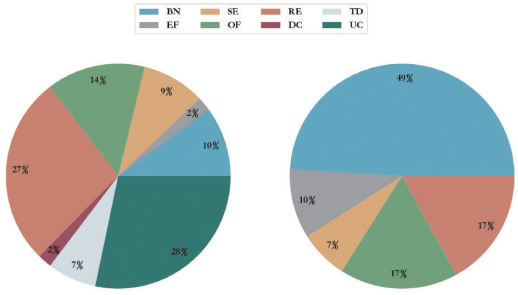
基于符号执行的智能合约漏洞检测方法包括Mythril、Slither、Osiris和Oyente.其中,Mythril利用符号执行和SMT(Satisfiability Modulo Theories)求解器对以太坊虚拟机(Ethereum Virtual Machine,EVM)的字节码进行分析,以模拟复杂程序路径并探索智能合约的行为.Slither则将Solidity智能合约转换为SlithIR形式的中间表示,然后应用数据流和污点跟踪等常用的程序分析技术来检测合约中的漏洞.Conkas是一种模块化静态分析方法,它使用中间表示(Intermediate Representation)来表征合约源代码,并使用符号执行来追踪可能导致漏洞的路径.Osiris将符号执行和污点分析进行结合,以寻找以太坊智能合约中的溢出漏洞,从而准确地定位并检测智能合约中的潜在安全问题.Oyente基于符号执行模拟智能合约中所有可能的执行路径,以找出漏洞.

通过与这些经过广泛验证的方法进行比较,可以评估SCG-Detector在实际应用中的性能.

4.2 数据集

本文选取了文献[18]和文献[19]使用的两个智能合约数据集作为实验对象,分别称为D_1(https://drive.google.com/file/d/1iU2J-BIstCa3ooVhXu-GljOBzWi9gVrG/view)和D_2(https://drive.google.com/file/d/1XFp3tZSMkWSkeLSHe_vrQjGZYZ3LzB2s/view),它们在智能合约漏洞检测相关研究中被广泛使用[20,21].D_1和D_2数据集分别包含12 000和616个智能合约样本,每个样本都明确标注了其所含漏洞的类型,包括8种类型:时间戳依赖(Timestamp Dependency,TD)、块号依赖(Block Number Dependency,BN)、危险的委托调用(Dangerous Delegate call,DC)、以太币冻结(Ether Frozen,EF)、未检查的外部调用(Unchecked external Call,UC)、可重入(Reentrancy,RE)、整数溢出(Integer Overflow,OF)、以太币数量严格相等条件错误(Dangerous Ether strict equality,SE).

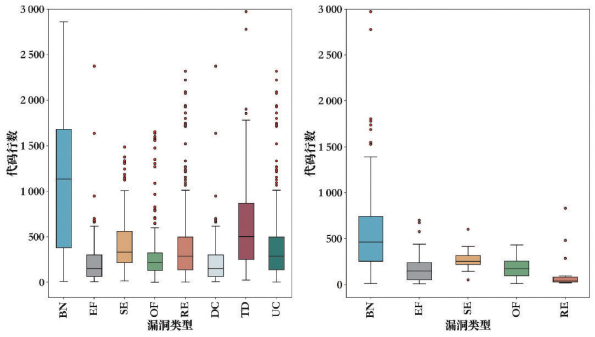
图4(a)和图4(b)中分别展示了D_1和D_2中各类漏洞的占比情况.从图4(a)可以看出,D_1中UC漏洞最多,占28%,DC和EF漏洞最少,仅各占2%;从图4(b)可以看出,D_2中BN漏洞最多,占49%,SE漏洞最少,仅占7%.因此,D_1和D_2数据集中的漏洞均存在较为严重的分布不均衡,这与实际应用场景相符.



(*a*) 数据集 D_1　　(*b*) 数据集 D_2

图 4　数据集中各类漏洞占比图

数据集 D_1 和 D_2 中智能合约的代码规模统计结果如图 5 所示. 从图中可以看出,数据集中智能合约的代码行数分布在 15~3 000 行之间. 其中,存在 BN 漏洞的合约在代码规模上的差异最大,最长为 2 900 行,最短仅有 20 行. 因此,D_1 和 D_2 中合约的代码规模差异明显,能够有效地反映实际场景中的代码规模差异.



(*a*) 数据集 D_1　　(*b*) 数据集 D_2

图 5　数据集中各类漏洞占比图

4.3　实现细节

实验采用深度学习框架 PyTorch Geometric (https://github.com/pyg-team/pytorch_geometric)和PyTorch(https://github.com/pytorch/pytorch)来实现图神经网络模型. 实验在一台配备有 64 GB 内存、AMD Ryzen 9 7950X 16-Core Processor 4.50 GHz 和 NVIDIA GeForce RTX 3090 Ti 的计算机上完成,运行的操作系统和开发环境分别为 Windows 11 专业版和 Python 3.9.12.

在 SCG-Detector 中,使用 Slither 对智能合约源代码进行解析以获取 AST,并通过 PyTorch 中的嵌入层 Embedding 将 SCG 的每个节点映射为一个 100 维的向量. 所训练的 GAT 模型由两个图注意力卷积层组成,每个图注意力卷积层的注意力头数量设置为 8,使用 Adam 优化器来优化所有参数,并选用 CrossEntropyLoss 作为损失函数,初始学习率为 0.000 1,隐藏层和输入层的维度分别设置为 100 维和 150 维. 在训练过程中,批处理大小设置为 32,训练轮数为 100 轮. 所有实验均使用十折交叉验证的方式进行.

5　实验结果及评价

在本节中,将按照所提出的 RQ 对实验结果进行分析,以说明 SCG-Detector 的有效性.

5.1　针对 RQ1 的分析

为评价 SCG-Detector 的整体性能,将 SCG-Detector 与其他智能合约漏洞检测方法进行比较. 表 1 展示了各种方法检测漏洞的性能比较. 由表 1 可以看出,SCG-Detector 的漏洞检测性能全面优于 sFuzz、ConFuzzius、Mythril、Slither 和 MANDO-GURU 5 种方法. 具体来说,相比于上述 5 种方法,SCG-Detector 的 Precision、Recall 和 F_1 分别提升了 10.98%~30.46%、28.72%~69.64% 和 24.29%~59.57%. 相比于 Conkas、Oyente 和 Osiris 方法,SCG-Detector 的 Recall 和 F_1 分别提升了 5.29%~31.92% 和 1.56%~18.17%,而 Precision 则下降了 2.47%~5.45%.

为验证 SCG-Detector 与其他方法在漏洞检测效果方面是否存在显著差异,使用 t 检验来比较各种漏洞检测方法的结果. t 检验的原假设为 SCG-Detector 与其他方法的漏洞检测结果没有显著差异. 从表 1 可以看出,所有基线方法在全部指标上的 p-value 均小于 0.05,因此拒绝原假设,即 SCG-Detector 与其他方法相比在漏洞检测效果上存在显著差异.

表 1　不同方法检测智能合约漏洞性能比较

方法	Precision (p-value)	Recall (p-value)	F_1 (p-value)
sFuzz	67.39 ($1.086\,9\times10^{-26}$)	26.36 ($3.103\,6\times10^{-17}$)	37.90 ($1.112\,3\times10^{-15}$)
Conkas	87.30 ($1.568\,8\times10^{-28}$)	80.13 ($4.410\,0\times10^{-24}$)	83.56 ($3.247\,6\times10^{-24}$)
ConFuzzius	54.37 ($9.848\,5\times10^{-22}$)	15.78 ($2.147\,2\times10^{-12}$)	25.55 ($9.816\,3\times10^{-15}$)
Mythril	73.85 ($7.630\,0\times10^{-26}$)	49.65 ($1.422\,9\times10^{-17}$)	59.38 ($2.574\,3\times10^{-21}$)
Osiris	89.44 ($2.547\,2\times10^{-24}$)	53.50 ($5.887\,8\times10^{-21}$)	66.95 ($1.193\,1\times10^{-20}$)
Slither	63.10 ($6.494\,0\times10^{-21}$)	44.23 ($1.146\,1\times10^{-18}$)	52.01 ($1.876\,0\times10^{-17}$)
Oyente	90.28 ($1.635\,9\times10^{-22}$)	66.40 ($1.132\,3\times10^{-22}$)	76.52 ($8.464\,8\times10^{-26}$)
MANDO-GURU	65.61 ($1.616\,4\times10^{-6}$)	56.70 ($3.626\,0\times10^{-11}$)	60.83 ($2.371\,9\times10^{-11}$)
SCG-Detector	84.83 (–)	85.42 (–)	85.12(–)

接下来，进一步分析SCG-Detector对8种不同类型漏洞的检测性能，并与Slither、Oyente等8种智能合约漏洞检测方法进行对比，如图6所示.

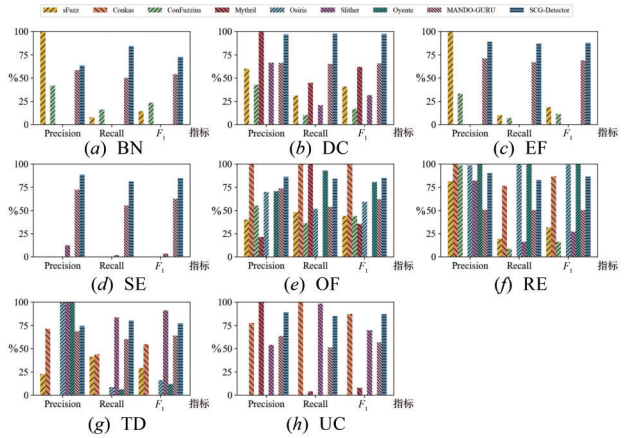


图6 不同方法检测各类型漏洞的性能比较

从图6可以看出，SCG-Detector检测8种漏洞的Precision、Recall和F_1指标均有较好性能. 其中，对于SE漏洞，SCG-Detector的Precision、Recall和F_1指标均优于其他所有方法；对于BN和EF漏洞，SCG-Detector的Precision、Recall和F_1指标均优于Conkas等7种方法，仅在Precision指标上低于sFuzz方法. 对于OF漏洞，SCG-Detector的Precision、Recall和F_1指标均优于sFuzz等5种方法，Precision和F_1指标低于Conkas方法，Recall指标低于Mythril等3种方法. 对于RE漏洞，SCG-Detector的Precision、Recall和F_1指标均优于sFuzz等4种方法，Recall和F_1指标低于Osiris和Oyente方法，Precision指标低于Conkas等4种方法. 对于DC漏洞，SCG-Detector的Precision、Recall和F_1指标均优于sFuzz等7种方法，仅Precision指标低于Mythril方法. 对于TD漏洞，SCG-Detector的Precision、Recall和F_1指标均优于sFuzz等5种方法，Precision指标低于Osiris等3种方法，Recall和F_1指标低于Slither方法. 对于UC漏洞，SCG-Detector的Precision、Recall和F_1指标均优于sFuzz等5种方法，Precision指标低于Mythril方法，Recall指标低于Conkas和Osiris方法，F_1指标低于Conkas方法. 此外，从表2中可以看出，sFuzz、Conkas、ConFuzzius、Mythril等7种方法仅能检测3~6种漏洞，尽管MANDO-GURU和SCG-Detector一样，均能够检测全部8种漏洞，但相比MANDO-GURU，SCG-Detector检测8种漏洞的平均Precision、Recall和F_1分别提升了19.22%、28.72%和24.29%.

针对RQ1的结论：SCG-Detector在所有评价指标上均优于sFuzz、ConFuzzius、Mythril、Slither方法，在Recall和F_1指标上优于Conkas、Oyente和Osiris方法，且SCG-Detector与其他方法相比在漏洞检测效果上存在显著差异. 此外，SCG-Detector相比于其他方法能检测更多类型的漏洞.

表2 各类方法能够检测的漏洞类型统计

漏洞类型	BN	DC	EF	SE	OF	RE	TD	UC
sFuzz	✓	✓	✓	×	✓	✓	✓	×
Conkas	×	×	×	×	✓	✓	✓	✓
ConFuzzius	✓	✓	✓	×	✓	✓	×	×
Mythril	×	✓	×	×	✓	×	×	✓
Osiris	×	×	×	×	✓	✓	✓	×
Slither	×	✓	×	✓	×	✓	✓	✓
Oyente	×	×	×	×	✓	✓	✓	×
MANDO-GURU	✓	✓	✓	✓	✓	✓	✓	✓
SCG-Detector	✓	✓	✓	✓	✓	✓	✓	✓

注：✓表示能够检测该漏洞，×表示不能检测该漏洞.

5.2 针对RQ2的分析

为评估所构建的SCG在漏洞检测方面的有效性，分别移除了数据依赖关系和控制依赖关系，并与SCG-Detector进行对比. 其中，未使用数据依赖关系、控制依赖关系和仅使用AST语法结构关系的SCG-Detector变体分别表示为SCG-Detector（w/o data）、SCG-Detector（w/o control）和SCG-Detector（w/o data & control）.

实验结果如表3所示，与SCG-Detector（w/o data）、SCG-Detector（w/o control）以及SCG-Detector（w/o data & control）变体相比，SCG-Detector在各项评价指标上均有显著提升. 其中，SCG-Detector在Precision指标提升了56.78%~72.13%，Recall指标提升了59.88%~71.80%，F_1指标提升了58.38%~71.98%. 此外，SCG-Detector（w/o control）与SCG-Detector（w/o data）相比，Precision、Recall和F_1分别提升了1.66%、1.04%和1.33%. 上述实验结果表明，在AST上添加数据依赖关系和控制依赖关系可以显著提升智能合约漏洞检测的性能，且控制依赖关系相较于数据依赖关系能够更好地表征合约特征.

为验证数据依赖关系和控制依赖关系对SCG-Detector的漏洞检测性能是否有显著影响，使用t检验来比较各种漏洞检测方法的结果. t检验的原假设为数据依赖关系和控制依赖关系对SCG-Detector的漏洞检测性能没有显著影响. 如表3所示，SCG-Detector与其他变体相比在全部指标上的p-value均小于0.05. 因此，拒绝原假设，即数据依赖关系和控制依赖关系对SCG-Detector的漏洞检测性能具有显著影响.

为进一步分析数据依赖关系和控制依赖关系对漏洞检测性能的影响，将SCG-Detector与其他变体在不同漏洞类型的合约上进行了实验，实验结果如图7所示. 从图7可以看出，SCG-Detector检测全部8种不同类型漏洞的Precision、Recall和F_1指标均优于SCG-Detector

表3　使用不同依赖关系的SCG-Detector漏洞检测性能比较

方法	Precision (*p*-value)	Recall (*p*-value)	F_1 (*p*-value)
SCG-Detector (w/o data)	26.39(1.399 6×10^{-14})	24.50(2.002 8×10^{-15})	25.41(1.805 3×10^{-15})
SCG-Detector (w/o control)	28.05(1.207 4×10^{-14})	25.54(3.004 3×10^{-15})	26.74(7.052 5×10^{-15})
SCG-Detector (w/o data & control)	12.70(3.166 5×10^{-18})	13.62(4.141 7×10^{-19})	13.14(1.971 1×10^{-18})
SCG-Detector	84.83(—)	85.42(—)	85.12(—)

(w/o data)、SCG-Detector (w/o control)以及SCG-Detector (w/o data & control). 其中,SCG-Detector与其他三种变体相比Precision指标最高分别提升了87.22%、82.22%和97.22%,Recall指标最高分别提升了87.78%、87.78%和97.78%,F_1指标最高分别提升了87.49%、85.49%和97.49%. 上述实验结果表明,引入数据依赖和控制依赖关系可以显著提升不同类型智能合约漏洞的检测性能.

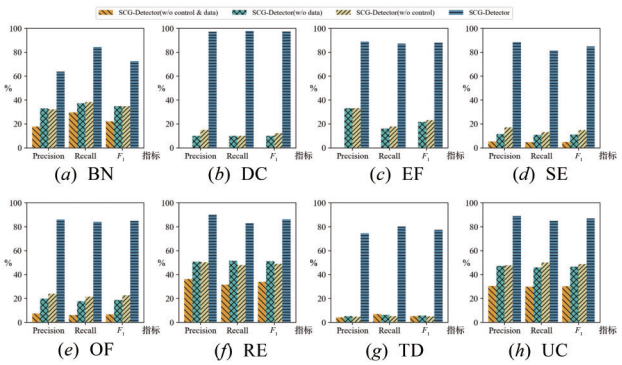


图7　使用不同依赖关系的SCG-Detector检测各类漏洞性能比较

此外,SCG-Detector (w/o data)和SCG-Detector (w/o control)在所有指标上均优于SCG-Detector (w/o data & control). 其中,两种变体的Precision指标相较SCG-Detector (w/o data & control)最高分别提升了33.17%和33.16%,Recall指标最高分别提升了20.00%和20.21%,F_1指标最高分别提升了21.76%和23.28%,进一步验证了数据依赖关系和控制依赖关系对智能合约漏洞检测性能的提升.

针对RQ2的结论:添加数据依赖关系和控制依赖关系后的SCG能够更全面地表征智能合约中各类漏洞的特征,从而显著提升SCG-Detector检测各类漏洞的性能,且SCG-Detector与其变体相比在漏洞检测效果上存在显著差异. 此外,控制依赖关系相较于数据依赖关系能够带来更大的性能提升.

5.3　针对RQ3的分析

为评估SCG-Detector所采用的GAT对漏洞检测性能的影响,实验分别使用GCN和GGNN模型替换GAT模型,将对应的变体表示为SCG-Detector (GCN)和SCG-Detector (GGNN),并与SCG-Detector进行比较,实验结果如表4所示. 从表中可以看出,SCG-Detector的漏洞检测性能优于SCG-Detector (GCN)和SCG-Detector (GGNN)两种变体. 具体来说,与SCG-Detector (GCN)和SCG-Detector (GGNN)相比,SCG-Detector的Precision指标提升了9.73%~33.36%,Recall指标提升了9.87%~51.2%,F_1指标提升了9.75%~44.01%. 实验结果表明,使用GAT能够有效提升SCG-Detector的智能合约漏洞检测性能.

为验证GAT对SCG-Detector的漏洞检测性能是否具有显著影响,我们使用*t*检验来比较各种图神经网络模型的检测结果. *t*检验的原假设为使用GAT或其他图神经网络模型对智能合约漏洞检测性能没有显著影响. 从表4可以看出,SCG-Detector与其他变体相比在全部指标上的*p*-value均小于0.05. 因此,拒绝该假设,即GAT与其他图神经网络模型相比在智能合约漏洞检测性能上存在显著差异.

表4　使用不同图神经网络的SCG-Detector漏洞检测性能比较

方法	Precision (*p*-value)	Recall (*p*-value)	F_1 (*p*-value)
SCG-Detector (GCN)	51.47 (3.793 7×10^{-8})	34.22 (1.060 0×10^{-13})	41.11 (1.001 3×10^{-12})
SCG-Detector (GGNN)	75.10 (2.861 8×10^{-2})	75.64 (3.268 8×10^{-3})	75.37 (1.335 1×10^{-3})
SCG-Detector	84.83 (-)	85.42 (—)	85.12 (—)

为进一步分析GAT模型对SCG-Detector检测不同类型漏洞的影响,将其与另外两种变体进行了实验,实验结果如图8所示. 从图中可以看出,对于Precision评价指标,SCG-Detector除在EF漏洞上略低于SCG-Detector (GGNN)外,在其他类型漏洞上均优于SCG-Detector (GCN)和SCG-Detector (GGNN). 对于Recall和F_1两项指标,SCG-Detector在所有漏洞上均优于SCG-Detector (GCN)和SCG-Detector (GGNN). 具体来说,SCG-Detector的Precision指标相较两种变体最高分别提升了47.31%和29.55%,Recall指标最高分别提升

了81.70%和15.85%，F_1指标最高分别提升了72.69%和22.94%. 总体来看，相比于其他图神经网络模型，使用GAT检测8种不同类型的漏洞均取得了最好性能.

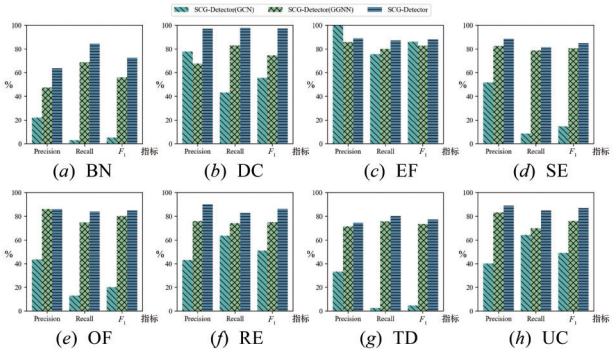


图8 使用不同图神经网络的SCG-Detector检测不同类型漏洞的性能比较

针对RQ3的结论：与其他图神经网络模型相比，GAT能够更有效地从SCG中学习合约语法结构信息和语义信息，从而提升SCG-Detector的整体性能以及检测不同类型智能合约漏洞的性能，且SCG-Detector与其变体相比在漏洞检测效果上存在显著差异.

6 讨论

本节将讨论SCG-Detector的局限性，并对有效性威胁进行分析.

6.1 局限性

实验结果表明，虽然SCG-Detector在各类智能合约漏洞的检测中表现出良好性能，但也存在部分误报和漏报，特别是BN和TD漏洞的漏报和误报情况相比其他漏洞更为严重. 为此，我们对存在BN和TD漏洞的合约进行了深入分析，对造成这一现象的主要原因总结如下.

BN和TD漏洞是由于在合约中使用块号和时间戳定义变量或将块号或时间戳作为条件判断变量而导致的. 从数据集中分别随机抽取了10个存在BN和TD漏洞的合约，发现其中使用块号和时间戳定义变量的合约分别有6个和5个，用于条件判断变量的合约分别有4个和5个. SCG-Detector通过由AST、数据依赖关系和控制依赖关系构成的SCG来表征合约，对于使用块号和时间戳定义变量的合约可以通过SCG中的数据依赖关系检测出其存在的BN和TD漏洞. 然而，对于使用块号和时间戳作为条件判断变量的合约，SCG目前还不能准确地表征语句中进行比较运算的变量之间的依赖关系，导致SCG-Detector在检测BN和TD漏洞时出现漏报和误报.

以图9中包含TD漏洞的合约为例，该合约中的TD漏洞由第7行中的block. timestamp造成. 由于SCG-Detector在分析数据依赖和控制依赖关系时，只关注了语句中进行算术运算的变量之间的依赖关系，未分析到block. timestamp> head这种语句中进行比较运算的变量之间的依赖关系，因此在检测该合约时无法准确检测出时间戳依赖漏洞.

```
1. pragma solidity ^0.4.24;
2. ......
3. contract FreezableToken is StandardToken {
4.      ......
5.      function releaseOnce() public {
6.          ......
7.          require(uint64(block.timestamp) > head);
8.          bytes32 currentKey = toKey(msg.sender, head);
9.          uint64 next = chains[currentKey];
10.         uint amount = freezings[currentKey];
11.         delete freezings[currentKey];
12.          ......
13.     }
14.     ......
15.}
```

图9 时间戳依赖漏洞示例合约

为解决此问题，在未来的工作中我们计划将语句中进行比较运算的变量之间的依赖关系添加进SCG中，以增强SCG的智能合约特征表征能力，从而进一步提高智能合约漏洞检测性能.

6.2 有效性分析

本节主要从内部有效性、外部有效性和构造有效性三个方面对SCG-Detector有效性的潜在威胁展开分析和讨论.

6.2.1 内部有效性

内部有效性威胁主要受到SCG-Detector和基线方法实现正确性的影响. 为缓解内部有效性威胁，在实现SCG-Detector时使用PyTorch、PyTorch Geometric等被广泛应用的第三方库来保证代码及模型的正确性. 另外，为确保Slither、Oyente等8种基线方法的正确性，使用作者提供的开源代码，并沿用论文中的参数设置，在相同的实验环境下进行实验，以确保达到相似性能.

6.2.2 外部有效性

外部有效性主要受到实验数据集的影响. 为缓解外部有效性威胁，使用文献[18,19]中的两个数据集来验证SCG-Detector的有效性. 这两个数据集的合约均来自以太坊，并被广泛应用于其他漏洞检测研究中. 数据集中包含众筹合约、博弈合约和投票合约等不同用途的智能合约，涵盖了较为广泛的应用领域，并包括重入、溢出和未检查外部调用漏洞等8种类型的漏洞，能

够有效评估SCG-Detector检测不同类型漏洞的能力.此外,数据集中的智能合约规模从15行代码到3 000行代码不等,能够有效反映实际场景中不同规模的智能合约,具有较为广泛的代表性和多样性.

6.2.3 构造有效性

构造有效性主要受到评价指标的影响.为缓解构造有效性威胁,在实验中选取了在漏洞检测任务中广泛应用的评价指标Precision、Recall和F_1,并使用开源库scikit-learn(https://github.com/scikit-learn/scikit-learn)计算这些指标以评估智能合约漏洞检测方法的性能.这些评价指标在许多关于智能合约质量保证的研究中被广泛应用[22],可有效评估合约漏洞检测方法的性能.

7 总结与展望

本文提出了一种基于图注意力网络的智能合约漏洞检测方法SCG-Detector.该方法在抽象语法树上添加数据依赖关系和控制依赖关系,以构建合约图,并用图注意力网络检测合约可能存在的漏洞,从而进一步提高漏洞检测的精度和有效性.在大规模数据集上的实验结果表明,SCG-Detector在漏洞检测性能总体上优于sFuzz、ConFuzzius、Mythril和Slither等8种方法.在未来的研究中,我们计划进一步扩展和深化对数据依赖的处理.目前的处理主要集中在算术运算关系上,但关系运算关系的处理同样重要,因此我们计划通过扩展处理数据依赖关系,进一步提升SCG-Detector的漏洞检测性能.这样将有助于更全面、更深入地理解数据依赖的复杂性,从而更有效地识别和检测潜在的安全漏洞,提升智能合约的安全性和可靠性.

参考文献

[1] 唐飞, 冯卓, 黄永洪. 基于区块链的公平可验证数据持有方案[J]. 电子学报, 2023, 51(2): 406-415.
TANG F, FENG Z, HUANG Y H. Fair provable data possession scheme based on blockchain[J]. Acta Electronica Sinica, 2023, 51(2): 406-415. (in Chinese)

[2] YANG S, GU X, SHEN B. Self-supervised learning of smart contract representations[C]//Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension. Piscataway: IEEE, 2022: 82-93.

[3] DEL CASTILLO M. The DAO attacked: Code issue leads to $60 million ether theft[EB/OL]. (2016) [2023]. https://www. baypayforum. com/blockchain-coins/the-dao-attacked-code-issue-leads-to-60-million-ether-theft.

[4] WANG W, SONG J, XU G, et al. Contractward: Automated vulnerability detection models for ethereum smart contracts[J]. IEEE Transactions on Network Science and Engineering, 2021, 8(2): 1133-1144.

[5] ZHUANG Y, LIU Z, QIAN P, et al. Smart contract vulnerability detection using graph neural network[C]//Proceedings of the 29th International Joint Conference on Artificial Intelligence. California: CAIO, 2020: 3283-3290.

[6] NGUYEN T D, PHAM L H, SUN J, et al. SFuzz: An efficient adaptive fuzzer for solidity smart contracts[C]//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. New York: ACM, 2020: 778-788.

[7] VELOSO N, TECNICO I S. Conkas: A modular and static analysis tool for Ethereum bytecode[EB/OL]. [2023]. https://fenix.tecnico.ulisboa.pt/downloadFile/1689244997262417/94080-Nuno-Veloso_resumo.pdf.

[8] TORRES C F, IANNILLO A K, GERVAISA, et al. Confuzzius: A data dependency-aware hybrid fuzzer for smart contracts[C]//2021 IEEE European Symposium on Security and Privacy. Piscataway: IEEE, 2021: 103-119.

[9] MUELLER B. Smashing ethereum smart contracts for fun and real profit[J]. HITB SECCONF Amsterdam, 2018, 9: 54.

[10] TORRES C F, SCHÜTTE J, STATE R. Osiris: Hunting for integer bugs in ethereum smart contracts[C]//Proceedings of the 34th Annual Computer Security Applications Conference. New York: ACM, 2018: 664-676.

[11] FEIST J, GRIECO G, GROCE A. Slither: A static analysis framework for smart contracts[C]//2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain. Piscataway: IEEE, 2019: 8-15.

[12] BADRUDDOJA S, DANTU R, HE Y Y, et al. Making smart contracts smarter[C]//2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). Piscataway: IEEE, 2021: 1-3.

[13] NGUYEN H H, NGUYEN N M, XIE C Y, et al. MANDO-HGT: Heterogeneous graph transformers for smart contract vulnerability detection[C]//2023 IEEE/ACM 20th International Conference on Mining Software Repositories. Piscataway: IEEE, 2023: 334-346.

[14] YAO W, SHAFIQ M, LIN X, et al. A software defect prediction method based on program semantic feature mining [J]. Electronics, 2023, 12(7): 1546.

[15] 凌春阳, 邹艳珍, 林泽琦, 等. 基于图嵌入的软件项目源代码检索方法[J]. 软件学报, 2019, 30(5): 1481-1497.
LING C Y, ZOU Y Z, LIN Z Q, et al. Approach to searching software source code with graph embedding[J]. Journal of Software, 2019, 30(5): 1481-1497. (in Chinese)

[16] MENG Y, XU D, ZHANG Z, et al. System dependency graph construction algorithm based on equivalent substi-

tution[C]//Proceedings of the Eighth International Conference on Internet Computing for Science and Engineering (ICICSE). Piscataway: IEEE, 2015: 106-110.

[17] ZHANG W, WEI L, CHEUNG S C, et al. Combatting front-running in smart contracts: Attack mining, benchmark construction and vulnerability detector evaluation[J]. IEEE Transactions on Software Engineering, 2023, 49(6): 3630-3646.

[18] LIU Z, QIAN P, YANG J, et al. Rethinking smart contract fuzzing: Fuzzing with invocation ordering and important branch revisiting[EB/OL]. [2023]. http://arxiv.org/abs/2301.03943.

[19] QIAN P, LIU Z, YIN Y, et al. Cross-modality mutual learning for enhancing smart contract vulnerability detection on bytecode[C]//Proceedings of the ACM Web Conference 2023. New York: ACM, 2023: 2220-2229.

[20] SO S, LEE M, PARK J, et al. VeriSmart: A highly precise safety verifier for ethereum smart contracts[C]//2020 IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2020: 1678-1694.

[21] LIU Z, QIAN P, WANG X, et al. Smart contract vulnerability detection: From pure neural network to interpretable graph feature and expert pattern fusion[EB/OL]. [2023]. http://arxiv.org/abs/2106.09282.

[22] PASQUA M, BENINI A, CONTRO F, et al. Enhancing ethereum smart-contracts static analysis by computing a precise control-flow graph of ethereum bytecode[J]. Journal of Systems and Software, 2023, 200: 111653.

作者简介

顾锡国 男，1998 年 7 月出生，青海西宁人．现为北京信息科技大学硕士研究生．主要研究方向为软件分析及测试技术．

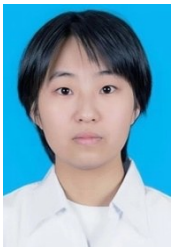
E-mail: xiguo_gu@bistu.edu.cn

王志伟 男，1982 年 12 月出生，北京人．现任国家网信办数据与技术保障中心处长．主要研究方向为互联网信息内容安全、网络安全、数据安全等．

E-mail: wangzhiwei@cac.gov.cn

陈 翔 男，1980 年 3 月出生，江苏南通人．现为南通大学副教授，硕士生导师，博士．主要研究方向为软件缺陷预测、软件缺陷定位、回归测试和组合测试．

E-mail: xchencs@ntu.edu.cn



何启帆 女，1999 年 10 月出生，河南周口人．现为北京信息科技大学硕士研究生．主要研究方向为可信人工智能．

E-mail: heqifan@bistu.edu.cn

崔展齐 男，1984 年 2 月出生，贵州金沙人．现为北京信息科技大学教授，硕士生导师，博士．主要研究方向为软件分析及软件测试技术．

E-mail: czq@bistu.edu.cn