计算机科学
COMPUTER SCIENCE

基于异构图注意力网络的智能合约漏洞检测方法

李诚宇, 黄可, 张锐恒, 陈伟

引用本文

李诚宇, 黄可, 张锐恒, 陈伟. 基于异构图注意力网络的智能合约漏洞检测方法[J]. 计算机科学, 2026, 53(2): 423-430.

LI Chengyu, HUANG Ke, ZHANG Ruiheng , CHEN Wei. Heterogeneous Graph Attention Network-based Approach for Smart Contract Vulnerability Detection [J]. Computer Science, 2026, 53(2): 423-430.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.241200144

基于异构图注意力网络的智能合约漏洞检测方法

李诚宇[1]　黄　可[2]　张锐恒[3]　陈　伟[4]
1 电子科技大学电子科技大学(深圳)高等研究院　广东 深圳 518110
2 电子科技大学计算机科学与工程学院　成都 611731
3 南京信息技术研究院　南京 210036
4 电子科技大学信息与软件工程学院　成都 610031
(202222280626@std.uestc.edu.cn)

摘　要　以太坊等区块链平台智能合约的安全漏洞一直是业界关注的焦点。使用字节码分析和检测智能合约漏洞是当前的主流方法之一,其中符号执行等传统方法需借助预定义的漏洞知识建立规则检测漏洞,存在效率低、精度差等问题。基于深度学习的检测方法则缺乏对字节码程序语义的深入理解,并且难以在过滤编译过程中产生噪声的同时捕捉完整的控制流与数据流信息。针对以上问题,提出了一种构建关键语义图检测智能合约漏洞的方法,首先定义了特定的去噪预处理规则实现对合约数据去噪,同时保留漏洞相关的关键语义信息,然后提出了一种能够捕捉程序丰富语义的异构图表示方法,并设计了一个基于异构图注意力网络(Heterogeneous Graph Attention Network,HAN)的漏洞检测模型。实验结果表明,所提方法优于现有的智能合约漏洞检测方法,对于拒绝服务、整数溢出、时间戳依赖和未检查函数返回值漏洞,其 F1 值分别提升了 17.75,5.94,28.94 和 27.85 个百分点。

关键词:智能合约;智能合约安全;图神经网络;智能合约字节码
中图分类号　TP311;TP309

Heterogeneous Graph Attention Network-based Approach for Smart Contract Vulnerability Detection

LI Chengyu[1],HUANG Ke[2],ZHANG Ruiheng[3] and CHEN Wei[4]
1 Shenzhen Institute for Advanced Study,UESTC,University of Electronic Science and Technology of China,Shenzhen,Guangdong 518110,China
2 School of Computer Science & Engineering,University of Electronic Science and Technology of China,Chengdu 611731,China
3 Nanjing Institute of Information Technology,Nanjing 210036,China
4 School of Information and Software Engineering,University of Electronic Science and Technology of China,Chengdu 610031,China

Abstract　Security vulnerabilities in smart contracts on blockchain platforms such as Ethereum have long been a focus of industry attention. Bytecode analysis and vulnerability detection have become one of the mainstream approaches for identifying smart contract vulnerabilities. However,traditional methods,such as symbolic execution,rely on predefined vulnerability rules,leading to inefficiencies and low precision. Deep learning-based methods,on the other hand,lack a comprehensive understanding of bytecode semantics and struggle to simultaneously filter noise generated during the compilation process while capturing complete control flow and data flow information. To address these challenges,this paper proposes a novel method for constructing critical semantic graphs to detect smart contract vulnerabilities. Firstly,a set of specific denoising preprocessing rules is defined to remove irrelevant data while preserving key semantic information related to vulnerabilities. Next,a heterogeneous graph representation method is introduced to capture rich program semantics. Finally,a vulnerability detection model based on the HAN is designed. Experimental results demonstrate that the proposed method outperforms existing approaches for smart contract vulnerability detection. For denial of service,integer overflow,timestamp dependency,and unchecked function return value vulnerabilities,the F1 scores of the proposed method are improved by 17.75,5.94,28.94,and 27.85 percentage points,respectively.

Keywords　Smart contract,Smart contract security,Graph neural network,Smart contract bytecode

到稿日期:2024-12-18　返修日期:2025-03-16
基金项目:国家自然科学基金(U2336204);四川省科技厅项目(2023YFG0112,2024YFHZ0015)
This work was supported by the National Natural Science Foundation of China(U2336204) and Science Foundation of Sichuan(2023YFG0112,2024YFHZ0015).
通信作者:陈伟(chenwei@uestc.edu.cn)

1 引言

以太坊(Ethereum)是一种基于区块链技术的开源平台，旨在实现去中心化应用(DApps)的开发和部署。以太坊建立了一个通用的区块链平台，以太坊智能合约是在以太坊区块链上执行的自动化合约或程序，由以太坊虚拟机(Ethereum Virtual Machine，EVM)运行。然而，随着智能合约的快速发展和智能合约带来的 Web3 技术在各个行业的落地，智能合约暴露出的安全问题也愈发严重。近年来，关于以太坊智能合约的安全事件频发，并且已经造成较为严重的经济损失。在所有安全事件中，损失最为严重的是 2016 年的 DAO 攻击事件[1]。在该事件中，攻击者利用智能合约的重入(Reentrancy)漏洞执行了一系列递归调用，导致 DAO 合约的资金被大量转移。攻击造成的损失估计达数百万以太币，在当时相当于数亿美元。直至今年，以太坊的攻击事件仍不断发生。据区块链安全公司慢雾科技发布的《2024 上半年区块链安全与反洗钱报告》[2]显示，2024 上半年 Web3 领域因 56 次合约漏洞事件造成约 1.04 亿美元损失，合约漏洞利用占安全事件手法榜首。

因此，多个用于检测以太坊智能合约漏洞的方法被提出。目前，绝大部分检测以太坊智能合约漏洞的方法以两种信息作为分析依据，即合约源代码和合约字节码。检测对象为合约源代码的研究可以充分利用源代码中的丰富自然语义和程序语义信息，因此相关的工作也较多，而检测合约字节码的工作则相对较少。然而，在以太坊平台，当合约在区块链部署运行后，以太坊链上信息只会保存合约的字节码，而源代码往往需要开发者主动开源，但目前在以太坊的大部分智能合约并未公开其源代码，因此检测合约字节码的工作是相当必要的。目前流行的字节码检测工具如 Mythril，Oyente 等使用符号执行方法进行漏洞检测，也有部分研究使用模糊测试方法进行检测，这些方法的优点是在执行较长时间时能达到较好的分支覆盖率，然而它们执行缓慢，针对单个合约检测就会消耗较长时间，对于当今普遍包含多个合约的去中心化应用效率较低。除此之外，也有许多研究使用人工智能方法开展漏洞检测工作，这些工作面对合约字节码语义的缺失，以及编译过程中产生的与源代码逻辑无关的操作码噪声等问题，未能提出较好的解决方案。

针对以上问题，本文进行了如下工作：

1)提出了一组规则，用于消除合约中与漏洞无关的代码。这部分代码通常在编译过程中生成，与合约的关键数据流和控制流没有关联，因此被定义为噪声。

2)通过使用开源的反编译工具，将合约字节码转换为中间表示(Intermediate Representation，IR)。在消除噪声后的中间表示基础上，提出了一种方法用于构建异构合约语义图，从而将待检测合约转换为异构图结构。

3)改进并使用异构图注意力网络模型，用于分类异构合约图。实验结果验证了该方法在常见智能合约漏洞检测中的优越性，相较于当前的主流检测方法，表现出了显著优势。

2 相关工作

由于智能合约漏洞检测在区块链安全中具有重要地位，当前有许多工作对此问题开展了研究。对于检测智能合约源码的工作，目前使用较为广泛的是 Slither[3]。Slither 是传统的代码静态分析框架，将智能合约转换为称为 SlithIR 的中间表示，并生成合约的控制流图。SlithIR 使用静态单分配形式和简化的指令集来简化智能合约的漏洞分析，并保留将智能合约代码转换为字节码时可能丢失的语义信息。Slither 接受了持续的维护和更新，对各个版本的智能合约以及不同规模的去中心化应用都具有良好的适配能力[4]。然而，Slither 对于漏洞的检测依赖既有的规则，规则定义的不完善会造成较高的误报率。近年来，使用深度学习方法的研究也纷纷出现。由于图这一结构能够自然地表示程序流程，因此图神经网络非常适合用于检测智能合约漏洞。在此领域，Zhuang 等提出了一种由智能合约源码构建合约图的方法[5]，将合约的代码语句转化为图的元素。之后，对图进行节点的删除与合并，以缩减图规模。最后，设计了一个时序传播网络(Temporal Message Passing，TMP)模型和一个图卷积神经网络模型，对合约图进行分类。在此方法上，Liu 等[6]从源代码中提取漏洞特定的专家模式特征，之后构建合约图，提取图特征后与漏洞特征相结合输出检测结果，提高了漏洞检测的精度。上述研究使用同构图表示合约结构，难以表示合约中的复杂语义信息(如循环结构、函数调用等)。Mando[7]首先由合约源代码构建 CFG 和 Call Graph，之后基于前者构建了合约异构图。此外，它开发了一个多元路径异构图注意力网络，可支持细粒度的行级漏洞检测工作。SCVHunter[8]首先基于中间表示构建异构语义图，之后基于异构图注意力网络进行漏洞检测。特别地，SCVHunter 允许用户自由指出图中更重要的节点，以更简单的方式利用专家知识，帮助自动捕捉更多与漏洞相关的信息。

在使用字节码进行漏洞检测的工作中，符号执行和模糊测试成为了主要的手段。Mythril[9]，DefectChecker[10]和 Oyente[11]使用符号执行方法，通过 SMT 求解器来确定某些条件是否可以被满足。Securify[12]通过给定一组既有的规则，然后检查依赖图来判断合约的行为是否正常。Manticore[13]设计了独特的动态执行引擎，使用符号执行的方式分析字节码中的漏洞。ConFuzzius[14]通过模糊测试检测智能合约的浅层代码，并通过符号执行和约束求解生成满足复杂条件的输入，以探索智能合约的深层代码。SMARTIAN[15]是一个实用的开源模糊测试工具。SMARTIAN 通过对智能合约字节码的静态分析来预测交易序列，并决定每个交易是否应满足特定的约束条件。在模糊测试阶段，该工具利用先前的信息构建初始种子库，并通过动态数据流进行分析，再基于数据流收集反馈，有效地引导模糊测试。与符号执行和模糊测试相比，深度学习方法具有高效、不受规则限制等优势。然而，由于字节码中几乎不包含自然语言的语义，因此使用源码开展分析的工作中所采用的 NLP 方法不适用于字节码漏洞检测。EtherGis[16]首先使用 EtherSolve[17]工具从合约字节码生成控制流图，然后使用专家模式处理控制流图中单个控制块的字节码片段，得到漏洞判断的权重值，之后使用图卷积网络对带权重节点的图进行分类判断。这种方法割裂了控制块间的语义信息。Huang 等[18]模拟字节码的执行流程，对

关键数据流进行切片，然后设计了一种无监督图嵌入算法，将代码图编码为可量化比较的向量。通过测量潜在漏洞合约与已知漏洞合约向量之间的相似性，可以识别出潜在的易受攻击的智能合约。Vulhunter[19]结合了机器学习与符号执行方法来得到合约的漏洞标签。以上研究未能在对合约去除无关字节码时保留合约的整体语义。

3 智能合约漏洞背景知识

根据智能合约漏洞登记表[20]，当前智能合约的漏洞已达37种，其中区块信息依赖漏洞、整数溢出漏洞、未检查的返回值漏洞和拒绝服务漏洞是较常见且具有中高危害性的漏洞，本文将这些漏洞作为研究对象。本章将介绍这些漏洞的含义与模式。

3.1 区块信息依赖漏洞

以太坊区块上会保存一些区块相关的信息，如当前的挖矿难度、生成区块的时间戳等。一些合约会借助这些信息来实现某些功能，如将这些信息作为生成随机数的种子，然而这样是不安全的。打包区块的节点可以自己修改这些信息。

图1给出了一个依赖区块时间戳信息而引起漏洞的示例，此示例为一个链上博彩游戏的合约代码，其在生成随机数时使用时间戳作为随机种子。由于打包区块的节点可以在一定范围内选取任意值作为时间戳，因此可以轻易构造使自己胜利的结果，从而不断窃取资金。

```
// 玩家发送以太币参与博彩
function play() public payable {
require(msg.value==0.1 ether,"Must send exactly 0.1 ether to play");
// 使用 block.timestamp 作为随机数种子
uint256 randomNumber=
  uint256(
  keccak256(abi.encodePacked(
    block.timestamp,msg.sender)
  ) % 2;
if(randomNumber==0){
// 玩家胜利，获得奖池中的所有资金
uint256 prize=address(this).balance;
(bool success,)=msg.sender.call{value:prize}("");
require(success,"Transfer failed");
//如果 randomNumber==1，玩家输了，资金留在合约中
```

图1 区块信息依赖漏洞示例

Fig. 1 Block information dependency example

3.2 未检查的返回值漏洞

未检查的返回值漏洞(Unchecked Low Level Calls)是智能合约开发中常见且严重的安全隐患之一，在基于以太坊的合约编写中更为突出。该漏洞主要出现在开发者使用低级调用(如call()，delegatecall()，send()等)时，没有对其返回值进行适当的检查。

以太坊智能合约中的低级调用是与外部合约或地址进行交互的重要机制。与transfer()或function call()等高级调用不同，低级调用不会自动回滚交易，即使发生错误，它们仍可能返回false而不是抛出异常。未检查这些返回值可能导致逻辑错误，恶意攻击者能够利用这一点阻止关键操作，甚至绕过合约的安全控制。例如，合约调用外部合约时，如果未能检测到call()返回false，外部合约的执行失败将被忽视，从而导致智能合约继续执行后续操作，可能使合约陷入意外的状态。

3.3 整数溢出漏洞

整数溢出发生在一个整数的结果超出其数据类型所能表示的范围时。例如，在Solidity中，uint256类型的整数范围是0到$2^{256}-1$。如果某个算术运算导致结果超出这个范围，则整数溢出就会发生，攻击者可以借此对合约展开攻击。

3.4 拒绝服务漏洞

智能合约的Dos漏洞主要有两种。第一种是基于Gas Limit的Dos漏洞。以太坊智能合约存在一种称为燃料(gas)的机制，任何消耗计算资源的操作都会产生一定的gas费用。当合约中一个函数处理的数据过多，可能会因为Gas消耗超过限制而引发拒绝服务攻击。例如，在一个函数中处理一个未知长度数组的所有元素，或者攻击者通过重入的方式使合约进入无限循环。第二种是基于失败的外部调用的漏洞。智能合约的状态取决于外部函数的执行结果，一旦外部调用失败或被拒绝，就会导致DoS攻击。当合约在使用一个循环来进行多个外部调用时，一个调用失败会导致接下来的调用无法执行。

4 合约漏洞检测方法与模型

本章将详细介绍提出的基于合约字节码构建合约语义图的方法，并利用人工智能技术进行漏洞检测。研究工作主要包含3个主要部分：字节码语义恢复后的噪声消除，异构合约语义图构建以及基于异构图注意力网络的漏洞检测。方法的整体架构图如图2所示。

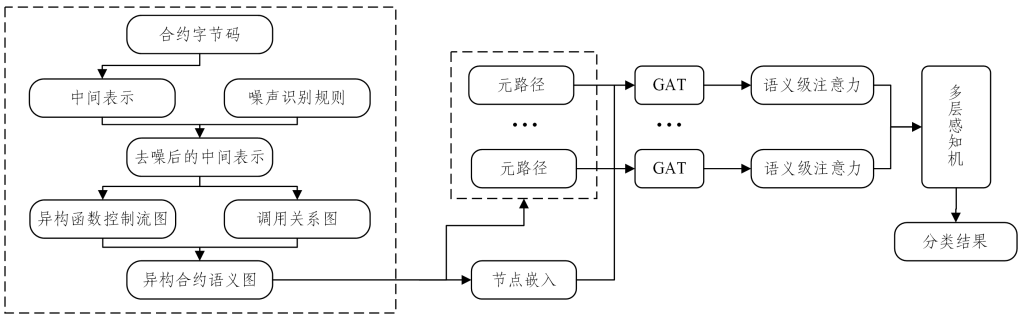

图2 所提方法的工作流程

Fig. 2 Workflow of the proposed approach

首先，使用反编译工具对以太坊智能合约的部署字节码(runtime-bytecode)进行处理，得到以IR形式表示的字节码中包含的语义信息，以及合约的控制流图。然后，通过一组定义的规则，从单个合约产生的IR中过滤掉与漏洞无关的IR，

提取得到代表关键数据流的代码片段。接着，将上述片段转换为异构的图结构。最后，利用图神经网络对异构图进行分类，作为漏洞检测的结果。

4.1 字节码语义恢复后的噪声消除

在进行漏洞检测之前，首先需要生成合约的语义图。理想情况下，语义图应尽可能包含与漏洞相关的代码片段，而将其他不相关的语义信息视为噪声。噪声应尽量被排除，以减少对检测过程的干扰。然而，在实际合约中，漏洞相关语句的比例通常较低。特别是在合约字节码中，导致这种现象的原因主要有两个：

1)存在漏洞的代码本身仅占合约的少部分。

2)在合约编译过程中，编译器会将一些与合约逻辑无关的指令添加到最终的字节码中。例如，为了使部署的合约中的公有函数能够被其他合约或用户调用，编译过程会插入函数选择器相关代码。函数选择器是函数签名的前 4 字节，在每次外部调用时会首先执行，用于判断接下来调用的函数。这部分代码显然与合约漏洞无关，因此被归类为字节码噪声。

除此之外，其他噪声来源还包括与 gas 消耗及回滚相关的判断代码，以及与循环或分支等控制结构相关的片段。这些代码与检测目标漏洞无直接关联，且控制流信息已通过反编译工具获得，因此同样被视为噪声。

为在保留漏洞相关语义的同时消除合约字节码中的噪声片段，本文定义了一组规则，用于明确需要保留的关键信息。利用这些规则并基于对现有研究和已知漏洞的分析，总结得到以下 3 个假设：

1)与函数参数相关联的代码与漏洞相关。部分安全漏洞与函数输入密切相关，当函数输入符合开发者预期时，函数可以正常运行，但当攻击者恶意构造特定的输入时，程序就可能出现非预期的表现，如整数溢出漏洞，该情况往往在开发者将变量与函数输入运算时未考虑数据范围时引起。

2)读写状态变量的代码与漏洞相关。以太坊智能合约中的变量有两种存放方式，只在合约调用过程中存在的临时变量存储在内存中，而长期存在的状态变量存放在存储(storage)中。状态变量往往与合约维护的重要信息关联，对其读写也会消耗更多的 gas，因此保留状态变量的读写对判断漏洞尤为重要。

3)读写区块信息的代码与漏洞相关。根据 3.1 节的描述，读写区块信息后得到的变量可能间接被对手操作，引起区块信息依赖漏洞。

基于上述假设，设计了一种用于过滤语句的方法。该方法借助 Gigahorse-toolchain[21] 对字节码进行处理，生成三地址码形式的 IR。随后，在 IR 的所有操作码中，保留满足上述 3 种假设的操作码语句，其余语句则被删除。

表 1 列出了具体被保留的操作码，其中假设 1 对应 CALLDATA 相关的操作码，假设 2 对应存储变量相关的操作码，假设 3 对应 BLOCKINFO 相关的操作码。除上述操作码外，ARITH 类别的操作码也会被保留，因为它们对变量进行代数运算或逻辑运算，生成新的变量，这意味着与漏洞相关的关键信息会沿着这些语句“流动”。另外，回滚、返回、跳转、停止等操作码也被保留，它们作为指示控制流的关键语句，表明了当前控制块在什么样的情景下结束。

表 1 GigahorseIR 的操作码描述与分类

Table 1 Description and classification of GigahorseIR opcode

操作码	标签	描述
BLOCKHASH, TIMESTAMP	BLOCKINFO	区块信息
CALLDATALOAD, CALLDATACOPY	CALLDATA	函数参数
CALL, CALLPRIVATE	CALL	函数调用
ADD, SUB, MUL…	ARITH	计算操作
SSTORE	SSTORE	存入存储变量
SLOAD	SLOAD	读取存储变量
JUMP, JUMPI	JUMP	跳转指令
REVERT	REVERT	回滚
RETURN	RETURN	返回
STOP	STOP	停止

在得到重点关注的操作码后，一组预定义的 datalog 规则将被用于识别 IR 中的噪声片段。规则包含了多个谓词与多个事实，谓词描述了数据间的一种关系，而事实是数据中满足谓词的一个元组。具体的识别规则如式(1)所示，其中部分事实由 Gigahorse 提供。

$$\frac{StatementOpcode(stmt, opcode),\ \neg KeyOpcode(opcode)}{NoiseStatement(stmt)} \tag{1}$$

式(1)给出了标记噪声语句的一条规则，其含义为：对于一个操作码为 opcode 的 IR 语句，如果 opcode 不在保留操作码集合内，那么就标记这条语句为噪声。在使用的两组事实中，KeyOpcode 由图 1 定义，而 StatementOpcode 的含义为 stmt 语句以 opcode 为操作码。

$$\frac{StatementOpcode(stmt, opcode),\ Unary(opcode)\ StatementUses(stmt, a, 0),\ NoiseVar(a)}{NoiseStatement(stmt)} \tag{2}$$

$$\frac{StatementOpcode(stmt, opcode),\ BinArith(opcode)\ StatementUses(stmt, a, 0),\ StatementUses(stmt, b, 1),\ NoiseVar(a), NoiseVar(b)}{NoiseStatement(stmt)} \tag{3}$$

$$\frac{StatementOpcode(stmt, opcode),\ TernArith(opcode)\ StatementUses(stmt, a, 0),\ StatementUses(stmt, b, 1),\ StatementUses(stmt, b, 2), NoiseVar(a),\ NoiseVar(b), NoiseVar(c),}{NoiseStatement(stmt)} \tag{4}$$

式(2)－式(4)给出了标记噪声语句的扩展规则。在先前的规则 1 中，不重要的 IR 语句被删除，而通过这 3 条规则，可以删除这些不重要的数据流分支上的所有语句。在这些规则内出现的事实中，UnaryArith, BinArith, TernArith 分别表示操作码是一元、二元、三元操作符，StatementUses 的参数分别代表判断的 IR 语句、变量和变量在操作码中的位置。当操作码使用的操作数均为已经被标记为噪声语句所定义的变量时(可以称为噪声变量)，将其标记为噪声。

$$\frac{NoiseStatement(stmt),\ StatementDefines(stmt,var,*)}{NoiseVar(var)} \tag{5}$$

$$\frac{StatementUses(stmt,var,*)\ NoiseVar(nv),\ StatementDefines(stmt,var,*)\ StatementOpcode(stmt,Opcode),\ FlowOp(Opcode)}{NoiseVar(var)} \tag{6}$$

式(5)、式(6)给出了标记噪声语句的两条规则。首先，被标记为噪声的语句定义的变量为噪声变量。相应地，这些噪声变量之间进行运算产生的新变量同样也为噪声变量。其中FlowOp谓词描述的是表1中Arith类型操作码中的会产生新变量的操作码。

为了直观展示本文的工作结果，表2列出了一段IR代码的示例，考虑到篇幅限制，这里给出的是人工编写的例子，而非真实合约字节码的反编译结果，因为它们往往数量庞大。在表中所示的语句中，根据规则1，编号为0，1，5，7，9的语句将被删除，因为它们的操作码不在保留的范围内。而根据规则2，编号为2的语句将被删除，它只使用了噪声语句定义的变量，因此也不被认为是程序语义的重点。

表2 IR代码示例

Table 2 IR code example

IR代码示例	
Begin block 0x0	
0x0	v0(0x80)=CONST
0x1	v1(0x4)=CONST
0x2	v2=LT v0,v1
0x3	v3=CALLDATALOAD v1
0x4	v4=ISZERO v3
0x5	v5(0x10)=CONST
0x6	JUMPI v5,v4
Begin block 0xc	
0x7	v7(0x0)=CONST
0x8	REVERT v7,v7
Begin block 0x10	
0x9	v9=CONST
	...

4.2 异构合约语义图构建

异构合约语义图构建流程为：首先使用Gigahorse来基于字节码反编译后得到的多组事实构建函数级别的控制流图，此时该图为同构图，节点和边没有类型或标签；然后，这些构造好的图将被转换为异构形式，从而借助节点和边之前的多样化关系表示丰富的程序语义。

4.2.1 异构函数控制流图构建

控制流图是一个过程或程序的抽象表现，代表了一个程序执行过程中会遍历到的所有路径。它用图的形式表示一个过程内所有基本块执行的可能流向，该形式也能反映一个过程的实时执行过程。控制流图中每一个节点代表一个基本块，也就是一段没有任何跳转的顺序代码。控制流图的每一条边代表控制流中的一次跳转。现有的使用人工智能方法对字节码开展的研究中也使用控制流图，但这些工作使用同构图，不能很好地表达程序的语义信息。

函数级控制流图包含了10种节点类型和2种边类型。在构建过程中，单个语句被作为一个节点，节点类型按IR对应的操作码(如表1中的标签描述)进行分类。边类型包含两种，sequential类型边用于连接顺序的代码语句，jump类型的边用于连接基本块中最后一条语句代表的节点和下一个基本块第一条语句代表的节点。构建异构函数控制流图的具体流程并不复杂，对于合约的每一个函数，从起始控制流块开始，遍历它的每一个后继块，对于块内的每一条语句，根据其操作码将其分类得到类型后加入异构图，并在它们之间连接sequential类型的边。在块与块之间，将相邻的两个节点以jump类型的边连接。

4.2.2 异构合约语义图构建

为了得到合约整体的语义图，需要根据合约的函数调用关系，将零散的函数控制流图进行合并。智能合约的函数调用分为两种，内部调用和外部调用。合约内部定义函数的调用为内部调用。跨合约调用可以分为两种情况：如果调用的另一个合约在dapp构建时，源码文件被一起编译，那么这种调用同样会转换为内部调用，这在开发者使用某些框架(如truffle[22])时较为常见。如果调用的另一个合约是部署在链上的合约，那么这个调用即为外部调用。外部调用需要发起交易，这涉及跨合约的通信，对此类跨合约调用的处理较为复杂[23]，本文仅考虑内部调用。因此，在异构函数控制流图构建过程中，捕捉所有callprivate语句，并记录起始控制块和目标控制块以标记函数调用关系。当所有异构函数控制流图构建结束后，根据先前记录的调用关系，将异构函数语义图用call类型的边连接，构建异构合约语义图。图3展示了一个异构合约语义图的示例，其中包含多种节点和边类型，通过图结构表达了程序的复杂语义信息。

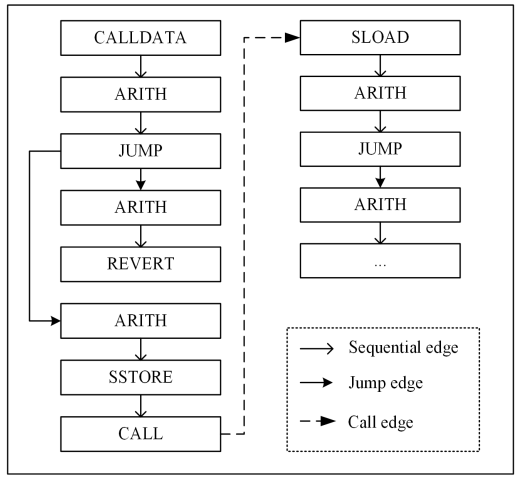

图3 异构合约语义图示例

Fig. 3 Heterogeneous contract semantic graph example

4.3 异构图注意力网络模型

本研究采用异构图注意力网络[24](HAN)处理由智能合约源码构建的异构合约语义图。异构图由不同类型的节点和边组成，用于捕捉智能合约中的复杂结构和语义信息。HAN的主要优势在于其利用注意力机制，自适应地聚合节点间的多种关系信息，非常适合处理此类异构数据。在此模型中，首先使用反射的方式提取异构图中的元路径。然后，如图4所示，应用节点级注意力和语义级注意力为节点生成统一的

嵌入表示。在得到节点嵌入后，计算元路径内的节点嵌入平均值，并将其作为元路径的嵌入表示，然后通过多层感知机对图进行分类，从而预测潜在的漏洞。

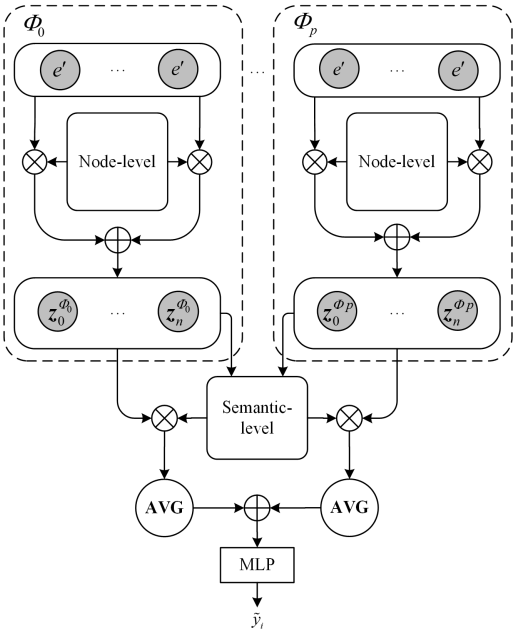

图 4 漏洞分类器的工作流程

Fig. 4 Workflow of the vulnerability classifier

4.3.1 元路径

元路径(Meta-path)是用于描述异构图中不同类型节点之间关系的一种路径模式。它定义了从一种类型的节点出发，通过一系列特定类型的边，最终回到同一类型或不同类型节点的一个路径序列。在异构图中，元路径提供了一种结构化的方式来表达这些多类型节点和边之间的关系。异构合约语义图中，有 10 种节点类型和 2 种边类型，如果考虑所有类型，则直接提取任意长度的元路径，则会得到海量数量的元路径，给后续工作带来挑战。因此，使用反射相邻节点的方式提取元路径，例如对于异构语义图相邻的两个节点 ARITH-JUMP，通过长度为 2 的元路径 ARITH-JUMP-ARITH 来反映其关系。

4.3.2 节点级图注意力网络

异构图的节点的初始嵌入表示将根据节点类型使用 Onehot 编码生成得到。没有使用基于深度学习的图嵌入生成方法的原因是，异构合约语义图的节点类型由 IR 操作码划分，这本身与程序语义密切相关。

表 3 符号说明

Table 3 Notations and explanations

符号	说明
Φ	元路径
$\boldsymbol{e}$	初始节点嵌入
$\boldsymbol{W}_{\varphi}$	类型 φ 的投影矩阵
$\boldsymbol{e}'$	投影后的节点嵌入
s_{ij}^{Φ}	基于元路径节点对(i,j)的重要程度
a_{ij}^{Φ}	基于元路径节点对(i,j)的注意力权重
$\boldsymbol{Z}_{\Phi}$	节点级注意力阶段的最终节点嵌入
α_{Φ}	元路径 Φ 的权重
$\boldsymbol{Z}$	最终节点嵌入

由于异构图中存在动态数量的节点类型，因此使用投影矩阵将节点嵌入投影到同一个特征空间中，如式(7)所示：

$$\boldsymbol{e}_i'=\boldsymbol{W}_{\varphi_i}\cdot\boldsymbol{e}_i \tag{7}$$

之后，给定元路径 Φ，使用 GAT 中的自注意力机制学习节点的重要性，如式(8)所示。随后，通过归一化得到元路径中各个节点对节点 i 的权重，如式(9)所示。

$$s_{ij}^{\Phi}=att_{\text{node}}(\boldsymbol{e}_i,\boldsymbol{e}_j;\Phi) \tag{8}$$

$$a_{ij}^{\Phi}=\text{softmax}_j(s_{ij}^{\Phi})=\frac{\exp(s_{ij}^{\Phi})}{\sum\limits_{k\in N_i^{\Phi}}\exp(s_{ik}^{\Phi})} \tag{9}$$

得到各个节点的注意力权重后，对元路径上的节点特征进行加权聚合，得到充分聚合节点语义的节点嵌入，其计算式如式(10)所示：

$$\boldsymbol{z}_i^{\Phi}=\sigma\Big(\sum_{j\in N_i^{\Phi}}a_{ij}^{\Phi}\cdot\boldsymbol{e}_j'\Big) \tag{10}$$

其中，σ 为激活函数，N_i^{Φ} 代表节点 i 在元路径 Φ 上的邻居节点集合。

4.3.3 语义级图注意力

在元路径内部使用自注意力机制更新得到的节点嵌入体现的语义与整体图中的节点语义之间依然存在差距，因此使用语义级注意力来促进不同元路径中的节点的特征融合。

与节点级注意力类似，先计算每个元路径的重要性，之后归一化得到元路径的语义级注意力权重值，如式(11)所示：

$$\alpha_{\Phi_p}=\text{softmax}\Big(\frac{1}{|V|}\sum_{i\in V}\boldsymbol{q}^{\mathrm{T}}\cdot\tanh(\boldsymbol{M}\cdot\boldsymbol{z}_i^{\Phi_p}+\boldsymbol{b})\Big) \tag{11}$$

其中，$\boldsymbol{b}$ 为偏置常量，$\boldsymbol{M}$ 为权重矩阵，$\boldsymbol{q}$ 为注意力向量，V 为异构图顶点集合。

得到元路径的权重后，融合上节得到的节点语义嵌入，获得最终的嵌入表示，如式(12)所示：

$$\boldsymbol{Z}=\sum_{p=1}^{P}\alpha_{\Phi_p}\cdot\boldsymbol{Z}_{\Phi_p} \tag{12}$$

图分类任务由带有 softmax 激活函数的多层感知器来完成。训练过程的损失函数为交叉熵，模型参数通过反向传播学习。在参数设置方面，设置了 8 个多头注意力机制，隐藏层大小为 32，训练轮数为 100，并使用 Adam[25] 与学习率调度器，学习率范围为 0.0005～0.01。

5 实验

5.1 数据集

当前关于智能合约漏洞检测的研究有很多，但并没有公认的基准数据集或大规模经过验证的可靠数据集，多数工作使用 smart-bugs[26] 作为数据集来源，但 smart-bugs 数据集缺乏人工验证的漏洞分类。对此，部分研究使用已有的漏洞检测工具来得到数据集标签，然而这影响了结果的客观性。本文利用文献[7]中的数据集，使用合约中声明的 solidity 版本对应的 solc 编译器，经过编译并提取 runtime-bytecode 作为本文的数据集，最终得到如表 4 所列的结果，其中“0”代表合约为正常合约，而“1”代表该合约具有对应类型的漏洞。

表 4 数据集标签分布

Table 4 Distribution of dataset labels

漏洞类型	0/个	1/个
拒绝服务漏洞	190	80
整数溢出漏洞	244	101
时间戳依赖漏洞	205	91
未检查返回值漏洞	165	110

5.2 实验结果与分析

在智能合约常见的 4 种漏洞上进行实验，实验机器的软硬件配置为：CPU 为 14 vCPU Intel(R) Xeon(R) Platinum 8362 CPU @ 2.80 GHz，GPU 为 RTX 3090(24 GB)，内存大小为 45 GB，在 Ubuntu 22.04 上使用 PyTorch 2.3.0，Python 3.12 和 Cuda 12.1 作为开发环境。根据 Mando 中的实验数据，与 6 种主流的传统漏洞分析工具以及 Mando 进行了对比，具体结果如表 5 所列。由于数据集中两种标签的样本数量并未达到 1∶1 的理想状态，因此使用了两项指标来评估对漏洞的检出能力和综合的分类准确度，buggy-F1 分数为对“1”标签的分类结果，而 macro-F1 分数为对两种标签分类结果的加权平均值。

表 5 与当前主流工具的实验结果的比较

Table 5 Comparison with experimental results of SOTA tools

(%)

方法	拒绝服务漏洞		整数溢出漏洞		时间戳依赖		未检查返回值	
	Buggy F1	Macro F1	Buggy F1	Macro F1	Buggy F1	Macro F1	Buggy F1	Macro F1
Securify	18.00	52.00	0	45.20	24.00	52.40	11.00	54.10
Mythril	41.00	60.10	73.00	84.10	23.00	50.80	14.00	55.70
Slither	13.00	42.70	0	45.20	44.00	57.30	10.00	53.30
Manticore	12.00	48.00	30.00	61.10	24.00	55.10	4.00	50.60
Smartcheck	52.00	69.90	22.00	56.10	44.00	64.20	11.00	54.10
Oyente	48.00	67.20	71.00	82.80	24.00	52.40	8.00	52.60
Mando	87.37	86.68	66.85	71.04	85.03	83.35	72.08	74.52
Ours	81.48	87.65	84.85	90.04	89.66	93.14	82.35	83.55

实验结果表明，本文提出的方法在智能合约漏洞检测任务上取得了显著的性能提升，特别是在拒绝服务漏洞、整数溢出漏洞、时间戳依赖漏洞和未检查返回值漏洞检测中的 F1 分数分别较最佳传统工具提升了 17.75，5.94，28.94，27.85 个百分点。对实验结果的进一步分析如下。

传统的智能合约漏洞检测工具(如 Securify 和 Mythril)在精度和召回率上均表现较差，其主要限制在于依赖预定义规则，这些规则对程序语义的捕获能力有限。此外，符号执行工具在处理复杂控制流时效率较低，容易受制于路径爆炸问题。相比之下，本文方法通过去噪技术有效地保留了与漏洞相关的关键语义，并通过异构图全面表达合约结构信息，从而显著提高了检测性能。

使用具有高误报率的工具进行漏洞检测工作难以得到客观的结果，需要智能合约安全专家对结果进行人工再次检查，因此误报率对于智能合约漏洞检测也是重要的评估指标。误报率分析选取了上述实验结果中表现较好的传统工具 Mythril 进行对比，由表 6 可知，除时间戳依赖漏洞之外，本文方法在误报率上均获得了不同程度的提升。

表 6 误报率分析

Table 6 Analysis of false positive rates

(%)

	Mythril	Ours
拒绝服务	12.11	7.90
整数溢出	15.98	3.69
时间戳依赖	0	1.46
未检查返回值	66.06	4.84

本文方法相较于同样使用图神经网络的方法，对除拒绝服务漏洞之外的 3 种漏洞的检出能力也有不同程度的提升。这一改进主要归因于提出的去噪策略有效剔除了合约中的非关键数据流和控制流，从而减少了模型训练中的噪声干扰。然而，对于拒绝服务漏洞检测效果不佳的原因可能在于该方法在 IR 层面未能充分捕捉与 gas 消耗相关的语义信息，导致无法识别因 gas 耗尽引发的拒绝服务攻击漏洞。

结束语 本文提出了一种新的方法，用于对以太坊智能合约字节码进行漏洞检测。在利用反编译器重建字节码语义信息的基础上，设计了独创的噪声过滤规则，并结合控制流图和调用关系的异构表示，构建出更能反映合约中可能存在的漏洞的合约语义图。经过真实世界的数据集评估，该方法相较于传统方法获得了较大提升。本文的工作有助于智能合约，尤其是对未公开代码的链上合约开展安全分析工作。由字节码构建合约语义图的工作，也可为其他下游任务(如识别智能合约中的金融风险)提供参考。

本文的工作还有进一步改进的空间，如向合约图中添加数据依赖关系，来更为显式地表示合约数据流。另外，对于经过读取和存储 EVM 内存而丢失的数据流，可以考虑设计新的方法来追踪。

参 考 文 献

[1] Wikipedia. The DAO [EB/OL]. (2024-08-16)[2024-12-03]. https://en.wikipedia.org/wiki/The_DAO.

[2] Slowmist. 2024 Mid-year Blockchain Security and AML Report. [EB/OL]. (2024-07-01)[2024-11-15]. https://www.slowmist.com/report/first-half-of-the-2024-report(CN).pdf.

[3] 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(WETSEB). IEEE, 2019: 8-15.

[4] ZHENG Z, SU J, CHEN J, et al. Dappscan: building large-scale datasets for smart contract weaknesses in dapp projects[J]. IEEE Transactions on Software Engineering, 2024, 50(6): 1360-1373.

[5] ZHUANG Y, LIU Z, QIAN P, et al. Smart contract vulnerability detection using graph neural networks [C]// Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence. 2021: 3283-3290.

[6] LIU Z, QIAN P, WANG X, et al. Smart contract vulnerability detection: from pure neural network to interpretable graph fea-

ture and expert pattern fusion [J]. arXiv:2106.09282,2021.
[7] NGUYEN H H,NGUYEN N M,XIE C,et al. Mando:Multi-level heterogeneous graph embeddings for fine-grained detection of smart contract vulnerabilities [C]// 2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA). IEEE,2020:1-10.
[8] LUO F,LUO R,CHEN T,et al. Scvhunter:Smart contract vulnerability detection based on heterogeneous graph attention network [C]// Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. 2024:1-13.
[9] Consensys. Mythril:Security analysis tool for EVM bytecode [DB/OL]. (2024-08-13) [2024-11-12]. https://github.com/Consensys/mythril.
[10] CHEN J,XIA X,LO D,et al. Defectchecker:Automated smart contract defect detection by analyzing evm bytecode [J]. IEEE Transactions on Software Engineering,2021,48(7):2189-207.
[11] LUU L,CHU D H,OLICKEL H,et al. Making smart contracts smarter [C]// Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016:254-269.
[12] TSANKOV P,DAN A,DRACHSLER-COHEN D,et al. Securify:Practical security analysis of smart contracts [C]// Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018:67-82.
[13] MOSSBERG M,MANZANO F,HENNENFENT E,et al. Manticore:A user-friendly symbolic execution framework for binaries and smart contracts [C]// 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE,2019:1186-1189.
[14] TORRES C F,IANNILLO A K,GERVAIS A,et al. Confuzzius:A data dependency-aware hybrid fuzzer for smart contracts [C]// 2021 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE,2021:103-119.
[15] CHOI J,KIM D,KIM S,et al. Smartian:Enhancing smart contract fuzzing with static and dynamic data-flow analyses [C]// 2021 36th IEEE/ACM International Conference on Automated Software Engineering(ASE). IEEE,2021:227-239.
[16] ZENG Q,HE J,ZHAO G,et al. EtherGIS:a vulnerability detection framework for ethereum smart contracts based on graph learning features [C]// 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC). IEEE, 2022:1742-1749.
[17] CONTRO F,CROSARA M,CECCATO M,et al. Ethersolve:Computing an accurate control-flow graph from ethereum bytecode [C]// 2021 IEEE/ACM 29th International Conference on Program Comprehension(ICPC). IEEE,2021:127-137.
[18] 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.
[19] LI Z,LU S,ZHANG R,et al. VulHunter:Hunting Vulnerable Smart Contracts at EVM bytecode-level via Multiple Instance Learning [J]. IEEE Transactions on Software Engineering, 2023,49(11):4886-4916.
[20] Smart Contract Weakness Classification (SWC) [EB/OL]. (2024-07-16)[2024-12-01]. https://swcregistry.io/.
[21] GRECH N,BRENT L,SCHOLZ B,et al. Gigahorse:thorough, declarative decompilation of smart contracts [C]// 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE,2019:1176-1186.
[22] TRUFFLE SUITE [EB/OL]. (2024-10-07) [2024-12-01]. https://archive.trufflesuite.com/docs/truffle/how-to/debug-test/use-truffle-develop-and-the-console/.
[23] YE M,NAN Y,ZHENG Z,et al. Detecting State Inconsistency Bugs in DApps via On-Chain Transaction Replay and Fuzzing [C]// Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. 2023:298-309.
[24] WANG X,JI H,SHI C,et al. Heterogeneous graph attention network [C]// The World Wide Web Conference. 2019:2022-2032.
[25] KINGMA D P. Adam:A method for stochastic optimization [J]. arXiv:1412.6980,2014.
[26] DURIEUX T,FERREIRA J F,ABREU R,et al. Empirical review of automated analysis tools on 47,587 ethereum smart contracts [C]// Proceedings of the 2020 ACM/IEEE 42nd International Conference on Software Engineering. 2020:530-541.

LI Chengyu, born in 1999, postgraduate. His main research interest is blockchain security.

CHEN Wei, born in 1978, Ph. D, associate professor. His main research interests include network security and blockchain security.

(责任编辑:李亚辉)