

基于漏洞关键代码的智能合约漏洞检测方法

彭 涛, 吴 胜

(江苏师范大学 计算机科学与技术学院, 江苏 徐州 221116)

摘 要: 智能合约在多个领域均取得了显著进展, 而确保其安全性和可靠性显得尤为关键, 这能够有效防范潜在的财务损失风险。鉴于智能合约具有不可篡改的特性, 在其部署前开展漏洞检测至关重要。尽管已有很多漏洞检测工具, 但它们大多依赖于静态的方法和预定义好的规则, 并且检测过程需要大量时间。鉴于此, 提出一种基于漏洞关键代码和深度学习框架的智能合约漏洞检测模型。该方法使用抽象语法树构建具有完整语义结构信息的漏洞关键代码, 以增加表达漏洞特征的能力。并且, 采用可拓展的深度学习特征提取模型, 包括 LSTM、GRU、Bi-LSTM 和 Bi-LSTM-Atten, 以提高智能合约漏洞检测性能。针对以太坊智能合约重入漏洞的真实数据集进行训练和测试, 实验结果显示, 该方法优于常用的 Mythril、Oyent、Smartcheck、TMP、CGE、DeeSCVH 等 6 种方法, 准确率达 94.32%, F1 分数达 88.69%。研究表明, 漏洞关键代码显著增强了该模型检测效果, 提升了漏洞检测效率。

关键词: 智能合约; 漏洞检测; 漏洞关键代码; 深度学习; 漏洞特征

DOI: 10.11907/rjtk.241972

扫描二维码阅读全文:

中图分类号: TP309

文献标识码: A

文章编号: 1672-7800(2026)002-0110-09



Smart Contract Vulnerability Detection Method Based on Vulnerability Critical Code

PENG Tao, WU Sheng

(School of Computer Science and Technology, Jiangsu Normal University, Xuzhou 221116, China)

Abstract: Smart contracts have made significant progress in various fields and it is vital to ensure their security and reliability to prevent potential financial losses. Due to the non-tamperability, it is important to perform vulnerability detection before smart contracts are deployed. Although many vulnerability detection tools are available, most of them rely on static methods and predefined rules, and the detection process takes a lot of time. In this paper, we propose a smart contract vulnerability detection model based on vulnerability critical code and deep learning framework. The method uses abstract syntax trees to construct vulnerability key codes with complete semantic structure information to increase the ability to express vulnerability features. And it uses extensible deep learning feature extraction models, including LSTM, GRU, Bi-LSTM and Bi-LSTM-Atten, to improve the performance of smart contract vulnerability detection. Training and testing are performed on a real dataset of Ethereum smart contract reentry vulnerabilities, and the experimental results show that this paper's method outperforms six commonly used methods, including Mythril, Oyent, Smartcheck, TMP, CGE, and DeeSCVH, and achieves an accuracy rate of 94.32% and an F1 score of 88.69%. The research shows that the vulnerability critical code has significantly enhanced the detection performance of this model, and the efficiency of vulnerability detection has been improved.

Key Words: smart contracts; vulnerability detection; vulnerability critical code; deep learning; vulnerability characterization

0 引言

区块链技术旨在创建一种去中心化、不可篡改的安全分布式账本, 用于记录和验证交易数据^[1]。该技术的起源

可追溯到比特币的白皮书, 该白皮书详细描述了一种去中心化数字货币系统的设计理念^[2]。比特币作为区块链技术的首个实际应用案例, 成功解决了双重支付问题, 为数字货币领域奠定了坚实基础。随着以太坊的推出, 区块链技术得到了快速发展。以太坊是首个引入智能合约(简称

收稿日期: 2025-01-04

基金项目: 江苏师范大学研究生科研与实践创新计划项目(2024XKT2599)

作者简介: 彭涛(1997-), 男, 江苏师范大学计算机科学与技术学院硕士研究生, 研究方向为软件漏洞检测; 吴胜(1978-), 男, 博士, 江苏师范大学计算机科学与技术学院副教授、硕士生导师, 研究方向为区块链、软件工程。本文通讯作者: 吴胜。

合约)概念的区块链平台^[3]。智能合约是一段能自动执行的程序代码,它的出现极大地拓展了区块链技术的应用范围,从去中心化应用到金融服务等各领域。

与其他程序段类似,智能合约也可能包含错误。设计不好的智能合约会暴露漏洞,恶意攻击者可以利用这些漏洞以获取经济利益,从而造成巨大的经济损失。2016年,就发生了TheDAO事件,黑客利用合同中的可重入漏洞窃取了价值6 000万美元的虚拟货币^[4]。文献[5]的数据表明,区块链网络已经因为安全漏洞遭受了超过数十亿美元的损失。与一般程序相比,漏洞带来的风险在智能合约中更加严重,因为一旦智能合约部署到区块链上,几乎不可能进行更新,且交易不可变,这意味着任何损失都无法挽回。除导致巨额的财务损失外,频繁的攻击还破坏了人们对区块链和智能合约应用程序的信任。因此,在部署智能合约之前进行漏洞检测至关重要。

目前,研究者已经提出了多种方法以检测智能合约中的漏洞。传统的漏洞检测技术,如程序分析^[6]、形式化验证^[7]、符号执行^[8]和模糊测试^[9]等方法。这些传统的检测方法依赖于专业知识,并且需要由专家预先定义安全漏洞模式属性,因而只能验证预先定义了安全属性的漏洞,可能会遗漏未知的漏洞或新型攻击。此外,采用专家定义漏洞模式属性的硬性规定还可能会导致高误报率。

随着深度学习技术的发展,深度学习方法在代码漏洞和恶意软件检测方面都表现出了卓越的性能,多种基于深度学习的技术被提出,用以检测智能合约漏洞。例如,使用循环神经网络(Recurrent Neural Network, RNN)^[10]和图神经网络(Graph Neural Network, GNN)^[11]以提取智能合约的漏洞特征。智能合约作为一段能自动执行的程序代码,其代码序列包含了丰富的信息。智能合约代码中的函数调用和数据流通常与漏洞有关,GNN及其变体因其能够有效提取不规则结构的特征,已被广泛应用于此领域^[10-11]。

然而,目前基于深度学习技术的智能合约漏洞检测方法仍存在一些不足。例如,提取的代码往往无法较好地表示漏洞的语义信息。现有的漏洞检测模型需作进一步改进,以捕捉有效漏洞特征并提高检测精度。

为了解决上述问题,本文提出基于漏洞关键代码和深度学习的智能合约漏洞检测方法。在该方法的漏洞特征提取阶段整合了4种深度学习模型,包括长短期记忆网络(Long Short Term Memory Network, LSTM)^[12]、门控循环单元(Gated Recurrent Unit, GRU)^[13]、双向长短期记忆网络(Bi-LSTM)^[14]和双向长短期记忆网络加上注意力机制(Bi-LSTM+Atten)^[15]。该方法可以通过学习易受攻击的智能合约的漏洞特征,自动识别出智能合约中是否存在漏洞。

为了实现该方法,本文提出了漏洞关键代码的概念。首先,通过ANTLR(ANother Tool for Language Recognition)将智能合约源代码解析成抽象语法树、函数、参数和用户

定义的变量^[16];其次,基于预定义好的漏洞特征,运用匹配算法获取漏洞关键片段,并通过追踪与这些漏洞关键片段存在数据流和控制流关联的代码元素,精准定位漏洞相关语句,从而组成漏洞关键代码;再次,通过定义声明语句、控制语句和外部调用异常语句等语句集,以定位具有语义结构信息的语句,并将范围语句添加到漏洞相关代码中的适当位置,构建了具有语义结构信息的漏洞关键代码。具有语义结构信息的漏洞关键代码既保留了足够的特征,又去除了冗余信息,增强了表达漏洞特征的能力,提高了漏洞检测效率。

1 相关工作

现有的智能合约漏洞检测方法大致可以分为两大类:基于专家知识和自动化检测工具的传统检测方法、基于机器学习和深度学习(简称“基于深度学习”)方法。传统检测方法包括形式化验证、符号执行、模糊测试和程序分析等。

1.1 传统检测方法

形式化验证方法基于数学和逻辑,对智能合约进行验证。首先,将智能合约抽象为数学模型,并定义需要验证的安全属性;其次,通过形式化验证工具自动探索所有可能的状态和行为,以确认智能合约是否满足定义的安全属性。Garfatta等^[17]提出一种名为F*的形式化验证框架,该框架将以以太坊虚拟机字节码的语义转化为一种函数式编程语言,以分析和验证智能合约的安全性。这种方法包含以太坊虚拟机字节码语义的完整形式化定义,基于这种形式化定义,能够生成可执行代码,而且可以与官方以太坊测试套件进行对比验证。形式化的验证方法通常需要与严格的逻辑证明相结合,因此不能实现高度自动化。

符号执行方法中,智能合约的代码被抽象为符号表达式而非具体数值,通过追踪这些符号值在合约执行过程中的变化以识别潜在漏洞。Securify即是先从合约的依赖图中提取精确的语义信息,再检查合规和违规模式,这些模式捕捉了证明合约安全或不安全的充分条件^[18]。Mythril通过解析和执行智能合约以太坊虚拟机字节码,生成控制流图和数据流分析,进行符号执行和违规模式匹配,从而检测和验证智能合约中的安全漏洞^[19]。符号执行方法由于检测智能合约漏洞而面临状态空间爆炸、路径约简不完备、计算资源消耗大和输入符号化局限等问题,应用不够广泛。

模糊测试技术通过模糊测试工具生成大量随机或非预期输入数据,形成大量测试用例,监控智能合约的执行过程,从而捕获异常行为^[20]。ContractFuzzer将智能合约的应用二进制接口(Application Binary Interface, ABI)规范生成随机或异常的输入数据,以覆盖智能合约的各种可能输入^[9]。通过一组测试预言机以检测智能合约的安全漏洞,

这些预言机常用于捕捉合约执行过程中出现的异常行为。模糊测试方法存在智能合约中覆盖率不足、难以发现复杂逻辑漏洞、依赖测试用例质量及性能瓶颈等问题,这限制了其在复杂合约中的应用效果。

程序分析技术将智能合约源代码转换为中间代码,然后使用模式匹配技术以查找错误。SmartCheck 通过将智能合约源代码转换为可扩展标记语言的中间表示形式,并使用 XPath 模式进行检查^[6]。Slither 将 Solidity 智能合约转换为一种称为 SlithIR 的中间表示形式,SlithIR 采用静态单赋值形式,并使用简化的指令集,以便于实现各种分析^[21]。基于程序分析的智能合约漏洞检测方法存在误报、路径爆炸、动态行为捕获不足等问题,且难以处理复杂合约的交互和优化而导致的代码差异,检测效率和覆盖率不佳。而且,传统的检测方法依赖于固定的检测规则,这导致智能合约漏洞误报率很高。

1.2 基于深度学习的漏洞检测方法

随着人工智能技术的发展,越来越多研究人员使用机器学习技术和深度学习技术进行智能合约漏洞检测。Gupta 等^[22]提出基于深度学习的智能合约漏洞检测方案,该方案分析了智能合约的操作码序列,通过 LSTM、ANN 和 GRU 提取出有助于识别恶意行为的特征。Qian 等^[23]通过提取智能合约的操作码序列并结合长短期记忆网络,聚焦于关键代码片段,从而更有效地检测出潜在的重入漏洞。SCScan 通过静态分析技术,从智能合约源代码中提取出可能与漏洞相关的特征^[24]。这些特征包括代码的结构、调用模式,以及特定操作码的使用频率等。将提取到的特征转化为特征向量,输入到 SVM 模型中进行分类。Huang 等^[25]提出一种结合智能合约源代码和操作码分析的方法,该方法利用源代码提供的逻辑信息和操作码的底层执行细节,通过深度学习模型进行漏洞特征提取,通过结合这两种代码,可以更准确地识别智能合约中的漏洞。Eth2Vec 使用一种基于文档嵌入模型的神经网络以处理智能合约字节码,该方法将字节码被转换为 JSON 文件,这些文件包含了指令级别、块级别、函数级别和合约级别的代码表示^[26]。在这些表示上进行无监督学习,以计算待检测合约与已知漏洞合约之间的代码相似度。Zhang 等^[27]通过深度语义提取检测智能合约中的漏洞,该方法结合了双向 LSTM 和注意力机制,从代码的标记序列中提取全局语义特征,同时从抽象语法树中提取结构化特征。

TMP 通过构建智能合约图来表示智能合约函数的语法和语义结构,并设计了一个消除阶段以规范化图表,突出显示主要节点^[28]。并且,Zhuang 等^[28]提出了图卷积神经网络(Graph Convolutional Networks, GCN)和一种新的时间消息传播网络,用于从规范化图表中学习以检测漏洞。Huang 等^[29]提出了一种基于颜色的检测方法,该方法将智能合约字节码转换为 RGB 颜色代码,并将其转化为固定大小的图像,将这些图像输入卷积神经网络(Convolutional

Neural Networks, CNN),进行自动特征提取和学习。Zhen 等^[30]使用图神经网络来表示智能合约的控制流图和操作码序列,通过对这些图结构的分析,图神经网络能够更好地理解智能合约的语法和逻辑关系,从而提高漏洞检测精度。Cai 等^[31]提出了一种基于图神经网络智能合约漏洞检测方法,利用抽象语法树、控制流图和程序依赖图构建联合代码图以综合表示合约的语法和语义特征。同时,通过程序切片技术减少无关节点干扰,并在模型训练中使用双向图表示增强上下文特征学习。Liu 等^[32]使用图结构分析技术以定位智能合约中的漏洞,该方法聚合来自多个关系图的信息,以捕捉控制流、数据流和语句执行顺序等多种关系。

然而,当前基于深度学习的漏洞检测方法依然存在一些局限性。例如,提取的代码往往难以充分表达漏洞的语义信息,从而影响检测效果。此外,现有的漏洞检测模型在捕捉有效特征方面仍需改进,以提升整体检测精度。

2 智能合约漏洞检测方法

2.1 总体架构

本文检测方法架构如图 1 所示,它包括数据预处理、深度语义提取和漏洞检测 3 个阶段。第 1 个阶段包括智能合约关键漏洞代码提取方法;第 2 个阶段是一个可扩展模块,它实现了特征提取,在该模块中整合了长短期记忆网络(LSTM)、门控循环单元(GRU)、双向短期记忆网络(Bi-LSTM)和双向短期记忆网络加上注意力机制(Bi-LSTM+Atten)4 个训练模型,这 4 个训练模型用于漏洞特征提取;第 3 个阶段的分类模块采用全连接层作为分类器,并利用 Softmax 函数对智能合约中是否有漏洞进行分类。

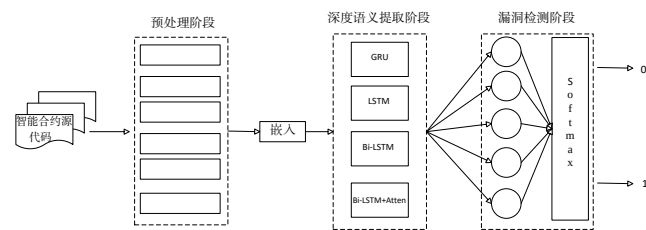


Fig. 1 Overall structure

图 1 总体架构

2.2 定义

定义 1(智能合约):给定一个智能合约 C ,假设 C 包含的函数定义为集合 $F=\{f_1, f_2, \dots, f_n\}$,表示为 $C=\{f_1, f_2, \dots, f_n\}$ 。每个函数 $f_i(1 \leq i \leq n)$ 是一个有序的语句序列集合 $\{S_{i1}, S_{i2}, \dots, S_{im}\}$,表示为 $f_i=\{S_{i1}, S_{i2}, \dots, S_{im}\}$ 。

定义 2(智能合约语句):每条语句 S_{im} 是一个有序的令牌集合 $\{T_{im1}, T_{im2}, \dots, T_{imj}\}$,表示为 $S_{im}=\{T_{im1}, T_{im2}, \dots, T_{imj}\}$ 。令牌 T 由标识符、操作符、常量和关键字组成。

定义 3(可重入漏洞):一个智能合约的函数 f_i 将以太币转移到另一个智能合约 sc_1 时, sc_1 中的回退函数会自动

触发。如果 sc_1 是恶意的,它可以在回退函数中再次调用原始合约的 f_1 函数,从而多次重复资金转移操作,导致以太币被不正当窃取。

定义4(漏洞关键片段):给定一组漏洞特征 P ,表示为 $P=\{P_1, P_2, \dots, P_n\}$ 。对于智能合约 C 中的函数 F ,如果任一语句 $S_j (1 \leq j \leq m)$ 匹配预定义的特征 P ,表示为 $S_j \in vcs$,而漏洞关键片段的全部内容 $vcs=\{S_1, S_2, \dots, S_m\}$ 。

定义5(漏洞关键代码):在漏洞关键片段 vcs 中找到与关键变量有关的数据依赖和控制依赖的语句,构成最终的漏洞关键代码 vkf ,表示为 $vkf=\{S_{j1}, S_{j2}, \dots, S_{jm}\}$ 。

定义6(语义结构):对于智能合约 C 定义3种语义结构信息语句,分别为 $FunSet_1$ 、 $ConSet_2$ 、 $ExcSet_3$ 。 $FunSet_1$ 为函数语句,通常以关键字(如"function")开头,其后跟随函数名和参数列表等信息。 $ConSet_2$ 为控制语句,通常包含条件语句(如"if" "else")、循环语句(如"while" "for")、跳转语句(如"break" "continue")和返回语句(如"return")等。 $ExcSet_3$ 为异常语句,通常包含异常处理语句(如"try" "catch")。综合表示为 $Set=FunSet_1 \cup ConSet_2 \cup ExcSet_3$ 。

定义7(具有完整语义结构信息的漏洞关键代码):对于漏洞关键代码 vkf ,通过定位具有语义结构信息的语句后构建具有完整语义结构信息的漏洞关键代码 $svkf$,表示为 $svkf=vkf \cap Set$ 。

2.3 预处理阶段

2.3.1 漏洞关键代码提取

智能合约漏洞存在于一个特定函数中,而且仅有部分语句与这些漏洞直接相关。因此,可以将智能合约划分为更小的代码片段(语句),以生成漏洞关键代码。如图2所示,先解析智能合约源代码,生成抽象语法树(AST)等相关信息,通过与预定义的漏洞特征 P 匹配以生成智能合约的(部分)漏洞关键片段。随后,根据漏洞关键片段,通过跟踪数据流和控制流(数据依赖和控制依赖)以获取与该语句相关的其他语句。这些所有相关语句共同构成了新的(完整的)漏洞关键代码。

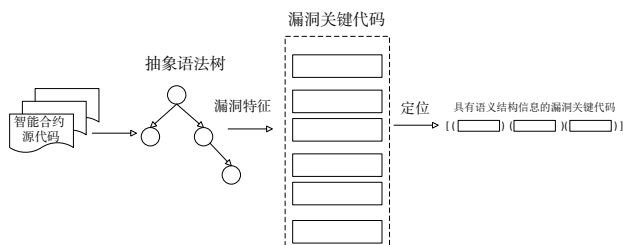


Fig. 2 Preprocessing phase

图2 预处理阶段

在算法1中,首先,需要定义用于匹配关键片段的漏洞特征 P ,漏洞特征 p 通常与智能合约的核心功能或安全性密切相关,如涉及资金转移的函数、权限控制逻辑、变量操作等。例如,在重入漏洞中与交易金额相关的常量 $call.value$ 通常作为漏洞特征 P 。漏洞特征 P 用于定位可能存在漏洞的关键片段(vcs)。接着,用ANTLR4解析Solidity

源代码,并生成抽象语法树(AST),抽象语法树中每个节点代表源代码中的一个元素,例如变量、函数、操作符和控制流语句等^[16]。通过抽象语法树提取智能合约中每个函数的重要元素,确保能够捕捉到合约的核心逻辑和功能。

如果某个语句符合漏洞特征 P ,则将其视为漏洞关键片段 vcs 。在找到漏洞关键片段 vcs 后,继续遍历函数 f 中的所有语句 s ,通过分析程序的控制流和数据流,识别与 vcs 具有数据依赖或控制依赖的其他语句。如果某个语句 s 与 vcs 存在数据依赖或控制依赖关系,则将该语句也添加到 vcs 中,以构建完整的漏洞关键代码(vkf)集合。

算法1:从源代码中提取漏洞关键代码

Input: 漏洞特征 p ;智能合约源代码 c

Output: 漏洞关键代码 vkf

// vcs 用于存放漏洞特征 p 的语句

1. $vcs \leftarrow \{\}$

// vkf 用于存放最终识别出的漏洞关键代码

2. $vkf \leftarrow \{\}$

3. For each statement s in c do

//检查当前语句 s 是否与给定的漏洞特征 p 匹配

4. If p matches s then

5. $vcs.add(s)$

6. $vkf.add(vcs)$

//找到第一个匹配后退出当前循环(防止重复添加)。

7. break

8. End if

9. End for

//再次遍历智能合约源代码 c 中的每条语句 s

10. For each statement s in c do

//检查当前语句 s 是否与候选集合 vcs 中的任一语句存在数据依赖或控制依赖关系

11. If s has data or control dependency with vcs then

//如果存在依赖关系

12. $vkf.add(s)$

13. End if

14. End for

15. Return vkf

2.3.2 语义结构信息添加

由算法1可知,漏洞关键代码捕获与漏洞特征匹配的漏洞关键片段,漏洞关键代码提取了合约在执行过程中可能会导致漏洞发生的相关代码片段,但忽略了智能合约代码中的语义结构信息,如语句的作用域范围。图3表示了一个重入漏洞的关键代码,代码第12行 $call$ 函数是一个低级别的外部调用方式,它可以调用目标地址上的任意函数。由于该函数直接发送以太币并调用目标合约中的函数,如果目标合约在接收以太币后立即触发恶意代码,这可能会导致重入攻击,即目标合约在控制权返回当前合约之前再次调用第10行 $transfer$ 函数,从而导致逻辑重复执行或资金被盗。然而,关键漏洞代码忽略了与漏洞相关的

关键语义结构信息,并不知道 transfer 函数的作用域范围。

```

1. function buy(
2.   address ERC20_token,
3.   address[] memory _exs,
4.   uint[] memory _indexes,
5.   uint256[] memory _values
6. ) public payable {
7.   for (uint i = 0; i < _exs.length; i++) {
8.     Bytes memory data = new bytes(_indexes[i + 1] - _indexes[i]);
9.     if (ERC20_token != address(0) && i > 0) {
10.      transfer(ERC20_token, _values[i]);
11.    } else {
12.      require(_exs[i].call{value: _values[i]}(data), "Call failed");
13.    }
14.  }
15. }

```

Fig. 3 Adding structure information statements

图3 添加结构信息语句

语义结构信息包括函数作用域、控制语句的范围等,对智能合约漏洞识别至关重要。在没有语义结构信息的情况下,关键漏洞代码可能无法识别出哪些代码是在特定条件下执行(如 if-else 语句)。通过引入语义结构信息,关键漏洞代码可以更全面地捕捉代码中的结构信息,从而更准确地识别和表达漏洞特征。为提高漏洞识别的精确性,可以在关键漏洞代码中添加语义结构信息,从而构建具有完整语义信息的关键漏洞代码。添加语义结构信息这一阶段主要包括定位具有语义结构信息的语句、在适当位置插入具有作用域范围的语句等两个步骤。

在关键漏洞代码中添加语义结构信息的第一步是语义结构语句定位。本文定义 6 定义了函数语句、控制语句和异常语句 3 类结构语句。逐行检查智能合约源代码中的语句,对每一行代码或语句,应用定义的规则集,检查该语句是否匹配某一条规则。例如,检查某行代码是否包含 function 关键字,以确定其是否为一个函数声明。如果某行代码匹配到规则集中的某一条规则,那么会将这行代码标记为包含语义结构信息的语句,并记录其起始和结束位置。

当识别到一个具有语义结构信息的语句时,遍历该作用域范围内的所有语句。在遍历过程中,判断关键漏洞代码中是否已存在语义结构作用域内的语句,如果存在,则将语义结构信息语句的开始语句和结束语句添加到程序片中,以确保关键漏洞代码能够完整地包含语义结构信息,如算法 2 所示。

通过语义结构信息语句定位,插入具有作用域范围的语句,生成了具有语义结构信息的漏洞关键代码(svkf),它不仅反映了代码的依赖关系,还保留了重要的语义结构信息。具有语义结构信息的漏洞关键代码(svkf)为漏洞检测提供了更丰富的上下文信息,增强了检测准确性和可靠性。

2.4 深度语义处理阶段

由于深度学习模型通常使用向量作为输入,因此需要

算法 2: 添加语义结构信息

Input: 关键漏洞代码 vkf

Output: 带有语义结构信息的关键漏洞代码 svkf

// set_i 代表一个具有特定语义结构的语句块

1. Set ← {set₁, set₂, ..., set_m}

// scopeSet 用于存储当前处理的 set 中的语句

2. scopeSet ← {}

// 将算法 1 中的关键漏洞代码 vkf 赋值给 svkf

3. svkf ← vkf

4. For each set_i in Set do

// returnScope(Set_i) 获取 set_i 语义结构信息

5. scopeSet ← returnScope(Set_i)

// returnLength(scopeSet) 获取当前语句集合长度

6. k ← returnLength(scopeSet)

// 遍历 scopeSet 中的每条语句 j

7. For j ← 0 to k do

// 搜索 scopeSet 中第 j 条语句在 svkf 中的位置

8. position ← searchStatement(svkf, scopeSet_j)

9. If position ≠ 0 then

// 获取起始语句位置

10. scopeStartPos ← returnStartPos(scopeSet_j)

11. scopeEndPos ← returnEndPos(scopeSet_j)

// 获取当前语句范围的起始语句内容

12. scopeStartStm ← returnStartSta(scopeSet_j)

13. scopeEndStm ← returnEndStat(scopeSet_j)

// 如果起始语句不在原始的关键漏洞代码 vkf 中

14. If scopeStartStm_i not in vkf then

// 将起始语句添加到 svkf 的正确位置

15. addSta(scopeStartPos, scopeStartStm, svkf)

16. If scopeEndStm not in vkf then

17. addSta(scopeEndPos, scopeEndStm, svkf)

18. End for

19. End if

20. Return svkf

将智能合约转换为向量表示。在转换每个智能合约向量为序列之前,先对其进行规范化,以确保统一的格式并减少与错误检测无关的冗余信息。首先,移除智能合约中所有非ASCII字符和注释,确保代码的整洁性。其次,将智能合约中定义的数值变量替换为令牌<NUM>;将智能合约中定义的字符串变量替换为令牌<STR>;将智能合约中定义的地址变量替换为令牌<ADDR>。通过这些规范化步骤,智能合约中的符号更加规范,避免了数据噪声的影响。如算法3所示。

算法3:漏洞关键代码规范化

```
Input: 带有语义结构的漏洞关键代码 svkf
Output: 规范化的漏洞关键代码 ssvkf
// ssvkf用于存放规范化后的漏洞关键代码
1. ssvkf ← {}
2. For each statement s in svkf
  // 从语句s中移除非ASCII字符和注释
3. Remove non-ASCII characters, comments from s
4.   For each token t in s
  // 如果令牌t是一个变量或参数
5.     If t is a variable or parameter then
  // 将变量或参数替换到一个符号名称,如<NUM>
6.       replace t to a symbolic name
7.     End if
8.   ssvkf.add(s)
9. End for
10. Return ssvkf
```

对规范化后漏洞关键代码中的每个语句进行解析,并将其拆分为一组令牌。随后,按照语句在代码中出现的顺序连接这些令牌。利用这些令牌集,使用Word2vec嵌入模型将漏洞关键代码转换为向量表示。通过这种嵌入方法学习的令牌嵌入可以捕捉到令牌之间的语义关系。

在可扩展的训练模块中构建了4个模型,其中包括GRU、LSTM、Bi-LSTM和Bi-LSTM+Atten模型。由于具有语义结构信息的漏洞关键代码具有重入漏洞关键特征和语义结构信息,因而能够反映代码的全局语义。基于这一特点,本文将漏洞关键代码视为序列数据,并借助自然语言处理中的多种强大模型,以解决漏洞检测问题。与RNN网络相比,GRU和LSTM具有更强的序列数据处理能力。Bi-LSTM使用双向LSTM,能够从漏洞关键代码中提取全局特征,有效克服了梯度消失问题,且能够同时捕捉序列中的历史和未来信息。Bi-LSTM+Atten使用Bi-LSTM提取智能合约代码的上下文语义,注意力模块可以更有效地提取智能合约漏洞特征。

这4个模型以向量作为输入,并通过标准的训练过程进行学习。最终,获得了4个经过训练的深度学习模型,用作智能合约漏洞特征提取器。

2.5 分类阶段

将全连接层模型作为分类器以检测智能合约漏洞,全

连接层每个神经元都与上一层的所有神经元相连。使用Softmax函数输出有漏洞和无漏洞的概率值,从而检测出智能合约中的漏洞。

3 实验与比较分析

为了评估本文方法的有效性,设计了实验以回答以下研究问题:

RQ1:与其他漏洞检测方法相比,文中方法在检测智能合约中的漏洞检测方面效果如何?

RQ2:不同深度学习模型的性能如何?

RQ3:漏洞关键代码对文中方法有何贡献?

3.1 实验设置与数据集

本文实验是基于深度学习框架Keras而完成,实验运行环境为Windows 10, Intel Core i5-13400 CPU, 16 GB RAM, NVIDIA GTX4060ti。

从Etherscan和GitHub上收集了1729个智能合约源代码用于构建数据集。将整个数据集随机分成5个大致相等的子集,选择其中4个子集作为训练集,剩下的1个子集作为验证集。模型在训练集上进行训练,而后在验证集上评估其性能。该过程重复5次,每次都选择不同的1个子集作为验证集,其他4个子集作为训练集。最终,将5次验证的结果进行平均,以得到模型整体性能。

3.2 评价指标

本文采用Accuracy(准确率)、Recall(召回率)、Precision(精确率)和F1-Score(F1分数)作为评价指标,这些指标可描述为如式(1)~式(4)所示。

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (3)$$

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

其中,真阳性(TP)表示正确检测到具有可重入漏洞的样本数量;假阳性(FP)表示未检测到具有可重入漏洞的样本数量;真阴性(TN)表示正确检测到没有可重入漏洞的样本数量;假阴性(FN)表示检测到没有漏洞但被标记为有漏洞的样本数量。

3.3 与其他常用检测方法比较

表1展示了在使用相同训练集和测试集的前提下,针对研究问题RQ1的实验结果。对比了本文方法与3种传统检测方法Mythril^[17]、Oyente^[8]、Smartcheck^[5],3种基于深度学习的自动化检测方法TMP^[25]、CGE^[29]和DeeSCVH^[33]等6种具有代表性的漏洞检测工具的效果。

表1反映了基于深度学习的方法与传统的非机器学习方法比较结果。基于EVM字节码通过CFG进行符号检查的Oyente在准确率上优于基于源代码分析的Mythril和

Smartcheck 工具。Mythril 在召回率上的表现不如 Oyente 和 Smartcheck 这两种工具。Smartcheck 的召回率远高于准确率,说明 Smartcheck 的误报率很高。这 3 种方法的总体性能存在明显偏差。

Table 1 Comparison of the proposed method with other commonly used detection methods

表 1 本文方法与其他常用检测方法比较 (%)

模型	Accuracy	Recall	Precision	F1-Score
Mythril	63.42	44.38	60.90	51.34
Oyente	70.26	75.47	52.96	62.24
Smartcheck	53.00	84.30	42.13	56.18
TMP	78.69	62.01	86.73	72.31
CGE	92.16	82.56	91.63	86.85
DeeSCVH	93.02	90.70	83.46	86.92
Our Method	94.32	90.37	87.02	88.66

TMP 构建了合约图,用来表示智能合约函数的语法和语义结构,并设计了消除阶段以规范化此图。CGE 利用图神经网络结合专家规则特征对合约源代码进行特征提取,用以检测漏洞。DeeSCVH 采用漏洞候选切片概念以帮助模型捕获漏洞的关键点,可能忽略了与漏洞相关的关键语义信息。在没有语义结构信息的情况下,程序切片可能无法识别出哪些代码是在特定条件下执行的。TMP 比传统方法具有更高的准确率和 F1-Score,但是它们都未能超过 80%。CGE 和 DeeSCVH 在上述 4 个指标上有了显著提升,这两种检测方法在准确率上比 TMP 提高了 14% 左右。

3 种传统检测方法的效果要逊于 3 种基于深度学习的方法,其原因在于,这 3 种传统检测方法依赖单一且固定的专家模式以进行漏洞分类,无法适应灵活变化的智能合约。

本文方法在可重入漏洞检测方面整体优于所有与之对比的 6 种方法,准确率达到 94.32%, F1-Score 达到 88.66%。与 CGE 相比,其准确率提高了 2.1% 左右, F1-Score 提高了 1.8% 左右。与 DeeSCVH 相比,其准确率提高了 1.3% 左右, F1-Score 提高了 1.7% 左右。原因在于,本文方法使用了带有语义结构信息的关键漏洞代码,删除了冗余信息,增加了漏洞语义特征。CGE 的精确率高于本文方法,但是 CGE 的召回率偏低,这说明 CGE 存在一定的误报率。本文方法精确率和召回率差值很小,说明在可重入漏洞检测中具有更高的准确性和更好的整体性能。

表 2 展示了基于深度学习的方法与传统的非机器学习方法在漏洞检测时间上的比较。第 2 列表示检测次数,第 3 列为总检测时间,第 4 列展示了检测单个数据所需的平均时间。结果显示,传统的非机器学习方法在平均检测时间上要长于基于深度学习的方法。Mythril 的单次检测时间最长,为 64.53 s, Oyente 的单次检测时间为 13.42 s, Smartcheck 的单次检测时间为 4.65 s, TMP 的单次检测时间为 1.64 s, CGE 的单次检测时间为 2.56 s, DeeSCVH 的单次检测时间最短,为 0.83 s。而本文方法单次检测时长为 1.24 s,表现出较好的时间优势。

Table 2 Comparison of the average detection time of the method in this paper with other commonly used detection methods

表 2 本文方法与其他常用检测方法平均检测时间比较

模型	Number	Total time /s	Average time /s
Mythril	100	6453	64.53
Oyente	100	1342	13.42
Smartcheck	100	465	4.65
TMP	100	164	1.64
CGE	100	256	2.56
DeeSCVH	100	83	0.83
Our Method	100	124	1.24

3.4 不同深度学习模型检测方法比较

表 3 展示了针对研究问题 RQ2 的实验结果。在特征提取的可拓展模块中嵌入了 4 种特征提取模型,包括 LSTM、GRU、Bi-LSTM 和 Bi-LSTM+Atten。GRU(门控循环单元)、LSTM(长短期记忆网络)和 Bi-LSTM(双向—门控循环单元),它们都是用于处理序列数据的神经网络模型。

Table 3 Comparison of different deep learning detection methods

表 3 不同深度学习模型检测方法比较 (%)

模型	Accuracy	Recall	Precision	F1-Score
LSTM	88.02	84.69	80.12	82.34
GRU	86.69	73.37	84.40	78.49
Bi-LSTM	90.18	88.46	83.21	85.74
Bi-LSTM+Atten	94.32	90.37	87.02	88.66

在这 4 个模型上均采用相同的训练集和测试集。由于 LSTM 和 GRU 在结构上相似,故这两种方法在可重入漏洞的检测效果上差不多。其中,在准确率和 F1-Score 这两个指标上, LSTM 比 GRU 分别提高了 1.3% 和 3.8%。

Bi-LSTM 是对 LSTM 的扩展,它由两个独立的 LSTM 网络组成,一个处理输入序列的正向信息,另一个处理反向信息。Bi-LSTM 能够同时考虑从前到后和从后到前的上下文信息,故 Bi-LSTM 比 LSTM 提取到了更多的可重入漏洞特征。在准确率和 F1-Score 这两个指标上, Bi-LSTM 比 LSTM 分别提高了 2.1% 和 3.4%。

Bi-LSTM+Atten 是在 Bi-LSTM 的基础上,引入注意力(Attention)机制。注意力机制是一种用于提升模型处理长序列数据效果的技术,在注意力机制中,查询、键和值都是从同一输入序列中得到。注意力机制通过计算输入序列中每个元素与其他元素之间的相似度,为每个元素分配一个权重,然后对序列进行加权求和,生成新的表示。这种机制允许模型在进行每个时间步的预测时,动态关注序列中最相关的部分,从而更好地捕捉长距离依赖关系。这弥补了 Bi-LSTM 捕捉长距离依赖关系能力不足的问题。在准确率和 F1-Score 这两个指标上 Bi-LSTM+Atten 比 Bi-LSTM 分别提高了 4.1% 和 2.9%。Bi-LSTM+Atten 作为本方法的特征提取模型时,其在 4 个指标上都明显优于其他模型。

本文实验参数设置如表 4 所示。Dropout 设置为 0.3,

批量设置为 32, 轮数设置为 100。

Table 4 Comparison of parameters of different deep learning models
表 4 不同深度学习模型参数比较

模型	Number of layers	Dropout	Batch size	Number of epochs
LSTM	2	0.3	32	100
GRU	2	0.3	32	100
Bi-LSTM	1	0.3	32	100
Bi-LSTM+Atten	1	0.3	32	100

在深度学习模型训练过程中, 学习率对检测性能的影响尤为显著。当学习率设置过低时, 训练过程可能过于缓慢, 难以在合理时间内达到收敛; 而当学习率过高时, 则可能导致训练的不稳定性, 甚至使模型无法收敛。图 4 展示了本文实验选择的 5 种不同学习率: 0.001、0.005、0.01、0.05 和 0.1。



Fig. 4 Comparison of the accuracy of deep learning detection methods at different learning rates

图 4 不同学习率下深度学习模型检测方法准确率比较

当学习率设置为 0.1 时, 检测性能显著低于其他学习率。随着学习率的逐步降低, 模型准确率逐渐提升, 当学习率下降至约 0.005 时, 检测性能达到最佳。因此, 实验中将学习率设置为 0.005。

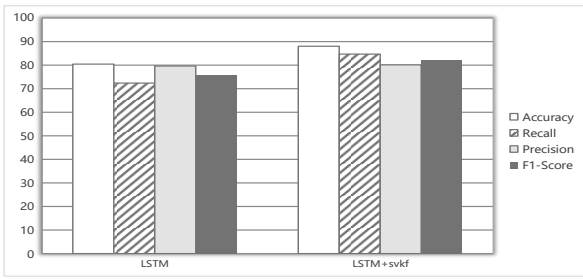
3.5 关键漏洞代码对本文方法的贡献

图 5 展示了针对研究问题 RQ3 的实验结果, 分别展示了基于智能合约源代码和基于具有语义结构信息的关键漏洞代码对可重入漏洞的检测效果。可以看出, 基于具有语义结构信息的关键漏洞代码对可重入漏洞的检测效果在 4 个评价指标上都得到了显著提升, 准确率提升都在 7.5% 以上, F1-Score 都提升了至少 6.1%。

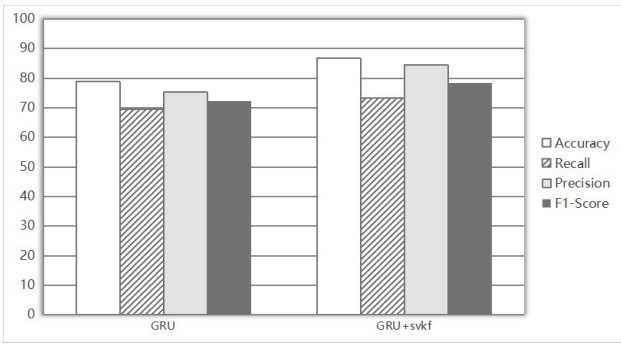
在总体性能最好的 Bi-LSTM+Atten 上, 准确率和 F1-Score 这两个指标分别提高了 10.3% 和 8.25%。这是由于针对可重入漏洞, 关键漏洞代码包含了漏洞关键点的数据依赖和控制依赖, 更好地学习漏洞特征。同时, 关键漏洞代码通过定位语义结构语句的位置并添加作用域范围的语句, 增加了代码表达漏洞特征的能力。

4 结语

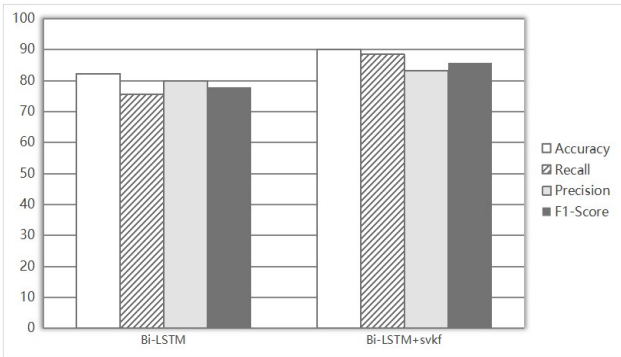
针对智能合约重入漏洞的安全问题, 本文提出了基于漏洞关键代码和深度学习技术的漏洞检测方法。漏洞关



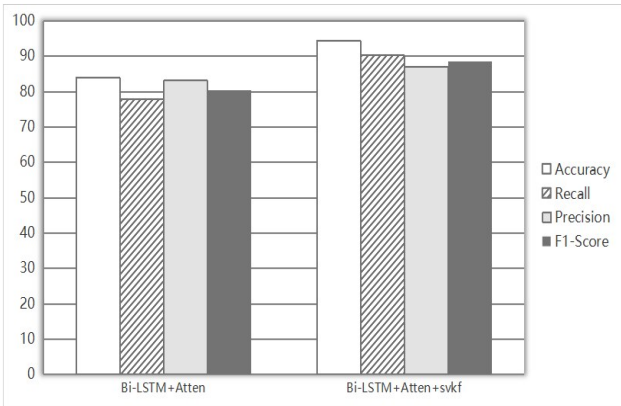
(a) Impact of vulnerability critical code on LSTM
(a) 漏洞关键代码对 LSTM 的影响



(b) Impact of vulnerability critical code on GRU
(b) 漏洞关键代码对 GRU 的影响



(c) Impact of vulnerability critical code on Bi-LSTM
(c) 漏洞关键代码对 Bi-LSTM 的影响



(d) Impact of vulnerable critical code on Bi-LSTM+Atten
(d) 漏洞关键代码对 Bi-LSTM+Atten 的影响

Fig. 5 Impact of vulnerability critical code
图 5 漏洞关键代码的影响

键代码聚焦于漏洞关键点,捕获了漏洞相关丰富的语义信息,通过定位语义结构信息语句的位置,添加漏洞关键代码的语义结构信息,增强漏洞特征的能力。该方法由一个可拓展的特征提取模型、全连接层和 Softmax 函数组成,用于检测智能合约的漏洞。实验结果显示,本文模型在重入漏洞的检测上展现出了良好的召回率、准确率和 F1 结果,优于其他 6 种常用的漏洞检测方法。并且,漏洞关键代码显著提升了本文模型检测效果,Bi-LSTM-Atten 模型被证明是检测效率较高的分类模型之一。

参考文献:

- [1] KHAN S N, LOUKIL F, GHEDIRA G C, et al. Blockchain smart contracts: applications, challenges, and future trends [J]. *Peer-to-Peer Networking and Applications*, 2021, 14: 2901–2925.
- [2] BUTERIN V. A next-generation smart contract and decentralized application platform [Z]. *Ethereum White Paper*, 2014: 1–36.
- [3] ZOHAR A. Bitcoin: under the hood [J]. *Communications of the ACM*, 2015, 58(9): 104–113.
- [4] TheDAOcontract [EB/OL]. <https://etherscan.io/address/0xbb9bc244d798123fde783fcc1c72d3bb8c189413>.
- [5] QIAN P, LIU Z, HE Q, et al. Smart contract vulnerability detection technique: a survey [DB/OL]. <https://arxiv.org/abs/2209.05872>, 2022.
- [6] 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, 2019: 1186–1189.
- [7] DHAWAN M. Analyzing safety of smart contracts [C]//Proceedings of the Conference: Network and Distributed System Security Symposium, 2017: 16–17.
- [8] 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.
- [9] JIANG B, LIU Y, CHAN W K. Contractfuzzer: fuzzing smart contracts for vulnerability detection [C]//Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018: 259–269.
- [10] LIPTON Z C. A critical review of recurrent neural networks for sequence learning [DB/OL]. <https://arxiv.org/abs/1506.00019>, 2015.
- [11] WU Z, PAN S, CHEN F, et al. A comprehensive survey on graph neural networks [J]. *IEEE Transactions on Neural Networks and Learning Systems*, 2020, 32(1): 4–24.
- [12] STAUEMEYER R C, MORRIS E R. Understanding LSTM—a tutorial into long short-term memory recurrent neural networks [DB/OL]. <https://arxiv.org/abs/1909.09586>, 2019.
- [13] DEY R, SALEM F M. Gate-variants of gated recurrent unit (GRU) neural networks [C]//2017 IEEE 60th International Midwest Symposium on Circuits and Systems, 2017: 1597–1600.
- [14] SHERSTINSKY A. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network [J]. *Physica D: Nonlinear Phenomena*, 2020, 404: 132306.
- [15] VASWANI A. Attention is all you need [DB/OL]. <https://arxiv.org/abs/1706.03762>, 2017.
- [16] ANTLR [EB/OL]. <http://www.antlr.org/>.
- [17] GARFATTA I, KLAI K, GAALOUL W, et al. A survey on formal verification for solidity smart contracts [C]//Proceedings of the 2021 Australasian Computer Science Week Multiconference, 2021: 1–10.
- [18] TSANKOV P, DAN A, DRACHSLER C 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.
- [19] Mythril: an open-source security analysis tool for ethereum smart contracts [EB/OL]. <https://github.com/ConsenSys/mythril>.
- [20] KLEES G, RUEF A, COOPER B, et al. Evaluating fuzz testing [C]//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018: 2123–2138.
- [21] 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, 2019: 8–15.
- [22] GUPTA R, PATEL M M, SHUKLA A, et al. Deep learning-based malicious smart contract detection scheme for internet of things environment [J]. *Computers & Electrical Engineering*, 2022, 97: 107583.
- [23] QIAN P, LIU Z, HE Q, et al. Towards automated reentrancy detection for smart contracts based on sequential models [J]. *IEEE Access*, 2020, 8: 19685–19695.
- [24] HAO X, REN W, ZHENG W, et al. SCSscan: a SVM-based scanning system for vulnerabilities in blockchain smart contracts [C]//2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications, 2020: 1598–1605.
- [25] HUANG H, GUO L, ZHAO L, et al. Effective combining source code and opcode for accurate vulnerability detection of smart contracts in edge AI systems [J]. *Applied Soft Computing*, 2024, 158: 111556.
- [26] ASHIZAWA N, YANAI N, CRUZ J P, et al. Eth2Vec: learning contract-wide code representations for vulnerability detection on ethereum smart contracts [C]//Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure, 2021: 47–59.
- [27] ZHANG H, ZHANG W, FENG Y, et al. SVScanner: detecting smart contract vulnerabilities via deep semantic extraction [J]. *Journal of Information Security and Applications*, 2023, 75: 103484.
- [28] 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.
- [29] HUANG T T H D. Hunting the ethereum smart contract: color-inspired inspection of potential attacks [DB/OL]. <https://arxiv.org/abs/1807.01868>, 2018.
- [30] ZHEN Z, ZHAO X, ZHANG J, et al. DA-GNN: a smart contract vulnerability detection method based on dual attention graph neural network [J]. *Computer Networks*, 2024, 242: 110238.
- [31] CAI J, LI B, ZHANG J, et al. Combine sliced joint graph with graph neural networks for smart contract vulnerability detection [J]. *Journal of Systems and Software*, 2023, 195: 111550.
- [32] LIU H, FAN Y, FENG L, et al. Vulnerable smart contract function locating based on multi-relational nested graph convolutional network [J]. *Journal of Systems and Software*, 2023, 204: 111775.
- [33] YU X, ZHAO H, HOU B, et al. DeeSCVHunter: a deep learning-based framework for smart contract vulnerability detection [C]//2021 International Joint Conference on Neural Networks (IJCNN), 2021: 1–8.

(责任编辑:孙娟)