文章编号:1671-9352(2026)03-0020-09 DOI:10.6040/j.issn.1671-9352.9.2025.001

基于漏洞子树的链码漏洞检测方法

林思怡[1,2],宋甫元[1,2*],付章杰[1,2]
(1.南京信息工程大学计算机学院网络空间安全学院, 江苏 南京 210044; 2.南京信息工程大学数字取证教育部工程研究中心, 江苏 南京 210044)

摘要:针对联盟链超级账本(Hyperledger Fabric)中链码的安全漏洞问题,提出了一种基于漏洞子树和预训练模型的深度学习漏洞检测网络。检测方法包括 2 个关键阶段:首先,通过自动化工具提取链码为抽象语法树,并设计了漏洞子树结构 VB-tree,确保模型专注于关键漏洞特征,在此基础上根据程序语句之间的数据和控制依赖关系转化为数据流图;其次,利用预训练模型对提取的特征进行处理,准确识别潜在漏洞。最后,从 Github 收集了 6 935 个不同领域开源项目的链码构建可用于评估方法有效性的数据集。实验结果表明,在检测链码中的 21 种漏洞时,模型的平均 *F*1 分数为 93.68%,优于现有的方法。

关键词:区块链;智能合约;漏洞检测
中图分类号:TP309 **文献标志码**:A
引用格式:林思怡,宋甫元,付章杰. 基于漏洞子树的链码漏洞检测方法[J]. 山东大学学报(理学版),2026,61(3):20-28,43.

Chaincode vulnerability detection method based on pre-training model

LIN Siyi[1,2], SONG Fuyuan[1,2*], FU Zhangjie[1,2]
(1. School of Computer Science, School of Cyber Science and Engineering, Nanjing University of Information Science & Technology, Nanjing 210044, Jiangsu, China; 2. Engineering Research Center of Digital Forensics Ministry of Education, Nanjing University of Information Science & Technology, Nanjing 210044, Jiangsu, China)

Abstract: Aiming at the problem of security vulnerabilities in chain codes in the consortium chain Hyperledger Fabric, a deep learning vulnerability detection network based on vulnerability subtrees and pre-trained models is proposed. The detection method includes two key stages: first, the chain code is extracted into an abstract syntax tree through an automated tool, and a vulnerability subtree structure VB-tree is designed to ensure that the model focuses on key vulnerability features. On this basis, it is converted into a data flow graph based on the data and control dependencies between program statements; second, the extracted features are processed using a pre-trained model to accurately identify potential vulnerabilities. Finally, chain codes of 6 935 open source projects in different fields are collected from Github to construct a dataset that can be used to evaluate the effectiveness of the method. Experimental results show that when detecting 21 types of vulnerabilities in chain codes, the average *F*1 score of the model is 93.68%, which is better than existing methods.

Key words: blockchain; smart contract; vulnerability detection

0 引言

区块链(Blockchain)被定义为有效执行计算机交易协议的合同条款[1],由所有节点遵循共识机制共同维护,交易记录一旦上链,即具备去中心化和不可篡改的性质。凭借这一特性,区块链展现出改变传统行业模式的潜力,受到全球广泛关注,并在金融、管理、医疗、物联网、供应链等领域得到深入应用[2-5]。Buterin[6]受比特币的启发,提出了基于区块链的开源分布式计算平台——以太坊。以太坊的推出为智能合约技术的发展奠定了基础,推动了以太坊等平台主导的新时代的到来[7-8]。智能合约[6]是一种运行在区块链上的自

收稿日期:2025-01-06; **网络出版时间**:2026-01-19
第一作者:林思怡(2000—),女,硕士研究生,研究方向为区块链. E-mail:2245754476@qq.com
*通信作者:宋甫元(1991—),男,讲师,硕士生导师,博士,研究生方向为隐私保护. E-mail:fysong@nuist.edu.cn

动化计算机程序,通过预先定义的规则和条件来管理、验证或执行合约的交易或协议,能够灵活嵌入各种资产和数据,实现高效的价值转移、资产管理和信息交换。目前,数以万计的智能合约已部署在各种区块链平台上,且数量仍在不断增长。

与传统的应用程序不同,智能合约是无法更改的,由代码控制并自动运行的程序一旦执行任何一方都不得否认或撤销,确保了执行过程的可信赖性。由于,智能合约在开发中可能存在安全漏洞,部署到公有链上后,更容易成为攻击目标,其管理的数字代币资产可能因此被窃取,造成巨大的经济损失[9]。近年来由于智能合约漏洞引起的安全事件屡见不鲜,2016 年,DAO 项目遭到黑客攻击,利用项目中的重入漏洞窃取了当时价值约 6 000 万美元的以太币[9];2017 年,Parity 签名钱包使用的库合约漏洞被攻击,导致近 3 000 万美元的以太币被冻结。2021 年,黑客利用漏洞盗取了当时价值约 6 000 万美元的加密货币。因此,智能合约安全漏洞检测已经成为区块链技术中亟待解决的问题。目前,智能合约的漏洞检测方法主要集中在以太坊,包括基于符号执行的方法[10-11],基于形式化的方法[12-13],基于模糊测试的方法[14-15]和基于深度学习的方法[16-17]。

Hyperledger fabric(HF)[18]是目前最流行的开源联盟区块链平台,只有经过授权许可的参与者才能加入网络,共识机制采用拜占庭容错算法或权威投票。对于贸易金融、政府公共服务和供应链管理等应用场景来说,HF 更适合作为智能合约的运行平台。HF 的智能合约也称为链码,可以用通用编程语言来编写,包括 Go、Java 和 Node.js,其中 Go 是 HF 中最常用的链码编写语言,也是本文检测方法研究对象。相较于现有的链码漏洞检测研究,本文在以下几个方面具有显著创新:首先,针对传统方法难以覆盖新型漏洞、漏报误报率高的问题,使用预训练模型进行漏洞检测,摆脱了对人工定义漏洞模式的依赖,具备更强的泛化能力和适应性;其次,为提升模型对漏洞关键特征的聚焦能力,设计了漏洞子树结构,相较于传统抽象语法树,该结构更加简洁,去除了冗余的深层节点,有效提高了检测效率与准确性;最后,本文所用方法在处理复杂、大型链码时展现出更高的性能与资源利用效率,为链码漏洞检测的智能化、自动化提供了新的解决思路和技术路径。

1　相关工作

1.1　预训练模型

随着深度学习(deep learning)的快速发展,诸如 ELMo[19]、GPT[20]和 BERT[21]预训练模型已经成为自然语言处理领域的一种革命性技术。该类模型最早应用于自然语言处理(natural language processing, NLP)领域,通过在大规模无监督文本语料库上进行预训练,学习语言的内在结构和模式,在应用时针对特定的下游任务对模型进行微调。然而,直接使用源代码进行预训练会忽略代码的固有结构,从而损失代码中重要的语义信息,不利于代码的理解过程和进一步的应用。相比于之前的方法,GraphCodeBERT[22]是利用代码结构来学习代码的预训练模型,Peculiar[23]将其运用于智能合约漏洞检测,检测准确率较高。

1.2　Hyperledger Fabric 和链码

HF 在功能上消除了依赖匿名矿工验证交易和使用相关货币激励的需求。HF 中的所有参与者在进行区块链交易之前都必须经过身份验证,确保网络中每个参与者的身份可追溯。由于 HF 不需要昂贵的挖矿计算交易,因此能够以更短的延迟扩展区块链网络。在 HF 平台上,链码承担着智能合约的角色,负责操作账本中的键值对数据,执行复杂的业务逻辑,并支持数据的读写操作。这种灵活的链码机制使得 HF 能够满足不同业务需求的智能合约执行,还使 HF 适用于广泛的企业级应用场景[24-25]。

目前针对链码漏洞检测的工具较少。ChainCode Scanner[26]是由 ChainSecurity 开发的静态安全分析工具,只要利用控制流图分析和依赖图分析。另外,Lv 等[27]提出的静态分析方法结合了抽象语法树分析、包依赖分析和功能依赖分析,较为全面地检测出潜在的风险,比 ChainCode Scanner 具有更高的覆盖率和更好的性能。Fujitsu 等[28]则将链码转换为抽象语法树并遍历,以识别潜在的安全风险,并输出所有检测到的风险及其类别。然而,这些静态分析方法在处理复杂逻辑或特定上下文时,可能存在误报和漏报问题,尤其面对高度复杂性和交互性的链码时,准确率和效率可能会受到限制。为解决该问题,文献[29]提出的 HFCCT 使用动态符号执行和静态抽象语法树分析技术相结合的链码漏洞检测框架,可以检测 15 种漏洞,经人工检

测,精确率达91%。但是根据测试结果,该方法会存在误报漏洞类型,数据流图对于相同的源代码在不同的抽象语法下是相同的,并且无法检测到所有类型的漏洞。另外,Xu等[30]提出的CCDetector基于知识图谱的方法对漏洞进行模式匹配,可以检测更多的漏洞类型,具有更好的可扩展性和通用性,但仍存在误报或漏报等问题。总体而言,链码漏洞检测技术,需要更多的研究和实践以提升准确性和覆盖范围,特别是在处理复杂性和安全性要求较高的场景中。与传统的机器学习技术相比,深度学习可以消除专家手动定义特征的需要,自动从复杂数据中提取有用的特征信息,并发现不可知的漏洞。大量研究表明,该技术可显著提高检测精度,同时降低漏洞检测的误报率。

2 链码漏洞

HF链码使用Golang、Java和Node.js等高级语言编写,本文主要针对Golang链码进行检测。目前,针对Golang链码漏洞的研究有限。在之前工作的基础上[27-29],从中总结了21种Golang链码的漏洞类型及其表现形式,并根据原因将其分为3类:非确定性漏洞、HF平台漏洞和常见作法引起的漏洞。

2.1 非确定性漏洞

非确定性漏洞主要包括黑名单引入漏洞、程序并发性漏洞、外部文件访问漏洞、外部库调用漏洞、全局变量漏洞、随机数生成漏洞、映射结构迭代漏洞、具体化对象地址漏洞、系统命令执行漏洞、系统时间戳漏洞和Web服务漏洞。

(1) 黑名单引入:智能合约中使用的一些函数可能会导致非确定性,比如随机数生成函数。因此,可以从阻止导入相关包的层面上防止使用相关函数。当前的参考库黑名单包含'net','os','time'和'math/rand'。

(2) 程序并发性:开发Golang链码时,可使用goroutine和channel实现并发。若未适当处理并发程序,可能导致条件冲突,造成链码执行顺序和行为非确定性。

(3) 外部文件访问:Golang支持访问外部文件,但由于不同节点具有不同的执行逻辑或不确定的数据源,因此无法保证通过访问区块链外部资源获得结果的一致性。

(4) 外部库调用:开发人员在使用第三方库以减少开发工作量时,需关注库的潜在缺陷,保证其安全性。

(5) 全局变量:由于全局变量的作用范围仅限定在单个对等节点内,当不同对等节点未执行所有交易时,可能导致全局变量的状态出现分歧,导致对等节点的账本状态不一致。

(6) 随机数生成:每个节点在独立运行时随机生成的结果不一致,特别是在链码背书节点由每个对等节点独立模拟时,使用随机数在多个背书节点中会导致不一致性。

(7) 映射结构迭代:在Golang中,使用"range"关键字可以遍历各种数据结构中的每个元素。然而,在应用到映射结构时,遍历顺序会被随机化,键值对的顺序并不是固定的。

(8) 具体化对象地址:在不同的环境中,变量内存地址存储的内容可能会有所不同,使用内存地址会引入一定的非确定性。

(9) 系统命令执行:在链码开发中,Golang可以通过'OS/exec'包执行外部命令,当该包无法保证每个对等节点都可以得到相同的命令结果时,应谨慎使用。

(10) 系统时间戳:由于节点的独立性,很难保证时间戳函数在同一时间被调用并获得相同的结果。与随机数生成漏洞类似。

(11) Web服务:处理来自区块链外部的信息时,开发人员需要访问Oracle实体,以获取智能合约所需的外部信息。如果该服务向每个对等节点返回不一致的结果,可能导致分类账出现不一致。

2.2 HF平台漏洞

HF提供的SDK具有一些功能,开发者应小心避免引入一些漏洞,主要包括范围查询风险漏洞、字段声明漏洞、跨通道调用漏洞和读写冲突漏洞等。

(1) 范围查询风险:在HF中,一些范围读取数据库函数存在幻读风险。这些函数包括访问Fabric状态数据库和获取私有数据(例如键的历史记录和状态)的方法,在背书阶段执行后不会在验证阶段重新执行,

这会导致无法检测到可能存在的幻象问题。

(2) 字段声明:在使用 Golang 开发 HF 链码时,需定义结构体并在 Init 和 Invoke 方法中访问字段,由于每个节点的背书政策可能不同,链码结构体中的字段值可能会不一致。为了确保链码的正确性和可预测性,建议在链码结构体中避免声明可变状态字段,以防止潜在的数据不一致问题。

(3) 跨通道调用:Hyperledger 提供了 InvokeChaincode 函数,用于在通道之间调用链码,但仅支持调用读取数据功能,无法进行交易创建或账本修改。在跨通道调用时,应确保只接收被调用链码方法返回的数据,以避免非预期数据提交。

(4) 读写冲突:HF 不支持读写一致性,即使在同一事务中,键值在读取之前被更新,也会返回更新之前的值,从而可能导致代码执行结果与预期不同。

2.3 常见做法漏洞

常见做法漏洞主要包括除零、操作类型不匹配、通道封闭、无法访问的代码、未经检查的输入参数和未处理的错误。

(1) 除零:在大多数编程语言中,除以零是会导致运行时错误或产生未定义的行为,链码也不例外。

(2) 操作类型不匹配:Go 是一种强类型编程语言,操作只能在相同类型的元素上执行。如果在不同类型的元素之间进行操作会导致类型错误。

(3) 通道封闭:在 Golang 语言中,向已关闭的通道发送数据是一个不允许的操作,如果向一个已关闭的通道发送数据,将会导致异常,导致程序终止。

(4) 无法访问的代码:如果一个过程由于某个语句出错而中断,那么在该语句之后的所有语句都将成为无法到达的代码。

(5) 未经检查的输入参数:开发人员在调用函数并传入参数之前应该检查参数,如果参数中包含非法元素,函数调用将失败。

(6) 未处理的错误:在函数返回错误值的情况下,开发人员应该创建变量来接收错误并适当处理它们。然而,为了加快开发速度和简化代码,开发人员在开发过程中经常会忽略函数返回的错误。这导致程序运行时生成的错误未被暴露,也无法找到和定位问题。

3 研究方法

本文所用研究方法包含:特征提取阶段和漏洞检测阶段。在特征提取阶段,漏洞检测方法通过分析智能合约的源代码,从中提取到关键信息和特征。在漏洞检测阶段,使用提取到的特征,漏洞检测方法利用各种技术和模型来识别可能存在的漏洞。检测方法工作流程由 2 个阶段组成:(1)图形生成阶段,从源代码转换的抽象语法树中提取数据流图;(2)漏洞检测阶段,以 GraphCodeBERT 为基础,载入其预训练参数,在搭建好的链码数据集上进行微调,具体流程如图所示,接下来将分别介绍这 2 个阶段。

3.1 图生成阶段

首先,将链码源代码提取成抽象语法树(abstract syntax tree, AST),针对源代码使用标准工具 Tree-sitter[31] 解析生成 AST,其中每个节点代表源代码中的语法结构(如表达式、语句或声明)。由于目标检测主要涉及用 Go 编写的链码,因此使用 go-tree-sitter 实现该过程,生成的 AST 提供了代码的完整语法信息。为了更精准地关注 AST 中的漏洞特征,将其进一步简化为 VB-tree 的新树形结构。该简化过程通过定义与漏洞相关的特定关键字来实现,该方法不仅提高了漏洞检测的效率和准确率,还使分析过程更加直观,在简化结构中,终端(叶子节点)被表示为变量序列。

如图 1 所示,简单的 Go 源代码最开始被解析为 AST,保留了程序的语法结构。假设 add 函数是潜在漏洞的来源,其风险可能来自第 4 行的调用。在第 9 行的 main 函数调用 add 来计算总和,若在此调用中引入外部函数或不安全操作,可能会导致安全漏洞。然而,main 函数中的条件语句并未直接与 add 函数交互,因此对检测漏洞的实用性有限。尽管完整的 AST 为 main 函数提供了完整的数据流信息,但也包含了增加复杂性的无关细节。相比之下,简化的 VB-tree 专注于 add 函数,大大减少了节点和边的数量。通过消除不必要的信息并专注于与已识别漏洞相关的关键组件,这种减少提高了分析效率。

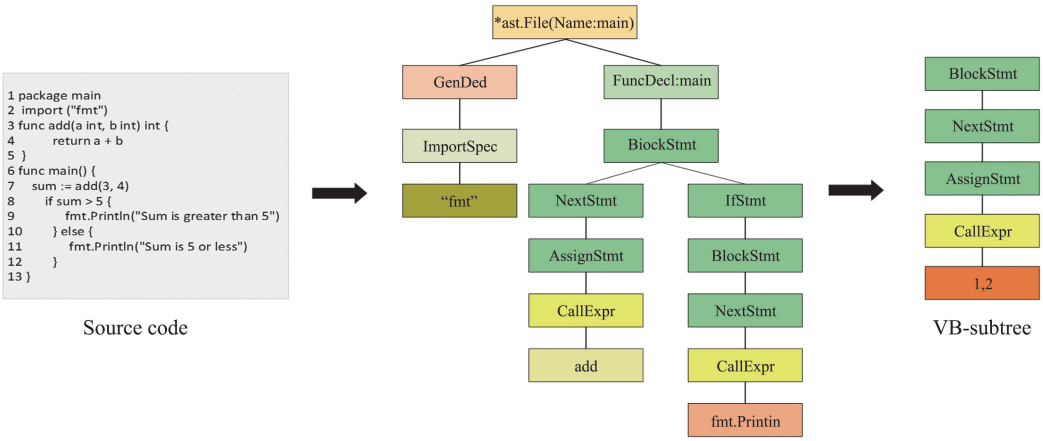

图 1 VB-tree 构建原理

Fig.1 VB-tree construction principle

将包含链码节点信息的抽象语法树转换为数据流图。将每个变量视为图的一个节点,从 V_i 到 V_j 的直接边 $E=\langle V_i,V_j\rangle$ 表示第 j 个变量的值来自第 i 个变量或由第 i 个变量计算得出,并将有向边集合表示为 $E=\{1,2,\cdots,i\}$,图 $G(C)=(V,E)$ 是用于表示源代码 C 的变量之间的依赖关系的数据流。

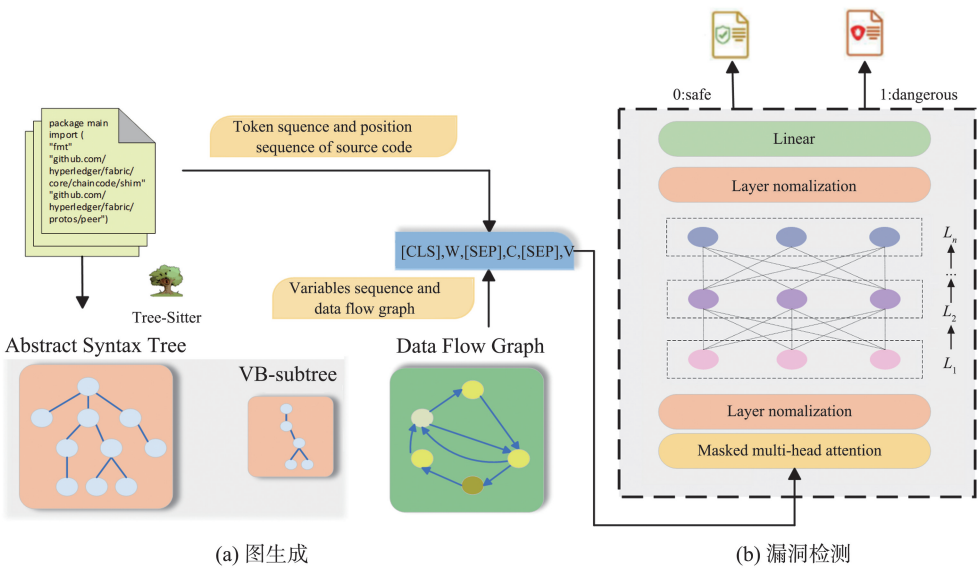

图 2 模型架构

Fig.2 Model architecture

3.2 漏洞检测

基于预训练模型搭建模型架构和进行训练过程,根据 21 种不同漏洞特征对模型进行预训练,从而增加可检测漏洞类型的数量。使用基于编程语言的 Transformer 神经架构 GraphCodeBERT 作为模型主干,并使用线性层来输出结果,在此基础上对其进行修改,使其能够执行漏洞检测任务。将数据流图作为模型的输入,具体地,根据源代码 $C=\{c_1,c_2,\cdots,c_n\}$ 获得相应的数据流图 $G=\{V,E\}$,其中 $V=\{v_1,v_2,\cdots,v_k\}$ 是一个变量集,$E=\{\varepsilon_1,\varepsilon_2,\cdots,\varepsilon_j\}$ 是指每个变量的值来自何处的向量边集合。最终将源代码和变量集连接到一个序列 $I=\{[\mathrm{CLS}],C,[\mathrm{SEP}],V\}$ 输入到模型中,其中[CLS]是 2 个集合前面的特殊标记,[SEP]是用于分割源代码 CC 和量集合 Var 的特殊符号。

在被放入模型之后,序列 I 被转换为输入向量 $\boldsymbol{X}_0$,经过模型的 $N(N=12)$ 个 Transformer 层以生成上下文表示,即 $\boldsymbol{X}^n=\mathrm{transformer}(\boldsymbol{X}^{n-1})$, $n\in[1,N]$,其中每个 Transformer 层包括结构上等效的 Transformer,并且向量 $\boldsymbol{X}^{n-1}$将首先在多头自注意操作生成向量 $\boldsymbol{H}^n$,然后在前馈层输出向量 $\boldsymbol{X}^n$。

$$\boldsymbol{H}^n=\mathrm{LN}(\mathrm{MHSA}(\boldsymbol{X}^{n-1})+\boldsymbol{X}^{n-1}),\tag{1}$$

$$\boldsymbol{X}^n=\mathrm{LN}(\mathrm{FNN}(\boldsymbol{H}^n)+\boldsymbol{H}^n),\tag{2}$$

其中,MHSA 是多头自注意机制,FNN 是两层前馈网络,LN 表示层归一化操作。对于第 n 个 Transformer

层,多头自注意力的输出被计算为:

$$Q_i = \boldsymbol{X}^{n-1}\boldsymbol{W}_i^Q,\ K_i = X^{n-1}W_i^K,\ V_i = X^{n-1}W_i^V, \tag{3}$$

$$\text{head}_i = \text{softmax}\left(\frac{Q_i\boldsymbol{K}_i^{\mathrm{T}}}{\sqrt{d_k}} + \boldsymbol{M}\right)\boldsymbol{V}_i, \tag{4}$$

$$\hat{\boldsymbol{X}}^n = [\text{head}_1, \cdots, \text{head}_m]W_n^O, \tag{5}$$

其中,前一层的输出$\boldsymbol{X}^{n-1} \in \mathbf{R}^{|I|\times d_h}$分别使用,模型参数$W_i^Q$, W_i^K, $W_i^V \in \mathbf{R}^{d_h\times d_k}$线性投影到由查询、键和值组成的三元组上,$m$ 是头的数量,d_k 是头的尺寸。$W_n^Q \in \mathbf{R}^{d_h\times d_h}$是模型参数,$\boldsymbol{M} \in \mathbf{R}^{|I|\times|I|}$是一个掩码矩阵,其中如果允许第 i 个令牌参与第 j 个令牌,则 $\boldsymbol{M}_{ij}=0$,否则为 $\boldsymbol{M}_{ij}=\infty$。

在模型的最后添加一个线性分类器,并使用 softmax 函数来输出预测概率

$$\hat{y} = \text{Sigmoid}(\hat{X}^n)。 \tag{6}$$

为了将图结构引入 Transformer 并表示变量之间的依赖关系,遵循 GraphCodeBERT 使用图引导的掩蔽注意力函数来建模标记关系。

本文使用掩码矩阵 $\boldsymbol{M}$ 表示图引导掩蔽注意力机制:

$$\boldsymbol{M}_{ij} = \begin{cases} 0, & \text{if query}_i \in \{[\text{CLS}],[\text{SEP}]\},\ \text{or query}_i,\ \text{key}_j \in C,\ \text{or}\ \langle \text{query}_i, \text{key}_j\rangle \in \text{Edge}\cup\text{Edge}', \\ -\infty, & \text{otherwise}。 \end{cases} \tag{7}$$

其中,$\text{query}_{\text{var}_i}$表示节点的 var_i 的查询,$\text{key}_{\text{var}_i}$是节点 var_i 的节点键。Edge′是代表源代码标记和数据流节点之间关系的集合,其中$\langle \text{var}_i, \text{code}_j\rangle/\langle \text{code}_j, \text{var}_i\rangle$,如果变量 var_i 是从源代码标记 code_j 中识别出来的。如果节点 var_i 和节点 var_j 之间存在直接边(即$\langle \text{var}_i, \text{var}_j\rangle \in \text{Edge}'$)或者它们是同一节点(即 $i=j$),则允许节点查询 $\text{query}_{\text{var}_i}$关注节点 $\text{key}_{\text{code}_i}$。否则,注意力将被屏蔽,将注意力分数添加到无限大的负值。通过向注意力得分查询 $\text{query}_j^{\mathrm{T}}\text{key}_i$ 添加无限负值,使用 softmax 函数后注意力权重变为零,那么注意力屏蔽函数就可以避免查询查询 query_j 所涉及的键 key_i。此外,当且仅当$\langle \text{var}_i, \text{code}_j\rangle/\langle \text{code}_j, \text{var}_i\rangle \in \text{Edge}'$时,允许节点 $\text{query}_{\text{var}_i}$和代码 $\text{key}_{\text{code}_i}$相互参与。

4　实验

实验环境采用 Ubuntu18.04 系统,64 G 内存,i7-9700 CPU 和 NVIDIA 4090ti 显卡,并使用基于 Pytorch 的深度学习框架训练和测试。使用交叉熵损失函数计算损失,采用 AdamW 梯度下降优化算法进行优化,并在测试集上对模型进行评估。此外,遵循 GraphCodeBERT 采用 12 层 Transformer 结构,其隐藏层维度为 768,自注意力头数为 12,整体参数规模达到 1.25 亿,以确保模型在代码理解任务中的高效训练和优化。

4.1　数据集

由于当前缺乏公开的链码漏洞数据集,本研究方法通过数据收集与漏洞注入策略构建了数据集。首先,从 GitHub 收集了 6 932 个涉及不同应用领域的 HF 链码,并手动标注了 21 种漏洞标签。为解决正负样本不平衡的问题,采用漏洞注入方法向部分链码中加入漏洞代码片段。例如,对于黑名单引入漏洞,将该漏洞注入到部分链码中,并将其标签设为“1”用于模型训练,其他链码则标记为“0”,利用该数据集评估所提方法的检测有效性和准确性。为了合理评估模型性能,按照 80%、10%和 10%的比例划分为训练集、验证集和测试集,其中训练集包含 5 546 个样本用于模型学习特征;验证集包含约 693 个样本,用于调整超参数和防止过拟合;测试集同样包含约 693 个样本,用于最终评估模型的泛化能力。

本文选定 3 个指标来评估检测能力:精确率(Precision)、召回率(Recall)和 $F1$。指标值越高,表明检测工具的性能较好。

(1)精确率(Precision)

精确率是衡量模型在所有被预测为正样本中,实际上有多少真正的正样本,计算公式为

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}, \tag{8}$$

(2)召回率(Recall)

召回率是衡量模型在所有实际为正样本中,成功预测为正样本的比例,计算公式为

$$\text{Recall}=\frac{\text{TP}}{\text{TP}+\text{FN}},\tag{9}$$

(3)$F1$

$F1$ 综合考虑精确率和召回率 2 个指标,常被用作一些分类问题的最终评估指标,计算公式为

$$F1=2\times\frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}},\tag{10}$$

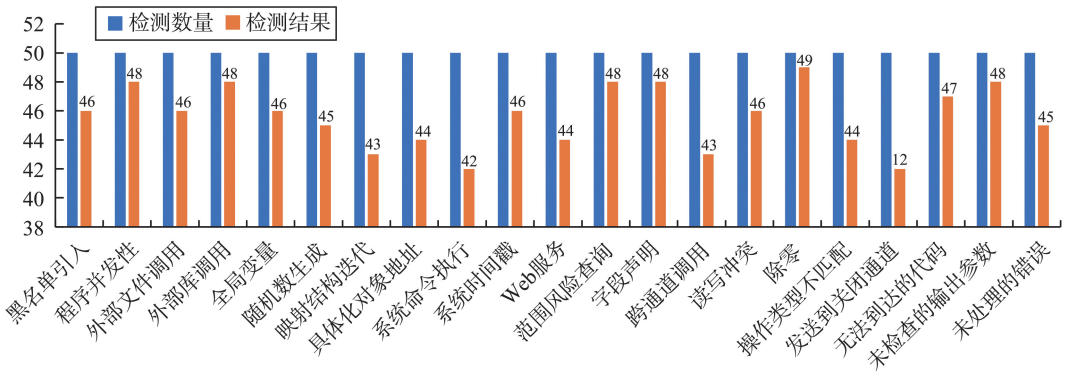

图 3 检测数量对比

Fig.3 Comparison of detection quantity

4.2 有效性实验

为了验证本研究方法的有效性,实验从数据集中随机选取 50 个链码,并在每个链码中注入相同类型的漏洞,即第 2 部分介绍的 21 种链码漏洞。然后,使用训练好的漏洞检测模型进行分析,并统计成功检测到漏洞的链码数量,即预测结果为 1 的样本数量。最终,通过计算检测准确率,并分析误报和漏报情况,评估模型在已知漏洞上的检测能力及其在不同链码结构中的适用性,检测结果如图 3 所示。

4.3 准确性实验

为评估所提 VB-tree 结构的准确性,在相同条件下对比了使用和不使用 VB-tree 的检测结果,结果如表 1 所示。

表 1 非确定性漏洞检测结果

Table 1 Non-deterministic vulnerability detection results

漏洞类型	Precision/%		Recall/%		$F1$/%	
	w/ VB-tree	w/o VB-tree	w/ VB-tree	w/o VB-tree	w/ VB-tree	w/o VB-tree
黑名单引入	93.83	94.31	91.62	92.46	92.71	93.38
程序并发性	95.42	95.28	93.17	90.34	94.28	92.74
外部文件调用	97.35	92.37	95.21	94.81	96.26	93.57
外部库调用	95.33	94.80	93.12	91.25	94.21	92.99
全局变量	92.30	93.10	93.81	90.27	93.05	91.66
随机数生成	95.64	92.63	93.15	92.60	94.37	92.61
映射结构迭代	95.91	91.32	92.65	90.58	94.25	90.95
具体化对象地址	95.08	93.96	91.27	90.25	93.13	92.06
系统命令执行	94.35	91.37	92.67	91.09	93.50	91.22
系统时间戳	97.41	96.87	94.60	90.43	95.98	93.54
Web 服务	98.32	96.25	91.85	90.31	94.97	93.19
范围查询风险	94.27	93.61	91.06	90.35	92.63	91.95
字段声明	93.51	94.39	94.29	92.54	93.89	93.46
跨通道调用	91.60	92.25	92.34	92.60	91.96	92.42
读写冲突	95.48	93.46	92.11	91.62	93.93	92.53
除零	93.26	91.94	92.37	94.83	95.02	95.31
操作类型不匹配	92.61	96.33	94.03	93.65	92.56	92.95
发送到关闭通道	91.70	97.36	95.20	92.90	94.22	93.65
无法到达的代码	–	93.46	–	91.62	–	92.53
未检查的输入参数	93.60	91.50	92.51	91.35	93.05	91.37
未处理的错误	–	90.06	–	91.75	–	90.89

注:其中“w/”表示使用,“w/o”表示未使用。

具体来说,在不使用 VB-tree 结构时,检测准确率最高的漏洞为系统时间戳漏洞,精确率达 96.87%,相比最优结果提升了 5.87%;而在使用 VB-tree 结构时,检测准确率最高的漏洞为 Web 服务漏洞,精确率达到 98.32%,平均准确率为 95.53%。此外,通过 VB-tree 与预训练模型的结合,本研究方法的平均 *F*1 达94.26%。

与特定函数或库引入的非确定性漏洞不同,HF 平台漏洞和常见实践漏洞的成因可能涉及特定结构的代码,因此在提取 VB-tree 时适当放宽了约束。本研究方法发在准确率、召回率和 *F*1 方面表现良好,*F*1 最高达 95.02%。需要指出的是,无法到达的代码漏洞和未处理错误漏洞可能以多种形式存在,因此 VB-tree 的提取在此类漏洞检测中的效果可能受限。但整体表现较好。

4.4　与其他方法的对比

如表 2 所示,在可检测漏洞数量方面,现有的链码漏洞检测工具包括 Chaincode Scanner、Fujitsu、Lv's Tool 和 HFCCT,所提方法可检测的漏洞类型明显多于所有现有工具。具体来说,Chaincode Scanner 可检测 12 种,无法检测诸如外部文件调用、外部库调用、Web 服务等漏洞;Fujitsu 和 HFCCT 在外部依赖和跨通道调用方面有所增强,但仍然无法检测某些特定漏洞,如除零、操作类型不匹配、发送到关闭通道、无法到达的代码等而本研究方法可以检测 21 种,显著超过现有工具。

表 2　不同工具可检测链码种类

Table 2　Chaincode typees that can be detected by different tools

漏洞类型	Chaincode Scanner	Fujitsu	HFCCT	本文研究方法
黑名单引入	√			√
程序并发性	√	√	√	√
外部文件调用		√	√	√
外部库调用		√	√	√
字段声明	√		√	√
全局变量	√	√	√	√
随机数生成	√	√	√	√
映射结构迭代	√	√	√	√
集体的对象地址	√	√		√
系统命令执行	√	√	√	√
系统时间戳	√	√	√	√
Web 服务		√	√	√
除零				√
操作类型不匹配				√
发送到关闭通道				√
无法到达的代码				√
未检查的输入参数	√		√	√
未处理的错误	√		√	√
范围查询风险	√	√	√	√
读写冲突		√	√	√
跨通道链码调用		√		√
未处理的错误			√	√
平均检测准确率	79.6%	83.2%	89.7%	93.68%

从检测准确率来看,本研究方法的平均检测准确率达到 93.68%,高于 HFCCT(89.7%)、Fujitsu(83.2%)和 Chaincode Scanner(79.6%)。这表明本研究方法在保证漏洞类型覆盖范围更广的同时,也具有更高的检测精度,能够有效减少漏报和误报问题。

此外,依托预训练模型的强大表示能力,本研究方法在面对新型漏洞时无需大幅修改即可适应并检测,从而提高了漏洞检测的覆盖范围和适应性。

5　结论

本文提出了一种基于漏洞子树的 Go 链码漏洞检测方法,为链码漏洞检测提供了一种新颖的研究思路,

使用该方法开发了相关工具,并搭建了一个新的数据集来定量评估检测有效性和准确性,本研究方法可检测21 种漏洞,并且 $F1$ 均达到 93.68%以上。目前,HF 项目公开可用的数据集和对链码漏洞类型的研究有限,不利于漏洞检测方法和工具的评估,漏洞类型的总结也并不完善。在未来的研究中,将重点关注数据集和漏洞类型,还将对链码机制进行更深入的研究,并实践链码部署以探索新的漏洞类型。

同时在未来的工作中,将优化特征提取模式,把更多的数据流信息集成到方法中来检测更多的漏洞,并将其应用于其他编程语言编写的智能合约的漏洞检测。

参考文献:

[1] 参考文献:[1] SZABO N. Smart contracts: building blocks for digital markets[J]. EXTROPY, 1996(16):18.

[2] SUN Nan, WANG Wei, TONG Yongxin, et al. Blockchain based federated learning for intrusion detection for Internet of Things[J]. Frontiers of Computer Science, 2024, 18(5):185328.

[3] CHEN Xingxing, CHENG Qingfeng, YANG Weidong, et al. An anonymous authentication and secure data transmission scheme for the Internet of Things based on blockchain[J]. Frontiers of Computer Science, 2024, 18(3):183807.

[4] QU Youyang, MA Lichuan, YE Wenjie, et al. Towards privacy-aware and trustworthy data sharing using blockchain for edge intelligence[J]. Big Data Mining and Analytics, 2023, 6(4):443-464.

[5] ZHANG Xiaofeng, LI Ling. A review of blockchain solutions in supply chain traceability[J]. Tsinghua Science and Technology, 2022, 28(3):500-510.

[6] BUTERIN V. A next-generation smart contract and decentralized application platform[J]. White Paper, 2014, 3(37):2-1.

[7] NAKAMOTO S. Bitcoin: a peer-to-peer electronic cash system[EB/OL]. https://bitcoin.org/en/bitcoin-paper.

[8] DANNEN C. Introducing Ethereum and solidity[M]. Berkeley:Apress, 2017.

[9] DEL C M. The DAO attacked: code issue leads to $60 million ether theft[J]. Saatavissa, 2016, 3:1-4.

[10] LUU L, CHU D H, OLICKEL H, et al. Making smart contracts smarter[C]//2016 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2016:254-269.

[11] TSANKOV P, DAN A, DRACHSLER-COHEN D, et al. Securify: practical security analysis of smart contracts[C]//2018 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2018:67-82.

[12] KALRA S, GOEL S, DHAWAN M, et al. Zeus: analyzing safety of smart contracts[C]//25th Annual Network and Distributed System Security Symposium (NDSS 2018). San Diego: Internet Society, 2018:1-12.

[13] BHARGARAN K, DELIGNAT-LAVAUD A, FOURNET C, et al. Formal verification of smart contracts: short paper[C]//2016 ACM Workshop on Programming Languages and Analysis for Security. New York: ACM, 2016:91-96.

[14] JIANG Bo, LIU Ye, CHAN W K. Contractfuzzer: fuzzing smart contracts for vulnerability detection[C]//2018 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE). New York: ACM, 2018:259-269.

[15] LIU Zhenguang, QIAN Peng, et al. Rethinking smart contract fuzzing: fuzzing with invocation ordering and important branch revisiting[J]. IEEE Transactions on Information Forensics and Security, 2023, 18:1237-1251.

[16] ZHUANG Yuan, LIU Zhenguang, QIAN Peng, et al. Smart contract vulnerability detection using graph neural networks[C]//30th International Joint Conference on Artificial Intelligence (IJCAI-21), 2021:3283-3290.

[17] ZHANG Zhuo, YAN Lei, YAN Meng, et al. Reentrancy vulnerability detection and localization: a deep learning based two-phase approach[C]//2022 37th IEEE/ACM International Conference on Automated Software Engineering (ASE). New York: ACM, 2022:1-13.

[18] CACHIN C. Architecture of the hyperledger blockchain fabric[C]//Workshop on Distributed Cryptocurrencies and Consensus Ledgers. 2016, 310(4):1-4.

[19] PETERS M E, NEUMANN M, IYYER M, et al. Deep contextualized word representations[C]//2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. New Orleans: ACL, 2018:2227-2237.

[20] RADFORD A, NARASIMHAN K, SALIMANS T, et al. Improving language understanding by generative pre-training[EB/OL]. 2018.

[21] DEVLIN J, CHANG M W, LEE K, et al. BERT: pre-training of deep bidirectional transformers for language understanding[C]//2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis: ACL, 2019:4171-4186.

(下转第 43 页)