文章编号:1671-4229(2023)04-0056-10

基于源码语义结构分析的智能合约漏洞检测方法

李 珊,王 斌,王 伟*
(北京交通大学 智能交通数据安全与隐私保护技术北京市重点实验室,北京 100044)

摘 要:近年来,智能合约已广泛应用于去中心化金融、供应链管理等领域。然而,智能合约漏洞给这些领域造成了严重的损失,由于智能合约部署在区块链上以后无法修改,开发人员写好智能合约后必须检测其安全性。现有的基于深度学习的检测方法大多使用字节码和源码,然而,基于字节码的检测方法无法定位到漏洞可能出现的位置且直接检测向量化源码准确率低。为了提高漏洞检测的准确率,增加检测结果的可解释性,文章提出了基于源码语义结构分析的智能合约漏洞检测方法。首先,将智能合约源代码转化为抽象语法树,研究源代码和抽象语法树的语法关系;其次,通过对以太坊中合约的抽象语法树属性特征和漏洞特性进行分析,发现5种漏洞相关属性特征,并围绕这些节点特征将抽象语法树进行切片,得到与漏洞特性相关的子树切片;最后,提取子树切片的结构特征和属性特征,并将其表示为图结构。使用具有更好图表示能力的图同构网络模型检测子树切片的图结构并利用33 812个以太坊上的智能合约进行实验,实验结果表明,文章所提出的方法有效性显著高于其他方法,Macro-F1超过90%,未检查返回值和重入这两种漏洞的F1-score分别达到97%和92%。

关键词:区块链;智能合约;抽象语法树;图同构网络
中图分类号:TP 391 **文献标志码**:A

A smart contract vulnerability detection method based on semantic structure analysis of source code

LI Shan, *WANG Bin*, *WANG Wei**
(Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing 100044, China)

Abstract: In recent years, smart contracts have been increasingly applied in fields such as decentralized finance and supply chain management. However, the vulnerability of smart contracts has caused serious losses to these fields. After the smart contract is deployed on the blockchain, it cannot be modified. Developers must check its security after writing the smart contract. Most of the existing detection methods based on deep learning use bytecode and source code, but bytecode based detection methods cannot locate the potential location of vulnerabilities, and the accuracy of direct vectorization source code detection is low. In order to improve the interpretability and accuracy of vulnerability detection, this paper proposes a smart contract vulnerability detection method based on semantic structure analysis. Firstly, we transform the source code of the smart contract into an abstract syntax tree and study the syntax relationship between the source code and the abstract syntax tree. Secondly, by analyzing the abstract syntax tree attribute features and vulnerability characteristics of real contracts in

收稿日期:2023-05-10;**修回日期**:2023-06-24
基金项目:国家重点研发计划资助项目(2022YFB2702903);国家自然科学基金联合基金重点支持项目(U21A20463,U22B2027)
作者简介:李珊(1999—),女,硕士研究生. E-mail:lishan@bjtu.edu.cn
* 通信作者. E-mail:wangwei1@bjtu.edu.cn
引文格式:李珊,王斌,王伟. 基于源码语义结构分析的智能合约漏洞检测方法[J]. 广州大学学报(自然科学版),2023,22(4):56-65.

Ethereum, we identified five vulnerability related attribute features and slices context around these node features. Finally, we extract the structural and attribute features of subtree slices and represent them as graph structures. We use graph isomorphic network models with better graph representation capabilities to detect these graph structures. We conducted experiments using 33 812 smart contracts on Ethereum. The experimental results indicate that the Macro-F1 of this method exceeds 90%. Specifically, the F1 scores for unchecked return values and re-entry vulnerabilities reached 97% and 92%.

Key words: blockchain; smart contract; abstract syntax tree; graph isomorphic network

区块链是一种分布式存储技术。2008 年,中本聪提出了比特币概念和区块链技术的架构理念[1]。区块链由点对点传输、加密算法和共识机制等多种技术实现,具有去中心化及不可篡改的性质[2]。区块链的本质是一个分布式存储的账本,用户的交易会在区块链网络中传播和验证,最终存储在每个节点上[3]。区块链在金融、游戏、医疗和教育等领域有巨大的应用价值[4-6],但比特币系统只能实现一些简单的计算。2014 年,Buterin[7]提出智能合约可编程平台以太坊,以太坊是一个全世界分布的计算机。以太坊虚拟机(Ethereum Virtual Machine,EVM)能够计算、存储和读取数据,它是图灵完备的[8]。智能合约是一种有状态的计算机程序,不仅能够处理数据,而且能够控制和管理区块链上的资产。具有图灵完备性的 EVM 可以执行多种用途的智能合约。以太坊开发了 EVM 上的智能合约编程语言 Solidity,类似于其他编程语言,Solidity 同样能够用来开发系统和游戏等。去中心化应用(Decentralized Application,DAPP)是以太坊提出的基于底层区块链的各种应用程序,它使用智能合约编程并且具有前端用户界面[9]。以太坊使得区块链可以更好地应用在更多领域。

不同于一般的计算机程序,智能合约一旦部署在区块链上无法修改。智能合约管理着数以亿万美元的资产,并且智能合约在区块链上是公开的。受利益驱动的攻击者一旦发现智能合约存在可利用的漏洞,会想尽办法从中获利[10-11]。例如最经典的 The DAO 事件,由于其智能合约存在重入漏洞而被黑客利用,导致 6 000 多万美元的损失,最终使得以太坊发生硬分叉。The Parity Wallet bug 由多重签名合约的漏洞导致,Parity 是当时以太坊使用最广泛的钱包之一,这次事件导致 15 万枚以太币被盗,造成重大损失[12]。由智能合约漏洞造成的损失逐年增长。合约部署在区块链之前,开发者必须检测智能合约的安全性。

智能合约开发人员安全意识不高、区块链运行机制等都可能导致智能合约存在安全风险。自从多个智能合约安全事件爆发以后,更多学者致力于研究智能合约漏洞检测方法。目前主流的检测方法有符号执行[13]、模糊测试[14]、形式化验证[15-17]和深度学习。然而,符号执行存在执行路径爆炸的问题,测试路径的覆盖率是模糊测试的重要挑战,深度学习的准确率受提取特征的影响,研究者对漏洞特征有不同的提取方式。

本文提出基于源码语义结构分析的智能合约漏洞检测方法,从智能合约源码抽象语法树(Abstract Syntax Tree,AST)中提取图结构数据特征。具体来说,首先,将源代码处理为表示程序语法信息的 AST 树形结构,AST 也能够保留代码的结构信息,因此,可以从 AST 中提取源码的上下文特征和语义特征;其次,通过对以太坊中合约 AST 的属性特征和智能合约漏洞特性的分析,发现与 5 种类型漏洞相关的属性特征,基于这些属性特征提取与漏洞更相关的子树切片,这不仅减少过多冗余信息对分类效果的影响,而且由于 AST 具有较好的可读性,且本方法提取的是与漏洞相关的子树切片,增加了分类结果在源码中的可解释性;最后,使用子树切片数据训练具有更好图表示能力的图同构网络(Graph Isomorphism Network,GIN)模型[18]。GIN 使用神经网络学习图结构特征,从切片中提取表示树结构的邻接矩阵,并且提取节点对应的属性信息,进一步地,需要将节点信息映射到邻接矩阵中生成图结构数据。相较于使用字节码检测智能合约漏洞的深度学习 ResNets 方法,本方法对于整数上溢漏洞、整数下溢漏洞、未检查返回值漏洞和重入漏洞的 F1-score 分别从 85% 提高到 89%,81% 提高到 84%,81% 提高到 97%,以及 70% 提高到 92%。

本文主要有以下贡献:①提出使用源码抽象语法树检测智能合约漏洞,通过分析智能合约漏洞特征与抽象语法树的属性信息,得到关键属性并切片得到关键片段;②为了更好地检测出漏洞,分析关键片段的结构和语义特征,将关键片段表示为由邻接矩阵和节点属性信

息组成的图结构,使用 GIN 检测智能合约漏洞;③可以检测5种类型漏洞,使用以太坊上的合约训练 GIN 模型。实验结果表明,本方法有效性显著高于其他方法,其 Macro-F1 超过 90%,特别是未检查返回值和重入漏洞的 F1-score 达到97%和92%。

1 相关工作

智能合约部署以后,即使发现漏洞也难以修改,因此,在部署前,开发人员必须对合约进行安全检测。目前已经有很多智能合约漏洞检测方法,主要使用字节码和源码检测漏洞[19-20]。智能合约需要编译为字节码部署在 EVM 上,字节码能够表达出智能合约的特征。最早使用字节码分析合约漏洞的方法是符号执行,符号执行在软件测试、程序验证以及漏洞检测等领域中有着非常重要的作用[13]。Oyente[21]和 Mythril[22]是比较具有代表性的方法。Oyente 完全使用符号执行技术,它通过设计漏洞约束条件分析合约安全性;Mythril 采用符号执行和污点分析技术,相较于 Oyente,增加了可检测漏洞类型。然而,如果遇见条件较多的合约,符号执行方法会产生过多的路径,难以求解。现在的合约普遍较长,所以符号执行求解效率较低。

为了提高检测效率,近年来,很多从字节码中提取特征使用机器学习和深度学习方法来训练检测模型。在前期工作中,笔者提出了 ContractWard[23],该方法是从合约字节码中提取 bigram 特征,并从大量样本中学习漏洞特征来训练机器学习模型。它的优点是速度快,但它以两个字节码为一组提取的特征上下文联系较弱。总的来说,基于字节码符号执行的方法能有效检测出漏洞,但需要更长的检测时间。基于字节码的深度学习方法能够快速检测合约漏洞,适用于当前大规模合约检测的场景,但难以定位漏洞出现在源码中的位置,不便于开发人员修改。

以太坊智能合约使用 Solidity 编写,和其他编程语一样,智能合约由合约类、方法及变量等组成。源码的含义容易被人们所理解,因此,从源码中提取特征的智能合约漏洞检测方法便于知道漏洞可能所在的位置。AME[24]是从源码中提取专家模式,通过使用图结构表示提取的数据流和控制流信息,来训练图神经网络分类的模型。DR-GCN 和 TMP[25]设计自定义规则将源代码描述为合约图,但由于源码中的变量、函数名等具有可自定义性,使用自定义的规则可能会漏掉一些关键特征,从而导致较高的漏报率。直接向量化源码,又存在不同表示方法但相同语义的代码转化为不同向量的问题,从而导致检测准确率降低。

抽象语法树是一种源代码的抽象树状表示。源代码经过词法分析和语法分析后,生成一种 json 格式的数据对象。Json 数据结构像一棵完整的树,它的每个节点都是源代码中的一部分。大多数的编程语言都要执行编译为 AST 的过程。AST 能表示出程序源码的语义特征和结构特征,因此,在 C、Java、Javascript 等语言的漏洞检测领域有很多应用。例如 ASTNN[26]将一棵完整的树拆分成多个子树,它设计实现了一种子树编码器,并使用循环神经网络将代码转化为向量。ASTNN 实现了代码克隆检测和自动分类。Dam 等[27]提出一种自动学习用于预测软件代码中漏洞的特征方法,将源代码解析为 AST,训练长短期记忆网络模型。Wu 等[28]提出一种新的基于简化 AST 的图卷积网络(SimAST GCN)来处理自动代码审查任务。

本文将智能合约源码转化为具有结构特征和语义特征的树结构,避免源码中自定义变量影响检测结果。通过分析每种漏洞的关键特征,发现这些特征在抽象语法树的属性中都可以表达出来。因此,切片提取关键特征的上下文信息,将这些信息向量化为图结构信息不仅有效提取了合约的特征,更重要的是,这些切片是合约源码抽象语法树结构的子树,这些子树更具有可读性,提高了深度学习的可解释性。

2 源码语义结构特征提取方法

本节首先介绍智能合约源代码的处理方法,将源码处理得到抽象语法树;其次,通过对智能合约漏洞特征的分析,设计漏洞特征提取规则,将抽象语法树切片得到漏洞相关关键片段;最后,从抽象语法树切片得到的子树结构中提取结构特征和语义特征。

2.1 源码预处理

和其他编程语言相同,智能合约编程语言 Solidity 的函数名、变量名等均可自定义。开发人员有不同的代码习惯,例如命名方式和编程习惯等。如果不将源代码进行预处理,直接将源代码向量化,智能合约源码中相同语义的代码可能会转化为不同向量,导致深度学习模型提取特征精度降低。因此,需要预处理源码,将源码转化为 AST,AST 是编译过程中表示程序语法信息的树形结构,能够很好地保留代码的结构,并且也能体现每个节点的语义信息。

智能合约的程序源码被解析成语句、声明和表达式

等,这些都被表示为树的节点。AST 对源代码语义有清晰的描述,比如“ContractDefinition”表示合约定义,“FunctionDefinition”表示函数定义,“ParameterList”表示参数等。此外,节点的类型、值等都由属性来描述。图 1 和图 2 为一个简短的智能合约转化为 AST 的样例,图 1 为使用 solc 编译器得到 json 格式数据,图 2 展示了从 json 文件中遍历得到形象化的树结构,节点具体的属性信息需要从 json 格式数据中提取。

```
pragma solidity ^0.4.19
contract IntegerMinmal{
   uint public count =1;
   function run(uint256 input) public{
       count-=input;
   }
}
```

图 1　智能合约样例

Fig. 1　Example of smart contract

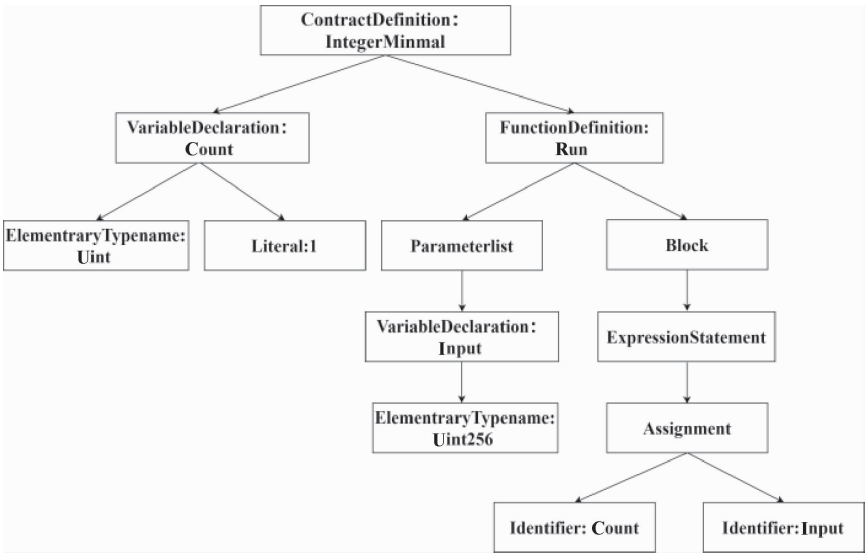


图 2　智能合约样例 AST

Fig. 2　Example of smart contract AST

2.2　合约漏洞特征分析

在实验过程中,如果提取全部的树结构,会产生冗余信息过多的问题。这不仅给向量化带来压力,也影响分类效果。因此,需要研究每种漏洞类型的特征,发现一些与漏洞息息相关的关键字,以关键字为核心,将整个源代码划分为逻辑上可执行的代码片段。一个代码片段由多行代码组成,这些代码彼此之间具有数据依赖性或控制依赖性。表 1 展示了漏洞相关属性特征。

表 1　漏洞相关属性特征

Table 1　Vulnerability key attributes

漏洞类型	关键属性
重入漏洞	function ( ) payable returns (bool)
时间戳依赖	now,timestamp
未检查返回值	transfer,send,call,callcode
整数上溢	operator add,mul
整数下溢	operator sub

(1)重入漏洞

重入是智能合约存在的严重漏洞。智能合约调用被攻击者设计过的外部合约,在 gas 充足的情况下,这些合约可以不断地通过 fallback 函数进入合约,并且如果合约是转账操作,则直接导致财产损失。重入漏洞的典型代码特征是 fallback 函数触发早于状态变量的修改。由于转账函数会触发 fallback 函数,如果攻击者将 fallback 函数重写使其可不断进入取款函数,此时就能够不断转账。重入漏洞造成的严重损失与转账操作密切相关。不同于 transfer( )和 send( )转账时有 gas 限制,当使用 call. value( )转账时,可以使用账户所有的 gas 来运行合约或执行转账操作。智能合约接收以太币必然会触发 fallback 函数,为了接收以太币,fallback 函数会被标记为 payable,在 AST 结构中的属性类型为“function ( ) payable returns (bool)”。

(2)时间戳依赖漏洞

分布式存储的区块链之间需要验证数据的一致性,如果合约中使用随机数会使得多个节点无法共识,因此,区块链不能使用真正的随机数。然而,随机数在编程中经常被使用,智能合约开发人员希望能生成可验证

的伪随机数。时间戳依赖是一种典型的伪随机数造成的漏洞,开发人员希望使用时间戳作为种子生成可验证的随机数。这些随机数往往用于改变合约状态,然而,以太坊规定只要前后两个时间戳差值在900 s就是合法的,矿工可以在这个区间操纵时间戳。因此,时间戳不适合用于生成随机数。时间戳是此漏洞的核心部分,以时间戳作为随机数决定利益的流向,会引起攻击者的注意。因此,以"now"和"timestamp"为中心提取其上下文信息。

(3)未检查返回值漏洞

以太坊虚拟机是基于栈的。智能合约发生调用等操作时,合约堆栈自动增加一位。EVM堆栈的最大深度为1 024,超过此深度将导致堆栈溢出。然而,当一些低级函数运行时发生错误,它们不会抛出异常,而是继续执行,直到gas用完或是堆栈溢出。

通常,智能合约的子调用发生异常会自动向上级传播。但是,一些底层调用发生错误只会返回布尔值false。这些调用包括send、call、delegatecall和callcode等,以这些低级调用为关键属性,提取其上下文结构。

(4)整数上溢和整数下溢

以太坊虚拟机为整数指定固定大小的数据类型,一个固定位数的变量只能表示一定范围的数字。如果超出范围,数据将丢失。例如将256存储到uint8,则得到的是0,发生了整数上溢。如果0减1,则为255。以太坊封装了SafeMath库函数,计算过程引用库函数可以避免此类问题发生。

整数上溢会发生在加法和乘法上。如果加法和乘法没有使用SafeMath库,则有发生溢出的风险,需要通过提取加法和乘法的属性信息来测试其安全性。整数下溢通常是减法溢出。SafeMath库会封装sub函数。如果简单使用减法,会有整数下溢的风险。

2.3 语义结构特征提取

抽象语法树主要包含attribute和children信息。抽象语法树以attribute表示每个节点的属性信息。从根节点进入通过children节点深入到子节点,遍历整个AST文件得到了体现控制依赖的树。

在实际提取过程中,以代码关键字为核心,深度优先遍历整个抽象语法树。由于一个合约中可能出现多个敏感片段,遍历完成后会得到多个关键节点。以每个关键节点为中心,向上提取多层父节点信息,向下提取多层子节点信息得到子树。

从切片下的子树中查询其children的信息。如果存在children,则两个节点之间存在控制依赖类型的边。最终,通过节点邻接矩阵的形式表示子树的结构信息。每个节点都有其对应的属性信息,其中包括name信息、id信息、type信息以及isPure等信息。提取子树中每个节点的属性信息,这些可以体现源代码的语义信息。

3 图同构网络模型

本方法从抽象语法树中提取特征。树结构的节点周围结构是独一无二的,不具有平移不变性特征。传统的CNN和RNN难以很好地提取这种复杂的图特征。图神经网络(Graph Neural Network,GNN)[29]解决图分类任务有不错的效果。然而,很多GNN的方法大多通过简单的特征映射和邻域聚合学习并更新节点向量,在处理复杂多变的图数据时,有一定的局限性。并且,实际的图中不同节点之间可能存在具有相同计算图的节点。图同构网络GIN是一种新的GNN变种,适合解决复杂图中某些节点信息的不准确学习影响检测结果的问题。

GIN学习图结构数据本层中自身节点信息和邻域节点信息,将这些信息与上一层的激活值聚合并添加MLP(Multiple-Layer Perception)层更新每层特征。GIN每层特征更新公式如式(1)所示:

$$h_v^{(k)} = MLP^k\left((1+\varepsilon^{(k)}) \times h_v^{(k-1)} \sum_{u \in N(v)} h_u^{(k-1)}\right), \quad (1)$$

其中,MLP是多层感知机,ε是可学习的参数,h是特征向量。将每个图表示学习到的特征连接在一起,得到一个完整的图表示。通过添加MLP聚合每层学习后的图表示向量,从而得到当前层的表示,最后集成每层的图分类结果得到最终结果。

GIN需要学习图结构数据,将节点信息映射到邻接矩阵中生成图结构数据。抽象语法树中每个节点都有其独特的id编号,使用id作为每个节点的符号表示。在邻接矩阵中,按序排列节点,表示出各节点间的联系。表示属性信息时,按照关系矩阵的顺序排列,提取每个节点的属性信息。此外,需要使用word2vec算法[30]将提取的节点信息映射到向量空间。Word2vec可以将相似的词语映射到接近的向量空间中,在许多情况下能更准确地表示语义相似性。此时,就可将邻接矩阵和向量化的节点信息一一对应起来,得到图结构信息。使用不同层数GIN实验,来选取最佳的检测结果,本文使用的模型框架如图3所示。

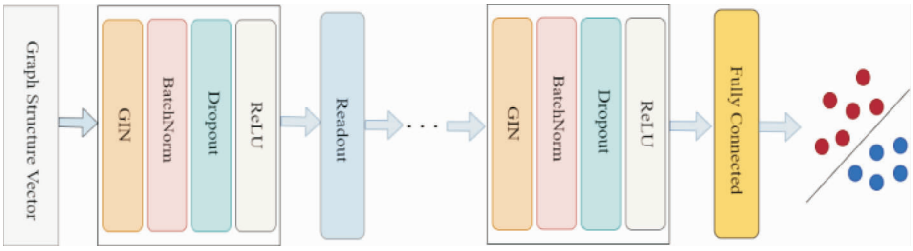


图 3　模型框架图

Fig. 3　Framework diagram of the model

4　实验与结果分析

在本小节中,将对上述方法进行充分实验与对比分析。首先,描述实验环境和使用的智能合约数据集;其次,对实验中关键参数进行分析对比,得到影响实验结果的关键因素;最后,详细分析了本方法的实验结果,并将本方法与其他一些主流工具和一些深度学习的方法进行对比。

4.1　实验环境设置

通过深度学习的方法对智能合约漏洞检测进行研究,使用 Pytorch 深度学习框架,Pytorch 简洁易用且内存使用效率高。实验中使用了大量的数据,去除了冗余特征,保留了与漏洞相关的关键特征,这大大减少了存储空间和计算量。实验环境设置如表 2 所示。

表 2　实验环境

Table 2　The experiment environment

软硬件环境	配置
CPU	Intel(R) Core(TM) i7-11800H @ 2.30GHz
操作系统	Windows 11
内存	DDR4 16G
GPU	NVIDIA RTX 3060
硬盘容量	1TB
深度学习框架	Pytorch 1.2.0

4.2　数据集

根据调研,目前无法找到数量充足且可靠的开源智能合约漏洞数据集。笔者从以太坊真实获取超过 3 万份合约,使用 Oyente、Mythril 和 Slither[31] 这些常见的智能合约漏洞检测工具标定数据集。在得到的数据集中,发现重入漏洞和时间戳依赖漏洞合约数量过少。从 TMP[25] 文章的开源数据集中,得到 273 个重入漏洞合约和 334 个时间戳依赖漏洞合约,最终,得到每种漏洞类型的合约如表 3 所示。在实验过程中,将 80% 作为训练集,20% 作为测试集。

表 3　数据集

Table 3　Dataset

漏洞类型	漏洞合约	正常合约
重入漏洞	425	2 504
时间戳依赖	924	3 895
未检查返回值	468	1 848
整数上溢	7 129	6 178
整数下溢	5 661	4 780

4.3　模型参数设置

在神经网络模型的训练过程中,许多因素都会影响模型效果。在实验过程中,向量化后的节点应该有相同的长度,但是,实际节点属性信息是不同长度。通过统计多数节点长度范围,将节点数据长度设置为一个比较合适的大小。在训练过程中,模型的学习率和模型层数对实验结果也有比较大的影响,需要对模型参数进行讨论,最终选择最合适的参数训练模型。

4.3.1　学习率

学习率控制模型参数的更新速度。学习率越大,模型参数变化越快。然而,学习率过大会导致参数更新的幅度太大,错过了模型的最优值;学习率过小可能会使得模型将局部最优点作为整体最优点。因此,学习率设置对模型检测效果有很大的影响。本实验将学习率设置为 0.1、0.01、0.001 和 0.000 1,使用准确率(accuracy)和 F1-score 评估模型效果,*Accuracy* 和 *F1-score* 的公式如下:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2}$$

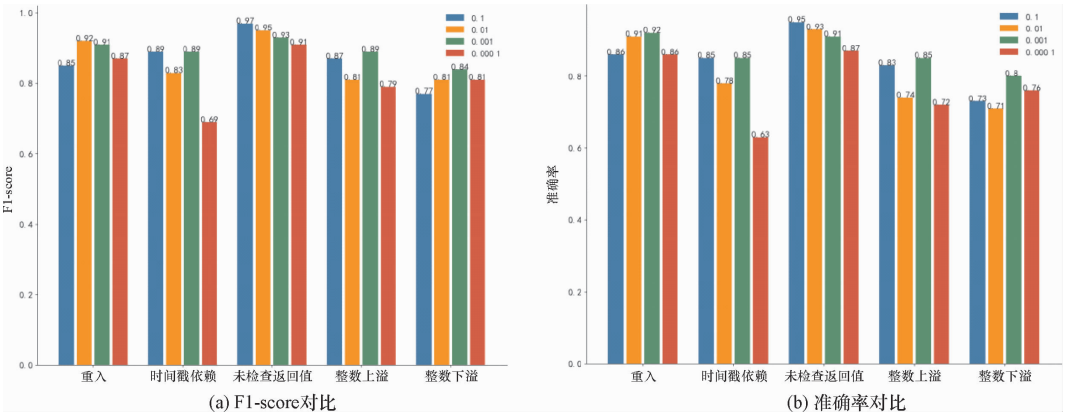
$$Precision = \frac{TP}{TP + FP}, \tag{3}$$

$$Recall = \frac{TP}{TP + FN}, \tag{4}$$

$$F1 - score = 2 \times \frac{Recall \times Precision}{Recall + Precision}。 \tag{5}$$

学习率对实验结果的影响见图 4。图 4(a)描述了学习率对 F1-score 的影响,图 4(b)表示学习率对准确率的影响。由于数据集分布不均衡,F1-score 可以更好地

表现出模型效果。结合图 4(a)和图 4(b)分析,重入漏洞的学习率设置为 0.01 和 0.001 的效果基本相同。但是学习率大,模型训练效率更高,因此,选取学习率为 0.01。时间戳依赖、整数上溢和整数下溢漏洞选取学习率为 0.001。未检查返回值漏洞的学习率为 0.1 时,模型训练效果最好。



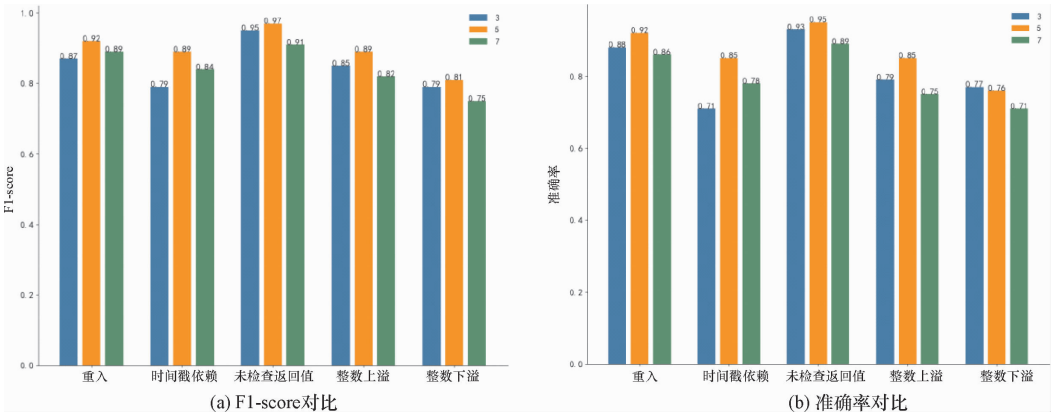
(a) F1-score对比 (b) 准确率对比

图 4 学习率对实验结果的影响

Fig. 4 The influence of learning rate on experimental performance

4.3.2 模型层数

模型层数对数据的学习效果有很大的影响。增加模型层数可以提高模型的表达能力,但模型层数增加会增加模型需要学习的参数,增加了模型的训练时间。如果模型层数设置不合适,可能会引起梯度消失的问题,因此,需要通过设置不同的层数,选取效果最好的模型。首先,将模型层数设置为 3,发现 3 个 GIN 层并不能很好地学习到数据集的漏洞特征;然后,增加模型层数,将层数分别设置为 5 和 7 层来观察检测结果;最后,通过准确率和 F1-score 来评估不同模型层数的效果,见图 5。图 5(a)描述了模型层数对 F1-score 的影响,图 5(b)表示模型层数对准确率的影响。当模型层数为 5 层时,模型准确率和 F1-score 都是最佳。因此,结合上文中的学习率,即可找到最佳模型参数。



(a) F1-score对比 (b) 准确率对比

图 5 模型层数对实验结果的影响

Fig. 5 The influence of model layers on experimental performance

4.4 实验结果与对比分析

为了验证提取的抽象语法树片段能够有效地表达智能合约漏洞特征。本小节详细分析了模型效果,多维度地评估模型有效性,并进一步对比了当前具有代表性的多个智能合约漏洞检测方法。

4.4.1 实验结果分析

为了评估 GIN 模型对不同类型漏洞的检测效果,笔者使用表 3 中的数据集在整数上溢、下溢、未检查返回值、时间戳依赖和重入这 5 种典型漏洞上训练检测模型。在 4.3 节中,经过对比实验得到最佳参数,实验结果如表 4 所示,模型使用 Accuracy、Precision、Recall、F1-score 和 AUC(Area Under Curve)作为评估指标。AUC 值为 ROC(Receiver Operating Characteristic)曲线所覆盖的区域面积,AUC 的值越大,分类器的分类效果越好。

由于实验所用数据集不是正负类特别均衡,综合性

指标 F1-score 和 AUC 更具分析意义。F1-score 结合精确率和召回率,能够很好地分析不平衡数据集。表 4 中,4 种漏洞类型 F1-score 都接近 90%。其中,未检查返回值漏洞的 F1-score 达到 97%。AUC 是 ROC 曲线的面积,它能够很好地体现分类器效果且不受阈值影响。在本实验中,未检查返回值、时间戳依赖和重入漏洞的 AUC 值均超过 90%,可以有效检测出漏洞。考虑到智能合约上线后难以修改,召回率对其有较大影响。除整数下溢漏洞外,其余的召回率均超过 90%。表 4 中的数据表明,本方法能够有效地检测智能合约漏洞。

表 4　GIN 的检测效果

Table 4　The detection performance of GIN detection model

指标	漏洞类型				
	整数上溢	整数下溢	未检查返回值	时间戳依赖	重入
Accuracy	0.85	0.80	0.95	0.85	0.92
Precision	0.87	0.92	0.95	0.87	0.94
Recall	0.90	0.77	0.98	0.90	0.90
F1-score	0.89	0.84	0.97	0.89	0.92
AUC	0.88	0.80	0.96	0.93	0.96

此外,对于 5 种漏洞类型的检测效果,笔者绘制出了 ROC 曲线(图 6),横坐标为假正率(False Positive Rate,FPR),纵坐标为真正率(True Positive Rate,TPR)。理论上,曲线越靠近左上部分,分类效果越好,从图 6 可以看出,重入漏洞和未检查返回值漏洞分类模型有更好的分类效果。

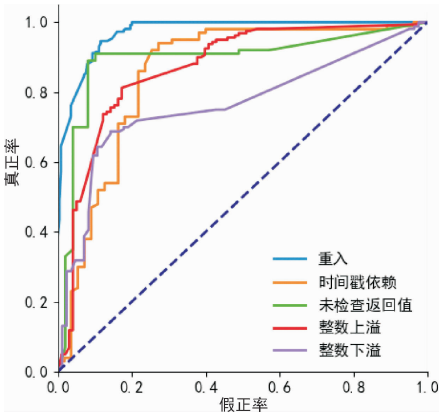


图 6　GIN 模型的 ROC 曲线

Fig. 6　The ROC curves of GIN

4.4.2　对比实验分析

为了进一步评估本方法基于源码语义结构分析的智能合约漏洞检测方法的有效性,选取了广泛使用的漏洞检测工具、使用图神经网络的检测方法和基于字节码的深度学习方法进行对比实验。Oyente[21] 和 Mythril[22] 是目前使用最广泛且被大家所熟知的检测工具。Oyente 是最早的智能合约漏洞分析工具,它使用符号执行的方法将字节码转化为控制流图并分析其中的路径可解性,设计分析模块找到漏洞位置。Mythril 是由以太坊开源社区所提供的基于字节码的安全分析工具,它使用符号执行和污点分析技术。DR-GCN[25] 是基于图神经网络的源码检测方法,它通过设计自定义规则提取合约结构信息。近年来,很多深度学习方法从字节码中提取特征,例如 KNN 和 DT 是使用 bigram 提取字节码特征训练的机器学习模型,LSTM 和 ResNets 是通过将字节码序列流转换为向量训练的深度学习模型[32]。使用统一的数据集对上述方法分析对比,实验结果如表 5 所示。

表 5　模型检测效果对比

Table 5　The detection performance comparison of eight detection models

方法		F1-score					Macro-F1
		整数上溢	整数下溢	未检查返回值	时间戳依赖	重入	
Bytecode-Based	Oyente	0.72	0.71	0.67	0.59	0.62	0.66
	Mythril	0.70	0.67	0.68	0.62	0.69	0.67
	KNN	0.71	0.65	0.85	0.80	0.77	0.76
	DT	0.70	0.67	0.80	0.69	0.76	0.72
	LSTM	0.79	0.82	0.77	0.72	0.69	0.76
	ResNets	0.85	0.81	0.81	0.88	0.70	0.81
Sourcecode-Based	DR-GCN	–	–	–	0.75	0.76	0.76
	GIN	0.89	0.84	0.97	0.89	0.92	0.90

从总体来看,深度学习的方法更好。由于符号执行的路径求解效率低,对长合约容易产生路径爆炸的问题,因而 Oyente 和 Mythril 的检测精度较低。DR-GCN 可以检测时间戳依赖漏洞和重入漏洞,它通过设计自定义规则提取源码中的函数。由于源码函数名具有自定义性,可能会产生较多的漏报。从检测结果可以看出,ResNets 是上述方法中检测结果最好的方法,它的检测结果超过了 80%。本文的方法相较于 ResNets 有较大的提升,主要在未检查返回值漏洞和重入漏洞方面提升了 16% 和 22%。综合上述方法,GIN 的结果明显优于其他方案。因此,基于抽象语法树中的固有属性提取关键树结构片段,从树结构中片段提取结构信息和语义信息能有效地表达出漏洞特征,GIN 模型能有效学习这些特征。

5 结 论

AST 是源代码经过处理后的得到的树状表示,AST 的每个节点都是源代码中的一部分。AST 描述了智能合约源代码的结构特征和语义特征。智能合约运行时需要执行编译为 AST 的过程,因此,研究智能合约漏洞在 AST 属性中表现出的固有特征,以这些特征为中心,提取其前后关联节点,并且使用 GIN 学习树结构。经过对特征向量长度和模型层数等不断调整,发现相比于其他的方法,GIN 具有更高的精准率和召回率。特别是,未检查返回值和重入漏洞的检测效果达到了 97% 和 92%,相较于其他方法中最优的效果提升了 16% 和 22%。

相比于使用字节码检测智能合约漏洞的深度学习方法,AST 更能表示出智能合约的数据依赖。特别地,切片出关键片段,也使得深度学习的检测结果可以从这些片段入手分析,这增加了深度学习的可解释性。在面对比较长的合约,符号执行的方法比如 Oyente 会产生路径爆炸的问题。模糊测试的方法比如 ContractFuzzer 等会由于路径覆盖率低导致代码很大一部分无法参与测试。本方法可以轻松遍历 AST,切片关键片段,并且不会漏掉任何片段。第 4 节的实验结果表明,本方法有更好的检测效果。

然而,本方法能提取到的漏洞特征是由于源码书写不当引起的,对于一些与代码相关性较小的漏洞,如区块链系统层的漏洞,则无法从源码中提取其特征。并且,由于深度学习需要可靠的数据集,目前无法对更多的漏洞类型分析检测。GIN 需要带标签数据集,所以本方法无法学习到未定义的新漏洞。未来希望将目标聚集在对未知类型漏洞的探索,因为现有的方法大多都是在研究已知漏洞类型检测方法,企图提高检测效果。然而,面对越来越多的合约安全事件,未知的漏洞更有风险,它可能会使用户遭受重大损失却没有丝毫防备。

参考文献:

[1] Nakamoto S. Bitcoin:A peer-to-peer electronic cash system[J]. Decentralized Business Review, 2008: 21260.

[2] Yaga D, Mell P, Roby N, et al. Blockchain technology overview[EB/OL]. [2023-04-18]. https://doi. org/10. 6028/NIST. IR. 8202.

[3] Vujičić D, Jagodić D, Ranđić S. 2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH), March 21-23, 2018[C]. Piscataway: IEEE, 2018.

[4] Azzi R, Chamoun R K, Sokhn M. The power of a blockchain-based supply chain[J]. Computers & Industrial Engineering, 2019, 135: 582-592.

[5] McGhin T, Choo K K R, Liu C Z, et al. Blockchain in healthcare applications: Research challenges and opportunities[J]. Journal of Network and Computer Applications, 2019, 135: 62-75.

[6] Bhaskar P, Tiwari C K, Joshi A. Blockchain in education management: Present and future applications[J]. Interactive Technology and Smart Education, 2021, 18(1): 1-17.

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

[8] Bhushan B, Khamparia A, Sagayam K M, et al. Blockchain for smart cities: A review of architectures, integration trends and future research directions[J]. Sustainable Cities and Society, 2020, 61: 102360.

[9] Cai W, Wang Z H, Ernst J B, et al. Decentralized applications: The blockchain-empowered software system[J]. IEEE Access, 2018, 6: 53019-53033.

[10] Wang Z, Jin H, Dai W, et al. Ethereum smart contract security research: Survey and future research opportunities[J]. Frontiers of Computer Science, 2021, 15: 152802.

[11] Wang B, Yuan X H, Duan L, et al. DeFiScanner: Spotting DeFi attacks exploiting logic vulnerabilities on blockchain[J]. IEEE Transactions on Computational Social Systems, 2022:1-12.
[12] Sayeed S, Marco-Gisbert H, Caira T. Smart contract: Attacks and protections[J]. IEEE Access, 2020, 8: 24416-24427.
[13] Baldoni R, Coppa E, D'elia D C, et al. A survey of symbolic execution techniques[J]. ACM Computing Surveys (CSUR), 2018, 51(3): 1-39.
[14] Beaman C, Redbourne M, Mummery J D, et al. Fuzzing vulnerability discovery techniques: Survey, challenges and future directions[J]. Computers & Security, 2022,120: 102813.
[15] Bhargavan K, Delignat-Lavaud A, Fournet C, et al. Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, October 24, 2016[C]. New York: ACM, 2016.
[16] Tsankov P, Dan A, Drachsler-Cohen D, et al. Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, October 15-19, 2018[C]. New York: ACM, 2018.
[17] Kalra S, Goel S, Dhawan M, et al. Network and Distributed Systems Security (NDSS) Symposium, February 18-21, 2018 [C]. San Diego: NDSS, 2018.
[18] Xu K, Hu W, Leskovec J, et al. How powerful are graph neural networks? [EB/OL]. (2018-12-26)[2023-04-20]. https://arxiv.org/1810.00826v2.pdf.
[19] Qian P, Liu Z, He Q, et al. Smart contract vulnerability detection technique: A survey[EB/OL]. (2022-09-13)[2023-04-20]. https://arxiv.org/pdf/2209.05872v1.pdf.
[20] Almakhour M, Sliman L, Samhat A E, et al. Verification of smart contracts: A survey[J]. Pervasive and Mobile Computing, 2020, 67: 101227.
[21] Luu L, Chu D H, Olickel H, et al. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, October 24, 2016[C]. New York: ACM, 2016.
[22] Mythril project. [EB/OL]. [2023-04-20]. https://github.com/ConsenSys/mythril.
[23] 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.
[24] 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]. (2021-06-17)[2023-04-20]. https://arxiv.org/pdf/2106.09282.pdf.
[25] Zhuang Y, Liu Z, Qian P, et al. Proceeding of 29th International Joint Conference on Artificial Intelligence, July 1, 2020 [C]. Morgan Kaufmann:IJCAI, 2020.
[26] Zhang J, Wang X, Zhang H, et al. 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), May 25-31, 2019[C]. Piscataway: IEEE, 2019.
[27] Dam H K, Tran T, Pham T, et al. Automatic feature learning for predicting vulnerable software components[J]. IEEE Transactions on Software Engineering, 2021, 47(1): 67-85.
[28] Wu B, Liang B, Zhang X. Turn tree into graph: Automatic code review via simplified AST driven graph convolutional network[J]. Knowledge-Based Systems, 2022, 252: 109450.
[29] Zhou J, Cui G, Hu S, et al. Graph neural networks: A review of methods and applications[J]. AI Open, 2020, 1: 57-81.
[30] Rong X. word2vec parameter learning explained[EB/OL]. (2014-11-11)[2023-04-20]. https://arxiv.org/pdf/1411.2738v1.pdf.
[31] Feist J, Grieco G, Groce A. 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), May 27-27, 2019[C]. Piscataway: IEEE, 2019.
[32] Wu Z, Li S, Wang B, et al. 2022 4th International Conference on Data Intelligence and Security (ICDIS), August 24-26, 2022[C]. Piscataway: IEEE, 2022.

【责任编辑: 卓祯雨】