计算机科学
COMPUTER SCIENCE

基于分层注意力网络和积分梯度的细粒度漏洞检测方法

李秋月, 韩道军, 张磊, 许涛

引用本文

李秋月, 韩道军, 张磊, 许涛. 基于分层注意力网络和积分梯度的细粒度漏洞检测方法[J]. 计算机科学, 2024, 51(12): 326-333.

LI Qiuyue, HAN Daojun, ZHANG Lei, XU Tao. Fine-grained Vulnerability Detection Based on Hierarchical Attention Networks and Integral Gradients [J]. Computer Science, 2024, 51(12): 326-333.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.231000174

基于分层注意力网络和积分梯度的细粒度漏洞检测方法

李秋月[1,3]　韩道军[1,2]　张　磊[1]　许　涛[1]
1 河南大学计算机与信息工程学院　河南 开封 475004
2 河南省智能技术与应用工程技术研究中心　河南 开封 475004
3 河南财政金融学院计算机与人工智能学院　郑州 450002
(hdj@henu.edu.cn)

摘　要　智能合约是一种基于区块链平台运行的去中心化应用程序,在数字货币、物联网、供应链等多个领域应用广泛。智能合约漏洞检测的研究对于保障数字资产安全、维护合约的可靠性与稳定性具有重要意义。目前的主流研究之一为利用深度学习模型自动学习代码特征,检测出智能合约漏洞,准确性较高,但是在漏洞解释方面具有局限性,不能提供细粒度的漏洞信息。针对目前基于深度学习的智能合约漏洞检测模型不能有效提供细粒度漏洞解释,且缺少细粒度标签的问题,提出一种基于分层注意力网络和积分梯度的细粒度漏洞检测方法。利用分层注意力网络进行粗粒度漏洞检测,通过两层注意力构建单词注意力编码层和函数注意力编码层分别学习源代码的函数级和合约级表示,以关注代码的不同令牌和语句;然后使用积分梯度方法进行细粒度解释,计算代码语句对漏洞预测的贡献度,以获取与漏洞相关的脆弱语句,实现无语句标签情况下的单词级别和语句级别的漏洞解释。在真实以太坊数据集 SmartbugsWilds,SmartbugsCurated 和 SolidiFIBenchmark 上的实验结果表明,该方法在 5 种漏洞类型上的平均准确率达到 80%以上,漏洞解释准确率提升 6%,可以更加准确地定位漏洞代码,帮助开发人员审查合约。

关键词:智能合约;漏洞检测;注意力机制;积分梯度
中图分类号　TP391

Fine-grained Vulnerability Detection Based on Hierarchical Attention Networks and Integral Gradients

LI Qiuyue[1,3],HAN Daojun[1,2],ZHANG Lei[1] and XU Tao[1]
1 School of Computer and Information Engineering,Henan University,Kaifeng,Henan 475004,China
2 Henan Engineering Research Center of Intelligent Technology and Application,Henan University,Kaifeng,Henan 475004,China
3 School of Computer and Artificial Intelligence,Henan Finance University,Zhengzhou 450002,China

Abstract　Smart contracts are decentralized applications that run on blockchain platforms and are widely used in many fields,including digital currencies,the Internet of Things,and supply chains. Research on vulnerability detection in smart contracts is of great importance for securing digital assets and maintaining the reliability and stability of contracts. One of the current mainstream researches is to use deep learning models to automatically learn code features,so as to detect vulnerabilities in smart contracts. It has high accuracy,but has limitations in vulnerability interpretation and cannot provide fine-grained vulnerability information. To address the problem that the current deep learning-based smart contract vulnerability detection model cannot effectively provide fine-grained vulnerability explanation and lacks of fine-grained labels,a fine-grained vulnerability detection method based on hierarchical attention network and integral gradient is proposed. Using hierarchical attention network for coarse-grained vulnerability detection,the word attention encoding layer and function attention encoding layer are constructed by two attention layers to learn the function-level and contract-level representations of the source code,respectively,to pay attention to the various tokens and statements of the code;and then the integrated gradient method is used to provide fine-grained explanations and calculate the contribution of code statements to vulnerability prediction,to obtain the vulnerability statements related to vulnerabilities,so as to realise the statement-less tags in the case of word-level and statement-level vulnerability interpretation. Experimental results on real Ethereum datasets SmartbugsWilds,SmartbugsCurated and SolidiFIBenchmark show that the proposed method

到稿日期:2023-10-25　返修日期:2024-04-04
基金项目:河南省高校青年骨干教师基金(2020GGJS027);国家自然科学基金(42371433);河南省科技攻关项目(232102240020,232102211056)
This work was supported by the University Young Core Instructor Foundation of Henan Province(2020GGJS027),National Natural Science Foundation of China(42371433) and Key Science and Technology Program of Henan Province(232102240020,232102211056).
通信作者:韩道军(hdj@henu.edu.cn)

achieves an average accuracy of more than 80% on five vulnerability types, with a 6% improvement in the accuracy of vulnerability interpretation, which can locate the vulnerable code more accurately and help developers to review contracts.

Keywords Smart contract, Vulnerability detection, Attention mechanism, Integrative gradients

1 引言

随着数字经济的进一步发展和社会信息化、智能化程度的持续提升，区块链作为一种重要的信息技术，应用领域不断扩大，为智能合约的发展和创新带来新的挑战和困难[1-2]。智能合约是一种基于区块链平台运行的去中心化应用程序，由计算机科学家和密码学家 Nick Szabo 在 1995 年首次提出，《Smart Contracts: Building Blocks For Digital Markets》一文中将智能合约定义为“一组以数字形式指定的承诺，包括各方履行这些承诺的协议”[3]。与传统合约相比，智能合约可以提高交易效率，具有去中心化、不可篡改等特性[4]，目前被广泛应用于数字货币、物联网、供应链等多个领域[5-7]。然而，智能合约本质上是公开的且不可更改的代码，若存在漏洞则会导致数字资产盗窃、合约功能失效等问题。2017 年，Parity 钱包智能合约出现了一个漏洞，导致价值超过 3000 万美元的以太币被锁定[8]。2021 年公开报道的 189 起典型安全事故中，DAPP 类合约安全事故共 145 起，占比为 76.71%，造成损失达 69.3 亿美元[9]。智能合约频发的安全事故阻碍了智能合约的发展，对智能合约进行安全性研究有利于智能合约的应用与发展。在智能合约部署上链之前对合约进行漏洞检测，可以及时地发现合约中存在的漏洞，最大程度避免资产损失。同时，自动化的漏洞检测方法可以帮助开发人员发现漏洞，减少开发人员的审查工作量。

目前，深度学习模型在智能合约漏洞检测中的应用已经受到了广泛关注。深度学习模型可以自动学习代码特征，提高漏洞检测的准确性，已在智能合约漏洞检测方面取得了很多研究成果[10]。然而，现有的基于深度学习的方法不能提供细粒度的漏洞信息，与基于符号执行、形式化验证等方法相比，可解释性差，不利于代码的审查工作。基于机器学习和深度学习的漏洞检测方法可以学习漏洞模式实现自动化的漏洞检测，降低审查成本。但基于深度学习的漏洞检测方法只能输出该合约是否存在漏洞，不能输出该漏洞在合约源代码中的具体位置信息，即无法定位漏洞的具体位置。漏洞检测模型若能提供与漏洞相关的脆弱性语句，则可以进一步帮助开发人员审查代码。利用漏洞检测方法检测合约源代码是否存在漏洞，并提供与漏洞相关的脆弱性语句，在本文中被称为细粒度漏洞检测。

为了提供与漏洞相关的代码行的细粒度信息，本文提出一种基于分层注意力网络和积分梯度的细粒度漏洞检测方法。该方法旨在实现对智能合约的粗粒度检测和细粒度解释，在不需要细粒度标签的情况下，实现细粒度漏洞检测，定位脆弱性代码语句。该方法将合约代码分为多个层级，使用两层注意力来分别学习源代码的函数级和合约级表示，以关注代码的不同令牌和语句。同时，引入可解释机器学习方法积分梯度对检测到的漏洞进行解释，计算每个特征对漏洞预测的贡献度，以获取与漏洞相关的脆弱语句。通过该方法，模型可以更加准确地定位漏洞，从而为开发者提供更多的漏洞信息，帮助开发人员审查代码。

本文第 2 章讨论了相关工作；第 3 章介绍了模型的总体框架和详细步骤；第 4 章展示了实验结果，并评估了模型的准确率和 F1 值；最后总结全文并展望未来。

2 相关工作

从基于符号执行、形式化验证的方法到基于机器学习、深度学习的方法，智能合约漏洞检测领域目前已经取得很多成果。虽然基于机器学习的漏洞检测方法在检测精度、检测时间上具有优势，但目前也存在着一些问题阻碍着该方法的进一步应用。智能合约通常包含几百行甚至上千行代码，而发生漏洞的代码通常是少数的。对于开发人员来说，审查全部代码的工作量是巨大的。因此，开发人员更希望漏洞检测工具在检测合约是否存在漏洞的同时，能够给出漏洞相关的信息，以减轻开发人员的代码审查量。基于符号执行、形式化验证的方法在进行漏洞检测时，通常会给出漏洞的具体信息，指出漏洞发生的具体位置。而由于机器学习的黑盒性，基于机器学习的漏洞检测方法的决策依据和具体决策过程是不可见的。模型只会检测合约是否存在漏洞，而不能提示哪些代码导致了该漏洞。另外，由于决策的不透明性，用户不知道模型判断合约存在漏洞的依据。这些因素限制了漏洞检测模型的进一步应用。近年来，有研究人员开始提出可解释的细粒度漏洞检测模型，以解决模型不能提供细粒度漏洞信息的问题。

为解决现有漏洞检测器的漏洞检测能力和定位精度仍未满足现实应用的问题，Li 等[11]提出基于深度学习的 C 程序源代码细粒度漏洞检测器 VulDeeLocator，其可以同时实现高检测能力和高定位精度。首先利用漏洞语法特征识别代码片段，作为漏洞检测初始候选。然后，将源代码转为中间代码，从中间代码中获取代码切片。对于这些中间表示，VulDeeLocator 将其编码为向量，用于训练神经网络。最后，模型输出更短或更少的代码段，可以表明漏洞发生的位置。VulDeeLocator 利用程序分析和深度学习技术，通过程序分析生成漏洞候选，并使用深度学习减少程序分析技术产生的误报，在漏洞检测和漏洞定位任务上都取得了较好的性能。但程序切片的选择依赖于漏洞语法特征，该方法自定义的 4 种漏洞语法特征可能不能完全获取到漏洞相关语句。Li 等[12]提出 IVDetect 模型，该模型是一种基于 C/C++ 语言的可解释漏洞检测器，其使用机器学习方法检测漏洞，给出粗粒度结果，同时使用智能助手（IA）给出漏洞检测的细粒度解释。在漏洞检测中，使用程序依赖图表示源代码，通过图卷积网络对程序依赖图学习进行漏洞检测。在漏洞解释中，IVDetect 寻找解释子图，解释子图被定义为 PDG 的最小子图，其中包含与漏洞最相关的语句。通过该模型得到的结果，使用者可以检查存在潜在漏洞的合约，并通过解释进一步调查代码中导致模型预测该漏洞的语句。Nguyen 等[13]提出一种基于对比学习

和信息论的细粒度漏洞检测模型。该模型利用互信息学习一组潜在变量，这些潜在变量表示源代码语句与相应函数漏洞的相关性。然后，他们提出新的聚类空间对比学习，以进一步改善表征学习和脆弱性相关代码语句的鲁棒选择过程。该模型在 C/C++数据集上进行实验，在 VCP，VCA 和 Top-10 ACC 指标上比基线的性能更好。

但在智能合约漏洞检测领域，现有的工作大多关注预测给定的程序代码是否易受攻击，而有关漏洞可解释性的研究工作较少。Nguyen 等[14]提出针对智能合约漏洞的细粒度检测模型 MANDO，MANDO 结合多粒度级别的异构控制流图和调用图，将合约表示为异构合约图。该方法使用节点级注意力提出新的异构图神经网络架构，以构建异构合约图的多粒度级别的嵌入，使用图分类、节点分类来实现粗粒度检测和细粒度检测。MANDO 模型可以提供漏洞的细粒度解释，但在细粒度检测任务的训练中需要大量语句级别的漏洞标签，然而可供模型训练的带有语句级别标签的真实以太坊合约数据集太少，模型在漏洞解释性能上存在着不足。

3 本文方法

3.1 方法概述

智能合约检测粒度可以分为文件级(File)、合约级(Contract)、函数级(Function)和语句级(Statement)。一个文件包含多个合约，每个合约实现不同的功能。文件级别漏洞检测对智能合约的全部源代码进行检测，其检测粒度太大，检测准确率不高。现有的工作通常在合约级粒度进行漏洞检测。合约级漏洞检测将整个合约源代码分为多个小的合约，对每个合约单独进行检测。合约级漏洞检测获取整个合约的漏洞结果，可以检测该单个合约是否存在漏洞，但其不能提供更小粒度的漏洞信息。若想获得更细粒度的漏洞信息，则需要在更小级别的粒度上进行检测，然而缺少函数级和语句级粒度的标签。为保证漏洞检测性能，并获取语句级别的细粒度漏洞信息，本文提出的细粒度漏洞检测方法在合约级别进行粗粒度漏洞检测，在语句级别进行细粒度漏洞解释。

细粒度漏洞检测方法流程如图 1 所示，包括智能合约漏洞检测模型训练和智能合约漏洞细粒度检测两部分。在合约漏洞检测模型训练中，首先对合约源代码进行预处理，然后构建分层注意力网络(Hierarchical Attention Networks，HAN)进行漏洞检测。分层注意力网络通过分层结构学习智能合约的函数表示并更新函数表示获得合约表示，然后利用合约表示进行漏洞检测，实现合约级别的粗粒度检测。分层注意力网络由两个具有注意力机制的 BiLSTM 层组成，从单词和函数级别学习源代码的函数表示和合约表示。通过在单词级别和函数级别应用注意力机制，可以使模型关注更重要的单词和函数，从而更准确地定位漏洞点。多次训练后，选择最优的检测模型作为最终的合约漏洞检测模型。在合约细粒度检测部分，输入 Solidity 源代码进行漏洞检测。如果源代码中存在漏洞，输出源代码的漏洞检测结果和源代码中的脆弱代码行。在细粒度解释过程中，使用积分梯度(Integrative Gradient，IG)对输入源代码的单词重要性进行分析，计算单词的显著性得分。然后对一个语句内的单词显著性得分进行求和作为该句子的显著性得分，选择显著性得分最高的语句作为与漏洞最相关的脆弱代码行，实现语句级别的细粒度解释。

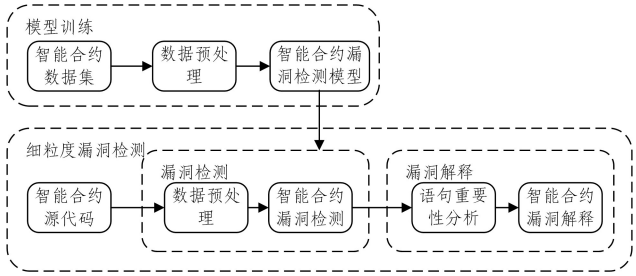


图 1 方法流程图

Fig. 1 Method flowchart

3.2 细粒度漏洞检测模型

3.2.1 分层注意力漏洞检测模型

为在语句粒度级别进行漏洞解释，本文利用源代码进行检测。一个合约通常实现多个函数，每个函数功能不相同。为使模型在更细粒度级别学习漏洞特征以方便定位漏洞，本文对合约源代码进行分层，将合约分为多个函数，学习每个独立的函数表示和整个合约表示。由于源代码中单词和语句对漏洞的贡献程度不一样，通常自定义函数调用、方法执行的代码片段比语言本身定义的关键字更重要[15]。考虑到这个特点，本章设计的漏洞检测模型使用分层注意力结构[16]，通过分层使模型更细粒度地学习漏洞特征，通过注意力机制使模型关注与漏洞最相关的单词和函数语句。模型结构如图 2 所示，模型由单词嵌入层、单词注意力编码层、函数注意力编码层和漏洞检测层组成。

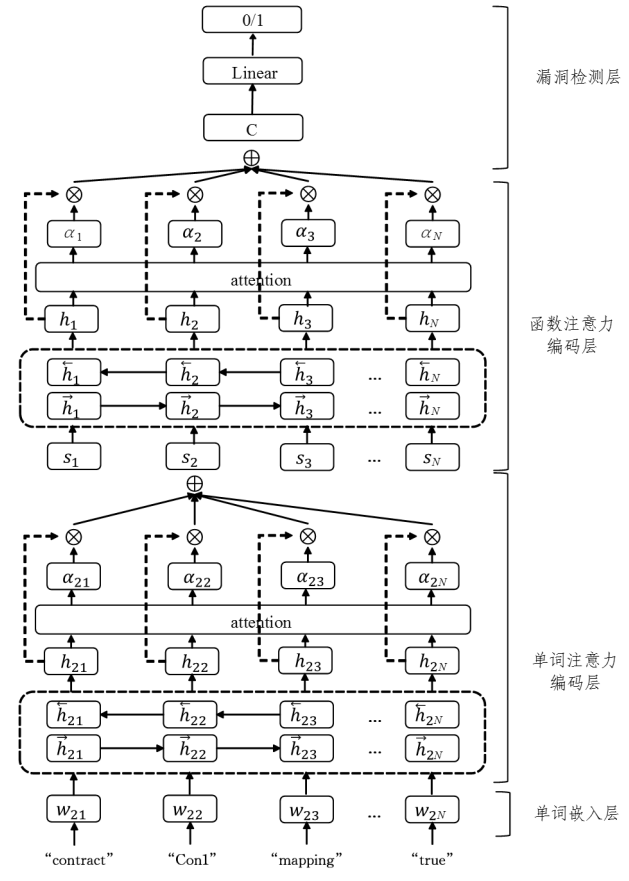


图 2 基于分层注意网络的漏洞检测模型结构

Fig. 2 Architecture of vulnerability detection model based on hierarchical attention networks

在单词注意力编码层和函数注意力编码层使用带注意力机制的 BiLSTM 获取函数表示和合约表示。LSTM 可以有效处理序列之间的关系，保留历史信息。源代码文本存在词序信息，使用 LSTM 来学习源代码表示可以传递当前单词之前的历史信息。为捕获当前单词和语句的上下文信息，本文使用 BiLSTM 处理单词和函数序列。BiLSTM 可以获取源代码的上下文表示，但它不能集中于在源代码的重要信息。源代码中各单词对漏洞预测的重要性不同，关注重要信息可以提高分类的准确性。注意力机制可以通过学习不同的权重来突出源代码中的重要单词和函数。相比对 LSTM 所有状态的输出进行求和获取函数表示和合约表示，加入注意力机制对输出加权求和可以突出重要信息。因此，本文在 BiLSTM 层之上使用注意力机制构建单词注意力编码层和函数注意力编码层，突出源代码中的重要单词和函数。BiLSTM 和注意力机制相结合可以为漏洞解释提供相关信息，并提高检测的准确性。

1）单词嵌入层。本文使用 word2vec 获取源代码单词的词嵌入表示，将源代码的每个单词转换为向量表示。在使用 word2vec 进行训练之前，对源代码进行预处理操作。首先，对源代码进行清理，去除源代码中的空行和注释。其次，对源代码进行分层。将通过分号分隔的语句作为一条完整的代码语句，一个函数内的所有语句组成函数代码作为单词注意编码层的输入。最后，对代码进行标准化。开发人员的编写习惯不同，导致源代码中的变量、函数和合约名称存在差异。为了消除这种差异性，本文模型对代码进行标准化操作。具体地，将源代码中定义的变量名、函数名、合约名替换为符号表示 $VarN$，$FunN$，$ConN$，其中 N 是被替换名称在合约中首次出现的索引。在对源代码进行预处理操作后，使用 word2vec 进行向量化表示。合约的源代码长度各不相同，源代码包含的语句数不同，每条代码语句的长度也不相同，因此转换后的特征向量具有不同的长度。通过截断和填充操作，将源代码表示为一个具有固定长度的特征向量，当语句数量和语句长度小于固定值时用“0”填充，语句数量和语句长度大于固定值时进行截断。语句数量和语句长度的固定值通过分析数据集得到。

2）单词注意编码层。在单词注意力编码层，使用 BiLSTM 对单词序列进行编码。由于并非所有单词都与漏洞相关，因此在编码层之上添加单词注意力层。BiLSTM 的输入是源代码中的每个单词嵌入向量 $\boldsymbol{w}_m^t$，BiLSTM 从两个方向综合句子信息，一个前向 LSTM 学习单词 $\boldsymbol{w}_m^1$ 到 $\boldsymbol{w}_m^n$ 的序列信息，一个反向 LSTM 学习单词 $\boldsymbol{w}_m^n$ 到 $\boldsymbol{w}_m^1$ 的序列信息。设 $\overrightarrow{\boldsymbol{h}}_m^t$ 和 $\overleftarrow{\boldsymbol{h}}_m^t$ 分别为前向 LSTM 和反向 LSTM 的第 t 个隐藏状态，串联 $\overrightarrow{\boldsymbol{h}}_m^t$ 和 $\overleftarrow{\boldsymbol{h}}_m^t$ 得到单词 $\boldsymbol{w}_m^t$ 的隐含表示 $\boldsymbol{h}_m^t$。

$$\boldsymbol{h}_m^t=[\overrightarrow{\boldsymbol{h}}_m^t,\overleftarrow{\boldsymbol{h}}_m^t] \tag{1}$$

为区分单词的不同重要性，本文采用注意力机制。使用注意力机制为每个单词赋予不同的权重，对单词进行加权和得到语句表示。相比使用隐藏向量 $\boldsymbol{h}_m^t$ 的平均值作为语句表示，注意力机制可以得到包含更多漏洞信息的语句表示。使用 $\boldsymbol{\alpha}_m^t$ 表示单词 $\boldsymbol{w}_m^t$ 的归一化置信度，$\boldsymbol{\alpha}_m^t$ 的计算如式(2)、式(3)所示：

$$u_m^t=\tanh(W_w\boldsymbol{h}_m^t+b_w) \tag{2}$$

$$\boldsymbol{\alpha}_m^t=\frac{\exp(\boldsymbol{c}_w^{\mathrm{T}}u_m^t)}{\sum_t\exp(\boldsymbol{c}_w^{\mathrm{T}}u_m^t)} \tag{3}$$

$$\boldsymbol{s}_m=\sum_t\boldsymbol{\alpha}_m^t\boldsymbol{h}_m^t \tag{4}$$

其中，$\boldsymbol{\alpha}_m^t$ 衡量了单词 $\boldsymbol{x}_m^t$ 的隐藏表示与上下文向量 $\boldsymbol{c}_w$ 之间的相似度；$\boldsymbol{s}_m$ 代表函数 m 的最终表示，由单词表示 $\boldsymbol{h}_m^t$ 的加权和得到。

3）函数注意编码层。基于单词注意编码层得到合约函数表示后，使用函数注意编码层学习合约表示。首先使用 BiLSTM 学习不同函数之间的依赖信息以更新函数表示，然后使用注意力机制获取权重，根据注意力权重对函数表示进行加权求和，获得合约表示。函数向量的隐藏状态同样表示为前向隐藏状态和反向隐藏状态的串联。为了区分不同函数对合约漏洞的影响，再次采用如下的注意力机制：

$$u_m=\tanh(W_s\boldsymbol{h}_m+b_s) \tag{5}$$

$$\boldsymbol{\beta}_m=\frac{\exp(c_s^Tu_m)}{\sum_m\exp(c_s^Tu_m)} \tag{6}$$

$$\boldsymbol{C}_i=\sum_m\boldsymbol{\beta}_m\boldsymbol{h}_m \tag{7}$$

其中，$\boldsymbol{h}_m$ 是函数 m 的隐藏状态表示，$\boldsymbol{\beta}_m$ 是函数 $\boldsymbol{s}_m$ 的归一化置信度。最后，使用 $\boldsymbol{C}_i$ 表示第 i 个合约源代码的合约表示。

4）漏洞检测层。使用函数注意编码层更新函数表示获取合约表示，然后使用全连接层进行漏洞分类，并使用交叉熵损失训练。

通过多次训练，选择最优的模型作为漏洞检测模型，将待测合约输入漏洞检测模型检测该合约是否存在漏洞。接下来对模型决策过程进行分析，获得脆弱性语句信息。

3.2.2 基于积分梯度的漏洞解释方法

基于深度学习的模型可以在各种任务上获得良好的性能，但缺乏可解释性限制了其在现实任务尤其是安全敏感任务中的广泛应用。为了克服这一弱点，许多学者研究可解释机器学习方法以帮助理解模型内部的工作机制。解释每个预测背后的基本原理，可以更容易理解预测结果，特别是在漏洞检测领域，通常需要更合理的预测。积分梯度（IG）是一种用于解释神经网络模型的可解释方法，是对简单梯度方法的改进。基于梯度的方法假设模型对特征越敏感，则此特征对模型越重要。该方法计算输入特征对模型输出的敏感度，称为特征的显著性得分。特征的显著性得分表明该特征对模型决策结果的影响程度。相比 SHAP，LIME 等对输入特征进行扰动的方法，IG 可以输出更加稳定的模型解释结果，其直接计算模型的解释，不依赖于近似值。为了对漏洞进行解释，需要找到源代码中与漏洞最相关的语句集，该语句对做出合约存在漏洞的决策的影响应大于其他语句。为此，本文使用积分梯度方法获取单词的显著性得分进行语句级细粒度解释。该方法在输入特征空间中对梯度进行积分，计算每个输入特征对模型输出的贡献。本文使用该方法计算漏洞检测模型中每个代码令牌对漏洞检测的影响程度，从而提供与漏洞相关的语句的细粒度解释。

定义训练好的漏洞检测模型为 $F(x)$，$x=(x_1,\cdots,x_n)$ 表示输入的合约源代码序列，x' 为基线输入，通过积分梯度方法获取输入 x 中与漏洞相关的语句 x_i。在进行解释时，给定一个输入 x 和目标输出类别 y，积分梯度方法通过计算基线 x' 到 x 的路径上的梯度来评估每个特征对输出类别 y 的影响。积分梯度的核心思想是通过逐步改变输入的特征来计算其对输出的影响。输入 x 的第 i 个特征 x_i 的重要性计算为基线 x' 到输入 x 的直线路径上的梯度路径积分，x_i 对模型输出的梯度积分表示如式(8)所示：

$$IntegratedGrads_i(x)=(x_i-x'_i)\times\int_{\alpha=0}^{1}\frac{\alpha F(x'+\alpha\times(x-x'))}{\alpha x_i}\mathrm{d}\alpha \tag{8}$$

应用积分梯度需要选择合适的基线，接近零值的输出分数的基线有利于将归因结果仅归结于输入值。在图像领域，通常选择全黑图作为基线。在对文本序列进行处理时，一般选择全零向量。在本文中，以全零向量作为基线值。在具体实现中，使用离散近似的方法计算积分梯度。积分的计算可以通过加和的方式来近似，可以在保障归因效果的基础上最大程度降低计算量。在实际计算时，在沿着基线值和输入值的路径上线性插入多个等间隔的值，在这些值上求对输出分数的梯度，然后进行权值加和。设 m 为积分的黎曼和近似的步值，计算式如下：

$$IntegratedGrads_i(x)=\frac{(x_i-x'_i)}{m}\times\sum_{k=1}^{m}\frac{F\left(x'+\frac{k}{m}\times(x-x')\right)}{x_i} \tag{9}$$

对于输入 x 的每个特征，计算归一化积分梯度，归一化积分梯度表示该输入特征对于模型输出的相对贡献，可以用于比较不同输入特征的重要性。获得每个单词 x_i 的显著性得分后，对语句内的单词显著性得分进行求和获取语句显著性得分。然后，根据语句显著性得分选择语句用作漏洞细粒度解释。

4 实验与分析

本章将在智能合约源代码数据集上进行分析实验，对细粒度解释漏洞检测模型的性能进行评估，验证模型细粒度解释的性能。本文对模型的合约级漏洞检测和语句级细粒度漏洞解释进行评估，以验证基于分层注意力网络的漏洞检测模型在源代码漏洞检测中的性能，以及漏洞解释方法在漏洞解释方面的性能。

4.1 数据集

本文使用 SmartbugsWilds[17]，SmartbugsCurated[17] 和 SolidiFIBenchmark[18] 数据集对细粒度漏洞检测模型进行评估。SmartbugsWilds 数据集包含 47 398 个真实以太坊环境中的智能合约。SmartbugsCurated 是一个智能合约漏洞数据集，包含漏洞类型和漏洞代码位置等漏洞注释信息。SolidiFIBenchmark 是一个手动注入漏洞的漏洞合约数据集，每个合约都包含 SolidiFi 工具注入的漏洞代码。同样，该数据集也包含对漏洞类型和漏洞代码行的说明。SmartbugsCurated 数据集共有 140 个漏洞合约，SolidiFIBenchmark 数据集包括 69 个注入漏洞代码的智能合约。本文基于 SmartbugsWilds 数据集训练模型并进行合约级漏洞检测评估，使用 SmartbugsCurated 和 SolidiFIBenchmark 进行细粒度漏洞解释评估。

4.2 实验设置

本文使用 word2vec 进行单词嵌入表示，词嵌入维度为 200；在单词注意力编码层和函数注意力编码层中设置 LSTM 隐藏单元数为 100。模型使用 Adam 优化器，学习率为 0.001，批次大小设置为 128，训练批次为 100。在进行细粒度漏洞解释时，设置积分的黎曼和近似的步值为 300。

4.3 结果及分析

在本章实验中，使用 F1-score 评价指标来评估漏洞检测模型的有效性。对于细粒度漏洞解释，使用准确率进行性能评估。若模型输出的解释语句包含漏洞语句，则将其视为有效解释，否则为无效解释。通过设置不同的 N 值（N 为模型输出的解释语句数量），本文评估了不同 N 值下的细粒度漏洞解释准确率。

4.3.1 合约级漏洞检测性能评估

首先，将本文方法与基于符号执行和基于深度学习等方法的漏洞检测工具进行比较（具体为 Oyente[19]，Mythril[20]，Securify[21]），分析漏洞检测模型的漏洞检测性能。Oyente 是一种基于动态符号执行方法的漏洞检测工具；Mythril 使用符号执行方法检测漏洞，是以太坊官方推荐的智能合约安全分析工具；Securify 将漏洞转化为模式匹配框架进行漏洞检测。同时，本文方法还与 CNN[22]，BiLSTM[23]，MANDO[14] 等深度学习模型进行比较。MANDO 是一个基于异构图神经网络的智能合约检测模型，其将合约表示为图结构，使用图神经网络进行漏洞检测。上述工具与模型在数据集上的实验结果如表 1 所列。

表 1 漏洞检测结果

Table 1 Vulnerability detection results

方法	F1/%				
	access control	arithmetic	Front running	reentrancy	Unchecked Low-level Calls
Oyente	13.00	56.30	17.00	61.18	34.00
Mythril	36.40	73.00	39.10	58.33	44.20
Securify	29.00	52.00	28.09	68.02	52.00
CNN	79.00	80.51	79.83	77.50	77.88
BiLSTM	80.46	77.26	84.45	78.60	75.86
MANDO	81.32	84.13	85.95	87.96	78.17
HAN	83.01	88.25	86.56	89.73	84.49

在合约级粗粒度漏洞检测中，本文使用的分层注意力结构漏洞检测模型在 5 种漏洞任务中 F1-score 均超过 83%，并在 arithmetic 和 reentrancy 漏洞检测任务上分别取得了 88.25% 和 89.73% 的性能。Oyente，Mythril 和 Securify 对特定漏洞类型可以实现较高的准确率，而基于深度学习的方法在各个漏洞检测任务中均能取得较为均衡的性能。Mythril 工具在 arithmetic 漏洞检测上具有较好的性能，而在 Unchecked Low-level Calls 检测中性能较差。与另外两种工具

相比，Mythril 检测工具对 arithmetic 漏洞的检测性能较优，但相比深度学习方法准确率较低。相比 Mythril 工具，深度学习模型在 arithmetic 检测任务有 4%～13%的性能提升。通过在 BiLSTM 上添加注意力机制，使得模型更关注于漏洞语句，在漏洞检测中能实现更高的准确率。

4.3.2 细粒度漏洞解释性能评估

在细粒度解释中，模型提供语句级别和单词级别的漏洞信息。相比合约级别、函数级别的漏洞检测，在语句级别进行漏洞解释可以帮助合约开发人员快速审查代码和定位漏洞。SmartbugsCurated 和 SolidiFIBenchmark 数据集包含漏洞代码行的信息，因此使用这两个数据集验证 IG 在漏洞检测模型上的检测效果。本文对传统工具和基于深度学习的方法进行对比实验。传统漏洞检测工具可以提供漏洞具体信息，本文使用 6 种检测工具进行对比实验。ManDO 模型通过对图节点分类可以提供漏洞节点信息，因此本文模型与该模型在细粒度语句解释上进行评估，在 5 种漏洞类型上的检测结果如表 2 所列。

表 2 漏洞解释准确率

Table 2 Vulnerability explanation accuracy

方法	准确率/%				
	access control	arithmetic	Front running	reentrancy	Unchecked Low-level Calls
Manticore	20.83	18.00	0.00	27.70	23.33
Mythril	29.16	60.00	42.85	55.56	66.70
Oyente	0.00	52.00	0.00	61.12	37.50
Securify	16.67	20.00	28.57	34.00	47.80
Slither	41.66	12.00	21.42	66.67	33.00
Smartcheck	27.08	8.00	0.00	62.20	44.44
ManDO	65.42	90.84	71.22	80.49	67.34
CNN+IG	71.87	92.00	65.31	81.57	73.29
LSTM+IG	75.00	88.00	43.47	84.21	66.19
HAN+IG	68.75	96.00	75.83	86.83	77.46

HAN+IG 的结构在 5 种检测任务上都取得了性能提升，其中对整数溢出漏洞的解释准确率最高。Smartcheck 和 Slither 检测工具在整数溢出漏洞检测上性能较差，Manticore 和 Oyente 检测工具在 Front running 漏洞检测上性能较差。相比深度学习模型，传统检测工具在不同漏洞类型的检测性能上存在较大差异。ManDO 可以对 arithmetic 和 reentrancy 漏洞实现有效的解释，但在 access control 和 Unchecked Low-level Calls 检测任务上同样存在不足。HAN+IG 在 access control 检测任务上的准确率相较于 CNN 和 LSTM 较低，通过分析发现可能是由于该漏洞类型的数据语句数量和语句长度差别较大。CNN+IG 的结构比 LSTM+IG 的结构性能更好。LSTM 模型学习时利用整个合约的状态进行分类，而 CNN 可以学习到局部特征，因此模型能关注到与漏洞相关的单词，但其在粗粒度漏洞检测中性能较差。HAN+IG 方法基于 LSTM，但通过注意力机制可以使模型关注到脆弱性单词，因此其实现了更高的准确率。通过对粗粒度检测和细粒度解释性能的综合分析可知，HAN+IG 在粗粒度漏洞检测和细粒度解释中获取了最好和最均衡的性能。

在上述准确率评估中，本文使用 $N=10$ 的语句排名设置。N 代表前 N 行解释语句，$N=10$ 表示取预测结果中排名前 10 的语句进行评估。通过对 N 设置不同的值，本文评估了解释语句数对解释准确率的影响。

表 3－表 7 展示了 N 取值为 1～10 时 5 种方法的准确率。对 5 种方法的解释性能进行分析可知，LSTM_IG 和 CNN_IG 在不同漏洞检测任务中性能表现不同，而 HAN+IG 通过注意力机制可以关注到脆弱性单词在漏洞解释中性能更好。$N1-N5$ 获得更高的准确率，表明模型可以在输出最少语句的同时准确定位漏洞代码。在 arithmetic 和 re-entrancy 漏洞解释中，HAN+IG 在 $N2$ 准确率上可以达到 60% 和 60.52%。在 5 种漏洞类型检测中，本文方法的 $N5$ 准确率均高于其他方法。LSTMATT+IG 为使用 BiLSTM 和注意力机制，与不使用注意力机制的 BiLSTM+IG 的方法相比，其在 $N1-N10$ 准确率上均有提升。而使用分层结构加注意力机制进一步提高了性能，与 LSTMATT+IG 的方法相比，HAN+IG 在准确率上的优势说明了分层结构的有效性。由实验结果可知，HAN+IG 方法可以有效获取与漏洞相关的脆弱性代码，实现漏洞代码定位。

表 3 access control 漏洞解释结果

Table 3 Results of access control vulnerability interpretation

解释方法	准确率/%									
	$N1$	$N2$	$N3$	$N4$	$N5$	$N6$	$N7$	$N8$	$N9$	$N10$
LSTM+IG	9.37	9.37	18.75	21.87	37.50	40.62	53.12	59.37	75.00	75.00
BiLSTM+IG	9.37	18.75	18.75	25.00	34.37	37.50	40.62	56.25	71.87	71.87
LSTMATT+IG	15.62	15.62	18.75	25.00	37.50	37.50	53.13	65.62	65.62	71.87
CNN+IG	15.62	21.87	28.12	34.37	37.50	50.00	56.25	68.75	68.75	71.87
HAN+IG	15.63	18.75	31.25	37.50	46.87	50.00	59.37	62.50	65.62	68.75

表 4 arithmetic 漏洞解释结果

Table 4 Results of arithmetic vulnerability interpretation

解释方法	准确率/%									
	$N1$	$N2$	$N3$	$N4$	$N5$	$N6$	$N7$	$N8$	$N9$	$N10$
LSTM+IG	16.00	24.00	44.00	52.00	56.00	64.00	72.00	76.00	80.00	88.00
BiLSTM+IG	28.00	36.00	48.00	56.00	64.00	72.00	72.00	80.00	88.00	96.00
LSTMATT+IG	12.00	12.00	16.00	52.00	64.00	76.00	76.00	84.00	88.00	88.00
CNN+IG	40.00	53.00	62.00	72.00	85.00	88.00	90.00	90.00	92.00	92.00
HAN+IG	40.00	60.00	68.00	80.00	88.00	88.00	92.00	92.00	96.00	96.00

表 5 front running 漏洞解释结果

Table 5 Results of front running vulnerability interpretation

解释方法	准确率/%									
	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10
LSTM+IG	4.34	4.34	13.04	17.39	17.39	21.73	21.73	34.78	43.47	43.47
BiLSTM+IG	4.43	8.69	8.75	10.24	21.73	21.73	30.43	30.43	34.78	39.13
LSTMATT+IG	7.14	14.28	14.28	21.42	28.62	28.57	35.71	42.85	57.41	64.28
CNN+IG	10.71	15.00	28.57	32.16	42.86	42.85	55.14	63.57	65.00	65.31
HAN+IG	8.33	16.67	25.00	41.66	51.60	56.08	58.34	66.68	72.50	75.83

表 6 re-entrancy 漏洞解释结果

Table 6 Results of re-entrancy vulnerability interpretation

解释方法	准确率/%									
	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10
LSTM+IG	0.00	0.00	13.15	42.10	60.52	65.78	68.42	68.42	71.05	84.21
BiLSTM+IG	21.05	39.47	57.89	73.68	73.68	78.94	81.57	84.21	86.84	89.47
LSTMATT+IG	26.31	60.52	65.78	78.94	78.94	78.95	79.32	80.26	81.57	84.21
CNN+IG	15.78	34.21	50.00	57.89	73.68	78.94	79.47	81.57	81.57	81.57
HAN+IG	39.47	60.52	63.15	76.31	81.57	82.63	83.94	85.00	85.00	86.83

表 7 unchecked low-level calls 漏洞解释结果

Table 7 Results of unchecked low-level calls vulnerability interpretation

解释方法	准确率/%									
	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10
LSTM+IG	23.94	28.16	28.17	35.21	36.61	50.70	53.52	60.56	61.97	66.19
BiLSTM+IG	5.63	12.67	15.49	18.30	28.16	30.98	35.21	39.43	43.66	47.88
LSTMATT+IG	4.22	4.22	12.67	33.80	39.43	47.88	53.52	70.42	70.42	70.42
CNN+IG	21.26	35.21	43.66	50.70	50.70	52.12	59.15	61.97	73.29	73.29
HAN+IG	32.39	33.80	51.26	53.52	56.33	60.56	63.38	66.20	70.42	77.46

4.3.3 示例分析

通过积分梯度对漏洞检测结果进行解释，可以获取智能合约源代码的单词和语句重要性得分。我们将单词显著性得分进行可视化，以分析模型学习到的每类漏洞的漏洞语句，如图 3—图 5 所示。图 3 为整数溢出漏洞检测实例，通过将单词显著性得分和语句显著性得分进行可视化可以发现，模型能较有效地学习到会引起漏洞的算术操作，模型对加减操作更为敏感。

```
contract TimeLock
mapping( address => uint ) public balances ;mapping( address => uint ) public lockTime ; function depo
balances [ msg.sender ]+=msg.value ; lockTime [ msg.sender ]=now+1weeks;
function increaseLockTime ( uint _secondsToIncrease ) public
lockTime [ msg.sender ]+= _secondsToIncrease ;
function withdraw () public
require ( balances [ msg.sender ]>0) ; require (now>lockTime [ msg.sender ]) ; uint transferValue =ba
```

图 3 整数溢出漏洞检测示例

Fig. 3 Example of integer overflow vulnerability detection

```
contract WALLET
function Put ( uint _unlockTime ) public payable
var acc=Acc [ msg.sender ]; acc.balance +=msg.value ; acc.unlockTime = _unlockTime >now? _unlockTim
function Collect ( uint _am) public payable
var acc=Acc [ msg.sender ]; if ( acc.balance >=MinSum&& acc.balance >=_am&&now> acc.unlockTime )
if ( msg.sender.call.value (_am)())
acc.balance-=_am ; LogFile.AddMessage ( msg.sender ,_am, " Collect " ) ;
function () public payable
Put (0) ;
struct Holder
uint unlockTime ; uint balance ;
mapping ( address => Holder ) public Acc ;Log LogFile ; uint public MinSum=1 ether ; function WALLET( ad
LogFile =Log ( log ) ;
```

图 4 re-entrancy 漏洞检测示例

Fig. 4 Example of re-entrancy vulnerability detection

```
contract Governmental
address public owner; address public lastInvestor ; uint public jackpot =1 ether ; uint public lastInv
owner=msg.sender ; if ( msg.value <1 ether ) throw ;
function invest ()
if ( msg.value < jackpot/2 ) throw ; lastInvestor =msg.sender ; jackpot +=msg.value/2 ; lastInvestmen
function resetInvestment ()
if ( block.timestamp <lastInvestmentTimestamp+ONE_MINUTE ) throw ; lastInvestor.send ( jackpot )
```

图 5 time_manipulation 漏洞检测示例

Fig. 5 Example of time_manipulation vulnerability detection

在检测不同的漏洞类型时，模型关注源代码的不同语句。由于 TextCNN 比 LSTM 更关注每个单词，因此模型在整数上溢与整数下溢中性能更佳。然而模型对于漏洞代码分布较远的代码漏洞检测还存在一定的不足，在漏洞语句的提示上还不够准确。

结束语 智能合约安全性是区块链安全的重要研究内容。近年来，许多研究者对智能漏洞检测进行研究。本文针对深度学习漏洞检测模型不能提供细粒度漏洞解释且缺少细粒度标签的问题，基于分层注意力网络和积分梯度方法提出了一种细粒度漏洞解释方法。该方法不需要语句级标签，即可提供粗粒度的漏洞检测和细粒度的漏洞解释。在真实以太坊数据集上进行了实验，结果表明该方法可以有效提供细粒度漏洞解释，定位脆弱性代码语句。在未来的工作中，将结合动态特征扩展模型的特征表示空间，对细粒度漏洞检测模型的语句表示的获取进行改进，在检测准确率和解释精度之间实现更加均衡的性能。

参 考 文 献

[1] NAKAMOTO S. Bitcoin A Peer-to-Peer Electronic Cash System

[J/OL]. https://bitcoin.org/bitcoin.pdf.

[2] BUTERIN V. A next Generation Smart Contract & Decentralized Application Platform [J/OL]. https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf.

[3] SZABO N. Smart Contracts: Building Blocks for Digital Markets [J/OL]. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html.

[4] ZOU W, LO D, KOCHHAR P S, et al. Smart Contract Development: Challenges and Opportunities[J]. IEEE Transactions on Software Engineering, 2021, 47: 2084.

[5] SCHAR F. Decentralized Finance: On Blockchain and Smart Contract-Based Financial Markets[J]. Federal Reserve Bank of St, 2020, 103(2): 153-174.

[6] ZHANG Y, KASAHARA S, SHEN Y, et al. Smart Contract-Based Access Control for the Internet of Things[J]. IEEE Internet of Things Journal, 2018, 6: 1594-1605.

[7] DUAN B, XIN K, ZHONG Y. Optimal Dispatching of Electric Vehicles Based on Smart Contract and Internet of Things[J]. IEEE Access, 2020, 8: 9630-9639.

[8] ATZEI N, BARTOLETTI M, CIMOLI T. A Survey of Attacks on Ethereum Smart Contracts (SoK) [C]// 6th International Conference on Principles of Security and Trust (POST) Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS). 2017: 164-186.

[9] FAIRYPROOF. Fairyproof's Review of 2021 Blockchain Security [EB/OL]. https://fairyproof.com/doc/Fairyproof'sReviewOf2021BlockchainSecurity_012722.pdf.

[10] CHAKRABORTY S, KRISHNA R, DING Y, et al. Deep Learning based Vulnerability Detection: Are We There Yet[J]. IEEE Transactions on Software Engineering, 2021, 48(1): 3280-3296.

[11] LI Z, ZOU D, XU S, et al. VulDeeLocator: A Deep Learning-Based Fine-Grained Vulnerability Detector[J]. IEEE Transactions on Dependable and Secure Computing, 2022, 19(4): 2821-2837.

[12] LI Y, WANG S, NGUYEN T N. Vulnerability detection with fine-grained interpretations[C]// Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021: 292-303.

[13] NGUYEN V A, LE T, TANTITHAM C K, et al. An Information-Theoretic and Contrastive Learning-based Approach for Identifying Code Statements Causing Software Vulnerability [J]. arXiv: 2209.10414, 2022.

[14] 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). 2022.

[15] SHEN C K. Research on Deep Learning-based Vulnerability Detection Methods for Smart Contracts[D]. Wuhan: Wuhan University, 2021.

[16] YANG Z, YANG D, DYER C, et al. Hierarchical Attention Networks for Document Classification[C]// Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2016: 1480-1489.

[17] FERREIRA J F, CRUZ P, DURIEUX T, et al. SmartBugs: A Framework to Analyze Solidity Smart Contracts [C]// 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2020: 1349-1352.

[18] GHALEB A, PATTAB K. How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection [C]// Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020: 415-427.

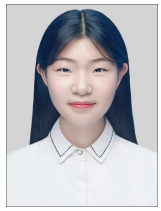
[19] LUU L, CHU D H, OLICKEL H, et al. Making Smart Contracts Smarter[C]// 23rd ACM Conference on Computer and Communications Security (CCS). 2016: 254-269.

[20] MUELLER B. Mythril-Reversing and bug hunting framework for the Ethereum blockchain [Z]. https://pypi.org/project/mythril/.

[21] 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.

[22] GU J, WANG Z, KUEN J, et al. Recent advances in convolutional neural networks[J]. arXiv: 1512.07108, 2015.

[23] HOCHREITER S, SCHMIDHUBER J. Long Short-Term Memory[J]. Neural Computing, 1997, 9(8): 1735-1780.



LI Qiuyue, born in 1998, postgraduate. Her main research interests include information security and blockchain.

HAN Daojun, born in 1979, Ph.D, professor, is a member of CCF (No. 28531). His main research interests include information security and blockchain.

（责任编辑：何杨）