基于抽象语法树裁剪的智能合约漏洞检测研究

刘泽润，郑红，邱俊杰

引用本文

刘泽润，郑红，邱俊杰.基于抽象语法树裁剪的智能合约漏洞检测研究[J]．计算机科学, 2023, 50(4): 317-322.

LIU Zerun, ZHENG Hong, QIU Junjie. Smart Contract Vulnerability Detection Based on Abstract Syntax Tree Pruning [J]. Computer Science, 2023, 50(4): 317-322.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220300063

基于抽象语法树裁剪的智能合约漏洞检测研究

刘泽润　郑　红　邱俊杰
华东理工大学信息科学与工程学院　上海 200237
(zerunliu@qq.com)

摘　要　随着区块链技术的发展,智能合约在不同领域都得到了广泛的应用,以太坊成为了最大的智能合约平台。同时,频发的智能合约漏洞造成了巨大的经济损失,智能合约漏洞检测成为了研究焦点,而以往的智能合约漏洞检测工具不能很好地利用合约源代码的语法信息。针对智能合约的可重入漏洞,首先,提出了一种基于深度学习的漏洞检测工具——SCDefender,以智能合约 Solidity 源代码的抽象语法树形式作为研究对象,使用基于树的卷积神经网络进行漏洞检测。其次,提出了抽象语法树裁剪算法以去除与漏洞检测任务无关的节点,保留抽象语法树中的关键信息。SCDefender 漏洞检测的精确度、召回率和 $F1$ 值分别为 81.43%,92.12%和 86.45%,具有较好的漏洞检测效果。消融实验表明,抽象语法树裁剪算法对 SCDefender 的漏洞检测任务具有重大贡献。

关键词:区块链;智能合约;漏洞检测;抽象语法树;深度学习
中图法分类号　TP309

Smart Contract Vulnerability Detection Based on Abstract Syntax Tree Pruning

LIU Zerun,ZHENG Hong and QIU Junjie
School of Information Science and Engineering,East China University of Science and Technology,Shanghai 200237,China

Abstract　With the development of blockchain technology,smart contracts have been widely used in various fields,and Ethereum has become the largest smart contract platform. At the same time,the frequent smart contract vulnerabilities have caused huge economic losses. The vulnerability detection of smart contract has become the focus of research,while the previous smart contract vulnerability detection tools can not make good use of the syntax information of the contract source code. Aiming at the re-entrancy vulnerability of smart contract,firstly,this paper proposes SCDefender,a vulnerability detection tool based on deep learning. Taking the abstract syntax tree form of the Solidity source code of smart contract as the research object,the tree-based convolutional neural networks is used for vulnerability detection. Secondly,an abstract syntax tree pruning algorithm is proposed to remove the nodes irrelevant to the vulnerability detection task and retain the key information in the abstract syntax tree. The accuracy,recall rate and $F1$ value of SCDefender vulnerability detection is 81.43%,92.12% and 86.45% respectively,which has a good vulnerability detection effect. Ablation experiments show that the abstract syntax tree pruning algorithm has an important contribution to the vulnerability detection task of SCDefender.

Keywords　Blockchain,Smart contract,Vulnerability detection,Abstract syntax tree,Deep learning

1　引言

智能合约(Smart Contract)[1]是 20 世纪 90 年代由计算机学家 Nick Szabo 提出的构想。但直到近年来,区块链技术[2]的诞生为其提供了一个去中心化平台,智能合约才得到了广泛应用。目前,智能合约体现为一段具有状态、由事件驱动、运行在区块链系统上的程序,用户人数最多的智能合约平台为以太坊(Ethereum)[3]。与此同时,针对智能合约发起的攻击也越来越多。智能合约作为一段自动执行的代码,一经发布就不可修改,且智能合约总是管理着大量的数字代币资产,攻击者对其进行攻击可以获得巨大的收益。2016 年,攻击者利用 The DAO 众筹合约的可重入漏洞对其发起攻击,导致了约 6000 万美元的损失[4-6]。2017 年,以太坊 Parity 电子钱包因多重签名漏洞而损失约 3000 万美元[7]。2018 年,美链公司发行的 BEC 代币合约出现整数溢出漏洞,导致其市值几乎归零[8]。

由于智能合约具有独特的运行环境、程序特性和生命周期,现有的软件缺陷检测工具难以直接应用于智能合约。针对智能合约的漏洞检测,越来越多的检测工具被提出。例如,KEVM framework[9]利用 K 框架对 EVM(Ethereum Virtual Machine)字节码进行形式化定义,采用形式化验证的方法进行程序分析,但其自动化程度不高,需要研究人员花费

到稿日期:2022-03-07　返修日期:2022-08-23
基金项目:国家自然科学基金(61472139);产学研项目:区块链关键技术研究(H300-41819)
This work was supported by the National Natural Science Foundation of China(61472139) and Industry University Research Project: Research on Key Technologies of Blockchain(H300-41819).
通信作者:郑红(zhenghong@ecust.edu.cn)

大量精力进行推理和建模工作。Maian[10]采用符号执行的方法，通过探索合约的调用路径进行漏洞检测，但其待探索的路径数量随分支状态的增加呈指数增长。ContractFuz-zer[11]采用模糊测试的方法，通过生成大量的测试用例来检测智能合约在运行时是否出现异常，但其会产生大量无用的测试用例，检测所花费的时间也较长。ConFuzzius[12]采用混合模糊测试的方法，将符号执行和模糊测试进行结合，以改善深层错误的检测效果。可见，当前智能合约的漏洞检测工具主要存在自动化程度较低、效率较低、检测时间较长等缺点。

近年来，随着深度学习领域的快速发展，使用深度学习技术对程序代码进行处理成为了研究热点。基于深度学习的代码漏洞检测技术可以弥补上述自动化程度低、效率低、检测时间长等不足。

为了更好地利用智能合约源代码的语法特征，本文提出了一个基于深度学习技术的智能合约漏洞检测工具——SCDefender，它将合约代码的抽象语法树形式作为研究对象，并针对智能合约可重入漏洞的特性进行裁剪优化，通过词嵌入方法将抽象语法树节点转化为向量，使用基于树的卷积神经网络(Tree-based Convolutional Neural Networks，TBCNN)[13]作为网络模型进行漏洞检测。结果显示，SCDefender 在检测过程中能更好地保留源代码的语法特征，并且在使用抽象语法树裁剪算法后其准确率、召回率、$F1$ 值分别提升了 6.94%，13.25%，9.83%。

本文的主要贡献包括：

(1)创新性地引入基于树的卷积神经网络对智能合约进行漏洞检测，以保留合约源代码的语法特征，减少序列化过程中的信息损失。

(2)提出了抽象语法树裁剪算法，以减少抽象语法树中与漏洞检测任务无关的节点，保留关键信息以改善检测效果。

2 相关工作与背景知识

2.1 基于机器学习的智能合约漏洞检测

目前，使用机器学习方法进行智能合约漏洞检测的相关工作并不多。ContractWard[14]将智能合约的操作码作为研究对象，使用 N-Gram 算法提取特征，并使用 XGBoost、随机森林等机器学习算法检测智能合约漏洞。Eth2Vec[15]同样将智能合约的操作码转化为向量，并使用神经网络进行漏洞检测，但将智能合约转化为操作码会损失过多的源代码语义信息。DR-GCN[16]提取智能合约中的控制流和数据流，将合约转化为图神经网络进行漏洞检测，但其只考虑了 3 种类型的节点，且构图过程较为复杂。SmartEmbed[17]使用深度学习模型计算待检测合约与漏洞库中的合约之间的相似度，将相似度超过一定阈值的待检测合约视为含有漏洞的合约，但其漏洞库中的合约漏洞模板较少，可能会漏掉书写形式与其差别较大的含漏洞的待检测合约。Peculiar[18]抽取智能合约的关键数据流，并使用预训练模型进行漏洞检测，但将合约转化为数据流会损失源代码的语法结构信息。

2.2 抽象语法树

抽象语法树(Abstract Syntax Tree，AST)是程序语言的中间表示，它将程序代码抽象为一种树状结构。相较于源代码和控制流图，抽象语法树可以直观地表示代码的语法结构。除此之外，因为程序的源代码中存在大量的用户自定义变量，若直接以程序的源代码训练词向量将导致词嵌入矩阵维度过大，而抽象语法树的节点类型是固定的，它可以将用户自定义的变量转化为固定的表现形式，其本身就可以解决词表爆炸问题。由于编写智能合约的 Solidity 语言[19]本身是一种强数据类型的语言，相较于 PHP，Python 等弱数据类型语言，其语法规则和数据类型更加严格和规范，将程序转化为抽象语法树可以更好地保留源代码的上下文语义信息。

3 研究方法

SCDefender 的整体检测流程如图 1 所示。首先，将智能合约的 Solidity 源代码转化为抽象语法树，对抽象语法树进行层次遍历以获得语料库，并采用 Word2Vec[20]获得每个节点类型的向量表示；然后根据可重入漏洞的特点对抽象语法树进行裁剪；最后将抽象语法树向量化，并使用基于树的卷积神经网络[13]进行漏洞检测。

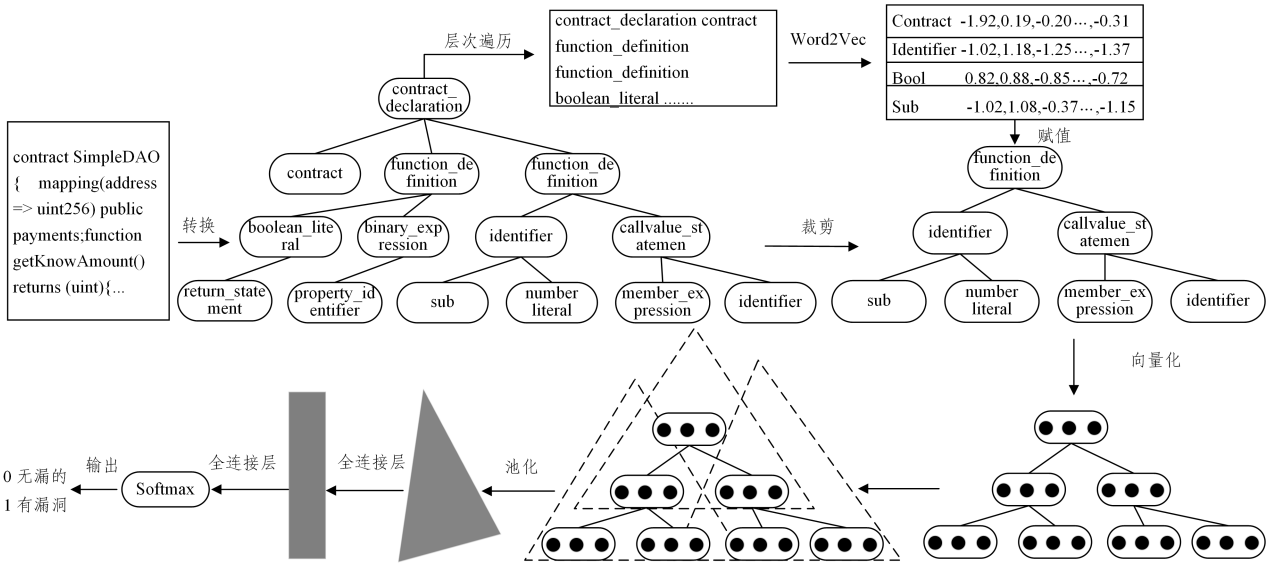

图 1 SCDefender 检测流程

Fig. 1 Detection process of SCDefender

3.1 抽象语法树的裁剪

虽然将源代码转化为抽象语法树有诸多优点，但抽象语法树总是包含过多的节点，这会导致关键信息被其他众多的无关节点隐藏，不利于后期使用深度学习检测合约漏洞。对于可重入漏洞，其关键语句在于 *call.value* 函数的调用，但仅依靠判断是否含有 *call.value* 函数不足以确定合约是否含有可重入漏洞。因此，我们将 *call.value* 函数及余额扣除语句作为关键信息对抽象语法树进行裁剪，从而避免无关信息的干扰。

SCDefender 使用开源工具 tree-sitter-solidity[1)] 将智能合约 Solidity 代码转化为抽象语法树。在研究过程中发现，由于该工具本身不够成熟，tree-sitter-solidity 会将 *call.value* 函数视作 msg 的普通成员变量，并将其转化为 *property_identifier* 节点类型，而忽视了 *call.value* 函数应是一个系统函数。为解决此问题，我们完善了该工具的语法规则，添加了 *call.value* 和 *call.value_statement* 节点类型。

将源代码转化为抽象语法树之后，需要设定关键节点作为抽象语法树裁剪算法的输入。对于可重入漏洞，因账户余额扣除语句总是位于 *call.value* 函数的周边，所以 SCDefender 将关键节点设置为 *call.value_statement*。使用算法 1 递归地遍历抽象语法树，在找到关键节点 K 之后，继续寻找关键节点所在函数的定义节点。最终，返回该定义节点，并将其作为关键抽象语法树的根节点，此抽象语法树将包含 *call.value* 函数及账户余额扣除语句的信息。

算法 1 抽象语法树的裁剪算法

输入：(R,K)

输出：R_c

```
1. 初始化：Fk←false，Ff←false，T←R/*Fk用来判断是否已找到关键节点，Ff用来判断是否已找到关键节点所在的函数，R为根节点*/
2. functionast_pruning(T)：
3.     if Ff=True then/*若找到关键节点所在函数，将不再继续递归*/
4.         return
5.     end if
6.     if node=K then/*判断当前节点是否是关键节点*/
7.         Fk←True
8.         return
9.     end if
10.    V←T.childs/*获得T的子节点队列*/
11.    for v in V do
12.        ast_pruning(v)/*递归地寻找关键节点*/
13.    end for
14.        if node=function_definition and Fk=True then/*寻找关键节点所在函数*/
15.            Rc←node
16.            Ff←True
17.            return
18.    end if
19. end function
```

3.2 节点的向量表示

抽象语法树并不能直接作为神经网络的输入，需要将树的节点转换为向量。传统的方法使用 one-hot 编码的方式，将每种节点类型和一个整数建立映射，但这种方法忽视了节点之间的语义联系，并且向量的长度会随着单词长度的增加而不断增加，不利于深度学习模型的训练。为了使节点含有较为丰富的语义信息，本研究使用 Word2Vec[20] 对节点进行向量化。同一深度的节点总是含有相似的语义，它们在层次遍历生成的序列中的位置也是相近的，这有利于词嵌入模型学习到节点之间的语义联系，因此我们使用算法 2 基于层次遍历的策略对抽象语法树进行序列化。除此之外，因为 Solidity 语言的数据类型较为严格，其所包含的 int，uint，bytes 等基本数据类型还根据取值范围的不同存在多种类型，所以抽象语法树也含有这些额外的节点类型。例如，bytes 类型除自身外还存在 bytes1-bytes32 类型。这些节点类型的差异对于漏洞检测任务没有帮助，因此 SCDefender 不考虑它们的取值范围，统一将这些节点化简为 int，uint，bytes 类型。SCDefender 使用 265 种节点类型，部分节点类型如表 1 所列。

算法 2 抽象语法树转序列算法

输入：(R,C_1,C_2,C_3)

输出：L

```
1. 初始化：L←Ø，Q←Ø
2. add R into L/*R为抽象语法树的根节点*/
3. add R into Q
4. while Q≠Ø do
5.     node←Q.pop/*移出队列Q的队头节点，并赋值给node*/
6.     V←node.childs/*获得node的子节点队列*/
7.     for v in Vdo
8.         if v∈C1 then/*判断v是否属于int类型家族*/
9.             v←int
10.        else if v∈C2 then/*判断v是否属于uint类型家族*/
11.            v←uint
12.        else if v∈C3 then/*判断v是否属于bytes类型家族*/
13.            v←bytes
14.        end if
15.        add v into L
16.        add v intoQ
17.    end for
18. end while
```

表 1 节点类型示例

Table 1 Example of node types

Category	Node types
Declaration	variable_declaration，interface_declaration，constant_variable_declaration
Arithmetic	sub，div，eq，add，mul，++，+=，/=，*=，
Intrinsics	yul_assignment，yul_decimal_number，yul_function_call，yul_evm_builtin
Control	if_statement，for_statement，do_while_statement，break_statement

1) https://github.com/JoranHonig/tree-sitter-solidity

3.3 可重入漏洞的检测

相比循环神经网络(Recurrent Neural Network, RNN)、长短期记忆网络(Long Short-Term Memory, LSTM)等序列神经网络结构,基于树的卷积神经网络[13]能更好地利用抽象语法树的结构信息。因此,SCDefender 利用基于树的卷积神经网络进行漏洞检测。传统的 TBCNN 将整棵抽象语法树作为研究对象,而本文对 3.1 节得到的关键抽象语法树进行卷积,将抽象语法树中的每个节点替换为对应的特征向量,完成抽象语法树的向量化。TBCNN 采用固定深度的树状滑动窗口,并在抽象语法树上滑动以提取整体特征。因为不同的 Solidity 智能合约代码转化后的抽象语法树形状不同,每一个非叶子节点的子节点数量也不固定,所以采用模拟二叉树[13]的方法,设立$\boldsymbol{W}_t$,$\boldsymbol{W}_l$,$\boldsymbol{W}_r$权重矩阵计算权重。对于一个包含 n 个节点的滑动窗口,假设这 n 个节点的特征向量为$\boldsymbol{x}_1,\cdots,\boldsymbol{x}_n$,则卷积运算的输出向量为:

$$\boldsymbol{y}=\mathrm{LReLU}\left(\sum_{i=1}^{n}\left[n_i^l\boldsymbol{W}^l+n_i^r\boldsymbol{W}^r+n_i^t\boldsymbol{W}^t\right]\boldsymbol{x}_i+b\right) \tag{1}$$

其中,$\boldsymbol{x}_i\in\mathbb{R}^D$,$b\in\mathbb{R}^C$,$\boldsymbol{W}^l$,$\boldsymbol{W}^r$,$\boldsymbol{W}^t\in\mathbb{R}^{C\times D}$,$D$ 为节点的特征向量的维度,C 为经过卷积操作的特征向量的维度。

权重系数n_i^l,n_i^r,n_i^t计算式如下:

$$n_i^t=\frac{d_i-1}{d-1} \tag{2}$$

$$n_i^r=(1-n_i^t)\frac{p_i-1}{n-1} \tag{3}$$

$$n_i^l=(1-n_i^t)(1-n_i^r) \tag{4}$$

其中,d 代表滑动窗口的深度,d_i 代表节点 i 在滑动窗口中的深度;n 代表节点的兄弟总数;p_i 代表节点 i 在滑动窗口中的位置。

抽象语法树经过卷积运算后,每一个节点的向量表示也包含了其子节点的信息,整棵语法树包含了丰富的结构语义信息。为了将提取到的特征输入到固定大小的神经网络层,采用最大池化收集树节点的信息,提取不同节点特征中每个维度的最大值并集中到一个向量中,以生成一个表示整棵抽象语法树的特征向量。最后,添加隐藏层,将特征向量作为输入,并使用 softmax 分类器对合约进行分类。使用二分类交叉熵函数作为损失函数,并使用 L2 正则化防止过拟合。

$$p=\mathrm{softmax}(\boldsymbol{W}h+b) \tag{5}$$

$$L=-\frac{1}{N}\sum_{i}^{N}\left[y_i\log(p_i)+(1-y_i)\log(1-p_i)\right]+\lambda\sum_{i}^{N}w_i^2 \tag{6}$$

4 实验

本节将在公开数据集上进行测试,并从多个角度对 SCDefender 的有效性进行分析。使用 Pytorch 平台搭建神经网络模型,在配有 Linux 操作系统、Intel Xeon 架构处理器、Tesla P100 16 GB 的服务器下运行。使用精确度(Precision)、召回率(Recall)、F1 值(F1-score)作为评价指标来衡量模型的性能。

$$Precision=\frac{TP}{TP+FP} \tag{7}$$

$$Recall=\frac{TP}{TP+FN} \tag{8}$$

$$F1\text{-}score=2\times\frac{Precision\times Recall}{Precision+Recall} \tag{9}$$

其中,TP 表示实际有漏洞同时被预测为有漏洞的合约数量;FP 表示实际无漏洞,但被预测为有漏洞的合约数量;FN 表示实际有漏洞,但被预测为无漏洞的合约数量。

4.1 词嵌入的训练

大型语料库有助于词嵌入模型的训练,本研究使用 SmartBugs Wild Dataset[21]作为语料库,该数据库里的智能合约数据均从以太坊网络获取,它包含 47 398 个无重复的 sol 格式的文件,共计 203 716 个智能合约。本研究使用 CBOW 算法[20]进行训练,将窗口大小设置为 5,词频阈值设置为 20,词向量大小设置为 60。为了直观地展示词嵌入模型的学习效果,使用 T-SNE 算法[22]将词向量维度降至 2 维以进行可视化展示。

图 2 给出了词向量降维后的部分节点。从图中可以观察到,相同类型的节点在图中的位置也是相近的,例如“>=”“<=”“==”“+”“!=”等算术类型、“yul_identifier”“yul_function_call”“yul_evm_builtin”等内联类型、“if_statement”“while_statement”“for_statement”等控制类型的内部之间的位置是相互接近的。这说明学习到的词向量可以有意义地表示不同的节点类型,为深度学习模型进行漏洞检测任务奠定了基础。

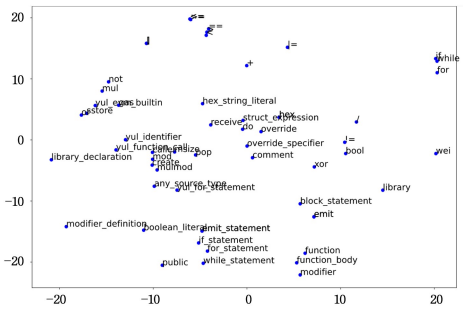

图 2 节点的词嵌入示例

Fig. 2 Example of node embedding

4.2 模型对比实验

本文使用 SCDefender 与其他模型和工具进行对比实验。使用的数据集为 SmartBugs Wild Dataset,文献[18]为其标注了标签。但我们在研究过程中发现,部分合约只存在空格和制表符之间的差别,这可能导致实验结果虚高,于是我们将其标注为相同的合约并进一步去重。将 20% 的数据设置为训练集,对训练集重采样,使正负样本平衡。训练过程中,将学习率设置为10^{-3},正则化参数设置为10^{-4}。实验结果如表 2 所列。

表 2 不同检测方法的实验结果对比

Table 2 Comparison of experimental results of different detection methods

Method	*Precision*	*Recall*	*F1-score*
Smartcheck	0.6134	0.4499	0.5191
RNN	0.1993	0.2287	0.2130
Bi-RNN	0.2154	0.2475	0.2303
LSTM	0.7336	0.8712	0.7965
Bi-LSTM	0.7529	0.8837	0.8131
SCDefender	0.8143	0.9212	0.8645

Smartcheck[23]采用中间表示法,将 Solidity 源代码转化

为XML格式的表现形式，并使用Xpath模式进行漏洞检测，但在实验中产生了大量的误报，其$F1$值为51.91%。在进行序列型神经网络实验时，我们对抽象语法树进行先序遍历以生成序列作为输入，均采用抽象语法树裁剪算法，将词向量维度设置为100，这样做的原因是序列型神经网络在较大的词向量维度下效果较好。从实验结果中可以发现，RNN和Bi-RNN的检测结果很差，$F1$值分别为21.30%和23.03%，这是由于RNN和Bi-RNN在网络训练过程中会发生梯度消失现象，因此其只能学习到局部区域的依赖关系。而LSTM和Bi-LSTM因为引入了门控机制，使得输入序列中的关键信息可以一直向下传递而不会丢失，因此其可以学习到长期依赖关系，$F1$值分别为79.65%和81.31%。而使用基于树的卷积神经网络的SCDefender相对于序列型神经网络而言，其不需要将抽象语法树转化成序列，从而保留了抽象语法树的层次结构信息，减少了额外的语法信息的损失，其精确度、召回率、$F1$值分别为81.43%，92.12%，86.45%，取得了较好的检测结果。

4.3 消融实验

4.3.1 裁剪操作的影响

本小节将探究提出的抽象语法树裁剪算法对智能合约漏洞检测的影响。在SCDefender运行中不采用抽象语法树的裁剪算法进行实验。为了保证研究的规范性，所使用的实验环境、数据集和超参数与4.2节的实验保持一致。实验结果如表3所列。

表3 是否使用裁剪算法的消融实验结果对比

Table 3 Comparison of ablation experimental results with and without pruning algorithm

Method	*Precision*	*Recall*	*F1-score*
With AST pruning algorithm	0.8143	0.9212	0.8645
Without AST pruning algorithm	0.7449	0.7887	0.7662

相较于不使用抽象语法树的裁剪算法，SCDefender在使用裁剪算法后其准确率、召回率、$F1$值分别提升了6.94%，13.25%，9.83%。因此可以得出结论：抽象语法树经裁剪算法去除无用节点后，可以大幅度地提高智能合约的漏洞检测效果，对于SCDefender的漏洞检测具有重要贡献。

4.3.2 节点向量长度的影响

本节将探究节点的词向量长度对检测效果的影响，找到最佳的节点向量长度。以5为步长、40为起点、75为终点进行实验，使用的词嵌入算法为CBOW，窗口大小设置为5，词频阈值设置为20，所使用的实验环境、数据集和超参数和以上实验保持一致。实验结果如图3所示。

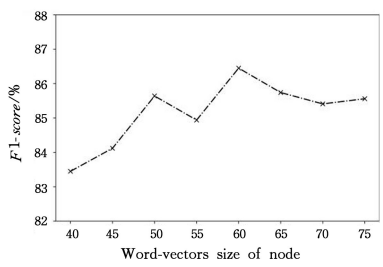

图3 节点的词向量长度的消融实验结果对比

Fig. 3 Comparison of ablation experimental results of word vectors size of node

可以看出，当词向量长度为60时，所得到的检测结果最好。当词向量的长度较短时，节点包含的信息较少，深度学习模型难以学习到足够的信息。当词向量的长度过长时，词嵌入矩阵较为稀疏，节点没有包含更多有用的信息，检测效果没有继续提升，并且会增加深度学习模型的训练时间。因此，SCDefender选取60作为节点的词向量长度。

结束语 本文针对以太坊智能合约的可重入漏洞进行研究，提出了一个检测工具。通过将智能合约Solidity程序代码转化为抽象语法树并进行向量化，使用基于树的卷积神经网络进行训练。针对可重入漏洞的特点，提出抽象语法树裁剪算法去除无用的节点，以提高检测效果。本文还探究了不同长度的词向量对检测效果的影响，选取了最佳的节点向量长度。实验表明，此工具可以有效地检测智能合约的可重入漏洞，并且本研究提出的抽象语法树裁剪算法可以大幅度地提升检测效果。

目前，带标签智能合约的数据集数量贫乏，严重阻碍了智能合约漏洞检测领域的发展，本文工作也依赖于先前研究人员对数据集的标注，只针对可重入漏洞进行检测。未来，我们将重点研究智能合约其他漏洞种类的收集及标注工作，并将本文提出的检测方法扩展至其他的漏洞类型。期待本文工作能为新的智能合约漏洞检测方法提供思路和参考，保护区块链生态安全的持续健康发展。

参 考 文 献

[1] SZABO N. Smart contracts: building blocks for digital markets [J]. EXTROPY: The Journal of Transhumanist Thought, 1996, 16(18): 2-20.

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

[3] WOOD G. Ethereum: A secure decentralised generalised transaction ledger[J]. Ethereum Project Yellow Paper, 2014, 151(2014): 1-32.

[4] SIEGEL D. Understanding the dao attack[EB/OL]. https://www.coindesk.com/understanding-dao-hack-journalists.

[5] MEHAR M I, SHIER C L, GIAMBATTISTA A, et al. Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack [J]. Journal of Cases on Information Technology (JCIT), 2019, 21(1): 19-32.

[6] ATZEI N, BARTOLETTI M, CIMOLI T. A survey of attacks on ethereum smart contracts (sok) [C] // International Conference on Principles of Security and Trust. Berlin: Springer, 2017: 164-186.

[7] DESTEFANIS G, MARCHESI M, ORTU M, et al. Smart contracts vulnerabilities: a call for blockchain software engineering? [C] // 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE). IEEE, 2018: 19-25.

[8] SUN J, HUANG S, ZHENG C, et al. Mutation testing for integer overflow in ethereum smart contracts[J]. Tsinghua Science and Technology, 2021, 27(1): 27-40.

[9] TIAN F. A supply chain traceability system for food safety based on HACCP, blockchain & Internet of things[C] // 2017 International Conference on Service Systems and Service

Management. IEEE, 2017: 1-6.

[10] NIKOLIĆ I, KOLLURI A, SERGEY I, et al. Finding the greedy, prodigal, and suicidal contracts at scale[C]// Proceedings of the 34th Annual Computer Security Applications Conference. 2018: 653-663.

[11] JIANG B, LIU Y, CHAN W K. Contractfuzzer: Fuzzing smart contracts for vulnerability detection[C]// 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2018: 259-269.

[12] TORRES C F, IANNILLO A K, GERVAIS A, et al. ConFuzzius: A Data Dependency-Aware Hybrid Fuzzer for Smart Contracts[C]// 2021 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, 2021: 103-119.

[13] MOU L, LI G, ZHANG L, et al. Convolutional neural networks over tree structures for programming language processing[C]// Thirtieth AAAI Conference on Artificial Intelligence. 2016: 1287-1293.

[14] 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, 2020, 8(2): 1133-1144.

[15] 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.

[16] ZHUANG Y, LIU Z, QIAN P, et al. Smart Contract Vulnerability Detection using Graph Neural Network[C]// IJCAI. 2020: 3283-3290.

[17] GAO Z, JAYASUNDARA V, JIANG L, et al. Smartembed: A tool for clone and bug detection in smart contracts through structural code embedding[C]// 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2019: 394-397.

[18] WU H, ZHANG Z, WANG S, et al. Peculiar: Smart Contract Vulnerability Detection Based on Crucial Data Flow Graph and Pre-training Techniques[C]// 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE). IEEE. 2021: 378-389.

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

[20] MIKOLOV T, CHEN K, CORRADO G, et al. Efficient estimation of word representations in vector space[J]. arXiv: 1301.3781, 2013.

[21] FERREIRA J F, CRUZ P, DURIEUX T, et al. SmartBugs: a framework to analyze solidity smart contracts[C]// Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. 2020: 1349-1352.

[22] VAN DER MAATEN L, HINTON G. Visualizing data using t-SNE[J]. Journal of Machine Learning Research, 2008, 9(11): 2579-2605.

[23] TIKHOMIROV S, VOSKRESENSKAYA E, IVANITSKIY I, et al. Smartcheck: Static analysis of ethereum smart contracts [C]// Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain. 2018: 9-16.

LIU Zerun, born in 1998, postgraduate, is a member of China Computer Federation. His main research interests include blockchain and deep learning.

ZHENG Hong, born in 1973, Ph.D, associate professor, postgraduate supervisor, is a member of China Computer Federation. Her main research interests include blockchain and deep learning.

(责任编辑:何杨)