DOI:10.12379/j.issn.2096-1057.2022.01.05

基于抽象语法树压缩编码的漏洞检测方法

陈传涛　潘丽敏　龚　俊　马　勇　罗森林

（北京理工大学信息系统及安全对抗实验中心　北京　100081）

（chencht163@163.com）

The Vulnerability Detection Method Based on Compression Coding of Abstract Syntax Tree

Chen Chuantao, Pan Limin, Gong Jun, Ma Yong, and Luo Senlin

(*Information System and Security & Countermeasures Experimental Center, Beijing Institute of Technology, Beijing* 100081)

Abstract　In source code vulnerability detection method based on abstract syntax tree, it is difficult to fully extract the syntax and structure features from the large-scale syntax tree, which lead to the problem of insufficient capability of vulnerability characterization and low detection accuracy. In response to above problem, a method for source code vulnerability detection which based on the abstract syntax tree compressed coding (ASTCC) is proposed. Firstly, the abstract syntax tree is divided into a group of subtrees by code statements, and then the subtrees are encoded by recursive neural network to extract the syntax information of code statements. Then, the subtree of the original syntax tree is replaced with its encoding node to reduce the depth and the number of leaf nodes of the original syntax tree while retaining the structural features. Finally, the tree based convolutional neural network with attention mechanism is used to detect source code vulnerabilities. Experimental results on NVD and SARD open datasets show that the ASTCC method reduced the size of the abstract syntax tree, enhanced the model's ability to represent source code vulnerabilities, and effectively improved the accuracy of vulnerability detection.

Key words　vulnerability detection; abstract syntax tree; tree based convolutional neural network; attentional mechanism; structure features

摘　要　针对基于抽象语法树的源代码漏洞检测方法难以从大规模语法树中充分提取语法和结构特征，导致漏洞表征能力不足、检测准确率低的问题，提出了一种基于抽象语法树压缩编码(abstract syntax tree compressed coding, ASTCC)的源代码漏洞检测方法.该方法首先将程序抽象语法树以代码语句为单元分割成 1 组子树；然后通过递归神经网络对子树进行编码以提取代码语句内语法信息；再将原始语法树中的子树替换为其编码节点，从而在保留结构特征的同时减小原始语法树的深

收稿日期：2021-05-27

基金项目：国家 242 信息安全计划项目(2019A021)；工信部 2020 年信息安全软件项目(CEIEC-2020-ZM02-0134)

引用格式：陈传涛，潘丽敏，龚俊，等. 基于抽象语法树压缩编码的漏洞检测方法[J]. 信息安全研究，2022，8(1)：35-42

度并减少了叶子节点数量；最后，通过带注意力机制的树卷积神经网络实现源代码漏洞检测.在NVD和SARD公开数据集上的实验结果表明，ASTCC方法能够降低抽象语法树的规模，增强模型对源代码漏洞的表征能力，有效提升漏洞检测的准确率.

关键词　漏洞检测；抽象语法树；树卷积神经网络；注意力机制；结构特征

中图法分类号　TP309

开源软件是当前信息技术产业的重要支撑，同时也成为攻击者的重点攻击目标[1].由于开源软件具有透明、开放、协作开发等特性，而开发人员的技术水平和安全意识参差不齐，导致开源软件自身容易存在漏洞代码.此外，攻击者还可以直接向开源软件注入恶意代码.因此，研究高效能源代码漏洞检测方法，对保护开源软件稳定运行、提升信息系统安全防御能力具有重要意义.

源代码漏洞检测技术是发现软件安全漏洞的主要手段之一，近年来利用深度学习技术检测源代码漏洞成为研究热点.抽象语法树(abstract syntax tree，AST)是程序语法结构的一种抽象化表示，是源代码在编译过程的中间产物，用以分析检查源代码的语法语义是否正确.编译器在生成AST时一般需要经过词法分析、语法分析和语义分析3个步骤：1)编译器首先对源代码字符流进行扫描，识别词法单元，并提取Token序列；2)在语法分析阶段，Token序列被分解成各类语法短语，并生成语法树；3)语义分析是编译过程的逻辑检查阶段，主要任务是对结构上正确的源代码进行语义审查，包括上下文相关性审查、Token类型匹配审查、类型转换等.现有基于抽象语法树的方法多将完整抽象语法树作为深度学习模型的输入，然而漏洞代码解析的抽象语法树通常规模较大，而语法、结构相关节点在树中上下不相邻，导致模型难以有效提取语法树中的特征信息.因此，如何从大规模语法树中学习源代码漏洞模式，实现高性能漏洞检测模型，是当前源代码漏洞检测课题的研究重点.

针对上述问题，本文提出一种结合程序语法和结构特征的源代码漏洞检测方法.通过对抽象语法树进行压缩编码，在保留语法和结构信息的同时降低树的复杂度；采用递归神经网络算法提取代码语句中的语法特征；通过树卷积神经网络捕捉代码语句间的结构信息；利用注意力机制生成程序向量表示，增强模型对程序语法和结构信息的表征能力，提高检测准确率.

1　相关工作

针对基于机器学习算法的漏洞检测在学术界已有大量相关研究工作.VulDeePecker[2]是最早应用深度学习算法的源代码漏洞检测方法之一，其灵感来源于自然语言处理技术.该方法将源代码视为文本序列，并采用基于序列的深度学习算法构建漏洞检测模型.然而，该方法忽略了程序语法和结构特征，导致模型对漏洞的表征能力不足，检测准确率低.对此，有大量工作针对源代码的抽象语法树构建漏洞检测模型[3-9].

不同于自然语言，程序编程语言具有复杂的语法和结构特征，如循环、嵌套、分支等，以及复杂的数据流与控制流等.Dam等人[3]提出一种基于长短时记忆模型的代码语义和语法特征自动学习方法，该方法从抽象语法树提取Token序列，然后构建LSTM模型自动学习代码语义和语法特征，挖掘潜在漏洞模式.实验结果表明，在22个跨项目漏洞检测案例中，平均查全率达到80%，证明该方法能够有效地检测实际漏洞.Mou等人[10]提出一种用于代码特征学习的树卷积神经网络(tree-based convolutional neural networks，TBCNN)，该方法可以直接在抽象语法树上卷积计算，捕捉程序语法和结构特征.Liang等人[6]在TBCNN的基础上，提出一种利用深度学习方法提取漏洞词汇特征和语法特征的漏洞定位方法，该方法通过将语义相似的语法实体进行分组以减少抽象语法树的冗余单元.然而，有研究指出，基于树结构的神经网络模型容易出现梯度消失的问题.由于树中语义相关节点之间的距离可能较远，导致模型在训练过程中梯度逐渐变得很小，尤其是当树的深度较大、叶子节点过多时梯度消失问题更加明显[11].

为了缓解梯度消失问题，Zhang 等人[11]提出一种基于抽象语法树的神经网络算法(AST-based neural network，ASTNN)，该方法将抽象语法树分解为 1 组较小的子树，然后分别对每个子树进行特征学习，再将子树的向量表示进行拼接得到程序向量表示.这种方法虽然能够有效缓解因树深度过大而导致的梯度消失问题，但其向量拼接方法难以保留不同子树间所含有的程序结构信息.

综上所述，现有源代码漏洞检测方法存在如下问题：1)基于代码语义的漏洞检测方法将源代码文本序列作为输入并采用序列模型学习代码语义信息，该方法关注于代码文本的语义信息而忽略了程序语法和结构信息，无法建立有效的漏洞特征表示方法；2)基于语法和结构特征的漏洞检测方法针对程序完整的抽象语法树进行特征学习，由于程序代码的抽象语法树规模通常较大，且代码语义相关节点在树中不相邻，导致模型难以有效提取漏洞特征，检测准确率低.

2 结合程序语法和结构特征的源代码漏洞检测方法

2.1 原理框架

针对大规模语法树的语法和结构特征提取不充分的问题，本文提出基于抽象语法树简化的源代码漏洞检测方法.该方法原理如图 1 所示，主要包括数据预处理、语句嵌入网络、程序嵌入网络和分类器 4 个模块.为充分提取代码语法和结构信息，首先将原始抽象语法树以代码语句为单元分割成多个子树，并将子树压缩为单个节点后按原始抽象语法树的结构生成新树.在子树和新树的基础上构建语句嵌入和程序嵌入 2 层网络.在语句嵌入网络中，采用递归神经网络(recursive neural network，RvNN)提取代码语句内的语法信息，生成语句向量表示；在程序嵌入网络中，利用树卷积神经网络算法捕捉代码语句间的结构信息，然后利用注意力机制生成程序向量.通过对抽象语法树的分割、聚合、重建，显著减小抽象语法树的深度并减少了节点数量，在保留程序语法和结构信息的同时捕捉节点语义信息，增强漏洞表征能力.

2.2 数据预处理

为了在不损失程序语法和结构特征前提下提取抽象语法树中的特征信息，需要对程序源代码进行数据预处理，包括源代码解析和抽象语法树简化.

2.2.1 源代码解析

本文提出的方法是一种函数级源代码漏洞检测方法，输入数据为函数代码段.使用开源工具 PyCparser[12]将源代码解析为抽象语法树，解析后

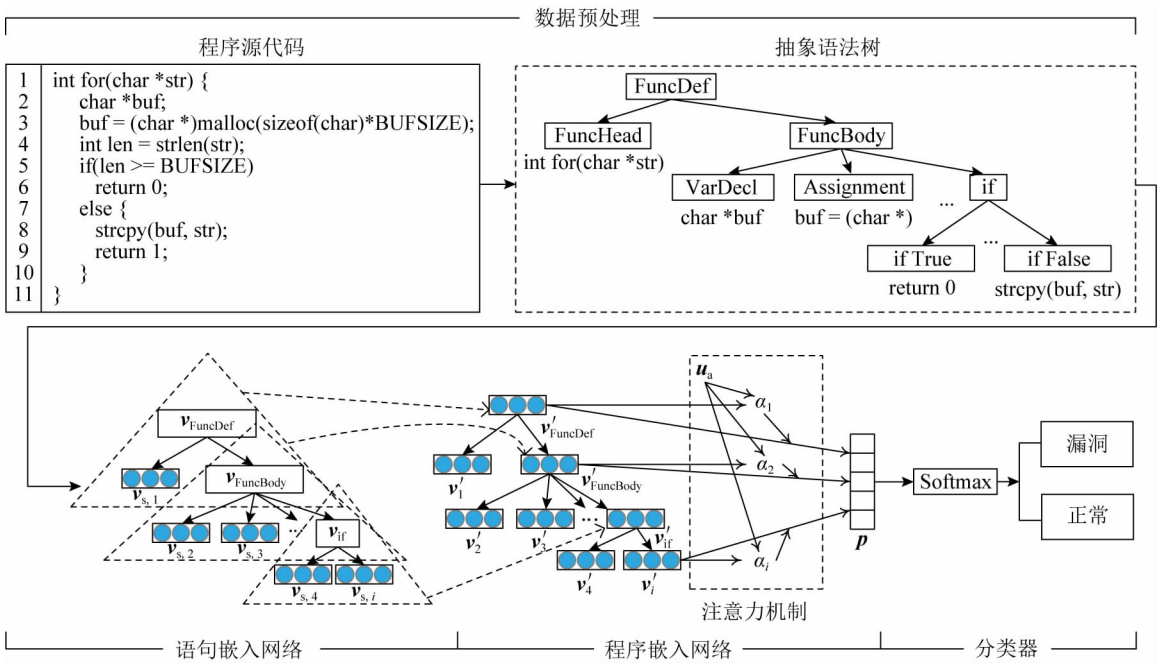

图 1 基于抽象语法树压缩编码的源代码漏洞检测方法原理图

1个函数对应1棵抽象语法树.以图1所示的程序源代码为例,其解析后得到的完整抽象语法树如图2上部分所示,为了简化表示,图2省略了部分代码语句所对应的子树.从图2可以看出,大约10行的源代码解析得到的抽象语法树中节点数多达50个以上.

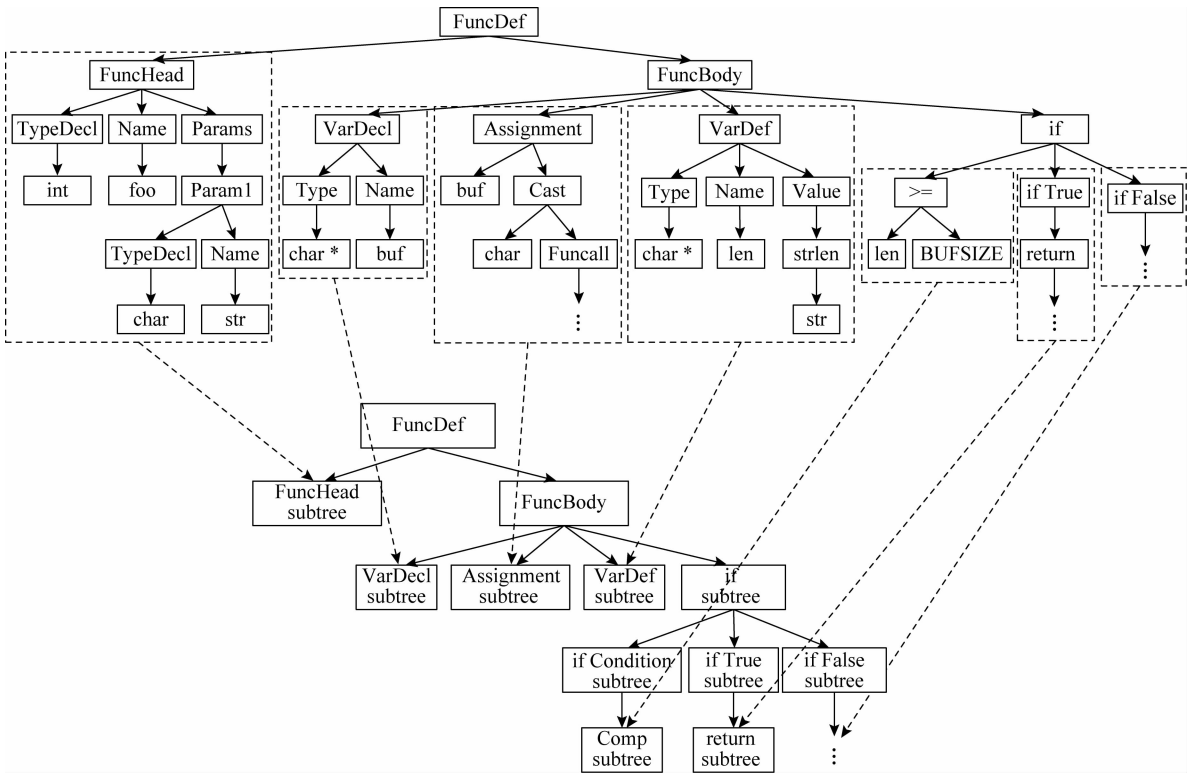

图2 抽象语法树简化过程

2.2.2 抽象语法树压缩编码

抽象语法树压缩编码如图2所示,图2的上半部分为原始抽象语法树,图2的下半部分为压缩编码后的语法树.具体包含如下3个步骤:

1)首先将原始抽象语法树以代码语句为单元分割成多个子树,如图2中虚线框所示.

2)使用RvNN算法学习子树中的语法信息生成语句嵌入向量,然后将子树压缩为单个节点,用语句嵌入向量表示,具体过程见2.3节.

3)根据语句子树在原始抽象语法树中的组织结构,将压缩后的节点重新构建为1棵新树,称为ASTCC(AST compressed encoding).与原始AST相比,ASTCC的深度和节点数量大大减小.此外,ASTCC中不仅保留了结构特征,其每个节点的向量表示中还含有语法信息.

2.3 语句嵌入网络

语句嵌入网络用于从1句源代码语句所对应的抽象语法树中捕捉代码语法信息,并得到语句嵌入向量.受ASTNN[11]启发,通过递归神经网络RvNN[13]构建语句嵌入网络.RvNN是针对自然语言的语法解析树的神经网络,不同于基于时间序列的训练神经网络(recurrent neural network,RNN),RvNN按树结构处理输入数据,提取空间结构特征.RvNN将树结构数据分解为一系列相同的基本单元,该基本单元在输入数据的结构上展开,并且沿着展开方向传递信息.因此,语句嵌入网络的结构随着输入数据的结构变化而变化,其中,RvNN神经元如图3所示:

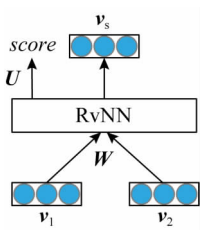

图3 RvNN神经元示意图

语法树中所有节点向量记为 v_i，其中 i 表示子节点的数量，初始化时使用节点上 Token 的词嵌入 v_t 作为初始值.父节点的向量表示 v_p 由其左右孩子节点所共同影响，计算公式：

$$v_p=\partial(W[v_1,v_2]^T+b), \tag{1}$$

其中 W 为权重矩阵，b 为偏置项，∂为激活函数.学习分数 $score$ 计算公式为

$$score=U^T p_i, \tag{2}$$

其中 U 为输出权重.为了加快训练速度，RvNN 中所有神经元共享权重 W 和 U.

语句嵌入网络在语法树上的训练过程由下向上，叶子节点信息传递到其父节点，层层递归直至根节点.根节点的向量表示中包含语句子树中所有节点的语法信息，称为语句嵌入向量，记为 v_s.

2.4 程序嵌入网络

程序嵌入网络采用基于树结构的卷积神经网络 TBCNN[10] 算法，以捕捉程序结构信息.不同于传统卷积神经网络（CNN），TBCNN 是一种在树结构数据上展开卷积计算，通过卷积核捕捉语法树的空间结构特征.网络结构如图 1 所示，图 1 中三角形虚线框为卷积核，其深度为固定值，广度随树结构变化而变化.

对于每个非叶子节点，其输出向量 v_p 由初始词嵌入 v_t 及子节点嵌入向量 v_s 所编码，计算公式为

$$v_p=W_t v_t+W_b\cdot\tanh\left(\sum_{c=1}^{N}\frac{N_c}{N}W_{n,c}v_{s,c}+b_n\right), \tag{3}$$

其中 N 为所有节点数量，N_c 为第 c 个节点的子节点数量，W_t 和 W_b 为组合权重，W_n 为所有子节点 $v_{s,c}$ 的系数权重，b_n 为偏置项，激活函数为 tanh.

卷积核在语法树中由下向上从叶子节点滑向根节点，每次滑动时更新非叶节点的值，记为 v'，计算方法为

$$v'=\tanh\left(\sum_{c=1}^{N_c}W_{conv,c}v_p+b_{conv}\right), \tag{4}$$

其中 $W_{conv,c}$ 为卷积权重矩阵，b_{conv} 为卷积核偏置项.更新所有节点信息后，使用注意力机制生成程序嵌入向量 p 的计算方法：

$$u_i=\tanh(W_a v'_i+b_a), \tag{5}$$

$$\alpha_i=\frac{\exp(u_i^T u_a)}{\sum_i\exp(u_i^T u_a)}, \tag{6}$$

$$p=\sum_i\alpha_i v'_i, \tag{7}$$

其中 u_i 为 v'_i 的隐藏层表示，u_a 为上下文嵌入向量，W_a 为注意力矩阵，b_a 为偏置项，α_i 为第 i 个节点的注意力权重系数.

2.5 分类器

得到程序嵌入向量 p 后，通过一层全连接网络构建分类器模型，使用 Softmax 作为激活函数：

$$y_i=\sigma(p_i)=\frac{\exp(p_i)}{\sum_{i=1}^{C}\exp(p_i)}, \tag{8}$$

其中 C 为类别数量，y_i 表示样本预测为第 i 类的概率.

检测模型利用随机梯度下降法进行训练，采用 L2 正则化的分类交叉熵作为损失度量.

3 实验分析

3.1 实验目的和数据来源

为测试 ASTCC 方法的可行性和有效性，本文进行了消融测试实验和对比分析实验.消融测试实验针对原始抽象语法树与压缩编码后的抽象语法树进行测试，以探究压缩编码对检测方法的贡献度；本文方法与 VulDeePecker[2] 和 TBCNN[10] 算法在相同数据集上进行对比实验，验证 ASTCC 的有效性.

实验数据采集来自 NVD[14] 和 SARD[15] 数据库，选用 C/C++程序中常见的 2 种漏洞类型：缓冲区错误（CWE-119）和资源管理错误（CWE-399）.NVD（national vulnerability database）是由美国国家标准与技术局创建和维护的通用漏洞数据库，包含漏洞详细信息，如漏洞编号、公布时间、漏洞来源、解决方案和补丁链接等.SARD（software assurance reference dataset）是一个软件保障参考数据集，专门供研究人员和软件开发人员进行工具测试，数据集中包括真实软件中的已知漏洞.根据 MITRE[16] 的研究，CWE-119 和 CWE-399 是最常见和最危险的软件漏洞，因此也被广泛用于测试源代码漏洞检测模型的性能.数据集概况如表 1 所示，按 8∶2划分训练集和测试集.

表 1 数据集概况

数据集	样本总数	正常样本	漏洞样本
CWE-119	39 753	29 313	10 440
CWE-399	21 885	14 600	7 285

3.2 实验环境

实验硬件环境为：Intel（R） Xeon（R） CPU E5-2680 v4 @ 2.40 GHz，64 GB RAM，NVIDIA GeForce GTX 1080 GPU.主要软件和开发工具包为：Python 3.7，TensorFlow v1.14.0，scikit-learn v0.22，PyCparser v2.18.

3.3 评价方法

CWE-119 和 CWE-399 漏洞检测任务属于二分类，实验采用机器学习领域常见指标评估检测模型的性能，包括误报率（*FPR*）、漏报率（*FNR*）、召回率（*Recall*）、准确率（*Accuracy*）及 F1 分数（f_1），计算方法如下：

$$FPR=\frac{FP}{FP+TN}, \tag{9}$$

$$FNR=\frac{FN}{TP+FN}, \tag{10}$$

$$Recall=\frac{TP}{TP+FN}, \tag{11}$$

$$Accuracy=\frac{TP+TN}{TP+FP+TN+FN}, \tag{12}$$

$$f_1=\frac{2\times Precision\times Recall}{Precision+Recall}, \tag{13}$$

其中：*FP* 指分类器错误地将正常样本判定为漏洞样本的数量；*FN* 指分类器错误地将漏洞样本判定为正常样本的数量；*TP* 指分类器正确地判定出漏洞样本的数量；*TN* 指分类器正确判定出正常样本的数量.

3.4 消融测试实验

3.4.1 实验过程和参数设置

基于抽象语法树压缩编码的漏洞检测方法消融测试实验主要过程如下：

1）使用 PyCparser 将数据集中所有源代码样本解析为 AST，作为基本数据格式；

2）采用深度优先的遍历顺序提取原始 AST 的节点 Token 序列；

3）基于训练集中的所有样本的 Token 序列构建语料库，训练 word2vec 模型并生成 Token 词嵌入；

4）针对原始 AST，使用词嵌入初始化节点向量，然后利用树卷积神经网络算法训练漏洞检测模型训练；

5）针对原始 AST，进行压缩编码，然后根据 2.3 节和 2.4 节所述方法构建语句嵌入网络和程序嵌入网络 2 层网络，训练检测模型，ASTCC 模型超参数如表 2 所示；

6）分别针对 2 种模型在测试集上进行漏洞检测实验.

表 2 中卷积核深度设置为 2，卷积层输出维度设置为 600.实验中发现当训练迭代次数超过 80 时，模型几乎达到了最佳的预测结果.因此将迭代次数统一设置为固定值 100，使得模型能够充分学习特征信息.当每批训练数据的样本数设置为默认值 256 时，可能会出现内存不足的情况，而当批大小设置过小时，模型可能会出现振荡问题，难以收敛.因此，实验中将批次大小设置为 128，其他参数使用默认值.

表 2 漏洞检测模型超参数

超参数	说明	值
word_dim	词嵌入维度	128
kernel_depth	卷积核深度	2
conv_dim	卷积层嵌入维度	600
iterations_number	迭代轮数	100
batch_size	批大小	128
drop_out	节点保留比例	0.5
learning_rate	学习率	0.001
L2	正则化选项	0.01

3.4.2 实验结果分析和主要结论

消融测试实验结果如表 3 所示，表中分别列出在缓冲区错误（CWE-119）和资源管理错误（CWE-399）数据集上的测试结果.

表 3 消融测试实验结果 %

数据集	方法	误报率	漏报率	查全率	准确率	F1
CWE-119	AST	0.97	27.85	72.15	90.58	82.80
	ASTCC	2.46	15.39	84.61	93.48	89.06
CWE-399	AST	6.06	15.01	84.99	91.13	85.73
	ASTCC	1.94	10.93	89.07	95.23	92.16

根据表 3 实验结果可以看出：

1）与原始 AST 相比，基于 ASTCC 的检测模型综合性能较优，在 CWE-119 和 CWE-399 数据集上检测准确率分别提升了 2.9%和 4.1%，F1 均提升了 6%以上，说明对抽象语法树的压缩编码能够提升模型检测性能.

2）ASTCC 方法在 CWE-399 数据集上的性能优于在 CWE-119 数据集，可能原因是 CWE-119 相比于 CWE-399 具有更复杂的漏洞模式，如涉及更多内存操作函数、从 source 到 sink 之间存在更复杂的数据流和控制流等.

3.5 对比分析实验

3.5.1 实验过程和参数设置

ASTCC 模型的超参数设置与消融测试实验所用相同，并在相同数据集上进行对比实验，对比方法的实现过程如下：

1）针对 TBCNN[10] 方法，在作者公开源码的基础上将其应用于源代码漏洞检测任务；

2）针对 VulDeePecker[2] 方法，根据原论文所述网络结构和参数进行复现.

3.5.2 实验结果分析和主要结论

对比分析实验结果如表 4 所示，表中分别列出不同方法在 CWE-119 和 CWE-399 数据集上的测试结果.

表 4 对比分析实验结果 %

数据集	方法	误报率	漏报率	查全率	准确率	F1
CWE-119	TBCNN	4.78	42.80	57.20	83.29	68.24
	VulDeePecker	3.73	41.78	58.22	85.71	71.87
	ASTCC	2.46	15.39	84.61	93.48	89.06
CWE-399	TBCNN	10.00	22.17	77.83	86.19	77.94
	VulDeePecker	8.69	20.29	79.71	87.66	80.25
	ASTCC	1.94	10.93	89.07	95.23	92.16

根据表 4 的对比分析实验结果可以看出：

1）ASTCC 在 CWE-119 和 CWE-399 数据集上的检测准确率分别达到 93.48%和 95.23%，且误报率和漏洞率均较低，ASTCC 综合表现最优.

2）与 TBCNN 相比，ASTCC 在 2 个数据集上的准确率分别提升了 10.19%和 9.04%，得益于 2 点改进工作：一是对 AST 的压缩编码减小语法树的复杂度，在不损失语法和结构特征的条件下缩小相关节点之间的距离，缓解了 TBCNN 中梯度消失的问题；二是通过构建语句嵌入和程序嵌入 2 层网络充分提取程序语法和结构特征，并利用注意力机制代替 TBCNN 中的池化层，以提取语法树中关键节点特征，增强模型漏洞表征能力.

3）与 VulDeePecker 相比，ASTCC 在 F1 指标上具有较大优势.原因在于 VulDeePecker 方法侧重于提取源代码的数据流特征和语义依赖信息，而忽略了语法结构特征.相比之下，ASTCC 针对程序语法和结构特征构建语句嵌入网络和程序嵌入网络，并通过注意力机制捕捉节点的语义依赖信息，增强模型对源代码漏洞的表征能力.

4 结 论

针对基于抽象语法树的漏洞检测方法难以充分挖掘大规模语法树中的漏洞特征而导致检测准确率低的问题，本文提出了一种基于抽象语法树压缩编码的源代码漏洞检测方法(ASTCC).首先，通过将原始抽象语法树以代码语句为单元进行分割、压缩、重建，在保留语法和结构信息的前提下较大程度地减小抽象语法树的深度并减少节点的数量，降低树的复杂度，达到对抽象语法树“压缩编码”的效果；然后，通过构建基于 RvNN 的语句嵌入网络以提取代码语句内的语法信息，再构建基于 TBCNN 程序嵌入网络以提取压缩编码树中结构信息，缓解 TBCNN 方法中由于语法树规模过大导致梯度消失的问题；最后，利用注意力机制捕捉压缩编码后的抽象语法树中节点关联信息，提升漏洞检测模型的性能.基于 NVD 和 SARD 公开数据集分别进行消融测试实验和对比分析实验.消融测试实验中，与原始抽象语法树相比，基于抽象语法树压缩编码的检测模型取得了更好的检测效果，F1 提升了 6%以上，说明对抽象语法树的压缩编码能够提升模型检测效果；在与 VulDeePecker 和 TBCNN 的对比分析实验中，ASTCC 方法在缓冲区错误(CWE-119)和资源管理错误(CWE-399) 2 类漏洞检测任务中均取得了较好效果.结果表明 ASTCC 能够有效降低抽象语法树的规模，增强模型对源代码漏洞的表征能力，提升源代码漏洞检测准确率.

参 考 文 献

[1] Lenarduzzi V, Taibi D, Tosi D, et al. Open source software evaluation, selection, and adoption: A systematic literature review [C] //Proc of the 46th Euromicro Conf on Software Engineering and Advanced Applications (SEAA). Piscataway, NJ: IEEE, 2020: 437-444

[2] Li Zhen, Zou Deqing, Xu Shouhuai, et al. VulDeePecker: A deep learning-based system for vulnerability detection [C] //Proc of the 2018 Network and Distributed System Security Symp. San Diego, CA: ISOC, 2018

[3] Dam H K, Tran T, Pham T T M, et al. Automatic feature learning for predicting vulnerable software components [J]. IEEE Trans on Software Engineering, 2021, 47(1): 67-85

[4] Duan Xu, Wu Jingzheng, Ji Shouling, et al. VulSniper: Focus your attention to shoot fine-grained vulnerabilities [C] //Proc of the 28th Int Joint Conf on Artificial Intelligence (IJCAI-19). Palo Alto, CA: AAAI Press, 2019: 4665-4671

[5] Li Jian, He Pinjia, Zhu Jieming, et al. Software defect prediction via convolutional neural network [C] /Proc of the 2017 IEEE Int Conf on Software Quality, Reliability and Security (QRS). Piscataway, NJ: IEEE, 2017: 318-328

[6] Liang Hongliang, Sun Lu, Wang Meilin, et al. Deep learning with customized abstract syntax tree for bug localization [J]. IEEE Access, 2019, 7: 116309-116320

[7] Lin Guanjun, Zhang Jun, Luo Wei, et al. Poster: Vulnerability discovery with function representation learning from unlabeled projects [C] //Proc of the 2017 ACM SIGSAC Conf on Computer and Communications. New York: ACM, 2017: 2539-2541

[8] Wang Song, Liu Taiyue, Tan Lin. Automatically learning semantic features for defect prediction [C] //Proc of the 38th IEEE Int Conf on Software Engineering. New York: ACM, 2016: 297-308

[9] **孙伟, 陈林. 基于抽象语法树的C#源代码SQL注入漏洞检测算法**[J]. **信息安全研究**, 2015, 1(2): 112-125

[10] Mou Lili, Li Ge, Zhang Lu, et al. Convolutional neural networks over tree structures for programming language processing [C] //Proc of the 30th AAAI Conf on Artificial Intelligence. Palo Alto, CA: AAAI Press, 2016: 1287-1293

[11] Zhang Jian, Wang Xu, Zhang Hongyu, et al. A novel neural source code representation based on abstract syntax tree [C] //Proc of the 41st Int Conf on Software Engineering (ICSE). Piscataway, NJ: IEEE, 2019: 783-794

[12] Bendersky E. Pycparser: Complete C99 parser in pure Python [EB/OL]. [2020-03-26]. https://github. com/eliben/pycparser

[13] Socher R, Pennington J, Huang E H, et al. Semi-supervised recursive autoencoders for predicting sentiment distributions [C] //Proc of the Conf on Empirical Methods in Natural Language Processing. Stroudsburg, PA: ACL, 2011: 151-161

[14] James O. NVD: National vulnerability database [EB/OL]. [2021-05-14]. https://nvd.nist.gov/

[15] James O. Software assurance reference dataset [EB/OL]. [2021-05-14]. https://samate.nist.gov/SRD/index.php

[16] Donald K. 2019 CWE top 25 most dangerous software errors [EB/OL]. [2021-01-21]. https://cwe. mitre. org/archive/2019_cwe_top25.html

陈传涛
硕士.主要研究方向为信息安全.
chencht163@163.com

潘丽敏
硕士,硕士生导师,高级实验师.主要研究方向为网络安全、文本安全、媒体安全、数据挖掘.
panlimin2016@gmail.com

龚　俊
博士研究生.主要研究方向为入侵检测、程序分析和信息安全.
3120170424@bit.edu.cn

马　勇
博士研究生.主要研究方向为自然语言语义理解、人工智能及操作系统内核技术.
znsoft@163.com

罗森林
博士,教授,博士生导师.主要研究方向为信息安全、数据挖掘、文本安全.
luosenlin2012@gmail.com