计算机科学
COMPUTER SCIENCE

基于增强AST的图神经网络函数级代码漏洞检测方法

顾守珂，陈文

引用本文

顾守珂，陈文. 基于增强AST的图神经网络函数级代码漏洞检测方法[J]. 计算机科学, 2023, 50(6): 283-290.

GU Shouke, CHEN Wen. Function Level Code Vulnerability Detection Method of Graph Neural Network Based on Extended AST [J]. Computer Science, 2023, 50(6): 283-290.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220600131

基于增强 AST 的图神经网络函数级代码漏洞检测方法

顾守珂　陈　文
四川大学网络空间安全学院　成都 610065
(gushouke@scu.edu.cn)

摘　要　软件漏洞逐年递增,安全问题愈发严重。在软件项目的交付阶段对原始代码进行漏洞检测可以有效避免后期运行时的安全漏洞,而代码漏洞检测依赖于有效的代码表征。传统的基于软件度量的表征方法与漏洞关联性较弱,难以对漏洞信息进行有效表征。近年来,机器学习为漏洞的智能化发现提供了新的思路,但该方法同样可能遗漏关键的代码特征信息。针对以上问题,文中在传统抽象语法树(AST)上增加控制依赖、数据依赖和语句序列边生成增强抽象语法树(EXAST)图结构,对原始代码进行表征以更好地处理代码结构化信息,并采用词向量嵌入算法(Word2Vec)将代码信息初始化为机器能够识别和学习的数值向量。同时,在传统的图神经网络(GNN)中引入门控循环单元(GRU),构建图识别模型,以缓解梯度消失并加强图结构中长期信息的传播,从而增强了代码执行的时序关系,提高了漏洞检测的准确度。最后在 SARD 公开数据集上对模型进行对比测试,实现了函数粒度的代码漏洞检测,相比传统的漏洞检测方法,准确率和 F1 分值分别最大提高了 32.54%和 44.99,实验结果证明了所提方法对代码漏洞检测的有效性。

关键词:漏洞挖掘;图神经网络;深度学习;抽象语法树;门控循环单元
中图法分类号　TP309

Function Level Code Vulnerability Detection Method of Graph Neural Network Based on Extended AST

GU Shouke and CHEN Wen
School of Cyber Science and Engineering,Sichuan University,Chengdu 610065,China

Abstract　With the increase of software vulnerabilities year by year,security problems are becoming more and more serious. Vulnerability detection of original code in the delivery stage of software project can effectively avoid security vulnerabilities in later run-time,and the discovery of code vulnerability depends on effective code characterization. The traditional characterization methods based on software metrics have weak correlation with vulnerabilities,so it is difficult to characterize vulnerability information efficiently. In recent years,machine learning has provided a new idea for intelligent discovery of vulnerabilities,but this method also has the problem of missing key information of code feature. To solve the above problems,control flow edge,data flow edge and next token edge are added to the traditional abstract syntax tree(AST) to generate an expanded abstract syntax tree (EXAST) graph structure,characterize the original code to better process the code structure information,and the word vector embedding model(word2vec) is used to initialize the code information into a numerical vector that the machine can recognize and learn. At the same time,the gate recurrent unit(GRU) is introduced into the traditional graph neural network(GNN) to build the model,which can alleviate the disappearance of the gradient,enhance the dissemination of long-term information in the graph structure to strengthen the timing relationship of code execution and improve the accuracy of vulnerability detection. Finally,the model is trained and tested on the SARD data sets to realize the function granularity code vulnerability detection,which can improve the accuracy of 32.54% and the F1 score of 44.99 compared with the traditional vulnerability detection method. Experimental results confirm the effectiveness of the method for code vulnerability detection.

Keywords　Vulnerability mining,Graph neural network,Deep learning,Abstract syntax tree,Gate recurrent unit

到稿日期:2022-06-13　返修日期:2023-03-17
基金项目:国家重点研发计划(020YFB1805405,2019QY0800);国家自然科学基金(U1736212,61872255,U19A2068);模式识别与智能信息处理四川省高校重点实验室(MSSB-2020-01)
This work was supported by the National Key Research and Development Program of China(020YFB1805405,2019QY0800),National Natural Science Foundation of China(U1736212,61872255,U19A2068) and Key Laboratory of Pattern Recognition and Intelligent Information Processing,Institutions of Higher Education of Sichuan Province(MSSB-2020-01).
通信作者:陈文(wenchen@scu.edu.cn)

1 引言

当前，软件漏洞已成为影响信息系统安全的重要因素。根据 NIST 报告[1]，2021 年 NVD(National Vulnerability Database)漏洞数量创下历史新高，单年收录了 20139 个安全漏洞，其中有 4065 个高危漏洞、12899 个中危漏洞和 3175 个低危漏洞，总体数量高于去年的 18351 个安全漏洞。漏洞数量呈逐年递增趋势，导致许多漏洞利用的攻击事件[2]，尤其带有漏洞安全隐患的代码，存在被攻击者利用的风险，对网络信息系统造成了极大的安全威胁。例如，最近披露的“log4j2”漏洞[3]，通过 IDAP 协议传入恶意内容，触发恶意代码的加载执行，已经造成广泛的安全影响。

常见的代码漏洞检测方法主要分为三大类：静态检测、动态检测和动静结合检测。早期静态检测有规则分析、代码相似性检测和符号执行等方法，常用的静态检测分析工具有 Flawfinder[4]，Klocwork[5]以及 Cobot[6]。然而静态检测方法误报率较高且过于依赖专家经验进行漏洞分析。动态检测可以对程序的动态行为进行监测，通过异常行为分析来检测潜在漏洞，典型的技术有动态污点分析和模糊测试等，常用的动态检测工具有 AFL[7]，Libfuzzer[8]等，可以达到较高的准确率，但是对目标程序的调试、分析和执行依赖于经验丰富的专业调试人员，存在检测速度慢、效率低和测试代码覆盖范围有限等问题。动静结合的检测将上述两种检测方法进行融合。然而，上述方法都依赖于一组已知的漏洞语法或行为模式，难以对新型漏洞进行检测。

近年来，人们开始探索新型智能化漏洞挖掘方法。研究表明[9]，机器学习和数据挖掘技术在检测常见的代码漏洞检测方面具有自学习、较少依赖专家经验等优势。现有基于机器学习的自动化漏洞检测方法主要基于源代码进行分析。首先采用代码度量等方法进行源码表征[10-11]，包括函数调用统计、软件复杂性、代码行数和嵌套程度等，将其作为识别潜在易受攻击代码片段的指标。但代码度量过程仅基于程序的整体属性，与代码漏洞的直接关联性较弱。文献[12-13]提出对源代码进行词法分析，生成序列化信息以进行代码表征，随后采用 SVM、神经网络等进行训练以产生漏洞检测模型。代码的词法分析蕴含了丰富的漏洞信息，有助于提升漏洞检测性能。

良好的代码表征可以减少源代码转换过程中造成的特征信息损失，这是代码漏洞检测的研究重点。文献[14]的研究表明，图结构适用于表示程序代码之间的复杂关联关系，能够有效地从源代码中获取有意义的信息，是一种有效的代码表征方式。抽象语法树(Abstract Syntax Tree，AST)是源代码语法结构的一种抽象化图形表示方法。AST 通过树状图表现编程语言的语法结构及语义关系，源代码中的每一种基本结构都可以由树上的节点表示。然而，基本的 AST 结构未能体现代码中大量的控制流、数据流信息，代码特征的表达能力受限。

本文提出了一种增强 AST 的图形化源码漏洞表征与检测方法，通过在图结构中增加控制边、数据依赖边和语义边并对节点进行整合得到 EXAST(Extended Abstract Syntax Tree)联合图结构，以减少源码特征信息损失，提高代码的表征能力。首先，使用源代码解析工具 Joern[15]提取出初始的抽象语法树结构，再通过增加控制流、数据流和语义信息最终完成增强抽象语法树(EXAST)的构建，随后采用 Word2Vec 算法中的连续词袋模型[16](Continuous Bag-of-Words，CBOW)对 EXAST 这一联合图结构进行向量化，最后将 Word2Vec 生成的 128 维向量输入引入门控循环单元(Gate Recurrent Unit，GRU)的图神经网络(Graph Neural Networks，GNN)中进行模型训练和检测。对比实验结果表明，该方法可以有效提高代码漏洞检测效果。本文的主要贡献如下：

(1)提出了一种基于增强抽象语法树(EXAST)的代码漏洞检测方法，用于对原始代码进行有效的表征。在 AST 结构的基础上，将不同级别的程序控制流、数据依赖流和语义流编码为异构边的 EXAST 联合图，即增强抽象语法树，节点间由 4 种不同类型的边进行连接。EXAST 联合图结构，减少了代码特征信息损失，有效提升了代码表征能力。

(2)将代码表征成图的结构与图神经网络相结合，采用了带 GRU 的图神经网络模型进行代码漏洞检测。在传统的 GNN 中引入 GRU 单元，用于增强 GNN 对输入序列处理的能力，通过增加门控装置来保存长期序列中的信息，增强网络长期记忆的能力，有效缓解反向传播时参数的梯度消失。

(3)在 SARD 公开数据集上进行对比实验，实验结果表明，与 5.3 节介绍的传统基线模型相比，本文模型的准确率平均分别高出 10.33% 和 15.69%，F1 得分平均分别高出 15.42% 和 19.27%，表明了代码漏洞检测的有效性。

本文第 2 节介绍了相关工作，包括常用的代码表征方法以及采用的机器学习、深度学习方法；第 3 节主要介绍了 EXAST 联合图结构的设计，并对表征后的图进行嵌入生成数值化向量，并将其作为后续神经网络的输入；第 4 节介绍了神经网络的构造；第 5 节在 SARD 数据集上进行实验，并与主流的代码漏洞检测方法进行对比，得出了实验结果；最后总结全文。

2 相关工作

传统基于代码审计的漏洞检测依赖于专家经验，人力成本较高。为了减少人力成本和人为主观因素对代码漏洞挖掘的影响，近年来，研究者开始使用机器学习和深度学习技术来对代码漏洞进行检测[17]，以实现自动化的漏洞挖掘。Younis 等[18]选取了 API 调用函数、程序源码行数和信息流等 8 个度量指标对原始代码进行表征，随后通过常见的机器学习算法如 SVM、朴素贝叶斯和随机森林等进行模型训练与预测。Yamaguchi 等[19]对源代码进行词法分析，将其解析为函数，提取出 API 符号，采用主成分分析法降维后计算函数相似性来辅助漏洞发现。上述方法关注代码本身的语法特征，而忽视代码上下文及结构信息。后续研究者受自然语言处理(Natural Language Processing，NLP)过程的启发，将自动化代码漏洞检测过程中的源代码视为一种特殊的文本[20-21]，在代码片断的上下文语境下结合语义和语法信息，对源代码进行分词并将其转化为语句序列(Token)，进一步使用词嵌入技术将 Token 转化为定长的数值向量后输入神经网络中进行训练，完成自动化的代码特征提取。如 Russell 等[12]提出

了一种基于 Token 表征的漏洞检测方法，采用自定义分析器对源码进行词法分析，将其转化为 Tokens 并映射到 k 维向量空间，将提取的特征输入随机森林完成代码漏洞的检测。为了进一步增强模型的检测能力，需要学习程序代码片断的语义上下文特征，通常采用递归神经网络（RNN），包括长短期记忆网络（LSTM）及其各种变体[20-23]进行代码特征提取。如 Li 等[20]提出的 VulDeePecker 检测模型，将源代码表征为 code gadget，编码后输入 BiLSTM 神经网络学习漏洞模式。

源码还可以表征成抽象语法树（AST）、数据流图（DFG）、控制流图（CFG）等多种结构，例如 Lin 等[24]提出了一种使用 AST 表示源代码的方法，解析每个函数的 AST，经过序列化、数值化后，选择双向长短时记忆神经网络（BiLSTM）模型进行特征提取，并将迁移学习应用于跨项目漏洞检测。Harer 等[25]自定义词法分析器，从代码构成的 CFG 图中提取出 116 维的特征向量，并构建了 3 个模型进行实验：第 1 个模型将初始向量送入极端随机森林进行训练；第 2 个模型以 TextCNN 卷积层特征作为极端随机森林的输入；第 3 个模型将 Word2Vec 输出的向量作为 TextCNN 的输入，训练并预测代码漏洞。代码的 CFG，DFG 图是由节点和边构成的抽象数据结构，与传统的多媒体图片、文本这类具有规则和顺序的结构化信息不同，抽象的代码图表现出了非欧几里德结构特征。针对此问题，Xu 等[26]提取了二进制程序控制流图，采用图神经网络计算图嵌入，通过函数相似性检测漏洞。Yu 等[27]使用 NLP 模型提取二进制代码并将其转化为 CFG 后的语义信息，通过信息传递网络（MPNN）后输出向量，同时在邻接矩阵上采用 CNN 模型提取 CFG 的节点顺序信息，向量拼接后通过 MLP 层进行分类预测。Duan 等[28]设计了特征编码器，将函数 CPG 转换为数字矩阵并作为注意力网络的输入，基于 Bi-GRU 神经网络建立分类模型，最终实现代码漏洞检测。

为了能够对代码中的丰富语法、语义、数据流和控制流等特征进行有效表征，本文采用增强代码图结构 EXAST 和图神经网络结合的方式进行代码漏洞检测。

3 代码表征

首先需要选择合适的代码划分粒度：如果针对整个代码文件进行表征和漏洞检测，则存在表征范围过于宽泛，由此产生的漏洞检测结果仅表明当前文件中是否存在漏洞，而有的文件可能包含数百个函数、数万行代码，要实现准确的漏洞定位较为困难。而细粒度地定位到具体代码片断的漏洞检测，则存在难以确定合适的代码切分片段和小段代码的表征较困难等问题。函数粒度的代码表征是对前两种方法的一种折中，具有易切分、识别效果好的特点，识别出漏洞后，后续人工处理工作量小，因此本文针对函数级代码漏洞进行特征提取、模型构建和检测。

当前基于机器学习的函数级代码漏洞检测方法多采用抽象语法树（AST）作为代码表征方式。AST 是一种用于描述程序代码语法结构的树形表示方式，但 AST 对程序的控制信息和数据依赖信息，以及代码中的语句执行顺序信息描述不足，导致代码表征过程中的部分特征信息损失[29]。而数据流图（DFG）、控制流图（CFG）等能够体现出程序的依赖信息。受此启发，我们通过在 AST 中增加控制依赖边、数据依赖边和代码顺序边并对节点进行整合得到 EXAST（Extended-AST）联合图结构，而采用图的形式对代码进行表征，具有更加丰富和完整的信息。

首先采用 Joern 代码解析工具以函数为单位自动化提取源代码的 AST 表示，再扫描并分析代码，增加数据依赖边、控制依赖边和语句序列边，生成联合图结构（EXAST）完成源代码的图编码。

由于神经网络无法直接处理代码图结构中的文本信息，本文基于数据集训练 Word2Vec 算法中的 CBOW 模型，针对每一个基本图结构进行初始化向量生成，将生成向量作为神经网络的输入完成分类模型的训练。我们采用基于 GRU 单元的 GNN 网络，旨在改善信息在图结构中的长期传播，缓解反向传播中的梯度消失问题。整个模型的架构如图 1 所示，其中自动化 AST 图的生成过程采用 NTLK 算法对源代码进行分词，构建字典（Build Dictionary）后训练出 CBOW 模型，之后将文本型向量 TextVector 提交给 word2vec 模型完成图嵌入，将 EXAST 映射成数值向量，并进一步输入 GRU-GNN 网络中对漏洞模式进行学习，以实现漏洞代码的自动化检测。

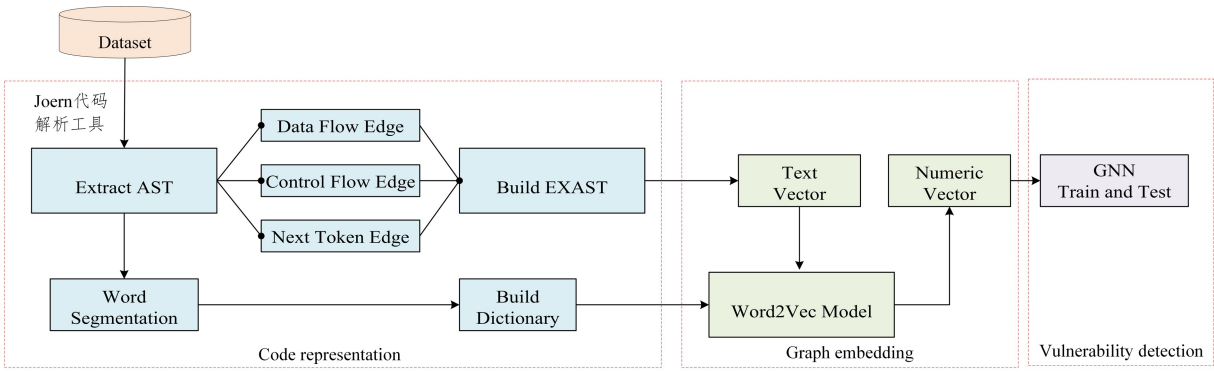

图 1 基于 EXAST 和图神经网络的漏洞检测方法

Fig. 1 Vulnerability detection based on EXAST and graph neural network

3.1 Extended-AST

AST 作为一种源代码树型语法结构，可以通过构造树型结构有效地反映代码之间的关系，保留了源代码信息，直观地表示了函数级控制流，实现了细粒度的程序表征，是代码解析

器用于理解程序的基本结构和检查语法错误的常用表示方法。本文使用 Joern 解析器构建代码的 AST。为了丰富程序的控制和运行信息，我们进一步使用了额外的控制流、数据流和顺序信息来扩展 AST，形成最终的代码表征方式，即 Extended-AST 联合图结构，其中控制依赖边可以体现源码的控制流程信息，数据依赖边可以跟踪操作数据流之间的流转依赖关系，语句序列边可以体现编程的逻辑顺序。最终将 EXAST 关系存储在单独的关系图 G 中。

有向图 G 由有序的二元组组成，$G=\langle V,E\rangle$，其中 V 是顶点的非空有限集合，E 表示有向边 $\langle v,w\rangle$ 的有序集，有 $V=\{v|v\in V\}$，$E=\{\langle v,w\rangle|v,w\in P(v,w)\}$，$P(v,w)$ 表示顶点 v 到顶点 w 有一条直接通路，其中 v 为始点，w 为终点。对于节点 $v\in V$，具有代码和类型两种属性，代码属性表示该节点的源代码，类型属性表示该节点所属类型，常见的类型属性有 DECL(声明)、METHOD(方法)、LOCAL(局部变量)等。

对于图 2 所示的 C 语言函数，生成的 EXAST 如图 3 所示，图 3 中的节点代表语句、代码块或值，边表示节点间的直接关系。接下来介绍边类型，以及如何综合各种子图，最后向量化联合图。

```
1 void test(){
2     int x = source();
3     if(x<20){
4         int y = 2*x;
5         sink(y);
6     }
7 }
```

图 2 一个 C 语言的例子

Fig. 2 An example of C function

(1)AST edge。从根节点开始，代码被分解为代码块、调用、语句、声明、表达式等，最后构造出树形结构。图中的方框代表 AST 节点，每个节点有 TYPE 和 CODE 两种属性，蓝色方框表示 AST 的根节点，黑色的箭头表示 AST 树中的父子关系。

(2)Data Flow edge 。添加表示程序中两个操作之间关系的数据依赖边，数据依赖边可以跟踪整个图中变量的使用情况，表达操作数据流之间的流转依赖关系。作为面向变量的数据流，涉及对变量的访问或修改，此时一个数据依赖边就可以表示某一相同变量的后续访问或修改。图 3 中的绿色实线箭头表示数据依赖关系，并在边缘上标注了所涉及的变量。

(3)Control Flow edge。添加控制依赖边可以表示一个操作将在另一个操作之后执行，描述了程序执行期间可能遍历的所有路径。图的边连接了在执行过程中具有执行流程顺序的基本块，表示控制传递关系，基本块可以通过边产生控制流操作，常见的控制流操作如 if，for 和 switch 等语句。控制依赖边在图 3 中用蓝色虚线显示，从 Entry 开始，到 Exit 结束。控制依赖边能够体现出代码整体的执行流程信息，使得模型可以学习到风险程序执行的逻辑特征，有利于准确的漏洞检测。

(4)Next Token edge 。传统的 AST 语法节点之间的关系描述中缺乏对源代码的自然执行顺序的表示。对源代码进行词法分析后得到的语句序列可以反映执行顺序，因此我们添加语句序列边(Next Token edge)将分词后的每个 Token 进行连接，使用红色虚线表示，以保留源代码序列所反映的编程逻辑，捕获语句的操作数或 API 使用顺序，以易于理解的方式保留语句序列之间的关系。

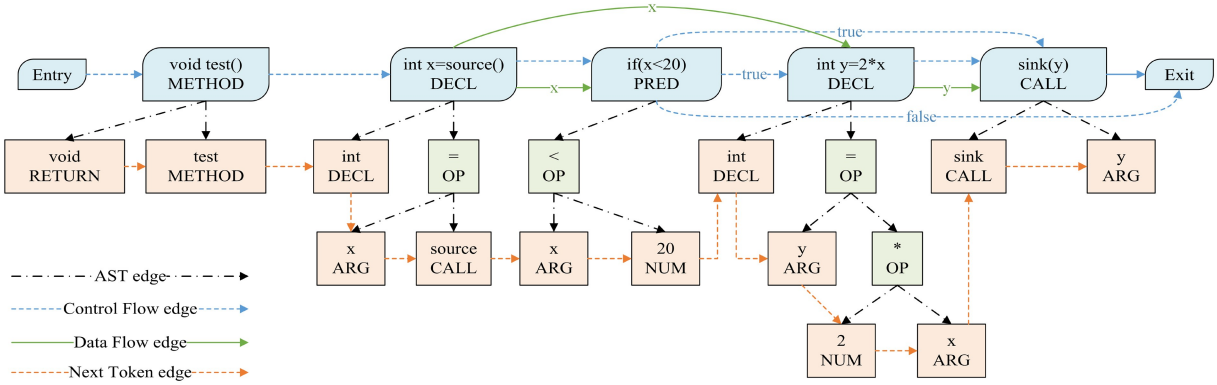

图 3 代码片段的图形表征(电子版为彩图)

Fig. 3 Graph representation of code slice

3.2 图嵌入

Extended-AST(EXAST)关系图是一个有向图 $G=\langle V,E\rangle$，V 表示 EXAST 中所有节点的集合，$V=\{v_1,v_2,\cdots,v_n\}$，其中 n 为节点数。$E=\{e_1,e_2,\cdots,e_m\}$ 表示 EXAST 中所有边的集合，其中 m 为有向边的数量。为了满足神经网络分类模型对输入数值化的要求，需要对 EXAST 中的节点和边进行嵌入。针对 AST edge，Data Flow edge，Control Flow edge，Next Token edge 这 4 种边，采用合并邻接矩阵表示来学习嵌入的边信息。除了“边”这种结构信息外，代码语义还需要通过单词嵌入来保存。节点中的源代码以字符串形式储存，需要对字符串进行分割，将变量、操作符、关键字分隔开，以保证转化后序列语义的正确性。但是源码中过多的字符串常量容易干扰代码表征中的有用特征信息提取，因此对于字符串常量，统一采用字符串“str”进行替换，减少无用信息的干扰。例如，对于代码块“printf(‘hello，world!’)”，进行字符串常量替换后为“printf(‘str’)”。将 EXAST 中的每一个基本块(即节点)作为一个输入序列，借鉴 NLP 的处理过程，将输入序列看作一段文本。通过 NLTK 分词工具将一个输入序列分词成若干“单词”。例如，对于代码块“int y=2 * x;”，通过 NTLK 分词后得到“int”“y”“2”“ * ”“x”“;”这 6 个“单词”。

由于文本型向量不能直接作为后续神经网络的输入，因此需要将 EXAST 图节点中的文本型向量转化为数字型

向量。本文构建一个映射关系（Word2Vec 模型）来将每一个文本型的“单词”链接到一个数值型的向量。通过单词嵌入可以将输入序列中的每个“单词”表示为一个固定维数的密集向量。传统的嵌入方法可以在代码上下文中将基本块序列信息中的“单词”转换为任意维度的向量，例如采用 one-hot 编码将上例中的“int”转化为六维向量[1,0,0,0,0,0]，将“*”转化为六维向量[0,0,0,1,0,0]。但这种方法会忽略“单词”之间可能存在的关系。为了更大化地利用“单词”之间的相互信息来进行特征的表征，我们应用 Word2vec 中的 CBOW 模型将“单词”嵌入成 128 维向量，语义相近的词在向量空间上的欧氏距离相近，这表明 CBOW 模型能较好地保留词之间的关系，进而减少代码表征中的信息损失和保留代码语义信息。所有源代码分词后构成代码语料字典，用于 Word2Vec 模型训练，得到映射关系。在代码语料库中每一个达到阈值的词都会与唯一的 128 维词向量构成一一映射关系。

为了让消息的聚合和传递能更好地体现出漏洞特征，我们将代码块中的代码编码后取平均，再与节点类型编码后的值求和，作为该节点最终的编码数值。最终完成图嵌入并传递到神经网络进行预测。对于节点 v，显然有 $v\in V$，每个节点都有类型和代码两种属性，v 节点类型属性为“DECL”，代码属性为“int x=source()”，其初始特征表示为 $\boldsymbol{x}_v=vec(DECL)+\sum_{i=1}^{k}vec(i)/k$，其中 $vec()$ 函数表示用预训练的 Word2Vec 模型完成数值化映射，k 表示代码分词后的“单词”数。

4 基于 GRU 的 GNN

基于聚合邻域信息的技术有图卷积网络[30]、图自编码器[31]及其变体等。本文使用邻域信息的传递与聚合来更新节点嵌入，且在邻域信息聚合中加入 GRU 单元[32]。对于图节点 v，嵌入层初始的 128 维嵌入向量 $\boldsymbol{x}_v$ 通过递归聚合和变换相邻节点的表示向量来计算隐层向量 $\boldsymbol{h}_v^t$，其中 t 为迭代轮数。节点通过将当前状态（即嵌入向量）作为信息与所有邻居节点交换信息。在每个节点上消息被聚合，然后用于更新下一个迭代隐含层向量。在指定迭代次数更新节点状态完成后，使用读出函数将节点状态聚合到单个嵌入向量。与标准的 GNN 不同，在关系图中传播和聚合信息后应用一个 GRU 单元格更新相同节点的状态，在传播过程中增加 GRU 门控单元增强了网络长期记忆能力，提高了图中信息长期传播的有效性，并有效缓解反向传播时参数梯度消失的问题[33]。整个网络的前向传递过程可以分为节点初始化、节点信息传递、节点信息更新和信息聚合 4 部分。对于节点初始化，在 3.2 节中已经描述了初始化节点 v 得到初始向量 $\boldsymbol{x}_v$。初始节点特征 $\boldsymbol{h}_{vj}^0$ 在初始向量 $\boldsymbol{x}_v$ 的基础上采用 padding 思想用 0 填充到输出维度的初始隐含层向量，即 $\boldsymbol{h}_{vj}^0=[\boldsymbol{x}_{vj},0]$。

$$\boldsymbol{m}_{v_i}^{t+1}=\sum_{v_j\in N(v_i)}\boldsymbol{W}_t\boldsymbol{h}_{v_j}^t+b \tag{1}$$

$$\boldsymbol{h}_{v_j}^{t+1}=GRU(\boldsymbol{h}_{v_j}^t,\boldsymbol{m}_{v_i}^{t+1}) \tag{2}$$

信息传递的过程如式(1)、式(2)所示，其中 t 表示时间（或迭代轮数）。在 t 时刻，每个节点的隐层向量通过传播函数来完成信息的传递过程。针对每一个节点 v_i 通过所有邻居节点隐层状态 $\boldsymbol{h}_{vj}^t$ 和相应边的状态信息 $\boldsymbol{W}_t$ 进行汇聚，边状态信息 $\boldsymbol{W}_t$ 由出边和入边的邻接矩阵拼接而成，同时加上偏置单元 b 能够对数据进行更好的拟合。节点信息更新函数式(2)调用门控循环单元（GRU），GRU 单元结构为更新门和重置门，通过更新门可以处理过往信息对当前时刻产生的影响，通过重置门可以将过往累积信息和当前输入信息进行综合，这种简单的结构可以提高训练的速度。通过这种门机制，图神经网络可以提取代码中的长期依赖关系，挖掘代码结构信息。式(2)接受两个输入分别为 $t+1$ 时刻生成的消息 $\boldsymbol{m}_{v_i}^{t+1}$ 和 t 时刻隐含层向量 $\boldsymbol{h}_{v_j}^t$，进行隐藏嵌入的更新。。

图神经网络主要针对节点层面或图层面进行预测任务，通过上述公式生成的节点特征可以作为预测层的输入，例如用于节点、边或图级。本文将源代码转化为 EXAST 图，需要针对图级别进行分类，以确定单个函数是否含有漏洞。针对图级预测需要聚合图的全局信息得到一个可以表示全图的向量，以便后续对该向量进行分类和回归操作。在图数据中引入池化操作进行信息聚合，将一个图表示为一个向量，图级表示向量的定义如式(3)所示：

$$\boldsymbol{h}_g=\tanh\Big(\sum_{v\in V}\sigma(i(\boldsymbol{h}_v^t,\boldsymbol{x}_v))\odot\tanh(j(\boldsymbol{h}_v^t,\boldsymbol{x}_v))\Big) \tag{3}$$

其中，$\sigma(i(\boldsymbol{h}_v^t,\boldsymbol{x}_v))$ 作为一种软注意机制，决定与当前图级任务相关的节点。i 和 j 是将 $\boldsymbol{h}_v^t$ 和 $\boldsymbol{x}_v$ 连接起来作为输入输出实值向量的神经网络。使用图分类的标准方法全图计算所有这些生成的节点嵌入，最后使用多层感知机（MLP）完成二分类任务。整个检测过程的实现过程如算法 1 所示。

算法 1 基于增强 AST 的图神经网络函数级代码漏洞检测系统

输入：以函数为单位切分源代码后的数据集 D

输出：Accuracy，Precision，Recall，F1

```
1. 初始化环境参数
2. AST,CFG,DFG←Joern with input from D
3. NODES←AST
4. for node_a,node_b in NODES
5.     edge←node_a,node_b
6. /* 提取控制依赖边、数据依赖边和语句序列边 */
7.     if edge in CFG. edge then
8.         edge←labeled as 'Control Flow edge'
9.     elif edge in DFG. edge then
10.        edge←labeled as 'Data Flow edge'
11.    elif node_a. nextToken==node_b. value then
12.        edge←labeled as 'Next Token edge'
13.    end if
14. end for
15. /* 生成 EXAST 联合图结构 G=〈NODES,EDGES〉后，进行图嵌入，图嵌入后的值以 json 格式保存 */
16. /* 节点嵌入 */
17. for node in NODES
18.    type_vector←word2vec[node. value. type]
19.    code_list←word_tokenize(node. value. code)
20.    code_vector←mean(word2vec[code_list])
21.    vector←type_vector+ code_vector
22. end for
23. /* 得到形如[源节点,边类型,目的节点]的结构表示连接关系 */
```

```
24. for edge in EDGES
25.    list.append(edge.get_source)
26.    list.append(edge.type)
27.    list.append(edge.get_destination)
28. end for
29. 图嵌入完成后以 json 文件保存,节点初始化后进入迭代
30. /* 训练阶段 */
31. for t=1,…n Step do
32.       GRU-GNN(G=(nodes,edges)) training with (1),(2)
33.    Res=sigmoid(MLP(h_g))
34.    基于交叉熵损失计算的梯度回传,更新
35. end for
36. /* 测试阶段 */
37. 输出评估指标
```

在检测系统实现算法中,第 1 和第 2 行使用代码解析工具 Joern 生成 AST;第 3－14 行在 AST 的基础上提取 3.1 节所述的控制依赖边、数据依赖边和语句序列边,生成 EXAST 图结构;第 17－22 行进行节点嵌入,将文本型向量通过 Word2Vec 模型转化为数字型向量,其中 Word2Vec 模型由源代码分词、构建字典后训练得到。节点有代码和类型两种属性,将类型属性映射后的数字向量与代码属性分词映射后求均值得到的数字向量之和作为该节点最终的数字向量;第 24－28 行得到 4 种边的连接关系;第 31－35 行基于交叉熵损失计算梯度回传,更新训练神经网络;第 37 行输出分类结果并计算评估指标。

5 实验结果与分析

5.1 数据集

本文数据集来源于软件保障参考数据集(SARD)[34],由美国国家标准技术研究所(NIST)维护。SARD 按照 CWE 编号收录了各种漏洞,CVE-ID 作为漏洞的唯一标识符,允许研究人员及安全从业人员等自行进行下载、实验、分析和评估,含常见漏洞种类如缓冲区错误(CWE-119,buffer error)、资源管理错误(CWE-399,resource management error)和跨界内存写(CWE-787,out-of-bounds Writes)等。本文实验的数据集漏洞的语言均为 C/C++,是在 SARD 上收集的两个特定类型漏洞——CWE-119 和 CWE-399。为了能让神经网络更好地学习到某种类型漏洞的特征,提升模型性能,针对不同类型的漏洞我们分别单独进行神经网络的训练,以挖掘不同类型的漏洞。本文中漏洞检测粒度单位达到函数级,因此从收集到的 C/C++源码中提取出正常函数和漏洞函数,正常函数打上标签"0",作为负类样本,漏洞函数打上标签"1",作为正类样本,最终构建的缓冲区错误和资源管理错误数据集的数据量如表 1 所列。

表 1 数据集信息

Table1 Data set information

type	number
CWE-119	17844
CWE-399	12382

我们使用这两个开源 C/C++代码数据集来进行评估,并将数据集按照 6:2:2 的比例划分成训练集、验证集和测试集。训练集用于进行监督学习,并训练模型来预测函数是否含有漏洞。验证集用于对模型的超参数进行调整和优化,测试集仅仅用于性能评估。

5.2 评估指标

本文评估神经网络模型的评估指标依赖于混淆矩阵,混淆矩阵的定义如表 2 所列。

表 2 混淆矩阵

Table 2 Confusion matrix

Confusion Matrix		Real samples	
		positive	negative
Forecasting samples	positive	TP	FP
	negative	FN	TN

表 2 中,TP(True Positive)表示真实类,即样本标签与模型预测结果均为正类;FP(False Positive)表示假正类,即样本标签为负但是模型预测结果为正;FN(False Negative)表示假负类,即样本标签为正但是模型预测结果为负;TN(True Negative)表示真负类,即样本标签与模型预测结果均为负类。可以通过混淆矩阵计算出模型的评价指标,对模型进行测试评估。

$$Accuracy=(TP+TN)/(TP+TN+FP+FN) \tag{4}$$

Accuracy 指标表示精确率,模型预测正确的样本与所有样本的比值,数值与模型性能成正比。

$$Precision=TP\ /\ (TP+\ FP) \tag{5}$$

Precision 指标表示正确率,模型预测为正类的样本中实际标签为正类的占比,数值与模型性能成正比。

$$Recall=TP\ /\ (TP+FN) \tag{6}$$

Recall 指标表示召回率,正类的样本中模型正确预测的比值,数值与模型性能成正比。

$$F1\_Score=2*Precision*Recall/(Precision+Recall) \tag{7}$$

F1 分数综合考虑了正确率和召回率,通过多种指标融合能够更好地反映出模型的性能。

5.3 基线模型

对于一个传统基线模型,本文选择了支持对 C/C++源代码进行静态扫描的工具 Flawfinder,该工具可以不用编码而直接在代码上进行扫描检测,简单快速且不需要编译,代表着早期检测方式。后续随着人工智能产业的兴起,大量基于机器学习的漏洞挖掘研究应运而生。代码表征可以将源代码转换为能够表现漏洞特性的向量形式,好的表征形式可以最大程度地从原始数据中提取特征。常用的代码表征形式有代码度量、Token 序列和图,后续实验选取不同的表征方式以验证本文中图表征对代码漏洞发现的有效性。我们在常用的机器学习模型中选择随机森林分类器作为第二个对比模型,使用 23 个常用软件度量(如代码行、嵌套程度、调用函数等)作为特征来训练一个随机森林分类器。该方法是从单个函数的源代码中导出特征向量,输入模型后给出该函数是否含有漏洞的结果判断,代表着以代码度量作为表征方式的检测方法。源代码到特征向量有多种编码方式,如 one-hot 编码或 Word2Vec 等。对源代码进行预处理后,通过构建"单词"字典来训练一个 Word2Vec 模型。将源代码序列化后视为自然

语言，经过训练好的 Word2Vec 模型输出后会得到 128 维的特征向量序列，这些向量可以输入深度学习模型进行训练。由于函数代码发生漏洞处可能会受到程序中早期语句的影响，也可能会受到后面语句的影响，因此使用双向长短期记忆网络(BiLSTM)对特征向量进行学习，以此作为第三个对比模型，代表着基于 Token 表征的漏洞检测方法。第四个对比实验代码表征方式为抽象语法树(AST)，通过提取函数 AST 结构，利用 Word2Vec 模型完成图嵌入，采用与本文方法相同的图神经网络进行训练，目的是验证本文提出的 EXAST 联合图结构对代码的表征能力。

5.4 实验结果分析

本文神经网络的实现使用了 PyTorch(版本 1.10.1)框架，Word2vec 嵌入是由 gensim 包(版本 4.0.1)提供，采取维度 128 维。该计算机系统是一台运行 Ubuntu 18.04 操作系统的服务器，CPU 相关信息为 Intel(R) Xeon(R) Silver 4210R CPU @ 2.40 GHz，64GBRAM，GPU 型号为 NVIDIA GeForce RTX 3090，24GB 显存，模型使用 GPU 进行训练加速，CUDA 版本号为 11.3。

本文方法中神经网络的设置为：学习率(Learning Rate)设置为 0.0001，嵌入层通过 Word2Vec 模型后输出 128 维，对于门控循环层，我们将其隐层维度设置为 200，batch size 设置为 128，采用自适应矩估计(Adam)优化算法加快模型的收敛。

对于 5.1 节的实验数据进行分析，我们对比了本文模型和 5.3 节中描述的其他 4 种不同的模型(即 Flawfinder，Random Forest，BiLSTM 和 AST+GRU-GNN)，依照评估指标进行了对比实验。测试集的实验结果如表 3 所列。

表 3 5 种模型在不同数据集上的结果

Table 3 Results of five models on different datasets

Dataset	Model	*Accuracy*	*Precision*	*Recall*	*F1*
CWE-119	Flawfinder	60.29	52.63	35.71	42.55
	RandomForest	72.06	66.67	64.29	65.45
	BiLSTM	77.01	80.43	77.08	78.72
	AST+GRU-GNN	83.35	97.36	67.13	79.47
	Our model	**83.51**	87.61	**77.01**	**81.97**
CWE-399	Flawfinder	54.41	43.48	35.71	39.22
	RandomForest	67.65	65.38	56.67	60.71
	BiLSTM	77.63	76.92	78.95	77.92
	AST+GRU-GNN	85.34	95.26	71.81	81.89
	Our model	**86.95**	**95.38**	**75.39**	**84.21**

Flawfinder 直接在源代码上进行词法分析，试图提取漏洞模式，但依赖于专家经验进行定义，并未考虑到代码的上下文语义信息，难以有效地提取出漏洞特征，导致这种基于规则的自动化漏洞挖掘工具难以达到理想的效果，有较大的提升空间。我们按照 5.1 节的划分方法对 CWE-119 和 CWE-399 数据集进行划分，将分割的训练集和验证集合并形成新的训练集，即在总数据集上按照 8:2 的比例进行划分。使用 23 个常用的软件度量作为特征来进行随机森林分类器的训练，随后在同一测试集上进行验证。BiLSTM 将源代码视为自然语言，与本文方法同样使用 Word2Vec 映射的 128 维向量作为初始嵌入，双向结构的网络层数为 3 层。从实验结果可以看到，采用深度学习自动进行特征提取分类的性能显著优于基于规则的静态扫描工具和依赖专家经验进行特征选择的随机森林。通过规则匹配和手工设计特征过于依赖先验知识，只能采用少量参数设计特征，难以利用大数据的优势，展现出来的性能不佳，而将深度学习与漏洞挖掘领域相结合可以有效地利用大数据自动学习特征，从训练数据中迅速学习到有效的特征表示。因此结合深度学习的本文方法相比前两种模型在性能上有较大提升，准确率增益平均达到了 21.63%。与同为深度学习的 BiLSTM 模型相比，本文方法构造了 EXAST 联合图结构作为非结构化特征，考虑到了程序的控制信息、数据依赖信息以及代码中的语句执行顺序信息而，对原始代码进行了有效的表征，相比 BiLSTM 模型采用的令牌序列表征方式信息损失更低，同时采用引入 GRU 的图神经网络处理非结构化特征并增强网络中的长期记忆能力，在 F1 分数这项指标上达到最优。与 BiLSTM 模型相比，在 CWE-119 上 F1 分数提高了 3.25，在 CWE-399 上 F1 分数提高了 6.29，虽然 BiLSTM 可以从上下文关系以及时序关系中提取出语义特征，但其忽视了代码块之间的相互依赖关系，在图中编码可以提取到更多隐含信息，极大保留了特征信息，性能更优。当采用相同的网络结构(GRU-GNN)时，只用 AST 进行函数代码表征，会因忽视关键的程序控制信息、数据依赖信息和代码中的语句执行顺序信息导致表征过程中出现信息损失，导致多项评估指标低于 EXAST。实验证明了本文提出的 EXAST 联合图结构表征方式能有效提升代码表征能力，在多个数据集上 F1 指标均取得最优。

结束语 本文针对软件漏洞检测这一热点问题，从源代码的角度进行分析，提出了一种基于门控循环图神经网络的漏洞检测方法，在 AST 的基础上，将源码转换为增强图结构 EXAST，利用这种结构有效地对代码进行表征，通过词向量嵌入初始化节点，通过引入 GRU 单元的 GNN 网络进行消息的传递和聚合，最后通过 MLP 层完成二分类任务，识别粒度达到了函数级。在开源数据集 CWE-119 和 CWE-399 上进行对比实验，尤其是针对传统的机器学习算法和静态扫描工具，本文方法的效果提升明显，且减弱了对专家经验的依赖和人为主观特征选取工作。与 BiLSTM 模型和 AST+GRU-GNN 的对比实验结果表明，本文方法更能够被有效应用于自动化漏洞挖掘。在未来的研究中，将进一步研究信息损失更少的代码表征方式，以及优化神经网络结构使模型具有更好的适应性。

参 考 文 献

[1] NIST. CVSS Severity Distribution Over Time [EB/OL]. [2021-12-10]. https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cvss-severity-distribution-over-time.

[2] PEISERT S, SCHNEIER B, OKHRAVI H, et al. Perspectives on the solarwinds incident[J]. IEEE Security & Privacy, 2021, 19(2): 7-13.

[3] CVE[EB/OL]. https://www.cve.org/CVERecord?id=CVE-2021-44228.

[4] Dwheeler. Flawfinder software official website [EB/OL]. https://dwheeler.com/flawfinder/.

[5] KlockWork: Best Static Code Analyzer for Developer Productivity[EB/OL]. https://www.perforce.com/products/klocwork.

[6] GAO Q, ZHANG S, CHEN X, et al. CoBOT: Static C/C++ bug detection in the presence of incomplete code[C]// IEEE/ACM 26th International Conference on Program Comprehension. 2018.
[7] AFL[OL]. https://lcamtuf.coredump.cx/afl.
[8] LibFuzzer[OL]. https://llvm.org/docs/LibFuzzer.html.
[9] LIN G, ZHANG J, LUO W, et al. Software Vulnerability Discovery via Learning Multi-Domain Knowledge Bases[J]. IEEE Transactions on Dependable and Secure Computing, 2021, 18(5): 2469-2485.
[10] PERL H, DECHAND S, SMITH M, et al. VCCFinder: Finding potential vulnerabilities in open-source projects to assist code audits[C]// Proceedings of the 22nd ACM SIGSAC Conference on Computer & Communications Security. 2015: 426-437.
[11] SHIN Y, MENEELY A, WILLIAMS L, et al. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities[J]. IEEE Transactions on Software Engineering, 2011, 37(6): 772-787.
[12] RUSSELL R, KIM L, HAMILTON L, et al. Automated Vulnerability Detection in Source Code Using Deep Representation Learning[C]// 2018 17th IEEE international conference on machine learning and applications. 2018. 757 –762.
[13] SHEN Y, MARICONTI E, VERVIER P A, et al. Tiresias: Predicting security events through deep learning[C]// Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018: 592-605.
[14] XU X, LIU C, FENG Q, et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]// Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 363-376.
[15] Joern[OL]. https://joern.readthedocs.io/en/latest.
[16] MIKOLOV T, CHEN K, CORRADO G, et al. Efficient estimation of word representations in vector space[J]. arXiv: 1301.3781, 2013.
[17] GRIECO G, GRINBLAT G L, UZAL L, et al. Toward large-scale vulnerability discovery using machine learning[C]// Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy. 2016: 85-96.
[18] YOUNIS A, MALAIYA Y, ANDERSON C, et al. To fear or not to fear that is the question: Code characteristics of a vulnerable function with an existing exploit[C]// the Sixth ACM Conference on Data & Applications Security & Privacy. 2016: 97-104.
[19] YAMAGUCHI F, RIECK K. Vulnerability extrapolation: Assisted discovery of vulnerabilities using machine learning[C]// 5th USENIX Workshop on Offensive Technologies(WOOT 11). 2011.
[20] LI Z, ZOU D, XU S, et al. Vuldeepecker: A deep learning-based system for vulnerability detection[J]. arXiv: 1801.01681, 2018.
[21] ZOU D, WANG S, XU S, et al. μvuldeepecker: A deep learning-based system for multiclass vulnerability detection[J]. IEEE Transactions on Dependable and Secure Computing, 2019, 18(5): 2224-2236.
[22] LIN G, WEN S, HAN Q L, et al. Software vulnerability detection using deep neural networks: a survey[J]. Proceedings of the IEEE, 2020, 108(10): 1825-1848.
[23] LI Z, ZOU D, XU S, et al. Sysevr: A framework for using deep learning to detect software vulnerabilities[J]. IEEE Transactions on Dependable and Secure Computing, 2022, 19(4): 2244-2258.
[24] LIN G, ZHANG J, LUO W, et al. POSTER: Vulnerability discovery with function representation learning from unlabeled projects[C]// Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 2539-2541.
[25] HARER J A, KIM L Y, RUSSELL R L, et al. Automated software vulnerability detection with machine learning[J]. arXiv: 1803.04497, 2018.
[26] XU X, LIU C, FENG Q, et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]// Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 363-376.
[27] YU Z, CAO R, TANG Q, et al. Order matters: semantic-aware neural networks for binary code similarity detection[C]// Proceedings of the AAAI Conference on Artificial Intelligence. 2020, 34(1): 1145-1152.
[28] DUAN X, WU J, JI S, et al. Vulsniper: Focus your attention to shoot fine-grained vulnerabilities[C]// International Joint Conference on Artificial Intelligence. 2019: 4665-4671.
[29] YAMAGUCHI F, GOLDE N, ARP D, et al. Modeling and discovering vulnerabilities with code property graphs[C]// 2014 IEEE Symposium on Security and Privacy. IEEE, 2014: 590-604.
[30] KIPF T N, WELLING M. Semi-supervised classification with graph convolutional networks[J]. arXiv: 1609.02907, 2016.
[31] KINGMA D P, WELLING M. Auto-encoding variational bayes [J]. arXiv: 1312.6114, 2013.
[32] LI Y, TARLOW D, BROCKSCHMIDT M, et al. Gated graph sequence neural networks[J]. arXiv: 1511.05493, 2015.
[33] CHUNG J, GULCEHRE C, CHO K, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling[J]. arXiv: 1412.3555, 2014.
[34] NIST software assurance reference dataset project[EB/OL]. https://www.nist.gov/itl/ssd/software-quality-group/software-assurance-reference-dataset-sard-manual.

GU Shouke, born in 1998, postgraduate. His main research interests include graph neural network, cyber security and vulnerability mining

CHEN Wen, born in 1983, Ph.D, associate professor, master supervisor, is a member of China Computer Federation. His main research interests include network security, information hiding and data mining.

(责任编辑:喻藜)