

基于图神经网络的代码漏洞检测方法

陈皓, 易平

(上海交通大学网络空间安全学院, 上海 200240)

摘要: 使用神经网络进行漏洞检测的方案大多基于传统自然语言处理的思路, 将源代码当作序列样本处理, 忽视了代码中所具有的结构性特征, 从而遗漏了可能存在的漏洞。提出了一种基于图神经网络的代码漏洞检测方法, 通过中间语言的控制流图特征, 实现了函数级别的智能化代码漏洞检测。首先, 将源代码编译为中间表示, 进而提取其包含结构信息的控制流图, 同时使用词向量嵌入算法初始化基本块向量提取代码语义信息; 然后, 完成拼接生成图结构样本数据, 使用多层图神经网络对图结构数据特征进行模型训练和测试。采用开源漏洞样本数据集生成测试数据对所提方法进行了评估, 结果显示该方法有效提高了漏洞检测能力。

关键词: 漏洞检测; 图神经网络; 控制流图; 中间表示

中图分类号: TP309

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2021039

Code vulnerability detection method based on graph neural network

CHEN Hao, YI Ping

School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

Abstract: The schemes of using neural networks for vulnerability detection are mostly based on traditional natural language processing ideas, processing the code as array samples and ignoring the structural features in the code, which may omit possible vulnerabilities. A code vulnerability detection method based on graph neural network was proposed, which realized function-level code vulnerability detection through the control flow graph feature of the intermediate language. Firstly, the source code was compiled into an intermediate representation, and then the control flow graph containing structural information was extracted. At the same time, the word vector embedding algorithm was used to initialize the vector of basic block to extract the code semantic information. Then both of above were spliced to generate the graph structure sample data. The multilayer graph neural network model was trained and tested on graph structure data features. The open source vulnerability sample data set was used to generate test data to evaluate the method proposed. The results show that the method effectively improves the vulnerability detection ability.

Keywords: vulnerability detection, graph neural network, control flow graph, intermediate representation

收稿日期: 2020-10-19; 修回日期: 2020-12-15

通信作者: 易平, yiping@sjtu.edu.cn

基金项目: 国家重点研发计划 (2019YFB1405000, 2017YFB0802900)

Foundation Item: The National Key R&D Program of China (2019YFB1405000, 2017YFB0802900)

论文引用格式: 陈皓, 易平. 基于图神经网络的代码漏洞检测方法[J]. 网络与信息安全学报, 2021, 7(3): 37-45.

CHEN H, YI P. Code vulnerability detection method based on graph neural network[J]. Chinese Journal of Network and Information Security, 2021, 7(3): 37-45.

1 引言

随着各种智能设备的持续突破和互联网的持续发展与广泛应用,网络空间促进了经济社会的繁荣进步,也正在全面改变人们的生产生活方式,深刻影响着人类社会历史的发展进程,但同时带来了新的安全风险和挑战。2016年,国家互联网信息办公室发布的《国家网络空间安全战略》^[1]指出,网络空间安全事关人类共同利益,事关世界和平与发展,事关各国国家安全,要夯实网络安全基础。互联网行业竞争激烈,软件开发周期越来越短,安全问题日益突出。随着软件数量的大规模增长以及复杂性的增强,传统方案面临着严重的技术瓶颈,这给软件安全漏洞的研究带来严峻的挑战,已经无法满足网络空间安全的防护需要。

面对挑战,基于机器学习和人工智能的脆弱性分析方法被应用于脆弱性分析领域,开始成为研究的热点方向。近年来,深度学习在处理分类问题中显示出了巨大的优势,已经在图像识别、自然语言处理等领域取得了突破性进展。与计算机视觉和自然语言处理等深度学习领域相比,源代码漏洞检测首先要解决的关键问题是样本的预处理与合适的深度神经网络结构设计。与图片样本和文本样本不同,源代码由于具有丰富的代码结构信息和语义信息,可以看作一种非欧几里得数据。现有的研究方案大多把源代码转化为文本序列,使用自然语言处理中常用的循环神经网络进行自动化特征提取,但这忽略了代码中由于跳转、循环、判断形成的控制流特征,往往会有较大的信息损失。如何提取源代码样本的特征信息,把源代码样本转换成适合后续神经网络训练的数据形式并最大化保留样本信息是关键问题之一。

本文通过代码中间语言的控制流图提取代码样本的结构特征,使用控制流图基本块中的代码序列提取代码样本的语义特征,将代码样本转化为图结构数据,以解决序列样本丢失代码结构特征信息的问题。另一个关键问题是针对具有丰富语法特征和语义特征的源代码样本,选用哪种合适的深度神经网络模型。图神经网络由于其设计原理更适合处理图结构数据,在图级别的分类任

务上也取得了非常好的效果^[2]。使用基于中间语言的控制流图信息对源代码进行漏洞检测的过程可以抽象为在图结构数据样本上进行图级别的分类任务,因此,图神经网络相较于其他模型更有优势。

本文的主要贡献如下。

1) 设计了一种基于中间语言控制流图的代码特征提取方案。通过提取源代码的控制流图结构信息,同时使用基本块的中间语言提取代码语义信息,嵌入图节点中,将源代码转化为图结构数据样本。

2) 使用图神经网络模型处理代码图结构样本。基于图结构的数据样本,设计了一种基于图卷积的深度神经网络模型,用来对样本数据进行特征提取和样本分类。

3) 基于设计方案设计了实验方案并对结果进行了分析。实验结果显示,本文设计的方案在基准数据集上取得了比传统静态扫描更高的准确率。

2 相关工作

近年来,深度学习与软件漏洞检测的交叉研究逐渐成为软件安全和人工智能领域共同的研究热点。由于传统软件漏洞检测有了较长时间的研究,各种检测方法较为成熟,其中的优劣势也较为明显。人工智能的加入大多着眼于与现有的软件检测方案结合,以解决现有检测方案的问题。传统的源代码静态检测技术主要分为基于逻辑推理的漏洞检测、基于中间表示的漏洞检测和基于相似性的漏洞检测。基于逻辑推理的漏洞检测将源代码进行形式化描述,然后利用数学推理证明等方法验证形式化描述的性质,从而推断是否含有某类漏洞。基于中间表示的漏洞检测则将源代码转换为易于分析检测的中间表示,然后检查其对应的漏洞规则,具体的检测方法有数据流分析、控制流分析、污点分析、静态符号执行等。基于相似性的漏洞检测主要检测由于代码复用等产生的漏洞。2010年,Pham等^[3]提出并开发了SecureSync工具,其将源代码转化为扩展的抽象语法树的中间表示,用于检测代码相似性进而检测漏洞。同样进行相似性检测,也可以选取不同的中间表示,基于图的中间表示综合考虑程序的

语法和语义特征,有很好的检测能力。2014年,Yamaguchi等^[4]提出了一种将经典程序分析的概念(抽象语法树、控制流图和程序依赖图)合并成为代码属性图的图结构,进而通过对常见漏洞的建模识别源代码漏洞。

随着深度学习、人工神经网络等算法的不断发展进步,各种新型算法逐步与传统检测算法先后结合,涌现出了一批具有交叉方案的研究成果。Lin等^[5]提出了针对函数级别的跨项目场景下的漏洞检测,使用抽象语法树AST表示函数,通过双向LSTM神经网络进行特征提取和学习。Russell等^[6]建立了包含C/C++开源代码的用于深度学习训练的函数级别漏洞数据集,并基于深度表示学习开发了一个大规模漏洞检测系统。在二进制代码检测领域,Xu等^[7]提出了一种基于神经网络的跨平台的二进制代码相似性检测模型,通过计算使用神经网络嵌入得到的函数向量距离来进行相似性检测。李珍等^[8]提出了基于图神经网络的图嵌入算法,并在此基础上提出一种图匹配网络模型来进行代码控制流图的功能相似性检测。Yu等^[9]在文献[7]的基础上提出了一种结合图神经网络和NLP算法提取二进制程序控制流图进行相似性检测的方法,一方面采用MPNN(message passing neural network)提取控制流图语法结构特征,另一方面采用BERT预训练框架提取代码语义特征。Duan等^[10]则提出了一种无监督的代码表示学习技术解决二进制相似性分析的问题,也是处理代码语义信息和控制流信息生成基本块嵌入。李珍等^[11]研究了近年来使用深度学习方案进行漏洞检测的方案,并以基于源代码相似性的漏洞检测系统和面向源代码的软件漏洞智能检测系统两个具体方案为例详细介绍了基于深度学习的漏洞检测过程,指出了目前存在的样本标注、漏洞模式和模型选择以及可解释性方面存在的问题。

图神经网络(GNN)是一类处理图数据的神经

网络的统称。图是一种数据结构,它对一组对象(节点)及其关系(边)进行建模。近年来,由于图结构的强大表现力,用机器学习方法和深度学习方法分析图的研究越来越受到重视。图神经网络是一类基于深度学习的处理图域信息的方法。由于其较好的性能和可解释性,GNN最近已成为一种广泛应用的图分析方法。图神经网络的概念最早由Gori等^[12]提出,并由Scarselli等^[13]进一步阐明。这些早期的研究以迭代的方式通过循环神经架构传播邻近信息来学习目标节点的表示,直到达到稳定的固定点。Gilmer等^[14]提出了一种MPNN架构用于化学领域分子特征的预测,创造性地将图信息传递的过程分为传播、更新、读出3个阶段,Yu等^[9]的研究基于MPNN在代码领域取得了比较好的效果。

3 方案设计

本文的目标是建立一种基于图神经网络的源代码漏洞检测方案。方案的整体设计包括代码预处理与中间语言编译、控制流图提取、代码语义信息提取、图神经网络模型设计等。方案的整体设计如图1所示。

3.1 代码预处理

在传统的源代码漏洞扫描方案中,由于不同类型编程语言在语法上存在较大差异,往往会针对不同类型语言开发不同的规则或者工具。本文采用LLVM Intermediate Representation(IR)的中间语言方案将源代码预编译为中间表示,使检测方法能够兼容不同类型的编程语言。

LLVM编译架构如图2所示。LLVM编译架构遵循前后端分离的设计理念,编译器前端可以把不同类型的高级语言编译成为LLVM IR,经过优化之后再由不同系统架构的编译器后端将LLVM IR编译成不同架构的二进制可执行文件。

LLVM IR有两种表示形式:一种是可读的汇编语言形式,以.ll文件存储;另一种是序列化之后的bitcode形式,以.bc文件存储。

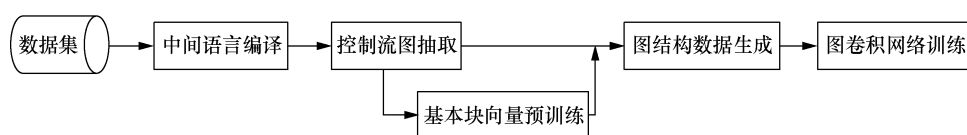


图1 方案的整体设计

Figure 1 Overall design drawing

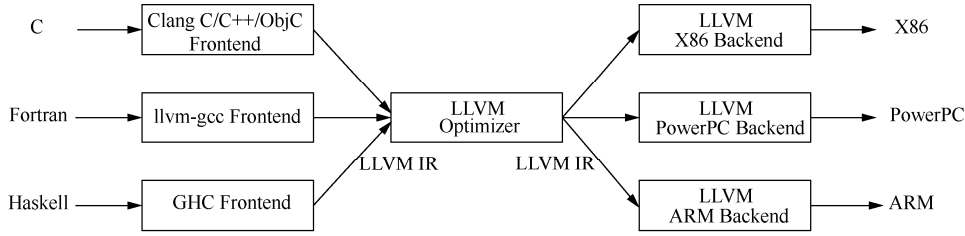


图 2 LLVM 编译架构
Figure 2 LLVM compilation architecture

本文通过 Clang 编译器将数据集代码样本编译成为可读形式的中间语言代码样本，作为下一步特征提取的基础。图 3 所示为数据集中的一个样本示例，可以看到在编译到 LLVM IR 之后，源代码中的变量名标识符（如 `data`、`result`）均被替换为 `%{number}` 形式的变量名，同时编译后的 LLVM IR 代码具有静态单赋值形式（SSA, static single assignment form），即每一个变量仅能被赋值一次。另外，编译到 IR 之后需要对代码文本进行预处理，主要包括函数名替换、特殊符号替换等。相较于使用源代码，一方面，中间语言代码的 SSA 形式和变量标识符替换有利于后续代码语义特征提取和预训练；另一方面，使用中间语言可以屏蔽不同类型高级语言的语法差异，能够兼容不同类型的高级语言，起到扩展训练样本的作用。

```
void CWE190_Integer_Overflow__char_fscanf_multiply_01_bad()
{
    char data;
    data = ' ';
    /* POTENTIAL FLAW: Use a value input from the console */
    fscanf(stdin, "%c", &data);
    if(data > 0) /* ensure we won't have an underflow */
    {
        /* POTENTIAL FLAW: if (data*2) > CHAR_MAX, this will ov
erflow */
        char result = data * 2;
        printHexCharLine(result);
    }
}

; Function Attrs: noinline nounwind optnone uwtable
define void @CWE190_Integer_Overflow__char_fscanf_multiply_01_
bad() #0 {
    %1 = alloca i8, align 1
    %2 = alloca i8, align 1
    store i8 %2, i8* %1, align 1
    %3 = load %struct._IO_FILE*, %struct._IO_FILE** @stdin, alig
n 8
    %4 = call i32 @__isoc99_fscanf
(%struct._IO_FILE* %3, i8* @getelementptr inbounds ([3 x i8], [
3 x i8]@_str, i32 0, i32 0), i8* %1)
    %5 = load i8, i8* %1, align 1
    %6 = sext i8 %5 to i32
    %7 = icmp sgt i32 %6, 0
    br i1 %7, label %8, label %14

; <label>:8:                                ; preds = %0
    %9 = load i8, i8* %1, align 1
    %10 = sext i8 %9 to i32
    %11 = mul nsw i32 %10, 2
    %12 = trunc i32 %11 to i8
    store i8 %12, i8* %2, align 1
    %13 = load i8, i8* %2, align 1
    call void @printHexCharLine(i8 signext %13)
    br label %14

; <label>:14:                                ; preds = %8
    ret void
}
```

图 3 源代码和 IR 样本示例
Figure 3 An example of source code and its LLVM IR

3.2 图结构数据的生成

本节需要实现代码样本控制流图的抽取、代码基本块的预训练和图结构数据拼接。为了能够充分提取代码样本的结构信息，本文采用代码的控制流图作为图神经网络输入样本的框架，同时对控制流图中的基本块代码进行预训练初始化，最后将训练好的基本块初始化向量填回到控制流图结构中，完成图结构数据的生成。

LLVM IR 代码的控制流图抽取采用 LLVM opt 工具实现。opt 命令是模块化的 LLVM 优化器和分析器，它将 LLVM IR 作为输入，在其上运行用于实现指定优化或分析的 PASS 模块，然后输出优化的文件或分析结果。通过 opt 调用生成 CFG 的 PASS 模块可以生成 dot 文件格式的控制流图。

针对需要根据控制流图基本块内的代码信息，对基本块进行初始化向量训练，本文方案采用 word2vec 算法^[15]实现基本块初始化向量的预训练。基本块中的代码是完全顺序结构，不再具有跳转指令，本文把基本块中的代码语句经过预处理之后当作 word2vec 训练语料中的一句 sentence，遍历所有的训练集基本块代码生成语料库。在该语料库上使用 word2vec 算法训练语料库词典，也就获得了所有单词的词嵌入向量，进一步通过取基本块中所有词向量的平均值即可获得基本块的初始化特征向量。

本文采用 pytorch 深度学习框架进行神经网络模型开发和训练。PyG (pytorch_geometric) 是基于 pytorch 的一个图神经网络框架，它定义了一种表示图结构数据的数据类型。本文实现了从 dot 形式的文件到 PyG 图数据类型之间的转化，转化后的图数据包括 $2 \times \text{num_edges}$ 维度的边特征向量、 $\text{num_nodes} \times \text{num_node_features}$ 维度的节点特征向量和一维的全局标签向量。其中， num_edges 为控制流图中边的数量， num_nodes 为控制流图

中节点的数量，也就是基本块的数量，`num_node_features` 表示基本块节点的特征维度，与前述 `word2vec` 算法的输出有关。从 `dot` 文件到 `PyG` 库 `data` 类型对象的转化过程如算法 1 所示。

算法 1 图数据类型转化算法

输入 `dot Object`

输出 `PyG.data Object`

算法

for `edge` in `dot.edges` do

`src, dst = edge.src, edge.dst`

`src_index = dot.nodes.get_index(src)`

`dst_index = dot.nodes.get_index(dst)`

`PyG.data.edge_index.add ([src_index, dst_index])`

for `node` in `dot.nodes` do

`node_value = preprocess(node.value)`

`node_vector = Word2Vec.infer (node_value)`

`PyG.data.x.add(node_vector)`

`PyG.data.y = dot.label #1/0`

return `PyG.data`

图 3 所示的代码样本的 `dot` 类型图文件如图 4 所示，经过算法 1 转化之后，生成的 `PyG.data` 类型的数据示例如图 5 所示。该样本有 3 个节点、3 条边，标签为 `bad`，采用 256 维的 `word2vec` 算法进行向量初始化，生成的 `PyG.data` 数据中边特征向量维度为 2×3 ，节点特征向量维度为 3×256 ，标签为 1，代表该样本为负样本。

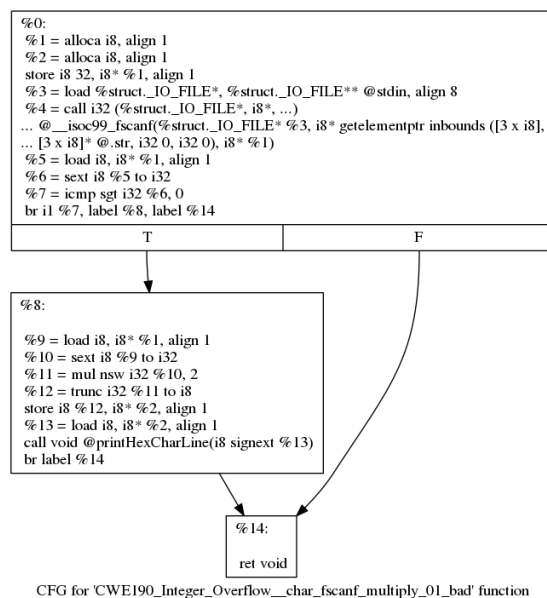


图 4 dot 类型样本示例

Figure 4 An example of dot sample

3.3 图神经网络设计

Gilmer 等^[14]提出的 MPNN 架构是目前大多数图神经网络的顶层抽象，这种信息传播模型的公式如下所示。

$$x_i^t = U^t(x_i^{t-1}, f_{j \in N(i)} M(x_i^{t-1}, x_j^{t-1}, e_{j,i})) \quad (1)$$

$$\hat{y} = R(\{x_i^T \mid i \in G\}) \quad (2)$$

整个网络的前向传递可以分为节点初始化、节点信息传播和状态更新、信息聚合 3 部分。每一次运行 t 时，每个节点 x_i^{t-1} 会通过传播函数 M

```
> data: Data(cwe=CWE190, edge_index=[2, 3], x=[3, 256], y=[1])
> special variables
> function variables
  cwe: 'CWE190'
  edge_attr: None
> edge_index: tensor([[0, 0, 1],
  face: None
> keys: ['x', 'edge_index', 'y', 'cwe']
  norm: None
  num_edge_features: 0
  num_edges: 3
  num_faces: None
  num_features: 256
  num_node_features: 256
  num_nodes: 3
  pos: None
> x: tensor([[ -7.0296e-01, -1.6377e-01,  5.2884e-01,  5.2311e-02, -9.0314e-01,
> y: tensor([1])
```

图 5 `PyG.data` 样本数据示例

Figure 5 An example of `PyG.data` sample

汇聚所有邻居节点 x_j^{t-1} 和相应边的状态信息 $e_{j,i}$ ，在传播结束之后节点会基于历史状态信息 x_i^{t-1} 和传播函数结果通过状态更新函数 U 进行状态更新，如式(1)所示。在消息传递过程中，隐藏层节点的状态可以进行更新，其中关键部分在于中间多次的信息传播和状态更新。通过读出函数 R 将所有节点状态汇聚成预测标签值输出，如式(2)所示。本文方案采用卷积神经网络^[15]实现，图卷积神经网络公式如式(3)所示。

$$x_i^t = \sum_{j \in N(i) \cup \{i\}} \frac{1}{\sqrt{\deg(i)} \cdot \sqrt{\deg(j)}} \cdot (\Theta \cdot x_j^{t-1}) \quad (3)$$

图卷积层需引入自身度矩阵到邻接矩阵中，解决自传递问题， Θ 是用于节点特征变换的权重矩阵，可以通过一个全连接层实现，同时使用

节点的度 $\deg$ 进行正则化。

本文提出的图神经网络结构由多组图卷积和池化层组成的基本单元来完成在图级别上的端到端分类学习问题，其网络结构如图 6 所示。具体来说，在每一组图卷积与池化层中，通过应用图卷积层，本文为每个图样本中的每个节点获得了特征向量转化为具有固定数量维数的表示，并通过池化图中所有节点的表示向量来获得整个图的表示。使用多组基本单元来抽取图结构中每个节点在不同层次上的特征。多组基本单元之间可以直接串联连接，也可以通过残差方式连接。文献[16]指出残差连接方式在多层网络上有更好的效果。残差连接方式如图 7 所示，其表达如式(4)所示。最后，通过应用多层感知器（MLP）和 Softmax 层，获得最终的图标签预测。

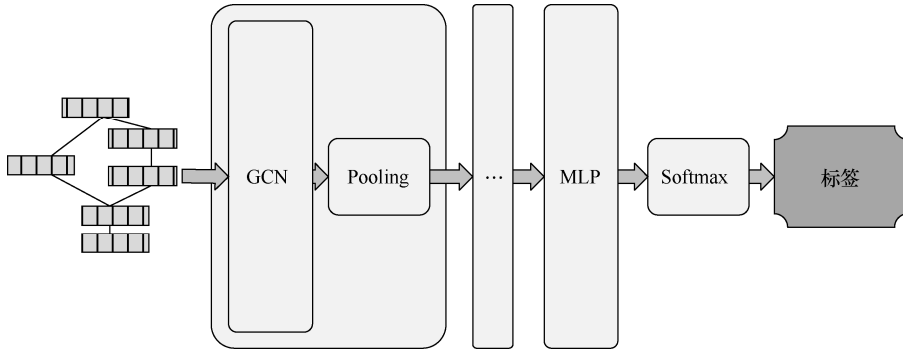


图 6 神经网络结构

Figure 6 The architecture of our neural network

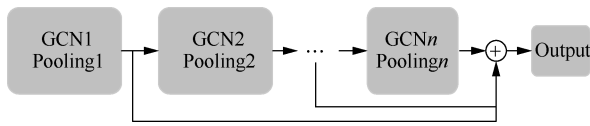


图 7 多层网络残差连接

Figure 7 Multi-layer network residual connection

$$\text{Output} = \sum_l [x_l + M(x_l)] \quad (4)$$

4 实验设计与结果分析

4.1 建立训练与测试数据集

本文采用的数据集是来源于美国国家标准技术研究所 (NIST) 的软件保障参考数据集 (SARD)^[17]。该数据集按照 CWE 编号收录了各种源代码和二进制程序漏洞示例，同时部分具有修复后的代码样本。本文主要采用其 C/C++ 语言的源代码数据

集，使用 Clang 编译器和相关模块获得其代码的 LLVM IR 中间语言表示和控制流图，进而根据方案设计生成图结构数据。本文实验选用了数据集中的整数溢出 (CWE-190, integer overflow or wraparound)、内存泄露 (CWE-401, memory leak) 和释放非堆内存 (CWE-590, free of memory not on the heap) 三大类漏洞代码数据，生成的函数级数据样本数量如表 1 所示。模型训练与测试数据集按照 7:3 比例划分。

表 1 数据集漏洞样本数量
Table 1 The sample number of dataset

样本类型	样本数量
CWE-190	12 802
CWE-401	4 574
CWE-590	6 899

4.2 模型评估指标

本文根据不同类型的数据集训练不同类型的神经网络模型，再根据测试结果评估模型效果。在对模型进行评估时，测试结果的混淆矩阵定义如表2所示。

表2 模型测试结果混淆矩阵定义 Table 2 Confusion matrix for model test results		
实际样本分类	模型预测结果	
	含漏洞	不含漏洞
漏洞样本	TP	FN
不含漏洞样本	FP	TN

根据混淆矩阵的定义，使用以下指标对模型测试结果进行评估。

$$(1) \text{Acc} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN})$$

Acc 被称为分类的准确率，表示模型检测正确的样本占有所有样本数的比例，Acc 越高越好。

$$(2) \text{TPR} = \text{TP} / (\text{TP} + \text{FN})$$

TPR 被称为 Recall (召回率)，表示分类正确的漏洞样本占实际所有漏洞样本的比例，TPR 越高越好。

$$(3) \text{FPR} = \text{FP} / (\text{FP} + \text{TN})$$

FPR 被表示错误检出的漏洞样本数量占有实际不含漏洞样本的比例，FPR 越小越好。

$$(4) \text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

Precision 被称为精确率，表示在模型检出的所有样本中含有漏洞的样本所占的比例，Precision 越高越好。

$$(5) F-1 = 2 \cdot \text{Precision} \cdot \text{Recall} / (\text{Precision} + \text{Recall})$$

F-1 是 Precision 和 Recall 的调和平均，综合了两者的结果，F-1 值越高，代表模型性能越好。

4.3 实验结果与分析

本文的模型测试环境为 CentOS 7.6 操作系统，Intel Xeon CPU E5-2630 处理器，180 GB 内存，GeForce GTX 1080 显卡，8 GB 显存。模型使用 GPU 显卡加速模型训练。

在图结构数据生成部分，word2vec 算法生成的词嵌入参数大小设定为 256，基本块初始化向量的大小即 num_node_features 为 256。

在图神经网络部分，本文在 CWE-190 数据集上测试了具有不同层数网络基本单位和不同层连

接方式的图神经网络的效果。本文选择了从 1 到 6 层图卷积层加池化层网络基本单元，分别使用残差链接和直接连接两种层间连接方式，其训练结果如图 8 所示。

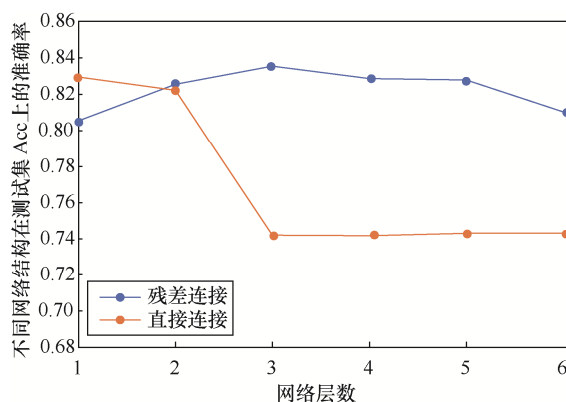


图8 不同网络结构在测试集上的准确率
Figure 8 Accuracy of different network structures on the test dataset

由图 8 可知，直接连接方案在大于 2 层网络层之后出现了准确率的大幅下降，而残差网络连接方案在多层网络环境下具有更好的效果，因此本文选择了效果最好的 3 层残差连接网络。

其他网络参数包括中间图层卷积隐藏层的输入输出通道数，其值均为 128，优化器选择 Adam，学习率为 0.001，损失函数为交叉熵。在完成 50 个 epoch 训练后，测试集上的损失基本稳定，模型能够成功收敛。模型训练准确率和损失曲线如图 9 所示。

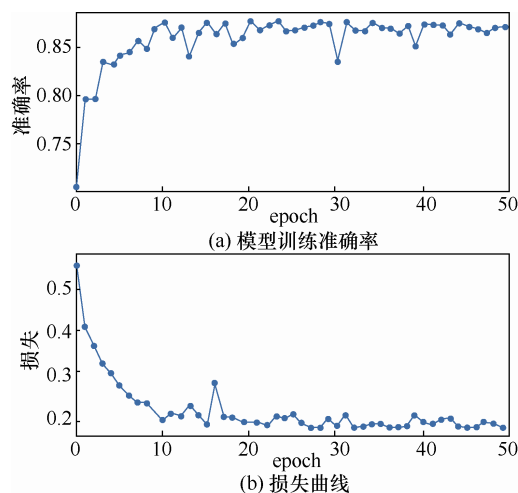


图9 CWE-190 模型训练中测试集准确率与损失曲线
Figure 9 Accuracy and loss curve of test dataset in CWE-190 model training

模型训练时间、测试时间和静态扫描工具扫描时间如表 3 所示。可以看到, 模型训练所需要时间较多, 但当模型训练完成后, 再使用训练好的模型进行漏洞测试时, 运行时间较短, 这也是使用学习模型方法解决漏洞检测问题的优势之一, 根据实验结果可以看到模型测试时间与传统静态扫描工具时间基本持平。

将本文提出的方法与现有的开源静态扫描工具和 LSTM 网络进行检测效果对比。目前没有广泛应用在中间表示的静态扫描工具, 本文选择了 Flawfinder^[18]和 RATS^[19]两种支持 C/C++ 代码静态扫描工具进行对比, 两者在实验时均是对 C/C++ 源代码样本进行扫描和检测, 同时使用 LSTM 网络对中间语言样本进行了训练和检测。测试集上的评估指标对比如表 4 所示。

实验表明, 本文提出的基于控制流图的特征提取与图神经网络模型在开源数据集上的准确率均大幅高于静态扫描工具, 在开源工具普遍召回率为 10% 左右的情况下, 本文的方法召回率可以达到 60% 至 70%, RATS 工具下 CWE-401 上取得

较高召回率 TPR 是由于其 FPR 较高, 即存在较高的误报率。与 LSTM 网络模型相比, 本文提出的图网络模型在各项指标上均更有优势, 同时本文提出的模型方法在测试时间基本持平的情况下取得了更高的准确率指标, 因此, 本文提出的方法相较于传统方法效果更优。

5 结束语

本文提出了一种基于图神经网络的代码漏洞检测方法, 一方面通过词向量初始化基本块节点, 另一方面通过多层图卷积神经网络在控制流图上进行信息传播和聚合, 通过 MLP 层对函数级别的中间语言代码样本进行漏洞检测。本文在开源数据集 3 类漏洞样本上进行分类实验, 取得了良好的效果, 证明该方法相较于传统静态扫描工具更加有效, 通过引入图神经网络进行漏洞特征提取和漏洞检测的可行性得到了证明。进一步的研究包括实验跨语言模型的检测, 验证基于中间语言的模型训练是否可迁移到其他高级语言; 改进基本块向量初始化办法以及实验其他类型的深度图

表 3 不同数据集上的训练、测试和扫描时间
Table 3 Training, testing and scanning time on different data sets

数据集	模型训练时间/s	模型测试时间/s	Flawfinder/s	RATS/s
CWE-190	757.034	2.116	1.97	0.86
CWE-401	263.315	0.718	1.46	0.087
CWE-590	397.540	1.068	1.52	0.15

表 4 不同模型在不同类型测试集上的评估指标对比
Table 4 Comparison of evaluation indicators of different models on different types of test datasets

DataSets	Methods	Acc	TPR	FPR	F-1
CWE-190	Flawfinder	0.534 3	0.103 7	0.027 1	0.183 4
	RATS	0.521 2	0.100 3	0.050 2	0.174 6
	LSTM	0.801 8	0.475 9	0.084 0	0.556 1
	Our Model	0.835 5	0.608 9	0.085 6	0.656 5
CWE-401	Flawfinder	0.413 1	0.185 4	0.322 0	0.253 6
	RATS	0.545 9	0.533 9	0.440 0	0.558 5
	LSTM	0.761 9	0.456 0	0.107 4	0.529 6
	Our Model	0.771 3	0.489 7	0.117 8	0.547 6
CWE-590	Flawfinder	0.563 0	0.123 3	0.004 3	0.218 7
	RATS	0.536 0	0.104 0	0.038 9	0.181 9
	LSTM	0.767 7	0.537 1	0.058 0	0.665 9
	Our Model	0.814 0	0.733 1	0.123 4	0.774 7

神经网络在该方法中的效果,使用更细粒度的图结构使图神经网络能够提取更细粒度的特征等。

参考文献:

- [1] 《国家网络空间安全战略》(全文)[J]. 中国信息安全, 2017(1): 26-31.
“National Cyberspace Security Strategy” (full-text) [J]. China Information Security, 2017(1):26-31.
- [2] WU Z, PAN S, CHEN F, et al. A comprehensive survey on graph neural networks[J]. arXiv:1901.00596, 2019.
- [3] PHAM N H, NGUYEN T T, NGUYEN H A, et al. Detection of recurring software vulnerabilities[C]//25th IEEE/ACM International Conference on Automated Software Engineering. 2010.
- [4] YAMAGUCHI F, GOLDE N, ARP D, et al. Modeling and discovering vulnerabilities with code property graphs[C]//IEEE Symposium on Security and Privacy. 2014.
- [5] LIN G, ZHANG J, LUO W, et al. POSTER: vulnerability discovery with function representation learning from unlabeled projects[C]// ACM Sigsac Conference. 2017.
- [6] 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 (ICMLA). 2018: 757-762.
- [7] 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-(CCS'17). 2017: 363-376.
- [8] LI Y, GU C, DULLIEN T, et al. Graph matching networks for learning the similarity of graph structured objects[C]//Thirty-sixth International Conference on Machine Learning(ICML 2019). 2019.
- [9] YU Z, CAO R, TANG Q, et al. Order matters: semantic-aware neural networks for binary code similarity detection[C]//The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20). 2020.
- [10] DUAN Y, LI X, WANG J, et al. DeepBinDiff: learning program-wide code representations for binary diffing[C]//Proceedings 2020 Network and Distributed System Security Symposium. 2020.
- [11] 李珍, 邹德清, 王泽丽, 等. 面向源代码的软件漏洞静态检测综述[J]. 网络与信息安全学报, 2019, 5(1):1-14.
- LI Z, ZOU D Q, WANG Z L, et al. Survey on static software vulnerability detection for source code[J]. Chinese Journal of Network and Information Security, 2019, 5(1): 1-14.
- [12] GORI M, GABRIELE M, FRANCO S. A new model for learning in graph domains[C]//IEEE International Joint Conference on Neural Networks. 2005.
- [13] SCARSELLI F, GORI M, TSOI A C, et al. The graph neural network model[J]. IEEE Transactions on Neural Networks, 2009, 20(1): 61-80.
- [14] GILMER J, SCHOENHOLZ S S, RILEY P F, et al. Neural message passing for Quantum chemistry[C]//Proceedings of the 34th International Conference on Machine Learning-Volume 70 (ICML'17). 2011: 1263-1272.
- [15] MIKOLOV T, CHEN K, CORRADO G, et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.
- [16] KIPF T N, WELING M. Semi-supervised classification with graph convolutional networks[J]. arXiv preprint arXiv:1609.02907, 2016.
- [17] NIST software assurance reference dataset project[EB].
- [18] Flawfinder[EB].
- [19] Rough-auditing-tool-for-security [EB].

[作者简介]



陈皓(1995-),男,山东潍坊人,上海交通大学硕士生,主要研究方向为深度学习与漏洞检测。



易平(1969-),男,河南洛阳人,博士,上海交通大学副教授,主要研究方向为人工智能安全。