

基于图对比学习的漏洞检测方法

夏鹏程^{1,2}, 曾凡平^{1,2}, 刘 通^{1,2}

¹中国科学技术大学 计算机科学与技术学院 合肥 中国 230027

²中国科学院 无线光电通信重点实验室 合肥 中国 230027

摘要 代码漏洞检测是软件开发过程中的一个重要阶段。随着软件系统变得越来越复杂和庞大,漏洞的数量和种类都在增加。现有的漏洞检测方法通常极度依赖于有标签的数据集,或者在缺乏标签时需要大量的手动工作。本文提出了一种基于图对比学习的漏洞检测新方法(CVD-GCL),致力于让漏洞检测技术摆脱标签的限制,完成对漏洞代码特征的深度学习,从而实现源代码进行智能化的漏洞检测。首先,将源代码转换为代码的四种图结构。然后,提出一个新的图嵌入方法,构建一个包含源代码语义、语法信息和代码结构的漏洞代码图数据集。其次,搭建了一个图对比学习框架,并用该框架来聚合图中的特征,不断增强模型捕获代码漏洞特征的能力。最后,利用经过训练的图神经网络编码器,进行漏洞检测任务。现有的方法侧重于有监督的漏洞检测,而本文首次提出并实现无监督的代码漏洞检测方法。为了验证本文提出的方法,本工作在两个著名的开源项目上进行了实验,这两个项目包括丰富且具有高复杂性的真实源代码。首先通过与其他图数据集构建方法进行对比实验,来验证本工作提出的数据集的高效性,证明了在漏洞检测领域,对图进行向量化时,考虑边的属性可以让检测精度更高。然后与最著名的有监督漏洞检测方法进行模型性能的对比,在使用同等规模的训练集情况下,本工作不仅节约数据集标注的成本,而且在检测的精度方面达到69.64%,充分证明了CVD-GCL在漏洞检测领域的有效性。

关键词 漏洞检测; 代码属性图; 图神经网络; 对比学习; 无监督学习

中图分类号 TP309.1 DOI号 10.19363/J.cnki.cn10-1380/tn.2025.09.12

Code Vulnerability Detection Technology Based on Graph Contrastive Learning

XIA Pengcheng^{1,2}, ZENG Fanping^{1,2}, LIU Tong^{1,2}

¹ School of computer science and technology, University of Science and Technology of China, Hefei 230027, China

² Key Laboratory of Wireless-Optical Communications, Chinese Academy of Sciences, Hefei 230027, China

Abstract Code vulnerability detection is an important stage in the software development process. As software systems become more and more complex and huge, the number and types of vulnerabilities are increasing. Existing vulnerability detection methods usually rely heavily on labeled datasets, or require a lot of manual effort when labels are lacking. Our work proposes a new method for vulnerability detection based on graph contrastive learning (CVD-GCL), which is dedicated to freeing vulnerability detection technology from the limitations of labels, realizing deep learning of vulnerability code features, and thus achieving intelligent vulnerability detection of source code. First, convert the source code into four different graphs of the code. Second, a new method of graph embedding is proposed to construct a dataset of vulnerable code graphs containing syntax information, semantic information and code structure of source code. Third, a graph comparison learning framework was constructed and used to aggregate the features in the graph, continuously enhancing the model's ability to capture code vulnerability features. Finally, the trained graph neural network encoder is used to perform the task of vulnerability detection. Existing methods focus on supervised vulnerability detection, while this paper proposes and implements an unsupervised code vulnerability detection method for the first time. To verify the proposed method in this paper, experiments were conducted on two well-known open source projects, which include rich and highly complex real source codes. Our work first validates the efficiency of the proposed dataset by conducting contrastive experiments with other graph dataset construction methods, proving that when embedding graphs for vulnerability detection, considering the attributes of edges can make the detection accuracy higher. Then compared with the most famous supervised vulnerability detection method, and under the condition of using the same scale of training set, our work not only saves the cost of labeling the dataset, but also achieves 69.64% accuracy in detection, which fully proves the effectiveness of CVD-GCL in the field of vulnerability detection.

Key words vulnerability detection; code property graph; graph neural networks; contrastive learning; unsupervised learning

通讯作者: 曾凡平, 博士, 副教授, Email: billzeng@ustc.edu.cn。

本课题得到科技部国家重点研发计划项目(No. 2018YFB2100300)资助。

收稿日期: 2024-01-08; 修改日期: 2024-03-22; 定稿日期: 2025-08-14

1 引言

软件系统在日常生活中参与度不断提高,这就要求社会更加重视软件的安全性和可靠性。编码错误或缺陷通常是软件漏洞的根源,一旦这些漏洞被利用,可能会导致严重的安全问题。漏洞会损害用户的隐私和安全,也可能导致巨大的经济损失。现代软件的复杂性使得手动检测漏洞既繁琐又易错。因此,自动化漏洞检测对提高软件系统的安全性和降低相关风险至关重要。

近年来,代码漏洞检测领域经历了显著发展,并得到了广泛应用。传统的代码漏洞检测方法主要分为静态和动态两类。静态分析是在不执行程序的情况下检查源代码,以便发现潜在漏洞。然而,静态分析除了需要专家定义漏洞规则外,还面临着对复杂代码结构和新型漏洞了解不足的挑战,这一过程耗时且高度依赖专业知识。另一方面,动态分析专注于检测程序执行过程中的错误,但可能会忽略特定执行场景下的漏洞问题,因为其主要关注于代码实际运行的情况,可能无法全面覆盖潜在的漏洞。

随着人工智能和深度学习的发展,利用深度学习技术的新一代漏洞检测方法得以出现。这些方法依赖于训练模型使用大量已知漏洞实例对漏洞特征进行学习,进而识别潜在的漏洞。然而,这些方法通常需要大量的标记数据进行训练,并且主要学习与漏洞相关的特征,这可能会限制对源代码的全面理解,因为它们可能忽略与漏洞无直接关系但仍然重要的代码结构或模式。

图对比学习在多个领域中展示了巨大的潜力。其能够揭示图结构的复杂模式,因此在社交网络分析、推荐系统和生物信息学等领域得到广泛的应用。另外,在代码漏洞检测方面,图对比学习也显示出了可行性,主要体现在以下几个方面。

(1) 复杂的代码结构表示: 代码通常包含复杂的结构,例如控制流和数据依赖关系。借助工具,可以有效地将代码转化为图。

(2) 统一多视图分析: 包括抽象语法树(Abstract Syntax Tree, AST)、控制流图(Control Flow Graph, CFG)和程序依赖图(Program Dependence Graph, PDG)等多种图表达形式,这些图汇集了对源代码不同角度的理解,为代码漏洞检测提供了多重信息。

(3) 稳健的泛化能力: 图对比学习能够解释代码的结构关系,获取更普适、更全面的特征,进而提

升模型的鲁棒性。这对于检测零日(0-day)漏洞至关重要。

(4) 减少对标签的依赖: 图对比学习能有效减轻对数据集标签的过度依赖,降低了数据集标注所需的成本和工作量。这种能力使得算法在无标签的情况下仍能有效学习并提高性能。

本工作提出了一种新的代码漏洞检测方法,通过将源代码转换为图,并运用图对比学习进行深入分析。这种方法能够处理代码的四种不同图结构,全面捕获其语法、语义和结构信息。利用无监督学习,能够从代码中提取全面且深入的信息,避免了对有标签数据集的过度依赖。为验证图对比学习在源代码分析和检测复杂漏洞方面的可行性,本工作提出了一个图数据集的构造方法,生成的图数据集包含了来自FFmpeg和Qemu两个实际项目源代码的四种图,同时对提出的CVD-GCL进行了详细的实验。本研究的贡献如下:

(1) 构造了一个适用于图神经网络学习的图数据集^①。该数据集涵盖了源代码四种主要类型的图结构。通过实验验证,本工作构建的图数据集相较于其他图数据集,在漏洞检测方面,可以获得更高的精度。

(2) 提出了一种新的漏洞检测方法(CVD-GCL)。通过利用源代码生成的代码属性图(Code Property Graph, CPG),并结合图对比学习,能够深层次获取代码信息,进而完成漏洞检测任务。结合本研究提出的图数据集,CVD-GCL可以与有监督学习方法相媲美,有效地完成代码漏洞检测任务。

(3) 研究了单种数据增强方式对漏洞检测的影响。在CVD-GCL模型中,使用不同的增强方式进行组合实验,以寻找在漏洞检测任务中更为有效的增强方法。

本文的其余部分如下: 第2节概述^[1]代码漏洞检测的相关研究。第3节详细介绍^[2]模型CVD-GCL。第4节涵盖实验部分,主要验证^[3]数据集构造方法的优越性、模型的有效性和各种数据增强方式对精度的影响。第5节对本工作进行了总结,并提出了未来的研究方向。

2 漏洞检测相关研究

代码漏洞检测技术经过多年的发展,传统的方法已经十分成熟。深度学习的快速发展为漏洞检测领域带来了新的可能。本节将简要讨论目前漏洞检

^① <https://www.kaggle.com/datasets/pridejoy/graph-dataset-for-qemuand-ffmpeg-project-code>

测技术的相关研究,同时指出它们所存在的局限性。

2.1 基于软件度量的漏洞检测

软件度量是漏洞检测的一种方法,通过分析代码的质量、复杂性等结构信息来发现和预测潜在的漏洞。基于软件度量的漏洞检测涉及对代码的度量进行特征化,将其作为机器学习的输入,通过学习存在漏洞的代码之间的关联,并且利用学到的关系来预测新输入的代码中可能存在的漏洞。

早期基于软件度量的方法主要包括:基于代码扰动^[1-4]、基于依赖性^[1-2]、基于代码复杂度^[1-2]等。然而,实证研究表明,这些早期的方法在一定程度上帮助了开发者改进代码质量和提高安全性,但同时也存在误报或者遗漏的问题^[4-6]。

为减少误报等情况,研究人员对软件度量的统计和提取方法进行了优化,以增强软件度量刻画漏洞代码特征的能力。文献[7]在前人工作基础上,利用函数依赖性、控制依赖等能有效描述漏洞特征的度量来辅助识别漏洞。另外,文献[8]通过对程序进行切片,从切片后的代码段提取软件度量,有效解决了代码中无关语句对决策的影响。

2.2 基于序列的漏洞检测

基于序列的方法就是将代码转化为 Token 序列,并使用神经网络对 Token 进行建模,从而生成代码的向量表示。对 Token 进行向量表示通常使用 N-gram^[9]或者 word2vec^[10]模型进行处理。根据转化为 Token 的原素材类别,可以分为以下几个方面。

基于代码和代码段的方法。文献[11-13]使用词法分析技术将代码转换为 Token 序列,然后使用诸如词袋等方法将 Token 序列转化为相应的向量表示。另外, Li 等人^[13-16]的一系列工作提出将代码中可能存在漏洞的代码截取出来,形成代码段(code gadget),然后对其进行 Token 化和向量化操作。

基于中间代码的方法。中间代码是在代码漏洞检测过程中产生的一种形式化的表示,通常是对源代码进行分析、转换或编译后得到的一种中间形式。最常用的中间代码是 LLVMIR(Low-Level Virtual Machine Intermediate Representation),它可以将高级源代码转成一种结构化的、与硬件无关的表现形式。文献[17]提出的工具 VulDeeLocator,将 SySeVR^[15]中源代码切片替换为生成的 LLVM 中间代码切片,以提供更丰富的代码信息。

2.3 基于图的漏洞检测

基于图的漏洞检测主要将源代码转换为代码的图结构,如 AST 等。利用图来描述代码的优势在于可以充分表达源代码的语义、语法信息和代码的实

际结构信息。根据对图的不同处理方式,可以分为以下两个方面。

将 AST 转化为 Token 的方法。文献[18]提出使用贪婪算法提取覆盖 AST 中所有节点的最短路径,不断更改路径起点得到路径集,然后通过 word2vec 将路径上下文的 Token 进行向量化。文献[19]则是使用深度优先遍历获取 AST 中所有的节点,将得到的节点作为 Token 用 word2vec 等技术进行向量化。文献[20-21]中的实验结果证明,AST 的结构复杂且包含大量无关信息,将 AST 作为漏洞检测的中间表示效果不佳。为此,文献[22]在原始的 AST 的基础上增加控制流信息,得到的 CSAST(Control Flow Abstract Syntax Tree)可以有效提升检测性能。

使用图神经网络的方法。这类工作主要集中在选取合适的中间表示图。Yamaguchi 等人^[23-24]的工作提出 CPG,用于表示代码的语义和结构信息。文献[20]在 CPG 的基础上融入了自然语言信息,以此生成的复合图(Composite Graph)可以更充分表达代码的结构和语义信息。文献[15]通过提取 AST 和 PDG 中的信息,创建了一种称为程序图(Program Graph)的中间表示。然而,实验^[21,25]表明,并非图越大,检测效果就越好,大量的无关信息可能干扰模型学习漏洞特征,导致效果下降。因此,文献[27]中的 DeepWukong 提出使用程序切片从 PDG 中提取子图,并利用图神经网络进行学习。

综上,几种典型的代码漏洞检测方式的优势和劣势如表 1 所示。

表 1 方法的优劣势

Table 1 Advantages and disadvantages of the method		
	优势	不足
基于度量	大部分软件度量简单且易于获取	无法深层次展示代码漏洞特征
基于序列	直观感受程序语法信息和语序信息	线性表示代码,未充分利用代码结构信息 对数据集要求高,处理大型程序时产生的图较为复杂
基于图	可以充分利用代码的信息	

3 CVD-GCL 系统设计

CVD-GCL 包含两个阶段:生成代码的图和进行图对比学习。在生成代码的图阶段,首先获取代码的 AST、CFG 和 PDG,然后合并为 CPG,并完成 CPG 的嵌入操作。而在图对比学习阶段,则是将这些图输入到图对比学习模型中,用于学习代码中存在的漏洞特征。

如图 1 所示, 本节首先在 3.1 节介绍代码的图生成的相关工作, 即漏洞代码的图数据集的构建过程。

然后在 3.2 节介绍 CVD-GCL 如何使用得到的图数据集进行图对比学习。

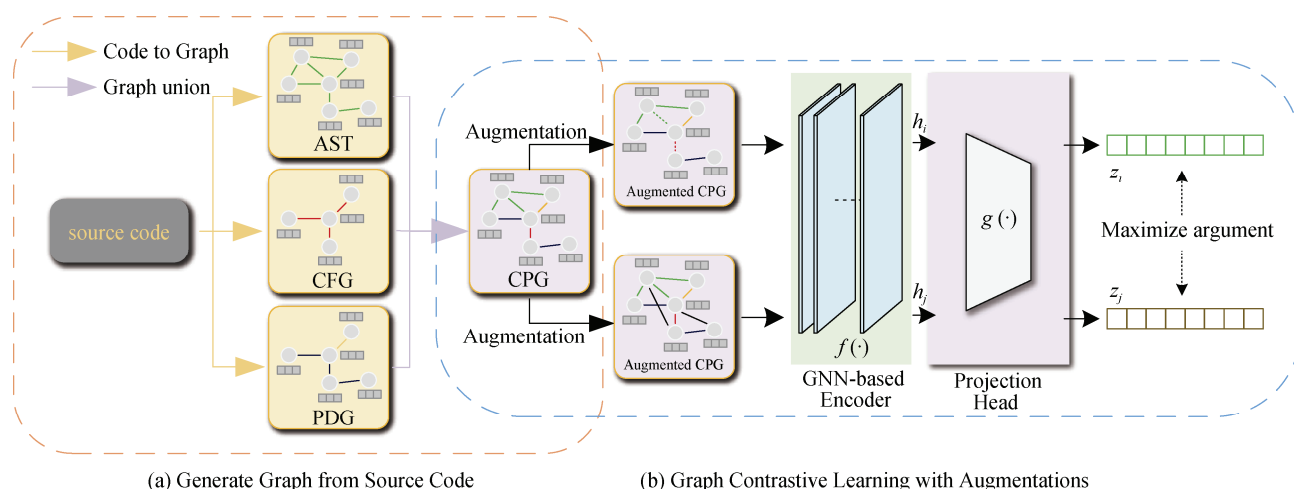


图 1 CVD-GCL 的整体框架
Figure 1 The framework of CVD-GCL

3.1 漏洞代码的图数据集的生成

在程序分析中, 使用不同的图表示(如 AST、CFG 和 PDG)来描述代码的底层语义。这些图能够捕获源代码中的语义、语法信息以及整个程序的逻辑结构。

软件漏洞通常隐藏得很深, 难以被发现。这使得在没有考虑代码的语义、语法和结构信息的情况下, 检测漏洞具有挑战性。本工作通过利用 AST、CFG、PDG 以及它们合并形成的 CPG 来表示源代码。与以往主要强调节点属性的方法相比, 本工作的方法更加关注连接这些节点的边的属性。

数据集的生成主要包括两个步骤: 图的选择和图的嵌入。如图 1(a)所示, 本工作获取了四种图, 然后对图进行规范化处理, 以获得满足实验需求的统一数据集。在 3.1.1 节, 本文通过实际代码展示四类图的实际表现形式; 在 3.1.2 节, 将详细介绍对图的处理方法, 即如何对获取的图进行处理, 以生成可用作图神经网络输入的图数据集。

3.1.1 代码的四种图

本研究通过一个实际代码, 使用 joern^①获取代码的图, 来详细说明在实际工作中代码的四种图所包含的信息。实际代码如图 2 所示。

(1) 抽象语法树(AST)。是由编译器的解析器生成的有序树结构, 可以精确地描述语句和表达式。如图 3 所示, 每个 AST 节点代表程序代码中的一个结构元素。AST 的结构反映了代码的语法布局, 其中父

子节点关系象征着代码中的语法之间的联系。

```

1 void foo()
2 {
3     int x = source();
4     if( x < max )
5     {
6         int y = 2 * x;
7         sink(y);
8     }
9 }

```

图 2 实际 C 语言代码
Figure 2 Actual C code

AST 的创建主要包括词法和句法分析。前者将源代码分解为 Token, 其中每个 Token 表示一个语法单元, 如关键字或运算符。后者根据这些 Token 的语法规则生成 AST。

在 AST 的节点中, 主要包含三个特征: 类型(type)、实际代码(code)和所属子图索引(subgraph index)。其中, 类型中不同的字符串表示这个节点的所属类型, 不同类型的描述如表 2 所示。实际代码表示节点在 AST 中所代表的具体代码, 其中叶子节点对应代码的 Token, 非叶子节点则代表 Token 序列, 即实际的代码语句。子图索引是在生成 AST 时, 因为一段代码可能包含多个函数、条件语句、循环或者其他语句, 所以将每个函数、条件语句或循环都可以作为一个子图。从图 3 中可以看出, 图 2 代码在生成 AST 后共产生了 7 个子图。

①<https://joern.io/>

表 2 节点类型
Table 2 Type of node

类型	描述
METHOD	方法或函数的起始点
BLOCK	代码块节点
LOCAL	局部变量节点
<operator>.x	运算或比较等操作节点
IDENTIFIER	标识符节点
CONTROL_STRUCTURE	控制结构节点, 如 if、while 等
LITERAL	常量节点, 如数字或字符串
METHOD_RETURN	方法或者函数的结束或返回点

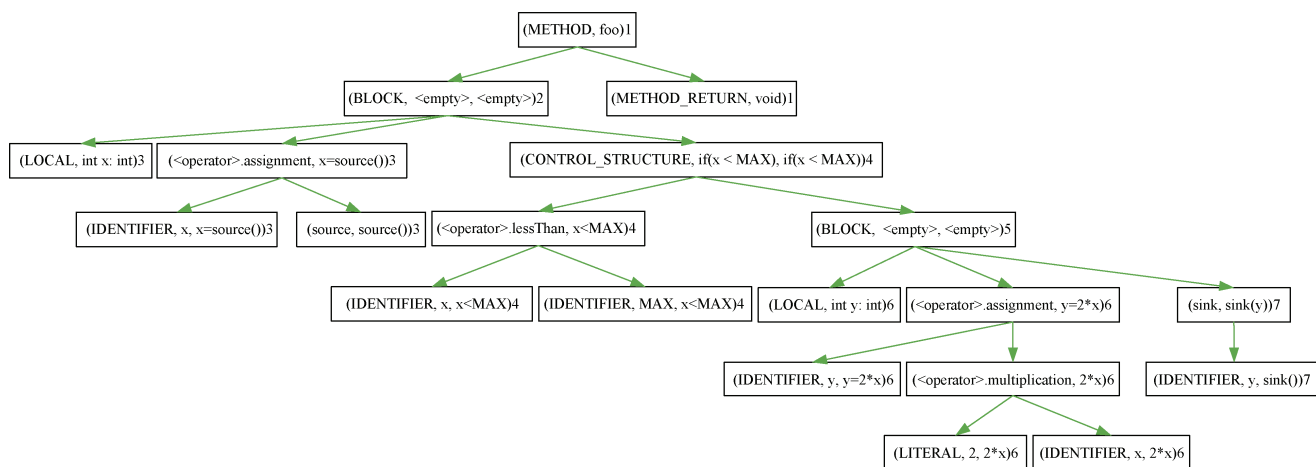


图 3 代码的 AST
Figure 3 AST of the code

本工作将 CFG 的节点与 AST 中的某些节点进行匹配。这种匹配可以分析和理解程序的控制流结构, 并将其与语法结构进行关联, 从而能够更全面地理解程序的结构和执行流程, 进而更好地为下游任务提供支持和服务。

(3) 程序依赖图(PDG)。是一种综合了控制依赖图(CDG)和数据依赖图(DDG)的数据结构, 旨在提供对程序依赖关系的全面视图。在 PDG 中, 节点通常代表程序中的语句或表达式, 边则表示这些语句之间的依赖关系。如图 5 所示, 与 CFG 工作相同, 将 PDG 的节点与 AST 中的部分节点进行匹配。在 DDG 中, 由于两个节点存在数据依赖, 边会描述两个节点之间关于哪个数据存在依赖。

这种依赖关系主要包括两种类型: 控制依赖关系(Control Dependencies), 即一条语句的执行可能受到另一条语句执行的影响; 数据依赖性(Data Dependencies), 包括一个语句的结果会影响另一个语句的流依赖性, 以及一个语句依赖于前一个语句结果的反依赖性。

(2) 控制流图(CFG)。作为编译器设计和程序分析中广泛采用的数据结构, CFG 封装了程序代码的控制序列。它概述了潜在的执行路径, 为静态程序分析奠定了基础。

如图 4 所示, CFG 以节点和边的形式展现程序中的基本块和控制流转移。节点通常代表基本块, 即连续的代码片段, 具有单一的入口和出口点, 表示控制从块的开始进入, 在块的结束处退出。边则表示基本块之间的控制流转移, 标记程序的跳转路径。通过这种方式, CFG 可以有效捕获程序的控制结构, 包括条件(if-else)和循环(for, while)分支。

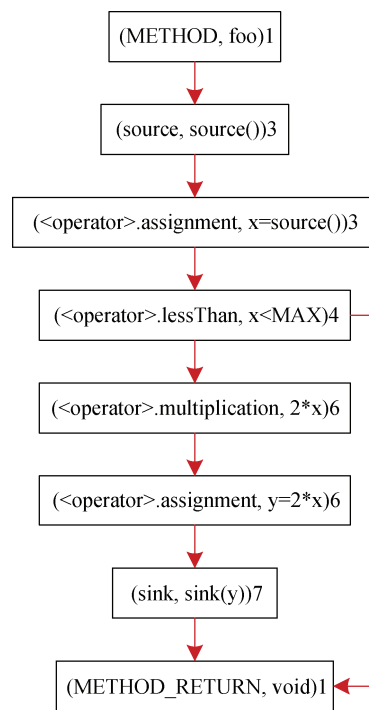


图 4 代码的 CFG
Figure 4 CFG of the code

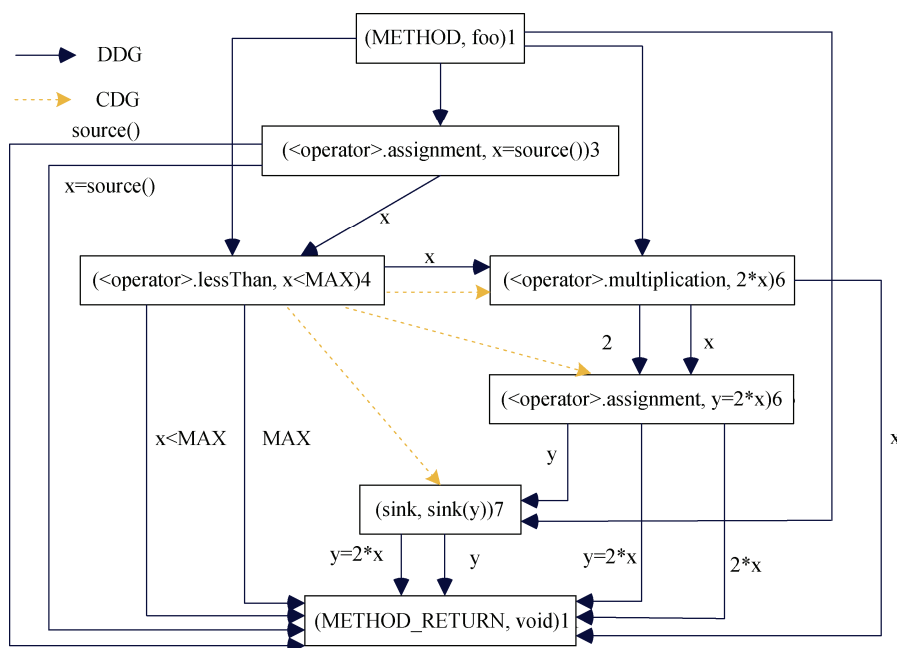


图 5 代码的 PDG

Figure 5 PDG of the code

PDG 的创建旨在提供对程序依赖性的全面和清晰的视图, 有助于进行程序的静态分析和调试。它通常被用于识别和定位程序中的错误, 比如死循环或者未初始化的变量。通过图形化来表示程序的依赖关系, PDG 可以帮助程序员更好地理解和分析程序, 发现潜在的问题并进行修复。

(4) 代码属性图(CPG)。其特点在于将源代码的语法、控制流以及代码依赖关系集成在一起。其形成过程涉及 AST、CFG 和 PDG 的整合。在 CPG 中,

所有的节点来自 AST 的节点, 图中的边来自 AST、CFG 和 PDG 的边。因此, CPG 提供了一种将源代码的多视角整合到一个统一的数据结构中的方法。这种整合能够全面捕获源代码多个方面的属性, 使复杂的程序分析变得更加简单和直观。

图 6 展示了 CPG 如何将这些不同层面的属性整合到一个统一的图结构中。这种综合呈现为程序分析提供了更丰富和全面的视角, 使得对程序的理解和审查更为全面深入。

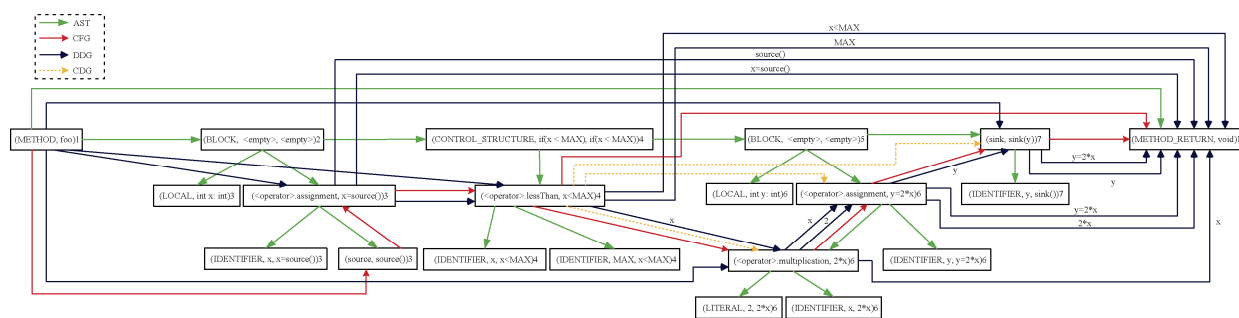


图 6 代码的 CPG

Figure 6 CPG of the code

3.1.2 图的嵌入

在得到上述图后, 接下来的重点在于识别节点和边的属性, 并制定算法进行图嵌入操作, 以便后续输入到图神经网络进行训练。CPG 是 AST、CFG 和 PDG 的结合, 包含三个图的所有信息。本节将基于 CPG, 依次介绍整图的定义、节点的嵌入方法和边

的嵌入方法。

(1) 整图的定义。CPG 是一个有向图, 节点和边具有不同的属性, 定义图为 $G=(V, E, L)$, 其中, V 表示节点集, E 是有向边的集合, L 是后续用于测试模型效果的标签集。如图 7 所示, 在 CPG 图中, 包含 4 种边的类型, 同时节点具有各自的节点属性。

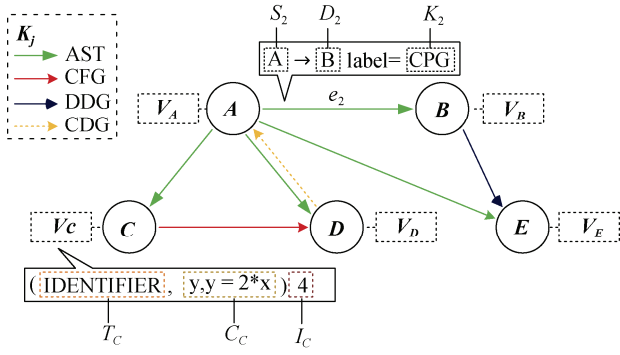


图 7 CPG 的详细属性

Figure 7 Detailed properties of CPG

首先, 定义节点集为 $V = \{v_1, v_2, \dots, v_n\}$, 其中 $v_i = \{T_i, C_i, I_i\}$, $i \in (1, 2, \dots, n)$, n 表示图中节点总数, T_i 表示节点 v_i 的类型(type), 如表 2 所列举, C_i 表示节点 v_i 中实际代码(code), I_i 是该点 v_i 在整图中所属的子图索引(subgraph index)。

然后, 定义边集合为 $E = \{e_1, e_2, \dots, e_m\}$, 其中 $e_j = \{S_j, D_j, R_j\}$, $j \in (1, 2, \dots, m)$, m 表示图中边的总数, S_j 表示边 e_j 的出发点(source), D_j 表示边 e_j 的终点(destination), R_j 则表示图中边的类型(repr), 其中 R_j 的类型有 AST, CFG, DDG 和 CDG, 分别是前文所述三种图包含的类型。

(2) 节点的嵌入过程。节点 $v_i = \{T_i, C_i, I_i\}$ 中, T_i 遵循预先设定的组合, 数量较少, 因此使用独热编码(One-Hot Encoding)。 C_i 代表源代码, 无法有效枚举, 本工作使用 Glove-trained^[28]词嵌入(word embedding)方式来捕捉上下文信息, 进行嵌入操作。由于 I_i 原本为数字, 所以保持 I_i 不变。关于本部分的详细配置请看实验部分 4.1 节。

(3) 边的嵌入过程。根据图中边的种类数量不同, 本工作将图分为两类: 只有一种边的单图(single graph)和有两种及以上不同类型边的复合图(composite graph)。单图包括 AST 和 CFG, 复合图有 PDG(DDG+CDG)和 CPG(AST+CFG+PDG)。

在图神经网络中, 图结构通常以邻接矩阵的形式表示。本工作通过对邻接矩阵的权值进行处理, 来区分不同边的类型。在定义矩阵权值的过程中主要考虑两个问题: (i)在两个节点之间可能存在不同类型的边, 需要设计一个方法表示节点之间的边的类型; (ii)两个节点之间可能存在多条相同类型的边, 且边的属性中也包含实际代码(DDG)。

针对问题(i), 对于单图, 由于图中只存在一种类型的边, 直接使用无权邻接矩阵(Unweighted Adja-

cency Matrix)来描述图的结构。

$$A[i][j] = \begin{cases} 1, & \text{if } \langle v_i, v_j \rangle \in E \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

如公式(1)所示, $A^{n \times n}$ 是图的邻接矩阵, 其中 n 表示图中节点的个数。如果有向图中的节点 v_i 存在指向 v_j 的边, 那么将邻接矩阵中第 i 行第 j 列置为 1, 反之则置为 0。

对于复合图, 如图 8 所示, 在同一位置出现不同类型的三条边, 使用带权邻接矩阵(Weighted Adjacency Matrix)来描述存在的边的类型。其中, 权重由一个 4 位的二进制数值转换为十进制数值来表示。如公式(2)所示。

$$A[i][j] = \begin{cases} 0bw_1w_2w_3w_4, & \text{if } \langle v_i, v_j \rangle \in E \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

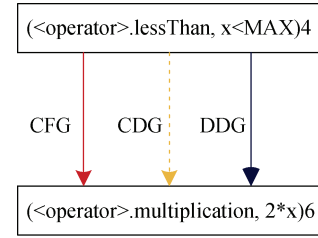


图 8 复合图中存在的多条边问题

Figure 8 The problem of multiple edges in a composite graph

由于边的数量通常可以反映图中的关系密度和复杂性, 所以本工作赋予权重的依据为各类型的边在 CPG 中的数量情况。因此, 在公式(2)中, w_1 代表 AST 边, w_2 代表 DDG 边, w_3 代表 CFG 边, w_4 代表 CDG 边。在有向图 G 中, 若节点 v_i 存在指向 v_j 的多条不同类型的边, 则对应位置设置为 1, 反之则为 0。如图 8 例子所示, 在邻接矩阵中, 两个节点之间边的权重为 0b0111, 转化为十进制为 7, 即两个节点之间的权重为 7。

针对问题(ii), 如图 9(a)所示, 两个节点之间存在相同类型的边, 且边的属性中包含实际代码。通过对超过 1000 个源代码的 CPG 进行分析, 我们观察到问题(ii)仅存在于 DDG。其理论依据在于, 当函数中存在分支控制流、循环结构、函数调用等情况时, 两个节点之间可能存在多个数据依赖或反依赖, 且边的属性中存在的实际代码是数据控制的对象。

经过进一步分析, 发现边的属性中出现的实际代码通常封装在边的出发点和终点的节点属性中。因此, 在定义邻接矩阵时, 将两个节点之间相同的边进行合并, 并将边的属性中的代码部分交给相邻

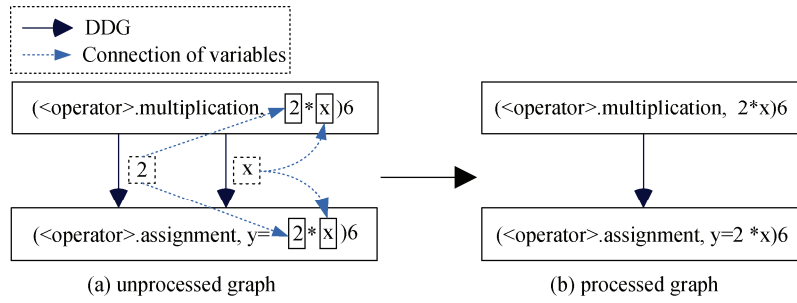


图 9 节点间存在相同类型的边

Figure 9 The same type edges between nodes

的节点来表达, 以更好地处理这些情况, 如图 9(b) 所示。

综上所述, 图嵌入的整体过程参考算法 1。

算法 1 图嵌入算法.

输入: 图 G

输出: 图的节点特征矩阵 X ; 图的邻接矩阵 A

- 1: 使用 One-Hot 对 T 进行编码, 得到 t ;
- 2: 使用 Glove 对 C 进行嵌入, 得到 c ;
- 3: 节点特征矩阵 $X \leftarrow \text{concat}(t, c)$;
- 4: 忽略所有边上的实际代码;
- 5: 将两个节点之间的相同类型的边保留一个;
- 6: IF G 是单图:
- 7: $A \leftarrow$ 使用公式(1);
- 8: ELSE:
- 9: $A \leftarrow$ 使用公式(2);
- 10: return A, X ;

3.2 CVD-GCL 图对比学习

如图 1(b)中所示, 图对比学习分为四个阶段: 数据增强、图神经网络编码、投影头(Projection Head)和对比损失函数优化模型, 图对比学习过程如算法 2 所示。

(1) 数据增强。数据增强的目的在于通过特定的数据处理方式, 在尽可能不影响图中关键信息, 保证增强后的图不会丢失对下游任务关键特征的情况下形成新的合理数据。

在图像的对比学习中, 旋转和裁剪是最基本也是最普遍的增强方式, 相对较为统一。然而, 由于图来自不同的领域, 不容易找到统一的增强方式。因此, 针对不同领域的图数据集, 需要有针对性地选择数据增强的方法。关于各种图增强方式的数学解释, 可以参考文献[29]。

图数据增强的定义为, 在一个由 $\mathcal{K}$ 个图组成的数据集中, 取出图 $G \in \{G_k : k \in \mathcal{K}\}$, 增强图 $G_1 \sim$

$q_1(\cdot|G)$, 其中 $q(\cdot|G)$ 表示增强图的分布。

本工作主要关注于图级别的数据增强。在每次迭代中, 使用两个图增强函数 q_1 和 q_2 对原图 G_{CPG} 进行增强, 得到两个不同的视图 G_{CPG_1} 和 G_{CPG_2} , 满足 $G_{\text{CPG}_1} \sim q_1(\cdot|G_{\text{OPG}})$, $G_{\text{CPG}_2} \sim q_2(\cdot|G_{\text{OPG}})$, 其中 G_{CPG_1} 和 G_{CPG_2} 为正样本对。

(2) 图神经网络编码(GNN encoder)。GNN 是对图上所有属性进行的一个可以优化的变换。根据图的结构和节点的特征 $X_v X_v$, 学习节点的嵌入 h_v 或者整图的嵌入 h_G 。考虑一个 K 层 GNN 编码器 $f(\cdot)$, 其在第 k 层的传播过程如下:

$$\begin{aligned} a_v^{(k)} &= \text{AGGREGATE}^{(k)} \left(\left\{ h_u^{(k-1)} : u \in \mathcal{N}(v) \right\} \right) \\ h_v^{(k)} &= \text{COMBINE}^{(k)} \left(h_u^{(k-1)}, a_v^{(k)} \right) \end{aligned} \quad (3)$$

其中, X_v 表示节点集合 V 中节点 v 的特征向量, a 表示聚合节点 v 邻居节点后得到的特征, $h_v^{(k)}$ 代表节点 v 在第 k 层的特征向量, 初始化 $h_u^{(0)} = X_v$ 。 $\mathcal{N}(v)$ 是节点 v 的邻居集合。整个过程是先聚合节点 v 邻居节点的上一层表示, 再与节点 v 自身的上一层表示结合。

本工作的编码器 $f(\cdot)$ 用于获取视图 G_{CPG_1} 和 G_{CPG_2} 的图特征向量 h_1 和 h_2 。本工作对 GNN 结构不做限制, 本工作使用的 GNN 编码器是图同构网络^[30] (Graph Isomorphism Network, GIN)。为保证在建模时实现内部映射, 使用多层感知机^[31] (Multilayer Perceptron, MLP) 来模拟函数的组合, 用公式(4)更新 GIN 的节点向量。

$$h_u^{(k)} = \text{MLP}^{(k)} \left(\left(1 + \varepsilon_{(k)} \cdot h_u^{(k-1)} \right) + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)} \right) \quad (4)$$

其中, ε 表示可学习参数。

(3) 投影头(Projection Head)。负责将输入数据编码为更加有意义且区分性更强的表示。这些表示通

常以低维向量的形式存在, 用于捕捉数据的重要特征。投影头使用一个非线性变换函数 $g(\cdot)$ 将 h_1 和 h_2 分别映射到另一隐空间(latent space)中, 得到 z_1 和 z_2 , 其主要作用是计算对比误差。如公式(5), 本文使用两层 MLP 来进行这种映射。

$$z = \text{MLP}(h) \quad (5)$$

(4) 对比损失函数。对比损失函数 $\mathcal{L}(\cdot)$ 被用来最大化 z_1 和 z_2 之间的一致性, 本工作采用的是 InfoNCE^[32], InfoNCE 可以分解为两个基本组成部分: 对齐性(Alignment)和均匀性(Uniformity)。对齐性与正样本相关, 希望让正样本之间的距离不断缩小。均匀性只与负样本相关, 目的在于让所有的样本的特征尽可能在单位超球面(unit hypersphere)上均匀分布。InfoNCE 的公式可以写为

$$\text{infoNCE} = -\log \frac{\exp\left(\frac{\text{sim}(z_i, z_j)}{\tau}\right)}{\sum_{k=1}^{2N} \mathbf{1}_{[k \neq i]} \exp\left(\frac{\text{sim}(z_i, z_k)}{\tau}\right)} \quad (6)$$

其中, τ 表示温度系数, 作用是控制模型对负样本的区分度。在 GNN 与训练期间, 每个 mini-batch 有 N 个图, 通过数据增强一共有 $2N$ 个图输入到模型中, 对比的正样本对是同一原图的两个增强视图, 负样本对则是原图的增强视图与其他图的增强视图组成的样本对。

图对比学习如算法 2 所示, 将经过 3.1 节的图输入到图对比学习模型后, 首先使用两个图增强函数 q_1 和 q_2 得到原图的两个增强视图; 其次使用 GNN 编码器 $f(\cdot)$ 对两个视图进行图中信息的学习; 然后投影头映射 $g(\cdot)$ 将编码后的结果进行映射, 映射到另一隐空间, 用于计算对比误差; 最后使用对比损失函数 InfoNCE 对同一个 batch 中的正负样本进行计算, 实质是最大化两个增强视图的表示之间的互信息。需要注意的是, 算法 2 第 5 行中正样本为原图的两个增强视图, 负样本为除正样本外的所有视图。

算法 2. 图对比学习算法.

输入: $G = (A, X)$

输出: 模型准确率 acc, F1 指标

- 1: For epoch $\leftarrow 1, 2, \dots$ do
- 2: $G_1 \leftarrow q_1(\cdot | G)$ 和 $G_2 \leftarrow q_2(\cdot | G)$; //数据增强
- 3: $h_1 \leftarrow f(\cdot | G_1)$ 和 $h_2 \leftarrow f(\cdot | G_2)$; //GNN 编码
- 4: $z_1 \leftarrow g(\cdot | h_1)$ 和 $z_2 \leftarrow g(\cdot | h_2)$; //映射
- 5: 使用公式(6)计算损失值;

- 6: 节点特征, 全局特征 $\leftarrow$ 池化后合并特征, 前向传播, 更新模型参数;
- 7: 使用 SVM 进行二分类, 测试效果, 得到 acc 和 F1;
- 8: return acc, F1;

4 实验设计与结果分析

本工作设计实验主要用于回答以下四个问题:

Q1: 与现有的方法相比, 本文提出的图嵌入方法(3.1 节)得到的图数据集效果如何?

Q2: 图对比学习在漏洞检测领域是否可行?

Q3: CVD-GCL 使用 CPG 进行训练, 与使用单个图(AST、CFG 和 PDG)训练相比, 性能变化如何?

Q4: 在漏洞检测中, 不同数据增强方式的影响?

4.1 实验设计

(1) 数据集介绍。本工作使用了由 Devign^[20]提供的带标签源代码数据集, 该数据集的信息列于表 3。

尽管图对比学习不依赖带标签的数据集, 为验证构建的图数据集的有效性和 CVD-GCL 模型的可行性, 在模型测试阶段, 需要使用标签信息。

表 3 数据集信息

Table 3 Data sets overview

项目	有漏洞代码	无漏洞代码	总数
FFmpeg	4981	4788	9769
Qemu	7479	10070	17549
总数	12460	17858	27318

其中数据集中每个代码的组成为一个四元组 $\{\text{project}, \text{commit_id}, \text{target}, \text{func}\}$, 其中 project 表示代码所属的项目, commit_id 表示代码的提交 id, target 表示该代码是否存在漏洞(有则为 1, 无则为 0), func 表示实际代码。

本工作构建的图数据集主要细节如表 4 所示。

表 4 图数据集信息

Table 4 Graph datasets overview

	AST	CFG	PDG	CPG
FFmpeg	FFmpeg_ast	FFmpeg_cfg	FFmpeg_pdg	FFmpeg_cpg
Qemu	Qemu_ast	Qemu_cfg	Qemu_pdg	Qemu_cpg
all	all_ast	all_cfg	all_pdg	all_cpg

(注: 共生成 12 个图数据集, 其中 all 表示 Qemu 和 FFmpeg 之和)

(2) 实验环境介绍。本工作的第一部分涉及对 27318 个源代码的处理, 生成可以输入图对比学习模型的图数据集; 第二部分使用 PyGCL 库来构建图对比学习模型。具体的实验环境配置和软件版本信息如表 5 所示。

表 5 实验环境详细参数

Table 5 Detailed parameters of experimental environment

硬件设置	CPU 信息	Intel Xeon W-2255 CPU @ 3.70GHz
	GPU 信息	NVIDIA GeForce RTX 3080
	内存大小	64GB
	磁盘大小	4TB
操作系统	Ubuntu 20.04.6 LTS	
软件信息	Python 3.8、Pytorch 2.0、Pytorch-Geometric 2.3、DGL	
	1.1+cu118、sklearn 1.2、Numpy、tqdm、NetworkX、PyGCL	

(3) 实验指标。本工作主要使用 2 个指标来衡量 CVD-GCL 的漏洞检测性能, 分别是精度和 $F1$ -指标。

(i)精度: 在所有被分类为正例(漏洞存在)的样本中, 真正例的比例。

$$ACC = \frac{TP}{TP + FP}$$

(ii) $F1$ -指标: 是精度和召回率的加权平均值, 综合考虑精度和召回率(Recall, R)。

$$F1 = \frac{2 \times ACC \times R}{ACC + R}$$

其中, TP 表示成功预测为漏洞的数量, FP 表示误报为漏洞的样本数, TN 表示成功预测为无漏洞的样本数, FN 表示误报为无漏洞的样本数。 R 表示准确预测为有漏洞的样本数量占全部实际有漏洞样本数量的比例, R 的公式如下:

$$R = \frac{TP}{TP + FN}$$

(4) 实验参数。生成 Glove 语料库、模型搭建和模型训练等方面涉及许多实验参数, 相关参数如表 6 所示。

(5) 实验中涉及的图增强方式如表 7 所示。在实验 1 和实验 2 中, CVD-GCL 使用的增强方式为 $q_1=\{ES\}$, $q_2=\{ND(pn=0.1), FM(pf=0.1), EA(pe=0.1)\}$ 。

(6) 详细实验过程。如下所示, 在获得 Devign 数据集后, (i)使用自动化脚本将数据集转化为源代码文件集; (ii)使用 joern 对文件集中的源代码进行逐个分析, 生成源代码的 AST、CFG、PDG 和 CPG, 并对四类图进行处理, 使得到的图符合图数据集处理的要求; (iii)如 3.1 节所述, 对获得的图进行嵌入操作, 获得图数据集; (iv)将图数据集输入到 3.2 节介绍的图对比学习模型中, 对 GNN 编码器进行学习; (v)使用测试集, 对 GNN 编码器的效果进行检测, 获得模型的精度和 $F1$ 得分。

过程.实验过程

1: 源代码文件集 $\leftarrow$ Devign 数据集;

- 2: 源代码图 $\leftarrow$ 源代码文件集;
- 3: 漏洞代码的图数据集的生成;
- 4: 将图数据集输入到模型, 进行训练;
- 5: 测试 GNN encoder 的精度和 $F1$ 得分;

表 6 实验参数列表

Table 6 List of experimental parameters

参数	解释	值
epoch	训练迭代次数	100
batch	训练时批量大小	256
train_ratio	训练集比例	0.8
validation_ratio	验证集比例	0.1
test_ratio	测试集比例	0.1
mode	图的对比方式	G2G
lr	Adam 学习率	0.01
num_layers	图卷积层的数量	2
hidden_dim	隐藏层的维度	32
τ	InfoNCE 温度系数	0.2
MAX_ITER	Glove 训练迭代次数	400
VECTOR_SIZE	Glove 编码维度	126
WINDOW_SIZE	Glove 窗口大小	15

(注: Glove 语料库主要用于对节点特征中的源代码进行嵌入)

表 7 图增强方式

Table 7 Different graph augmentation methods

增强方式	解释	参数
Edge Adding (EA)	随机增加 p 的边	pe
Feature Masking (FM)	随机掩盖 pf 特征	pf
Feature Dropout (FD)	随机丢弃 pf 输入特征	pf
Edge Attribute Masking (EAR)	随机掩盖 pe 的边属性	pe
Node Dropping (ND)	随机删除 pn 节点及相邻边	pn
Node shuffling (NS)	随机将 pn 节点重新排序	pn
Ego-net Sampling (ES)	自我网络抽样	-

(注: 表中的参数都表示比例)

4.2 实验结果及分析

(1) 图数据集效果验证。为评估本工作提出的代码图嵌入方法的性能, 本工作与 Devign 提出的图嵌入方法进行对比。首先, 本工作依次将得到的两个图数据集作为输入分别放入 Devign 模型和 CVD-GCL 中。实验过程中, 严格控制实验条件, 确保实验的有效性和可对比性, 实验结果如表 8 所示。

对比工作是 Devign 提出的数据集处理模式, Devign 将 AST, CFG, DFG 以及 NCS(Natural Code Sequence)进行联合, 使用预训练的 word2vec 模型和构建整个项目源代码文件上的代码语料库, 对源代码进行编码, 最后将其作为节点的初始表征, 以此得到作为输入的数据集。

表 8 图数据集的性能比较

Table 8 Performance comparison of graph dataset

方法	数据集	Qemu		FFmpeg		Qemu+FFmpeg	
		ACC	F1	ACC	F1	ACC	F1
Devign	D_1	69.84	73.07	64.58	73.55	66.26	71.16
	D_2 (CPG)	72.53	69.35	69.41	70.97	70.55	69.21
CVD-GCL	D_1	59.13	74.25	52.71	64.79	55.19	69.42
	D_2 (CPG)	71.62	67.08	65.43	65.72	71.55	63.29

(注: D_1 表示 Devign 提出的图嵌入的处理方式得到的数据集, D_2 为本工作提出的 CPG 图数据集, 在此过程中 Devign 方法使用图数据集的标签信息, CVD-GCL 不使用标签)

针对 Q1, 观察表 8, 对实验结果进行分析。在 Devign 方法中, 使用 D_2 图数据集得到的结果优于 D_1 图数据集, 精度最高提升 4.83%, 当将 D_1 图数据集应用到 CVD-GCL 中时, 发现精度大幅度降低, 降幅最大达到 11.87%。作者认为, 产生这种现象的原因有两点。其一, 由于 D_1 除使用 AST, CDG 和 DFG 外, 还额外引入 NCS, 这就使得 CVD-GCL 在对图进行学习时, 学习了许多与分类任务不相关的特征; 其二, 由于未考虑到图的边属性, 使得输入的邻接矩阵只用于表示节点之间存在边, 而忽略了边的含义。通过数据可以回答 Q1, 使用 CPG 构建图数据集时, 保留每个节点的源代码部分, 同时考虑边的属性, 对于漏洞检测任务的精度有明显的正向作用, 本工作提出的图数据集处理方法优于现有的工作提出的图数据集处理方法。

(2) CVD-GCL 模型性能评估。最新的漏洞检测方法中, 有监督方法占据主导地位, 无监督方法工作空白。本工作与传统的 CNN+attention 和 Path+attention、著名的 Devign 模型和最新的 Funded 模型和 VulCNN 模型进行详细的对比, 来验证 CVD-GCL 的性能, 以及证明图对比学习在漏洞检测方面的可行性。

(i) Devign: 该模型对编程代码的语义进行编码, 捕获漏洞特征。提出的 Conv 模块, 从门控递归单元中获取图的异质节点特征作为输入, 进行图级别的分类。

(ii) Funded^[25]: 该模型将源程序的 AST 和 PDG 进行组合, 构建程序图。同时使用四个基于门控循环单元的堆叠嵌入模型组成门控图神经网络(GGNN)。将程序图输入到 GGNN 来进行漏洞检测。

(iii) VulCNN^[33]: 本模型将源代码转化为 PDGs 后, 在利用图神经网络进行源代码漏洞挖掘的基础上引入了社交网络中心性分析。从而在尽可能保留程序细节的同时, 将函数的源代码转化为图像, 再利用图像分类技术进行漏洞检测。

(iv) CNN+attention^[34]: 该模型将代码看作自然语言, 最初使用词袋的方式对代码进行嵌入。后来意

识到结构的重要性, 模型利用 AST 来分配适当权重, 然后将权重输入到卷积神经网络中进行学习。

(v) Path+attention^[35]: 本模型主要方法是将源代码表示为固定长度的向量, 确保模型可以有效捕捉代码中语义的细微差别。

Funded 和 VulCNN 使用本工作的原始代码数据集, 通过各自的构建方式生成数据集, CVD-GCL 和 Devign 使用本工作提出的图数据集, 输入到五个对比模型以及 CVD-GCL 来进行评估, 实验结果如表 9 所示。

针对 Q2, 通过表 9 前五, 来评估 CVD-GCL 的漏洞检测能力。与 CNN+attention 和 Path+attention 相比, CVD-GCL 在精度和 F1 得分方面都表现出优越的结果, 精度最高提高 18.43%, 平均提升超过 14%。CVD-GCL 与较近的有监督学习工作相比, 效果略有不。然而, 由于 CVD-GCL 属于无监督学习, 不依赖于标签, 在同等规模大小的数据集中, 精度与 Devign 相比最大差距为 4.85%, 平均差异仅降低 2.75%。与最新的工作 Funded 相比, 精度上最大降低 5.19%。与 VulCNN 工作相比, 精度最多仅降低 2.26%。由此可以说明, 本工作提出的 CVD-GCL 可以有效完成漏洞检测任务, 即使在没有标签的情况下, 分类精度可以接近有监督学习。同时证明图对比学习运用在漏洞检测领域是可行且有效的。

针对 Q3, 观察表 9 中后四行可以看出, Qemu 中 AST 的精度比 CPG 高 0.81%, 作者认为出现这样的情况可能由于 Qemu 项目中, 多数存在的漏洞是不存在相互控制和依赖的, 使用 AST 可以覆盖大部分类型漏洞。而 CPG 由于存在额外的信息, 导致信息冗余, 造成检测精度的降低。除上述情况外, 其余情况都是 CPG 的精度最高。在 Qemu+FFmpeg 数据集中, 使用 CPG 和 PDG 之间的精度差异最为显著, 差距为 5.37%。同时可以看出, 与其他的图相比, 使用 CPG 的 F1 得分都略低。由此可见, CVD-GCL 从单个图中提取漏洞特征的能力与 CPG 相比, 还有所不足。

表 9 不同方法的漏洞检测结果

Table 9 The vulnerability detection results of different methods

方法	输入	Qemu		FFmpeg		Qemu+FFmpeg		Max Diff		Avg Diff	
		ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1
CNN+attention	Y	52.46	59.45	50.13	67.56	51.21	61.29	18.43	12.27	16.55	8.55
Path+attention	Y	54.30	67.12	51.97	70.12	53.48	68.48	16.49	9.30	14.68	4.82
Funded	F, Y	72.53	70.17	67.42	70.58	71.33	69.54	5.19	8.95	2.54	0.79
VulCNN	V, Y	74.79	72.51	69.23	69.77	72.18	70.86	2.26	2.34	0.85	0.81
Devign(CPG)	X, A, Y, E	74.45	74.45	67.14	70.46	70.13	72.45	4.85	9.64	2.75	5.63
CVD-GCL(AST)	X, A, E	70.41	72.66	61.69	60.35	65.44	67.70	4.20	0.94	2.51	0.54
CVD-GCL(CFG)	X, A, E	67.83	72.11	60.49	59.28	65.13	70.85	4.50	2.93	3.33	1.62
CVD-GCL(PDG)	X, A, E	68.03	72.96	63.41	60.54	64.27	71.94	5.37	4.02	2.58	1.84
CVD-GCL(CPG)	X, A, E	69.60	71.72	64.21	60.82	69.64	67.92	-	-	-	-

(注: 方法中 AST, CFG, PDG 和 CPG 表示本工作提出的相对应的图数据集, X 为图中节点特征矩阵, A 为图的邻接矩阵, E 为是否考虑边的类型, Y 表示数据集的标签, F 表示 Funded 自定义数据集, V 表示 VulCNN 自定义数据集, Max Diff 和 Avg Diff 分别代表当前方法与 CVD-GCL(CPG) 相比, 最大差距和平均差距)

(3) 图数据增强方法的选择。为提高漏洞检测的准确性和鲁棒性, 使用不同的图增强方式生成不同视图是一种常见的策略。本工作重点评估单个增强方式对漏洞检测模型的影响。具体而言, 本工作针对原图, 使用两个不同的数据增强方式进行数据增强, 获得原图的两个不同视图, 使用精度作为度量标准, 来评估不同增强方式对模型的影响。本实验使用表 4 的 all_cpg 图数据集作为输入, 图的增强方式如表 7 所列举, 实验结果如表 10 所示。

表 10 CVD-GCL 在不同增强方式下的精度
Table 10 The ACC of CVD-GCL under different augmentation methods

精度	EA	FM	FD	EAR	ND	NS	ES
EA	67.06	67.67	66.71	67.36	67.46	67.58	67.99
FM	-	67.10	67.31	68.17	66.89	67.19	67.08
FD	-	-	68.09	66.98	67.19	68.06	67.03
EAR	-	-	-	66.78	66.73	67.14	67.16
ND	-	-	-	-	67.10	67.54	66.78
NS	-	-	-	-	-	67.26	66.84
ES	-	-	-	-	-	-	67.14

(注: 表中所涉及比例参数的值都为 0.1, 横坐标为 q_1 , 纵坐标为 q_2)

针对 Q4, 观察表 10, 可以探索不同图增强方式对漏洞检测任务的影响。从表中得知, 将 FM 与 EAR 相结合, 显著提高了模型的精度, 达到 68.17%, 超过其他实验结果。同时, 大多数增强方式得到的结果差距不大, 精度仅存在轻微变化。尽管多数增强方式的精度在 67%, 但如 FM 和 FD 的组合, 可以将精度提升到 68%。这表明在特定的情况下, 一些增强方法可能具有优势。另外, 使用 EA 和 FD 的组合方式使得

模型的精度降低到 66.71%, 这表明某些特定的增强方法可能会与漏洞检测任务相冲突, 从而造成漏洞检测任务的精度下降。

5 总结和未来工作

本工作提出一种新的漏洞检测框架 CVD-GCL, 使用图对比学习进行代码中漏洞特征的学习。CVD-GCL 将源代码的语义信息、语法信息和结构信息整合到 CPG 中, 然后使用图对比学习模型获取源代码中深层次的漏洞信息。以实际开源项目为数据集进行实验, CVD-GCL 在不依赖于标签的情况下, 精度可以与现有的有监督漏洞检测方案相当。本工作验证 CVD-GCL 在漏洞检测中表现出良好的性能, 并且说明图对比学习在漏洞检测领域的可行性。

在未来的研究中, 以下几个方面值得关注。评估复合增强方案, 寻找更优的增强方法; 使用少量有标签数据集训练图对比学习后的图神经网络模型, 判断训练后的模型效果; 改进图对比学习的内在框架, 寻找在漏洞检测领域更优的模型。

参考文献

[1] Zimmermann T, Nagappan N, Williams L. Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista[C]. 2010 Third International Conference on Software Testing, Verification and Validation, 2010: 421-428.

[2] Morrison P, Herzig K, Murphy B, et al. Challenges with Applying Vulnerability Prediction Models[C]. The 2015 Symposium and Bootcamp on the Science of Security, 2015: 1-9.

[3] 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.

- [4] Shin Y, Williams L. Can Traditional Fault Prediction Models Be Used for Vulnerability Prediction?[J]. *Empirical Software Engineering*, 2013, 18(1): 25-59.
- [5] Perl H, Dechand S, Smith M, et al. VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits[C]. *The 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015: 426-437.
- [6] Meneely A, Srinivasan H, Musa A, et al. When a Patch Goes Bad: Exploring the Properties of Vulnerability-Contributing Commits[C]. *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013: 65-74.
- [7] Du X N, Chen B H, Li Y K, et al. LEOPARD: Identifying Vulnerable Code for Vulnerability Assessment through Program Metrics[C]. *2019 IEEE/ACM 41st International Conference on Software Engineering*, 2019: 60-71.
- [8] Zagane M, Abdi M K, Alenezi M. Deep Learning for Software Vulnerabilities Detection Using Code Metrics[J]. *IEEE Access*, 2020, 8: 74562-74570.
- [9] Brown P F, DeSouza P V, Mercer R L, et al. Class-Based n -Gram Models of Natural Language[J]. *Computational Linguistics*, 1992, 18(4): 467-479.
- [10] Church K W. Word2Vec[J]. *Natural Language Engineering*, 2017, 23(1): 155-162.
- [11] 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.
- [12] Yan H, Luo S L, Pan L M, et al. HAN-BSVD: A Hierarchical Attention Network for Binary Software Vulnerability Detection[J]. *Computers & Security*, 2021, 108: 102286.
- [13] Ni Z, Li B, Sun X B, et al. Analyzing Bug Fix for Automatic Bug Cause Classification[J]. *Journal of Systems and Software*, 2020, 163: 110538.
- [14] Li Z, Zou D, Xu S, et al. Vuldeepecker: A deep learning-based system for vulnerability detection[J]. *arXiv preprint arXiv:1801.01681*, 2018.
- [15] Li Z, Zou D Q, Xu S H, 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.
- [16] Zou D Q, Wang S J, Xu S H, et al. μ VulDeePecker: A Deep Learning-Based System for Multiclass Vulnerability Detection[J]. *IEEE Transactions on Dependable and Secure Computing*, 2021, 18(5): 2224-2236.
- [17] Li Z, Zou D Q, Xu S H, et al. VulDeeLocator: A Deep Learning-Based Fine-Grained Vulnerability Detector[J]. *IEEE Transactions on Dependable and Secure Computing*, 2022, 19(4): 2821-2837.
- [18] Li Y, Wang S H, Nguyen T N, et al. Improving Bug Detection via Context-Based Code Representation Learning and Attention-Based Neural Networks[J]. *Proceedings of the ACM on Programming Languages*, 2019, 3(OOPSLA): 1-30.
- [19] Lin G J, Zhang J, Luo W, et al. POSTER: Vulnerability Discovery with Function Representation Learning from Unlabeled Projects[C]. *The 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017: 2539-2541.
- [20] Zhou Y, Liu S, Siow J, et al. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks[J]. *Advances in neural information processing systems*, 2019, 32.
- [21] Zheng W N, Jiang Y, Su X H. VulSPG: Vulnerability Detection Based on Slice Property Graph Representation Learning[C]. *2021 IEEE 32nd International Symposium on Software Reliability Engineering*, 2021: 457-467.
- [22] Zhang H B, Bi Y F, Guo H Z, et al. ISVSF: Intelligent Vulnerability Detection Against Java via Sentence-Level Pattern Exploring[J]. *IEEE Systems Journal*, 2022, 16(1): 1032-1043.
- [23] Yamaguchi F, Golde N, Arp D, et al. Modeling and Discovering Vulnerabilities with Code Property Graphs[C]. *2014 IEEE Symposium on Security and Privacy*, 2014: 590-604.
- [24] Yamaguchi F, Maier A, Gascon H, et al. Automatic Inference of Search Patterns for Taint-Style Vulnerabilities[C]. *2015 IEEE Symposium on Security and Privacy*, 2015: 797-812.
- [25] Wang H T, Ye G X, Tang Z Y, et al. Combining Graph-Based Learning with Automated Data Collection for Code Vulnerability Detection[J]. *IEEE Transactions on Information Forensics and Security*, 2021, 16: 1943-1958.
- [26] Duan Ya-Nan. Research on software vulnerability detection method based on code property graph and graph convolutional neural network[Master dissertation]. Harbin Institute of Technology. Harbin, China, 2020.
(段亚男, 基于代码属性图和图卷积神经网络的软件漏洞检测方法 研究[硕士学位论文]. 哈尔滨工业大学, 2020)
- [27] Cheng X, Wang H Y, Hua J Y, et al. DeepWukong: Statically Detecting Software Vulnerabilities Using Deep Graph Neural Network[J]. *ACM Transactions on Software Engineering and Methodology*, 2021, 30(3): 1-33.
- [28] Pennington J, Socher R, Manning C. Glove: Global Vectors for Word Representation[C]. *The 2014 Conference on Empirical Methods in Natural Language Processing*, 2014: 1532-1543.
- [29] Ding K Z, Xu Z, Tong H H, et al. Data Augmentation for Deep Graph Learning: A Survey[J]. *ACM SIGKDD Explorations Newsletter*, 2022, 24(2): 61-77.
- [30] Xu K, Hu W, Leskovec J, et al. How powerful are graph neural networks [J]. *arXiv preprint arXiv:1810.00826*, 2018.
- [31] Tolstikhin I O, Houlsby N, Kolesnikov A, et al. MLP-Mixer: An All-MLP Architecture for Vision[C]. *Neural Information Processing Systems*, 2021.
- [32] He K M, Fan H Q, Wu Y X, et al. Momentum Contrast for Unsupervised Visual Representation Learning[C]. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020: 9726-9735.
- [33] Wu Y M, Zou D Q, Dou S H, et al. VulCNN: An Image-Inspired Scalable Vulnerability Detection System[C]. *2022 IEEE/ACM 44th International Conference on Software Engineering*, 2022: 2365-2376.
- [34] Hu J, Shen L, Sun G. Squeeze-and-Excitation Networks[C]. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018: 7132-7141.
- [35] Chen B, Barzilay R, Jaakkola T. Path-augmented graph transformer network[J]. *arXiv preprint arXiv:1905.12712*, 2019.



夏鹏程 于 2021 年在华北科技学院计算机科学与技术专业获得学士学位。现在中国科学技术大学计算机科学与技术专业攻读硕士学位, CCF 会员。主要研究方向为: 软件漏洞检测、软件安全、图神经网络等。Email: pcxia@mail.ustc.edu.cn



曾凡平 于 2009 年在中国科学技术大学信息安全专业获得博士学位。现任中国科学技术大学副教授。主要研究方向为: 网络信息安全、软件分析与测试、物联网的云边端资源协同优化。Email: billzeng@ustc.edu.cn



刘通 于 2020 年在池州学院机械设计制造及其自动化专业获得学士学位。现在中国科学技术大学计算机技术专业攻读硕士学位。主要研究方向为: 软件安全、云计算、边缘计算、协同调度等。Email: tongliu997@mail.ustc.edu.cn