

基于抽象语法树特征迁移的软件漏洞 检测方法 (AST-FMVD)

李子俊¹, 李涛^{2*}, 陈浩东³, 余琴¹, 乔梦晴¹,
李琳², 王颀³, 万振华³, 宋荆汉³

- (1. 武汉科技大学 计算机科学与技术学院, 湖北 武汉 430065;
2. 智能信息处理与实时工业系统湖北省重点实验室, 湖北 武汉 430065;
3. 深圳开源互联网安全技术有限公司, 广东 深圳 518000)

摘要:深度学习在漏洞检测的应用取得了显著的进展。现有漏洞检测算法需要大量的标记数据,通过有监督的方法构建检测模型,在多语言环境中,由于语言的多样性和标记训练样本的缺乏,检测模型可能存在泛化性问题,特别是在小样本领域中可能表现不佳。为了解决这一困境,迁移学习可以作为一种解决方案,迁移学习的核心思想是以“举一反三”为核心的算法框架,将某个领域的知识迁移到另一个领域的学习中,从而打破样本数据的制约。该文提出了一种基于特征迁移的漏洞检测方法。通过语义相似性对代码的语法树节点信息进行聚类,可以快速并准确地构建好不同语言之间的节点映射关系,同时在语法树的映射过程中引入上下文感知技术帮助解决歧义或模糊的语法结构,提高解析性能。该方法实现检测样本从未知领域变换到已知领域,利用在原有领域构建的深度学习模型,可以将新领域任务迁移到已知领域,最终解决跨域的知识迁移的应用,并将该方法取名为 AST-FMVD。最后通过 Java 的漏洞检测模型对含有特定漏洞文件的进行检测,实现模型在 Python 领域中的迁移应用,证明了 AST-FMVD 的可行性,并通过实验证明 AST-FMVD 可以实现源域中的训练模型在目标领域仍可以保证原模型良好的检测水平。

关键词:深度学习;迁移学习;零样本;漏洞检测;抽象语法树

中图分类号:TP389;TP309

文献标识码:A

文章编号:1673-629X(2024)06-0081-08

doi:10.20165/j.cnki.ISSN1673-629X.2024.0065

Software Vulnerability Detection Method Based on Abstract Syntax Tree Feature Migration (AST-FMVD)

LI Zi-jun¹, LI Tao^{2*}, CHEN Hao-dong³, YU Qin¹, QIAO Meng-qing¹,
LI Lin², WANG Jie³, WAN Zhen-hua³, SONG Jing-han³

- (1. School of Computer Science and Technology, Wuhan University of Science and Technology,
Wuhan 430065, China;
2. Hubei Provincial Key Laboratory of Information Processing and Real Time Industrial Systems,
Wuhan 430065, China;
3. Shenzhen Kaiyuan Internet Security Technology Co., Ltd., Shenzhen 518000, China)

Abstract: Deep learning has made significant progress in vulnerability detection. Existing vulnerability detection algorithms require a large amount of labeled data and build detection models through supervised methods. In a multi-language environment, due to the diversity of languages and the lack of labeled training samples, the detection model may have generalization problems, especially in the field of small samples, where performance may be poor. To solve this dilemma, transfer learning can serve as a solution. The core idea of transfer learning is the "learning by analogy" algorithm framework, transferring knowledge from one domain to another, thereby breaking the constraints of sample data. We propose a feature-based transfer vulnerability detection method. By clustering the syntax tree node information of the code through semantic similarity, the node mapping relationship between different languages can be quickly and accurately constructed. At the same time, context-aware technology is introduced in the syntax tree mapping process to help solve ambiguous or vague

收稿日期:2023-07-08

修回日期:2023-11-09

基金项目:武汉市重点研发计划(2022012202015070);武汉科技大学研究生教改研究项目(Yjg202111)

作者简介:李子俊(2002-),男,研究方向为信息安全;通信作者:李涛(1979-),男,博士,教授,研究方向为推荐系统人工智能、信息安全。

grammatical structures, improving parsing performance. The proposed method enables the detection samples to transform from unknown domains to known ones, and utilizing the deep learning model built in the original domain, the new domain task can be transferred to the known domain, ultimately solving the application of cross-domain knowledge transfer. It is named AST-FMVD. Finally, we use the Java vulnerability detection model to detect files containing specific vulnerabilities, realizing the model's transfer application in the Python domain, proving the feasibility of AST-FMVD, and experimentally demonstrating that AST-FMVD can ensure the original model's good detection level in the target domain.

Key words: deep learning; transfer learning; zero-shot; vulnerability detection; abstract syntax tree

0 引言

传统的漏洞挖掘方法在小规模软件方面已取得了成果,但在应对大型复杂软件系统和变化多样的新型漏洞时,通常无法满足需求^[1]。软件的高复杂性和安全漏洞的形态多样化给软件安全漏洞研究带来了严峻的挑战。目前,大量的研究工作尝试将深度学习^[2]应用于漏洞挖掘领域,以实现自动化和智能化漏洞挖掘。

通常构建高性能的学习模型,往往依赖于大样本的训练集来提高模型的准确性。Zhen 等人^[3]提出的漏洞检测模型依靠多种漏洞集构成的高容量的样本进而提高模型的准确性。但是不同领域中漏洞样本数量往往是受限的,通过传统的机器学习方法无法保证小样本模型的准确性。在图像处理领域中,深度神经网络展现出了良好的可迁移性^[4-6],通过对模型进行微调处理可以将预有模型在不同领域中迁移应用。随着对迁移学习的进一步认识和研究^[7-8],目前主流的迁移学习方式主要基于样本、特征、模型和关系这四个层面开展迁移工作,并在漏洞检测领域逐步得到应用。Lutz^[9]在旧漏洞的知识体系下,通过对新的漏洞学习过程中模型参数的微调,实现在小样本漏洞领域下基于模型层面的迁移学习。基于模型层面的微调处理,通过目标领域的小样本空间对模型参数进行优化处理,一定程度上解决了数据稀疏性的问题,但是无法摆脱学习模型对样本数据的依赖性。Zaharia 等人^[10]提出一种源代码的跨语言表示,用于将源代码转换成一种中间表示形式,能够保留源代码在不同编程语言中的安全漏洞模式,减少对编程语言词法和语义结构的依赖性。参考迁移学习在安全领域中的应用,如何构建有效的特征映射方法成为了研究的重要课题。

在代码漏洞信息分析过程中,通常对应的代码模块会聚焦于具体的漏洞特征,并利用漏洞的表征信息将待检测的整个代码模块抽象成以漏洞特征为中心的抽象表征形式。Feng 等人^[11]发现基于抽象语法树(AST)的代码表征方法适合进行漏洞分析,并对语法树按照功能级别提取语法特征能够更加准确地捕获漏洞信息,并减少数据冗余。聚焦于漏洞分析,不同语言之间的抽象语法树的变换不同于源码层面的代码语言翻译工作,AST 旨在从源代码中提取关键的代码结构和语义信息,以便发现潜在的漏洞。虽然转换后的语

法树存在部分信息缺失,但是映射关系会保留关键的漏洞特征,并不影响模型进行漏洞检测。除此之外,语法特征通过 Word2Vec^[12-15]捕获语义关系可以忽略一些不重要的语义差异。

针对漏洞检测模型的迁移问题,该文提出了一种基于特征层面的迁移方法,名为 AST-FMVD。在语法树进行遍历抽取漏洞特征的过程中,首先对目标语言进行语法树的解析工作,获得代码的 AST 后,开始进行迁移过程。在构建初步的节点映射关系时,从收集好的代码库解析出全部代码的关键字信息,称为“token”并计算 token 之间的语义相似性,将相似的 token 聚类在一起。根据聚类结果,将具有相似特征的节点划分为同一类别,然后根据特征层面的语义相似性建立对应语法树节点之间的映射关系。即对于不同语言中具有相似特征的节点,将它们映射到同一个类别,从而建立起节点在不同语言之间的映射关系。除此之外,该方法专注于不同语言的 AST 之间结构层面的映射关系并引入上下文感知技术,解决可能出现的歧义或模糊的语法结构。该方法为零样本漏洞检测提供了一种可行性的解决方案,在不需要标记样本的情况下,使用原始模型在原领域中的检测效果可以有效地迁移到目标领域中。实验针对 Python 的语法树映射到 Java 中,并利用原始 Java 领域中的检测模型实现目标领域开展检测工作,最终在目标领域中展现出模型在原始领域中优秀的检测效果。

1 相关工作

机器学习为构建漏洞检测模型提供了更多的方法,从最初基于专家规则的检测技术,逐步发展到基于机器学习和深度学习的预测模型,近年来通过引入迁移学习技术可以缓解传统学习方法对大量标注样本的需要。

1.1 漏洞检测模型

基于专家规则的漏洞检测模型的核心是收集专家在软件安全和漏洞分析方面的经验,将其转化为检测规则,然后制定形成漏洞检测路径,从而构建漏洞检测模型。典型的专家系统如 Ilgun 是在 1995 年提出的侵入检测方法^[16],该方法使用状态转换来检测入侵。使用专家定义的规则模拟系统的正常行为,并将这些规

则转化为状态机。任何偏离正常状态转换的行为都被标记为潜在的入侵。专家系统的构建需要手工编写大量的专家规则,维护成本高,可扩展性有限。

随着新技术的发展和漏洞数据库的形成和应用,基于统计分析和案例分析的漏洞检测模型使用数据挖掘技术从代码仓库中自动总结检测规则。Lane 等人^[17]使用序列匹配和学习技术来检测异常行为,使用数据挖掘和统计分析技术自动总结漏洞检测规则并从软件代码中收集统计知识。这种方法适用于有大量样本数据的情况,但不适用于具有稀疏样本的特定漏洞的情况。

基于机器学习的漏洞检测模型将漏洞检测问题形式化为一个分类问题,从源代码、二进制代码或运行时数据中提取并选择特征。Kruegel 等人^[18]分析二进制代码以识别并预防潜在的缓冲区溢出攻击。他们提出了一种基于机器学习的缓冲区溢出检测方法,通过为机器学习分类模型训练精心构建的样本特征,使用支持向量机和随机森林实现漏洞检测。这减少了不相关的特征并增强了模型性能。

神经网络技术已逐渐应用于漏洞检测模型的构建。基于深度学习的漏洞检测模型利用深度神经网络,如卷积神经网络(CNNs)或递归神经网络(RNNs),来分析代码结构和语义以进行漏洞检测。Mou 等人^[19]使用 CNNs 和 RNNs 的组合来捕获源代码的结构和模式,为进一步的代码分析和漏洞检测奠定了基础。Russell 等人^[20]探讨了如何使用深度表示学习,特别是长短期记忆(LSTM)网络,来检测源代码中的漏洞。此外,与传统的基于规则和基于签名的漏洞检测技术相比,这种方法对新的和未知的漏洞模式更具适应性。

尽管机器学习和深度学习技术在漏洞检测模型中取得了进展,但学习模型的构建通常依赖于大规模的标记训练样本。因此,在漏洞挖掘领域,学习模型的构建仍然面临样本空间不平衡和样本数量有限的挑战。

1.2 迁移学习技术

迁移学习是机器学习的一个子领域,旨在将从源任务中学到的知识转移到目标任务。在漏洞检测领域,迁移学习起到了关键的作用,特别是在样本不平衡和数量有限的情况下。考虑到漏洞类型的多样性和不断演变的特性,传统的漏洞检测方法可能在新的或罕见的漏洞类型上表现不佳。迁移学习的应用可以缓解这个问题,主要在四个层面进行:样本、特征、模型和关系。

Pan^[21]提出了一种基于样本的迁移学习方法,通过加权重采样来平衡源域和目标域之间的分布差异。它能够有效地转移不同类型的漏洞样本,并为迁移学

习提供了一个基础的概览。

特征层面的迁移学习关注于提取和选择对目标任务有用的特征。Long 等人^[22]提出了一种名为 Joint Adaptation Networks (JAN) 的深度迁移学习方法,该方法通过最大化源域和目标域之间的距离来增强特征对齐。JAN 还包括一个多核选择机制,允许适应各种域的转变。

迁移工作也可以从模型或关系的角度进行^[23]。基于模型的迁移学习关心如何将源任务中学到的模型或参数转移到目标任务。常见的技术包括对预训练模型的参数进行微调,其中从源任务学到的参数作为目标任务的初始参数,然后进一步优化。通过实际应用,迁移学习已被证明可以显著减少手动注释的需要,并提高新漏洞类型的预测准确性。尽管迁移学习在漏洞检测中展现出了潜力,但仍然存在挑战,如匹配源任务和目标任务之间的差异,以及负迁移的风险。

2 基于抽象语法树特征映射的模型迁移方法

AST-FMVD 框架如图 1 所示。

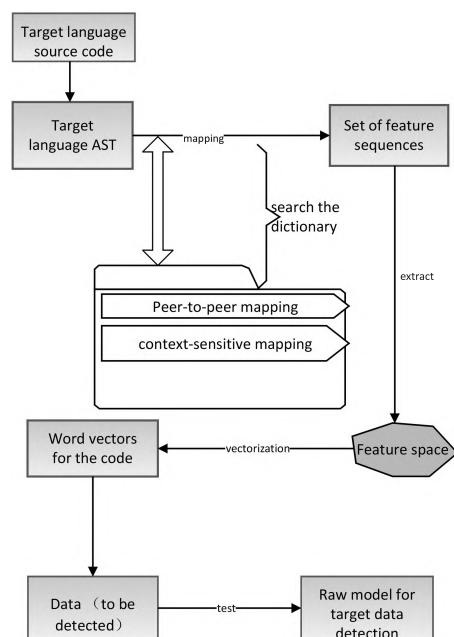


图 1 AST-FMVD 框架

首先,目标语言被转化为语法树形式的表示。随后,按照预定义的映射规则字典进行语法树节点的映射转换。映射字典记录了在跨语言语法树特征转换过程中不同类型节点的替换规则。这些规则基于代码结构和代码语义信息的双向映射,解决了跨语言传输漏洞特性的挑战,从而实现了跨域知识传输的应用。AST-FMVD 技术框架使得在不同的语言之间进行特征转移成为可能,基于特征层次的迁移过程有助于保留源代码的最小特征,减少了对编程语言词汇和语义

结构的依赖。

映射规则字典的构建:

映射规则字典的构建是实现语法转换的关键步骤。这个过程主要涉及两个步骤:

- 使用上下文感知技术,针对上下文的结构类型节点,根据上下文窗口大小,从源代码中提取相应的周围代码片段作为上下文信息。根据映射规则,将提取的上下文信息和特定节点信息映射到对应的目标节点。

- 建立映射点对点的映射规则,对于不需要进行上下文感知映射的节点类型(如声明类型、运算符类型和特殊字符类型),需要发现不同语法树节点语义之间的相似度。通过 k-means 算法按照相似度大小对不同语言的节点进行聚类分析来构建点对点的映射规则。

2.1 上下文感知技术的构建方法

语法树节点中存在类似于变量声明,运算操作,标点符号或者注释等节点信息具有十分明确的语义,不需要额外的上下文信息进行特征映射,只需调用预先构造好的映射字典即可。但是针对上下文的结构类型节点,这些节点表示代码的结构,例如 if 条件语句、for 循环语句、while 循环语句等。在进行特征迁移时,需要考虑这些节点周围的上下文信息,以确保代码结构

的正确映射。

基于上下文感知技术的语法映射方法主要分为两个部分,分别是确定上下文窗口大小和基于上下文规则映射的构建。该类映射方法针对的节点类型是对代码控制流结构、函数、类等进行抽象的,是代码的基本结构和语义表示。

如图 2 所示,对于需要上下文感知映射的节点,根据上下文窗口大小,从源代码中提取相应的周围代码片段作为上下文信息。根据映射规则,将提取的上下文信息和特定节点信息映射到对应的 Java 节点。根据上下文信息和特定节点的语义和结构,找到 Java 语法树中对应的节点类型和位置。最终实现上下文的结构类型节点的映射工作。例如语法树的 if 条件语句节点,根据定义的上下文窗口大小,确定条件判断表达式的代码行数和条件分支内的代码行数(if 分支和 else 分支)。提取条件判断表达式和条件分支内的代码行数,形成上下文信息。在完成语法树的结构映射之后,内部的节点信息属于单点映射,只是在不同语言中的节点名称不同,仅需对照映射字典中的节点名称替换规则,对特征序列中的节点名称替换即可。将融合得到的上下文信息与当前节点的特征信息进行合并,得到最终的特征表示。

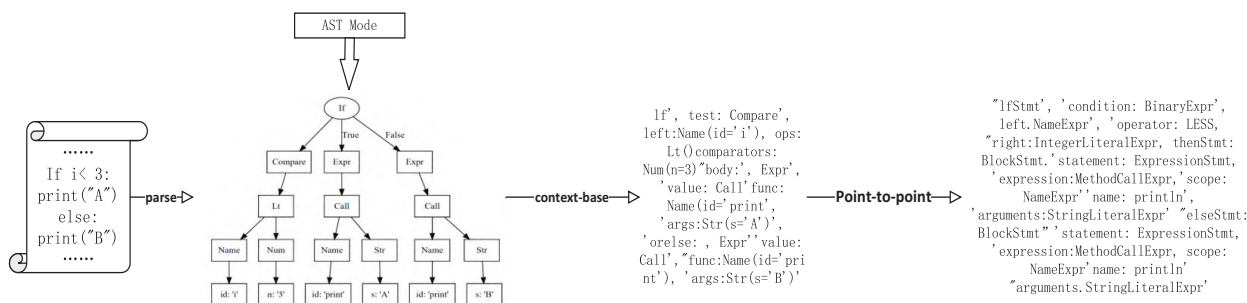


图 2 基于上下文结构的映射

表 1 上下文窗口大小确定方法

语法树节点类型	上下文窗口大小确定方法
if 判断节点	if 分支和 else 分支,并将其作为上下文窗口
While 循环节点	循环的循环条件表达式和循环体,并将其作为上下文窗口
For 循环节点	循环条件表达式和循环体,并将其作为上下文窗口
Switch 语句节点	条件表达式和各个 case 分支,并将其作为上下文窗口
Function 节点	函数的函数名、参数列表和函数体,并将其作为上下文窗口
Class 节点	类的类名、类成员变量和类成员函数,并将其作为上下文窗口
Try 语句节点	Try 语句的 try, catch, finally 块,将其作为上下文窗口

如表 1 所示,基于上下文的结构类型的语法树节点仅含有 7 种,分别是 if 语句节点;While 循环节点;For 循环节点;Switch 语句节点;Function 节点;Class 节点;Try 语句节点。针对这 7 种语法类型,通过自定

义函数来识别和确定代码块并根据相应的代码块扩展上下文窗口。当确定好上下文窗口之后,对于每个需要进行上下文感知映射的节点,将其特定信息和提取得到的上下文信息进行合并,形成融合了代码结构和

语义的完整特征表示。由于不同代码语言在这些结构上的表示方式是相似的,那么该规则的制定具有语言通用性,而动态生成上下文窗口的方法也具有语言特异性,可以很好地过滤代码的噪声信息。

2.2 点对点映射的构建方法

点对点映射方法的构造是一种基于语义相似性的节点映射算法。构建过程如图3所示。选择将收集到

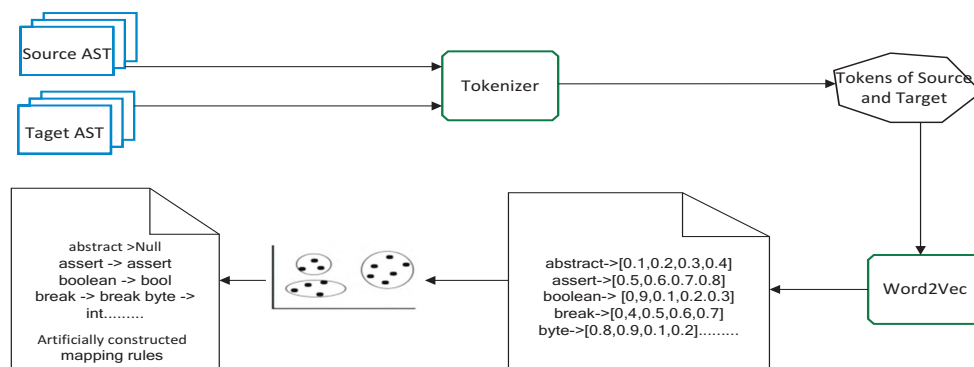


图3 基于语义相似性的映射构建方法

为了构建不同语言节点之间基于语义信息的映射关系,首先需要收集不同编程语言中的所有 token 信息。token 对应语法树节点名称的相似性,最终构造映射字典中的映射规则。token 是代码的基本组成部分,

的目标文件和原始文件的语法树进行序列化形成包含不同语言的 token 信息,对 token 进行 Word2Vec 向量化处理,然后利用余弦相似度的方法来度量 token 之间的相似度。最后通过 k-means 算法按照相似度大小对 token 完成聚类,将相似的节点分配到同一个簇中,从而建立了节点之间的语义相似性关系。

编译器在对代码进行词法分析时会将代码拆分成一个个 token,并进一步进行语法分析和执行。按照词法单元的属性,将 token 分成了 10 类。

表2 token 类型

Identifiers	Keywords
Operators and Separators Literals	Literals
Comments	Strings
Numbers	Boolean Constants
Whitespace and Line Breaks	Special Characters

如表2所示,其中标识符具有语言特定性不适合使用通用性的映射方法,特殊字符(如冒号、逗号、点号等),数字,布尔常量,字符串往往是具有固定含义的符号,不涉及到具体的代码结构和语义转换,且不具有语言的差异性。并且语法树不包含注释信息和代码的格式信息。那么这些类型的 token 信息在语法树映射过程中保留原语言的特性,不做映射处理。此外一些常见的操作符,例如加法运算符(+)、比较运算符(例如==,<,>等)等,在大多数编程语言中都是类似的,而针对一些操作符在不同编程语言中具有不同的表示形式或者功能。例如,C语言中的逻辑与运算符是“&&”,而在Python中是“and”的情况,只需要在预先构建好的字典进行简单的规则添加即可。综上所述,需要通过语义相似性发现映射规则的 token 类型分别为字面量类(如 int, bool, String 等)和关键字(如 if, else, for 等)类型。AST-FMVD 采用 k-means 的相似性算法按照相似度大小对 token 进行聚类,将相似的节点分配到同一个簇中,从而建立了节点之间的语

义相似性关系。

X 表示所有 token 的集合,并用 $S(X_i, X_j)$ 代表 token X_i 和 X_j 之间的余弦相似度。具体步骤如下:

(1) 计算相似度矩阵:对于所有的 token X_i 和 X_j , 计算它们之间的余弦相似度 $S(X_i, X_j)$ 。使用 Word2Vec 向量化表示和余弦相似度公式来完成。

(2) 初始化聚类分配:为每个 token 初始化一个聚类 $C(X_i)$ 分配,将每个 X_i 分配给它自己的聚类。

(3) 迭代更新聚类分配:迭代更新聚类分配,直到收敛。对于每个 X_i ,基于特定条件执行以下操作:

- 如果 X_i 是一个异常值,将其分配给与 X_i 具有最小相似度的另一个 token 的聚类,即 $C(X_i) = \underset{X_j \in X, X_j \neq X_i}{\operatorname{argmin}} S(X_i, X_j)$ 。
- 如果包含 X_i 的聚类已经包含来自不同语言的两个以上的 token,将 X_i 分配给与具有最小相似度的另一个 token 的聚类,即 $C(X_i) = \underset{X_j \in X, X_j \neq X_i}{\operatorname{argmin}} S(X_i, X_j)$ 。
- 如果包含 X_i 的聚类包括来自不同语言的两个

token, 并且不是一个异常值, 将分配给与其具有最大相似度的另一个 token 的聚类, 即 $C(X_i) = \underset{X_j \in X, X_j \neq X_i}{\operatorname{argmin}} S(X_i, X_j)$ 。

(4) 检查是否收敛: 每次迭代后, 检查聚类分配是否有变化。如果分配保持不变, 则算法收敛。

通过将不同语言的两类 token 的语法树节点进行相似聚类分析, 可以得到大多数节点的直接映射关系, 但是聚类结构中存在有部分的离群点和同一语言下的节点聚类簇。针对这些类型节点, 可以单独抽取分析其对应语言中的语义规则, 增补节点映射规则, 如 Python 中的 Module 替换成 Java 的 CompilationUnit, Raise 对应 Java 的 ThrowStatement。

3 实验

当前的实验使用了来自 OWASP Benchmark^[23] 的 Java 漏洞特征。OWASP Benchmark 的主要目标是评估安全测试工具的能力, 包括准确性、覆盖率、扫描速度等。它量化了这些工具的扫描能力, 从而更好地比较它们的优点和缺点。在此实验中, 来自 OWASP Benchmark 的多类数据集被送入一个 Bi-LSTM 模型进行检测。

在从包含 CWE89 漏洞的 Python 源代码中收集漏洞的过程中, 已经从之前的 Java 漏洞检测模型中获得了大量带有 CWE89 漏洞的 Java 源代码。因此, 该文采用了 Java 到 Python 转换器^[24-25], 这是一个编程语言翻译工具, 将 Java 源代码中的相应漏洞类型转换为 Python 源代码。这种方法在短时间内有效地收集了包含 SQL 注入漏洞 (CWE89) 的 Python 文件。

表 3 列出了测试集中存在的漏洞的类型和数量。在评估不同映射深度对漏洞检测影响的实验中, 正样

表 3 Benchmark 数据集中不同类型漏洞的数量

CWE	Quantity	CWE	Quantity
0	1 213	89	249
79	246	330	218
22	133	327	130
328	129	78	126
501	83	614	36
90	27	643	15

本包括 249 个没有漏洞的 Python 文件, 而负样本则包括 249 个包含 SQL 注入漏洞的 Python 文件。这些样本作为实验的测试集。

对比与证明实验:

执行语法树特征向量检测需要将每个样本的相应特征转化为固定维度的数值向量。在词嵌入的过程中, 会利用之前从 Java 训练中构建的词汇表。这有助于将特征从高维度空间转化为较低维度的连续向量空间。最终, 这些转换后的特征向量将被输入到预训练的 Java 检测模型中。

本次实验重点关注了 Java 编程语言中存在的 11 种原始漏洞类型 (22, 327, 328, 330, 501, 614, 643, 78, 79, 89, 90), 它们被标记为 cl.1-11。采用了 AST-FMVD 模型迁移方法, 其中每个漏洞类别被视为正样本和负样本。此外, 新的语言类型 (包含 CWE89 漏洞的 Python 文件) 被标记为 cl.10'。为了解决样本分布的不平衡问题, 数据集经过了过采样技术。这个过程确保了数据集中每个漏洞类别都有公平的代表性。随后, 为每个标签计算了性能指标, 得到的平均值被用作评估迁移方法效果的最终指标。实验结果显示在表 4 中。

表 4 AST-FMVD 在不同语言空间之间的迁移性能

Metrics		Initial vulnerability types				
CWE Type		cl.1	cl.2	cl.3	cl.4	cl.5
Recall		0.91	0.97	0.96	0.99	0.97
F1 score		0.91	0.90	0.94	0.95	0.91
FPR		0.91	0.93	0.95	0.93	0.95
FNR		0.09	0.10	0.06	0.01	0.03
Precision		0.04	0.04	0.03	0.14	0.05
Metrics		Initial vulnerability types				
CWE Type		cl.7	cl.8	cl.9	cl.10	cl.11
Recall		0.98	0.85	0.92	0.90	0.83
F1 score		0.86	0.82	0.91	0.85	0.67
FPR		0.89	0.84	0.91	0.89	0.74
FNR		0.10	0.00	0.04	0.01	0.01
Precision		0.20	0.16	0.14	0.33	0.11
		New Lang				
CWE Type		cl.10'				
Recall		0.98	0.85	0.92	0.90	0.83
F1 score		0.86	0.82	0.91	0.85	0.67
FPR		0.89	0.84	0.91	0.89	0.74
FNR		0.10	0.00	0.04	0.01	0.01
Precision		0.20	0.16	0.14	0.33	0.11

零样本迁移实验在特征级别的比较:

Zaharia 曾提出了 CLaSCoRe^[10] (一种用于实现基于深度学习的软件漏洞检测的跨语言表示方法),用于实施深度学习软件漏洞检测的迁移应用。它旨在将源代码转换成中间表示,跨不同编程语言保留漏洞信息,从而减少对编程语言的词法和语义结构的依赖。使用 SVM 作为漏洞检测模型,它在 C, C++ 和 Java 语言的数据集上进行了训练。最终,由 CLaSCoRe 构建的特征迁移用于检测 C# 中的安全漏洞。在此过程中,没有在训练中使用 C# 代码片段。当将其他语言的漏洞检测模型通过 CLaSCoRe 迁移到 C# 时,平均准确率为 0.89, SVM 的平均召回率为 0.80。对于原生 C# 语言模型检测,平均准确率为 0.97。从模型的召回率来看,使用基于 CLaSCoRe 的表示并在 C# 数据集上进行

训练,可以识别 C# 数据集中 95% 的漏洞。然而,通过特征转移使用其他语言的现有模型的检测模型的识别率降至 80%。尽管 CLaSCoRe 在原生 C# 语言模型检测中具有 0.97 的高准确率,但在从其他语言转移时,其平均准确率降至 0.89,召回性能较差,表明该模型在面对不同编程语言时的适应性和通用性不足。就平均召回率而言,使用 CLaSCoRe 方法后,召回率显著下降,从原来的 0.90 降至 0.72。相反,当 AST-FMVD 进行跨语言转移时,对于 CWE89 类型文件的平均检测性能仅略有下降,为 88.5%, F1 分数为 86.7%,召回率保持在 85%。比较实验结果如表 5 所示。结果表明,AST-FMVD 具有出色的可迁移性,即使在不同的编程语言环境中,仍能保持高水平的检测性能。

表 5 CLaSCoRe 和 AST-FMVD 之间的软件漏洞检测结果比较

Metrics	CLaSCoRe		AST-FMVD	
	Results	Loss Ratio/%	Results	Loss Ratio/%
Recall	From 0.90 to 0.72	20	From 0.85 to 0.85	0.0
F1 score	From 0.89 to 0.72	19	From 0.89 to 0.86	2.3
Precision	From 0.89 to 0.80	10	From 0.89 to 0.88	1
Transfer from (Languages)	C, C++, Java	Java, Python		

4 结束语

该文提出了一种基于抽象语法树特征空间映射的零样本漏洞检测方法,解决了在不同编程语言编写的文件中检测 SQL 注入类型漏洞的问题。通过执行特征转移和映射,同时强调在映射过程中的特定漏洞类型的提取和保留,从而避免了不必要的信息干扰。通过最大化保留代码中原始漏洞特征,保持了原始模型在其原生领域的检测有效性。此外,该方法利用了现有的预训练模型,减少了从头开始训练模型所需的时间和资源成本,使其变得实际和可行。

尽管这种零样本漏洞检测方法具有优点,但也存在一些局限性。该方法的泛化性能可能受到限制,因为它依赖于提取和保留特定漏洞类型的特征;此外,作为一种静态代码分析方法,它无法实现漏洞定位,无法精确定位源代码中漏洞的确切位置。这些都有待进一步完善。

参考文献:

[1] 李 韵,黄辰林,王中锋,等.基于机器学习的软件漏洞挖掘方法综述[J].软件学报,2020,31(7):2040-2061.
[2] 顾绵雪,孙鸿宇,韩 丹.等.基于深度学习的软件安全漏洞挖掘[J].计算机研究与发展,2021,58(10):2140-2162.
[3] LI Z, ZOU D, XU S, et al. Vuldeepecker: a deep learning-based system for vulnerability detection [J]. arXiv: 1801.

01681, 2018.
[4] YOSINSKI J, CLUNE J, BENGIO Y, et al. How transferable are features in deep neural networks? [J]. Advances in Neural Information Processing Systems, 2014, 27: 3320-3328.
[5] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition [C]// Proceedings of the IEEE conference on computer vision and pattern recognition. Las Vegas: IEEE, 2016: 770-778.
[6] SOYSAL O A, GUZEL M S. An introduction to zero-shot learning: an essential review [C]// 2020 international congress on human-computer interaction, optimization and robotic applications (HORA). Ankara: IEEE, 2020: 1-4.
[7] 庄福振, 罗 平, 何 清, 等. 迁移学习研究进展 [J]. 软件学报, 2015, 26(1): 26-39.
[8] 杨 强, 童咏昕. 迁移学习: 回顾与进展 [J]. 中国计算机学会通讯, 2018, 14(9): 36-41.
[9] LUTZ O, CHEN H, FEREIDOONI H, et al. ESCORT: ethereum smart contracts vulnerability detection using deep neural network and transfer learning [J]. arXiv: 2103.12607, 2021.
[10] ZAHARIA S, REBEDEA T, TRAUSAN-MATU S. Detection of software security weaknesses using cross-language source code representation (CLaSCoRe) [J]. Applied Sciences, 2023, 13(13): 7871.
[11] FENG H, FU X, SUN H, et al. Efficient vulnerability detection based on abstract syntax tree and deep learning [C]// IEEE INFOCOM 2020-IEEE conference on computer com-

- munications workshops (INFOCOM WKSHPS). Toronto; IEEE, 2020; 722–727.
- [12] MIKOLOV T, CHEN K, CORRADO G, et al. Efficient estimation of word representations in vector space [J]. arXiv; 1301.3781, 2013.
- [13] MIKOLOV T, SUTSKEVER I, CHEN K, et al. Distributed representations of words and phrases and their compositionality [J]. Advances in Neural Information Processing Systems, 2013, 26; 3111–3119.
- [14] HUANG E H, SOCHER R, MANNING C D, et al. Improving word representations via global context and multiple word prototypes [C] // Proceedings of the 50th annual meeting of the association for computational linguistics (volume 1: long papers). Jeju Island; ACL, 2012; 873–882.
- [15] BIESIALSKA M, RAFIEIAN B, COSTA-JUSSÁ M R. Enhancing word embeddings with knowledge extracted from lexical resources [J]. arXiv; 2005.10048, 2020.
- [16] ILGUN K, KEMMERER R A, PORRAS P A. State transition analysis: a rule-based intrusion detection approach [J]. IEEE Transactions on Software Engineering, 1995, 21 (3) : 181–199.
- [17] LANE T, BRODLEY C E. Sequence matching and learning in anomaly detection for computer security [C] // AAAI Workshop: AI approaches to fraud detection and risk management. Providence; AAAI, 1997; 43–49.
- [18] KRUEGEL C, KIRDA E, MUTZ D, et al. Automating mimicry attacks using static binary analysis [C] // USENIX security symposium. Baltimore; USENIX, 2005; 11.
- [19] MOU L, LI G, ZHANG L, et al. Convolutional neural networks over tree structures for programming language processing [C] // Proceedings of the AAAI conference on artificial intelligence. Phoenix; AAAI, 2016.
- [20] 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). Orlando; IEEE, 2018; 757–762.
- [21] PAN S J, YANG Q. A survey on transfer learning [J]. IEEE Transactions on Knowledge and Data Engineering, 2009, 22 (10) : 1345–1359.
- [22] LONG M, ZHU H, WANG J, et al. Deep transfer learning with joint adaptation networks [C] // International conference on machine learning. Tutorials; PMLR, 2017; 2208–2217.
- [23] NICKEL M, ROSASCO L, POGGIO T. Holographic embeddings of knowledge graphs [C] // Proceedings of the AAAI conference on artificial intelligence. Phoenix; AAAI, 2016.
- [24] SARPONG P A, LARBI L S, PAA D P, et al. Performance evaluation of open source web application vulnerability scanners based on OWASP benchmark [J]. International Journal of Computer Applications, 2021, 174 (18) : 15–22.
- [25] COCO J E, OSMAN A H, OSMAN I N. JPT: a simple java-python translator [J]. Computer Applications: An International Journal, 2018, 5 (2) : 1–5.