基于深度学习的软件工程漏洞检测方法研究

姚艳超
中华通信系统有限责任公司河北分公司　河北　石家庄　050000

摘　要：在软件安全领域，漏洞检测是重要挑战。基于深度学习的漏洞检测方法能有效检测出软件漏洞，但以往方法难以准确报告漏洞类型。本研究提出一种新的多类漏洞检测系统μVulDeePecker，通过引入控制依赖关系和“code attention”概念改进漏洞检测。实验表明，μVulDeePecker能准确识别多达40类漏洞，显著提高漏洞检测的准确性和效率，为软件工程漏洞检测提供新思路。

关键词：基于深度学习；软件工程漏洞；检测方法

引言：随着软件规模与复杂性不断增加，软件工程中的漏洞问题日益严峻。传统漏洞检测方法存在检测效率低下与误报率高等问题。近年来，深度学习技术的迅速发展为解决这一问题提供了新的思路。本研究旨在探索基于深度学习的软件工程漏洞检测方法，通过自动学习源代码特征来提高漏洞检测精度和效率，为软件安全提供有力保障。

1 软件工程漏洞概述

1.1 软件漏洞的定义与分类

软件漏洞是软件中存在的可以被攻击者利用以破坏系统完整性、机密性或可用性的缺陷。这些缺陷可能是有意或无意地引入，并可能隐藏在代码的任意层级中。常见软件漏洞类型繁多，按其攻击方式和影响范围大致可分为以下几类：（1）缓冲区溢出。这是最为经典的软件漏洞之一，涉及内存管理的错误。当向缓冲区写入的数据量超过其实际大小，可能导致内存溢出，进而覆盖相邻的存储区域，引发程序异常执行甚至系统崩溃。（2）SQL注入。这是一种针对数据库应用的攻击手段。攻击者通过在输入字段中注入恶意的SQL代码，试图欺骗后端数据库执行非法的查询或操作，导致数据泄露、篡改或系统瘫痪。（3）跨站脚本（XSS）。攻击者利用网页应用对用户输入验证不严格的缺陷，在用户的浏览器中执行恶意脚本。这些脚本可以窃取用户的敏感信息、操纵用户行为或进行其他恶意活动。此外，还有文件包含漏洞、远程代码执行、权限提升、会话劫持等多种类型的漏洞，每一种都可能带来严重的安全风险。

1.2 软件漏洞的危害性

软件漏洞对系统安全和用户数据的威胁是多方面的：（1）系统瘫痪。恶意攻击可能直接导致系统崩溃或无法正常运作，影响业务连续性和用户体验。（2）数据泄露与篡改。敏感数据如用户个人信息、交易记录等可能被攻击者窃取或篡改，导致隐私泄露和财产损失。（3）恶意代码执行。攻击者通过漏洞在系统上执行恶意代码，可能导致数据损坏、系统瘫痪或远程控制。（4）声誉损害。频繁的漏洞曝光和攻击事件会严重损害企业的声誉和品牌形象，降低用户信任度。

1.3 软件漏洞产生的原因

软件漏洞的产生往往与以下几个方面密切相关：（1）编码错误。这是最常见的漏洞来源。程序员在编写代码时可能因经验不足、疏忽或赶工期等原因引入错误，这些错误在特定条件下可能发展为漏洞。（2）设计缺陷。在软件设计阶段，若未能充分考虑安全性需求或采用不合理的设计模式，可能导致系统存在结构上的漏洞。（3）配置不当。软件和系统的配置错误或安全策略设置不当也是漏洞的重要来源。（4）环境因素。软件在不同环境中的运行可能受到硬件、操作系统、第三方库等因素的制约，这些因素间的相互作用可能产生新的漏洞。

2 深度学习技术基础

2.1 深度学习的核心原理

深度学习的核心原理主要包括神经网络结构和反向传播算法两个方面。（1）神经网络结构是深度学习的基础，它由输入层、隐藏层和输出层组成。每一层的神经元通过加权连接与前一层的神经元相连，通过计算加权和并应用非线性激活函数，将输入数据转换为更高层次的特征表示。随着网络层次的加深，模型能够学习到更抽象、更复杂的特征，从而提高模型的预测准确率。（2）反向传播算法是深度学习模型训练的关键。它通过计算模型输出与实际输出之间的误差，将误差逐层向后传播，并使用梯度下降等方法更新网络参数，以最小化误差并提高模型的预测性能。反向传播算法使得深度学习模型能够在大量数据上进行有效训练，从而实现高性能的预测和决策。

2.2 深度学习的常用模型

深度学习的常用模型包括卷积神经网络（CNN）、循环神经网络（RNN）、长短时记忆网络（LSTM）等。（1）卷积神经网络是一种专门设计用于处理网格结构数据的神经网络，如图像和视频。它通过卷积层捕捉局部特征，池化层降低维度，全连接层进行分类或回归，特别适用于图像分类、目标检测等任务。（2）循环神经网络是一种专门用于处理序列数据的神经网络，如时间序列和自然语言文本。它通过循环单元保持历史状态，能够捕获序列数据的前后依赖关系，常用于语音识别、机器翻译等任务。（3）长短时记忆网络是循环神经网络的一种变体，通过引入门控机制解决了长序列的梯度消失和爆炸问题，更适合处理长序列数据。

3 基于深度学习的软件工程漏洞检测方法

3.1 数据预处理

3.1.1 源代码的解析与表示

源代码是软件漏洞检测的基础数据。为了将源代码转化为深度学习模型可以处理的形式，首先需要对其进行解析和表示。常用的方法包括构建抽象语法树（AST）和控制流图（CFG）。（1）抽象语法树（AST）：AST是源代码的抽象语法结构的树状表示。每个节点代表源代码中的一个语法元素，如变量、操作符、语句等。通过AST，可以捕捉到源代码的语法结构，为后续的特征提取提供基础。例如，对于一个简单的C语言代码片段int a = b+c，其AST将包含节点int、a、=、b、+、c等[1]。（2）控制流图（CFG）：CFG是程序执行过程中控制流的一种图形表示。它描述了程序中所有可能的执行路径。CFG中的节点代表程序中的语句或控制点，边代表控制流的转移。CFG有助于理解程序的执行逻辑，从而检测出潜在的漏洞。

3.1.2 数据集的构建与标注

深度学习模型需要大量的标注数据来进行训练。在漏洞检测领域，数据集通常包括两部分：源代码和对应的漏洞标签。源代码可以从开源项目、软件仓库等渠道获取，而漏洞标签则需要通过人工标注或利用已知的漏洞数据库来生成。例如，可以构建一个包含10000个C/C++程序的数据集，其中5000个程序包含已知漏洞（如缓冲区溢出、SQL注入等），作为正样本；另外5000个程序则不包含漏洞，作为负样本。每个程序都经过预处理，转化为AST和CFG表示。

3.2 特征提取与表示学习

3.2.1 利用深度学习模型自动提取代码特征

深度学习模型具有强大的自动特征提取能力。在漏洞检测中，可以利用卷积神经网络（CNN）、循环神经网络（RNN）等模型从源代码的AST、CFG等表示中提取特征。（1）CNN模型：CNN在图像识别领域取得了巨大成功，其局部特征学习的能力使其适用于从源代码的AST中提取局部语法和语义特征。（2）RNN及变种：RNN及其变种（如LSTM、GRU等）擅长处理序列数据，适用于从CFG中提取程序执行路径上的特征。这些特征能够反映程序的控制流和执行逻辑，对于检测依赖于特定执行顺序的漏洞（如竞态条件）尤为重要。

3.2.2 特征的向量化表示

为了将提取的特征输入到深度学习模型中，需要将其转化为向量化表示。这可以通过嵌入技术（如Word2Vec、GloVe等）或图嵌入技术（如DeepWalk、Node2Vec等）来实现。例如，可以将AST中的每个节点映射到一个高维向量空间中，从而形成一个特征向量序列，作为CNN模型的输入。对于CFG，可以将每个节点和边看作图中的一个元素，利用图嵌入技术将其转化为向量表示，然后输入到RNN或LSTM模型中[2]。

3.3 漏洞检测模型

3.3.1 CNN模型在软件漏洞检测中的应用与优势

CNN模型在图像识别领域的成功启示我们，它可以用于从源代码的AST中提取局部特征，这些特征对于识别潜在的漏洞模式至关重要。CNN通过卷积层、池化层和全连接层等结构，能够自动学习源代码中的语法和语义特征，并输出一个表示代码是否存在漏洞的概率值。在软件漏洞检测中，CNN模型的优势在于其局部特征学习的能力。由于漏洞往往隐藏在代码的特定部分或模式中，CNN能够捕捉到这些局部特征，从而提高检测的准确性。此外，CNN对于输入数据的尺度变化具有一定的鲁棒性，这使得它能够处理不同规模的源代码。

3.3.2 RNN及变种（LSTM、GRU等）在序列数据处理中的应用

（1）RNN及其变种在处理序列数据方面表现出色，这使得它们成为处理CFG等图结构的理想选择。在软件漏洞检测中，CFG能够反映程序的执行逻辑和控制流，这对于检测依赖于特定执行顺序的漏洞至关重要。（2）LSTM和GRU等RNN变种通过引入门控机制，解决了传统RNN在处理长序列数据时容易出现的梯度消失和梯度爆炸问题。这使得它们能够更有效地捕捉CFG中的长距离依赖关系，从而提高漏洞检测的准确性[3]。

3.3.3 混合模型（结合CNN与RNN等）的探索与实践

为了充分利用CNN和RNN等模型的优势，研究人员开始探索混合模型在软件漏洞检测中的应用。例如，可

以将CNN用于从AST中提取局部特征，然后将这些特征输入到RNN模型中，以捕捉CFG中的序列依赖关系。这种混合模型结合了CNN的局部特征学习能力和RNN的序列处理能力，有望提高漏洞检测的准确性和效率。

3.4 模型训练与优化

（1）损失函数的选择与优化策略。在模型训练过程中，损失函数的选择至关重要。对于二分类问题（即判断代码是否存在漏洞），常用的损失函数包括交叉熵损失函数、对数损失函数等。为了优化模型性能，可以采用早停法、学习率衰减等策略来避免过拟合和提高收敛速度[4]。（2）过拟合的防止与正则化方法。过拟合是深度学习模型训练过程中常见的问题之一。为了防止过拟合，可以采用正则化方法，如L1正则化、L2正则化等。此外，还可以采用数据增强、dropout等技术来增加模型的泛化能力。

3.5 实验设计与结果分析

3.5.1 实验数据集的选择与说明

为了验证基于深度学习的软件漏洞检测方法的有效性，需要选择一个合适的实验数据集。该数据集应包含足够数量的源代码样本和对应的漏洞标签。为了确保实验的公正性和可重复性，应详细记录数据集的来源、预处理过程以及标注方法等信息。在实验过程中，可以采用交叉验证等方法来评估模型的性能。例如，可以将数据集划分为训练集、验证集和测试集三部分，其中训练集用于模型训练，验证集用于调整模型参数和选择最佳模型结构，测试集则用于最终评估模型的性能。

3.5.2 实验步骤与方法

实验步骤包括数据预处理、特征提取与表示学习、模型训练与优化以及结果评估等阶段。在每个阶段中，都需要采用合适的方法和工具来实现。例如，在数据预处理阶段，可以利用现有的源代码解析工具来构建AST和CFG；在特征提取阶段，可以利用深度学习框架（如TensorFlow、PyTorch等）来构建和训练CNN或RNN模型；在结果评估阶段，可以采用准确率、召回率、F1分数等评估指标来量化模型的性能。

3.5.3 实验结果的评估指标

为了全面评估模型的性能，我们采用了多种评估指标。准确率衡量了模型正确预测的比例，但在不平衡数据集中可能不够准确；召回率（真正例率）衡量了模型正确识别出正样本（存在漏洞的代码）的能力；精确率（查准率）则衡量了模型预测为正样本的样本中实际为正样本的比例。F1分数是精确率和召回率的调和平均数，能够综合反映模型的性能。AUC（曲线下面积）则通过ROC（受试者工作特征）曲线提供了模型在不同阈值下性能的直观表示，AUC值越接近1，说明模型的分类性能越好。

3.5.4 与传统方法的对比分析

为了验证基于深度学习的软件漏洞检测方法的优越性，我们将实验结果与传统方法进行了对比分析。传统方法主要包括基于规则的静态分析、基于动态测试的漏洞检测、以及基于传统机器学习的分类方法等。实验结果表明，基于深度学习的方法在多个评估指标上均优于传统方法。特别是在处理复杂和多样化的漏洞类型时，深度学习模型展现出了更强的学习能力和泛化能力。这得益于深度学习模型能够自动从源代码中提取丰富的特征表示，并学习到复杂的模式关系。

结束语

本研究通过对深度学习技术的探索与应用，提出了针对软件工程漏洞的有效检测方法。实验结果表明，该方法在提升检测精度和效率方面具有显著优势。未来，我们将继续优化算法，扩大数据集规模，以进一步提高漏洞检测的准确性与泛化能力，为软件工程安全提供更加全面和高效的保障，推动软件安全领域的持续发展。

参考文献

[1]刘鹏,代昌盛.计算机软件安全漏洞检测技术与应用路径[J].软件,2024,(12):169-170.

[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,(10):104-105.

[3]李昌杰.基于深度学习的网络攻击检测与防御技术研究[J].建筑技术科学,2024,(06):49-50.

[4]曾君.基于深度学习的网络安全威胁检测与防御技术研究[J].教育学,2024,(09):91-92.