基于词法分析的代码漏洞模式识别

摘要

软件技术与社会生活的方方面面越来越紧密，软件开发又不可避免的产生各种漏洞，而漏洞识别与定位十分耗费人力物力。如何快速识别并定位漏洞以提升软件运行的稳定与安全成为越来越严峻的问题。随着深度学习的发展，出现了一些可快速自动化的识别软件漏洞的方法，譬如基于代码抽象语法树（以下简称AST）或程序数据流图(以下简称PDG)的漏洞识别方法，从AST或PDG中提取漏洞特征进行模式识别，但有些代码语义譬如通过AST进行抽象后会消失，不利于漏洞的识别。 本文提出一种基于词法分析的代码漏洞特征提取方案，以利用更广的语义依赖进行漏洞识别。

关键字：漏洞识别，词法分析，抽象语法树，程序数据流，深度学习

引言

    软件漏洞与软件开发相伴相生。由于软件规模的不断扩大，导致软件复杂度也不断变高，兼之开发人员对软件开发技术熟练程度、对开发逻辑理解等存在差异，不可避免的在软件开发过程中不自觉的引入各种漏洞（业界也称为“Bug”），而且随着业务的迭代，老旧的程序代码可能不适应新业务的需求发展，也导致出现各种漏洞。

    漏洞是潜伏在软件系统中的，发现这些各种各样的漏洞通常是一种被动的行为，一般在软件运行期时系统出现问题了，漏洞才会被开发人员识别并修复，这使得维护软件运行的稳定性相对滞后。而且如果软件系统复杂性很高，开发人员识别漏洞的时间也可能相对变长，对开发人员的技术水平要求也可能相对提高，从而使得漏洞识别是一件十分耗费人力的事。

    其次，已经有一些静态的漏洞分析工具可以帮助开发人员较快的识别漏洞。如 Clang Static Analyzer(1)，这些工具一般是通过人工预设的漏洞语义判断逻辑，譬如：如果一个对象分配方法与其释放方法要成对调用，否则就判断代码片段有内存泄漏漏洞。这要求静态分析工具要随漏洞语义的变化而不断更新其的判断逻辑，十分耗费物力财力。而且有些漏洞要在软件运行期才有可能表现出来，静态分析工具无法动态识别漏洞，覆盖率低，导致很容易出现漏报或误报的情况。

    2016年，Zhen Li等提出了称为“VulPecker”[1]的自动识别系统漏洞的方法，基于代码克隆技术，利用代码相似性从源代码中检测漏洞。但这些方法本质还是通过静态分析方法预先抽取已知的漏洞特征，通过相似性比较来检测代码是否有漏洞，但不能自动提抽取代码特征来识别新的漏洞。2018年，Zhen Li等又先后提出了VulDeePecker[2] 和SySeVR[3] 两个利用机器学习可自动识别漏洞的方案，其系统的主要思想是先通过AST找出可以可疑漏洞点（变量或函数方法），然后找出与漏洞点语义相关的语句进行训练学习，其模型的效果有不错的漏洞低误报率和低漏报率，相比现有的静态分析工具有漏洞识别成功率有很大提升。但其可疑漏洞点的假设仍然离不开静态的判断规则，人工假设性过强。Aram Hovsepyan[4] 等基于词袋模型（bag-of-word）将其Java代码进行编码，然后用支持向量机自动进行代码段的漏洞识别，但其训练与测评方法只局限于单个数据集，算法通用性与泛化能力有待考量。Yulei Pang[5] 等在此基础上，使用N-gram模型对代码片段进行编码，然后也使用支持向量机自动进行代码漏洞识别，获得不错的性能。Jón Arnar Briem[6] 等将AST的结点编码为向量，然后输入到神经网络进行漏洞识别学习，得益于AST能比较好的抽象出程序代码元素间的相互依赖关系，此模型对漏洞的识别率比较高，但其只提供一种验证方案，无法考虑其算法通用性。Fen Hantao[7] 等也对基于AST的漏洞识别方法进行研究，使用Bi-GRU[9] 作为主要的模型架构。还有张启航[8] 同样也是提出了基于AST的漏洞识别方案，其主要区别是没使用AST的全部结点对代码进行编码，而只选取了变量、方法名和程序控制逻辑结点，一定程序度减轻了这类基于AST编码方案的复杂度，且使用GPT[10] 和Bi-GRU作为漏洞特征的抽取模型架构，增加了代码内部各元素对于其漏洞语义的注意力因素，对漏洞识别率的提升也有很好的帮助。

     使用AST会把代码元素的组织架构去掉，因此以上基于AST的漏洞识别方法都没有使用代码片段的所有元素进行特征抽取。而有些漏洞在代码原有结构中才比较容易看出来，譬如offset by one error，递归中忘记终结条件，等。本文将使用词法分析后的所有代码元素对代码进行编码，以解决这种将代码抽象后结构缺失后的问题，并使用卷积神经网络进行漏洞特征提取，以从更广为的语义依赖范围进行漏洞模式识别。

**参考文献**

[1] Zhen Li,Deqing Zou,Shouhuai Xu, et al. Vulpecker: an automated vulnerability detection system based on code similarity analysis[Z], 2016.

[2] Zhen Li,Deqing Zou,Shouhuai Xu, et al. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection[Z], 2018.

[3] Zhen Li,Deqing Zou,Shouhuai Xu, et al. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities[Z], 2018.

[4] Aram Hovsepyan,Riccardo Scandariato,Wouter Joosen, et al. Software vulnerability prediction using text analysis techniques[Z], 2012.

[5] Yulei Pang,Xiaozhen Xue,Akbar Siami Namin. Predicting vulnerable software components through n-gram analysis and statistical feature selection[Z], 2015.

[6] Jón Arnar Briem,Jordi Smit,Hendrig Sellik, et al. Using Distributed Representation of Code for Bug Detection[Z], 2019.

[7] Hantao Feng,Xiaotong Fu,Hongyu Sun, et al. Efficient Vulnerability Detection based on abstract syntax tree and Deep Learning[Z], 2020.

[8] 张启航. 基于抽象语法树的代码缺陷检测技术设计与实现[D], 2020.