基于模糊测试的SQL注入漏洞挖掘与定位技术研究

吴昊天, 刘嘉勇, 贾 鹏
(四川大学网络空间安全学院, 成都 610065)

摘 要: 随着Java语言成为最流行的开发语言之一,其在金融、政府和企业等行业的重要Web应用系统开发中发挥关键作用. 然而,Java Web应用的普及也使其成为网络攻击者的主要目标,导致安全漏洞频发,其中一种危害最大的漏洞就是SQL注入漏洞. 现有的SQL注入漏洞检测方法以漏洞扫描、机器学习和模糊测试等技术为主. 但这些技术在测试用例生成方面面临挑战:漏洞扫描依赖固定测试用例,机器学习技术样本集训练成本高,而传统模糊测试的用例变异缺乏针对性,从而导致上述技术在SQL注入漏洞挖掘方面的能力较低. 因此,针对于上述问题,本文提出一种基于模糊测试的SQL注入漏洞挖掘与定位方法. 该方法首先结合遗传算法的选择、交叉和变异过程,并在变异阶段引入绕过安全措施的变异策略和基于打分机制的变异策略调度机制,使得初始小样本集合能快速收敛至能有效绕过代码过滤措施的高质量个体. 提高了测试用例多样性和对漏洞挖掘的针对性,从而提升SQL注入漏洞检测能力. 此外,本方法利用字节码插桩技术动态获取目标应用程序的运行时状态信息,并结合网络报文来实现漏洞的检测和精确定位,进而提升漏洞挖掘的准确性和多层次漏洞信息的记录能力. 为进一步验证本方法的有效性,本文基于上述策略实现GAFuzz工具,并将其与现有漏洞挖掘工具和标准遗传算法进行对比实验. 实验结果表明GAFuzz在生成高效攻击用例、漏洞挖掘及提供漏洞描述信息方面均展现出较优性能.

关键词: 模糊测试; SQL注入; 测试用例生成; 遗传算法; 漏洞定位
中图分类号: TP391.1 **文献标志码:** A DOI: 10.19907/j.0490-6756.2024.053001

A study on SQL injection vulnerability discovery and localization techniques based on fuzz testing

WU Hao-Tian, LIU Jia-Yong, JIA Peng
(School of Cyber Science and Engineering, Sichuan University, Chengdu 610065, China)

Abstract: As Java becomes one of the most popular programming languages, it plays a critical role in web application development in sectors like finance, government, and business. However, the widespread use of Java web applications has made them a major target for cyber attacks, leading to frequent security vulnerabilities, especially SQL injection flaws. Existing detection methods for SQL injection vulnerabilities, such as vulnerability scanning, machine learning, and fuzz testing, face challenges in test case generation. These methods either depend on fixed test cases, incur high training costs for machine learning models, or lack tar-

收稿日期: 2023-12-30 **修回日期:** 2024-02-27 **接受日期:** 2024-02-27
基金项目: 全军共用信息系统装备预研项目(31511080306)
作者简介: 吴昊天(2000—), 女, 安徽阜阳人, 硕士研究生, 研究方向为模糊测试和Web安全. E-mail: wuhaotian41@163.com
通讯作者: 贾鹏. E-mail: pengjia@scu.edu.cn

geted mutation in traditional fuzz testing, reducing their effectiveness in detecting SQL injection vulnerabilities. To address these issues, this paper introduces a fuzz testing-based method for identifying and locating SQL injection vulnerabilities. This method integrates genetic algorithm processes (selection, crossover, and mutation) with mutation strategies that bypass security measures and a scoring mechanism for mutation strategy scheduling. This approach allows for the rapid convergence of a small initial sample set to high-quality cases that effectively bypass code filtering, enhancing the diversity and targeting of test cases for vulnerability mining. Additionally, the method uses bytecode instrumentation to dynamically capture runtime state information of applications and network data for accurate vulnerability detection and localization. The effectiveness of this method is further demonstrated by the development and testing of the GAFuzz tool, which is compared with existing vulnerability mining tools and standard genetic algorithms. The results show that GAFuzz excels in generating effective attack cases, mining vulnerabilities, and providing detailed vulnerability information.

Keywords: Fuzz testing; SQL Injection; Test case generation; Genetic algorithm; Vulnerability localization

1 引　言

随着互联网的不断发展，Web应用程序得到了广泛的应用，成为许多敏感信息流动的平台. 因此，针对Web应用程序的攻击层出不穷. 国家信息安全漏洞共享平台（CNNVD）的统计数据[1]表明，在漏洞影响的对象类型中，Web应用占比27.8%，位于第二. 统计数据也表明，Web应用漏洞产生的原因除了本身架构设计的错误外，28.8%都是由于输入验证错误导致，由此产生一系列参数注入型漏洞攻击，像SQL注入漏洞等.

SQL注入[2]是在前端输入参数向后端传递的过程中，由于没有进行安全限制和过滤，使得攻击者输入的攻击负载可以直接拼接到正常执行的SQL语句中，从而执行攻击语句来获取相关敏感信息. SQL注入漏洞也由于其简单和多方面的可利用性，造成了许多安全事件. 如2023年的MOVEit攻击就成功利用SQL注入漏洞勒索施耐德电气、西门子能源等公司[3]，造成了巨大的损失. 因此针对于复杂多变的SQL注入攻击，如何有效地发现Web应用程序中的SQL注入漏洞并准确定位漏洞点已成为当前研究的焦点.

模糊测试技术[4]的主要思想就是传输大量随机数据到应用程序中，通过监控应用程序出现的异常来进行漏洞挖掘. 由于其自动化程度高、使用成本低的特性已经成为web安全领域主流的漏洞挖掘方法. 然而，传统模糊测试的测试用例的随机性并不能充分应对以下问题：SQL注入漏洞中对于SQL用例语法的严格要求、Web应用程序中的安全过滤措施等. 这使得模糊测试针对于SQL注入漏洞的挖掘性能大大降低. 并且当前SQL注入检测工具主要是通过分析Http响应报文进行漏洞检测，存在一定的误报. 并且在检测出漏洞后只能输出URL接口、触发Payload和报文字节数等相关信息，不利于研究人员对漏洞进行定位与复现.

因此，针对上述问题，本文提出一种基于遗传算法[5]的方式来生成模糊测试挖掘SQL注入漏洞的测试用例，并且使用字节码插桩技术对Java语言实现的SQL漏洞程序代码进行插桩，基于程序函数内部信息对漏洞进行检测和定位. 本文的主要工作及贡献如下所示.

（1）使用遗传算法生成模糊测试的测试用例，利用其交叉和突变特性，丰富测试用例的结构和组合的多样性，并在适应度值的指导下迭代生成精英个体集合.

（2）将绕过安全检测的方法融入到遗传算法的变异步骤中，并引入自适应变异调度策略，根据相关参数对变异策略进行打分和调度. 这种方法增大了攻击向量绕过安全策略的概率，并通过优化变异策略的调度使得遗传算法以较快的收敛速度得到更高质量的个体.

（3）提出一种基于字节码插桩技术的SQL注入漏洞判定方法. 在该方法中通过插桩目标程序内部SQL注入相关的危险函数，得到程序运行时的细粒度信息，并结合报文信息同时对漏洞进行检测和定位，相较于黑盒测试工具可以降低漏洞检测的误报率. 同时将得到的插桩信息作为遗传算法中适应度值计算的一部分，用于迭代优化生成精英个体.

2　相关工作

2.1　基于静态分析的SQL注入攻击检测技术

静态检测技术[6]的主要思想是通过词法分析、类型推断、数据流分析等技术对目标程序源代码进行分析,并根据SQL注入漏洞的特征进行代码匹配从而找到漏洞. Wan等[7]提出一种利用语法分析的方法来检测SQL注入攻击,通过提取公共子结构和语法解析树中的重复子结构来识别可疑SQL语句中的恶意部分. 但是SQL注入攻击语句的多样性使得构建语法树的难度增加且过程耗时. Hlaing等[8]通过在预定义的词典单词中搜索标记来检测SQLIA,词典中收集了大部分SQL注入语句的命令或者单词,通过比较输入语句与已规定的词典内容是否相吻合来检测SQL注入漏洞. 但该方法的检测准确性的高低严重依赖于词典对于SQL攻击语句特征覆盖的完整性,使得该检测方法很容易被绕过.

2.2　基于机器学习的SQL注入攻击检测技术

为了提高SQL注入攻击检测的准确率并减少人工成本,研究者们开始采用机器学习[9]进行检测. 该类技术的基本思想是通过对现有SQL语句数据库进行特征分析、模型训练,然后进行二分类问题的求解.

Li等[10]提出使用一种自适应的深度森林方法,用于检测SQL注入攻击. 该方法将上一层输出的平均值和原始特征向量的连接作为每层的输入值并引入自适应算法AdaBoost使用每层的错误率来动态调整特征的权重. 实验表明,该方法比经典的机器学习方法具有更好的性能. Gandhi等[11]提出一种基于混合CNN-BiLSTM的模型来识别SQL注入攻击. 该模型通过使用CNN网络用于SQL语句的特征提取,BiLSTM网络用于学习特征间的依赖关系,以此来提升对SQL注入攻击检测的性能. 但是这种复合的网络架构复杂度较高,训练过程中需要消耗的计算成本高.

同时,由于SQL注入攻击利用的简单便利性,现实世界中攻击者往往会在恶意有效载荷中混合编码字符、大小写等多种方式来绕过过滤器. 注入载荷不断变化的多样性和特征集的增加,使得机器学习算法受到固定数量的特征和训练成本的限制.

2.3　SQL注入模糊测试工具研究现状

测试用例的质量是影响模糊测试漏洞检测能力的重要因素之一. 在SQL注入漏洞模糊测试领域,Hammersland等[12]提出了一种半自动伪随机数据的测试用例生成的Web应用程序模糊化方案,该方案在固定脚本的攻击结构中将注入点标记为模糊符号,并在模糊过程中使用随机生成的测试用例替换它. 但是该工具一方面测试用例具有较大的随机性,无法针对性测试SQL注入漏洞;另一方面该工具只能根据返回的Http状态码监控程序运行的状态,必须要人工来分析响应才能定位测试用例是否触发漏洞.

后来研究者们逐渐采用智能优化算法提高自动化测试中测试用例的生成质量. Cai等[13]将遗传算法应用到模糊测试的测试用例生成过程中,显著提高了测试用例的生成效率. 但该算法在收敛度性能和测试用例覆盖深度方面仍然存在一定的问题. Huang等[14]通过引入模拟退火机制优化遗传算法的相关参数,并将其应用于软件测试数据的自动生成,提高了测试用例自动生成的效率. 陆紫光[15]使用一种改进的自适应遗传算法用于Web模糊测试领域中测试用例的生成,经结果分析表现出良好的性能. 但是该算法在变异阶段直接将编码的数字进行反转,仍然存在一定的缺陷. 上述结果表明,遗传算法在测试用例生成中具有良好的优化性能.

在SQL注入模糊测试领域,已有多种工具被开发应用. SQLMap工具[16]是当前最流行的SQL注入检测工具,专为自动化扫描和检测URL中的SQL注入而设计,覆盖了如报错注入、布尔盲注及Union联合注入等多种注入类型. 然而,使用该工具往往需手动指定注入点和绕过技术. WFuzz[17]则是一个专为Web应用设计的模糊测试工具,利用字典攻击或组合输入对目标Web应用进行爆破. 模糊测试性能严重依赖于测试用例的质量. WaPiti[18]是一个黑盒Web应用安全审计工具,旨在检测常见安全漏洞,通过发送特定请求并分析响应来识别潜在漏洞. 尽管其可以将漏洞检测结果网页化,具有一定的可读性,但对于漏洞信息描述仍只限于报文和日志.

综上,遗传算法在测试用例的生成领域具有很好的性能,但是在当前研究中,对于变异阶段仍然存在一定的缺陷. 而当前普遍使用的SQL注入

检测工具的测试用例大部分都是基于内部字典固定的，不能够灵活针对于被测目标的相关过滤措施进行绕过或者进行绕过策略的调整，致使测试用例难以成功注入至预定目标．除此之外，这些工具大多停留在URL接口层面的漏洞检测，未能精确地定位到潜在漏洞所在的代码部分．

3　研究方法

3.1　总体框架概述

本文所提出的基于模糊测试检测SQL注入的框架如图1所示．整体框架有两部分组成：基于优化的遗传算法生成模糊测试测试用例和基于ByteBuddy插桩技术对漏洞进行定位和检测．

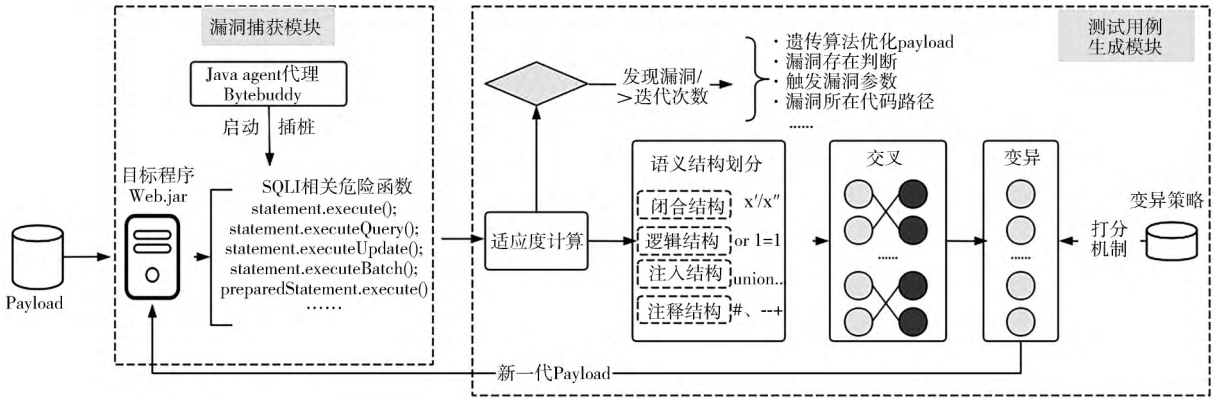

图1　系统总体架构图

Fig. 1　Overall architecture of the system

首先目标程序端利用Java agent代理和ByteBuddy字节码插桩技术在启动目标Web应用程序的同时，在程序内部与SQL注入相关的危险函数处进行插桩拦截，获取程序内部信息，像执行参数、执行结果、抛出异常、函数所处代码路径和函数所在代码行数等信息．这些信息将被添加到对应URL接口的响应报文字段中，通过http通信技术传输到遗传算法模块中，进行适应度计算和漏洞检测判断．

遗传算法端的主要功能是在一次次迭代中生成可有效绕过Web应用程序过滤措施的测试用例．在该过程中，首先将初始样本集提交到目标程序中进行模糊化测试，然后根据目标程序端返回的响应报文和各种插桩获得的信息进行适应度计算，接着根据遗传算法的原理，重复进行个体选择、交叉和变异等操作，直至发现漏洞或者循环达到设置代数．最终整个模块将输出是否存在漏洞、漏洞的定位信息和进化的个体集合．

3.2　目标程序端插桩

为了获取Web应用程序内部执行的各种细粒度信息，本文将使用Java agent代理和ByteBuddy相结合的技术[19]进行字节码插桩的实现．

Java agent是Java虚拟机提供的一种机制，允许在程序运行时对字节码进行修改和增强．它主要通过在目标程序的main方法执行前调用premain()方法，并在其中调用transform()方法，使得后续加载的所有类都会被拦截并可以进行字节码转换．

ByteBuddy是一个轻量级的用于生成和修改Java字节码的库，其主要优势在于提供了一系列API和注释直接对字节码进行操作．

图2概述了本文中ByteBuddy插桩的执行流程．首先会在premain()函数中实例化AgentBuilder对象，也就是Java agent代理．代理通过在命令行中的-javaagent参数中设置启动．然后premain()函数中实例化拦截器对象transform，当函数Advice. on(HookMethod)匹配到拦截的危险函数时，visit()函数将访问拦截代码Advice. java. 此时，拦截代码会注入到目标危险函数处．

因为如表1所示的函数提供了执行动态SQL查询的功能，并且如果这些函数没有对用户输入进行充分的验证和转义，就可以导致攻击者通过注入恶意SQL代码来造成安全漏洞．因此，在已公布的Java SQL注入漏洞中，表1所示这些函数经常被攻击者利用来引发危害．因此本文主要选取表1所示函数作为危险函数．同时，由于这些函数的参数、执行结果和抛出异常等信息与SQL注入漏洞挖掘的准确性相关联，因此通过对表格内所示函数的插桩可以提高漏洞挖掘能力．

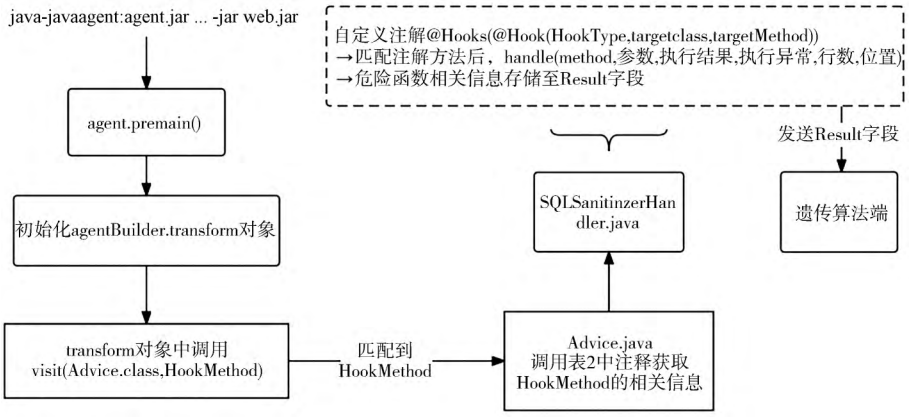

图 2　ByteBuddy 插桩流程示意图

Fig. 2　Schematic diagram of ByteBuddy piling process

表 1　SQL 注入危险函数

Tab. 1　SQL injection danger functions

类名称	函数名称
java. sql. Statement	execute( )
	executeQuery( )
	executeUpdate( )
	executeLargeUpdate( )
	executeBatch( )
	executeLargeBatch( )

Advice 代码则主要使用表 2 所示的注释对目标危险函数进行信息获取. 针对于 SQL 注入,本文主要使用@OnMethodExit 注解使 Handler 处理方法在危险函数执行后调用,便于获取函数执行对象和异常来进行漏洞的检测. 并且通过获取当前 Advice 类线程执行的堆栈信息获取危险函数所处代码路径和行数.

表 2　Advice 类注释及其作用

Tab. 2　Advice class annotations and their effects

注解名称	作用
@OnMethodEnter	表示注解的方法会在拦截的目标函数执行前调用
@OnMethodExit	表示注解的方法会在拦截的目标函数执行后调用
@Origin	获取拦截的目标函数对象
@AllArguments	获取目标函数的所有参数
@Thrown	获取目标函数抛出的异常
@Return	获取目标函数执行后的对象

在插桩过程中得到的信息将存储至自定义的 Result 结构体中,存储的信息结构如图 3 所示. 函数行数和堆栈信息主要用于漏洞的定位,函数相关信息则用于漏洞的检测.

3.3　遗传算法生成测试用例

在本节中,主要解释了如何将遗传算法集成到 SQL 注入漏洞模糊测试中,包括数据预处理、适应度计算、个体选择、交叉、变异策略调度的相关原理实现.

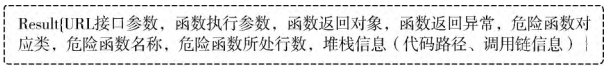

图 3　插桩信息存储结构示例

Fig. 3　Example of pile information storage structure

3.3.1　*数据预处理*　由于 SQL 语句在执行时具有严格的语法和语义要求,因此为了保证后续交叉和变异操作中其结构不被破坏,在该步骤中需要根据语法语义结构对输入的 Payload 进行序列划分,使得后续操作均以划分的各个序列为基本操作单元.

在这里,本文将一个 Payload 划分为 4 个子序列,每个序列的定义如表 3 所示. 这时,当 Payload 为 T_{sql} = 1′union select 1, 2 , database( ) −− 时,其划分后的序列如图 4 所示.

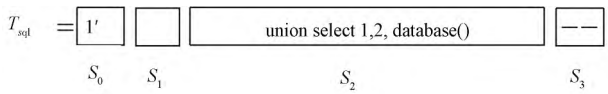

图 4　Payload 序列划分示例

Fig. 4　Example of Payload sequence partitioning

3.3.2　*适应度计算*　在遗传算法中,个体的适应度越高,其性能表现越优秀. 在本文中,为了衡量测试个体检测 SQL 注入漏洞的能力高低,结合 SQL 注入漏洞的特征,适应度主要被以下因素影响:是否检测出漏洞、响应页面的变化量和测试序列的有效载荷量. 计算公式为:$F = E_s + H_S + V_s$,计算规则如下所示.

(1) 当测试用例中某个个体能成功触发 SQL

注入漏洞时，模糊测试流程可以结束，该个体适应度值应设置为最高．SQL注入按照执行效果可以分类为报错注入、联合注入、布尔盲注、堆叠注入和时间盲注几大类型．因此结合上述各种类型的SQL注入漏洞的检测原理和在目标程序中插桩得到的函数相关信息，该部分适应度得分E_s的具体计算规则如表4所示．

（2）注入攻击载荷后页面的变化量H_S通过计算有效载荷注入前后响应页面的字符数的差值进行量化．在标准SQL查询场景中，预期的响应只呈现有限的数据．但在注入攻击的情况下，响应可能会显示额外的查询结果．尤其在布尔盲注中，会通过页面的不同响应来推测敏感数据信息．因此，较大的页面变化量可能指示着成功的SQL注入攻击．该指标计算公式如式（1）所示．

表3　Payload**序列化分标准**

Tab. 3　Payload serialization standard

序列	名称	描述
S_0	闭合序列	用于闭合正常的SQL查询语句，如单引号或双引号等；
S_1	逻辑序列	逻辑表达式判断序列，如or 1=1或and 1>2等；
S_2	注入序列	表现注入类型特征的攻击语句，如union xxx、extractvalue()、sleep()函数等；
S_3	注释序列	用于在攻击语句结尾注释后续sql语句，如--+、#等；

表4　**漏洞检测标准**

Tab. 4　Vulnerability detection standards

注入类型	检测特征	适应度得分	总计
报错注入	函数执行抛出特定异常，像SQLException:XPATH syntax error或者包含数据名名称等敏感信息	1	2
	函数执行的SQL语句包含报错注入的危险函数特征，像updatexml()、extractvalue()等	1	
联合注入	函数成功执行，返回相应对象类，且返回多条查询结果	1	2
	函数执行的SQL语句包含union关键字	1	
堆叠注入	函数成功执行，返回相应对象类，且返回多条查询结果	1	2
	函数执行的SQL语句包含“；+ select/update等关键字”的结构特征	1	
布尔盲注	函数成功执行，返回相应对象类	1	2
	函数执行的SQL语句包含“length/Ascii/substring(...)+[>/</=]...”的结构特征	1	
时间盲注	函数成功执行，返回相应对象类	1	2
	函数执行的SQL语句包含“if(...sleep(...)/benchmark(...)...)”结构特征	1	

$$H_S=\frac{\left|N(P_{\text{normal}})-N(P_{\text{injection}})\right|}{N(P_{\text{normal}})} \tag{1}$$

（3）测试序列的有效攻击载荷量V_S则通过计算输入到URL接口的初始测试序列$S(I)$和在SQL注入相关危险函数处的执行参数$S(O)$（即经过程序过滤的测试序列）的相似度这一指标来度量．二者相似程度越高，说明模糊测试序列被程序过滤得更少，其更容易绕过安全过滤策略进而触发漏洞．该部分使用Levenshtein距离来计算相似性，具体的计算方式如式（2）所示．

$$V_s=\mathrm{Dis}(I,O)=\frac{1}{1+\mathrm{Lids}_{(I,O)}} \tag{2}$$

Levenshtein距离具体计算原理如式（3）所示．它表示I的前i个字符与O的前j个字符之间的Levenshtein距离．

$$\mathrm{Lids}_{(I,O)}(i,j)=\begin{cases}\max(i,j),\ \text{if}\ \min(i,j)=0\\ \min\begin{cases}\mathrm{Lids}_{(I,O)}(i-1,j)+1\\ \mathrm{Lids}_{(I,O)}(i,j-1)+1,\ \text{otherwise}\\ \mathrm{Lids}_{(I,O)}(i-1,j-1)+1\end{cases}\end{cases} \tag{3}$$

3.3.3　**交叉与变异**　交叉与变异是遗传算法中最重要的两个步骤，可以丰富个体种群的多样性以提高触发漏洞的能力．在这两个操作进行前，需要从初始Payload中根据适应度高低选择一部分个体作为交叉和变异的父代集合．算法1展示了遗传算法中选择、交叉和变异的主要思想．

算法1　遗传算法交叉变异操作

输入：初始Payload集合pop和适应度值；

输出：变异后生成的新一代集合pop_m；

1）while $t < D$:

2)　$P_i=(F_i/F_{sum}$　for P_i in pop)
3)　pop_s = sort(pop, P_i)
4)　pop_c = select(pop_s, P)
5)　pop_cc = crossover(pop_c, pop, P_c)
6)　pop_m = mutate(pop_cc, stra, P_m)
7) **end while;**
8) **select:**
9)　for(i, P_i in pop_s, P):
10)　　r = random. random()
11)　　if r <= P_i:
12)　　pop_c. append(pop_i)
13) **crossover:**
14)　for father in pop_c:
15)　　child = father
16)　　r_child[S_0, S_1, S_2, S_3] = random
17)　　mother = random(pop)
18)　　if random_child[S_i] <= P_c
19)　　child[S_i] = mother[S_i]
20)　　pop_cc . append(child)
21) **mutate:**
22) update_stra_score(F, V_s, frequency_{stra})
23) stra_list = sort(score, stra)
24)　for Indi in pop_cc:
25)　　r_Indi[S_0, S_1, S_2, S_3] = random
26)　　if random_child[S_i] <= P_m
27)　　child[S_i] = stra_list[i](child[S_i])
28)　　pop_m . append(child)

在选择操作中(算法第8~12行),本文所采用的选择策略是轮盘赌策略[20]. 该选择策略是根据个体适应度的值确定其被选择的概率,第i个个体被选择概率的计算公式如式(4)所示. 对于集合中的每个个体生成一个0-1的随机数,当随机数小于个体的选择概率P_i,将该个体添加到集合pop_c中,用于后续操作. 因此,个体适应度值越高,其被选择的概率越大.

$$P_i=\frac{F_i}{\sum_{n=1}^{M}F_n} \tag{4}$$

交叉就是指个体集合两两进行随机序列的交换. 在本文SQL注入的测试用例生成中则是在划分序列的基础上,进行相同位置序列的互换. 该操作既丰富了个体种群的多样性又保证了SQL语句的整体语法结构没有被破坏.

在交叉操作实现过程(算法13~20行)中,首先根据交叉概率P_c确定从初始种群pop(m个)中选择的交叉父种群pop_c($m\times P_c$个). 然后随机选择初始种群pop的一个序列作为交叉的另一个父个体. 然后为个体的每个序列(S_0, S_1, S_2, S_3)生成对应的4个0-1的随机数,当序列对应的随机数大于交叉概率P_c时,该序列发生交叉操作. 这时得到子代集合pop_cc($m\times P_c$个).

在交叉得到的子代集合上后续将进行变异操作. 尽管当前Web程序端考虑到用户输入的不合法性过滤了一些不合法字符,但是仍有部分方法可以绕过这些安全策略,从而完成SQL注入. 通过对现有过滤机制的研究,本文将采用如下共计8种变异策略.

(1) 字符编码. 在该策略中,本文主要选取了Java Web程序中常见的几种编码方式,向URL编码、base64编码、十六进制和Unicode编码方式.

(2) 随机大小写. 有些安全策略可能只是纯过滤大写或者小写字符,因此本文对字符串随机位置进行大小写变换即可绕过该方式. 如union可能的变异结果为UnIoN.

(3) 敏感关键字冗余. 某些变异策略可能只对敏感关键字,像select、union等过滤一次. 这时通过在关键字内部重复插入关键字可有效抵挡该措施.

(4) 同义词替换. 该方法主要是将常被过滤的字符替换为同功能的其他字符来绕过过滤. 像将and替换为&&,等号替换为like,空格替换为/**/等.

(5) 混淆关键字. 在该部分中使用内联注释符(/*! */)对常见关键字进行结构上的改变以绕过对于固定结构的过滤.

变异过程(算法21~28行)中,个体的各个序列部分具有一定的变异概率P_v,当每个序列部分对应产生的随机数小于P_v时,该序列进行变异操作.

相较于传统遗传算法的随机选择变异策略,本文将在变异阶段根据适应度值、整体使用频次、序列过滤情况对使用的变异策略进行打分. 打分的基本策略为

1) 所有变异策略初始分数为0;

2) 当个体适应度值大于等于该轮平均适应度

时，其对应的变异策略得分＋1分；

3）当变异策略当轮使用频次大于等于当轮变异序列的个数/变异策略总数时，其对应的变异策略－0.5分；

4）当经变异后的序列的有效载荷没有被安全策略过滤时，其对应变异策略得分＋1分；

打分策略2）和4）的目的是使得整体的变异趋势向高质量个体靠拢，增大触发漏洞的可能性．打分策略3）则是为了保持个体集合的多样性，避免出现过多重复集合．在得分计算完成后，根据得分情况优先选择较高得分的变异策略进行变异操作，返回集合pop_m.

为了防止程序陷入无限循环，需要为遗传算法设置迭代阈值D. 当没有发现漏洞和迭代次数小于阈值时，遗传算法将重复上述操作，直至触发漏洞或达到阈值．

4　实验结果与分析

4.1　实验相关设置

4.1.1　*实验环境与数据集*　本文实验是在处理器为11th Gen Intel（R）Core（TM）i7-11700 @ 2.50 GHz的Windows 10操作系统环境下运行．遗传算法模块和Java agent模块分别是在Python3.6和jdk1.8环境下开发和运行的．

本文将Github上的“SQL Injection Payload List”项目[21]中的部分语句作为初始测试用例，其中包含了能够触发不同类型的SQL注入漏洞的有效负载．

在整体实验过程中，本文选取当前流行的SQL注入模糊测试检测工具SQLMap、WFuzz、Wapiti和标准遗传算法SGA以及自适应遗传算法AGA作为比较对象．并选取Java web应用WebGoatv8.1、Vulnerable-app、和Hello-java-sec作为被测目标．

4.1.2　*算法参数设置*　遗传算法的性能与其实现过程中的各个参数密切相关，在本文实验中遗传算法的相关参数设置如表5所示．

表5　遗传算法相关参数

Tab. 5　Genetic algorithm related parameters

参数名称	设置数值
种群大小	100
变异概率	0.5
交叉概率	0.1
迭代阈值	50

4.2　遗传算法有效性验证

4.2.1　*算法性能评估*　该部分实验旨在验证基于评分机制的自适应变异策略调度对遗传算法中个体适应度和算法收敛性的影响情况．

在本部分实验中将本文遗传算法GAFuzz与标准遗传算法SGA以及自适应遗传算法AGA作对比，比较其在3个目标上平均适应度值的变化情况．算法在每个目标上的一个SQL接口上执行10次，每次的最大迭代次数为100. 在该部分实验中当且仅当迭代次数等于最大迭代次数时停止运行．然后计算每轮过程中3个目标的平均适应度值，并将其进行归一化处理．统计结果如图5所示．

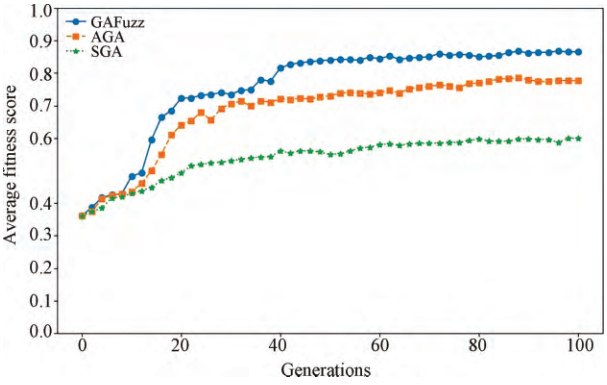

图5　算法平均适应度对比试验

FIg. 5　Comparison test of average fitness of the algorithm

从图5数据可以看到，本文提出的遗传算法在应用打分策略动态选择变异策略后，相较于随机选择变异策略的标准遗传算法和自适应遗传算法，其得到个体的平均适应度值较高，说明了本算法在模糊测试的测试用例生成上的有效性，使其更容易触发漏洞．

从曲线变化情况来看，本算法在10～20代时，其增长速度明显优于标准遗传算法SGA，略高于AGA算法，说明基于打分机制选择变异策略，使得个体更快速地倾向于向高适应度方向发展．

并且在40～50代时，本算法的平均适应度值就开始逐渐收敛．由于遗传算法是在交叉变异过程中进行随机方向的优化，因此存在收敛速度较慢的问题．自适应遗传算法AGA是通过自适应策略动态选择交叉和变异的参数来提高收敛速度．从图中数据可以看出本算法的收敛速度仍然高于AGA算法．说明在SQL注入漏洞检测场景下，自适应调度变异策略比自适应参数调度策略更利于个体向高适应度方向发展．

4.2.2　*测试用例质量评估*　在该部分实验中将

主要通过覆盖率这一指标来验证经过遗传算法优化后,测试用例的质量是否得到提高.其中覆盖率的值越大,说明优化生成的测试用例能够覆盖更多的代码指令和分支,更有可能触发漏洞.

在该部分实验中将重复运行10次遗传算法来优化生成测试用例,同时使用Jacoco分别统计这10次优化的测试用例和初始测试用例在目标应用程序WebGoatv8.1、Vulnerable-app、和Hello-java-sec中的SQL注入接口代码上的覆盖率情况,统计结果如表6所示.

表6使用字节码指令覆盖率和分支覆盖率作为衡量指标.字节码指令覆盖率表明了在所有的字节码指令中,执行指令的比例.分支覆盖率则统计了代码中if和switch分支语句的执行情况.

表6　优化测试用例覆盖率对比表

Tab. 6　Optimize the test case coverage comparison table

Goal	字节码指令覆盖率		分支覆盖率	
	初始Payload	优化后Payload	初始Payload	优化后Payload
WebGoat	82.0%	+9.5%	77.0%	+8.3%
Hello-Java-Sec	85.0%	+2.6%	83.0%	+1.7%
Vulner ableApp	87.0%	+4.2%	84.0%	+3.8%
平均	84.7%	+5.4%	81.3%	+4.6%

表6中第2列和第4列显示了初始测试用例的指令覆盖率和分支覆盖率情况,第3列和第5列则显示了优化后的测试用例相较于初始测试用例的百分比增加.相较于初始用例,优化后的测试用例在3个目标程序上的覆盖率指标均有所提高,其中在WebGoat这一较为复杂的目标上提升效果最显著,指令覆盖率提高了9.5%,分支覆盖率提高了8.3%.

这是因为Webgoat目标代码的逻辑较为复杂,包含一系列针对测试用例的过滤条件,同时在代码中也存在较多的分支语句.这种复杂性导致传统的测试用例生成方法可能无法全面覆盖所有条件和分支.然而,通过遗传算法进行优化的测试用例生成方法,能够生成覆盖上述复杂条件的测试用例.这种方法能够提高测试用例的覆盖率,特别是在处理具有大量分支和过滤条件的复杂代码时,其提升效果尤为显著.

上述结果说明经过遗传算法优化后的测试用例更具有多样性,能够覆盖更多的代码执行,增大了发现漏洞的可能性.

4.3　漏洞检测与定位能力验证

表7展示了本工具与SQLMap、WFuzz和Wapiti这3个对比工具在3个目标程序上的漏洞检测与定位情况("×"表示了可以识别的漏洞信息).表7中第2列数据为在被测目标上所测试的接口数量总数目和该靶场目标中已知的SQL注入漏洞数量.第4列数据为各个工具在各被测目标中所报告的SQL漏洞数量总数以及所报告漏洞总数中真实存在SQL注入漏洞的数量.同时,本部分使用漏洞报告的正确率P_T和误报率P_F这两个指标来说明本文方法在漏洞检测和定位能力方面的可行性.两个指标计算公式如下所示.

$$P_T=\frac{N_T}{N_G} \tag{5}$$

$$P_F=\frac{N_{\text{report}}-N_T}{N_{\text{report}}} \tag{6}$$

式中,N_T表示各漏洞检测工具所识别的真实存在SQL注入漏洞的接口数量,N_G表示各目标靶场中真实存在SQL注入漏洞的数量,N_{report}表示各漏洞检测工具所报告的SQL注入漏洞的数量.

从漏洞检测数量上来看,本工具与SQLMap工具检测能力基本持平,优于WFuzz和Wapiti工具.在WebGoat目标上,本工具相较于WFuzz和Wapiti多检测出2个漏洞.经过对相应漏洞处的源码的分析得知,这两处漏洞都采用了一定的过滤措施,分别为空格过滤和关键字过滤.这两个工具中均没有相关措施进行绕过.因此这两个工具无法通过初始测试用例触发漏洞.本工具则可以通过/**/替换空格和关键字复写的变异策略完成注入.而在SQLMap中由于其可以通过参数"--tamper=space2comment"同样指定/**/替换空格因此也可以检测到该漏洞.但是SQLMap中并没有关键字冗余策略,并且SQLMap在指定策略时需要人工手动添加,自动化程度低.

在Hello-Java-Sec和VulnerableApp目标上,本工具与SQLMap在漏洞检测能力上基本持平,

这主要是由于这两个目标程序中没有较为复杂的安全过滤措施，但是相较于WFuzz和Wapiti工具，本工具仍然可以通过遗传算法丰富初始测试用例的多样性，来达到注入的目的.

从漏洞定位信息来看，本工具在识别漏洞后能够获取与其相关的详尽数据，URL的接口和响应报文信息、漏洞触发Payload、底层函数的细节、漏洞位置的具体代码行数，以及从主函数至漏洞点的函数调用链路信息. 相比之下，其他3个工具只能给出URL接口的报文数据和触发Payload. 特别地，WFuzz在运行后只返回各个Payload传输完成后对应地响应报文的字节数和状态码，测试人员还需手动检查异常以验证SQL注入漏洞的存在性.

同时从误报率进行分析，相较于SQLMap、Wfuzz和Wapiti这些黑盒漏洞扫描工具，本文所提出的方法漏洞检测的误报率较低，表明本文通过字节码插桩技术在目标运行时对各种状态信息进行拦截，并结合这些信息作为遗传算法中是否出现漏洞的这种灰盒检测模式具有较好的性能，可以降低漏洞挖掘过程的误报率.

除此之外，与其他模糊测试工具有所不同，即使在程序未发现Web应用程序的漏洞时，本工具也可以输出遗传算法优化后地测试用例. 这有利于测试人员通过优化后的个体集以手动方式再次测试应用程序. 由于这些个体是在适应度值的引导下优化生成的，所以在一定程度上提高手动验证的效率.

表7　各工具在Java Web目标的漏洞检测与定位情况

Tab. 7　Vulnerability detection and location of various tools in Java Web targets

测试目标	测试接口数量/漏洞数量	工具名称	报告漏洞数/真实存在漏洞	漏洞定位信息					正确率	误报率
				URL信息	Payload	代码行数定位	底层函数	函数调用链		
WebGoat	52/11	本工具	13/11	×	×	×	×	×	1.000	0.154
		SQLMap	13/10	×	×				0.910	0.231
		WFuzz	15/8	×	×				0.727	0.467
		Wapiti	13/9	×	×				0.818	0.308
Hello-Java-Sec	18/5	本工具	7/5	×	×	×	×	×	1.000	0.286
		SQLMap	8/5	×	×				1.000	0.375
		WFuzz	6/3	×	×				0.800	0.500
		Wapiti	7/4	×	×				0.800	0.428
VulnerableApp	20/9	本工具	8/7	×	×	×	×	×	0.778	0.142
		SQLMap	9/7	×	×				0.778	0.222
		WFuzz	9/6	×	×				0.667	0.333
		Wapiti	9/7	×	×				0.778	0.85

5　结　论

针对于Java Web应用程序中SQL注入漏洞的模糊测试中的测试用例生成和漏洞检测与定位两大核心问题，本文结合遗传算法和字节码插桩技术，提出一种基于模糊测试的SQL注入漏洞检测与定位方法. 在该方法中，首先通过遗传算法的语义序列的划分、交叉、变异等阶段对初始Payload进行迭代生成，并在变异阶段引入绕过安全过滤措施的变异策略和基于打分机制的变异策略调度来优化遗传算法的性能. 同时基于ByteBuddy的字节码插桩技术来能够获取程序内部执行的细粒度数据，并据此数据对漏洞进行精确检测与定位. 实验结果表明，本文遗传算法的性能相较于标准遗传算法和自适应遗传算法性能更高，能更快生成适应度值较高的测试用例. 在漏洞检测方面，本文相较于其他检测工具可以检测出更多的漏洞，尤其在程序设置安全策略的情况下，效果更为明显.

由于本文所提出的基于插桩技术的灰盒漏洞检测思路仅限于Java语言开发的Web应用以及遗传算法在数据预处理和适应度计算中仅针对于SQL注入漏洞，因此本文在场景应用上具有一定的局限性. 但是通过实验验证，本文所提出的方法在WEB应用中注入类漏洞的挖掘检测过程中具有更好的性能. 未来将在本文的基础上，考虑进一步探索如何检测多种开发语言上的Web应用程序中的多种Web漏洞类型，以及增加对多种语言开

发的WEB应用的插桩检测,来提高本方法的适用性和通用性.

参考文献:

[1] CNVD. 国家信息安全漏洞共享平台[EB/OL].[2024-01-25]. https://www.cnvd.org.cn/flaw/statistic.

[2] Qian L, Zhu Z, Hu J, *et al*. Research of SQL injection attack and prevention technology [C]//2015 International Conference on Estimation, Detection and Information Fusion (ICEDIF). [S.l.]: IEEE, 2015: 303.

[3] Security Affairs. Schneider electric and siemens energy are two more victims of a MOVEit attack [EB/OL]. [2024-01-25]. https://securityaffairs.com/147865/data-breach/schneider-electric-siemens-energy-moveit.html.

[4] Manès V J M, Han H S, Han C, *et al*. The art, science, and engineering of fuzzing: A survey [J]. IEEE Transactions on Software Engineering, 2019, 47: 2312.

[5] McCall J. Genetic algorithms for modelling and optimisation [J]. Journal of computational and Applied Mathematics, 2005, 184: 205.

[6] Sirisang W, Suttichaya V. Analyzing SQL injection statements using common substructure of parse tree [C]//2017 21st International Computer Science and Engineering Conference (ICSEC). [S.l.]: IEEE, 2017: 1.

[7] Wan M, Liu K. An improved eliminating SQL injection attacks based regular expressions matching [C]//2012 International Conference on Control Engineering and Communication Technology. [S.l.]:IEEE, 2012: 210.

[8] Hlaing Z C S S, Khaing M. A detection and prevention technique on sql injection attacks [C]//2020 IEEE Conference on Computer Applications (ICCA). [S.l.]: IEEE, 2020: 1.

[9] Alghawazi M, Alghazzawi D, Alarifi S. Detection of sql injection attack using machine learning techniques: A systematic literature review [J]. Journal of Cybersecurity and Privacy, 2022, 2: 764.

[10] Li Q, Li W, Wang J, *et al*. A SQL injection detection method based on adaptive deep forest [J]. IEEE Access, 2019, 7: 145385.

[11] Gandhi N, Patel J, Sisodiya R, *et al*. A CNN-BiLSTM based approach for detection of SQL injection attacks [C]//2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE). [S.l.]: IEEE, 2021: 378.

[12] Hammersland R, Snekkenes E. Fuzz testing of web applications [J]. Retrieved, 2008, 7: 2010.

[13] Cai H, Yu X, Deng L. Research on the generation method of test cases in fuzzing [C]//2015 3rd International Conference on Mechatronics and Industrial Informatics (ICMII 2015). [S.l.]: Atlantis Press, 2015: 345.

[14] Huang M, Liu S N, Jiao X. The applied research of improved genetic algorithm in data automatic generation of software test [C]//2021 IEEE 9th International Conference on Computer Science and Network Technology (ICCSNT). [S.l.]: IEEE, 2021: 26.

[15] Lu Z G. Research on Web vulnerability mining algorithm based on improved fuzzy testing [D]. Nanning: Guangxi University, 2018. [陆紫光. 基于改进模糊测试的Web漏洞挖掘算法研究[D]. 南宁: 广西大学, 2018.]

[16] Stamparm. SQLMap[EB/OL]. [2024-01-25]. https://github.com/sqlmapproject/sqlmap.git.

[17] Xmendez. Wfuzz[EB/OL]. [2024-01-25]. https://github.com/NanoGitHub/wfuzz.git.

[18] Devl00p. WaPiti[EB/OL]. [2024-01-25]. https://github.com/wapiti-scanner/wapiti.git.

[19] Chi H Z, Kong D Z. Research on Java method enhancement technology [J]. Reliability and Environmental Testing of Electronic Products, 2022, 40: 75. [迟慧智, 孔德智. Java方法增强技术研究[J]. 电子产品可靠性与环境试验, 2022, 40: 75.]

[20] Zhang C, Zhan Z H. Comparison of genetic algorithm selection strategies [J]. Computer Engineering and Design, 2009, 30: 5471. [张琛, 詹志辉. 遗传算法选择策略比较[J]. 计算机工程与设计, 2009, 30: 5471.]

[21] Harikirank. sql-injection-payload-list[EB/OL].[2024-01-25]. https://github.com/payloadbox/sql-injection-payload-list8.

(责任编辑: 伍少梅)