分 类 号_____ 学 号 M200772017

学校代码 10487 密 级_____

硕 士 学 位 论 文

SQL 注入检测技术研究

学 位 申 请 人: 熊婧

学 科 专 业: 计算机软件与理论

指 导 教 师: 曹忠升 副教授

答 辩 日 期: 2009 年 5 月 29 日

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering

Research on SQL Injection Detection

Candidate : Xiong Jing
Major : Computer Software and Theory
Supervisor : Associate Prof. Cao Zhongsheng

Huazhong University of Science and Technology
Wuhan 430074, P. R. China
May, 2009

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期： 年 月 日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□ ，在_____年解密后适用本授权书。

本论文属于

不保密☑。

（请在以上方框内打“ √ ”）

学位论文作者签名： 指导教师签名：

日期： 年 月 日 日期： 年 月 日

摘　　要

SQL 注入攻击作为 Web 应用程序面临的一类最常见的攻击方式，对于其检测技术的研究已经受到越来越多的关注。为了解决现有基于应用的 SQL 注入检测技术容易产生误报且忽视对存储过程中 SQL 注入攻击进行检测的问题，研究了基于应用程序特征语义的检测方法和基于构造路径的存储过程 SQL 注入检测方法。

通过对应用程序应用语义所具备的特点进行分析，设计并实现了基于应用程序特征语义的检测方法。通过对特征语义的分析，提出特征模式的概念和划分方法；给出了语义模式的定义及提取过程；设计了语义模式双重简化算法以消除语意模式集中和语义模式内部存在的冗余。在充分考虑了语义模式的特点和现有不同 SQL 注入攻击行为特征的基础上，设计并实现了依次对语义模式所代表的应用语义、详细语义及语句条件语义进行检测的三层检测算法。

通过对存储过程中引起 SQL 注入攻击的原因进行分析，设计并实现了基于构造路径的存储过程 SQL 注入检测方法。通过对执行流程的分析，给出了路径树的定义及其构造过程；设计并实现了基于路径树的构造路径提取算法。设计了基于构造路径的双层检测过程。检测过程中，替换构造路径中的输入参数为用户输入值，得到最后执行的语句，依次对语句所代表的特征向量的结构和详细语义信息进行检测。

设计实现了由数据库服务器、安全增强平台、Web 服务器、客户端以及攻击模拟程序构成的实验平台。实验结果表明，在攻击模拟程序覆盖已有攻击类型的情况下，所设计的检测方法可以达到 90%以上的检测率，且产生的误报率均低于 5%；在模拟的客户端数小于 500 的情况下，增加两种检测功能后，对安全增强平台的性能所造成的影响比较小。

关键词： 注入检测，特征语义，语义模式，存储过程，构造路径

Abstract

SQL injection is one of the most popular attack methods to web applications and the researches on intrusion detection are getting more and more attentions nowadays. In this paper, a study to the detection based on application special semantic and the detection of SQL injection in procedures, based on path construction, is made, to solve the problems in existing methods based on applications from two points: false positive and the ignorance of SQL injection in database procedures.

By analyzing the unique advantages of applying semantic of applications, propose a detection method based on the special semantic of applications. Raise special patterns and measures after an analysis of application special semantic, Define semantic pattern and extracting process, Raise a double predigested algorism based on semantic pattern to eliminating the centralization and the redundancy inside of semantic patterns. On a full account of the features of semantic pattern and existing methods of SQL injection, design and implement a three-level algorithm, detecting from application semantic of semantic pattern, full semantic and statement condition semantic.

By analyzing the reason of why database procedures can lead to a SQL injection, propose a SQL injection detection method based on path constructions. By analyzing the executing processes of procedure, define the path tree and construction algorithm, design and implement tree traversal algorithm to get the construction path of executing processes. Raise a process of double-level detection based on path construction, through which replace parameters in the path with user inputs, get the final statements to be executed, detecting the structure of the special eigenvector and detailed semantic information in turn.

Design and implement the experiment platform, including backend database server, enhanced security platform, web server/client and attacking simulating program. The result indicates that when the simulating program covers all attack methods, detecting algorithm

can reach a detecting rate of over 90% , false positive is less than 5%; when the number of users in the scenario is less than 500, with two new detecting methods enabled, the performance of enhanced security platform has little influence.

Key words： SQL injection detection, special semantic, semantic pattern, stored procedure, path construction

目　　录

1 绪 论

1.1 课题背景

互联网的迅速发展使 Web 应用程序在各个不同领域得到了越来越广泛的普及。随着 Web 应用的不断深入，Web 应用程序也不断的趋于多样化和复杂化，随之而来的应用层漏洞也越来越多。这些 Web 应用程序及其后台数据库中往往都保存着对公司或组织极为重要的数据，其重要性和价值对攻击者有很大的吸引力，很容易遭受到蓄意攻击。Web 应用程序的开发者往往专注于满足用户日益增加的需求却忽视了安全性保障，即便是开发人员遵循了最安全的编码原则，飞速发展的应用程序开发工具仍然有可能使得代码存在漏洞，因此，基于数据库的 Web 应用系统所面临的安全问题日益突出。

SQL 注入攻击（SQL Injection Attack）[1]是 Web 应用程序所面临的一类最为常见的攻击。攻击者通过修改应用程序的 Web 表单的输入域或页面请求的查询字符串等输入参数插入一系列的 SQL 命令来改变数据库查询语句，从而欺骗服务器执行恶意的 SQL 命令，实现对后台数据库未经授权的访问。

通过 SQL 注入技术来实现攻击的入侵者往往都已经绕过了基于主机（Host-based）和网络（Network-based）的入侵检测系统获得了合法身份，因此，仅仅依靠工作在文件和系统命令级的底层主机和网络入侵检测技术[2,3]是很难检测到这一类攻击者的非法行为，检测的精度无法得到保证，因而，利用应用系统的独特语义特征检测应用入侵中的 SQL 注入攻击行为的研究在近年来得到重视。所谓的应用入侵检测（Application Intrusion Detection）就是使用应用语义来检测更细微的异常行为，特别是检测内部用户稍微偏离正常行为的滥用[4]，SQL 注入攻击行为即属于这种细微的异常行为。又由于现有的应用系统如数据库管理系统（Database Management System，DBMS）等，大多具有自己的数据结构、独特的应用语义和固定的使用方式，因此，利用应用程序自身特征来进行 SQL 注入攻击检测，以保障应用系统的安全是十分可行的。

基于 DBMS 的入侵检测技术是当前基于应用的入侵检测领域中研究得较为成熟的技术之一。现有的基于 DBMS 的检测技术中，针对 SQL 注入攻击进行检测的方案

主要是结合模式识别、数据挖掘和关联规则等技术，通过挖掘用户查询频繁项等途径来实现的。这些技术的发展在一定程度上提高了 SQL 注入检测的效率，但都仅仅考虑了 DBMS 自身所具备的特征，而没有与应用系统自身所具备的应用语义相关联，存在着较大的误报率和漏报率；同时对于能够检测到的 SQL 注入攻击类型也存在着一定的限制，例如若用户对应用程序的操作，对应在数据库中为存储过程调用语句时，是有可能存在 SQL 注入漏洞的，但是现有的检测方案中对此几乎都没有提及，甚至认为使用存储过程是可以防范 SQL 注入攻击的。因此，可以看出，在基于应用的入侵检测领域，对 SQL 注入检测技术的研究不仅有极大的实用价值，还存在着很大的发展空间。

本课题来源于华中科技大学数据库与多媒体技术研究所承担的 863 计划项目“高安全等级数据库管理系统及其测评关键技术研究”课题，课题编号为 2006AA01Z430。作为该课题一个重要的有机组成部分，旨在深入研究基于应用的入侵检测领域的 SQL 注入攻击检测技术。

1.2 国内外概况

1.2.1 基于应用的入侵检测技术的发展

从 1980 年 Anderson 首次提出入侵检测（Intrusion Detection）[5]的概念至今，国内外对入侵检测技术进行了广泛深入的研究。入侵检测主要有两种模式，即通过检测用户行为是否偏离正常模式的异常检测（Anomaly Detection），以及通过检测用户行为是否符合某个已知攻击模式的误用检测（Misuse Detection）。误用检测必须预先建立入侵模式库，检测阶段对审计数据进行分析检查是否包含入侵标识，又由于现今攻击形式趋于多样性，模式库难以达到应有的完备性，因此其检测精度受到极大的限制。而异常检测在训练阶段建立系统的正常特征轮廓，检测阶段通过用户活动与正常轮廓间的偏离程度来判断异常行为，能够检测出来新的攻击，具有更大的实用性和研究价值，因此大多数的研究工作都将重点放在对异常检测技术的研究上。

异常检测领域的研究成果大多都集中在基于主机和网络的入侵检测系统上。主机入侵检测系统（Host-based Intrusion Detection System，HIDS）用于审计用户的活动，

如用户的登陆、命令操作和资源使用等，典型的系统如 Haystack[6]，Denning 的入侵检测模型[7]等。网络入侵检测系统（Network-based Intrusion Detection System，NIDS）用户审计用户的网络活动，如对网络流量、协议分析、网络管理等数据的分析来检测入侵，典型的系统包括 Snort[8]、Bro[9]、NFR[10]等。按照技术发展的时间顺序，HIDS 和 NIDS 的异常检测主要实现技术有：统计分析、专家系统、神经网络、计算机免疫系统和数据挖掘等。

近年来，随着攻击形式的多样化，对于应用系统来说，HIDS 和 NIDS 已经无法保证检测的精度和效率，例如只能通过应用系统本身的语义和结构特点加以检测的内部权限滥用行为，HIDS 和 NIDS 都是无能为力的。因而在这种情况下，通过利用应用系统的独特语义特征检测入侵的基于应用的入侵检测技术，由于能够在一定程度上弥补 HIDS 和 NIDS 所存在的不足而得到了重视，并在近年来取得了较大的发展。当前基于应用的检测技术大多集中于一些实际应用情况中所遭受攻击的可能性比较大的领域，主要有以下一些研究领域。

1．基于 DBMS 的检测技术

基于 DBMS 的入侵检测是基于应用的入侵检测领域中开展得最早的领域，早期的数据库入侵检测技术主要是针对数据库推理[11]和存储篡改[12]的检测，现今的研究主要集中在基于数据库事务、用户查询模式等检测方法上。由于基于 DBMS 的检测技术均涉及到了对于 SQL 注入攻击的检测，因此后面的部分将进行详细论述，在此不再赘述。

2．基于 Web 服务器和应用程序的检测技术

互联网的发展使得 Web 服务器和基于 Web 的应用程序逐渐成为新的入侵目标。例如利用 Web 服务器的 CGI（Common Gateway Interface）或 ASP（Active Server Pagea）程序漏洞的入侵、利用 Web 程序漏洞的蠕虫等。Almgren 等提出了与 Apache Web Server 集成的用来检测对 Web 服务器的欺骗入侵等的应用模块[13]；Tatyana 等提出了在 Apache Web Server 中实现的集存取控制和入侵检测于一体的 API[14]。

3．基于其他应用软件和通用方法的检测技术

如基于电子邮件服务器的应用检测系统[15]，主要是使用基于内容的检测技术实现

对垃圾邮件的检测；基于应用调用模式的检测技术，即通过系统编程语言库调用序列进行应用检测[16]；以及可与多个应用系统紧耦合的应用入侵检测接口[17]等。

4．基于中间件的检测技术

文献[18]提出了布置在中间件（Middleware）中的应用系统防卫机制，以及使用适应性中间件进行安全防卫的应用系统结构。

5．基于应用语义的检测技术

在许多场合中，独立于应用语义对数据库事务或用户进行检测并不足以识别用户的异常行为。如某会计非法将自己的月工资增加一万元，独立于应用语义上的检测方法如对表存取统计、数据文件存取统计、会话统计或上述的各种检测方法都无法发现异常，这种异常检测只能建立在数据库的应用语义上。

Sielken 提出了基于应用语义的入侵检测思想[19]，并列举了基于应用语义限制和统计的例子。应用语义限制（如医生只能查看他所治疗病人的病历，医生开出的处方只能是他专业范围内的）构成基于规则的异常检测系统；应用语义统计（如病人服某种药的次数和剂量应与其它相同的处方之间有一定的相似处，病人购药的订单应大多数发生在白天上班时间等）构成基于统计的异常检测系统。

当前某些特定领域存在着较为成熟的利用应用语义来检测异常的方法，例如在金融信息系统[20]和通讯信息系统[21]中，对基于应用语义的检测方法有较为深入的研究，但在入侵检测领域却仍处于基础研究阶段，而由于应用语义的独特性和精确性，基于应用语义的检测方法可以有效提高入侵检测的准确性和粒度，因此，基于应用语义的检测方法将是基于应用的入侵检测领域比较重要的研究方向之一。

在上述各个不同领域的检测技术发展的同时，由于意识到了某些特定攻击手段所带来的极大危害性，一些研究工作很早就将研究重点转移到对这些攻击行为进行检测的技术上，例如 SQL 注入攻击和缓冲区溢出（Buffer Overflow）等，也因此取得了一定的研究成果。由于本文主要研究的是 SQL 注入检测技术，因此本文后续部分首先对基于应用的入侵检测领域中，SQL 注入检测技术的研究与发展进行总结，并在此基础上，提出本文的研究思路以及课题主要研究工作。

1.2.2 基于应用的 SQL 注入检测技术现状

基于应用的入侵检测领域中的 SQL 注入检测技术，从实现技术上来讲，大致可以分为两类。其一是基于应用程序本身的检测技术，主要从分析应用程序的源代码入手，展开进一步研究；其二是基于 DBMS 的检测技术，主要是从分析应用程序在数据库中的操作日志入手进行探讨。下面分别对这两类检测技术进行详细的探讨。

1．基于应用程序本身的 SQL 注入检测技术

（1）用户输入的有效性检查

早期工作都把对用户输入进行有效性检查作为研究重点，最简单的技术是在检查过程中消除用户输入中的单引号和破折号[22]，但是攻击者只要构造一些可以躲避检查的输入就可以很轻易的实现攻击目的，因此编码人员必须对用户输入之后的代码进行进一步的分析；当前主流的 Web 应用程序框架大多也是通过检验用户的每个输入值是否符合参数的预定义类型来预防 SQL 注入，如 Structs 的验证器（Validator）。但是，只有当验证器是禁止用户的输入值中包含有元字符时，该验证器才可以使应用程序避免受到 SQL 注入攻击。因此，对输入值的有效性检查只能算是一种最基本的辅助手段。

（2）使用 Prepare Statement 方式执行 SQL 语句

SQL 语法允许通过使用 Prepare Statement 的形式执行 SQL 语句，即可以将查询中的输入值与该查询语句的结构分离开来[23]。开发者预先设计好 SQL 语句的结构，在运行时将用户输入值直接填充到该结构中。SQL 语句的 Prepare Statement 执行方式增加了 SQL 注入的难度，因为该语句的结构是不允许被改变的。Hibernate 强制用户使用 prepare 的方式执行 SQL 语句。但是，如果要使用该方式来执行 SQL 语句，就必须重写（Rewrite）应用程序源代码，由此所带来的工作量不可忽视。

（3）静态分析

文献[24]提出了一种基于 Java 源程序的字符串处理方法，该方法根据源程序的控制流图（Control Flow Graph，CFG）来构建模型用以模拟字符串的生成过程。当前大多数的代码审查技术[25]都基于该静态分析方法。Wassermann 等提出了一种静态分析和自动推理结合的方法对添加了输入值后的 SQL 语句进行检查，但该方法仅仅只能检查

出来利用重言式的注入攻击形式，对其他的 SQL 注入攻击形式都无能为力[26]。JDBC checker[27,28]主要是通过对动态生成的 SQL 语句进行类型正确性检查来判断是否存在注入现象，但是它无法检测出那些句法结构和语句类型都正确的常用 SQL 注入类型，例如带有“or 1=1”、“union”的注入类型。

（4）动态检查

Paros 是一款不需要重写 Web 应用程序脚本文件的漏洞扫描工具[29]。它根据预先定义好的攻击代码对脚本中存在 SQL 注入威胁的代码处进行试探，然后通过检查 HTTP 回应报文的内容来判断是否发生 SQL 注入攻击，如果确定了攻击产生则可以认定漏洞的存在。文献[30]使用相同的方法用以检查应用程序源代码中存在的注入漏洞。

（5）静态分析和运行时监控的结合

部分研究工作提出了基于模型的静态分析和运行时监控的 SQL 注入检测技术[31-34]，静态分析阶段通过分析用户输入前的查询语句结构来建立各种不同的模型，运行时检查添加了用户输入值后所形成的查询语句，通过其是否与已有模型相匹配来判定注入攻击的发生。

文献[31]通过一种上下文无关语言来描述在静态分析阶段所提出的模型，该模型只能接受合法的查询语句。由于该模型使用了一个秘密关键码（Secret Key）来标识用户输入值在语句中的位置，因此其安全性的保障必须以攻击者无法得知这个关键码为前提，同时还要求应用程序开发者通过重写代码将关键码添加到动态构造的 SQL 语句中。AMNESIA[32,33]在其静态分析阶段对所有正常的数据库操作语句进行建模，动态阶段截获发往数据库的 SQL 语句，检查是否存在与其对应的模型，若不存在则认定为攻击行为。Buehrer 等提出了一种树形结构的分析模型[34]，基于该模型的检测方法要求应用程序开发者使用特定的中间语言库来重写代码。

（6）其他

部分研究工作提出了基于机器学习（Machine Learning）的检测技术[35,36]，存在的问题大多都是由于学习阶段训练集的不完整导致的大量误报和漏报。另外还有最近提出的为预防 SQL 注入攻击的自动化代码生成技术[37,38]，但目前处于基础研究阶段。

综合上述可以看出，基于应用程序本身的检测技术极大程度上都依赖于静态分析

的精确度，而应用的发展使得应用程序的编码形式越来越复杂，静态分析的难度会越来越大，且某些特定类型的模糊代码或者是查询生成技术也会极大的降低静态分析阶段的精确度，此为其一；其二，大多数的 SQL 注入攻击来自于内部用户稍微偏离合法行为的滥用，所执行的语句都属于正常操作语句，且与应用程序本身的语义功能相关，而仅对 SQL 语句进行结构上的分析是无法检测出来这种细微的滥用行为的，因此对其他应用入侵检测技术展开研究是必然的趋势。

2．基于 DBMS 的 SQL 注入检测技术

借助于主机和网络入侵检测领域已有的较为成熟的检测技术，数据库入侵检测领域虽然起步较晚，但是却取得了较快的发展。

国内数据库入侵检测领域的研究主要是基于聚类分析、分类分析、关联分析、序列模式分析等一些数据挖掘方法[39-45]，国外的研究工作主要可以分为以下 5 类。

（1）基于数据关系和结构

数据库具有自己独特的数据结构和数据关系，有自己的事务处理机制，利用数据库中数据关系和结构的特点能够形成数据库独有的入侵检测方法。

Y Hu 等提出利用数据读写之间的依赖关系来检测恶意事务[46,47]。在数据库事务中，对某个数据更新前必须读写某些数据集，在数据更新后也必须写某些数据，由此形成数据间的读集（Read-Set）、预写集（Pre-Write Set）、后写集（Post-Write Set）之间的依赖关系，不符合这些关系的事务被认为是恶意事务。由于数据库中数据关系的复杂性，该方法在实现上存在性能和效率问题。

（2）基于用户查询数据项

Christina 等在其原型系统 DEMIDS（Detection of Misuse in Database Systems）中提出使用用户轮廓（User Profiles）检测数据库用户的误用行为，用户轮廓是在关系数据库中用户和角色的典型行为[48]。DEMIDS 重点在于检测合法用户的滥用行为，其使用频繁项（Frequent Itemsets）来表示用户的正常行为，频繁项是用户参考的属性及其值，频繁项是按照数据结构、模式语义（主键、外键依赖）和用户日志使用一定的挖掘算法挖掘出来的。在挖掘用户的频繁项算法中，DEMIDS 使用了距离的概念来测量属性出现在同一操作语句中的可能性，距离的度量使用了用户查询语句中的主键和外

键的函数依赖关系，融入了 DBMS 的数据结构特性和用户行为的语义特性，可以精确地描述用户对数据库进行操作的行为模式。

Christina 等的改进算法将可以体现特定领域知识的概念体系应用到用户轮廓的挖掘中，从而可以形成不同抽象级和粒度的用户轮廓，并使用兴趣度（Interestingness Measure）来发现用户的行为模式[49]。但该方法假设合法的用户使用数据库的方式有一定程度的一致性，如果这个假设不成立或者检测阈值配置不当，将会导致较高的误警率。

Bertino 等在 Christina 的基础之上提出了基于用户角色轮廓的实时入侵检测机制以及三种不同精度的数据定义[50]。

（3）基于用户查询语句

数据库具有自己独特的事务处理机制和 SQL 语言查询，对用户使用 SQL 语句的模式进行检测是数据库入侵检测的一项重要内容。

指印（Fingerprints）技术[51,52]是一种基于 SQL 语句的入侵检测方法，指印是从合法事务中的 SQL 语句中推出的正则表达式，代表用户正常的行为，用户的事务语句如果偏离指印集则表示可能的异常行为。指印技术特别适应类似于对互联网上的数据库入侵检测，比如 SQL 注入，因为在这些应用中往往使用数据库应用来查询数据库，而这些应用只通过一定接口使用固定的几种查询格式，不允许用户自定义查询，在这种情况下，即使事务较大、用户较多，误检率也较低。使用事务语句作为用户查询模式的优势在于检测阶段可以将用户查询在执行前与用户模式对比，以避免恶意查询的执行影响系统正常运行或造成不可挽回的后果。缺点在于易产生过多的用户查询模式，如指印技术易产生过多的“指印”。

Y Zhong 等的方法通过计算用户查询语句与用户模式的偏离度来确定查询的异常度，由于偏离度计算公式存有一定的误差，该方法也易造成较高的误报率[53]。

（4）基于数据库事务

一个特定的数据库应用系统所能实现的功能是有限的，其一项功能的完成往往对应了数据库中一个或多个事务的执行，又由于特定的数据库事务一定具备了特定的操作轨迹，基于事务级的检测主要是通过记录应用程序在一个事务中的执行轨迹，生成

合法的轨迹集合，用户执行操作顺序必须遵循集合中该事务的操作顺序，任何不符合该操作顺序的用户事务将被认为是非法的。

Vieira 等提出了一种基于此思想的事务级检测方案[54]，该方法建立在 IDS 能够事先获得应用程序的事务轮廓的基础上，可以有效地检测恶意事务但是无法检测出现在事务内部的常规 SQL 注入攻击类型。Fonseca 等在此基础上提出了一个针对 Web 应用程序的入侵检测系统模型[55]，该模型从指令和事务两个层次实现了对多种 SQL 注入攻击行为的检测。该方法在指令级通过一系列字符串模型对 SQL 语句进行检测，由于对所有字符串变量建模的困难性，它增加了误报率；其事务级检测分为两个阶段，学习阶段收集每个事务内部的 SQL 语句执行轨迹形成模板，检测阶段通过对事务模板的匹配来检测恶意事务，但它并没有考虑到应用程序的事务之间以及事务内部中存在的重复性，因此导致模板集合中存在大量冗余且模板长度过长，对系统效率产生了较大的影响。综合前面的研究成果，Vieira 等对数据库入侵检测技术进行了综合阐述[56]；同时设计并实现了一种针对于 SQL 注入以及 XSS 攻击的 Web 应用程序漏洞探测工具[57]。

（5）其它。以上 5 种检测技术对 SQL 注入攻击的检测都只限于对一般的 SQL 操作语句中的攻击行为的检测，而均未提及对利用数据库存储过程存在的 SQL 注入漏洞来实现攻击的行为的检测。

K Wei 等提出了一种对存储过程中 SQL 注入攻击行为的检测方法[58]，该方法通过对存储过程中的每一个执行参数都生成一个有穷自动机来表示该执行参数所形成语句的结构，将用户输入值作为有穷自动机的输入状态集合，根据添加用户输入值前后有穷自动机最后所生成语句的结构判断是否存在注入的情况。该方法能够检测出的攻击类型非常有限，对于通过使用 union 增加一个查询语句等 SQL 注入类型，会产生漏报；其次，该方法对于定义中存在较多流控制结构的存储过程，该方法会产生误报。

综合上述可以看出，首先，从实现技术角度来说，现有的基于 DBMS 的应用检测技术中，大多都是从 DBMS 自身所具备的特征出发进行考虑的，并没有与应用系统所具备的应用语义相关联，误报率较大，而且在检测率和效率上也有待改进；其次，从能够检测到的 SQL 注入攻击类型来说，大多数检测技术都着重于对一般 SQL 语句中存在的攻击行为进行检测，而忽视了对于存储过程引起的 SQL 注入攻击行为的检测。

1.3 课题主要研究工作

从课题背景可以看出，对基于应用的 SQL 注入检测技术进行研究是有必要的。由国内外概况可知，从实现技术角度上考虑，在数据库层的检测技术中，很少有工作考虑到与应用程序本身具备的应用语义相关联，又因为基于应用语义的检测实际是具有自身独特优势的，所以结合基于数据库层的检测技术和基于应用语义的方法进行研究是具有可行性的；从能够检测到的攻击类型来说，对于存储过程中出现的 SQL 注入攻击检测尚只有一篇研究工作提及，且存在着较大的误报和漏报。因此，本文从上述两个角度入手，分别对基于应用语义的数据库层检测技术和针对于存储过程中 SQL 注入攻击的检测技术进行深入研究。根据上面的分析，本课题主要解决的问题如下。

1．基于应用程序特征语义的检测方法的设计与实现

通过对应用程序应用语义所具备的独特优势的分析，提出基于应用程序特征语义的检测方法，该方法工作在学习和检测两个阶段。解决的问题依次为：定义及划分特征模式的问题，定义及转化语义模式的问题，语义模式集的简化处理思路以及合理的学习算法的选取，基于特征语义的三层检测方法的处理过程。

2．基于构造路径的 SQL 注入检测方法的设计与实现

提出基于构造路径的存储过程 SQL 注入检测方法，该方法工作在学习和检测两个阶段。解决的问题依次为：存储过程可注入性判定问题，可执行参数集的简化处理思路，Path-Tree 的构造算法以及基于 Path-Tree 的构造路径提取算法的设计与实现；基于构造路径的检测方法的处理过程。

3．实验平台的设计与实现

通过对实验平台所必需具备的条件展开分析，提出合理的实验平台结构；分别对两种检测方法进行详细的实验设计；以及进行结果分析。

2　基于特征语义的 SQL 注入检测

本章主要研究了由学习和检测阶段组成的基于特征语义的 SQL 注入检测方法。首先，在问题分析的基础上阐述了该方法的研究思路；然后，从学习阶段入手，给出了语义模式的提取算法，并提出了对该算法进行简化的思路和过程；最后，给出了检测阶段中三层检测算法及其实现流程。

2.1　问题分析

应用系统往往都具有特定的应用功能，同时也会提供给用户规范的使用说明，或者会对不同用户的行为存在不同的限制，诸如此类的信息即为应用系统的应用语义。应用系统的应用语义往往是入侵者难以掌握的，独立于应用系统应用语义对数据库事务和用户进行检测很难能发现正常用户的滥用行为。比如恶意用户绕过数据库应用系统直接操作数据库服务器，他在服务器中的操作符合其所拥有的权限，但是却不符合数据库应用系统的应用语义，这种恶意行为如果撇开应用系统的应用语义，将不会被检测出来。

根据本文 1.2.1 节的描述，虽然 Sielken 已经提出在入侵检测中使用应用语义并列举了基于应用语义限制和统计的例子[19]，但并没有提及对 SQL 注入攻击的检测。因此，将应用语义的概念引入到对 SQL 注入检测技术的研究中具备了一定的可行性。

本文所提出的检测方法工作在学习和检测两个阶段。

通过对用户使用应用程序的行为特点进行分析，提出应用程序特征语义的概念用以描述应用程序的功能特征，不同的功能对应着不同的特征语义；且将其形式化描述为特征模式，并提出基于用户任务的特征模式划分方法；而为了更精确的描述特征语义的概念，给出特征模式到语义模式的转化过程；通过分析初始语义模式集中存在的冗余问题，给出语义模式集的双重简化处理过程。为了使得到的语义模式集能够尽可能完整的表示应用程序的所有功能，采用增量学习的方法。

提取正常语义模式与待检测语义模式进行匹配时，通常意义上的匹配过程是直接

进行全模式的匹配，当被检测的数据量较大时，这样的全模式匹配对效率造成的影响比较大，同时由于不同类的攻击行为的特性各异，对于某些攻击性比较明显的行为，是不需要进行全模式匹配的，只有对隐藏得较为深的攻击行为，才有必要通过全模式匹配进行检测。针对这种通用性方法存在的问题，设计一种三层检测算法，只有当上层的检测通过后才进行下层检测，否则可以直接断定出现 SQL 注入行为，当所有层次的检测均通过后，可以认定该行为不具备攻击性，因此可以保证不同的攻击行为在不同的层次被检测出来，充分考虑各种攻击行为自身的显著特点，不仅可以有效的提高检测的效率，同时可以为检测后的恶意等级评定提供依据。同时，对于一般检测方法无法检测出来的,特征非常细微的攻击行为能够在最后一层的检测过程中被检测出来，因此在能够检测的类型方面，所提出的检测方法亦优于同类的工作。

根据前面所论述的本章所要解决的问题，本章后面的小节将给出深入的分析。第二节主要分析学习阶段所涉及到的关键技术，分别从特征语义的定义和划分、语义模式的提取及压缩三个方面展开论述；第三节主要介绍所提出的多层检测算法，首先介绍算法的核心流程，然后分小节对每层检测技术进行详细的探讨；最后一节对本章所提出的基于特征语义的检测方法优于同类的研究工作之处进行总结，同时指出所存在的不足之处，提出之后的研究方向。

2.2 基于特征模式的模式提取

本节深入探讨学习阶段所涉及到的关键技术。首先，通过分析用户使用应用程序的行为特点，提出应用程序特征语义的概念，在此基础上给出基于特征语义的特征模式的划分方法；然后，给出语义模式的定义和提取过程；最后，通过分析语义模式提取过程中存在的冗余问题，提出简化的思路和过程。

2.2.1 特征模式定义及划分

数据库应用系统是完成一系列维护，更新和查找数据操作的应用系统。每一个应用系统都围绕一个应用主题（比如酒店管理，商品库存）提供了一些相关的功能与使

用规范，这些功能及其对应的使用规范即为数据库应用系统的应用语义。而站在用户的角度，它们都是为了使用户能够实现系统功能而服务的，因此可以认为应用系统的应用语义与其实际功能是存在一定对应关系的。

当用户完成应用系统的某一项功能时，可以认为是用户完成了应用程序的某一项任务。对应在数据库中，相当于执行了一系列的 SQL 语句，这里称之为一个操作集合，因此这个操作集合是可以反映出系统的某一项功能特征的。且对于一个应用系统来说，一旦建立，其所具备的功能在一段时间内可以视作是稳定无变化的，不同应用系统所具备的功能无论是在数量上还是特征上都是不一样的，这里我们用应用系统特征语义的概念来描述应用系统的功能特征。

为了更好的描述后续概念，这里我们首先将数据库中单个 SQL 操作形式化描述为特征向量。

定义 2.1： 五元组$V<T,O,P,A,C>$来描述每个 SQL 语句所代表的特征信息，称之为特征向量，其中：T 为操作类型集合（select、update、delete、insert），O 为关系对象集合，P 为谓词集合，A 为选择字段集合，C 为约束字段集合。

由定义 2.1 可知，每一个 SQL 操作都可以表示成特征向量的形式，对于结构较简单的 SQL 语句，其特征向量中的操作类型集合大多都只包含一个元素，而对于较复杂的 SQL 语句，例如嵌套查询和相关子查询，其操作类型集合中至少都包含了两个或两个以上的元素。

定义 2.2： 给定一个特征模式$Q<QID,V_1,V_2,\cdots\cdots,V_m>$用来描述应用系统中的某一种特征语义，其中 QID 为该特征模式的唯一标识，$V_i(1\le i\le m)$为该特征模式所包含的具有一定顺序的特征向量序列。

由定义 2.2 可知，一个特征模式都对应着一种特征语义，即可以表示应用系统的某一项功能。由于每个特征模式中包含着多个特征向量，定义 2.3 给出了特征模式中各个组成成分之间的关系。

定义 2.3： 对于特征模式$Q<QID,V_1,V_2,\cdots\cdots,V_m>$，已知 QID 唯一标识一个特征模式 Q，存在 QID 到特征向量序列$<V_1,V_2,\cdots\cdots,V_m>$的一个映射，因此可以定义函数

$$f:QID\rightarrow\{T_1T_2\cdots\cdots T_m\}$$

其中，$T_i \in V_i(1 \le i \le m), V_i \in < V_1, V_2, \cdots\cdots, V_m >, < V_1, V_2, \cdots\cdots, V_m > \subset Q$。以及函数

$$g(QID, i) = y, y =< O_i, P_i, A_i, C_i >$$

其中，$O_i \in V_i(1 \le i \le m), P_i \in V_i(1 \le i \le m), A_i \in V_i(1 \le i \le m), C_i \in V_i(1 \le i \le m)$，

$V_i \in < V_1, V_2, \cdots\cdots, V_m >, < V_1, V_2, \cdots\cdots, V_m > \subset Q$。

由定义 2.3 可以看出任意一个特征模式与它所包含的特征向量各组织成分之间的关系。函数 f 表示任意一个特征模式，都存在与其相对应的操作类型集合，该操作类型集合$\{T_1 T_2 \cdots\cdots T_m\}$即是此特征模式所包含的所有特征向量中的操作类型集合的集合。函数 g 表示的是根据特征模式的标识 QID 以及其中特征向量的序列号可以得到该序列号所代表的特征向量的详细信息。

根据定义 2.3 的描述，某个应用系统的全部功能所代表的特征模式可描述为有限集合 S，$S = \{Q_1, Q_2 \cdots\cdots Q_m\}$，其子集$Q_i$即对应系统的某一功能描述，称 S 为该应用系统的特征模式集。又由于特定应用系统的特征语义是有限的，且不同的功能对应不同的特征语义，因此 S 中的特征模式应当是有限且不存在交叉冗余。

集合 S 的子集Q_i对应着应用系统某一项特征语义，即某一具体的功能，也就相当于我们所划分的一个操作集合，因此若要得到能够精确描述应用程序特征语义的特征模式，就必须保证其对应的操作集合必须能够尽可能准确的代表应用程序的某一项功能。根据本节前面的论述，用户执行某一项功能相当于完成一项任务，对应在数据库中的操作集合往往对应着一个或多个事务的连续执行，因此，若是将完成同一任务的事务序列都归结到一个操作集合之中，就能够将应用系统的不同功能对应的操作集合区分开，所以，必须使用一定的方法对操作日志中的事务按照用户任务来分组，每一组事务共同完成一个用户任务，它对应的就是用户所完成的功能。

通常情况下，在机器负载不是太重时，两个用户任务之间的时间间隔比一个用户任务中执行两个事务的时间间隔大，如图 2.1 所示。

由于数据库操作日志中记录了每个事务的开始和提交时间，因此可以根据上一个事务的结束和下一个事务开始的时间来计算两个事务之间的间隔。这里我们根据给定的一个阈值来代表两个任务的间隔，当两个事务之间的间隔比给定阈值大的时候，认为第一个事务是一个用户任务的结束，下一个事务不属于同一个用户任务。如果阈值

选取不适当，会出现同一任务中所执行的事务被切割到两个操作集合中的情况，因此只有在阈值恰当的情况下，所划分出的操作集合才能够比较准确的描述用户所执行的各个任务，可见阈值的合适选取是很重要的。

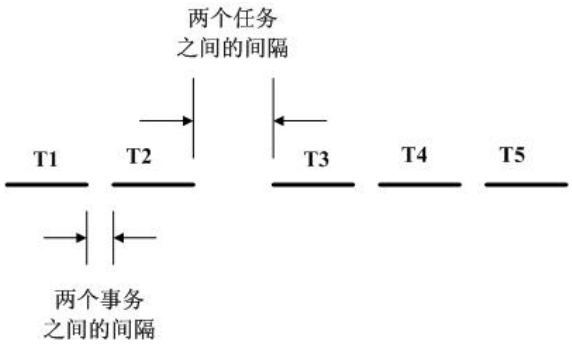


图 2.1 时间间隔图

2.2.2 语义模式提取

由 2.2.1 节论述可知特征模式可以准确的反映应用系统的语义功能，而由定义 2.3 可知特征模式与其所包含的特征向量存在一定的映射关系，而特征模式的定义却无法很好的反映出定义中所描述的这种关联性，由此本节提出语义模式的概念。

语义模式是以特征模式为基础上提取出来，不仅可以准确的反映出应用程序的特征语义，同时也包含了特征模式内部存在的各种映射关系，能够精确描述出应用程序特征语义的特征。

定义 2.4： 语义模式定义为$W < QID, Rf, Rg, M >$，$Rf = f(QID) = \{T_1 T_2 T_3 \cdots T_M\}$称为操作类型序列集合，$Rg = \{g(QID, i) \mid 1 \le i \le M\}$称为详细语义信息集合，其中 QID 唯一标识该语义模式，M 为该语义模式的长度，函数 f，g 定义如定义 2.3 中描述。

由定义 2.4 可以看出所定义的语义模式是由三个部分组成，QID 为所生成的语义模式的标识，同时对应着应用系统的一个特征模式，M 为该特征模式所包含的特征向量的长度，称之为该语义模式的长度，Rf 是由特征模式号为 QID 的特征模式所包含的所有特征向量中的操作类型集合所构成的序列，Rg 为一个集合，集合的大小为特征模式号为 QID 的特征模式所包含的特征向量的数目，根据定义 2.3 可以知道集合中每一个元素的内容。

定义 2.5： 语义模式集定义 $WT=\{W_1,W_2,W_3,\cdots\cdots,W_k\}$，其中，$W_i$ 表示为 $<QID_i,Rf_i,Rg_i>$，即为应用系统某项特征语义所对应的语义模式，k 为该应用程序的功能数目。

定义 2.5 给出了学习阶段最终所得到的应用程序全部语义模式的描述，后续的检测阶段便可以提取语义模式集中的语义模式作为规则进行匹配，进而发现异常情况。

综合前面的描述，提取语义模式的过程是首先根据基于用户任务的划分方法对应用程序在数据库中的操作日志进行操作集合的划分，将操作集合表示为特征模式，最终转化为语义模式的形式，得到语义模式集。

2.2.3 语义模式约简

对于一个特定的应用系统，它所具有的功能是有限的，因此，同一用户完成不同的功能所执行的操作集合中，有可能会存在相同的事务，且不同的用户也可能会使用到相同的功能，所以从数据库日志所提取出来的语义模式集以及语义模式内部都存在着冗余的问题。

例如一个商品销售系统，不同的用户在一次操作过程中都执行了一次购买功能，那么无论他们的购买商品的品种或者数量是否一样，他们此次操作对应的都是相同的特征语义，因此，在没有经过任何简化处理的情况下，所得到的初始语义模式集中便会存在相同的语义模式，即语义模式间存在冗余。又例如用户在一次操作过程中连续执行了多次购买功能以及在一次操作过程中只执行一次购买功能的情况，这两种情况本质上对应的都是购买功能，按照所提的特征语义的概念，应该将其归结为同一类特征语义，但是因为基于用户任务的划分方法在虽然能够准确的划分不同的特征语义，却并不具备语义识别的能力，所以无法分辨这种本质上虽然一致但表现形式存在较大差异的情况，我们称这种情况为语义模式内部存在冗余。

为了解决上述语义模式间及语义模式内部存在的冗余问题，本节给出语义模式集的双重简化算法。首先，通过对语义模式进行等价判定，给出消除语义模式间冗余的方法；然后，提出可压缩的语义模式的概念，并在此基础上给出消除语义模式内部冗余的方法；最后，给出语义模式约简算法的形式化描述，并给出其时间复杂度分析。

1．消除语义模式间冗余

当语义模式集中存在大量的冗余时，在检测阶段若扫描语义模式集匹配模式时，会进行多次不必要的匹配，对检测效率的影响很大，为了避免这种重复匹配的过程，提出了一种解决语义模式集中存在的冗余语义模式问题的方法。首先给出语义模式间的等价定义，随后给出根据等价定义消除冗余的步骤。假设 WT 根据前面的方法所提取出来的，某一应用程序所对应的语义模式集。

定义 2.6： 对于 $\forall W_i < QID_i, Rf_i, Rg_i, M_i >$，$W_j < QID_j, Rf_j, Rg_j, M_j > \in WT$，其中，$Rf_i = f(QID_i) = \{T_{1i}T_{2i}T_{3i}\cdots T_{Mi}\}$、$Rg_i = \{g(QID_i, k) \mid 1 \le k \le M_i\} = \{O_{ik}, P_{ik}, A_{ik}, C_{ik}\}$，$Rf_j = f(QID_j) = \{T_{1j}T_{2j}T_{3j}\cdots T_{Mj}\}$、$Rg_j = \{g(QID_j, k) \mid 1 \le k \le M_j\} = \{O_{jk}, P_{jk}, A_{jk}, C_{jk}\}$，则 $Rf_i \equiv Rf_j$ 当且仅当

$$i \ne j，\ M_i = M_j 且 \forall k \in (0, M_i) 或 (0, M_j) : T_{ki} = T_{kj}；以及$$

$Rg_i \equiv Rg_j$ 当且仅当

$$i \ne j，\ M_i = M_j 且 \forall k \in (0, M_i) 或 (0, M_j) : O_{ik} = O_{jk}, A_{ik} = A_{jk}, P_{ik} = P_{jk}, C_{ik} = C_{jk}。$$

定义 2.6 给出了不同语义模式中的组成成分即操作类型序列集合 Rf 和详细语义信息部分 Rg 的等价定义，以此为基础给出语义模式等价的定义。

定义 2.7： 对于 $\forall W_i < QID_i, Rf_i, Rg_i, M_i >$，$W_j < QID_j, Rf_j, Rg_j, M_j > \in WT$，$W_i$ 与 W_j 等价当且仅当 $Rf_i \equiv Rf_j$ 且 $Rg_i \equiv Rg_j$，记作 $W_i \equiv W_j$。

由定义 2.6 和定义 2.7 可知，对于语义模式集中的任意两个语义模式，当这两个语义模式中的操作类型序列集合与详细语义信息部分均等价时，可以认为这两个语义模式是等价的。根据给出的等价定义，找出与每个语义模式等价的语义模式集合，可以将互相等价的语义模式归为集合的一个分划。

定义 2.8： 对于 $\forall W_i < QID_i, Rf_i, Rg_i, M_i > \in WT$，$[W_i]_{WT} = \{W_j \mid W_i \equiv W_j \wedge W_j \in WT\}$，即表示语义模式集 WT 中与语义模式 W_i 等价的所有语义模式集合。

定理 2.1： 集合$\{[W_i]_{\mathrm{WT}} \mid W_i \in \mathrm{WT}\}$构成 WT 的一个分划，记为 π^{WT}。

证明：对任意的 $W_i \in \mathrm{WT}$，$[W_i]_{WT}$ 至少包含了自身，因此 $[W_i]_{WT}$ 非空。又任意两个集合 $[W_i]_{WT}$ 和 $[W_j]_{WT}$，或者就是同一个集合，或者 $[W_i]_{WT} \cap [W_j]_{WT} = \varnothing$，因此要证明的是

$\bigcup_{Wi\in WT}[W_i]_{WT}=WT$。显然有 $\bigcup_{Wi\in WT}[W_i]_{WT}\subseteq WT$。对于任意的元素$W_j\in WT$，有$W_j\in[W_j]_{WT}$，

而$[W_j]_{WT}\subseteq\bigcup_{Wi\in WT}[W_i]_{WT}$，因此$W_j\in\bigcup_{Wi\in WT}[W_i]_{WT}$，从而$WT\subseteq\bigcup_{Wi\in WT}[W_i]_{WT}$，故有$WT=\bigcup_{Wi\in WT}[W_i]_{WT}$。

由此可知，集合$\{[W_i]_{\mathrm{WT}}\mid W_i\in\mathrm{WT}\}$可以构成 WT 的一个分划，定理得证。

提取 WT 的分划π^{WT}中每一个分划块π的任意一个语义模式构成集合WT_{abbr}，称之为应用程序的简化语义模式集。根据定理 2.1 可知由于WT_{abbr}中的任意两个W_i与W_j均属于不同的分划块，因此均不存在$W_i\equiv W_j$的情况。

综合上述讨论可知，在对语义模式集中的所有语义模式进行等价判定后，可以得到语义模式集的一个分划，每个分划块都是一个语义模式子集，该子集中的语义模式互相等价，即代表的是相同的应用程序特征语义，也就是代表着同样的功能，任意选取一个语义模式即可以表示应用程序该功能，因此直接在每个分划块中任意选取一个语义模式作为代表，组成一个新的语义模式集，即WT_{abbr}，且由于经过了分划处理，WT_{abbr}中是不存在互相等价的语义模式的，保证了不会存在两个语义模式代表的是同一个功能。因此，若原语义模式集 WT 是完整的，即包含了应用程序所有的特征语义，那么WT_{abbr}也能够包含该应用程序所有的特征语义，在功能上与 WT 是等价的。

2．消除语义模式内部冗余

通过对简化语义模式集WT_{abbr}中每个语义模式进行内部模式分析，找出对应功能相同的语义模式，消除语义模式内部冗余。为了较好的描述这种内部冗余情况，首先给出可压缩的语义模式的定义。

定义 2.9： $\forall Wi<QID_i,Rf_i,Rg_i,M_i>\in WT_{abbr}$，其中$Rf_i=f(QID_i)=\{T_{1i}T_{2i}T_{3i}\cdots T_{Mi}\}$ $Rg_i=\{g(QID_i,k)\mid 1\le k\le M_i\}=<O_{ik},P_{ik},A_{ik},C_{ik}>$，称$W_i$是可压缩的语义模式，当且仅当$W_i$中的组成成分$Rf_i$可以表示成如下的形式：

$$Rf_i=f(QID_i)=\{T_{1i}\cdots T_{pi-1}[T_{pi}\cdots T_{qi}(k)]T_{hi}\cdots T_{Mi}\}$$

其中，k 为等价次数，di = qi – pi + 1，hi = pi + di * k，且满足

$$\forall x\in(pi,hi-1)\wedge(x+di)\in(pi,hi-1)，\ g(QID_i,x)\equiv g(QID_i,x+di)\text{ 及}$$

$$\{T_{pi}\cdots T_{qi}\}\equiv\{T_{pi+di}\cdots T_{qi+di}\}\equiv\{T_{pi+di*2}\cdots T_{qi+di*2}\}\equiv\cdots\equiv\{T_{pi+di*(k-1)}\cdots T_{qi+di*(k-1)}\}$$

称$\{T_{1i}\cdots T_{pi-1}\}$为前缀序列、$\{T_{hi}\cdots T_{Mi}\}$为后缀序列、$\{T_{pi}\cdots T_{qi}\}$为等价序列。

定义 2.9 给出了一个语义模式是否能够进行压缩的充分必要条件，必须同时满足的是其操作类型集合序列中存在着等价的子序列，等价的次数是不定的，但是同时必须满足的是所有重复的子序列中的每个操作类型集合所对应的详细特征语义信息也必须是等价的，那么这个多次出现的子序列对应在数据库操作日志中就相当于是某些功能中某些行为的多次使用，例如多次购买不同的物品。因此，数据库操作日志中存在的这种重复性的操作是可以通过对简化语义模式集中对应的语义模式进行模式分析提取压缩模式来进行形式上的简化。将语义模式表示成压缩形式后，后续的检测过程中的模式匹配阶段，对操作类型集合序列进行匹配时，便不需要对整个序列逐一匹配了，对于中间的等价子序列只需要匹配一次即可，极大的提高了检测的效率，后续的算法分析阶段将会进行更具体的探讨。

根据定义 2.9 对WT_{abbr}中的每一个语义模式都进行类似的处理，由于并不是所有的语义模式都是可压缩的，因此为了方便后续的检测过程进行匹配，我们将语义模式定义为可压缩的和不可压缩两种形式进行分别存储，以保证检测的准确性和方便性，这里首先将WT_{abbr}中可压缩的语义模式部分提取出来表示为$WT_{abbr_compress}$，称之为可压缩语义模式集，不可压缩的部分表示为WT_{abbr1}。

正是由于这种内部冗余的存在，造成了有些原本对应了相同特征语义的语义模式并不符合等价定义，而进行了压缩模式分析之后，可以解决这个问题。下面给出可压缩的语义模式之间等价的定义。

定义 2.10： $\forall W_i$，$W_j \in WT_{abbr\_compress}$，$W_i \equiv W_j$ 当且仅当W_i，W_j 满足下面的条件：

（1）Rf_i，Rf_j 可以表示成

$$Rf_i = f(QID_i) = \{T_{1i}\cdots T_{pi-1}[T_{pi}\cdots T_{qi}(k)]T_{hi}\cdots T_{Mi}\}\ 且$$

$$Rf_j = f(QID_j) = \{T_{1j}\cdots T_{pj-1}[T_{pj}\cdots T_{qj}(l)]T_{hj}\cdots T_{Mj}\}\ （k \neq l）;$$

（2）$\{T_{1i}\cdots T_{pi-1}\} \equiv \{T_{1j}\cdots T_{pj-1}\}$，$\{T_{pi}\cdots T_{qi}\} \equiv \{T_{pj}\cdots T_{qj}\}$，$\{T_{hi}\cdots T_{Mi}\} \equiv \{T_{hj}\cdots T_{Mj}\}$；

（3）$\forall x \in (1i, pi-1) \vee (pi, qi) \vee (hi, Mi) : g(QID_i, x) \equiv g(QID_j, x)$。

由定义 2.9 可知，对于任意两个可压缩的语义模式，若各自操作类型集合序列的前缀序列和后缀序列以及等价序列分别等价，且各自三种序列所对应的详细特征语义

信息也等价的话，那么这两个可压缩的语义模式是等价的。与定义 2.8 类似的道理，根据定义 2.10 对$WT_{abbr_compress}$进行分划处理，得到进一步简化的语义模式集合。

定义 2.11： $\forall W_i \in WT_{abbr\_compress}$，$[W_i]_{WT_{abbr\_compress}} = \{W_j \mid W_i \equiv W_j \wedge W_j \in WT_{abbr\_compress}\}$，即表示可压缩的语义模式集$WT_{abbr_compress}$中与$W_i$等价的所有可压缩语义模式集合。

定理 2.2： 集合$\{[W_i]_{WT_{abbr_compress}} \mid W_i \in WT_{abbr_compress}\}$构成$WT_{abbr_compress}$的一个分划，记为$\pi^{WT_{abbr_compress}}$。

定理 2.2 的证明与定理 2.1 证明类似。提取$WT_{abbr_compress}$的分划中每一个分划块的任意一个语义模式构成集合WT_{abbr2}，称之为简化的可压缩语义模式集。可以看到，WT_{abbr2}是在$WT_{abbr_compress}$的基础上，消除了所有可压缩语义模式间的冗余所得到的，相当于在第一次消除语义模式间冗余的基础上，通过对语义模式内部存在的冗余进行分析，达到再次消除语义模式间冗余的目的，使语义模式集得到了最大程度的简化，同时在功能上，WT_{abbr2}与$WT_{abbr_compress}$是等价的。

3．双重约简算法描述及复杂度分析

根据消除语义模式间和语义模式内部冗余的过程，提出语义模式集的双重简化算法，算法主要分为两个流程，首先进行第一重简化过程，即对 WT 中的所有语义模式进行等价判定以消除其中的冗余语义，得到简化的语义模式集WT_{abbr}；然后在此基础之上，对简化的语义模式集中的语义模式依次进行可压缩模式判定，将可压缩的语义模式提取出来构成子集$WT_{abbr_compress}$，直接对$WT_{abbr_compress}$中的所有压缩模式再次进行等价判定，得到$WT_{abbr_compress}$简化后的可压缩语义模式集WT_{abbr2}；最后将其与WT_{abbr}中的不可压缩的语义模式子集WT_{abbr1}结合得到最终的约简语义模式集，即WT_{last}。通过上述探讨，下面给出语义模式双重约简算法的形式化描述如算法 2.1。

算法 2.1 语义模式集双重约简算法

输入：初始语义模式集 WT

输出：约简语义模式集WT_{last}

描述：

1．扫描初始语义模式集 WT，根据定义 2.7 中描述的语义模式等价原则找出 WT 中存在的所有等价类；

2．根据定义 2.8 中的描述可知所有的等价类即构成语义模式集的一个划分记为 π^{WT}；

3．扫描分划块 π^{WT} 中的每一个分划 π，提取 π 中任意一个语义模式 W_i 加入到简化的语义模式集 WT_{abbr} 中；

4．For each $W_j \in WT_{abbr}$ do

根据定义 2.9 可压缩的语义模式定义对 W_j 进行压缩模式提取；

If W_j 是可压缩的 then

将 W_j 加入到可压缩的语义模式集 $WT_{abbr\_compress}$ 中；

Else 将 W_j 加入到不可压缩的语义模式集 WT_{abbr1} 中；

End if

End for

5．扫描可压缩的语义模式集 $WT_{abbr\_compress}$，根据定义 2.10 中描述的可压缩语义模式等价原则找出 $WT_{abbr\_compress}$ 中所有的等价类；根据定义 2.11 中的分划方法对所有的等价关系进行划分，得到 $WT_{abbr\_compress}$ 的划分 $\pi^{WT_{abbr\_compress}}$；

6．扫描分划块 $\pi^{WT_{abbr\_compress}}$ 中的每一个分划 π，提取 π 中任意一个语义模式 W_i 加入到简化的语义模式集 WT_{abbr2} 中；

7．分别扫描集合 WT_{abbr1} 以及 WT_{abbr2}，将其所有的语义模式依次加入到约简语义模式集 WT_{last} 中；

8．算法结束。

算法 2.1 的运行时间主要消耗在第一步对语义模式集 WT 基于语义模式等价关系的等价类求解以及第五步对可压缩的语义模式集 $WT_{abbr\_compress}$ 基于可压缩语义模式等价关系的等价类求解的过程上。由于 $WT_{abbr\_compress}$ 最坏情况是与 WT 同样大小的，因此求解算法时间复杂时对任一步骤进行分析即可，选取第一步进行分析。

假设算法输入的语义模式集大小为 N。结合语义模式自身所具备的特点以及语义模式等价的定义，此步过程中可以首先根据语义模式的长度字段对集合进行划分，将长度相同的语义模式放入同一个子集，然后分别对子集进行处理。因此假设 WT 中存在 M 种不同长度的语义模式，因此问题归结为将大小为 N 的集合 WT 划分为 M 个不同的子集，每个子集中的语义模式长度相同，由于只需要进行整数比较，利用基数排

序算法可以使划分过程在$o(N)$时间内完成。设划分后的子集大小分别为$N_1, N_2 \cdots N_m$，各子集语义模式的长度分别为$L_1, L_2 \cdots L_m$，对每个子集中的语义模式进行等价判定，对于大小为N_i的子集来说，最多可以划分为N_i个等价类，比较次数为$\frac{N_i(N_i-1)}{2} * L_i * 2 = L_i N_i(N_i-1)$，且由于实际情况下，M 的值是远小于 N 的，因此不难得出第一步的求解过程在最坏情况下，时间复杂度为$o(N+N^2)$即$o(N^2)$。

综合上述，可知算法 3.1 在最坏情况下，时间复杂度为$o(N^2)$。

2.2.4 增量学习过程描述

本文 2.2.3 和 2.2.4 节对特征模式的划分和语义模式的提取及简化过程进行了详细的探讨，并给出了相关的算法描述。本节主要结合前面所提出的划分方法以及提取和简化算法，对基于特征语义的 SQL 注入检测方法的学习过程进行整体描述。

基于用户任务的特征模式划分方法能够较好的保证所提取特征语义的精确性。将特征模式转化成语义模式，是为了更好的描述特征模式的内部函数关系，即更好的描述应用系统的特征语义。语义模式的双重简化过程主要从效率的角度出发，为了减少检测阶段的匹配次数，对得到的语义模式集进行最大程度的化简。只有当初始语义模式集能够包含应用程序全部的特征语义时，简化后得到的约简语义模式集才能够保证尽可能的完整和最简，因此在学习算法的选取过程中，必须充分考虑语义模式集完整性，否则会使得检测阶段产生大量的误报。

为了解决完整性的问题，我们将学习过程设计为增量学习模式，在应用程序运行一段时间后，审计记录达到一定的数量时，启动学习过程，使学习算法和应用程序同时运行，同时监控所生成的语义模式数。在审计记录不断增加的情况下，学习过程不断完善语义模式集，当系统监控到在较长的一段时间内，语义模式数量趋于稳定时，停止应用程序和学习过程，此时得到的语义模式集可以保证是比较完整且最简。

增量学习能够保证得到完整的语义模式集是由其特点决定的。增量学习模式下，学习过程将运行多次，每次只处理一部分的审计记录，且每次学习过程都包括了语义模式提取和双重简化过程，可以保证每次学习完毕后新生成的语义模式集都处于最简

状态。当前一次学习过程完毕后，继续提取下一部分审计记录进行学习，当得到新生成的语义模式后，并不是直接将其添加至上次学习所得到的语义模式集中，而是首先根据语义模式等价原则与已存在的语义模式进行比较，若已经存在与之等价的语义模式则忽略，反之将其加入至语义模式集中。若是在较长一段时间内语义模式数量保持平衡，则说明新生成的语义模式能够在语义模式集中找到与之等价的语义模式，不需要将其加入到语义模式集中。

因此，结合了基于用户任务的特征模式划分和语义模式双重简化过程的增量学习算法是能够保证所得到的语义模式集尽可能的完整和最简的，本文的实验部分将对此进行验证。

2.3 基于特征语义的三层检测

通过对各种 SQL 注入攻击类型的特点进行分析，同时考虑到检测效率的提高，给出基于特征语义的三层检测算法。首先，给出算法基本流程和思路；然后，给出该算法中的每一层检测的详细过程；最后，给出算法的形式化描述。

2.3.1 算法总体介绍

基于特征语义的 SQL 注入检测方法工作在学习和检测两个阶段。大多数基于机器学习的检测算法同样是工作在这两个阶段，学习阶段提取规则生成规则库，检测阶段根据规则库中的规则进行匹配。

从本文 2.2 节对学习阶段中涉及到关键技术所进行的深入探讨可以看出，本文与同类研究工作在学习阶段的不同之处在于：首先，提出了特征模式的概念以及基于用户任务的特征模式划分方法，以保证特征语义的精确性；其次，为了更好地反映特征语义的内部关系，提出了语义模式的概念，将特征模式转化为语义模式，继而实现语义模式提取和简化，尤其是在简化过程中充分的考虑到了检测阶段的效率问题，避免了后续的重复匹配；最后在整体的学习算法中使用了增量学习技术，保证了学习阶段生成语义模式集的完整性。因此，无论是从表示应用特征语义的精确性来说还是从效

率的角度上提及，学习阶段的工作都要优于一般的学习算法。

检测阶段基本上都是以学习阶段所生成的语义模式集为基准，将由当前的审计记录所生成的语义模式与正常语义模式进行匹配来实现对恶意行为的检测。本文提出的检测方法与同类的研究工作在检测阶段的不同之处在于：检测过程并不像一般的方法直接提取规则，进行匹配，而是根据所定义的语义模式的特点，同时对现有的所有 SQL 注入攻击类型的特点进行分析之后，进行分层检测。

首先，算法的每一层检测都具有针对性，能够检测出不同 SQL 注入类型，由于入侵检测领域大多数的研究工作对于 SQL 注入攻击的检测都不够全面和深入，因此仅仅能够检测到攻击特征比较明显的操作，例如用户在正常的应用行为中恶意删除某些重要信息，或者是越权访问系统信息等等，而本文提出的检测方法，对于诸如此类特征比较明显的攻击行为，在检测的第一层往往就能够被检测出来。从所能检测出来的攻击类型上看，本文的方法具有一定的优势。

其次，在入侵检测领域，衡量一个检测算法的标准除了比较能够检测到的攻击类型之外，还必须关注的是检测率和误报率。本文 2.1 节的论述中已经提及，特征语义是从应用程序应用语义的角度出发进行定义的，对于描述一个应用系统来说，基于应用语义的方法具备了其他方法所没有的精确性和独特性，所以此方法的选取是恰当的；同时，由于所定义的语义模式中的特征向量是在充分考虑 SQL 操作特点的基础上提出来的，在粒度上达到了最小化，能够反映最细微的语义信息。由此可以看到，恰当的方法加上细粒度的语义模式使得检测率得到了保证。又因为只有通过了三层保障之后才能够最终确定用户当前行为是否合法，使得误报率可以降到最低。

最后，在检测效率上，对语义模式集进行简化的过程已经使得最后进行匹配的模式数量降低，一定程度上提高了检测效率。在检测进行阶段，由于每经过一层检测之后都会减少下一层检测的输入集，因此分层的检测思路相对于一次性的全部匹配方法，同样存在着效率上的提高。

下面的 2.3.2 节首先对检测方法中具体的三层检测过程进行深入的讨论，然后给出总体算法描述以及对算法复杂度进行分析。

2.3.2 算法描述

通过对各种攻击行为所具有的特点进行分析，将检测过程分为三个层次，只有依次通过了这三层检测才能确定用户行为的合法性，且三层检测算法的前两层检测均为语义模式中的部分组成成分的匹配过程，最后一层检测属于语句级的检测，检测级别越深入，检测粒度越细。为了更好的描述三层检测过程各自具备的特点，通过具体的实例进行说明。

例 2.1： 选取部分数据库标准测试程序 TPC-W 中的典型用户行为的日志信息作为生成语义模式集的初始数据来源。任意选取经过划分后的一个操作集合，集合中的 SQL 操作序列为：

select scl_qty from shopping_cart_line where scl_sc_id = 11 and scl_i_id = 3995;

insert into shopping_cart_line (scl_sc_id, scl_qty, scl_i_id) values (11 ,1 ,3995);

select count(*) from shopping_cart_line where scl_sc_id = 11 ;

update shopping_cart set sc_time = current_timestamp where sc_id =11;

select * from shopping_cart_line join item on scl_i_id = i_id where scl_sc_id = 11;

1．操作类型序列匹配

第一层检测主要是对操作类型序列进行匹配，对应在语义模式中，相当于是对语义模式中的组成成分 Rf 进行匹配。

由本文 2.2 节的论述可知，在用户对应用程序的正常使用过程中，当执行某一项功能时，对应了数据库中的一系列操作，通过使用基于用户任务的方法对该应用程序的所有操作日志进行划分后，我们可以得到对应该功能的这个操作集合，根据这个操作集合可以生成该功能所对应的正常语义模式信息。如果应用系统的合法用户在执行该项功能的过程中，还恶意地执行了一些非法操作比如删除或者更新某些用户表的信息，虽然这些操作在实际情况中是可以执行成功的，但这样的操作却是不符合应用系统的应用语义的。

对于例 2.1 中所列举出的操作集合，可以看到，该操作集合代表的行为是用户在向表中插入一条记录之后，随即进行查询操作，然后修改该条记录的某一个字段的信

息。若用户在执行此一系列的操作过程中，违背常理的多执行了一条删除表信息的操作，或者是刻意的进行了其他的更新操作，而应用系统本身又是不允许这样做的，那么这种隐藏在正常的行为之中的，但是却不符合应用语义的行为就应当是被认为带有攻击性的行为。还有一种情况是用户所执行的单个操作语句虽然属于正常操作，但是在一次执行过程中的操作序列所代表的应用语义在应用系统中原本就不存在。

这类攻击行为的特征比较明显，因为用户执行完毕后，通过对当前审计记录进行同样的处理过程，最终能够将其表示成为与正常语义模式相同格式的模式信息，称为待检测的语义模式。那么，攻击情况下，由于所执行的操作序列与正常操作序列不一致，因此所得到的待检测的语义模式与对应了同样功能的正常语义模式中的操作类型序列集合是无法进行匹配的，或者是根本就找不到能够匹配的正常语义模式，由此可以发现异常。

在匹配过程中，根据所发现的序列中不匹配的操作的类型可以进行恶意等级鉴定。

2．详细语义信息匹配

第二层主要是对语义模式中的详细语义信息进行匹配，在对应的语义模式中，相当于是对其组成成分 *Rg* 进行匹配。详细语义信息匹配是在第一层的操作类型序列匹配成功的基础下进行的，主要是针对特征较细微的攻击行为的检测。

与第一层检测针对的攻击类型不一样，此类攻击行为并没有执行违背应用语义的操作，或者是直接在正常的行为中增加恶意的操作，而是通过修改操作集合中的正常操作语句来实现注入的，此类攻击情况最常见，攻击方式比较多样化。下面通过两个比较典型的例子进行说明。

例 2.2： 通过 union 一个恒真的查询语句，以例 2.1 中操作集合中第一条语句为例，用户通过一定的攻击手法使得在数据库中执行的语句实际上为 select scl_qty from shopping_cart_line where scl_sc_id = 11 and scl_i_id = 3995 union all select scl_qty from shopping_cart_line where 1=1。

因此可以看到，实际上返回的结果是全表的内容，那么当后面再对所得到的结果进行更新操作时，相当于更新的是全表记录，破坏了数据库中的数据正确性。

例 2.3： 通过 or 一个恒真的条件表达式，同样以例 2.1 中的第一条语句为例，用

户通过一定的攻击手法使得在数据库中执行的语句实际上为 select scl_qty from shopping_cart_line where scl_sc_id = 11 and scl_i_id = 3995 or 1=1。

因此同样的道理，返回的结果仍然是全表记录，对后续的操作会产生影响。

这一类的攻击是仅通过对 Rf 的匹配无法检查出来的，因此必须附加对 Rg 的检查。第二层检测能够检测出大多数的 SQL 注入攻击类型，是检测准确度的有效保障。

3．条件表达式检查

第三层检测为条件表达式检查，是在操作类型序列匹配和详细语义信息匹配都能成功的前提下进行的。由于第三层检测是语句级的检测，因此与语义模式是无关的，之所以加上这层检测是因为存在着可以通过第二层检测的细微的攻击行为，如例 2.4 所示。

例 2.4： 以例 2.1 中的第四条语句为例，用户通过一定的攻击手法使得在数据库中执行的语句实际上为 select count(*) from shopping_cart_line where scl_sc_id = scl_sc_id，之所以能够执行成功是因为语法分析检查并不会发觉 scl_sc_id = scl_sc_id 的错误，因此查询条件成为了恒真式。

现有的所有 SQL 注入检测的方法是无法检测出如此细微的攻击行为的，而类似的攻击行为是 SQL 注入攻击中较为常见的一种，因此有必要进行语句级的检测。

根据上面对三层检测过程的具体论述，下面给出基于特征语义的多层检测算法的形式化描述如算法 2.2。

算法 2.2 基于特征语义的三层检测算法

输入：当前审计记录，约简语义模式集 WT_{last}

输出：响应记录

描述：

1．提取当前审计记录，根据基于用户任务的特征模式划分方法对审计记录进行划分，得到划分后的操作集合 S；

2．For each $s \in S$ do

根据定义 2.2 中对特征模式的定义将操作集合 s 转化为特征模式 Q_s；

根据定义 2.4 中对语义模式的描述将Q_s转化为语义模式W_s；

根据定义 2.9 中可压缩语义模式定义对W_s进行可压缩模式提取；

已知约简语义模式集WT_{last}=不可压缩模式集WT_{abbr1} ∪ 可压缩模式集WT_{abbr2}；

If W_s是可压缩的 then WT_{nomal} =WT_{abbr2}；

Else WT_{nomal} = WT_{abbr1}；

End if

If 对于$\forall W_i \in WT_{nomal}$， Rf_s与Rf_i均不等价 then

第一层检测发现异常，转向步骤 3； continue；

Else

设WT_{nomal}中与Rf_s等价的语义模式子集为WT_{nomal_1}；

If 对于$\forall W_i \in WT_{nomal_1}$， Rg_s与Rg_i均不等价 then

第二层检测发现异常，转向步骤 3； continue；

Else

提取W_s所对应的原始操作集合 s；

If s 中的每一条语句都可以通过条件表达式检查 then

检测完毕，未发现任何异常，转向步骤 4；

continue；

Else

第三层检测发现异常，记录检查未通过的语句，转向步骤 3；

continue；

End if

End if

End if

End for

3．分析当前恶意行为，输出响应记录，返回；

4．当前行为正常，输出检测结果，返回。

检测算法的效率问题在后面的实验部分将有验证。

2.4 小结

基于特征语义的检测方法分为学习和检测阶段。语义模式的概念，不仅可以准确的反映出应用程序的特征语义，同时也包含了特征模式内部存在的各种映射关系；语义模式双重约简算法最大程度上消除了语义模式集中的冗余，避免了检测阶段的重复检索，在一定程度上提高了检测的效率。基于特征语义的三层检测算法，不仅能够检测出一般方法无法检测出来的特征非常细微的攻击行为，且误报率要低于一般方法。

3　基于构造路径的存储过程 SQL 注入检测

本章主要研究了由学习和检测阶段组成的基于构造路径的存储过程 SQL 注入检测方法。首先，在问题分析的基础上阐述了该方法的研究思路；然后，从学习阶段入手，给出了存储过程路径树的构造算法，并提出了基于路径树的构造路径提取和简化算法；最后，给出了检测阶段中基于构造路径的两层检测算法及其实现流程。

3.1　问题分析

由本文 1.2 节可以知道，目前比较成熟的 SQL 注入检测技术大多都忽视了对于存储过程中 SQL 注入检测技术的研究，甚至有的研究工作认为使用存储过程是一种有效的防御手段。由于当前所有的商用数据库系统都支持在存储过程中执行动态构造的 SQL 语句，攻击者往往能够利用动态 SQL 语句构造过程中存在的漏洞，通过设计适合的输入参数以达到注入的目的。

例 3.1：如图 3.1 所示，存储过程 GetInfo 根据用户的输入返回表 Details 的内容，设其输入参数@LoginID、@Password 均与用户输入相关。

```
1、  CREATE procedure GetInfo @LoginID varchar (128),
       @Password varchar (128) ……  As
     ……
12、If (@Statuslevel—0) and (@LoginID is null)
13、Begin
14、   Select @Public = 'select * from Details where LoginID ="Guest" ';
15、   If @City is not null
16、         Select @Public  = @Public +'" and city="'+@City+'"';
17、   Else
18、         Select @Public  =@Public +'and city="Beijing" ';
19、    Exec(@Public);
20、End
21、Else
22、   If (@Password is not null)
25、      Select @Checkuser = 'select * from Details where  LoginID='"
               +@LoginID+'" and password="'+@Password+' " ';
26、      If (@Statuslevel >0 ) and  (@Statuslevel <10 )……
35、      Else
38、         If @City is not null ……
40、         Else  ……
     ……
```

图 3.1 存储过程 Getinfo

若攻击者提供给参数@LoginID 的值为" 'or 1=1——' "，@Password 为任意字符串

设为"null"，则最终构造的 SQL 语句形式为：Select * from Details where LoginID =' ' or 1=1——' and password='null'，该语句中注释符后的内容将被忽略掉，数据库中实际执行的语句为 Select * from Details，攻击者将得到 Details 表中的所有内容。以上所列举的例子仅仅只是攻击者所能利用的攻击方式中的一种。存储过程所带来的威胁已经逐渐成为当前 SQL 注入检测技术下一步要解决的问题之一。

根据本文 1.2.2 节中的介绍可以知道，当前仅有 Ke Wei 的研究工作提及了对于存储过程中 SQL 注入的检测[58]，该检测方法通过比较存储过程中有可能构造而成的原始 SQL 语句与组合了用户输入值之后的 SQL 语句结构之间的异同实现了对 SQL 注入攻击的检测，该方法对于用对于用户正常输入单引号的情况会产生误报，并且没有全面考虑存储过程可能生成的所有 SQL 语句，对于存在较多流控制结构的存储过程，该方法还会产生漏报。

本文提出一种基于构造路径的存储过程 SQL 注入检测方法，该方法工作在两个阶段。在学习阶段，从存储过程的执行流程出发，通过分析执行参数的形成过程得到其构造路径，形成检测规则；在检测阶段将规则中的输入参数替换为用户输入值，得到最终执行的 SQL 语句，通过对该 SQL 语句进行结构和语义上的检测来判断存储过程是否存在 SQL 注入漏洞，解决了同类研究工作可能生成的 SQL 语句未考虑全面的问题，有效地减少了误报和漏报。

根据前面所论述的本章所要解决的问题，本章后面的小节将给出深入的分析。第二节主要分析学习阶段提取构造路径过程中所涉及到的关键技术，分别从分析存储过程的可注入性、构造每个执行参数的 Path-Tree 以及遍历 Path-Tree 得到构造路径三个方面展开论述；第三节主要介绍所提出的基于学习阶段所得的构造路径的检测算法，首先介绍算法的核心流程，然后分小节对每层检测技术进行详细的探讨；最后一节对本章所提出的基于构造路径的存储过程检测方法优于同类的研究工作之处进行总结，同时指出所存在的不足之处，提出之后的研究方向。

3.2 构造路径的提取

本节对构造路径提取过程中涉及到的关键技术进行深入探讨。首先，给出存储过程构造依赖分析的概念以及存储过程可注入性判定，并给出可注入执行参数集的简化思路及过程；然后，提出了存储过程 Path-Tree 的构造算法；最后，给出基于的 Path-Tree 的路径提取算法的思路和实现流程。

3.2.1 构造依赖分析

利用程序依赖性分析[59, 60]的思想，对 SQL 语句在动态构造过程中使用的语句和变量进行依赖性分析，得到变量间的关联性，作为对存储过程进行可注入性判定的基础。

将不同的数据库系统中执行动态构造 SQL 语句的函数称为标记函数，称标记函数的参数为执行参数。一般情况下，执行参数即为存储过程中定义的局部变量，对于执行参数的形式为字符串相加的情况，引进一个新变量且初始化为执行参数的值，将其作为该标记函数的执行参数。

存储过程看作一个由它所包含的语句构成的集合，对于给定的存储过程 q，设 s 表示任意一条语句，$s \in q$ 即代表 s 是 q 中的一条语句，且 s 的定义变量集 Def_S(s)={x | x 是 s 中值被改变了的变量}，引用变量集 Ref_S(s)={x | x 是 s 中被引用的变量}。

定义 3.1： 给定一个存储过程 q，$\forall s \in q$，定义 s 中的依赖关系如下：

1．Def(s，x)={y| $y \in$ Ref_S(s) 且 x 的定义引用了 y}；

2．Before(s，x)={t|t 在 s 之前执行且 $x \in$ Def_S(t) }；

3．对于 $\forall x \in$ Def_S(s)，x 的定义依赖集为

$$Dep\_D(s, x) = \{(t, y) | y \in Def(s, x) \wedge t \in Before(s, y) \wedge t \neq s\}$$

若 $(t, y) \in Dep\_D(s, x)$，称语句 s 定义的变量 x 依赖于语句 t 定义的变量 y；

4．对于 $\forall x \in Ref\_S(s)$ 且 $x \notin Def\_S(s)$，x 的引用依赖集为

$$Dep\_R(s, x) = \{(t, y) | t \in Before(s, x) \wedge y \in Def(t, x) \wedge t \neq s\}$$

若 $(t, y) \in Dep\_R(s, x)$，称语句 s 引用的变量 x 依赖于语句 t 中定义 x 所引用的变量 y。

例 3.2：以图 3.1 中存储过程 GetInfo 的行 19 中的语句为例，则有 Def_S(19)=null，

Ref_S(19)={@Public}，Before(19，@Public) ={14，16，18}，且 Def(14，@Public)=null，Def(16，@Public)={@Public，@City}，Def(18，@Public)={@Public}，因此 Dep_R(19，@Public)={(16，@City)，(16，@Public)，(18，@Public)}。

定义 3.2：给定存储过程 q，$\forall s \in q$，设 s 定义或引用的变量 x 的依赖集为 Dep_Tf(s，x)，Dep_Tf(s，x)的定义如下：

1．当 $x \in Def\_S(s)$ 时，若 $Def(s,x) = \varnothing$，则 $Dep\_Tf(s,x) = \varnothing$；若 $Def(s,x) \neq \varnothing$，则 $Dep\_Tf(s,x) = Def(s,x) \cup (\bigcup_{(t,y)\in Dep\_Tf(s,x)} Dep\_Tf(t,y))$；

2．当 $x \in Ref\_S(s)$ 且 $x \notin Def\_S(s)$ 时，若 $Dep\_R(s,x) = \varnothing$，则 $Dep\_Tf(s,x) = \varnothing$；若 $Dep\_R(s,x) \neq \varnothing$，则 $Dep\_Tf(s,x) = Dep\_R(s,x) \cup (\bigcup_{(t,y)\in Dep\_Tf(s,x)} Dep\_Tf(t,y))$。

由于执行参数在标记函数所在的语句的引用变量集，因此任意一个执行参数 x 的依赖集可以由定义 3.2 中的第二种情况得到。

3.2.2 可注入性判定

实际情况中，并非所有的存储过程都存在可注入性攻击，因此在定义 3.2 基础上提出存储过程的可注入性判定。

定义 3.3： 给定存储过程 q，p 表示 q 的任意一个与用户输入相关的输入参数，p 是可注入的(Injection_par(p))当且仅当 p 为字符串类型。

根据定义 3.3 设 q 的可注入输入参数集为 In(q)，则 In(q)={p | p 为 q 的输入参数且满足 Injection_par(p)}。

定义 3.4： 给定存储过程 q，x 表示 q 中的任意一个执行参数，x 为可注入的(Injection_exe(x))当且仅当 $\exists p \in (Dep\_Tf(q，x) \cap In(q))$。

根据定义 3.4 可设 q 的可注入执行参数集 TF(q)={x | x 为 q 中的执行参数且满足 Injection_exe(x)}，并以此作为待检测执行参数集。结合定义 3.3 及 3.4 给出存储过程的可注入性判定准则。

定理 3.1： 存储过程 q 是可注入的(Injection_p(q)))当且仅当 q 的可注入执行参数

集 TF(q)不为空。

证明：对于任意一个存储过程 q，若 TF(q)为空，由定义 3.4 可知，q 中不存在可注入执行参数，根据定义 3.3，对于任意的执行参数 x，不存在这样一个输入参数 p，使得 p 既满足可注入性同时又存在于 x 的参数依赖集之中。那么，由用户精心构造的恶意输入参数对 x 的构造过程没有影响，则说明执行 x 不会导致 SQL 注入攻击的产生。又因为 x 的任意性，因此可知存储过程 q 不存在 SQL 注入漏洞，由此定理得证。

例 3.3： 以图 3.1 所示存储过程为例，输入参数@LoginID、@Password 是可注入的，执行参数@Public 的参数依赖集中包含@City，因此具有可注入性，根据定理 3.1，存储过程 GetInfo 具有可注入性。

对于一个可注入的存储过程的待检测执行参数集，其包含的不同执行参数的参数依赖集有可能存在相同或者互相包含的关系，因此有必要对待检测集合进行简化处理。

对于 TF(q)中的任意两个执行参数，根据它们各自参数依赖集的大小给出执行参数之间包含关系的定义。

定义 3.5： 对于∀q：Injection_p(q)，$\exists x_1$，$x_2 \in$ TF(q)且设集合 A、B 分别为 A={p | p In(q)∧p Dep_Tf(q，x1)}、A={p | p In(q)∧p Dep_Tf(q，x2)}，则称 x1 包含于 x2(x1≤x2)当且仅当 A⊆B。

由定义 3.5 可知，执行参数之间的包含关系直接取决于各自参数依赖集的大小，由此得到下面推论。

推论 3.1： 对于∀q：Injection_p(q)，则∀x_1，$x_2 \in$ TF(q)：Safe(x_1)∧($x_2 \leqslant x_1$)⇒Safe(x_2)，其中，Safe(x)表示在不同用户输入下，执行参数 x 所形成的 SQL 语句均不存在 SQL 注入漏洞。

证明：若 x1 所形成的任一语句均不存在 SQL 注入漏洞，即说明 x1 的参数依赖集 Dep_Tf(q，x1)所包含的可注入输入参数均不会导致 SQL 注入攻击的发生，又由定义 3.5 可知 Dep_Tf(q，x_2)⊆Dep_Tf(q，x_1)，因此易知 Dep_Tf(q，x2)中的可注入输入参数同样也不会导致 SQL 注入攻击的发生，由此可证明推论 3.1 成立。

对于任意的两个待检测执行参数集中的执行参数 x、y，若它们之间存在 x≤y 或 y≤x 的关系，则称 x、y 具有可比性。

定义 3.6： 对于$\forall x \in TF(q)$，$[x]_{TF}=\{y \mid y \in TF(q) \wedge (x \leqslant y \vee y \leqslant x)\}$，即表示 TF(q)中所有与 x 具有可比性的执行参数的集合。

定理 3.2： 集合$\{[x]_{TF} \mid x \in TF(q)\}$构成 TF(q)的一个分划，记为$\pi^{TF(q)}$。

证明：根据定义 3.6，$\forall x \in TF(q)$，$[x]_{TF} \neq \varnothing$，又$\forall [x]_{TF},[y]_{TF}$，或为同一个集合，或$[x]_{TF} \cap [y]_{TF} = \varnothing$，因此只需证明$\bigcup_{x \in TF(q)} [x]_{TF} = TF(q)$。

显然可知$\bigcup_{x \in TF(q)} [x]_{TF} \subseteq TF(q)$，又因为$\forall y \in TF(q)$，有$y \in [y]_{TF}$，而$[y]_{TF} \subseteq \bigcup_{x \in TF(q)} [x]_{TF}$，因此$y \in \bigcup_{x \in TF(q)} [x]_{TF}$，从而$TF(q) \subseteq \bigcup_{x \in TF(q)} [x]_{TF}$，故有$\bigcup_{x \in TF(q)} [x]_{TF}=TF(q)$。

由此可知，集合$\{[x]_{TF} \mid x \in TF(q)\}$可以构成 TF(q)的一个分划，定理得证。

在定义 3.6 的基础上给出任意的一个分划块中最大执行参数的定义。

定义 3.7： 给定$\forall q$：Injection_p(q)，对于$\forall \pi \in \pi^{TF(q)}$，若$\exists x \in \pi$满足$\forall y \neq x \in \pi$：$y \leqslant x$，则称$x=\pi_{max}$表示 x 是分划块$\pi$中最大执行参数。

设由 TF(q)的分划中每一个分划块的最大执行参数构成的集合为$Todetected(q)=\{\pi_{max} \mid \pi \in \pi^{TF(q)}\}$，称为简化的待检测执行参数集。

推论 3.2： 给定$\forall q$：Injection_p(q)有

$$\forall \pi_{max} \in Todetected(q)：Safe(\pi_{max}) \Rightarrow Security(q)$$

其中，Security(q)表示该存储过程 q 不存在 SQL 注入漏洞。

证明：根据定义 3.7，$\forall \pi_{max} \in Todetected(q)$，$\pi_{max}$为$\pi^{TF(q)}$中的一个分划$\pi$的最大执行参数，则$\forall x \neq \pi_{max} \in \pi$，有$x \leqslant \pi_{max}$成立，又由推论 3.1，当$\forall x \neq \pi_{max} \in \pi$：$x \leqslant \pi_{max}$成立时，有$Safe(\pi_{max}) \Rightarrow Safe(x)$成立，则$\forall x \in \pi$：Safe(x)，即对于$\forall \pi \in \pi^{TF(q)}$，$\pi$中所有的执行参数都无法导致 SQL 注入攻击的产生，因此可以认为存储过程 q 不存在 SQL 注入漏洞，由此得证。

综合上述，在后续的检测过程中只需要对简化的待检测执行参数集进行检测即可，而并不需要检查 TF(x)，有效减小待检测参数的数量，避免了大量重复性的分析工作。因此下一步应该解决的问题是如何获得这些执行参数的动态构造过程即构造路径。

3.2.3 Path-Tree 的构造

本小节主要介绍的是如何根据上一节所得到的相关操作集构建得到参数构造路径所需要的 Path-Tree。首先，给出 Path-Tree 的构造思想及相关定义；然后，给出其构造算法的形式化描述。

1．构造思想及定义

执行参数构造路径的求解可以通过对存储过程执行流程的分析而得到，运用流图模型中块的概念[61]分析存储过程语句部分，将其主体部分表示为块序列，根据存储过程的块序列，构造能够表示其执行流程的路径树 Path-Tree，其中存储过程语句部分为存储过程中不包括任何变量声明的部分。

程序流程中的块分为节点和段两种类型，其中节点包括判断、连接和程序单元的入口点和出口点，在大多数商用数据库系统所定义的存储过程中，常用的判断类型的语句包括 If（表达式）以及 While（表达式），连接点主要为分支结构 If-Else 中的关键字 Else，关键字 Begin 和 End 作为程序单元的入口点和出口点。根据上述概念对图 3.1 中存储过程 GetInfo 的部分代码进行块的划分，如图 3.2 所示，通过块的划分，存储过程可以看作是由一系列的判断、连接、程序单元入出口以及段类型的块组成的序列，称之为块序列，下面给出形式化其形式化描述。

图 3.2 GetInfo 分块示意图

定义 3.8： 存储过程 q 的块序列表示为 $BlockSequence = \{B_1, B_2, \cdots\cdots, B_n\}$，其中 n 为其序列长度， $Class(B_i) \in \{Decision, Junction, Label, Segment\}$ 表示块 Bi 所属的类别，$Label \in \{Entrance, Exit\}$ 且 $Class(B_i) = Decision$、$Junction$、$Entrance$、$Exit$、$Segment$ 分别代表 Bi 属于判断、连接、程序单元入口、出口以及段类型。

由定义 3.8 中可以看出，一个存储过程的块序列实质上是静态的表示出了该存储过程的程序结构，它无法准确全面的描述出该存储过程的执行流程，在此提出了 Path-Tree 的概念，其构造思想即来源于对存储过程块序列的分析，序列中每一个 Class(Bi)=Decision 以及 Junction 的元素均代表 Path-Tree 中的一个节点，首先不考虑节点的其他属性项，以图 3.1 中的存储过程 GetInfo 为例，按照其构造思想所得到的 Path-Tree 的基本结构如图 3.3 所示。

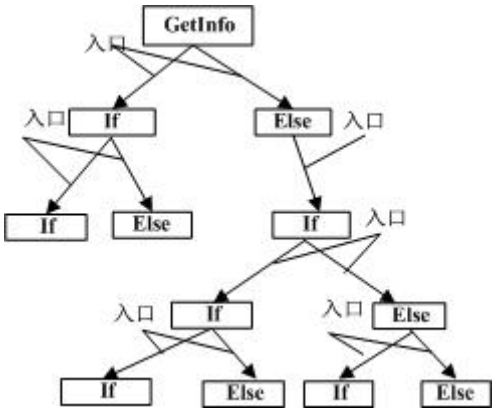


图 3.3 GetInfo 所代表的 Path-Tree 结构

由图 3.3 可以看出，首先，Path-Tree 的结构对应了存储过程的执行流程，树中任意的中间节点和叶节点均由其所对应的判断或连接类型的块中包含的关键字（keyword）标识。例如，判断类型 If（表达式），其对应块中包含的关键字即为 If，因此用 If 标识该判断类型所在的节点；对于连接点 Else，用 Else 表示其所在节点。

其次，对于标识为 If 的节点，其对应块的后续块序列中必然存在与 If 语句体的开始和结束相对应的入口类型和出口类型的块，称这两个块间的块序列为 If 的子序列。例如，图 3.2 中行号为 12 的语句 If（表达式）即对应图 3.3 中 Path-Tree 的根节点的左子节点，行号为 13 和 20 的语句代表的是 If（语句体）的开始和结束，由行 13 至 20 间的语句所形成的块序列中，块的类型依次为入口，段，判断，段，连接，段，段，出口，称由入口和出口块之间的块所构成的序列为 If 的子序列。根据相同的处理方法，对于连接点 Else，同样可以找到其对应的子序列。

综合上述，在 Path-Tree 构建过程中，判断当前节点是否存在子节点的处理方法为：

（1）若该节点的子序列中仍然存在判断或连接类型的块，即表明该节点存在子节

点，称子序列中判断或连接类型的块为划分块，每一个划分块都对应该节点的一个子节点。其中，第一个划分块之前的所有块所包含的语句依次组合构成一个语句序列，作为该节点的一个属性项，称为该节点的段类型属性项；对子序列中的任意一个划分块都采取同样的处理方法；对于最后一个划分块的子序列之后的所有块，依次将其包含的语句增添到前面所有划分块的段属性项中。

（2）若该节点不存在子序列，则说明该节点为叶节点，且这两个块间的子序列中所有段类型的块均作为该节点的段类型属性项。

根据上述论述给出完整的 Path-Tree 定义。

定义 3.9： Path-Tree 的定义符合如下条件：

①根结点表示为三元组

$$Root : (tree\_id, pro\_name, child\_seqs)$$

其中，tree_id 为当前节点在树中的编号且唯一标识该节点，pro_name 为 Path-Tree 所代表的存储过程名，$child\_seqs : \{child_1, child_2, \cdots\cdots, child_n\}$ 表示的是根节点的子节点序列，序列中的每一个元素 $child_i (1 \le i \le n)$ 均可以表示为一个二元组 $child_i : (child\_id, child\_ptrs)$，其中 child_id 为 childi 在序列 child_seqs 之中的编号，child_ptrs 是由 Root 指向子节点 childi 的指针；

②其他节点结构均可以表示为六元组：

$$PathNode : (tree\_id, keyword, dep\_stat, child\_seqs, parent\_id)$$

其中，tree_id 及 child_seqs 的含义与其在根节点中的含义相同；keyword 即为其所对应的判断或连接类型的语句中包含的关键字，dep_stat 为标识为 keyword 的节点的段类型属性项，parentid 为父节点编号。

结合定义 3.9，以图 3.1 中存储过程 GetInfo 为例所构造的完整 Path-Tree 结构如图 3.4 所示。

2．Path-Tree 的构造算法描述及复杂度分析

由定义 3.9 可知，节点的属性项 dep_stat 为段类型，又已知段是一段顺序执行的单入口且单出口的程序语句序列，因此对 Path-Tree 中每一个节点的遍历即相当于执行该节点的属性项 dep_stat 中的语句序列。

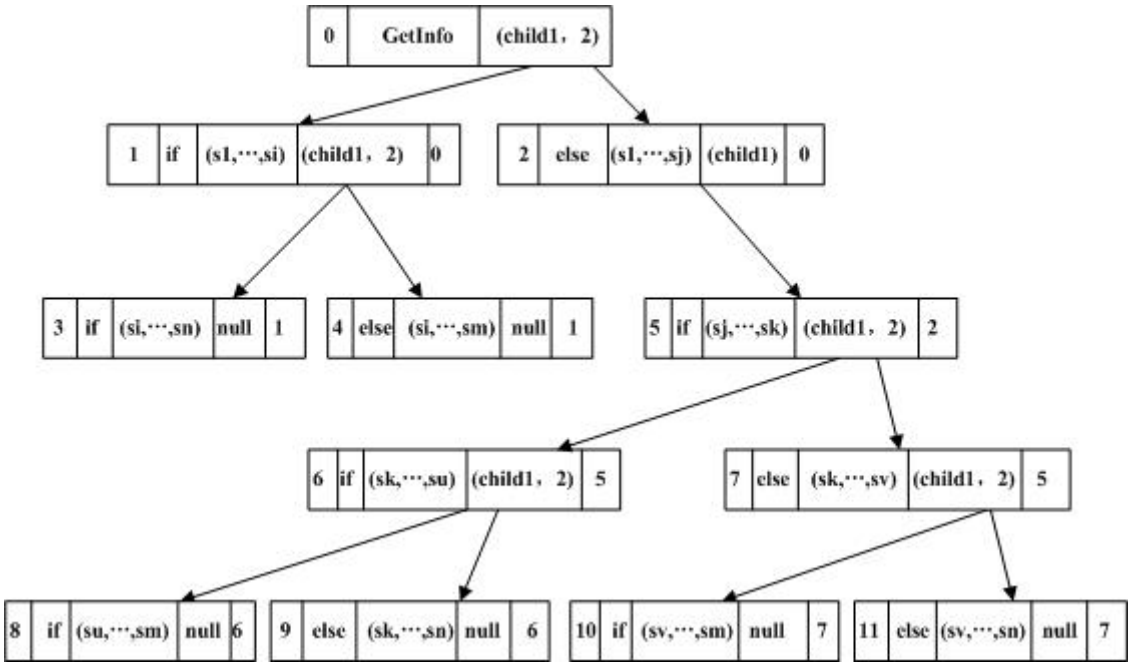


图 3.4 GetInfo 的 Path-Tree

Path-Tree 是根据存储过程块序列作出的描述存储过程执行流程的树型结构图。Path-Tree 的构造算法以根节点、存储过程的待检测参数集和块序列作为输入，新建节点的插入位置是由当前节点是否存在子节点来决定的，且对新建节点初始化时，并没有将节点对应块包含的所有语句均赋值给属性项 dep_stat，而是删除了与待检测参数集不存在任何依赖关系的语句。算法输出即为完整的 Path-Tree。

Path-Tree 的构造算法的形式化描述如算法 3.1 所示。

算法 3.1 Path-Tree 的构造算法

输入：存储过程 q 的块序列 BlockSequence(q)、待检测执行参数集 Todetected(q)、根节点 root

输出：Path-Tree

描述：

1．顺次提取 BlockSequence(q)中类别为 Decision、Junction 以及 Label 的块组合为链表 ProcedureList，扫描 ProcedureList，根据定义 3.13 后的论述，通过匹配 Label 类别中属于 Entrance 和 Exit 的块判断 Root 是否存在分支结构；

2．If 如果 Root 存在分支结构 Then

2.1．分支数赋值给 Root 的 childseq 项；

2.2．For Root 的每一个分支处 do

定位该分支处在 BlockSequence(q)中相对应的块 Bi 并新建节点 PathNode 作为 Root 的一个子节点；

对 PathNode 进行初始化：(1)每新建一个节点节点编号 treeid 增 1；(2)keyword 的值初始化为 Bi 的属性 Desicioni；(3)parentid 的值初始化 Root 的编号；(4)childseq 初始化为 null；(5)根据定义 3.2 得到 Todetected(q)中所有执行参数的参数依赖集 $A=\{Dep\_Tf(q，x)|x \in Todetected(q)\}$；

For Bi 的属性 Segmenti 中的每一条语句 s do

根据定义 3.1 得到 s 的定义依赖集 $Dep\_D(s)$ 和引用依赖集 $Dep\_R(s)$；

If 如果 $A \cap Dep\_D(s) = Null$ 且 $A \cap Dep\_R(s) = Null$ Then

删除语句 s；

End if

将改变后的 Segmenti 赋值给 Bi 的属性项 dep_stat；

End for

删除 BlockSequence(q)中 PathNode 的对应信息得到 New_BlockSequence(q)；

If 如果当前节点 PathNode 存在分支结构 Then

分支数赋值给 PathNode 的 childseq 项；

New_BlockSequence(q)赋值给 BlockSequence(q)；

PathNode 赋值给 Root；

递归调用算法 3.1，输入参数为 BlockSequence(q)，TF(q)，Root；

Else

PathNode 作为 Root 的当前子树的叶节点；

End if

End for

Else

即 q 中不存在分支结构，退出算法；

End if

3．算法结束。

算法 3.1 包含了其自身的递归调用，因此我们可以用递归方程或者递归式来表示其运行时间。算法的运行时间主要消耗在步骤 2.2 的执行过程中，又由于在其执行过程中，除去递归调用的时间，其它步骤均为的判断、赋值和计算操作，执行的次数与当前的输入规模大小一致，因此只需要线性的时间便可以完成，由此可知算法的运行时间实则主要消耗在递归处理的过程中。

根据上述分析，首先假设算法总的运行时间为$T(n)$，这里 n 表示问题的输入规模。算法递归的思想与根据先序遍历序列生成多叉树的思想类似，其递归的次数与所生成的 Path-Tree 中的节点数是一致的，递归调用的顺序遵循了自上而下，从左至右的原则。因此我们可以直接用 n 来表示递归的次数，即所生成的 Path-Tree 的节点总数。

因此，可以将算法 3.1 总的运行时间表示为如下形式。

$$T(n)=\begin{cases} c & \text{如果} n=1 \\ T(n-1)+cn & \text{如果} n>1 \end{cases}$$

构建 Path-Tree 的最终目的是为了得到执行参数所有可能的构造路径，通过对其构造路径的分析得到最终形成的语句。在对 Path-Tree 的遍历过程中，对节点的遍历即相当于执行该节点的段类型属性项中的语句序列，因此关键的问题是如何利用 Path-Tree 生成其待检测执行参数集中所有参数的构造路径。

3.2.4 基于 Path-Tree 的路径提取

本节首先给出算法中所涉及的相关定义与推论，然后给出基于 Path-Tree 的路径提取算法的形式化描述及算法复杂度分析。

定义 3.10： 给定存储过程 q，s_1、s_2 为其中任意两条语句，若 s_1 执行后可能会直接执行 s_2，则称 s_1 为 s_2 的直接前驱，s_2 为 s_1 的直接后继。语句 s 的所有后继集组成的集合称为 s 的直接后继集 Afterward(s)。

定义 3.11： 对于∀q：Injection_p(q)且∀x∈Todetected(q)，$Path_x=\{s_{x1}, s_{x2}, \cdots, s_{xn}\}$(n 为路径长度)表示 x 构造完成之前所执行的一个语句序列，其中 $s_{xi}\in Afterward(s_{x(i-1)})$，称 $Path_x$ 为 x 的一条构造路径。

对于任意一个待检测执行参数 x，设 x 的构造路径集合表示为 PATH(x)，对于PATH(x)中的任意两条路径，其中一条路径中的语句序列可能完全包含于另外一条路径中，也有可能 x 在这两条路径下所依赖的变量集合存在相等或包含的关系。

定义 3.12： 对于 $\forall$ q：Injection_p(q) 且 $\forall$ x $\in$ Todetected(q)，$\forall$ $Path_{x1}$，$Path_{x2} \in PATH(x)$，m、n 分别为其路径长度，则称 $Path_{x1}$ 包含于 $Path_{x2}$($Path_{x1} \leqslant Path_{x2}$)当且仅当满足

$$\bigcup_{i=1}^{m}(Def\_S(s_{1i}) \cup Ref\_S(s_{1i})) \subseteq \bigcup_{j=1}^{n}(Def\_S(s_{2j}) \cup Ref\_S(s_{2j}))。$$

对于任意两条构造路径 $Path_{x1}$、$Path_{x2}$，若满足 $Path_{x1} \leqslant Path_{x2}$ 或 $Path_{x1} \leqslant Path_{x2}$，则称 $Path_{x1}$ 与 $Path_{x2}$ 具有可比性，否则称 $Path_{x1}$ 与 $Path_{x2}$ 是不可比的。同时，设与任意的一条路径 $Path_x$ 存在可比性的路径所构成的集合为 $[Path_x]_{PATH}$，即 $[Path_x]_{PATH}=\{Path_{xi} | Path_{xi} \in PATH(x) \wedge (Path_x \leqslant Path_{xi} \vee Path_{xi} \leqslant Path_x)\}$。

定理 3.3： 集合$\{[Path_x]_{PATH} \mid Path_x \in PATH(x)\}$构成 PATH(x)上的一个分划，记为 $\pi^{PATH(x)}$。

证明：定理 3.3 的证明类似于定理 3.2 的证明。

由于每一个分划块中的构造路径均具有可比性，因此可以对每一个分划块都可进行最大构造路径的提取。

定义 3.13： 对于$\forall$ q：Injection_p(q)且$\forall$ x$\in$Todetected(q)，对于$\forall$ $\pi \in \pi^{PATH(x)}$，若$\exists Path_x \in \pi$满足$\forall Path_{x1} \neq Path_x \in \pi$且 $Path_{x1} \leqslant Path_x$，则称 $Path_x=PATH_{MAX}(\pi)$是分划块π的最大构造路径。

对于由 x 的每一个分划块的最大构造路径构成的最大构造路径集合，设 $S_{max}(x)$表示经过执行参数 x 经过最大构造路径集合后形成的语句集合。

推论 3.3： 给定$\forall$ q：Injection_p(q)且$\forall$ x$\in$Todetected(q)，对于$\forall$ s$\in S_{max}(x)$，若 s 在任意取值下都无法导致 SQL 注入攻击的产生，则有 Safe(x)成立。

证明：由定义 3.13 知，任意的 s 对应一个分划块，结合定义 3.12 可知，对于分划块π，若由最大构造路径 $PATH_{MAX}(\pi)$产生的语句不存在 SQL 注入攻击，那么由π中其他构造路径所产生的语句同样也不会导致 SQL 注入攻击的产生。因此，若 x 的所有

最大构造路径均不能导致 SQL 注入攻击，则 x 的路径集合 PATH(x)中所有其他构造路径均无法导致攻击的产生，即满足 Safe(x)，由此推论成立。

从上面的论述可以知道，对于排除路径之间包含关系这一问题的处理思路主要包括：提取每一个待检测执行参数的构造路径集合、划分每一个执行参数的路径集合、提取每一个执行参数的最大构造路径集合。推论 3.3 保证了对执行参数最大构造路径集合的判断可以作为判定该执行参数是否存在 SQL 注入漏洞的依据。

由此需要解决的问题归结为对执行参数最大构造路径集合的求解，有效地减少待检测的构造路径数，提高算法的效率。

如果算法依照上面的三个步骤，先求解，再分划，最后提取每一个分划块的最大构造路径，那么，在最坏的情况下，算法将会达到很高的计算时间复杂度。因而路径提取算法设计为将上述三个步骤合并到 Path-Tree 的遍历过程中，通过合适的遍历规则，得到执行参数的最大构造路径集合，力图减少路径提取时对 Path-Tree 的遍历次数，以提高算法的效率。

基于上述分析，设计了两种 Path-Tree 的遍历规则，详细介绍了如何对分别代表分支结构和循环结构的判断类型节点以及连接类型的节点进行处理的方案。

1. 程序内部的分支结构导致了执行流程的不确定性，当程序执行到判断类型的语句时，其下一步的执行直接取决于该判断语句的表达式是否满足，因此同一个分支结构中不同分支体不可能出现在同一次执行过程中，即不能同时出现在同一条构造路径中，所以，不同分支体所在的构造路径必然是存在于不同分划块中。

由此可以将程序中的分支点作为划分的一个依据，即对 Path-Tree 进行遍历提取路径时必须首先对节点的类型进行判断。

2．在对 Path-Tree 的遍历过程中，若遍历到判断类型的节点，则忽略分支结构中的判断语句，直接执行其任意一个分支体；将循环语句体其看作顺序执行的语句序列，因为循环体若在一次执行过程中没有导致 SQL 注入攻击，那么多次执行必然也不会导致攻击的产生。因此，经过类似处理后，每一次遍历过程生成的构造路径均为最大构造路径。

为了更精确的描述路径生成过程，首先设计了基于 Path-Tree 的遍历算法，该算法

采用递归的形式遍历 Path-Tree 输出节点链表，该节点链表依次保存遍历过程中所经历的节点。基于 Path-Tree 的路径提取算法以简化的待检测执行参数集为输入，对每一参数的处理过程中均调用基于 Path-Tree 的遍历算法，得到返回的节点链表，最终输出该参数的路径集合。

这里还涉及到如何确定执行参数 x 的路径数目的问题。

采取的处理思路是首先确定一个最大路径数，确保任何参数的路径数目均不超过这个最大路径数，在算法执行过程中，对于某一个执行参数，通过比较当前生成的构造路径与其上一次生成的构造路径的异同，决定是否将当前路径加入到路径集合中，当出现两条路径相等的情况时，即说明 x 的路径集合生成完毕。这个最大路径数是根据 Path-Tree 中连接类型的节点数目而计算的，连接点的数目即控制流合并的次数。

基于上述分析，下面给出路径提取算法的形式化描述，如算法 3.2 所示。

算法 3.2 基于 Path-Tree 的路径提取算法

输入：根节点 Root、Todetected(q)

输出：输出 Todetected(q)所有参数的最大路径集合

描述：

For Todetected(q)中的每一个参数 x do

设 x 的最大路径集合为 PATH'(x)且初始化为空；

For 未达到最大路径数 do

调用算法 3.3，输入参数为 Root；//得到第 i 条路径对应的节点链表 PathNodeList；

设 $Path_{xi}$ 为第 i 条路径表示的语句集合；

For PathNodeList 的每一节点类型的项 pathnode do

If pathnode. dep_stat 中不包含与参数 x 存在依赖关系的语句 then

Continue；

Else 将与之存在依赖关系的语句为 s 加入 $Path_{xi}$；

End for

将 $Path_{xi}$ 加入到 PATH'(x)；

If $Path_{xi}$ 与 $Path_{x(i-1)}$相等 then 跳出循环；

End for

End for

算法 3.3　基于 Path-Tree 的遍历算法

输入：遍历开始的节点 current_node

输出：节点链表 PathNodeList

描述：

If 如果 current_node 不为空或存在子节点 Then

将 current_node 加入到节点链表 PathNodeList；

For current_node 的每一个子节点 childnode do

If childnode.keyword 为“if”且 childnode 的遍历标记为 TRUE then

递归调用算法 3.3 自身，输入参数为 childnode；

Else if childnode.keyword 为“else”且 Path-Tree 的执行标记为 TRUE then

（1）修改 childnode 左边兄弟节点遍历标记为 FALSE；

（2）置 childnode 遍历标记为 TRUE；（3）修改 childnode 的 keyword 值为“if”；

（4）修改执行标记为 FALSE；

Else if childnode.keyword 为判断类型中表示循环结构的关键字 then

递归调用算法 3.3 自身，输入参数为 childnode；

End if

End for

return　PathNodeList；

End if

算法 3.2 以最大路径数为循环次数，在循环体中首先调用算法 3.3 得到节点链表，然后通过检查链表中的每个节点的段类型属性项来决定下一步的处理过程。

当属性项中存在对存储过程的调用语句时，其中包括对自身的调用以及对其它存储过程的调用，则将该语句从属性项中除去，对于被调用的存储过程，依据所提方法进行分析。除此之外，若发现该属性项中的某一条语句存在有与当前执行参数构成依

赖关系的变量，则将该语句加入到当前语句序列中；若不存在则继续处理链表中的下一个节点；直到当前节点链表处理完毕，此时语句序列表示 x 的一条路径，将其加入到 x 的路径集合中。

假设算法 3.2 输入的待检测执行参数集 Todetected(q)的大小为 M。算法第二层循环结构中的最大路径数根据连接类型的节点数目计算得来，结合实际情况，可以假设对于任意的一个执行参数，其最大路径数均为常数 C。同时假设调用算法 3.3 后所得到的节点链表的平均长度为 N，因而第三层循环体的执行次数最多为 N。

我们首先对算法 3.2 的运行时间进行分析。算法 3.2 的执行流程中，For 循环体的执行次数取决于当前节点的子节点数目，每进入一次循环体至多存在一次对自身的调用，与 3.1 的时间复杂度分析相类似，可以利用递归式来表示其运行时间。设 Path-Tree 中每个节点的子节点个数平均为 L，即为算法的输入规模大小，算法 3.2 的运行时间表示为$T(L)$，同时由于其他处理步骤均可以在常数时间内完成，所以$T(L)$可以表示为$T(L)=\begin{cases} c & \text{如果} L=1 \\ T(L-1)+c & \text{如果} L>1 \end{cases}$，得到最坏情况下的时间复杂度为$o(L)$。

因而可以得到算法 3.3 在最坏情况下的运行时间为$o(M*C*(o(N)+o(L)))$，由于算法 3.2 是以当前节点的子节点数 L 为循环次数生成节点链表的，因此必然有$N \le L$。由此$o(M*C*(o(N)+o(L))) \le o(M*2C*o(L))=o(M*L)$。

综合上述，可知算法 3.3 在最坏情况下，时间复杂度为$o(ML)$。

3.3 基于构造路径的检测

根据推论 3.3，判定一个执行参数 x 是否存在 SQL 注入漏洞的问题可以转化为对其最大构造路径集合的检查，因此，只需要对 PATH'(x)中的每一条构造路径进行 SQL 注入检测即可。

基于构造路径的检测分为学习和检测两个阶段。

学习阶段提取系统正常运行时的存储过程执行语句构造其相应的 Path-Tree，利用基于 Path-Tree 的路径提取算法得到执行参数 x 的待检测的路径集合 PATH'(x)，对于集合中的任一条路径，为了得到经过该路径最终形成的 SQL 语句，消去路径所包含的

语句序列中出现的与输入参数无关的局部变量，得到仅包含有输入参数的 SQL 语句，并将该 SQL 语句以规则的形式存储至规则库中。

对于$\forall Path_x \in PATH'(x)$，设经过 $Path_x$ 所形成的语句表示为 $S_i(x)$，其中 i 表示 $Path_x$ 在 PATH'(x)中的序号。

定义 3.14： 五元组<T，O，M，A，C>来描述 $S_i(x)$所代表的特征信息，称之为 $S_i(x)$的特征向量，其中：T 为语句中出现的操作类型的集合（select、update、delete、insert），O 为关系对象集合，M 为谓词集合(and、or、like、order by、group by)，A 为语句的选择字段集合，C 为语句的条件表达式中出现的属性列名。

检测时，首先根据存储过程的名字查询规则库得到该存储过程的所有规则，依次将所有规则中的输入参数替换为存储过程被调用时用户传递的输入值，得到最终被执行的 SQL 语句，根据定义 3.14 中的描述，将转换后的 SQL 语句表示为五元组的形式。

检测步骤可以分为以下两个阶段。

1．依次匹配替换用户输入值前后 $S_i(x)$的特征向量，即匹配五元组中的每一个元素是否相等，若相等则继续进行下一层的检测，若不匹配则说明转换后的 SQL 语句结构被篡改。

2．对完整的 SQL 语句的条件表达式部分进行有效性检查，检查是否存在重言式。

结合以上两个步骤的检测结果最终确定该用户输入值是否合法。

在应用系统的功能未作任何更改之前，学习阶段只需要运行一次即可，检测阶段仅需要查询规则库提取规则而无需对存储过程进行分析，最大限度的降低了对系统性能的影响。

3.4 小结

本章提出的基于构造路径的存储过程 SQL 注入检测方法工作在两个阶段。通过分析执行参数的形成过程得到其构造路径，形成检测规则；检测时将规则中的输入参数替换为用户输入值，得到最终执行的 SQL 语句，通过对该 SQL 语句进行结构和语义上的检测来判断存储过程是否存在 SQL 注入漏洞，解决了同类研究工作可能生成的 SQL 语句未考虑全面的问题，有效地减少了误报和漏报。

4　实验结果与分析

本章主要对所提出的两个检测算法进行实验研究。首先介绍实验平台结构，然后分别根据各自的实验研究内容进行实验设计，主要从两种检测方法所能够达到的检测率、误报率和漏报率、能够检测到的 SQL 注入攻击类型，以及对系统的性能影响几个角度进行验证。

4.1　实验平台设计

4.1.1　平台设计需求分析

对实验平台设计的需求分析主要从基于所提算法实现的检测系统、合理的实验审计数据来源、具有说服性审计对象三个方面展开论述。

1．基于所提算法实现的检测系统

为了更好地对两种检测方法的各项技术指标进行验证以及便于对实验结果进行分析，首先将本文所提两种检测方法实现为数据库安全平台入侵检测模块中的 SQL 注入攻击检测子模块。由此，对于两种检测方法的实验验证可以基于该子模块进行。

2．合理的实验审计数据来源

对于一个检测系统来说，首先必须有合理且符合需求的审计记录来源。本文所提两种检测方法所需要的应用程序的审计记录均由通用数据库安全增强平台的监控及审计模块提供。通用数据库安全增强平台是介于应用层和数据层之间的中间件，它的作用是截获应用程序通过 JDBC/ODBC 访问数据库的数据包，进行实时的安全监控，对不安全的数据包采取适当的安全策略进行响应处理，并将处理后的数据包转发给数据库执行。其中审计模块接收由监控模块发送的 SQL/Procedure 数据包，可以根据应用的需要，获得指定审计对象的操作记录，并将操作日志存储至后台数据库中，该操作日志即为 SQL 注入攻击检测系统所需要的审计记录来源。

3．具有说服性审计对象

在入侵检测领域中，对于任何一个检测算法，衡量算法好坏的技术指标主要有四

个：（1）检测结果所能达到的检测率；（2）检测结果中出现的误报率；（3）检测算法的效率问题；（4）算法能够检测出的攻击类型。本文所设计的实验内容也将基于以上四个方面进行，在此之前，为了便于实验设计，同时也为了使我们的实验数据具有说服性，选取数据库标准测试平台 TPC-W 作为数据库安全增强平台的审计对象。

TPC-W 测试平台[62,63]通过执行一系列具有代表性的网络事务交互来模拟典型的 Internet 电子商务环境。这些网络事务涉及查询、更新、浏览、在线订购和系统管理。TPC-W 定义了十四种不同类型的网络交互事务，TPC-W 事务类型种类更加复杂，是现实世界中数据库应用系统的一个真实写照。因此，选取 TPC-W 作为数据库安全增强平台的审计对象可以使实验所需要的审计记录更加贴近现实情况。

根据上述三个方面的论述，可以对本文的实验平台结构进行合理的设计。

4.1.2 平台总体结构设计

本文所设计的平台结构如图 4.1 所示。

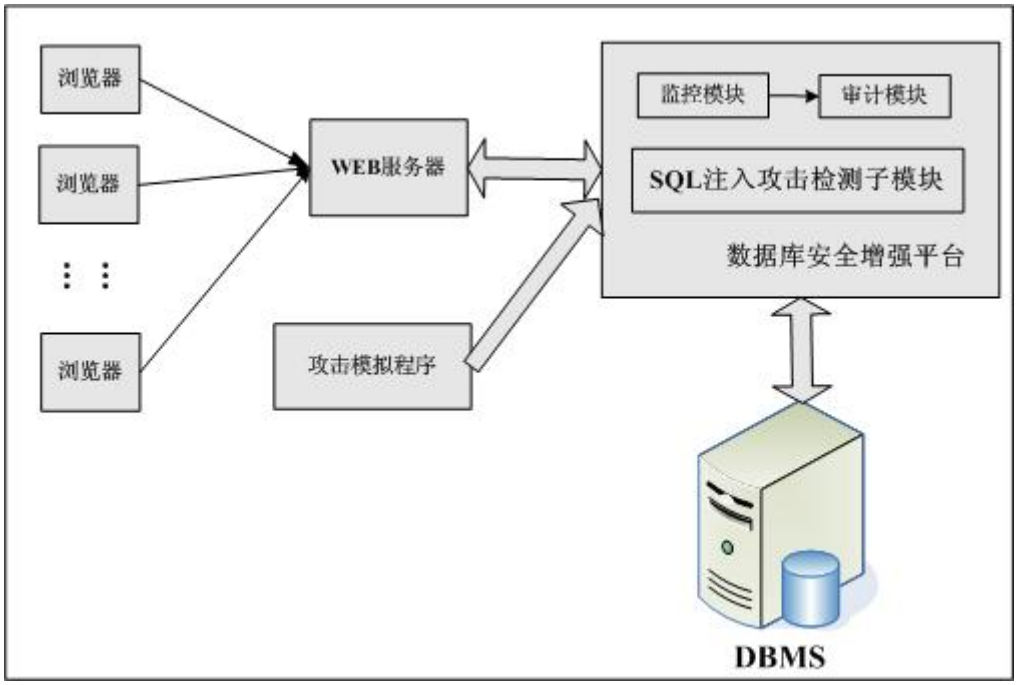


图 4.1 实验平台体系结构

TPC-W 数据库测试平台主要由四部分组成，中间系统、后台数据库服务器、Web 服务器和客户端。在该实验中，选择 SQLServer 2000 作为后台数据库服务器，Web 服

务器为 weblogic8.1，rbe 客户端远程模拟浏览器，用户通过该浏览器进行网上交易。

如图 4.1 所示，N 台客户机与 Web 服务器进行通信，每台客户机上运行 TPC-W 测试程序来生成 M 个 EB（模拟浏览器），这样就可以模拟 M*N 个终端同时进行 TPC-W 所定义的 Web 交互操作。Web 服务器用于接受客户端的 Web 请求，解析后通过 JDBC 接口发送对应的 SQL 命令给数据库服务器，数据库安全增强平台截获发往数据库的数据包进行解析，对审计对象的操作进行记录，生成 SQL 注入检测系统所需要的审计记录，处理完毕后再将数据包发往数据库服务器，数据库服务器处理完毕，把结果返回给安全增强平台，由平台将结果返回至 Web 服务器群，Web 服务器接受返回的数据，并封装成 HTML 页面，返回给对应的客户机。SQL 注入检测系统提取所生成的审计记录即可工作。同时根据本文检测算法的需求，设计了相应的攻击模拟程序，该攻击模拟程序所产生的操作语句能够被安全平台所捕获，记录至审计日志中。

在所设计的实验平台之上，本章后面的部分将分别针对于两种检测算法进行详细的实验设计，并对实验结果进行对比分析。

4.2 基于特征语义的检测实验

本节实验研究内容主要包括：增量学习算法有效性、能够检测到的 SQL 注入攻击类型数；检测率、误报率和漏报率、对系统性能的影响。根据以上研究内容，本节首先介绍实验的数据来源和方法，然后对所得到的实验结果进行深入分析。

4.2.1 实验设计

1．实验数据

（1）攻击模拟程序：通过修改实验的测试平台 TPC-W 中涉及数据库语句提交的类中的方法，来达到在正常行为中注入 SQL 语句的目的，所设计的攻击语句能够覆盖所有 SQL 注入攻击类型；（2）学习阶段的数据：在实验平台之上，安全增强平台审计模块将 TPC-W 设置为审计对象，运行 TPC-W 一段时间，可由监控模块得到 TPC-W 的操作日志；（3）检测阶段的数据：在实验平台之上，同时运行攻击模拟程序和 TPC-W，

可得到包含有 SQL 注入攻击语句的审计记录。

2．实验方法及步骤

（1）学习阶段在运行TPC-W的同时，运行基于特征语义的检测方法中所提出的增量学习算法，监测所生成的语义模式集的数量，经过一定的时间后，语义模式集数量趋于恒定，验证增量学习算法的有效性；（2）设计模拟攻击程序，使每次实验时审计数据中只存在单一类型的攻击语句，验证检测算法能够检测到的类型；（3）设计攻击模拟程序，保证涵盖所有攻击类型，且每种攻击类型的语句至少运行一次，实验进行六次，每次都运行不同的时间以生成不同数量的审计记录，对比检测率及误报率；（4）对比增加了SQL注入攻击检测子模块中的基于特征语义的检测方法前后，安全增强平台的性能，衡量标准为TPC-W测试通过测量单位时间内所处理的Web交互数量（Web Interaction Per Second，WIPS）。

4.2.2 实验结果与分析

1．增量学习算法的有效性：根据 4.2.1 中实验方法 1 中的描述，记录不同时刻，学习过程所生成的语义模式集中的语义模式数，得到的对照结果如图 4.2 所示。

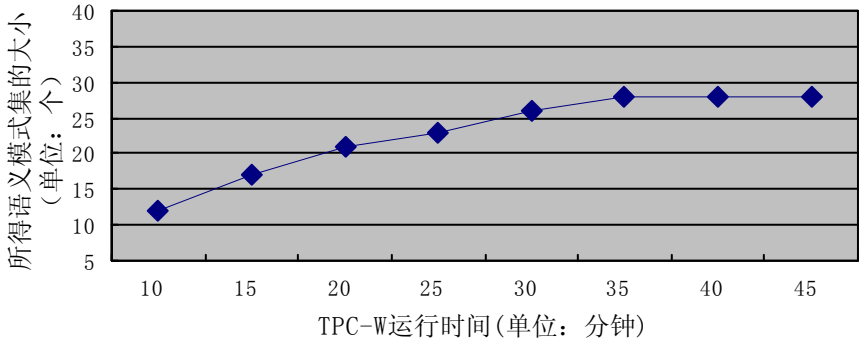


图4.2 增量学习算法运行时间与语义模式数的对比

从图 4.2 中可以看出，由于增量算法的作用，语义模式数在经过了一段时间的上升之后，最终能够在某一水平上保持平衡，此时说明对于 TPC-W 正常行为的学习过程可以结束，所得到的语义模式集能够保证尽可能的完整，同时可以说明的是，在语义模式数达到平衡之后，由于学习算法仍然在不断提取审计记录进行学习，但是语义

模式集能够保持数量上的不变，主要是因为语义模式简化算法在不断的消除冗余，才能够保证语义模式集维持在平衡的状态。

2．根据 4.2.1 中实验方法 2 中的描述，对每一种攻击类型进行单独测试，所得到能够检测到的攻击类型如表 4.1 所示。

表 4.1 可以检测到的类型

级别	攻击类型
语句级攻击类型	增加或删除 select 后面的查询字段
	增加或删除正常操作中的表
	增加或删除正常操作中的 WHERE 子句中的条件(and)
	增加或删除正常操作中的 WHERE 子句中的条件(or)
	打乱选择字段或者 WHERE 子句条件的顺序
	OR 恒真式
	使用 UNION，UNION ALL，INTERSECT，MINUS 建立复杂 SQL 语句
	改变 where 后条件表达式中的属性列名
	改变 set 后条件表达式中的属性列名
	改变 having 后条件表达式中的属性列名
事务级类型	添加任意违背应用语义的事务
	删除事务中的某些 SQL 语句
	打乱事务中的操作顺序在事务中插入 SQL 语句
	事务完成之前将其提交
	事务完成之前将其回滚

从表 4.1 可以看出，所能检测到的攻击类型涵盖了语句级和事务级的所有情况，在基于应用的入侵检测领域中的所有检测技术中，所得结果尚属前列。

3．检测率、误报率以及漏报率：分别进行六次实验，使 TPC-W 和模拟攻击程序分别运行不同的时间，所得结果如表 4.2 所示。

首先对检测的衡量指标即检测率、误报率及漏报率给出如下定义。

（1）检测率=$\frac{\text{检测出的异常事件个数}}{\text{总的异常事件个数}}\times 100\%$；

（2）误报率=$\frac{\text{误报异常事件个数}}{\text{总报警事件个数}}\times 100\%$；

（3）漏报率=$\frac{\text{漏报异常事件个数}}{\text{总异常事件个数}}\times 100\%$。

表 4.2 不同记录数下的检测结果对比

实验序号	总事件数	SQL 注入攻击事件数	检测到的 SQL 注入攻击事件数	检测率	误报数	误报率	漏报数	漏报率
1	460	45	40	88.9%	1	2.44%	5	11.1%
2	978	73	72	98.6%	1	1.37%	1	1.4%
3	1275	124	120	96.9%	3	2.44%	4	3.1%
4	2400	227	206	90.75%	6	2.83%	21	9.3%
5	3785	342	303	88.6%	11	3.5%	39	11.4%
6	4005	397	361	90.9%	10	2.70%	36	9.1%

从表 4.2 可以看出，随着 SQL 注入攻击事件的增加，所得到的误报率基本上保持在一个比较低的水平，而同类的检测技术所存在的误报率基本上都处于 20%以上，对比之下，误报率有了较大程度的降低。同时检测率有了一定程度上的提高。但是可以看出在某些情况下所产生的漏报率较高，经过分析得知，所得到的正常语义模式数量比实际的正常语义模式要多，主要是因为特征模式的划分方法尚存在不完善的方面，使得产生了系统不存在的应用语义，检测时候将异常事件当成正常的了，所以会产生漏报的情况。

4．性能对比：安全增强平台增加 SQL 注入攻击检测子模块中的基于特征语义的检测功能前后，随着用户数增加，TPC-W 测试所得到的每秒交易数对比如图 4.3。

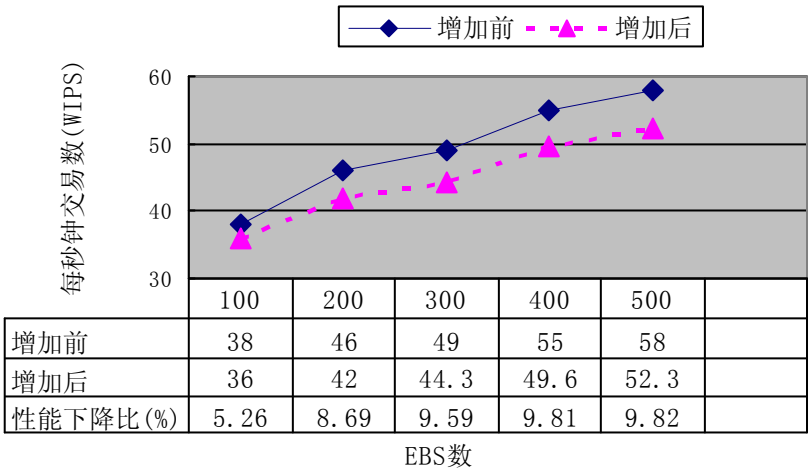


图4.3 增加前后性能对比

从图 4.3 可以看出，基于特征语义的检测功能增加前后，随着用户数增加，平台性能趋于好转。增加基于特征语义的检测功能之后，随着用户数增加，性能下降比逐渐增大，这是因为 EBS 数越多，服务器需要处理的用户事务随之增多，检测的审计数据量随之增大，导致整体性能下降，但是在用户数为 400EBS 和 500EBS 之间，性能下降比相差仅为 0.01，说明系统性能并不会随着 EBS 数的增加无限制的出现大幅度的下降。在 500EBS 内能够保证性能下降比处于 10 个百分比之内。

4.3 基于构造路径的检测实验

本节实验研究内容主要包括：检测率和误报率；能够检测到的 SQL 注入攻击类型；检测效率，对系统性能的影响。根据以上研究内容，本节首先对实验数据来源和方法进行详细设计，然后对所得到的实验结果进行深入分析。

4.3.1 实验设计

首先，介绍学习以及检测阶段各自所需要的实验数据来源；其次，阐述具体的实验方法和步骤。

1．实验数据

由于 TPC-W 测试平台并没有将操作封装在存储过程中，为了达到测试目的，修

改了 TPC-W 源程序。根据定义 3.3 及 3.4，将其所有操作中有可能存在注入的 7 种 SQL 操作改写为存储过程，并在后台数据库中创建相应的存储过程。

为了确保实验数据集的代表性，所修改的存储过程满足的条件为：

（1）可注入的输入参数及执行参数的个数均不相同；（2）执行的 SQL 语句包括了增删查改四种操作类型；（3）在执行流程复杂度上依次递增。

学习阶段，在实验平台上将修改后的 TPC-W 测试程序运行一段时间，由此得到包含存储过程执行语句的正常审计数据；检测阶段，在实验平台上同时运行修改后的 TPC-W 测试程序以及攻击模拟程序，该攻击模拟程序主要执行的是带有 SQL 注入攻击参数的存储过程调用语句，由此得到的审计数据中存在具有 SQL 注入攻击性的存储过程调用语句。

2．实验方法及步骤

本文实验分为学习阶段以及检测阶段。

学习阶段，由基于 Path-Tree 的路径提取算法得到改写的 7 个存储过程的构造路径；检测阶段同时运行 TPC-W 客户端以及攻击程序，提取构造路径对包含有攻击语句的审计数据进行检测，所设计的攻击程序中包括的 SQL 注入类型涵盖了目前已有的针对存储过程中 SQL 注入检测技术研究工作中提及的注入类型。

相关实验分为三个部分：

（1）在检测阶段分别对 TPC-W 客户端以及攻击程序同时运行不同时间所得到的审计数据进行检测率的对比；（2）对本文能够检测到的攻击类型和同类研究工作进行对比；（3）对比增加了 SQL 注入攻击检测子模块中的基于构造路径的检测功能前后，安全增强平台的性能，衡量标准为 TPC-W 测试通过测量单位时间内所处理的 Web 交互数量。

4.3.2 实验结果及分析

1．对记录数分别为 1000、2000、3000 以及 10000 的审计记录进行检测，检测结果如表 4.3 所示。

表 4.3 不同记录数下的检测结果

总事件数	存储过程执行数	SQL 注入攻击数	检测率	误报数	误报率
1000	47	14	100%	3	21.43%
2000	523	268	100%	11	4.10%
3000	1023	548	100%	17	3.10%
10000	3749	1956	100%	25	1.27%

从表 4.3 可以看出，对所产生的攻击语句的检测可以达到 100%的检测率，当审计记录为 1000 条时，存在比较高的误报率，但是明显可以看出，随着审计记录数的增加，误报率随之下降，且当记录数增加到 10000 时，误报率由原来的 21.43%减至 1.27%。经过分析可知，产生误报的原因主要是由于算法在进行构造路径提取的过程中，未完全考虑有可能出现的各种 SQL 语句结构。

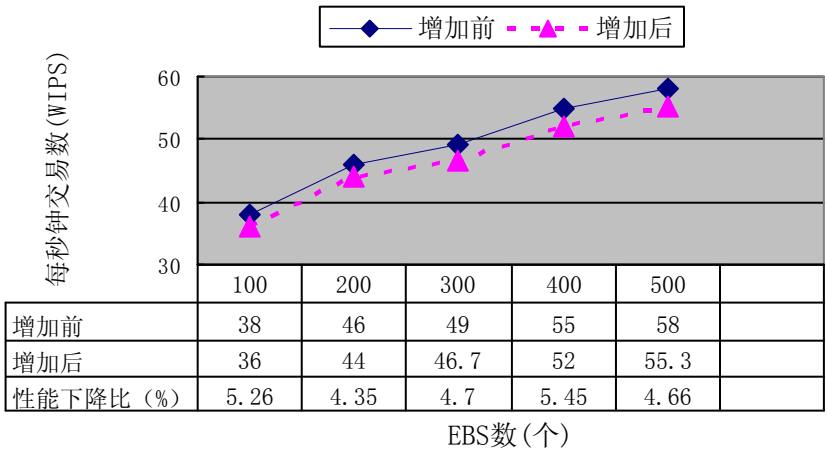
2．可以检测到的类型以及与同类研究工作进行对照的结果如表 4.4 所示。

表 4.4 与同类工作中可以检测到的类型对比

攻击类型	同类工作检测的类型	本文可以检测的类型
重言式中的’or 1=1’情况	可以检测	可以检测
重言式中的’ or ‘ ’=’ ’ ‘情况	未提及	可以检测
‘or’后的计算表达式恒成立	未提及	可以检测
增加额外语句（包括增删查改）	可以检测	可以检测
使用 union 等增加复杂查询	未提及	可以检测
使用 group by、order by 或 having 子句	未提及	可以检测
用户输入合法单引号	误报	未报错
其他注入类型	可以检测	可以检测

从表 4.4 可以看出，相对于同类的研究工作，本文所提出的检测方法能够检测到更多的攻击类型。

3．安全增强平台增加 SQL 注入攻击检测子模块中的存储过程检测功能前后，随着模拟的客户端数的增加，TPC-W 测试所得到的每秒交易数对比如图 4.4。



	100	200	300	400	500	
增加前	38	46	49	55	58	
增加后	36	44	46.7	52	55.3	
性能下降比（%）	5.26	4.35	4.7	5.45	4.66	

图4.4 增加前后性能对比

从图 4.4 可以看出，基于构造路径的检测功能增加前后，随着用户数增加平台性能都趋于好转。增加基于构造路径的检测功能之后，性能下降比并没有随着用户数增加而上升，在 500EBS 内性能下降比基本上保持在 5 个百分比左右，说明对一定的用户数内，对系统所造成的影响比较小。

4.4 小结

首先，本章对所提出的两种检测方法进行了正确性的证明，结果表明：无论在检测率、误报率还是能够检测到的攻击类型都优于同类的研究工作；其次，实验也表明，在用户数小于 500EBS 的情况下，分别增加两种检测功能后的安全增强平台的性能与增加之前的平台性能相比较，平台性能分别下降了 10%和 5%左右。

5 总结与展望

5.1 全文总结

SQL 注入攻击作为 Web 应用程序面临的最常见的攻击方式之一，其检测技术已经受到越来越多的关注。由于 SQL 注入攻击本身所具有的特点，传统的基于主机和网络的入侵检测技术是无法保证其检测准确度的，近年来发展迅速的基于应用的入侵检测技术由于使用应用语义来检测细微的异常行为，因此对于 SQL 注入攻击这种属于内部用户稍微偏离正常行为的滥用攻击行为具有自己独特的检测优势，相关研究领域也已经提出了不少较为成熟的检测技术。

基于 DBMS 的入侵检测技术属于基于应用的入侵检测技术的一种。由于数据库本身所具有的特点，对于 SQL 注入攻击检测的研究在所有基于应用的入侵检测技术中，检测效果最为显著，但是因为大多数的研究工作将重心都放在了模式识别、数据挖掘和关联规则等技术上，因此对于基于应用语义方面的研究甚少，且大多数的研究工作都忽视了数据库存储过程所带来的 SQL 注入攻击的威胁。本文即从这两个角度入手，对基于 DBMS 的 SQL 注入检测技术展开研究，论文的主要工作成果有以下几个方面。

1．针对当前基于应用语义的检测方法存在的问题，提出了一种基于特征语义的 SQL 注入检测方法，工作在学习和检测两个阶段，主要内容包括以下 5 个方面。

（1）通过分析已有的基于应用语义的检测技术存在的问题和用户使用应用程序的行为特点，提出了应用程序特征模式的概念，并对其进行了形式化的描述，应用程序的某种特征模式即对应着其某项功能；为了得到应用程序的特征模式集，提出了基于用户任务的特征模式划分方法，该方法能够准确的对应用程序的功能特征进行区分。

（2）为了更好的描述应用程序的应用语义，通过分析特征模式内部存在的映射关系，提出语义模式的概念，并完成了由特征模式到语义模式的转换，从而得到正常的语义模式集。

（3）通过对语义模式集分析得知，语义模式集以及语义模式内部都存在着冗余的问题，基于该问题，提出了语义模式的双重约简算法，该算法有如下特点：

首先，第一层简化过程中，给出了语义模式等价的概念，并基于该等价定义提出了对语义模式集进行分划的思想,通过提取分划中每一个分划块中任意一个语义模式，得到消除了语义模式间冗余的简化模式集，并对上述处理过程进行了正确性证明；

其次，第二层简化过程中，通过分析语义模式内部存在的冗余情况，提出了可压缩的语义模式的定义，并根据该定义中的准则对简化语义模式集进行压缩处理，得到可压缩的语义模式子集；给出可压缩的语义模式等价的概念，基于该等价定义对可压缩的模式子集进行类似的分划处理，得到消除了模式内部冗余的最简语义模式集；

最后，通过理论分析，证明在初始的语义模式集是完整的情况下，经过双重压缩算法后得到的约简语义模式集也是趋于完整的，并论述了该双重压缩过程有利于检测阶段效率的提高。

（4）为了保证初始语义模式集尽可能的完整，综合特征模式的划分、语义模式的提取和简化过程，提出了在增量模式下进行学习的思想，对增量学习过程进行了整体描述。

（5）通过对当前已有 SQL 注入攻击行为特点的分析，同时为了解决一般检测方法在匹配过程中存在的效率问题，提出了一种三层检测算法，该算法具有的特点为：

首先，只有当上层的检测通过后才进行下层检测，否则可以直接断定出现 SQL 注入行为，当所有层次的检测均通过后，可以认定该行为不具备攻击性，因此可以保证不同的攻击行为在不同的层次被检测出来，充分考虑各种攻击行为自身的显著特点可以有效地提高了检测的效率；

其次，对于同类基于应用语义的检测方法无法检测出来的，特征非常细微的攻击行为能够在最后一层的检测过程中被检测出来，因此在能够检测的类型方面，所提出的检测方法亦优于同类的工作。

2．针对大多数研究工作都忽视了数据库存储过程中所存在的 SQL 注入漏洞的情况，提出基于构造路径的存储过程 SQL 注入检测方法，工作在学习和检测两个阶段，主要内容包括以下 5 个方面。

（1）通过分析存储过程自身的特点，提出了存储过程构造依赖的概念，并给出其形式化描述，在此基础上给出了存储过程输入参数、执行参数以及其自身的可注入性

判定，根据上述可注入判定准则获取后续处理阶段所需要的各种参数集合。

（2）定义了可执行参数之间的包含关系，通过对可执行参数集中所有参数之间包含关系的分析，提出了可执行参数集的简化处理过程。

（3）给出了路径树 Path-Tree 的构造算法，算法以根节点和存储过程的块序列作为输入，采用递归的方法自上而下，由左至右构建 Path-Tree，算法主要研究内容包括：

首先，提出了存储过程块序列的概念，给出形式化的描述，阐述了由存储过程的定义部分得到其对应块序列的过程，及在此过程中所必须遵循的原则，通过实例对该过程进行了说明；其次，给出了 Path-Tree 的形式化定义，通过对 Path-Tree 特点的分析，阐述了由块序列到 Path-Tree 的构造过程，以及在此过程中对无关的信息进行删除所基于的准则，通过实例该过程进行了说明。

（4）给出了基于 Path-Tree 的路径提取算法，该算法遍历 Path-Tree 得到构造路径供检测阶段使用，该算法主要研究内容包括：

首先，提出了构造路径的定义，同时定义了构造路径之间的包含关系，根据这种包含关系给出消除构造路径集中冗余的过程，由此得到最简的构造路径集合，并对最简性给出理论证明；其次，通过对所要得到的构造路径的分析，提出了 Path-Tree 的遍历过程中遵循的遍历规则，同时给出每一条遍历规则中所包含的关键技术的解决思路。

（5）针对所提出的构造路径的特点，检测阶段将所得构造路径中的输入参数替换为用户输入值，得到最终执行的 SQL 语句，通过对该 SQL 语句进行结构和语义上的检测来判断存储过程是否存在 SQL 注入漏洞，解决了同类研究工作可能生成的 SQL 语句未考虑全面的问题，有效地减少了误报和漏报。

3．设计实验平台，基于平台对所提两个算法进行实验内容设计，主要从检测方法所能够达到的检测率和误报率、能够检测到的 SQL 注入攻击类型，以及对所在系统的性能影响几个角度进行验证，主要内容包括以下 3 个方面。

（1）设计了合理的实验平台，该平台由后台数据库服务器、安全增强平台、Web 服务器以及客户端以及攻击模拟程序构成。首先，该平台选取数据库标准测试平台 TPC-W 作为安全增强平台的审计对象，能够真实的模拟现实中的运行环境；其次，安全增强平台提供的监控和审计功能保证了实验所需审计数据的正确性和合理性。

（2）分别对两种检测方法进行详细的实验设计，包括实验数据、实验方法和步骤。

（3）分别进行结果分析，并与同类研究工作进行对比。结果表明：无论在检测率、误报率还是在能够检测到的攻击类型方面，两种检测方法都优于同类的研究工作；其次，实验也表明，在 SQL Server 负载较轻的情况下，分别增加两种检测功能后的安全增强平台的性能与增加之前的平台性能相比较，只下降了 10%和 5%左右。

5.2 展望

为了继续完善 SQL 注入检测技术，本课题还需要在多个方面进行进一步研究。

首先，本文所提出的基于特征语义的检测方法在学习阶段使用了阈值来对操作日志进行划分，虽然在阈值选取合适的情况下，能够很准确的表示应用程序的功能特征，并且得到很高的检测率，但是这种划分方法始终是存在着不确定性的，有待改进。在本文进一步研究工作中，可以尝试结合数据挖掘方面的相关算法进行划分。

其次，基于特征语义的检测方法中学习阶段使用了增量学习算法，算法结束标志是监控到语义模式数量在一个时间范围内处于平衡，这个时间范围若选取得过大，则会对系统效率带来较大影响，若选取得过小则无法保证语义模式集的完整性。因此时间范围的选取方法有待改进。

再次，存储过程 SQL 注入检测技术方面的研究工作在国内基本上没有，国外也较少研究工作提及，但是由存储过程中存在的 SQL 注入漏洞所引发的攻击行为日益常见，因此对于存储过程中的 SQL 注入检测技术的研究应当得到重视。本文所提出的检测方法虽然能够达到比较高的检测率，能够检测到的攻击类型也比同类研究工作要多，但是却仍旧存在着很大的限制，通用性是本文进一步研究工作中应该首要重视的问题。

此外，由于本文当前主要侧重于检测技术上的研究，衡量标准也注重于检测率和误报率还有检测到的类型，未提及到系统的层面，因此对于一般检测系统所具备的实时检测和响应处理功能尚未考虑进去，这也是本文有待进一步完善的地方。

最后，对于 SQL 注入检测技术的研究，应该不仅限于本文所基于的两个角度，因此扩展本文的研究工作是非常有必要的。

由于时间仓促和作者水平有限，本文中存在错误在所难免。敬请批评指正。

致谢

在即将结束硕士研究生的学习生活之际，我首先要感谢我的导师曹忠升副教授。

诚挚感谢曹老师在这两年多的时间里，在我的课题研究工作遇到瓶颈的时候，给我提供建设性的意见和解决问题的思路；在我负责研会工作期间倍感压力的时候，给我以精神上的鼓励和思想上的指导；在平常的学习和生活遇到困难的时候，给我以前进的动力和自信。曹老师对研究工作的严谨细致、对教学工作的认真负责、对学生的引导启发让我感受到了他作为一名优秀的老师，所散发出来的人格魅力。从他的身上所学习到的，都将是我人生中的一笔宝贵的财富。在本文研究和写作过程中，曹老师耐心地审阅了本文提纲和全稿，提出了许多宝贵的意见。值此论文完成之际，谨向恩师表示深深的谢意和良好的祝愿！

特别要感谢我的课题指导老师朱虹教授。朱老师对我的课题研究进行了悉心的指导，也给予了我极大的帮助，使我能顺利地完成课题研究工作，从而完成本文的撰写。朱老师高尚的科研风范、优秀的工作能力和对学生负责的态度，促使我积极投入研究工作中，令我受益匪浅。

感谢同课题组的应小全同学在我两年的课题研究工作中，牺牲自己的时间帮助我解决理论以及实验中碰到的难题，感谢他所给予的这种无私的帮助。感谢同实验室的沈剑、汪志鹏、娄小川、华学勤、何维英以及叶传虎等同学，在完成课题研究和论文写作的过程中对我的支持和帮助。另外，要感谢蔡畅同学在毕设实验期间对我的帮助。

感谢达梦数据库有限公司以及数据库与多媒体技术研究所为我提供了良好的学习和科研环境，感谢华中科技大学所有曾经给我授课和帮助的老师们。

感谢我的父母，我每一步的成长和进步都凝聚着他们的心血和汗水。感谢我们寝室的姐妹们和本科同学，在这两年的生活中陪伴着我一起走过，有了你们，平凡的生活中才会有如此多的惊喜。同时还要感谢潘洁每天都为我们解决一些琐碎的事情。

感谢各位专家和评委的耐心审阅，他们提出了很多宝贵的意见和建议。

最后，衷心感谢所有亲人给予我的关心、爱护、支持和帮助。

参考文献

[1] 陈小兵. SQL 注入攻击及其防范检测技术研究. 计算机工程与应用, 2007, (11): 150~152

[2] J Allen, A Christie, W Fithen, et al. State of the Practice of Intrusion Detection Technologies: [Ph.D. Thesis]. America: the Software Engineering Institute of Carnegie Mellon University, 2000

[3] S Axelsson. Intrusion Detextion Systems: A Survey and Taxonomy. in: Proceedings of the 14th USENIX conference on System administration. Seattle, Washington, USA, 2000. 208~217

[4] J P Anderson. Computer security threat monitoring and surveillance. in: Proceedings of the 1st Aerospace Computer Secutity Applications Conference. Washington: IEEE Computer Society Press, 1985. 30~36

[5] 林冬梅, 钟勇, 秦小麟. 应用入侵检测研究与发展. 计算机科学, 2007, 34(7): 10~13

[6] S E Smaha. Haystack: an intrusion detection system. in: Proceedings of the 4th Aerospace Computer Secutity Applications Conference. Washington: IEEE Computer Society Press, 1988. 37~44

[7] D E Denning. An intrusion detection model. IEEE Transaction on Software Engineering, 1987, (13): 222~232

[8] M Roesch. Snort-lightweight intrusion detection for networks. in: Proceedings of the 13rd USENIX conference on System administration. Seattle, Washington, USA, 1999. 229~238

[9] V Paxon. Bro: A system for detection network intruders in real-time. Computer Networks, 1999, 31(23~24): 2435~2463

[10] W Lee, C Park, S Stolfo. Towards Automatic Intrusion Detection using NFR. in: Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring. Santa Clara, CA, USA, 1999. 77~86

[11] R Yip, K Levitt. Data Level Interface Detection in Database Systems. in: Proceedings of the 11st IEEE Computer Security foundation Workshop. Rockport, Massachusetts, USA, 1998. 179~189

[12] J M Dermott, D Goldschlag. Towards a model of storage jamming. in: Proceedings of the 9th IEEE Computer Security Foundations Workshop. kenmare, Ire-land, 1996. 176~185

[13] M Almgren, U Lindqvist. Application-Integrated Data Collection for Security Monitoring. in: Proceedings of the 4th International Symposium on the Recent Advances in Intrusion Detection(RAID 2001). Davis, USA, 2001. 22~36

[14] T Ryutov, B C Neuman, D Kim. Integrated Access Control and Intrusion Detection for Web Server. IEEE Transactions on Parallel and Distributed. System, 2003, 14(9): 841~850

[15] J S Salvatore, S Hershkop, K Wang. A Behavior-Based Approach to Securing Email System. in: Proceedings of Mathematical Methods, Models and Architectures for Computer Networks Security, St. Petersburg, Russia, 2003. 57~81

[16] Y L Jones, Y Li. Applicartion Intrusion Detection using Language Library Calls. in: Proceedings of the 17th Annual Computer Security Applications Conference(ACSAC). New Orleans, Lousiana, 2001. 442~449

[17] M G Welz, A C M Hutchison. Interfacing Trustes Applications with Intrusion Detection System. in: Proceedings of 4th Symposium on Recent Advances in Intrusion Detection(RAID). Davis, California, USA, 2001. 37~53

[18] R E Schantz, F Webber, P Pal. Protecting Application against Malice Using Adaptive Middleware. in: Proceedings of International Workshop on Certification and Security in E-Services. Montreal, Quebec, Canada, 2002. 73~108

[19] R S Sielken. Application Intrusion Detection. in: Proceedings of the 15th Annual Computer Security Applications Conference(ACSAC), 1999. 117~134

[20] S Stolfo, D Fan, W Lee. Credit card fraud detection using meta-learning: Issues initial results. in: Proceedings of AAAI Workshop on AI Approaches to Fraud Detection and Risk Management. Menlo Park, CA, USA, 1997. 96~103

[21] S Rosset, U Murad, E Neumann. Discovery of Fraud Rules for Telecommunications Challenges and Solutions. in: Proceedings of Knowledge Discovery and Data Mining(KDD). San Diego, CA, USA, 1999. 409~413

[22] N Ye, Q Chen. An Anomaly Detection Technique Based on a Chi-Square Statistic for Detecting Intrusions into Information Systems. Quality and Reliability Eng, 2001, 17(2): 105~112

[23] G E Liepins, H S Vaccaro. Intrusion Detection: Its Role and Validation. Computers Security, 1992, 11(4): 347~355

[24] A S Christensen, A Moeller, M I Schwartzbach. Precise analysis of string expressions. in: Proceedings of the 10th International Static Analysis Symposium, 2003. 1~18

[25] V B Livshits, M S Lam. Finding security vulnerabilities in java applications with static analysis. USENIX Security Symposium, 2005. 155~165

[26] G Wassermann, Z Su. An Analysis Framework for Security in Web Applications. in: Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems (SAVCBS), 2004. 70~78

[27] C Gould, Z Su, P Devanbu. JDBC checker: A static analysis tool for SQL/JDBC applications. in: Proceedings of the 26th International Conference on Software Engineering (ICSE). Washington D C: IEEE Computer Society, 2004. 697~698

[28] C Gould, Z Su, P Devanbu. Static checking of dynamically generated queries in database applications. in: Proceedings of the 26th International Conference on Software Engineering (ICSE). Washington D C: IEEE Computer Society, 2004. 645~654

[29] R Feiertag, S Rho, Lee Benzinger et al. Intrusion detection inter-component adaptive negotiation. Computer Networks, 2000, 34:605~621

[30] Y W Huang, F Yu, C Hang, et al. Securing web application code by static and runtime protection. in: Proceedings of the 13th International World Wide Web Conference (WWW). New York: ACM Press, 2004. 40~52

[31] Z Su, G Wassermann. The Essence of Command Injection Attacks in Web Applications. in: Proceedings of the 33th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. Charleston, South Carolina, USA, 2006. 372~382

[32] W G Halfond, A Orso. AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks. in: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. Long Beach, CA, USA, 2005. 174~183

[33] W G Halfond, A Orso. Combining static analysis and runtime monitoring to counter SQL-injection attacks. in: Online Proceeding of the 3th International ICSE Workshop on Dynamic Analysis(WODA), 2005. 22~28

[34] G T Buehrer, B W Weide, P A G Sivilotti. Using Parse Tree Validation to Prevent SQL Injection Attacks. in: Proceedings of the 5th International Workshop on Software Engineering and Middleware. Lisbon, Portugal, 2005. 106~113

[35] Y W Huang, S K Huang, T P Lin, et al. Web application security assessment by fault injection and behavior monitoring. in: Proceedings of the 11th International World Wide Web Conference (WWW). New York: ACM Press, 2002. 30~42

[36] F Valeur, D Mutz, G Vigna. A Learning-Based Approach to the Detection Attacks in Web Applications. Annual Symposium on principles of Programming Languages(POPL), 2006. 372~382

[37] S Thomas , L Williams. Using automated fix generation to secure SQL Statements. in: Proceedings of the 3th International Workshop on Software Engineering for Secure Systems (SESS), 2007. 236~242

[38] Y Kosuga, K Kono, M Hanaoka. Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection. in: Proceedings of the 23th Annual Computer Security Applications Conference(ACSAC), 2007. 107~117

[39] 曹忠升, 李晶. 安全数据库系统中在线入侵检测的设计与实现. 计算机工程与科学, 2005, 27 (9): 16~18

[40] 刘大勇, 张玉清, 沈钧毅. 事务级数据库入侵检测系统的设计. 中国农业大学学报, 2006, 11(4): 109~113

[41] 王刚. 数据库入侵检测系统的设计与实现: [硕士学位论文]. 沈阳: 东北大学图书馆, 2005

[42] 钟勇. 安全数据库异常检测和若干关键技术研究: [博士学位论文]. 南京: 南京航天航空大学图书馆, 2006

[43] 李卫强. 基于数据库的入侵检测技术的研究: [硕士学位论文]. 长沙: 中南大学图书馆, 2007

[44] 李晓蕊. 数据库入侵检测技术的研究: [硕士学位论文]. 北京: 北京理工大学, 2008

[45] 周敬利, 王晓锋, 余胜生. 一种新的反 SQL 注入策略的研究与实现.计算机科学, 2006, (11): 64~68

[46] Y Hu, B Panda. A data mining approach for database intrusion detection. in: Proceedings of the 2004 ACM Symposium on Applied Computing. Nicosia, Cyprus, 2004. 711~716

[47] Y Hu, B Panda. Identification of Malicious Transactions in Database Systems. in: Proceedings of the 7th International Database Engineering and Application Symposium(IDEAS). Hong Kong, SAR, 2003. 329~335

[48] C Y Chung, M Gertz, K Levitt. Discovery of Muti-Level Security Polices. in: Proceedings of the 14th Annual IFIP WG 11.3 Working Conference on Database Security. Amsterdam, Netherlands, 2000. 173~184

[49] C Y Chung, M Gertz, K Levitt. DEMIDS: A Misuse Detection System for Database Systems. in: Proceedings of 3th International IFIP TC-11 WG11.5 Working Conference on Integrity and Internal Control in Information System, 1999. 159~178

[50] E Bertino, A Kamra, E Terzi, et al. Intrusion detection in RBAC-administered databases. in: Proceedings of the 21th Annual Computer Security Applications Conference, 2005. 162~176

[51] S Y Lee, W L Low, P L Wong. Learning Fingerprints for A Database Intrusion Detection System. in: Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS). Zurich, Switzerland, 2002. 264~280

[52] W L Low, J Lee, P Teoh. DIDAFIT: Detecting Intrusions in Databases through Fingerprinting Transactions. in: Proceedings of the 4th International Conference on Enterprise Information Systems(ICEIS). Paphos, Cyprus, 2002. 264~269

[53] Y Zhong, X L Qin. Research on Algorithm of User Query Frequent Item sets Mining. in: Proceedings of the 3th International Conference on Machine Learning and Cybernetics. Shanhai, China, 2004. 1671~1676

[54] M Vieira, H Madeira. Detection of malicious transactions in DBMS. in: Proceedings of the 11th IEEE Intl. Symposium Pacific Rim Dependable Computing. Los Alamitos. 2005. Washington D C: IEEE Computer Society, 2005. 205~216

[55] J Fonseca, M Vieira, Henrique Madeira. detecting malicious SQL. Lecture Notes In Computer Science, 2007. 236~245

[56] J Fonseca, M Vieira, H Madeira. Integrated Intrusion Detection in Databases. LADC 2007, (4746): 198~211

[57] J Fonseca, M Vieira, H Madeira. Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks. in: Proceedings of the 13th IEEE International Symposium on Pacific Rim Dependable Computing, 2007. 365~372

[58] K Wei, M Muthuprasanna, Suraj Kothari. Preventing SQL Injection Attacks in Stored Procedures. in: Proceedings of the 2006 Australian Software Engineering Conference, 2006. 56~64

[59] 陈振强, 徐宝文. 一种并发程序依赖性分析方法. 计算机研究与发展, 2002, 39(2): 160~164

[60] 徐宝文, 张挺, 陈振强. 递归子程序的依赖性分析及其应用. 计算机学报, 2001, 24(11): 1178~1184

[61] 杜子德. 程序控制流图:一种可视化的程序设计工具. 计算机研究与发展, 1995, 32(12): 15~20

[62] D F Garcia, J Garcia. TPC-W e-commerce benchmark evaluation. Computer, 2003, 35(2): 42~48

[63] D A Menasce. TPC-W: a benchmark for e-commerce. Internet Computing, IEEE, 2002, 6(3): 83~87

附录 攻读学位期间发表学术论文目录

[1] 熊婧, 曹忠升, 朱虹等. 基于构造路径的存储过程 SQL 注入检测. 第二十五届全国数据库学术会议论文集, 计算机研究与发展, 2008, 45(Suppl.): 125~129（第一作者单位：华中科技大学计算机学院）