计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(1):98−105 [doi: 10.15888/j.cnki.csa.006167]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于代理模式的 SQL 注入过滤方法①

韩宸望[1,2], 林　晖[1,2], 饶绪黎[3], 黄　川[1,2]

[1](福建师范大学 数学与计算机科学学院, 福州 350117)
[2](福建师范大学 福建省网络安全与密码技术重点实验室, 福州 350117)
[3](福州职业技术学院 信息技术工程系, 福州 350108)

摘　要: 针对 Web 安全中的 SQL 注入问题, 提出了一种新的 SQL 注入过滤方法——LFS (length-frequency-SQL syntax tree) 过滤方法. LFS 方法包括学习和过滤两个阶段, 其中, 学习阶段在安全的环境下, 通过爬虫和数据库代理构建 URL 和 SQL 语句映射表; 过滤阶段通过对 URL 长度、访问频率及 SQL 语法树这三个方面进行检测, 以此实现对用户输入进行过滤, 防止 SQL 注入攻击. 仿真实验及结果分析表明 LFS 方法相较于传统的关键字过滤和正则表达式过滤能够更有效的防止 SQL 注入攻击.

关键词: SQL 注入攻击; Web 安全; 用户输入过滤; SQL 语法树

引用格式: 韩宸望,林晖,饶绪黎,黄川.基于代理模式的 SQL 注入过滤方法.计算机系统应用,2018,27(1):98–105. http://www.c-s-a.org.cn/1003-3254/6167.html

SQL Injection Filtering Method Based on Proxy Mode

HAN Chen-Wang[1,2], LIN Hui[1,2], RAO Xu-Li[3], HUANG Chuan[1,2]

[1](School of Mathematics and Computer Science, Fujian Normal University, Fuzhou 350117, China)
[2](Fujian Provincial Key Laboratory of Network Security and Cryptology, Fujian Normal University, Fuzhou 350117, China)
[3](Department of Information Technology Engineering, Fuzhou Polytechnic, Fuzhou 350108, China)

Abstract: To solve the SQL injection problem in the Web security, a new SQL injection filtering method named LFS (length-frequency-SQL syntax tree) is proposed in this study. The LFS includes two phases: the learning and the filtering phase. In the learning phase, the URL and the SQL statement mapping table are built based on the crawler and the database agent in a secure environment. In the filtering phase, the URL length, the access frequency, and the SQL syntax tree are detected to filter the user input to prevent SQL injection attacks. Simulation experiments and results analysis denote that the proposed LFS method can prevent SQL injection attacks more effectively than the traditional keyword filtering and regular expression filtering methods.

Key words: SQL injection attack; Web security; user input filtering; SQL syntax tree

随着 Web 3.0 时代的到来以及 B/S 模式的迅速发展, Web 技术在许多领域都得到了广泛的应用, 且已成为世界经济发展中的关键技术之一[1]. 与此同时, Web 的安全问题也日趋严峻, Web 安全威胁已经成为网络安全威胁最主要的来源之一. 因此, 对 Web 安全的研究也成为了目前网络安全领域的重点和热点[2]. 其中, SQL 注入作为 Web 应用程序面临的最普遍、最高危的攻击之一, 连续多年位列 OWASP(开放式 Web 应用程序安全项目, Open Web Application Security Project) 年度十大攻击行为, 并且多次排名第一.

基于上述分析, 本文结合 URL 长度、访问频率及 SQL 语法树, 提出了一种新的 SQL 注入过滤方

① 基金项目: 国家自然科学基金 (61363068, 61472083); 福建省引导基金 (2016Y0031); 福州市科技局基金 (2015-G-54, 2015-G-84)
收稿时间: 2017-04-16; 修改时间: 2017-05-02; 采用时间: 2017-05-16; csa 在线出版时间: 2017-12-22

法——LFS (length-frequency-SQL syntax tree) 过滤方法. 该方法包括学习和过滤两个阶段. 学习阶段在安全的环境下进行, 通过爬虫和数据库代理, 根据用户提供的爬虫启始数据进行爬取, 生成 URL 和 SQL 语句的映射表. 过滤阶段工作在现实环境下, 通过检测长度、连接频率、SQL 语法树特征, 以此来检测 SQL 注入攻击. 仿真实验及结果分析表明 LFS 方法相较于传统的关键字过滤和正则表达式过滤能够更有效的防止 SQL 注入攻击.

1 相关工作

针对 SQL 注入攻击检测, 国内外研究人员已经做了大量的研究工作, 并且取得了一定的研究成果. 张燕等[3]提出了一种基于数据挖掘技术的 SQL 注入攻击检测方法. 该方法首先收集了数据库日志中内部查询树; 然后, 根据查询树的语义、语法特征和查询树类型提取查询树的特征向量; 最后, 根据多项式核函数 SVM 对这些特征向量进行分类, 从而实现 SQL 攻击检测.

黄保华等人[4]提出了一种 SQL 语句块摘要树模型, 用于检测 SQL 注入攻击. 基于该模型的 SQL 注入检测对 SQL 语句序列进行了检测, 有效地提高了拦截率, 但是在数据库连接共享的环境中, 该模型由于需要对顺序执行的 SQL 序列进行检测导致实施起来存在一定的困难. 赵宇飞等人[5]根据 SQL 注入攻击的网络流量与正常用户请求的网络流量有较大的区别, 以网络流量分析的角度, 提出了 LFF(length-frequency-feature) 检测方法. 张志超等人[6]结合人工神经元网络, 提出了一种基于人工神经元网络的 SQL 注入漏洞的分析模型. 该模型利用人工神经网络算法对用户输入的 SQL 语句进行检测, 从而判断用户是否在进行 SQL 注入攻击.

王伟平等人[7]利用正则表达式对 SQL 注入攻击的特征进行描述, 并提出了基于正则表达式的 SQL 注入攻击过滤方案. 该方案与关键字过滤相比, 具有更高的识别率和较低的误报率. Ivan Ristc 开发的开源项目 ModSecurity[8]和 Roesch M 开发的入侵检测软件 Snort[9]同样根据 SQL 注入攻击的特征定制 SQL 注入攻击的特征规则集, 并以此过滤用户的输入来抵挡 SQL 注入攻击. 田玉杰等人[10]提出了一种基于分类的 SQL 注入攻击双层防御模型. 该模型采用了基于 Http 请求分类的用户输入过滤, 以此降低用户输入过对正常数据的误报率. 应用 SQL 语法结构比较和于参数化分类的动态查询匹配, 从而提高了用户输入过滤的拦截率和语法结构比较的检测效率.

Ain Zubaidah Mohd Saleh 等人[11]将提出了一种 Web 应用程序漏洞检测方法. 该方法结合了 Boyer-Moore 字符串查找算法, 能够有效地检测多种 Web 安全漏洞, 例如 SQL 注入, XSS 等 Web 安全漏洞. Nency Patel 等人[12]改进了 Aho-Corasick 模式匹配算法, 并将该算法应用于 SQL 注入攻击防御. 传统的防止 SQL 注入的模式匹配往往采用静态的模式匹配算法. 但是改进的 Aho-Corasick 模式匹配算法能够捕捉 SQL 注入引起的新异常模式, 并将反复出现的新异常模式自动加入模式匹配库中. Jemal Abawajy 等人[13]针对 SQL 注入攻击在 RFID 系统中的危害, 提出了一种基于策略的 SQL 注入攻击检测方法. 该方法通过制定合法的 SQL 查询语句规则形成规则库, 之后对 RFID 系统传输的数据进行拦截验证, 并根据合法的 SQL 查询语句规则库进行检测, 以此来防御 SQL 注入.

同时, 还可以通过采用参数化查询, 在代码层对 SQL 注入攻击进行防御[14]. 使用参数化查询访问数据库, 能够防止 SQL 注入, 使 Web 应用程序更加安全. 此外, 通过对存储过程的访问权限的正确配置也能够防止 SQL 注入, 提高系统安全性. 存储过程将一组 SQL 语句集编译后存储在数据库中, 以此来提高安全性、防止 SQL 注入. 例如, 黄龙军[15]在网上考试系统中利用 SQL Server 存储过程有效地防御 SQL 注入攻击, 提高了系统的安全性.

2 SQL 注入过滤方法概述

2.1 SQL 注入攻击概述

SQL 注入攻击是一种由攻击者通过影响应用程序向后台数据库传递的 SQL 查询而引发的攻击. SQL 注入攻击是一种通过操纵 Web 输入来修改后台 SQL 语句以利用代码进行攻击的技术[16]. 虽然不同种类的 SQL 注入的攻击方式有所区别, 但它们的原理和过程基本相同. SQL 注入攻击过程和原理如图 1 所示.

攻击者在登录界面使用账号 admin'和密码 admin 进行登录. 客户端浏览器向 Web 服务器发送 URL 为: http://www.test.com/login.asp?username=ad min'&password=admin 的请求. 之后 Web 服务器向数据库服务器

发送 SQL 语句, 该 SQL 语句为: select * from accounts where username='admin' and password='admin'. 数据库执行该 SQL 语句后向 Web 服务器返回敏感信息, 并最终由 Web 服务器将该敏感信息转发给了客户端浏览器, 从而被攻击者获取.

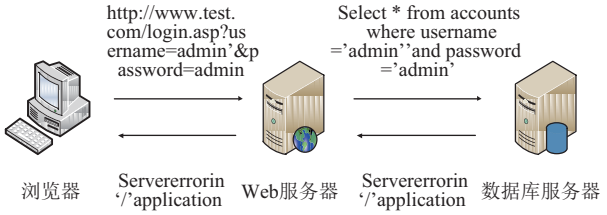

图 1 SQL 注入攻击过程

2.2 SQL 注入过滤方法

对于特定的 Web 应用程序, 其发送的 HTTP 请求报文的 URL 和参数与 Web 应用程序收到请求报文执行的 SQL 语句存在一定的映射关系. 并且任何拥有 SQL 攻击特征的输入都将改变原有的 SQL 语句的语法结构. 因此若用户输入构成的 SQL 语句的语法结构和期望中的语法结构不一致, 则该输入为 SQL 注入. 本文方法基于上述前提, 提出了一种新的 SQL 注入过滤方法, 该方法将 SQL 注入防护分为两个阶段: 学习阶段和过滤阶段.

2.2.1 学习阶段

学习阶段工作在安全环境下, 利用爬虫和数据库代理, 根据用户提供的启始文件 (XML 文件), 该文件包含爬虫的启始数据, 对该 Web 应用程序进行爬取, 生成该 Web 应用程序的 HTTP 请求报文中的 URL 和 Web 应用程序执行的 SQL 语句的映射表 (XML 文件). 其运行环境如图 2 所示.

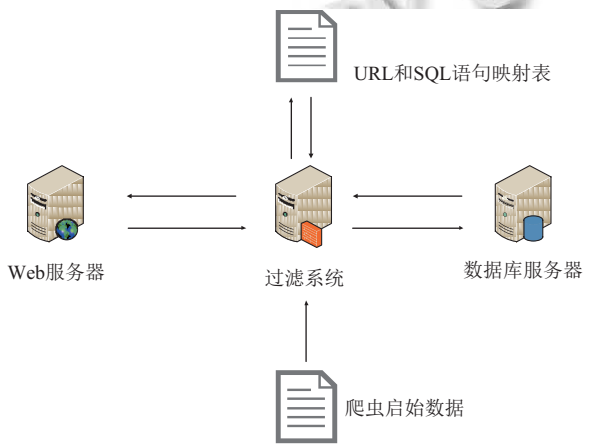

图 2 学习阶段运行环境

以图 1 为例, 假设用户发送的 URL 为 http://www.test.com/login.asp?username=admin&password=123456&type=admin, 则之后 Web 应用程式执行的 SQL 语句为: select * from accounts where username='admin' and password='123456' and type='admin'. 过滤系统先对该 SQL 语句进行解析生成对应的语法树, 之后将语法树的结点与爬虫发出的 HTTP 报文中的数据进行对比, 其数据发出时必须不重复, 并且不包含有 SQL 关键字如 select, and, or 等. 若数据一致并且该数据在原 SQL 语句中被''(一对单引号) 所引用, 则该数据类型为字符型, 将 SQL 语句中对应的数据改为该数据对应的 Http 数据报文中的参数名. 若数据一致, 该数据为数字并且在原 SQL 语句中没有被''所引用, 则该数据类型为数字型, 将 SQL 语句中对应的数据改为该数据对应的 Http 数据报文中的参数名并在该参数名两侧加上*符号. 据此将 Web 生成的 SQL 语句改为: select * from accounts where username='username' and password='passwor d' and type='type'. 若该 url 会根据用户输入的执行不同的 SQL 语句, 则需在相应的启始文件写入各种情况的启始数据并将该对应的<param>的 change 属性设置为 true. 生成的映射表 (XML 文件) 利用选择标签<if>和<elseif>分情况存储 SQL 语句. 假设上面的例子根据 type 属性执行不同的 SQL 语句, 则图 3 和图 4 分别给出了相对应的启始文件和 URL 与 SQL 语句映射表的 XML 文件. 表 1 和表 2 分别给出了启始文件的标签和映射表标签. 其中图 3 的 sql 标签的元素 where username ='username' and password = 'password'中, 其中对 SQL 的抽取的原理如下.

```
<host host="www.test.com">
    <url url="/login.asp">
        <type>get</type>
        <param name="username">admin</param>
        <param name="password">123456</param>
        <param name="type" change=true>admin</param>
    </url>
    <url url="/login.asp">
        <type>get</type>
        <param name="username">user</param>
        <param name="password">123456</param>
        <param name="type" change=true>user</param>
    </url>
</host>
```

图 3 启始文件数据

```
<host host="www.test.com">
   <url url="/login.asp">
    <if test="type=='admin'">
       <type>get</type>
       <sql feature="13" prase="where">
       where username='username' and password='password' and type='type'</sql>
    </if>
    <elseif test="type=='user'">
       <type>get</type>
       <sql feature="13" prase="where">
       where username='username' and password='password' and type='type'</sql>
    </elseif>
   </url>

</host>
```

图 4 URL 和 SQL 语句映射表

表 1 爬虫启始数据 XML 文件标签

标签	属性	说明
<host>	host	<host>标签定义请求的主机名, 主机名由host属性给定
<url>	url	<url>标签定义请求url, 请求url由标签中的url属性给定
<type>		<type>标签定义请求方法, 请求方法由元素给定, 例如post、get
<param>	name change	<param>标签定义http报文中的参数和值, 参数由name属性给定, 值由标签属性给定, change属性表示该url是否会根据该参数的值执行不同的SQL语句

表 2 映射表标签

标签	属性	说明
<host>	host	<host>标签定义请求的主机名, 主机名由host属性给定
<url>	url	<url>标签定义请求URL, 请求URL由标签中的url属性给定
<type>		<type>标签定义请求方法, 请求方法由元素给定, 例如post、get
<sql>	feature prase	<sql>标签定义该url对应的抽取的SQL语句、SQL语句特征和解析启始结点, 抽取的SQL语句由元素给定, SQL语句特征由feature属性给定, 解析启始结点由prase属性给定
<if>	test	<if>标签为选择标签, 其中test为选择条件, 为真则采用标签元素的内容
<else>		<else>标签为选择标签, 若<if>标签中的test为否, 则采用<else>元素中的内容
<elseif>	test	<elseif>标签为选择标签, 若该标签中的test为真, 则采用<elseif>元素中的内容

SQL 注入攻击往往会改变原有的 SQL 语句语法树的结构. 以 SQL 语句“select a from b where c=?”(其中? 为占位符) 为例, 对于正常的用户输入, 假设用户输入为 admin, 则其 SQL 语法树结构如图 5 所示. 对于 SQL 注入攻击, 假设用户输入为 admin ‘or’ 1 ‘=’ 1 , 则其 SQL 语法树结构如图 6 所示. 由图 5 和图 6 中的语法树结构可见, 由于 SQL 语句“select a from b where c=?”(其中? 为占位符) 中用户输入的数据位于 where 子树, 所以其它 SQL 语法树中并没有改变结构. 因此对 SQL 的抽取没有必要抽取整个 SQL 语句, 只需根据用户输入数据所对应的位置, 对该子树进行解析并记录下该子树的根节点和特征 (该子树的结点总数), 抽取对应的 SQL 语句. 以 SQL 语句“select a from b where c=?”(其中? 为占位符) 为例, 则对应抽取的 SQL 语句为: where c=?(其中? 为占位符, 根据与爬虫发出的数据的对比结果确定的 Http 数据报文中的参数名), 并记录该子树根结点为 where, 特征为 4.

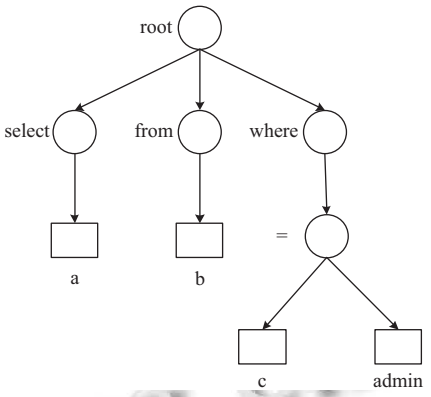

图 5 正常的 SQL 语法树

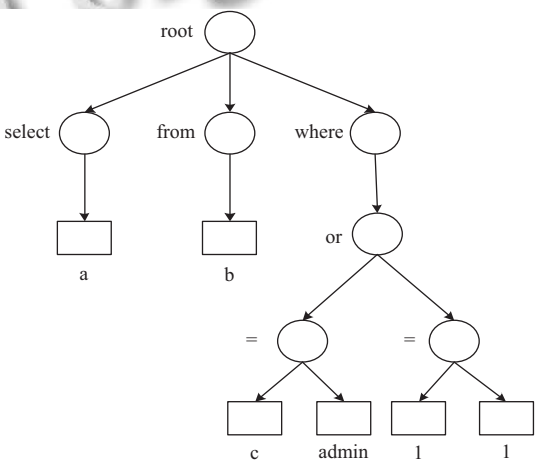

图 6 注入攻击 SQL 语法树

2.2.2 过滤阶段

过滤阶段工作在现实环境下, 通过检测长度、连接频率, 并根据映射表和用户的输入构建 SQL 语句, 再对该 SQL 语句进行解析获取特征值, 之后将该特征值与映射表的 SQL 语句特征进行匹配, 以此来检测 SQL 注入攻击. 其工作环境如图 7 所示.

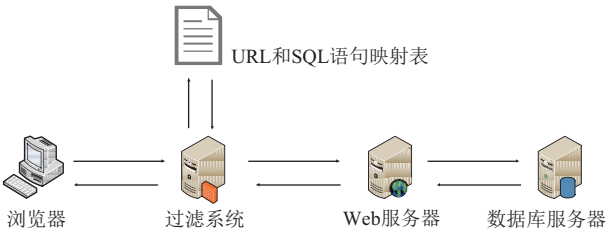

图 7 注入攻击 SQL 语法树

在 HTTP 的 get 请求方式中, SQL 注入攻击过程往往需要向 URL 中拼接 SQL 语句, 所以 SQL 注入的 URL 长度相较于正常的 URL 长度会有所区别. 因此

当 HTTP 的 URL 长度超过某个阈值时, 则该用户输入可能为 SQL 注入攻击, 需要进行进一步判断. 文献[7]中研究表明, 大部分的 HTTP 的 get 请求方式中 URL 的长度普遍在 5–70 byte 之间, 所以本文设置 URL 长度阈值为 70 byte, 当 URL 长度大于 70 时, 则记录该请求由人工进行进一步判断.

SQL 注入攻击的 HTTP 连接频率往往高于正常用户的 HTTP 连接频率. 所以, 单位时间内 HTTP 请求的三元组<源 IP, 目的 IP, 目的端口>的频率能够作为检测 SQL 注入的一个标准. 文献[7]中研究表明, 正常网络的 HTTP 连接频率为 4.87 次/s, 但其中大部分连接频率远低于该值. 因此, 本文将连接频率的阈值定为 3 次/s. 检测中, 如果某个三元组的连接频率大于 3 次/秒, 则记录该请求由人工进行进一步判断.

SQL 注入攻击往往会改变原有的 SQL 语法树结构. 因此, 可以根据用户的输入提取映射表中对应的 SQL 语句信息, 之后根据相应的信息进行 SQL 解析. 再与映射表的 SQL 语句特征进行匹配, 若匹配成功则为正常输入. 若匹配失败则为 SQL 注入攻击. 以上文中的图 4 为例, 假设用户的正常输入为 admin 和 123456, 则对应的解析后的 SQL 语法树子树如图 8 所示. 假设 SQL 注入攻击的输入为 admin 和 123456’or’1’=’1, 则对应的解析后的 SQL 语法树子树如图 9 所示. 由此可见用户正常输入的 SQL 语法树子树的特征与 SQL 注入攻击的 SQL 语法树子树的特征有所区别. 因此, 可以根据 SQL 语法树子树的特征来检测 SQL 注入攻击.

3 SQL 注入过滤方法详细流程及模块介绍

3.1 学习阶段流程

本文提出的 SQL 注入过滤方法的学习阶段的具体流程描述如下:

步骤 1. 读取用户提供的爬虫启始数据放入待抓取 URL 队列并生成 URL 与 SQL 语句的映射表. 之后读取待抓取 URL 队列的队头数据.

步骤 2. 判断步骤 1 中提取的 URL 是在已抓取 URL 队列中, 若存在则直接结束, 准备读取下一个待抓取 URL. 若不存在则进行步骤 3.

步骤 3. 发送 HTTP 请求, 并根据其中的 URL 和请求方式在映射表中相应的位置生成<url>和<type>标签. 之后获取 SQL 语句并解析成 SQL 语法树.

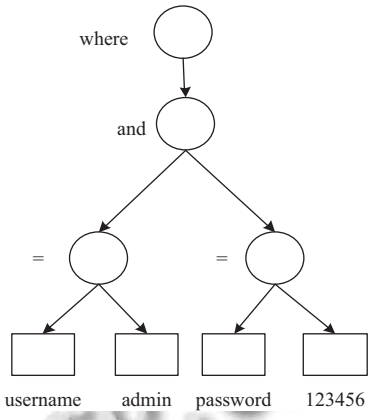

图 8 正常输入的 SQL 语法树中的 where 子树

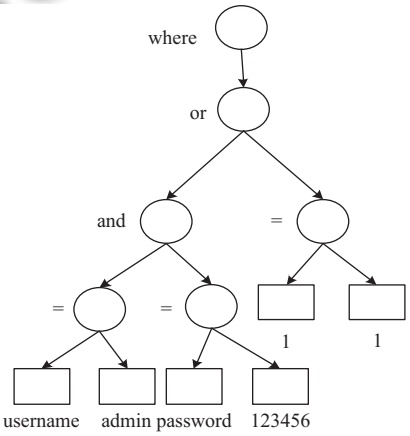

图 9 SQL 注入攻击的 SQL 语法树中的 where 子树

步骤 4. 将步骤 3 生成的 SQL 语法树的叶子结点逐个与 HTTP 报文中参数的数据进行比较. 若数据不一致, 则比较下一个结点; 若所有结点都不一致, 则直接结束, 准备读取下一个待抓取 URL; 若数据一致, 则进行步骤 5.

步骤 5. 判断步骤 4 的数据是否被’’所引用, 若被’’所引用, 则该数据类型为字符型, 将 SQL 语句中对应的数据改为该数据对应的 Http 数据报文中的参数名. 若该数据为数字并且在原 SQL 语句中没有被’’所引用, 则该数据类型为数字型, 将 SQL 语句中对应的数据改为该数据对应的 Http 数据报文中的参数名并在该参数名两侧加上*符号.

步骤 6. 抽取 SQL 语法树, 记录下抽取子树的根结点和特征. 之后在映射表中对应的<url>标签中生成对应的<sql>标签.

步骤 7. 准备读取下一个待抓取 URL, 重复步骤 1 到步骤 6.

3.2 过滤阶段流程

本文提出的 SQL 注入过滤方法的过滤阶段的具体流程描述如下:

步骤 1. 首先, 将对 HTTP 请求方式进行判断. 如果是 GET 方式, 提取 URL 进入步骤 2, 如果是 POST 方式, 提取实体进入步骤 3.

步骤 2. 对 URL 的长度进行检测, 若长度大于 70 byte, 则记录该请求由人工进行进一步判断, 之后进入步骤 3. 否则直接进入步骤 3.

步骤 3. 提取当前 HTTP 请求中的三元组<源 IP, 目的 IP, 目的端口>, 若其不存在于三元组表中, 则创建该三元组. 若存在于三元组表中, 则相应的三元组连接数加 1, 并对该三元组进行连接频率检测, 若连接频率大于 3 次/秒, 则记录该请求由人工进行进一步判断, 之后进入步骤 4. 否则直接进入步骤 4.

步骤 4. 根据 URL 查找学习阶段生成的 URL 和 SQL 语句映射表, 获取相应的 SQL 语句, 子树根结点, 语法树特征等信息. 从 HTTP 请求中提取相应的参数值, 构建 SQL 子句. 根据映射表中的子树根结点, 对该 SQL 子句进行进行对应的 SQL 解析得到 SQL 语法子树. 若解析析失败则进行二次解析; 若解析成功继续步骤 5. 二次解析时过滤模块自动生成完整的 SQL 语句进行解析. 以上文中的 where username='admin' and password='123456' and type='admin'为例, 生成 select a from b where username='admin' and password='123456' and type='admin'进行解析. 若仍解析失败则判断该输入为 SQL 注入攻击, 解析成功则继续步骤 5.

步骤 5. 将步骤 4 生成的 SQL 语法子树的特征 (SQL 语法子树的结点总数) 与映射表中的特征进行匹配. 若一样则将该输入判定为正常输入; 若不一样该则将输入判定为 SQL 注入攻击.

步骤 6. 获取下一条 HTTP 请求, 重复步骤 1 到步骤 5.

3.3 模块介绍

本文提出的方法的设计模块图如图 10 所示.

1) 数据提取模块: 负责提取 HTTP 报文中的数据. 在学习阶段时, 并将相关的数据存入配置文件中. 在过滤阶段, 并结合映射表生成相应的 SQL 语句.

2) 爬虫模块: 该模块包括待抓取 URL 队列和已抓取 URL 队列. 爬虫从待抓取 URL 队列中依次读取 URL, 之后根据 URL 访问对应网页. 同时将已经访问的 URL 发送给已抓取 URL 队列, 以避免重复抓取. 对于刚抓取的网页, 则从中抽出所包含的所有链接信息, 并在已抓起队列中检测, 若发现链接还没被抓取, 则将该 URL 放入待抓取 URL 队列末尾. 直至待抓取 URL 队列为空时才停止抓取网页.

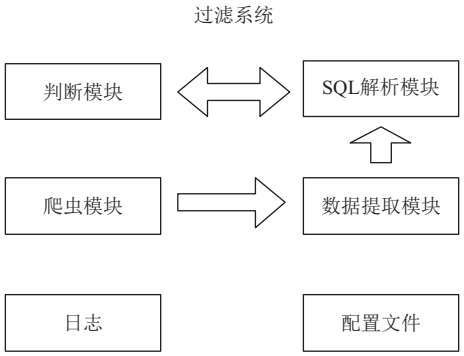

图 10 系统模块图

3) 判断模块: 负责对 HTTP 连接的 URL 长度、连接频率进行检测. 若 URL 长度大于 70 byte 或连接频率大于 3 次/s, 则记录该请求由人工进行进一步判断. 并比较 SQL 解析模块生成的 SQL 语法树的特征值与映射表中的特征值, 根据结果判断该用户输入是否为 SQL 注入攻击.

4) SQL 解析模块: 负责对 SQL 语句进行解析, 生成相应的 SQL 语法树. 该模块包括词法分析和语法分析. 词法分析负责将输入的 SQL 语句进行分词. 语法分析在词法分析的基础上, 判断输入的 SQL 语句是否符合语法逻辑. 当语法分析正常结束后, 则输出相应的抽象语法树.

5) 日志: 负责接收其他模块的记录请求, 并将疑似 SQL 注入攻击的 HTTP 请求记录在日志中.

6) 配置文件: 配置文件包括用户提供的启始文件 (XML 文件) 和 URL 和 SQL 语句映射表 (XML 文件). 其中启始文件包含用户提供的爬虫启始数据. URL 和 SQL 语句映射表中包含了 URL 和 SQL 语句的映射信息.

4 仿真实验与性能分析

本文通过搭建实际的测试平台来检测和分析 LFS 方法的性能. 测试平台使用 tomcat 服务器, 后台数据库采用 MySQL 数据库, Web 应用程序为小型 J2EE 招聘系统, 涵盖登录, 查看简历等功能并代码中存在 SQL 注入漏洞. 测试数据为 SQL 注入攻击输入, 包含 SQL 关键字的用户输入和用户正常输入各 100 条, 其中 SQL

注入攻击输入由 SQLMAP 生成. 为了测试 LFS 过滤方法对于 SQL 注入攻击的拦截率和误报率, 本文对 LFS 过滤, 关键字过滤和正则表达式过滤 (含 30 条过滤规则) 进行了比较. 测试结果如图 11、图 12 和表 3 所示.

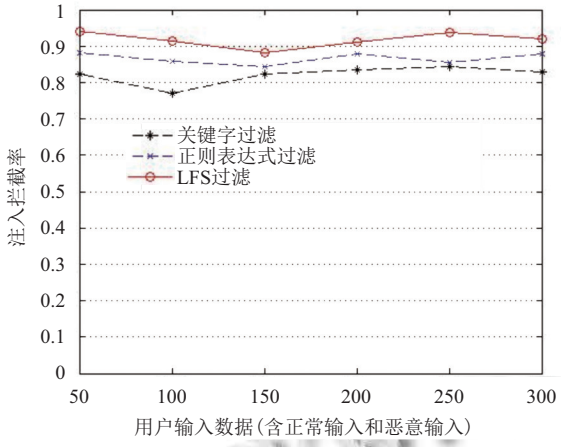

图 11 SQL 注入拦截率

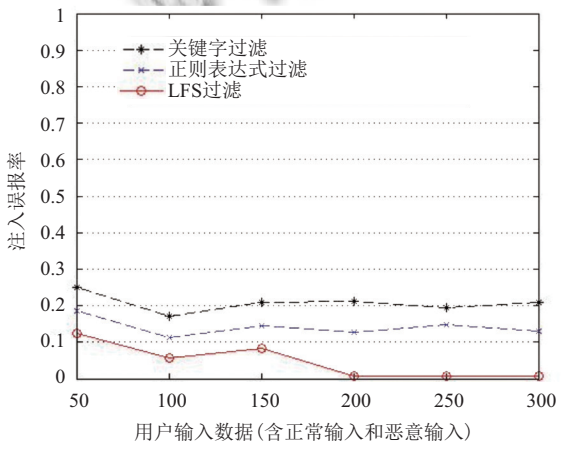

图 12 SQL 注入误报率

表 3 SQL 注入拦截率和误报率

测试技术	拦截率(%)	误报率(%)
关键字过滤	83	21
正则表达式过滤	88	13
LFS过滤	92	7

测试结果表 3、图 11 和图 12 表明了本文提出的 LFS 过滤方法相较于关键字过滤和正则表达式过滤有更高的拦截率和更低的误报率, 能够较好的对 SQL 注入进行防御.

为了测试 LFS 过滤方法对于 Web 应用程序性能的影响, 本文分别对 Web 应用程序未加载过滤模块和 Web 应用程序加载了过滤模块这两种情况分别进行测试, 测试结果如图 13、图 14、图 15 和表 4 所示.

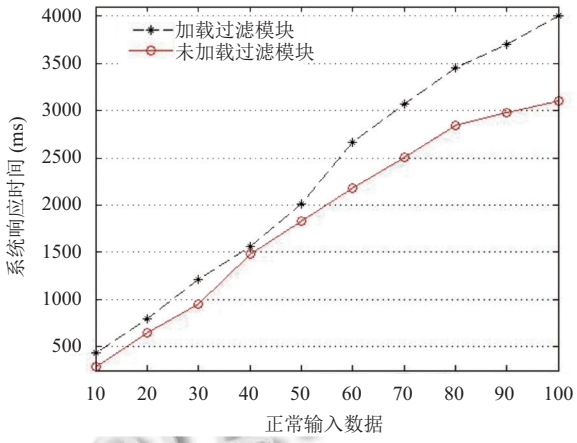

图 13 正常输入测试时间

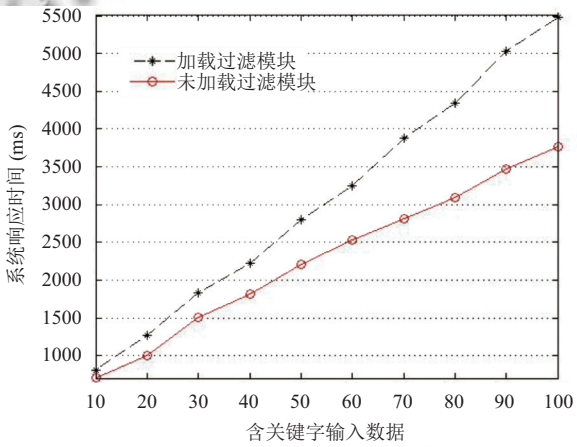

图 14 含关键字的正常输入测试时间

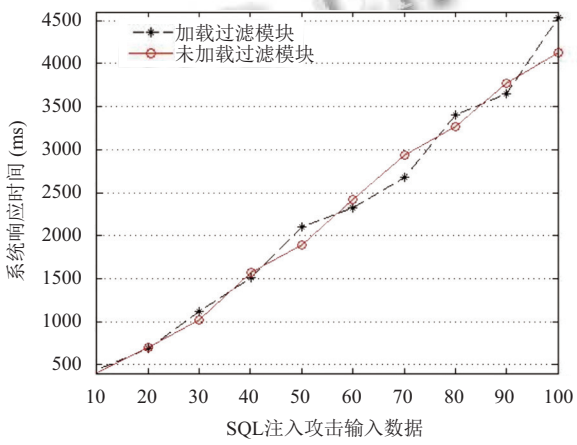

图 15 SQL 注入攻击测试时间

图 13 表明, 对于不含关键字的正常访问, 加载过滤模块与否对 Web 应用程序响应时间有略微的影响, 加载了过滤模块将导致 Web 应用程序响应时间有略微的延迟, 但对用户体验影响不大. 图 14 表明, 对于含有关键字的正常访问, 过滤模块需要对该用户请求进

行 URL 长度和连接频率的检测. 之后需要根据映射表和用户输入的数据进行 SQL 解析, 但是用户输入中含有 SQL 关键字 (如 select、order by 等) 可能会改变了原有的 SQL 语句的语法, 造成 SQL 解析失败, 进行二次解析, 致使系统响应时间较长. 图 15 表明, 对于 SQL 注入攻击, 当没有加载过滤模块时, SQL 注入攻击能够改变 Web 应用程序的 SQL 语句, 使得后台数据库执行该 SQL 语句, 从而导致 Web 应用程序响应时间有明显; 当加载了过滤模块时, 由于要对 url 长度和连接频率进行检测, 并且过滤模块对复杂的 SQL 语句的解析时间较长, 导致系统响应时间增加. 表 4 则给出了加载过滤模块与否, 在三种输入中的平均系统响应时间. 因此对于 SQL 注入攻击, 不论加载了过滤模块与否和系统响应时间都会有所延迟, 导致了两者的系统响应时间较为接近. 因此, 该过滤模块对于系统的响应延迟在可接受的范围内, 对用户体验的影响不大.

表 4 系统响应时间 (单位: ms)

测试内容	未加载过滤模块	加载过滤模块
不含攻击关键字的正常访问	31.04	40.09
含有攻击关键字的正常访问	37.69	54.77
SQL注入攻击	41.30	45.38

5 结束语

论文针对 SQL 注入攻击问题展开研究, 并结合爬虫、数据库代理等技术, 提出了一种新的 SQL 注入过滤方法——LFS 过滤方法. 首先, 通过采用爬虫和数据库代理技术构建 URL 和 SQL 语句映射表, 完成学习过程; 其次, 通过对 URL 长度、访问频率及 SQL 语法树这三个方面进行检测, 实现对用户输入进行过滤, 防止 SQL 注入, 完成过滤过程; 最终, 将学习过程和过滤过程有机结合, 实现了新的 LFS 过滤方法. 仿真实验和性能分析结果表明, 论文提出的 LFS 过滤方法能够有效的防止 SQL 注入攻击, 在拦截率和误报率方面优于关键字过滤方法和正则表达式过滤方法.

然而随着系统业务的不断提高, 应用程序可能会随着用户输入的不同而产生不同的 SQL 语句, 这给本方法带来很大的挑战. 一方面导致本方法在学习阶段可能无法获取所有的 SQL 语句, 另一方面由于 SQL 语句的数量不可控, 可能导致学习阶段产生的 URL 和 SQL 语句映射表数据量巨大, 从而导致在过滤阶段耗时有显著增加. 我们同时注意到可变语句 (随着用户输入的不同而产生不同的 SQL 语句) 的产生有和大相似度, 例如相关的 SQL 语法树结构有很大的相似性, 变化的只有 SQL 语句的局部. 因此, 在未来的研究工作中, 我们考虑采用基于 SQL 语法树的相似度的方法来对学习阶段产生的 URL 和 SQL 语句映射表进行压缩, 以此减少过滤时间.

参考文献

1 郑成兴. 网络入侵防范的理论与实践. 北京: 机械工业出版社, 2006.

2 高鹏, 严望佳. 构建安全的 Web 站点. 北京: 清华大学出版社, 1999.

3 张燕. 数据挖掘提取查询树特征的 SQL 注入攻击检测. 电子技术应用, 2016, 42(3): 90–94.

4 黄保华, 马岩, 谢统义. 用于 SQL 注入检测的语句块摘要树模型. 信息安全与技术, 2012, (3): 34–37.

5 赵宇飞, 熊刚, 贺龙涛, 等. 面向网络环境的 SQL 注入行为检测方法. 通信学报, 2016, 37(2): 88–97. [doi: 10.11959/j.issn.1000-436x.2016034]

6 张志超, 王丹, 赵文兵, 等. 一种基于神经网络的 SQL 注入漏洞的检测模型. 计算机与现代化, 2016, (10): 67–71. [doi: 10.3969/j.issn.1006-2475.2016.10.014]

7 王伟平, 李昌, 段桂华. 基于正则表示的 SQL 注入过滤模块设计. 计算机工程, 2011, 37(5): 158–160.

8 Becher M. Web application firewalls. Akademikerverlag: Universiti Teknologi MARA, 2012.

9 Roesch M. Snort-lightweight intrusion detection for networks. Proceedings of the 13th USENIX Conference on System Administration. Berkeley, CA, USA. 1999. 229–238.

10 田玉杰, 赵泽茂, 王丽君, 等. 基于分类的 SQL 注入攻击双层防御模型研究. 信息网络安全, 2015, (6): 1–6.

11 Saleh AZM, Rozali NA, Buja AG, *et al*. A method for web application vulnerabilities detection by using boyer-moore string matching algorithm. Procedia Computer Science, 2015, (72): 112–121. [doi: 10.1016/j.procs.2015.12.111]

12 Patel N, Shekokar N. Implementation of pattern matching algorithm to defend SQLIA. Procedia Computer Science, 2015, (45): 453–459. [doi: 10.1016/j.procs.2015.03.078]

13 Abawajy J, Fernando H. Policy-based SQLIA detection and prevention approach for RFID systems. Computer Standards & Interfaces, 2015, (38): 64–71.

14 Wei K, Muthuprasanna M, Kothari S. Preventing SQL injection attacks in stored procedures. Proceedings of 2006 Software Engineering Conference. Sydney, Australia. 2006. 191–198.

15 黄龙军. 存储过程技术在网络考试系统 SQL 注入攻击防御上的应用. 计算机系统应用, 2013, 22(1): 103–106.

16 Clarke J. SQL Injection Attacks and Defense. Amsterdam: Elsevier, 2009.