中山大学研究生学刊（自然科学、医学版）
第36卷第4期 JOURNAL OF THE GRADUATES VOL. 36№4
2015 SUN YAT-SEN UNIVERSITY（NATURAL SCIENCES、MEDICINE） 2015

JAVA WEB 安全之 SQL 注入与防范*

狄 俊
（吉首大学，湖南 416000）

【内容提要】简要介绍了 SQL 注入攻击的原理，SQL 注入攻击实现过程，对目前大量使用的攻击“SQL 注入”漏洞的方法进行了演示，并且采用该方法对目前普遍使用的防范“SQL 注入”漏洞的方法进行了检测，通过检测结果最终得出目前防范“SQL 注入”漏洞最为有效的方法。

【关键词】SQL 注入；网络安全；JAVA WEB 安全；SQL 注入防范

1 引言

中国自 1994 年 4 月接入国际互联网开始，中国网民数量呈爆炸式增长。随着用户数量的增长，互联网的安全显得尤为重要。据新浪排名的统计，在诸多 WEB 攻击手段中使用“SQL 注入”手段攻击网站的方式排名第一。2011 年 12 月 21 日，黑客在网上公开了知名程序员网站 CSDN 的用户数据库，高达 600 多万个明文的注册邮箱账号和密码遭到曝光和外泄，成为中国互联网历史上一次具有深远意义的网络安全事故。2014 年 12 月，由于第三方 12306 购票网站的安全性做得不够，导致“SQL 注入”漏洞，导致十三万 12306 账户泄露。防范“SQL 注入”攻击手段在 WEB 安全中显得尤为重要。“SQL 注入”就是通过将恶意 SQL 命令插入到 WEB 表单中或通过域名请求的方式提交到服务器欺骗服务器执行恶意 SQL 命令，而不是按照设计者的意图去执行 SQL 命令。目前至少有 70% 以上的 Web 站点存在着 SQL 注入的缺陷，恶意用户便可以利用服务器、数据库配置的疏漏和精心构造的非法语句通过程序或脚本侵入服务器获得网站管理员的权限和数据库的相关内容，严重的还可以获得整个服务器所在内网的系统信息，它们的存在不仅对数据库信息造成威胁，甚至还可以威胁到系统和用户本身。黑客通过“SQL 注入”可将网站数据库全部拷贝下来，当网站数据库泄露后危害不言而喻。

* 收稿日期：2015－12－16
作者简介：狄俊，男，1995 年 09 月生，湖南岳阳人，吉首大学电子信息科学与技术专业，本科生；E-mail：462499532@qq.com

2 “SQL 注入”

2.1 测试环境介绍

测试基于 Debian Linux 系统，open jdk 8 做 Java RunTime Environment；最新 MySql 作为数据库，默认字符集为 UTF－8；使用 Tomcat8 做 WEB 服务器；全部代码编写均在 Eclipse 上进行；使用 sqlmap “SQL 注入”漏洞检测工具作为辅助检测工具。系统平台的搭建未使用任何 JAVA WEB 框架，采用纯 Servlet + Jsp 的方式搭建，有效避免了框架对测试带来的影响。

2.2 “SQL 注入”原理

假设我们在浏览器中输入 URL www. xxx. com，由于它只是对页面的简单请求无需对数据库动进行动态请求，所以它不存在 “SQL 注入”漏洞，当我们输入 www. xxx. com? id = 1 时，我们在 URL 中传递变量 id，并且提供值为 1，由于它是对数据库进行动态查询的请求（其中? id = 1 表示数据库查询变量），所以我们可以该 URL 中嵌入恶意 SQL 语句[1]。

2.3 “SQL 注入”演示

使用 Jsp + Servlet 技术构建一个简单的动态 WEB 系统，本系统数据库查询代码如下

```
Connection conn;
ResultSet rs;
Statement stm;
Map < String, String > artic = new HashMap < > ();
String sql = “select * from artic where id = ” + id;
Class. forName( “com. mysql. jdbc. Driver”);
conn = DriverManager. getConnection( “jdbc: mysql: //localhost/SQLInjection”, “root”, “root”)
stm = conn. createStatement ();
rs = stm. executeQuery (sql);
while (rs. next ())
{
artic. put( “title”, rs. getString( “title”));
artic. put( “artic”, rs. getString( “artic”));
System. out. println( “执行的 SQL 语句为: ” + sql);
}
```

可以看到该系统从 URL 中获取 id，然后直接执行 SQL 命令，未对用户输入 URL 做任何检测，在这种情况下导致该系统出现 “SQL 注入”漏洞。通过访问 http: //localhost: 8080/SQLinjection/index? id = 1 and 1 = 1 网站访问正常，此时执行的 SQL 语句为 select * from artic where id = 1 and 1 = 1 由数据库特性可知 and 表达式后面的条件成立，语句执行且页面返回正常。当访问 http: //localhost: 8080/SQLinjection/index?

id = 1 and 1 = 2 时，网站开始报错，由于 and 后面条件不成立，所以 SQL 语句没有执行，导致页面无法正常加载，此时可以断定，该系统存在“SQL 注入”漏洞[2]。使用开源工具 sqlmap 检测结果如下（检测命令为在工作目录下：python sqlmap.py -u http://localhost:8080/SQLinjection/index? id = 1）

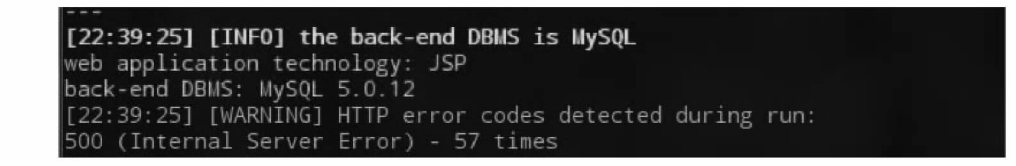

图 1 未做任何防“SQL 注入”措施检测结果

由于存在“SQL 注入”漏洞导致该系统搭建平台及数据库及数据库版本号均已经泄露。

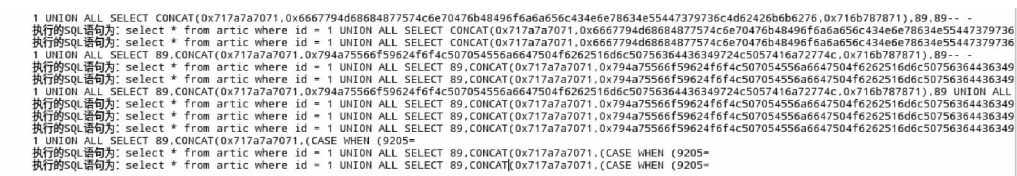

图 2 检测过程中后台输出 SQL 语句

由图二可以看到检测过程中后台输出大量恶意 SQL 代码使用 sqlmap 对该系统作进一步测试（检测命令为在工作目录下：python sqlmap.py -u http://localhost:8080/SQLinjection/index? id = 1 -D SQLInjection - -dump -all）

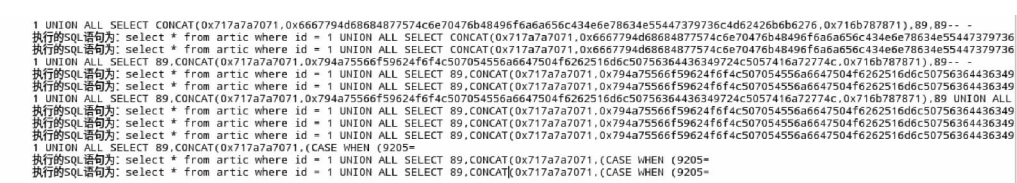

图 3 未做任何防“SQL 注入”措施系统进一步测试结果

由图三可以看到本系统所使用的数据库中的数据均可以通过“SQL 注入”漏洞 dump 出来

3 “SQL 注入”的防范

3.1 “SQL 注入”防范的手段

对用户输入数据进行检测。通过使用正则表达式对用户输入数据进行检测，或者通过对特定的恶意 SQL 指令关键词进行匹配进而过滤恶意指令达到强制要求用户输入符合要求的数据的目的，达到防范“SQL 注入”漏洞的目的[3]。

强制使用参数化语句。在 JAVA WEB 中使用 PreparedStatement 类对执行的 SQL 指令进行预编译，使得不符合要求的 SQL 指令无法执行，不允许在不同的插入时间改变查询的逻辑结构，如源代码中查询语句为“select * from xx where id = ?”若用户输入 id 为恶意 SQL 语句，如 id 为“order by”则此时查询语句则组合为“select * from xx where id = order by”由于使用了 SQL 语句参数化，所以用户输入的恶意 SQL 指令“order by”会当作 id 进行处理，并不会执行从而达到防范“SQL 注入”漏洞的目的[4]。

3.2 “SQL 注入”防范演示

使用第一种方法对用户输入数据采用正则表达式的方法进行检测

```
Connection conn;
ResultSet rs;
Statement stm;
Map < String, String > artic = new HashMap < > ();
if (! id. matches ("(\b (select | update | and | or | delete | insert | trancate | char | into | substr | ascii | declare | exec | count | master | into | drop | execute) \b"))
{
String sql = "select * from artic where id = " + id;
Class. forName ("com. mysql. jdbc. Driver");
Conn = DriverManager. getConnection ("jdbc: mysql: //localhost/SQLInjection", "root", "root");
stm = conn. createStatement ();
rs = stm. executeQuery (sql);
while (rs. next ())
{
artic. put ("title", rs. getString ("title"));
artic. put ("artic", rs. getString ("artic"));
System. out. println ("执行的 SQL 语句为:" + sql);
}
}
```

可以看到在系统执行 SQL 语句之前先对用户输入的 URL 参数 id 进行了正则表达式检测，对于参数 id 中只要出现 SQL 指令就进行过滤。通过访问 http: //localhost: 8080/SQLinjection/index? id = 1 and 1 = 1 网站访问正常。访问 http: //localhost: 8080/

SQLinjection/index? id=1 and 1 = 2 网站访问正常。说明正则表达式对在参数中插入恶意 SQL 指令进行了有效的过滤[5]。但还无法说明网站是否存在 “SQL 注入” 漏洞。再次使用 sqlmap 工具进行检测结果如下：（检测命令为在工作目录下：python sqlmap.py -u http://localhost:8080/SQLinjection/index? id=1）

```
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1 AND 5886=5886

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
    Payload: id=1 AND (SELECT * FROM (SELECT(SLEEP(5)))kJas)

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: id=1 UNION ALL SELECT 80,CONCAT(0x7171766271,0x675579467070
5154414a77704d79427874774569486a726458446a6a654c63646a76477767776569,0x7
17a627a71),80-- -
---
[13:25:32] [INFO] the back-end DBMS is MySQL
web application technology: JSP
back-end DBMS: MySQL 5.0.12
[13:25:32] [WARNING] HTTP error codes detected during run:
```

图4 做了正则表达式过滤后检测结果

从检测结果可以看出，尽管进行了过滤，但是还是存在 “SQL 注入” 漏洞。

强制参数化 SQL 语句后再执行

```
Connection conn;
ResultSet rs;
Map<String, String> artic = new HashMap<>();
String sql = "select * from artic where id = ?";
Class.forName("com.mysql.jdbc.Driver");
conn = DriverManager.getConnection("jdbc:mysql://localhost/SQLInjection",
"root", "root");
PreparedStatement pres = conn.prepareStatement(sql);
pres.setString(1, id);
rs = pres.executeQuery();
while(rs.next()){
artic.put("title", rs.getString("title"));
artic.put("artic", rs.getString("artic"));
System.out.println("执行的 SQL 语句为:" + sql);
}
```

在 SQL 语句执行之前使用 PreparedStatement 类对 SQL 语句参数化，对不符合的 SQL 语句则不进行执行。通过访问 http://localhost:8080/SQLinjection/index? id=1 and 1=1 网站访问正常。访问 http://localhost:8080/SQLinjection/index? id=1 and 1 = 2

网站访问正常。说明使用 PreparedStatement 类参数化 SQL 语句有效[6]。使用 sqlmap 工具进行检测结果如下：（检测命令为在工作目录下：python sqlmap.py -u http://localhost:8080/SQLinjection/index?id=1）

```
[12:59:39] [WARNING] GET parameter 'id' is not injectable
[12:59:39] [CRITICAL] all tested parameters appear to be not injectable.
 Try to increase '--level'/'--risk' values to perform more tests. Also,
you can try to rerun by providing either a valid value for option '--str
ing' (or '--regexp') If you suspect that there is some kind of protectio
n mechanism involved (e.g. WAF) maybe you could retry with an option '--
tamper' (e.g. '--tamper=space2comment')
[12:59:39] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 140 times
```

图 5　参数化 SQL 语句后检测结果

由图五可以看到对于参数化 SQL 语句后，网站已经不存在“SQL 注入”漏洞了。

3.3　“SQL 注入”防范的手段的比较

对于采用对用户输入数据检测的方式防范“SQL 注入”漏洞，由于用户输入的数据具有多样性、不可预测性，存在多种绕过方式。目前普遍存在的绕过方式有：大小写绕过、替换关键字、使用编码编码 URL、使用注释混乱 URL、等价函数与命令等方法绕过过滤，且采用正则表达式过于复杂难以编写、出现错误难于查询原因、过滤的方法存在过滤不严等缺陷。

对于采用 PreparedStatement 类对 SQL 语句参数化经过实验发现这种方法确实能有效的防止“SQL 注入”漏洞的产生。使用 PreparedStatement 类对 SQL 语句参数化，SQL 语句在程序运行前已经进行了预编译，在程序运行时第一次操作数据库之前，SQL 语句已经被数据库分析，编译和优化，对应的执行计划也会缓存下来并允许数据库已参数化的形式进行查询，当运行时动态地把参数传给 PreprareStatement 时，即使参数里有敏感字符如 or 1=1 也数据库会作为一个参数一个字段的属性值来处理而不会作为一个 SQL 指令，所以，这样就能有效的防范“SQL 注入”漏洞[7]。

4　总结

本文从检测用户数据过滤关键词的方法和使用 PreparedStatement 类对 SQL 语句参数化的方法两个常用防范“SQL 注入”漏洞的途径进行了比较，最后得出结论，采用关键词过滤的方法防范“SQL 注入”漏洞虽然有效，但是存在过滤不严、存在多种绕过方式的缺陷。而使用 PreparedStatement 类对 SQL 语句参数化不存在这种缺陷，且能有效的防范“SQL 注入”漏洞。在 JAVA WEB 项目中对于要执行 SQL 语句的部分要先经 PreparedStatement 类对 SQL 语句参数化后再执行。

参考文献:

[1] 高明，辛阳. SQL 注入攻击防范方案的分析与设计 [A]. 2011 年通信与信息技术新进展——第八届中国通信学会学术年会论文集 [C]. 2011

[2] 陈小兵，张汉煜，骆力明，黄河. SQL 注入攻击及其防范检测技术研究 [J]. 计算机工程与应用，2007，11：150 - 152 + 203.

[3] 王云，郭外萍，陈承欢. Web 项目中的 SQL 注入问题研究与防范方法 [J]. 计算机工程与设计，2010，05：976 - 978 + 1016.

[4] 陈小兵，张汉煜，骆力明，黄河. SQL 注入攻击及其防范检测技术研究 [J]. 计算机工程与应用，2007，11：150 - 152 + 203.

[5] 周敬利，王晓锋，余胜生，夏洪涛. 一种新的反 SQL 注入策略的研究与实现 [J]. 计算机科学，2006，11：64 - 68.

[6] 练坤梅，许静，田伟，张莹. SQL 注入漏洞多等级检测方法研究 [J]. 计算机科学与探索，2011，05：474 - 480.

[7] 王云，郭外萍，陈承欢. Web 项目中的 SQL 注入问题研究与防范方法 [J]. 计算机工程与设计，2010，05：976 - 978 + 1016.

JAVA WEB SQL injection attacks and the process of SQL injection attacks

DiJun

(Jishou University，HuNan，416000)

【**Abstract**】This paper briefly introduces the principles of SQL injection attacks and the process of SQL injection attacks. The current commonly used approaches for attacking "SQL injection" vulnerabilities were also demonstrated. Besides，the current commonly used approaches against "SQL injection" vulnerabilities were detected with the same method.

【**Keywords**】SQL injection；network security；WEB JAVA security；SQL injection prevention

【责任编辑：金　龙】