Atitit 防止sql注入 函数过滤函数过滤法

目录

[1.1. 1.escape处理 2](#_Toc14506)

[1.2. 2.使用预编译语句 3](#_Toc29466)

[1.3. 3.使用存储过程 3](#_Toc4610)

[1.4. 4.检查数据类型 4](#_Toc26262)

[1.5. 5.使用安全函数 4](#_Toc16674)

[2. php中等处理 4](#_Toc9236)

虽然可以用mysql\_real\_escape\_string()函数过滤用户提交的值，但是也有缺陷。而使用PHP的PDO扩展的 prepare 方法，就可以避免 sql injection 风险。

mysql\_real\_escape\_string() 函数转义 SQL 语句中使用的字符串中的特殊字符。

下列字符受影响：

* \x00
* \n
* \r
* \
* '
* "
* \x1a

如果成功，则该函数返回被转义的字符串。如果失败，则返回 false。

防数据库攻击的正确做法：

<?php

function check\_input($value)

{// 去除斜杠

if (get\_magic\_quotes\_gpc())

{

$value = stripslashes($value);

}// 如果不是数字则加引号

if (!is\_numeric($value))

{

$value = "'" . mysql\_real\_escape\_string($value) . "'";

}

return $value;

}

SQL注入概念：程序对于用户的输入未作处理就直接放到SQL语句中执行，导致用户输入的特殊字符可以改变语句的原有逻辑，结果可执行任意的SQL语句。

防御主要分为两种：数据类型判断（针对数字型注入）和特殊字符转义（针对字符型注入）

## 1.escape处理

一般会用到两个函数mysql\_real\_escape\_string()和addslashes()

mysql\_real\_escape\_string()函数会转义：' " \r \n NULL Control-Z

addslashes()函数会转义：' " 反斜线和NULL

这种方式对付通过SQL注入绕过登录认证，即所谓的万能密码，特别有效。也可用来对付字符型注入，因为攻击者需要注入单引号来达到引号平衡。

但是，此处需要注意“二次注入“的问题

以PHP为例，当开启了magic\_quotes\_gpc之后，将会对特殊字符转义，比如，将'转义为\'

SQL语句：insert into userinfo(id,username,password) values (1,'$username','$password')

通过网站插入数据：id为1、username为admin' -- 、password为123456，那么SQL语句如下：

insert into userinfo(id,username,password) values (1,'admin\' -- ','123456')

但是secbug\'在插入数据库后却没有“\”

+----+------------+-----------+

| id |  username  |  password |

+----+------------+-----------+

|  1 | admin' --  |   123456  |

+----+------------+-----------+

假设程序允许用户更改密码，用户名为admin' -- ,上面的查询就变成了这样：

update users set password = '[new\_password]' where username = 'admin' -- '

因此攻击者通过注册一个admin' --用户修改了admin的密码

为了预防SQL注入攻击，程序将用户输入进行了转义，但是这些数据却又在“未被转义”的查询窗体中重复使用。

## 2.使用预编译语句

一般来说，防御SQL注入的最佳方式，就是使用预编译语句，绑定变量。

使用预编译的SQL语句，SQL语句的语义不会发生改变，在SQL语句中，变量用？表示，攻击者无法改变SQL的结构，即使攻击者输入试图改变SQL语句结构的输入，程序也仅会将其当作普通字符串来处理。

Java例子：

int id = Integer.parseInt(request.getParameter("id"));

String sql = "select id, username, password from users where id=?";

PreparedStatement ps = this.conn.prepareStatement(sql);

ps.setInt(1,id);

ResultSet res = ps.executeQuery();

Users user = new Users();

if(res.next()){

//封装user对象属性

}

## 3.使用存储过程

存储过程是在大型数据库系统中，一组为了完成特定功能或经常使用的SQL语句集。

存储过程可避免SQL注入，但也可能会存在注入问题，因此应该尽量避免在存储过程中使用动态的SQL语句。如果无法避免，则应该使用严格的输入过滤或者是编码函数来处理用户的输入数据。

CallableStatement cs = connection.prepareCall("{call sp\_getAccountBalance(?)}");

cs.setString(1,custname);

其中sp\_getAccountBalance是预先在数据库中定义好的存储过程。

示例：存储过程如果编写不当，依然有SQL注入的风险

   create proc findUserid @id varchar(100)

as

   exec('select \* from Student where StudentNo='+@id);

go

传入参数3 or 1=1将查询出全部数据，改进代码如下：

   create proc findUserid @id varchar(100)

as

   select \* from Student where StudentNo=@id

go

当再次传入参数3 or 1=1时，SQL执行器会抛出错误：

消息245，级别16，状态1，过程findUserid，第3行

在将varchar值'3 or 1=1' 转换成数据类型int时失败

## 4.检查数据类型

检查输入数据的数据类型，在很大程度上可以对抗SQL注入。

settype($offset,'integer');

$query="SELECT id,name FROM products ORDER BY name LIMIT 20 OFFSET $offset;";

或者

$query=sprintf("SELECT id,name FROM products ORDER BY name LIMIT 20 OFFSET %d;",$offset);

还有：is\_numeric() , ctype\_digit()

## 5.使用安全函数

OWASP ESAPI中的实现：

ESAPI.encoder().encodeForSQL(new OracleCodec(),queryparam);

在使用时，

Codec ORACLE\_CODEC = new OracleCodec();

String query = "SELECT user\_id FROM user\_data WHERE user\_name = '" + ESAPI.encoder().encodeForSQL(ORACLE\_CODEC,req.getParameter("userID")) + "' and user\_password = '" +ESAPI.encoder().encodeForSQL(ORACLE\_CODEC,req.getParameter("pwd")) + "'";

————————————————

# php中等处理

****2、魔术引号****

****说明：****

当php.ini里的magic\_quotes\_gpc=On时。提交的变量中所有的单引号（'）、双引号（"）、反斜线（）与 NUL（NULL 字符）会自动转为含有反斜线的转义字符。

魔术引号（Magic Quote）是一个自动将进入 PHP 脚本的数据进行转义的过程。（对所有的 GET、POST 和 COOKIE 数据自动运行转义）

PHP 5.4 之前 PHP 指令 magic\_quotes\_gpc 默认是 on。

本特性已自PHP 5.3.0 起废弃并将自 PHP 5.4.0 起移除，在PHP 5.4.O 起将始终返回 FALSE。

【技术分享】SQL注入防御与绕过的几种姿势 - 安全客，安全资讯平台.html