本文由于在知乎上搜索到的部分答案，看到网上举例的JAVA例子有点不全，此答案是相对全的答案，而且举了PHP语句的例子，其实在JAVA上也完全可以学到，我们利用的这是该思想。所以通过该文章把这思想给学习到即可。

对于服务器配置层面的防范，应该**保证生产环境的Webserver是关闭错误信息的，**比如PHP在生产环境的配置文件php.ini中的display\_errors应该设置为Off，这样就关闭了错误提示，下面我们更多的从编码的角度来看看如何防范SQL注入。

上篇文章用**两个实例分析了SQL注入攻击的技巧**，可以看到，但凡有SQL注入漏洞的程序，都是因为程序要接受来自客户端用户**输入的变量或URL传递的参数，并且这个变量或参数是组成SQL语句的一部分**，对于用户输入的内容或传递的参数，我们应该要时刻保持警惕，这是安全领域里的「**外部数据不可信任**」的原则，纵观Web安全领域的各种攻击方式，大多数都是因为开发者违反了这个原则而导致的，所以自然能想到的，就是**从变量的检测、过滤、验证下手**，确保变量是开发者所预想的。

1、**检查变量数据类型和格式**

如果你的SQL语句是类似where id={$id}这种形式，数据库里所有的id都是数字，那么就应该在SQL被执行前，检查确保变量id是int类型；如果是接受邮箱，那就应该检查并严格确保变量一定是邮箱的格式，其他的类型比如日期、时间等也是一个道理。总结起来：**只要是有固定格式的变量，在SQL语句执行前，应该严格按照固定格式去检查，确保变量是我们预想的格式**，这样很大程度上可以避免SQL注入攻击。

比如，我们前面接受username参数例子中，我们的产品设计应该是在用户注册的一开始，就有一个用户名的规则，比如5-20个字符，只能由大小写字母、数字以及一些安全的符号组成，不包含特殊字符。此时我们应该有一个check\_username的函数来进行统一的检查。不过，仍然有很多例外情况并不能应用到这一准则，比如文章发布系统，评论系统等必须要允许用户提交任意字符串的场景，这就需要采用**过滤等其他方案了**。

**2、过滤特殊符号**

对于无法确定固定格式的变量，一定要进行特殊符号过滤或转义处理。以PHP为例，通常是采用**addslashes函数**，它会在指定的预定义字符前添加反斜杠转义，这些预定义的字符是：**单引号 (') 双引号 (") 反斜杠 (\) NULL。**

来看2条SQL语句：

$uid = isset($\_GET['uid']) ? $\_GET['uid'] : 0;

$uid = addslashes(uid);

$sql = "SELECT uid,username FROM user WHERE uid=**'{$uid}'**";

以及

$uid = isset($\_GET['uid']) ? $\_GET['uid'] : 0;

$uid = addslashes(uid);

$sql = "SELECT uid,username FROM user WHERE uid={$uid}";

上面两个查询语句都经过了php的addslashes函数过滤转义，但在安全性上却大不相同，在MySQL中，对于int类型字段的条件查询，上面个语句的查询效果完全一样，由于第一句SQL的变量被单引号包含起来，SQL注入的时候，黑客面临的首要问题是必须要先闭合前面的单引号，这样才能使后面的语句作为SQL执行，并且还要注释掉原SQL语句中的后面的单引号，这样才可以成功注入，由于代码里使用了**addslashes函数**，黑客的攻击会无从下手，但第二句没有用引号包含变量，那黑客也不用**考虑去闭合、注释**，所以即便同样采用addslashes转义，也还是存在SQL攻击漏洞。

对于PHP程序+MySQL构架的程序，在动态的SQL语句中，使用单引号把变量包含起来配合addslashes函数是应对SQL注入攻击的有效手段，但这做的还不够，像上面的2条SQL语句，根据「检查数据类型」的原则，uid都应该经过intval函数格式为int型，这样不仅能有效避免第二条语句的SQL注入漏洞，还能使得程序看起来更自然，尤其是在NoSQL(如MongoDB)中，变量类型一定要与字段类型相匹配才可以。

从上面可以看出，第二个SQL语句是有漏洞的，不过由于使用了addslashes函数，你会发现黑客的攻击语句也存在不能使用特殊符号的条件限制，类似where username='plhwin'这样的攻击语句是没法执行的，但是黑客可以将字符串转为16进制编码数据或使用char函数进行转化，同样能达到相同的目的，如果对这部分内容感兴趣，可以[点击这里查看](https://link.zhihu.com/?target=http%3A//cbb.sjtu.edu.cn/course/database/lab8.htm" \t "_blank)。而且由于SQL保留关键字，如「HAVING」、「ORDER BY」的存在，即使是基于黑白名单的过滤方法仍然会有或多或少问题，那么是否还有其他方法来防御SQL注入呢？

**3、绑定变量，使用预编译语句**

MySQL的mysqli驱动提供了预编译语句的支持，不同的程序语言，都分别有使用预编译语句的方法，我们这里仍然以PHP为例，编写userinfo2.php代码：

<?php

header('Content-type:text/html; charset=UTF-8');

$username = isset($\_GET['username']) ? $\_GET['username'] : '';

$userinfo = array();

if($username){

//使用mysqli驱动连接demo数据库

$mysqli = new mysqli("localhost", "root", "root", 'demo');

//使用问号替代变量位置

$sql = "SELECT uid,username FROM user WHERE username=?";

$stmt = $mysqli->prepare($sql);

//绑定变量

$stmt->bind\_param("s", $username);

$stmt->execute();

$stmt->bind\_result($uid, $username);

while ($stmt->fetch()) {

$row = array();

$row['uid'] = $uid;

$row['username'] = $username;

$userinfo[] = $row;

}

}

echo '<pre>',print\_r($userinfo, 1),'</pre>';

从上面的代码可以看到，我们程序里并没有使用addslashes函数，但是浏览器里运行 *http://localhost/test/userinfo2.php?username=plhwin' AND 1=1-- hack* 里得不到任何结果，说明SQL漏洞在这个程序里并不存在。

**实际上，绑定变量使用预编译语句是预防SQL注入的最佳方式**，使用预编译的SQL语句语义不会发生改变，在SQL语句中，变量用问号?表示，黑客即使本事再大，也无法改变SQL语句的结构，像上面例子中，username变量传递的 *plhwin' AND 1=1-- hack* 参数，也只会当作username字符串来解释查询，从根本上杜绝了SQL注入攻击的发生。

**数据库信息加密安全**

相信大家都还对2011年爆出的CSDN拖库事件记忆犹新，这件事情导致CSDN处在风口浪尖被大家痛骂的原因就在于他们竟然明文存储用户的密码，这引发了科技界对用户信息安全尤其是密码安全的强烈关注，我们在防范SQL注入的发生的同时，也应该未雨绸缪，说不定下一个被拖库的就是你，谁知道呢。

在Web开发中，传统的加解密大致可以分为三种:

1、对称加密：

即加密方和解密方都使用相同的加密算法和密钥，这种方案的密钥的保存非常关键，因为算法是公开的，而密钥是保密的，一旦密匙泄露，黑客仍然可以轻易解密。常见的对称加密算法有：AES、DES等。

2、非对称加密：

即使用不同的密钥来进行加解密，密钥被分为公钥和私钥，用私钥加密的数据必须使用公钥来解密，同样用公钥加密的数据必须用对应的私钥来解密，常见的非对称加密算法有：RSA等。

3、不可逆加密：

利用哈希算法使数据加密之后无法解密回原数据，这样的哈希算法常用的有：md5、SHA-1等。

在我们上面登录系统的示例代码中，$md5password = md5($password); 从这句代码可以看到采用了md5的不可逆加密算法来存储密码，这也是多年来业界常用的密码加密算法，但是这仍然不安全。为什么呢？

这是因为md5加密有一个特点：同样的字符串经过md5哈希计算之后生成的加密字符串也是相同的，由于业界采用这种加密的方式由来已久，黑客们也准备了自己强大的md5彩虹表来逆向匹配加密前的字符串，这种用于逆向反推MD5加密的彩虹表在互联网上随处可见，在Google里使用md5 解密作为关键词搜索，一下就能找到[md5在线破解网站](https://link.zhihu.com/?target=http%3A//www.cmd5.com/" \t "_blank)，把我们插入用户数据时候的MD5加密字符串e10adc3949ba59abbe56e057f20f883e填入进去，瞬间就能得到加密前的密码：123456。当然也并不是每一个都能成功，但可以肯定的是，这个彩虹表会越来越完善。

所以，我们有迫切的需求采用更好的方法对密码数据进行不可逆加密，通常的做法是为每个用户确定不同的密码加盐（salt）后，再混合用户的真实密码进行md5加密，如以下代码：

<?php

//用户注册时候设置的password

$password = $\_POST['password'];

//md5加密，传统做法直接将加密后的字符串存入数据库，但这不够，我们继续改良

$passwordmd5 = md5($password);

//为用户生成不同的密码盐，算法可以根据自己业务的需要而不同

$salt = substr(uniqid(rand()), -6);

//新的加密字符串包含了密码盐

$passwordmd5 = md5($passwordmd5.$salt);

**小结**

**1、不要随意开启生产环境中Webserver的错误显示。  
2、永远不要信任来自用户端的变量输入，有固定格式的变量一定要严格检查对应的格式，没有固定格式的变量需要对引号等特殊字符进行必要的过滤转义。  
3、使用预编译绑定变量的SQL语句。  
4、做好数据库帐号权限管理。  
5、严格加密处理用户的机密信息。**