

# 3 countermeasure strategies against SQL injection attacks

CS201P Computer Security, Winter 2020

Aftab Hussain

University of California, Irvine

## 1. Input Validation

Find out all the different kinds of the special characters that can be used and write a filtering program that blindly catches those characters. May not be that straightforward, as some characters may mean different things in different contexts. For example, a `'` or `#` may be valid constituents of the data itself for certain fields (such as “address”). You’ll need a smart parser to detect the different contexts, and thus would require some implement it on behalf of the developers.

## 2. Escaping Characters in Input

Following from the first strategy, this strategy essentially shifts a part of the developer’s burden to the user by asking users to distinguish contexts using escape characters. For example if an apostrophe is part of a name, the user would need to indicate this fact by adding an escape character (e.g. `\`) before the apostrophe. In a SQL query, this would prevent the SQL parser from treating it as an end quote for a string input.

This strategy is widely used not just in SQL query handling, but most tools that take in user input. In Apache, when working with sql queries embedded in php, this mechanism can be activated by including the following in the configuration file (php.ini):

```
magic_quotes_gpc = on
```

You could also use the PHP built-in function `mysql_real_escape_string($input)` in the PHP code itself as follows:

```
<?php
// Connect
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
    OR die(mysql_error());

// Query
$query = sprintf("SELECT * FROM users WHERE user='%s' AND password='%s'",
    mysql_real_escape_string($user),
    mysql_real_escape_string($password));
?>
```

### 3. Separating data and code in the input

This strategy is the safest among the 3 strategies discussed here. Similar to how `execve()` separates the data and code input by having separate input arguments and essentially creating separate channels for data and code, SQL also provides a similar mechanism to do this separation. It provides the prepared statement. Originally built for performance reasons in compiling database queries efficiently, the prepared statement also assists in this security mechanism of separating data and code. Details of this is discussed in “Section 3.4 Task 4: Countermeasure — Prepared Statement”, of the [SEED SQL Injection Lab](#).

### Reference

Kevin Du, Syracuse University, SQL Injection Attack Lecture  
<https://www.youtube.com/watch?v=P8HCLkDInA&feature=youtu.be>