What is SQL injection and how to prevent it?

2 min read · Last change October 18, 2017

When working with databases, one of the most common security vulnerabilities in web applications is definitely SQL injection attacks. Malicious users can insert SQL queries into inputs handled by code that interacts with datbases in order to cause unwanted behavior.

SQL injection example with PDO

```
// GET data is sent through URL: http://example.com/get-user.php?id=1 OR id=2; $id = $_GET['id'] ?? null; // You are executing your application as usual // Connect to a database $dbh = new PDO('mysql:dbname=testdb;host=127.0.0.1', 'dbusername', 'dbpassword'); // Select user based on the above ID // bump! Here SQL code GET data gets injected in your query. Be careful to avoid // such coding and use prepared statements instead $sql = "SELECT username, email FROM users WHERE id = " . $id; foreach ($dbh->query($sql) as $row) { printf ("%s (%s)\n", $row['username'], $row['email']); }
```

Just imagine worst case scenarios with injected SQL:

```
"'DELETE FROM users */"
```

How to avoid SQL injection as per the example above? Use prepared statements:

```
$sql = "SELECT username, email FROM users WHERE id = :id"; $sth = $dbh->prepare($sql, [PDO::ATTR_CURSOR => PDO::CURSOR_FWDONLY]); $sth->execute([':id' => $id]); $users = $sth->fetchAll();
```

mysqli example

When using a MySQL database, you can also use mysqli with prepared statements, or the `mysqli_real_escape_string()` function, but you can also just use PDO instead.

```
// get data is sent through url for example, http://example.com/get-user.php?id=1 OR id=2; $id = $_GET['id'] ?? null; // in your code you are executing your application as usual $mysqli = new mysqli('localhost', 'db_user', 'db_password', 'db_name'); if ($mysqli->connect_error) {
```

```
die('Connect Error (' . $mysqli->connect_errno . ') ' . $mysqli->connect_error); } // bump! sql injected code gets inserted here. Be careful to avoid such coding // and use prepared statements instead $query = "SELECT username, email FROM users WHERE id = " . $id; if ($result = $mysqli->query($query)) { // fetch object array while ($row = $result->fetch_row()) { printf ("%s (%s)\n", $row[0], $row[1]); } // free result set $result->close(); } else { die($mysqli->error); }
```

Let's fix this with prepared statements. They are more convenient because `mysqli_real_escape_string()` doesn't apply quotes (it only escapes it).

```
// Get data is sent through url for example, http://example.com/get-user.php?id=1 OR id=2; $id = $_GET['id'] ?? null; // In your code you are executing your application as usual $mysqli = new mysqli('localhost', 'db_user', 'db_password', 'db_name'); if ($mysqli->connect_error) { die('Connect Error (' . $mysqli->connect_errno . ') ' . $mysqli->connect_error); } // bump! sql injected code gets inserted here. Be careful to avoid such coding // and use prepared statements instead $query = "SELECT username, email FROM users WHERE id = ?"; $stmt = $mysqli->stmt_init(); if ($stmt->prepare($query)) { $stmt->bind_param("i", $id); $stmt->execute(); $result = $stmt->get_result(); while ($row = $result->fetch_array(MYSQLI_NUM)) { printf ("%s (%s)\n", $row[0], $row[1]); } }
```

See also

Other useful reading to check out:

- Prepared Statements in PHP MySQLi to Prevent SQL Injection
- OWASP
- SQL injection - a community paradoxon
- Bobby Tables - A guide to preventing SQL injection.