SQL Injection Attack prevention In Website

The only sure way to prevent SQL Injection attacks is input validation and parametrized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors can be used with SQL Injection to gain information about your database.

If you discover an SQL Injection vulnerability, for example using an Acunetix scan, you may be unable to fix it immediately. For example, the vulnerability may be in open source code. In such cases, you can use a web application firewall to sanitize your input temporarily.

How to Prevent SQL Injections (SQLi) – Generic Tips

Preventing SQL Injection vulnerabilities is not easy. Specific prevention techniques depend on the subtype of SQLi vulnerability, on the SQL database engine, and on the programming language. However, there are certain general strategic principles that you should follow to keep your web application safe.

Step 1: Train and maintain awareness

To keep your web application safe, everyone involved in building the web application must be aware of the risks associated with SQL Injections. You should provide suitable security training to all your developers, QA staff, DevOps, and SysAdmins. You can start by referring them to this page.

Step 2: Don’t trust any user input

Treat all user input as untrusted. Any user input that is used in an SQL query introduces a risk of an SQL Injection. Treat input from authenticated and/or internal users the same way that you treat public input.

Step 3: Use whitelists, not blacklists

Don't filter user input based on blacklists. A clever attacker will almost always find a way to circumvent your blacklist. If possible, verify and filter user input using strict whitelists only.

Step 4: Adopt the latest technologies

Older web development technologies don't have SQLi protection. Use the latest version of the development environment and language and the latest technologies associated with that environment/language. For example, in PHP use PDO instead of MySQLi.

Step 5: Employ verified mechanisms

Don't try to build SQLi protection from scratch. Most modern development technologies can offer you mechanisms to protect against SQLi. Use such mechanisms instead of trying to reinvent the wheel. For example, use parameterized queries or stored procedures.

Step 6: Scan regularly (with Acunetix)

SQL Injections may be introduced by your developers or through external libraries/modules/software. You should regularly scan your web applications using a web vulnerability scanner such as Acunetix. If you use Jenkins, you should install the Acunetix plugin to automatically scan every build.

PHP

PHP is a little more disorganized than how Perl handles parameters. The standard MySQL extension doesn't support parameterization, but that extension has been out of date for more than five years and you should definitely use one of the alternatives instead. The PostgreSQL extension does:

```
$result = pg_query_params( $dbh, 'SELECT * FROM users WHERE email = $1', [$email] );
```

Note that the query must be in single quotes or have the `$` escaped to avoid PHP trying to parse it as a variable. (Actually, in this case PHP will not see `$1` as a variable and will not interpolate it, but for the sake of good practice, single-quote any strings with dollar signs that you want to keep as dollar signs.

However, you should probably be using an abstraction layer.
The ODBC and PDO extensions both support parameterization and multiple databases:

Using mysqli

The MySQL Improved extension handles bound parameters.

```
$stmt = $db->prepare('update people set name = ? where id = ?');
```

```
$stmt->bind_param('si',$name,$id);
```

```
$stmt->execute();
```

Using ADODB

ADODB provides a way to prepare, bind and execute all in the same method call.

```
$dbConnection = NewADOConnection($connectionString);
```

```
$sqlResult = $dbConnection->Execute(
```

```
    'SELECT user_id,first_name,last_name FROM users WHERE username=? AND password=?',
```

```
  [$_REQUEST['username'], sha1($_REQUEST['password'])]
);
```

Using the ODBC layer

```
$stmt = odbc_prepare( $conn, 'SELECT * FROM users WHERE email = ?' );
$success = odbc_execute( $stmt, [$email] );
```

Or:

```
$dbh = odbc_exec($conn, 'SELECT * FROM users WHERE email = ?', [$email]);
$sth = $dbh->prepare('SELECT * FROM users WHERE email = :email');
$sth->execute([':email' => $email]);
```

Using the PDO layer

Here's the long way to do bind parameters.

```
$dbh = new PDO('mysql:dbname=testdb;host=127.0.0.1', $user, $password);
$stmt = $dbh->prepare('INSERT INTO REGISTRY (name, value) VALUES (:name, :value)');
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);

// insert one row
$name = 'one';
$value = 1;
$stmt->execute();
```

And a shorter way to pass things in.

```
$dbh = new PDO('mysql:dbname=testdb;host=127.0.0.1', $user, $password);
```

```
$stmt = $dbh->prepare('UPDATE people SET name = :new_name WHERE id =
:id');
```

```
$stmt->execute( ['new_name' => $name, 'id' => $id] );
```

Applications & Frameworks

CakePHP

When using the MVC framework CakePHP, most of your database communication will be abstracted away by the Model API. Still, it is sometimes necessary to perform manual queries, which can be done with Model::query. In order to use prepared statements with that method, you just need to pass an additional array parameter after the SQL query string. There are two variants:

```
// Unnamed placeholders: Pass an array containing one element for each ?
$this->MyModel->query(
  'SELECT name FROM users WHERE id = ? AND status = ?',
  [$id, $status]
);
// Named placeholders: Pass an associative array
$this->MyModel->query(
  'SELECT name FROM users WHERE id = :id AND status = :status',
  ['id' => $id, 'status' => $status]
);
```

This behavior is documented in the CakePHP Cookbook. (It is described for the `fetchAll()` -method, but `query()` uses `fetchAll()` internally).

Fat-Free

In Fat-Free you are able to easily use free form SQL queries from the DB\SQL class as well as the built in mappers.

```
$db = new DB\SQL(
   'mysql:host=localhost;port=3306;dbname=mysqldb',
   'admin',
   'wh4t3v3r'
);

// Raw SQL fetchAll
$results = $db->exec(
   "SELECT name FROM users WHERE id = ? AND is_active = ?",
   [ $id, $is_active ]
);

// Raw SQL insert/update with named parameters
$db->exec("INSERT INTO users (name, email) VALUES (:name, :email)", [ ':name' => $name, ':email' =>
$email ]);

// insert used with the mapper
```

```
$user = new \DB\SQL\Mapper($db, 'users');
$user->name = 'Bobby Tables';
$user->email = 'bobby@bobby-tables.com';
$user->save();

// update
$user = new \DB\SQL\Mapper($db, 'users');
$user->load([ "id = ?", $id ]);
$user->name = 'Momma Db';
$user->save();
```

That all of code for PHP website. if use this all of code for your website that will be safe for SQLI attacker.

I'm also using all of this code.

In This Login Page When I Try To With (''''') These Like 'Lima And Click Login I Found A Error Like My SQL,

This Is The Error I Have. Its Called To SQL Injection .If A Sqli Hacker Find This Try To Pass Some Database Query .If Some Database Query Pass Here They Can Do A Lot Of Thing In This Website. They Can Even Drop Or Delete Our Whole Database. And Access The Admin Panel And Change The Whole Data Or Destroy Your Database

Fatal error: Uncaught mysqli_sql_exception: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'lima' AND password = 'd41d8cd98f00b204e9800998ecf8427e'' at line 1 in C:\xampp\htdocs\food-order\admin\login.php:64 Stack trace: #0 C:\xampp\htdocs\food-order\admin\login.php(64): mysqli_query(Object(mysqli), 'SELECT * FROM t...') #1 {main} thrown in **C:\xampp\htdocs\food-order\admin\login.php** on line **64**

Now Go The Visual Studio Where I Write Code My Website Login Page And Go There For Some Changes ,Because I Want My Website For Safety And Also SQLi Attacker Don't Attack My Website.

So That Is The Code Which I Try To Prevent

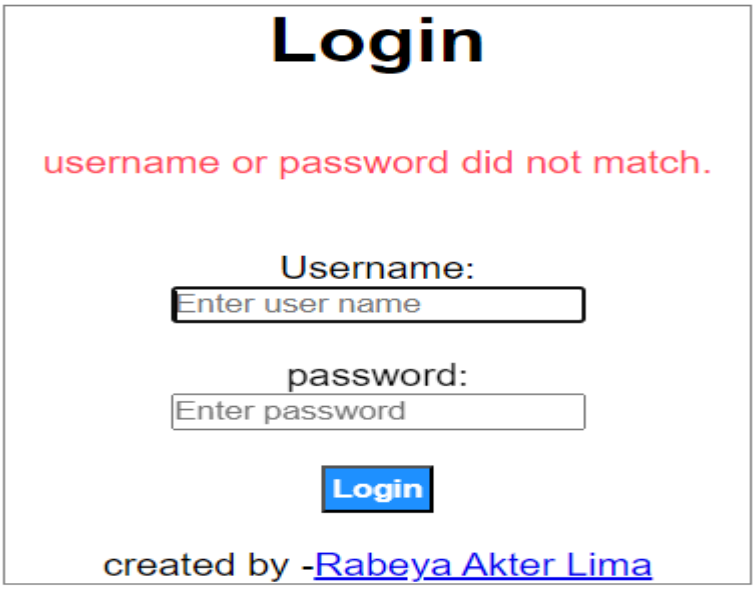
```php
<?php
    //check the submit button is clicked or not
    if(isset($_POST['submit'])){
        //process for Login
        //1.get the data from login form

        $username = $_POST['username'];
        $password = md5($_POST['password']);


        //2. sql to check the user with user and password exists or not

        $sql ="SELECT * FROM tbl_admin WHERE username='$username' AND password = '$password'";
        //execute the query
        $res= mysqli_query($conn, $sql);

        //4. count rows to check whether the user exits or not
        $count =mysqli_num_rows($res);

        if($count==1)
        {
            //user available

            $_SESSION['login']="<div class = 'success'>Login successful</div>";
            $_SESSION['user']=$username;//check the user is loged in or Not and logout willunset it


            //redirect to  to home page/ dashboard
            header("location:".SITEURL.'admin/');

        }
        else
        {
            //user not available and login fail
            //user available

            $_SESSION['login']="<div class = 'error text-center'>username or password did not match.</div>";

            //redirect to  to home page/ dashboard
            header("location:".SITEURL.'admin/login.php');

        }

    }
?>
```

- mysqli::real_escape_string — Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection
- mysqli::query — Performs a query on the database
- mysqli_stmt::$num_rows — Returns the number of rows fetched from the server
- mysqli_result::fetch_assoc — Fetch the next row of a result set as an associative array

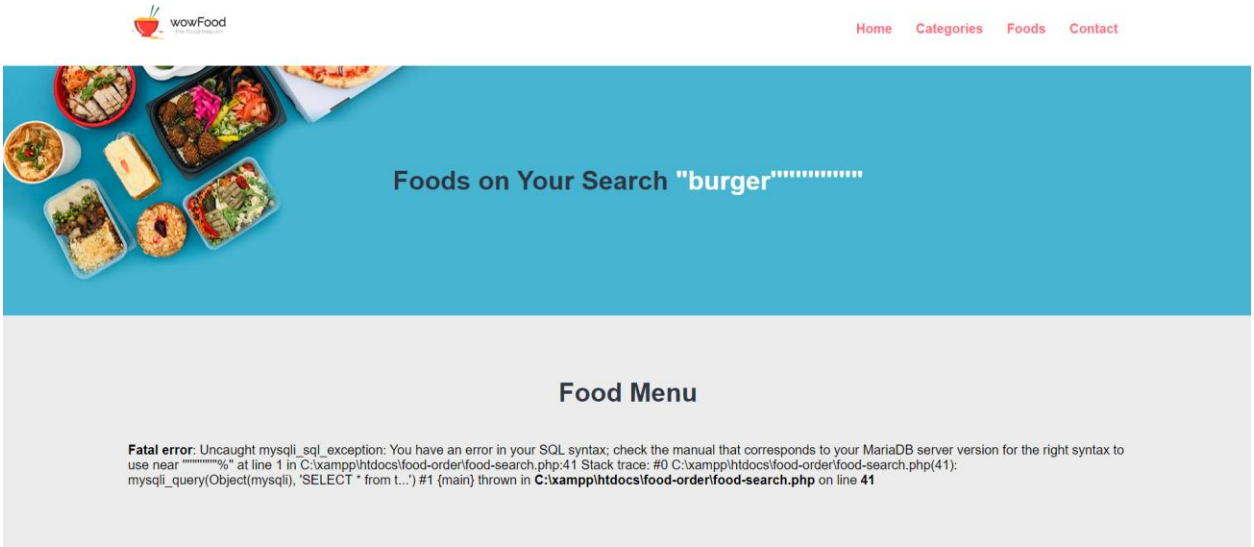
For SQLI Attack Prevention I Use This String

```php
$username = mysqli_real_escape_string($conn, $_POST['username']);

        $raw_password =  md5($_POST['password']);
        $password = mysqli_real_escape_string($conn , $raw_password);
```

When I Use That String There Is Not Showing Any SQL Error. Now If Hacker Try SQLI That Impossible To Try SQLI Attack



In This Website When I Search With (''''') Like Burger''''' And Click **Searchbar** I Found A Error Like My SQL, This Is The Error I Have. Its Called To SQL Injection .If A Hacker Find This Try To Pass Some Database Query .If Some Database Query Pass Here They Can Do A Lot Of Harm In This Website. They Can Even Drop Or Delete Our Whole Database.



So That Is The Code Which I Try To Prevent

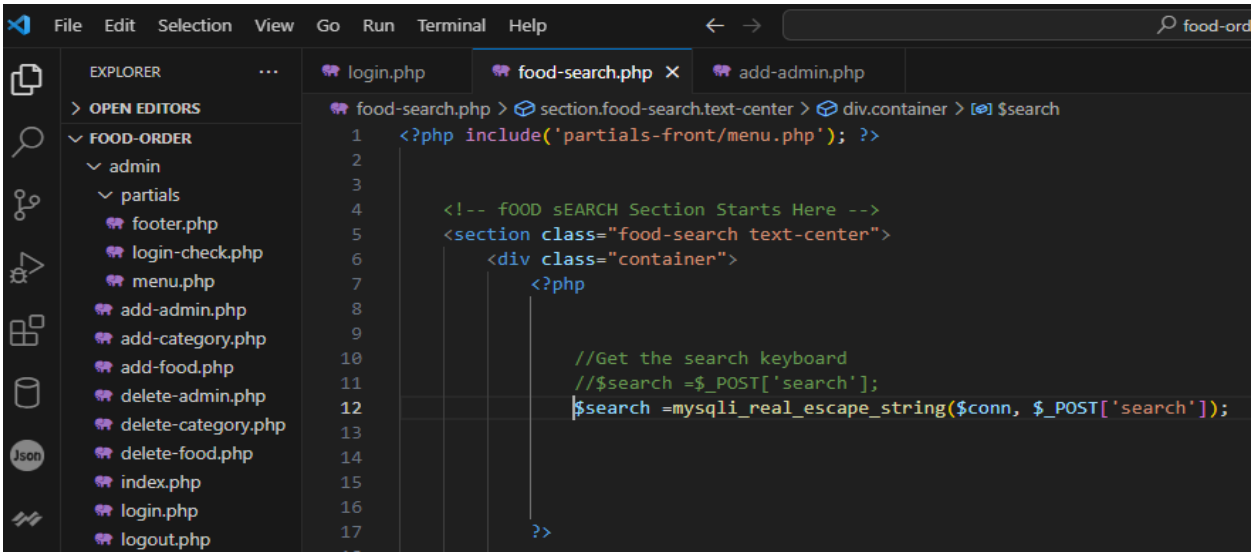
```php
<?php include('partials-front/menu.php'); ?>


    <!-- fOOD sEARCH Section Starts Here -->
    <section class="food-search text-center">
        <div class="container">
            <?php


                //Get the search keyboard
                $search =$_POST['search'];



            ?>
```

For SQLI Attack Prevention I Use This String

```php
$search =mysqli_real_escape_string($conn, $_POST['search']);
```

I Use This String Because

- mysqli::real_escape_string — Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection



When I Use That String There Is Not Showing Any SQL Error. Now If Hacker Try SQLI That Impossible To Try SQLI Attack