Project 1 - Vulnerabilities

Ariana Gemelgo (89194), Gustavo Inácio (85016),
Hugo Moinheiro (84931), Mariana Pinto (84792)

November 2021

Contents

1 Introduction

For this project a Book Database was agreed to be created. A simple website that allows the listing of searched books by name, author or publisher.

The languages used were PHP for the backend, MySQL for the database and HTML with CSS for the frontend. To deploy the website Docker was chosen, creating two files: dockerfile and docker-compose. These are useful to run the application. All the instructions are available to be consulted in the README file.

The asked vulnerabilities and the respective fixes were the main focus for the work, therefore it was ensured that the right features were added for that purpose.

2 CWE-89: SQL Injection

2.1 Vulnerable App

To show this type of vulnerability, a Log In form was implemented. The user is required to type a username and a password in order to enter the main page. For this example, the username `admin` and password `admin` are going to be used. This is a user resident in the database, therefore allowing to enter the account.

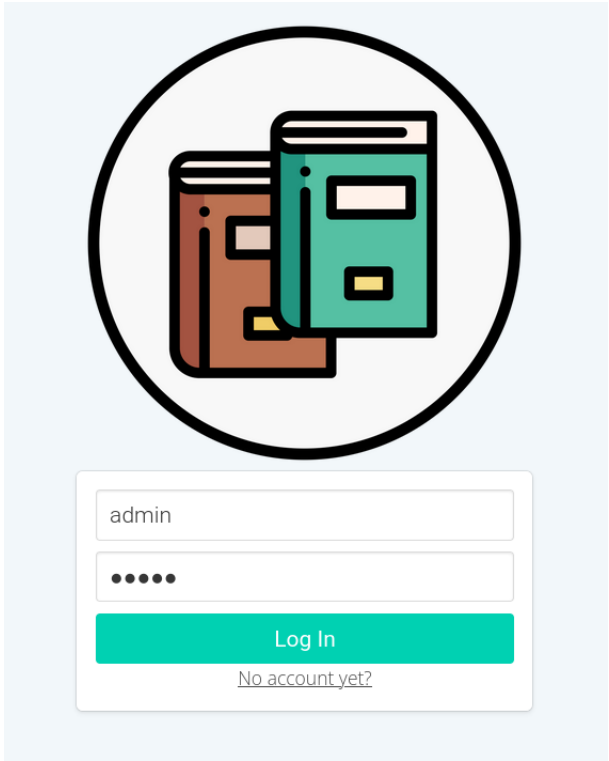

Figure 1: Log In success

On the other hand, if the user is not registered in the database they won't be able to access and an error will show up.

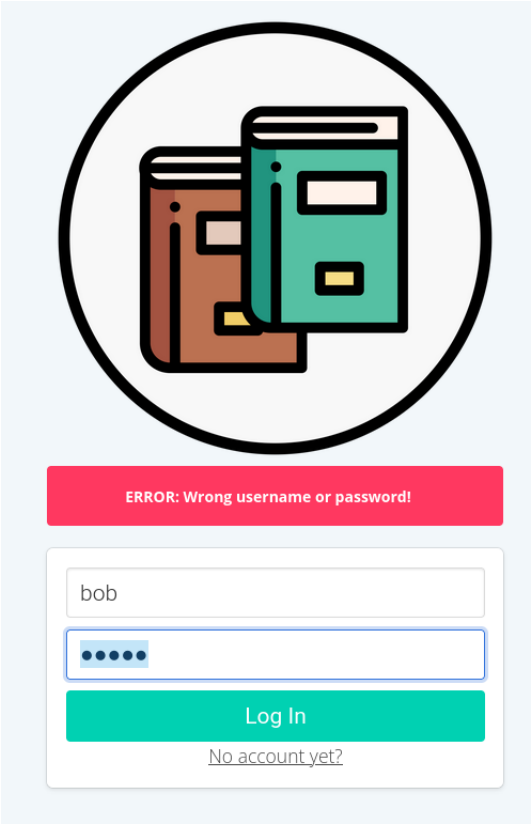

Figure 2: Log In error

This is easily bypassable. If `' or '1'='1` is used as the password with a known username, in this case, `admin`, the access is granted. As well as, using both `' or '1'='1` as username and password.

The vulnerability shown is happening due to two reasons. The password should be hashed and the values given by POST which come directly from what the user typed are being contatenated. Therefore, it is very simple to manipulate the query.

```
$username = $_POST['username'];
$password = $_POST['password'];

$query = "SELECT user_id, username FROM Client WHERE username='{$username}' and password='{$password}'";

$result= mysqli_query($connection, $query);
```

Figure 3: Vulnerable code in Log In form

2.2 Secure App

To fix the vulnerability above, password hashing was added. Now the values given by POST are being checked before giving them to the query. `mysqli_real_escape_string()` was the function chosen for that task, since it removes special characters normally used in SQL Injection.

```
$username = mysqli_real_escape_string($connection,$_POST['username']);
$password = mysqli_real_escape_string($connection,$_POST['password']);

$query = "SELECT user_id, username FROM Client WHERE username='{$username}' and password=md5('{$password}')";

$result= mysqli_query($connection, $query);
```

Figure 4: Secure code in Log In form

3 CWE-79: Cross-site Scripting

3.1 Vulnerable App

To demonstrate this vulnerability the search option of the main page is going to be used. When a user tries to search for a book in the database, if there's a book name, author or publisher that matches the query, the results are presented in a table, so the result of the search can be consulted. By contrast, if no book is found a message will appear informing the user. An empty search displays all the books in the database.

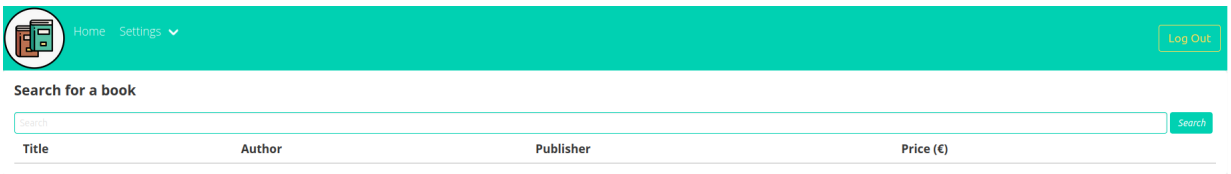

Figure 5: Main page

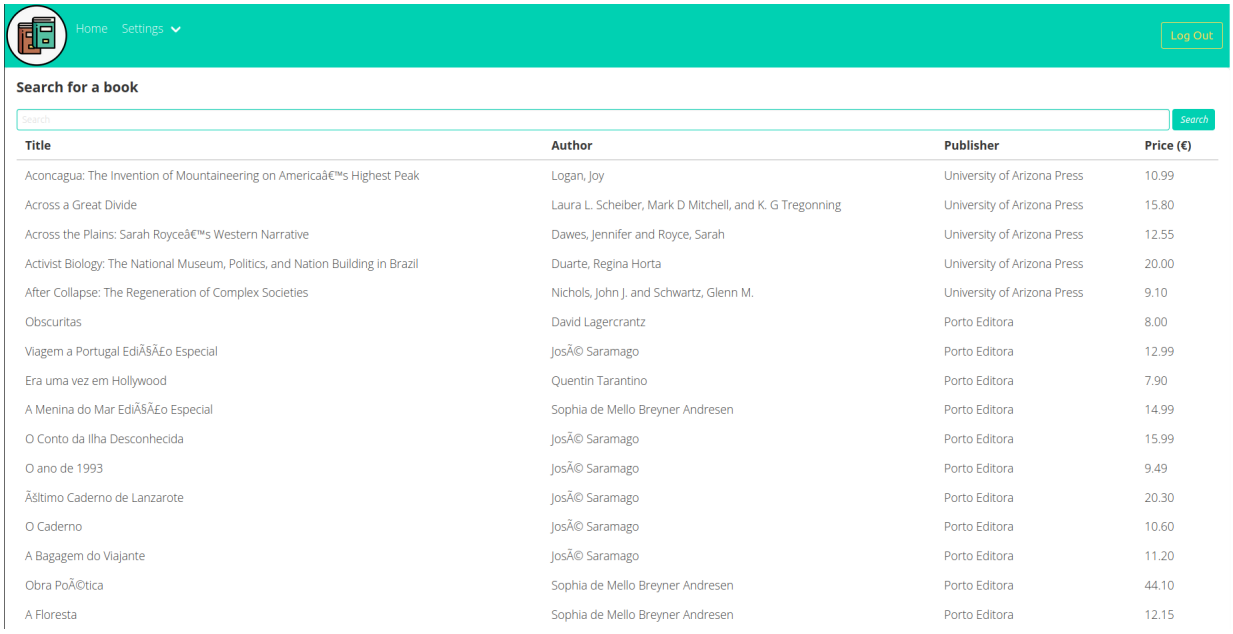

Figure 6: Result of a search with the query "a"

While using the search bar form, a XSS attack can occur. XSS is an attack in which malicious scripts are injected. In this case, while using the search bar, a script like `<img src="https://ichef.bbci.co.uk/images/ic/1024x576/p09j7x4c.jpg">` can be sent, compromising its functionality.

The vulnerability is available because special caracteres are not taken into account and the query is passed without being filtered or verified.

```
else echo '<tbody>
            <tr>
              <td>The book '.substr($book, 1, -1).' was not found in the database.</td>
            </tr>
          </tbody>';
?>
```

Figure 7: Vulnerable code in Search form

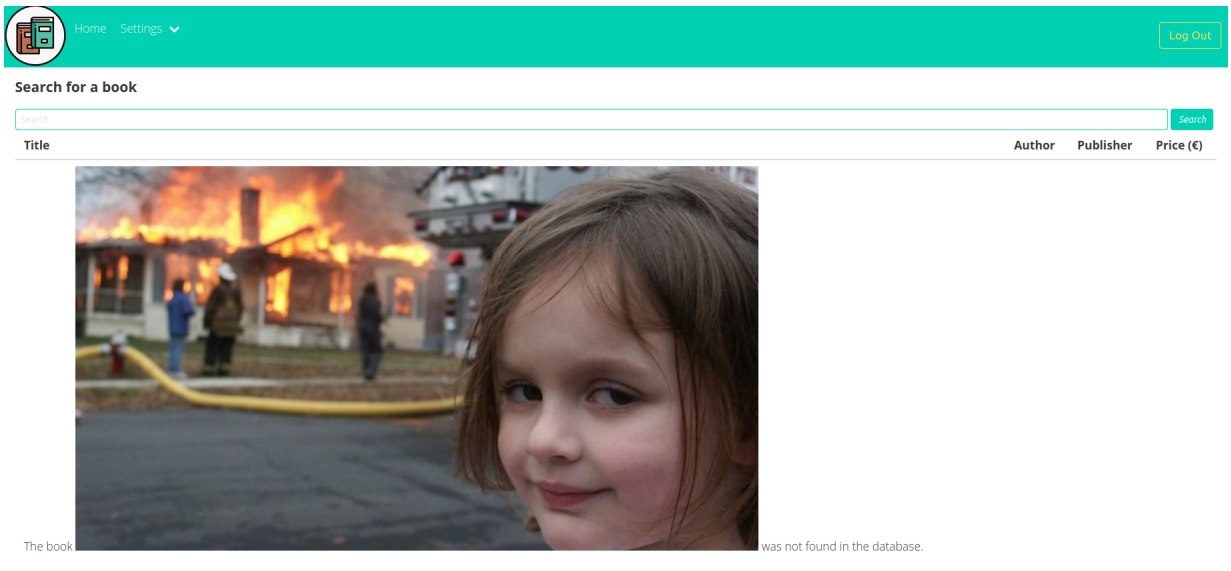

Figure 8: Script Injection Pop up

3.2 Secure App

To prevent the Cross-site Scripting attack, the function `htmlspecialchars()` was picked. What this function does is converting special characters into HTML entities. The script injection is no longer read.

```
else echo '<tbody>
            <tr>
              <td>The book '.htmlspecialchars(substr($book, 1, -1)).' was not found in the database.</td>
            </tr>
          </tbody>';
?>
```

Figure 9: Secure code in Search form

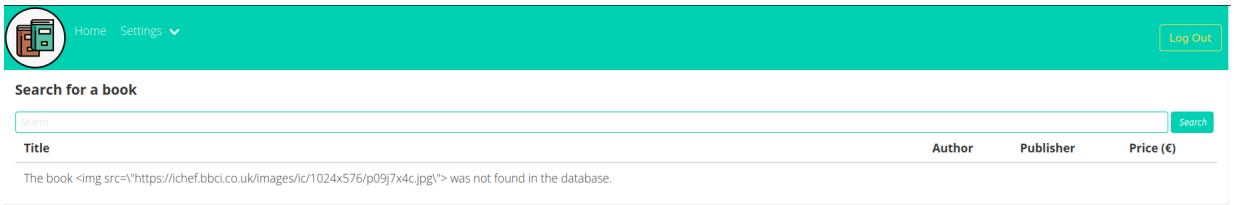

Figure 10: Search form without the vulnerability

4 CWE-521: Weak Password Requirements

4.1 Vulnerable App

Even though password encryption is frequently used, the process of picking a password shouldn't be neglected. If the password is weak and predictable this may be considered a vulnerability.

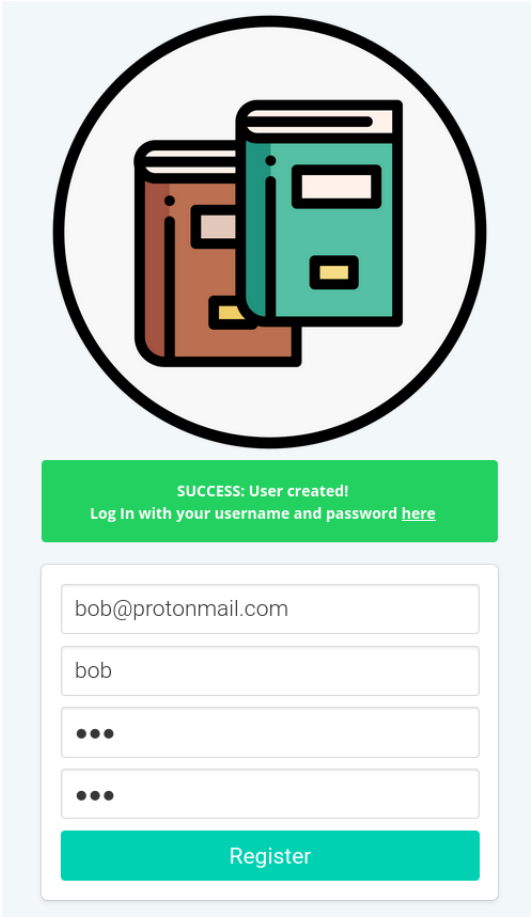

Figure 11: Vulnerable Register form success

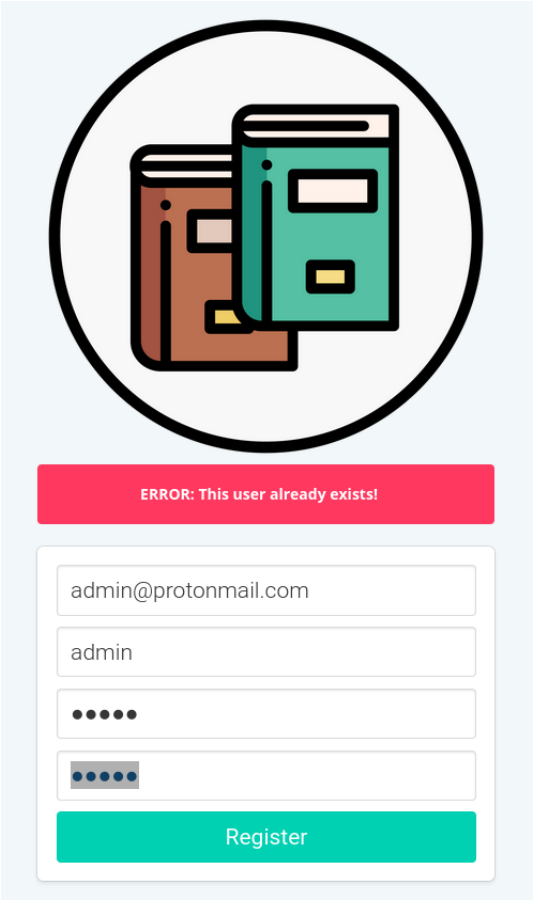

Figure 12: Vulnerable Register form error

In this case, the register form doesn't check if the password has the minimum requirements to be a strong password.

```php
<?php
session_start();
include('db_config.php');

if(empty($_POST['email']) || empty($_POST['username']) || empty($_POST['password']) || empty($_POST['conf_password'])){
    header('Location: register.php');
    exit();
}

$email = trim($_POST['email']);
$username = trim($_POST['username']);
$password = trim($_POST['password']);
$confpass = trim($_POST['conf_password']);

$check = "SELECT COUNT(*) AS total FROM Client WHERE username = '$username'";
$result = mysqli_query($connection, $check);
$row = mysqli_fetch_assoc($result);

if($row['total'] == 1){
    $_SESSION['invalid_user'] = true;
    header('Location: register.php');
    exit;
}

if(strcmp($password, $confpass) != 0){
    $_SESSION['nomatch_password'] = true;
    header('Location: register.php');
    exit;
}

$query = "INSERT INTO Client(username, password, email) VALUES ('$username', '$password', '$email')";
$result = mysqli_query($connection, $query);
$_SESSION['user_created'] = true;

header('Location: register.php');
exit;

?>
```

Figure 13: Vulnerable code Register form

4.2 Secure App

To ensure the user chooses a safe password there must be a warning indicating which are the steps to build a strong password. For that, a regex will be used.

The verification is done by checking if the writen credentials have at least one lowercase letter, one uppercase letter, one number, one special character and the minimum length of 8 characters. Adding this to the previous code, gets the following result.

```php
$email = mysqli_real_escape_string($connection,trim($_POST['email']));
$username = mysqli_real_escape_string($connection,trim($_POST['username']));
$password = mysqli_real_escape_string($connection,trim($_POST['password']));
$confpass = mysqli_real_escape_string($connection,trim($_POST['conf_password']));

$uppercase = preg_match('@[A-Z]@', $password);
$lowercase = preg_match('@[a-z]@', $password);
$number    = preg_match('@[0-9]@', $password);
$specialChars = preg_match('@[^\w]@', $password);

$check = "SELECT COUNT(*) AS total FROM Client WHERE username = '$username'";
$result = mysqli_query($connection, $check);
$row = mysqli_fetch_assoc($result);

if($row['total'] == 1){
    $_SESSION['invalid_user'] = true;
    header('Location: register.php');
    exit;
}

if(strcmp($password, $confpass) != 0){
    $_SESSION['nomatch_password'] = true;
    header('Location: register.php');
    exit;
}

if(!$uppercase || !$lowercase || !$number || !$specialChars || strlen($password) < 8) {
    $_SESSION['password_weak'] = true;
    header('Location: register.php');
    exit;
}
```

Figure 14: Secure code Register form

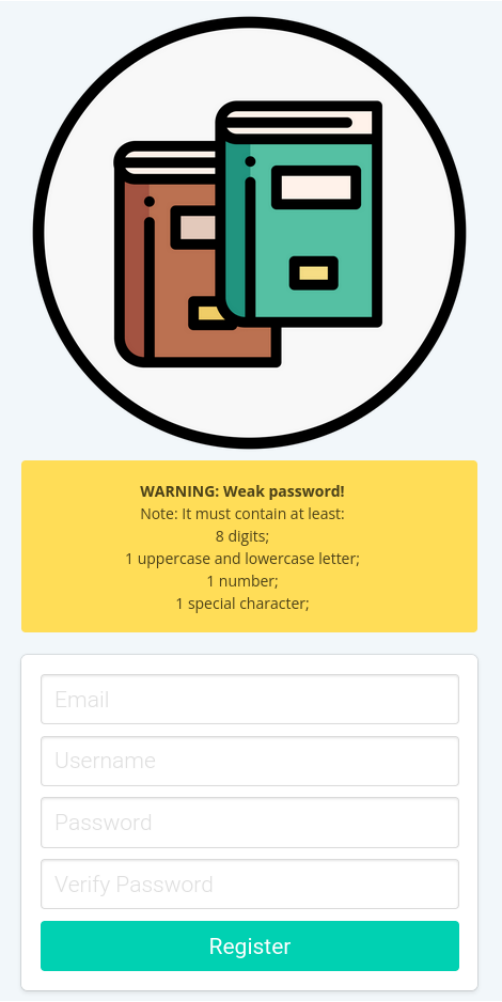

Figure 15: Secure Register form password error

The same was added to the changing password form.

5 CWE-620: Unverified Password Change

5.1 Vulnerable App

After Loging In, the user is able to change their password to a new one, if needed.

In order to do that, the user is asked to introduce the new password and repeat it, to verify if it was typed correctly.

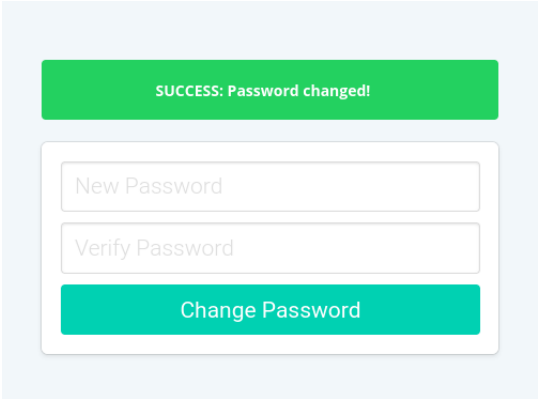

Figure 16: Vulnerable Password Change form success

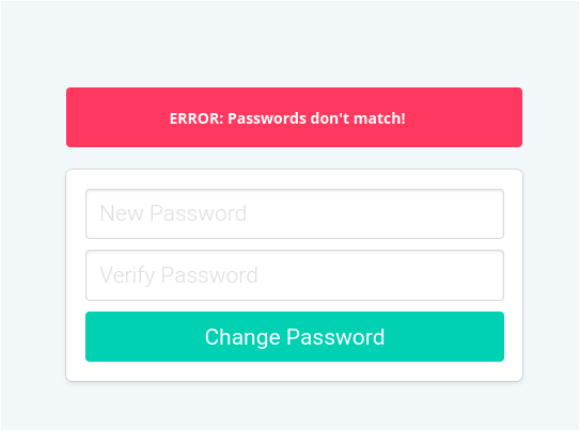

Figure 17: Vulnerable Password Change form error

This constitutes a problem, since there is no way of verifying if the person changing the password is the owner of the account. For instance, in Chapter 2, getting access to the account without knowing the password or the username

is a possibility. The hacker would then be able to change the password, thus gaining the privileges associated with that user.

```php
<?php
session_start();
include('db_config.php');

if(empty($_POST['newpassword']) || empty($_POST['conf_newpassword'])){
    header('Location: pass_page.php');
    exit();
}

$password = trim($_POST['newpassword']);
$confpass = trim($_POST['conf_newpassword']);
$username = $_SESSION['username'];

if(strcmp($password, $confpass) != 0){
    $_SESSION['nomatch_password'] = true;
    header('Location: pass_page.php');
    exit;
}else{
    $query = "UPDATE Client SET password='$password' WHERE username='$username'";
    $result = mysqli_query($connection, $query);
    $_SESSION['password_changed'] = true;
    header('Location: pass_page.php');
    exit;
}

?>
```

Figure 18: Vulnerable code in Password Change form

5.2 Secure App

To solve this problem, an old password authentication must be implemented, making sure the one who is changing the password is the user who made the account registration.

```php
<?php
session_start();
include('db_config.php');

if(empty($_POST['newpassword']) || empty($_POST['conf_newpassword']) || empty($_POST['oldpassword'])){
    header('Location: pass_page.php');
    exit();
}

$oldpassword = md5(mysqli_real_escape_string($connection,trim($_POST['oldpassword'])));
$newpassword = mysqli_real_escape_string($connection,trim($_POST['newpassword']));
$confpass = md5(mysqli_real_escape_string($connection,trim($_POST['conf_newpassword'])));
$username = $_SESSION['username'];

$uppercase = preg_match('@[A-Z]@', $newpassword);
$lowercase = preg_match('@[a-z]@', $newpassword);
$number    = preg_match('@[0-9]@', $newpassword);
$specialChars = preg_match('@[^\w]@', $newpassword);

if(strcmp(md5($newpassword), $confpass) != 0){
    $_SESSION['nomatch_password'] = true;
    header('Location: pass_page.php');
    exit;
}

if(!$uppercase || !$lowercase || !$number || !$specialChars || strlen($newpassword) < 8) {
    $_SESSION['password_weak'] = true;
    header('Location: pass_page.php');
    exit;
}
```

Figure 19: Secure code in Password Change form

Inserting the password `admin` of the account `admin`, previously mentioned, it was possible to switch passwords successfully.

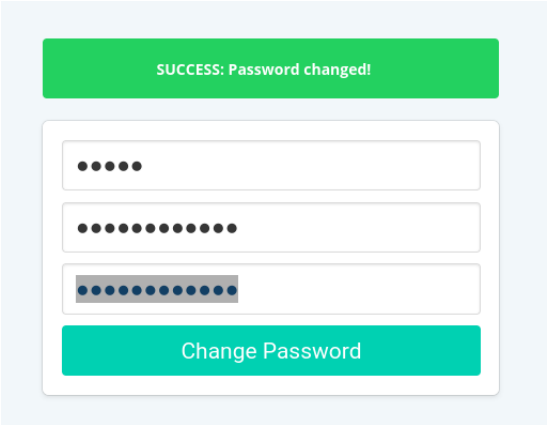

Figure 20: Secure Password Change form success

Failing with the wrong previous password.

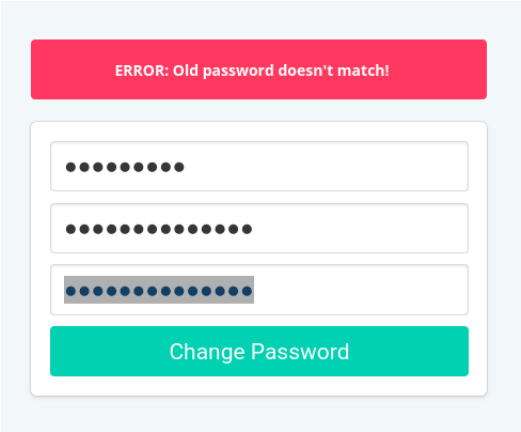

Figure 21: Secure Password Change form error

6 Conclusion

The security of an application is equally important to the users, as to the developers. Who puts a web application together has to have into consideration, everytime a new feature is implemented, all the ways of bypassing the new implementation, that way they can guarantee their prevention.

In conclusion, both SQL Injection and Cross-Site Scripting are well known vulnerabilities that should always be taken into account to protect the web applications from such type of attacks. The other two vulnerabilities were chosen (even thought they may look less relevant) to show how protecting user's data, specially passwords is very important.