MI130 - Web Application Security
Semester Project

Secure Webshop in PHP

Bennet Setzer 924697
Tim Kröger 922506
Nadine Kraft 924585

June 17, 2019

Contents

1. REQUIREMENTS

The requirement for this course was to build a secure Web Shop in a team with up to 4 persons.

1. Contents:
    - Product page (at least 8 items)
    - Insert items into basket (configurable, no static items)
    - Checkout process
    - Sign In Username / Password handling
    - Number of remaining items with verification that the desired order quantity is also available and rejection of an order quantity if the desired quantity exceeds the existing quantity.
    - An article order influences the stock of articles
    - Shopping cart functionality (cookies) without login
    - Shopping cart view includes correction possibilities of the existing order
    - Orders can be stored in a personalized manner and can be called up and printed out at any time for evaluation in the backend.
    - Multi-step payment process (checkout)
    - Login or registration
    - Possibility of correction of existing personal data
    - Selection of various payment options
    - Confirmation display of personal, bank and article-related data with possibility of printing after order activation
2. Features:
    - Secure Session handling with Cookies and database
    - Timeout for Session and Tokens
    - Secure Storage of passwords
    - CSRF handling
    - XSS handling
    - SQL-injection handling
    - Secure login

2. DEVELOPMENT PROCEDURE

During our first project meeting, we decided the general structure and type of shop we wanted to create. Based on this structure, we split the work into different work packages and assigned them to the members of our team. We choose an agile development approach and had weekly meetings during or after the Labs in order to discuss the progress on our current work packages. The work packages can be seen in the following table:

The Source code of our application was managed using Git and can be accessed via `https://github.com/Ben297/webshop`

Team Member	Bennet Setzer	Tim Kröger	Nadine Kraft
Task	MVC Pattern	MVC Pattern	Sessions
Task	Project Management	Shopping Cart	Mockups/Workflow
Task	Sessions	Cookie (Basket)	Documentation
Task	Authentication	Stock handling	Design Views
Task	DB Management	Testing	Checkout
Task	Checkout	Safety Aspects	Testing
Task	Account Overview	Documentation	Safety Aspects
Task	Safety Aspects		
Task	Testing		
Task	Documentation		

3. SOFTWARE ARCHITECTURE

To ensure a well structured and safe software architecture, we used the MVC pattern as shown in the image above:

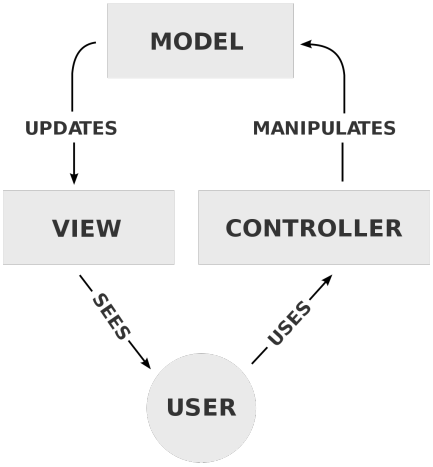

Figure 1. MVC Pattern

Source: `https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/1200px-MVC-Process.svg.png`

Our project structure is based on Dave Hollingworth example structure for an PHP-MVC: `https://github.com/daveh/php-mvc`

4. DATABASE DESIGN

The following graphic shows our Database design. Xampp uses MariaDB as its Database.

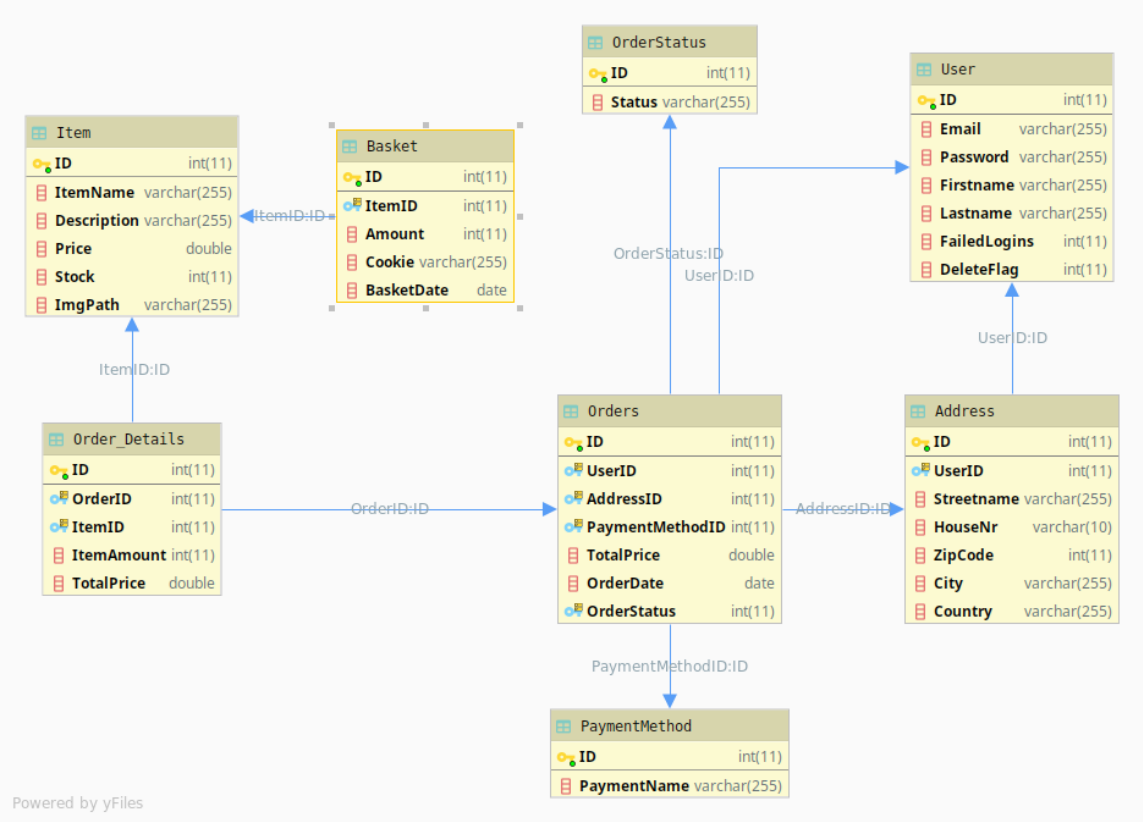

Figure 2. Database Model

5. SESSION HANDLING

```php
session_start([
    'name' => "session",
    'cookie_secure' => false,
    'cookie_httponly' => true,
    'cookie_lifetime' => 0,
    'sid_length' => 192,
    'sid_bits_per_character' => 6,
]);
```

Listing 1. File:index.php

Session handling is done via the default PHP Session handler. To provide additional security, the cookie is set to be http only. This ensures that it cant be accessed via JavaScript.The cookie_secure flag is set to false in this case because we do not have a https in this Project. But of cause we would set it to true in a real world scenario.

We used the default behaviour of PHP for the session timeout. Which is only valid for one browser session. Additionally we implemented activity-tracking for sessions. So basically the time a sessions is started, will be saved in a session variable and will be evaluated in the default methods of all sites. If the saved session time is older then 30 Min the session will be destroyed and the user can start a new session. If its under 30 min the session variable will be updated with the current time stamp.

6. SECURE STORAGE OF PASSWORDS

When a user registers his account, he will create a new password. The password will be hashed and then stored together with the corresponding user information in the user table of our database. The hashing is completed with the *password_hash()* function of PHP using the bcrypt algorithm and it is automatically salted.

Since the length of the created hash might change in the future, we stored the passwordhash as a varchar with the length of 255.

7. AUTHENTICATION

7.1 Registration

For the registration form the user has to submit all the information requested. In this project we just asked for the information we really need to value the users privacy. After the input is checked for empty values we validate the email address and tell the user to use a valid email if necessary. After these points are checked we insert the data into the database and hash the password as described above. Possible 'fail' flags in the session are unset.The login status is set to true and the user is therefor logged in.

7.2 Login

To authenticate himself to the server the user provides his email address and password. We check the database for the provided email and verify the password with the *password_verify()* function. If this is successful the user get the logged in status and we regenerate the session. Then there are basically two different cases: Failed login with existing account and non existing account. For both the webshop will be closed after five login attempts. Of course, in a real life scenario you would also have to store the IP and other identifiers to ensure that the potential attacker cannot Brute Force his way into your system. Also we are not providing information weather the email or the password is so an attacker cannot probe for accounts.

8. CSRF HANDLING

To protect the site from CSRF we implemented a CSRF-Token in all forms. The following functions Checks if the posted CSRF-Token is valid and existing before working with the post variabels.

```
1
2       public static function checkCSRF()
3       {
4           if (!empty($_POST['csrf_token']))
5               if (hash_equals($_SESSION['csrf_token'], $_POST['csrf_token']))
6                   return true;
7               else
8                   return false;
9           else
10              return false;
11      }
```

Listing 2. File:Helper.php

At the session start the CSRF-Token gets generated via the *openssl_random_pseudo_bytes()* function with the strong_crypto flag set to true to increase the randomness of the generated bytes. The generated bytes will then get hashed with by the md5 hash function.

```
1       if (empty($_SESSION['csrf_token'])) {
2       $crypto_strong = true;
3       $_SESSION['csrf_token'] = md5(openssl_random_pseudo_bytes(32,$crypto_strong));
4   }
```

Listing 3. File:index.php

The CSRF-Token is then embedded in all forms as an hidden input field.

9. XSS HANDLING

In the lecture we learned that Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.

XSS protection is achieved by:

1. Escaping user supplied data before displaying it on the page
2. A Content Security Policy (CSP) blocking all JavaScript from being executed

   We tested our site with the examples from the lecture:

   - Test 1: Testing JavaScript in the user details form

     ```
     1       &searchfilter=hallo<script>alert('xss');</script>
     ```

   - Test 2: Testing Document Domain

     ```
     1       &searchfilter=hallo<script>alert(document.domain);</script>
     ```

- Test 3: Testing Cookie with JavaScript

```
1       &searchfilter=hallo<script>alert(document.cookie);</script>
```

- Test 4: Testing Cookie without JavaScript

```
1       &searchfilter=hallo<svg/onload=alert(document.cookie)>
```

- Test 5: Testing Cookie with Eval

```
1       &searchfilter=hallo<a href='javascript:eval(alert(document.cookie))'>
```

  All tests resulted in a redirect to the google page.

We don't use search parameters in our URL and sanitize all user input. By doing this, we are able to prevent XSS attacks. Furthermore, we added a CSP to our website in the index file. Therefor we had to include all the bootstrap dependencies locally as well as Jquery.Also we deny the possibility to use our site in an Iframe.

10. SQL-INJECTION HANDLING

SQL-Injection Handling is provided by use of prepared statements with parameterized Queries. First, the SQL code is defined and after this, the parameters are passed to the query. Therefore, an attacker would not be able to change the intent of a query. The need to manually quote and escape the parameters is eliminated. In our project, we make use of the PHP Data Objects for accessing the database.

```
1   public function insertAddress($address,$userID)
2   {
3       $this->dbh= Model::getPdo();
4       $stmt =  $this->dbh->prepare('INSERT INTO Address VALUES (NULL,?,?,?,?,?,?)'
            );
5       $stmt->bindParam(1, $userID,\PDO::PARAM_INT);
6       $stmt->bindParam(2, $address['streetname'],\PDO::PARAM_STR);
7       $stmt->bindParam(3, $address['houseNr'],\PDO::PARAM_INT);
8       $stmt->bindParam(4, $address['zipCode'],\PDO::PARAM_INT);
9       $stmt->bindParam(5, $address['city'],\PDO::PARAM_STR);
10      $stmt->bindParam(6, $address['country'],\PDO::PARAM_STR);
11      return $stmt->execute();
12  }
```

Listing 4. User.php

We tested the Log-in with the following example statements from the lecture:

- 'OR 1=1'
- ' –
- " or true–
- ')) or true;
- '; DELETE FROM TABLE tbl_users WHERE user LIKE '%' –

None of these attacks worked.

We further tested the website for SQL-Injection Bugs:

1. Submit a single quote (') as input in url query:
   Resulted in a redirect to google.
2. Submit two single quotes.
   Resulted in a redirect to google.
3. Try string or numeric operators: Tested for

   ```
   1   http://localhost/detailpage/showDetail/12
   ```

   resulted in an empty detail page

   ```
   1   http://localhost/detailpage/showDetail/&item=3+ORDER+BY+1+--
   ```

   Resulted in an error message

11. CHECKOUT PROCESS

To protect the basket and checkout process from manipulation during the payment process, we constructed the checkout to behaviour in the following matter:

1. User puts item in the basket
2. User can change amount or delete item
3. User submits order request
4. Check if user is logged in
5. Check if the user wants to check his given address(can change it)
6. Select which payment method to use
7. Show order overview
8. Order confirmation
9. Decrease the stock amount after the checkout process

12. INFORMATION LEAKAGE

All Routes which are not known to the router are redirected to a 404 page when the debug-flag in the Config.php is set to FALSE. All requests a handled in the index.php(.htaccess) so attackers cannot probe out server directory structure. Furthermore we have a robot.txt file which disallows all web crawlers at the moment.

13. CHALLENGES

During our project we encountered numerous challenges, e.g.: Developing with PHP 7 without using a framework. We had to redo the project after realizing that our initial implementation of the MVC-Pattern wouldn't work. Keeping the right structure of the MVC-Pattern was at times difficult but as we worked more with this architecture we got a better understanding and appreciation for it. Like always time was of course a limiting factor. additionally logical flaws were sometimes a problem in the development process. For example unsetting a flag in the session after a process or what to do if you don't use the given button and instead invoked a specific route. But we managed to conquered these challenges quiet well and learned a lot during the process.