Assignment 2: Improving Code for Security

Sarah Gillard

CS202: Network and System Security

March 16, 2024

Prompt Part A- Improving Front-End

Below is part of the code for a login page. In a minimum of 300 words, identify and discuss security issues this code could cause and provide a new code that prevents these security issues.

Prompt Code:

```
<form method='get' action='user_credential_check.php'>

<label for='userId'>Your User ID: </label>

<input type='text' id='userId' name='userId' size='20'>

<label for='password'>Your Password: </label>

<input type='text' id='password' name='password' size='20'>

<input type='submit' value='Submit'>

</form>
```

The provided code has several security vulnerabilities that could potentially compromise user data and further harm the network. We should start by altering the HTTP method to enhance the code's security. Currently, the code employs the GET method for HTTP. The Hypertext Transfer Protocol (HTTP) delineates the protocols for web servers and browsers to correctly render web pages (Aravindan, S., 2015). We should switch to the POST method for improved security. The GET method is utilized for retrieving data from a designated resource, whereas the POST method facilitates data transmission to a server to create or update a resource. POST offers better security as it refrains from storing parameters in the browser history, unlike GET. Consequently, when transmitting sensitive information such as passwords, it's imperative to abstain from using GET due to this vulnerability. The code also lacks input validation which makes it susceptible to attacks such as SQL injection and cross-site scripting (XSS) which could cause issues such as data breaches. Cross-site scripting occurs when an attacker injects malicious script into the code of a trusted web page and SQL injection is an injection attack where an attack runs malicious queries. As a preventative measure, we should add the required fields to the password. A password that is at least 20 characters will be more secure. To make this a requirement, we add the word “required” after “size = ‘20’” in the code.

The code also lacks the use of secure password input fields as they are currently displayed in plain text which can expose them to hackers as users type them. For added security, we should change the password input type to “password” to encrypt the characters a user enters for their password. The new changes to the code below will help protect against vulnerabilities and are important security measures to take.

Improved Code:

```
<form method = ‘post’ actions = ‘user_credential_check.php’>
        <label for = ‘userId’>Your User ID: </label>
        <input type = ‘text’ id = ‘userId’ name = ‘userID’ size = ‘20’ required>
        <label for = ‘password’>Your Password: </label>
        <input type = ‘password’ id = ‘password’ name = ‘password’ size = ‘20’ required>
        <input type = ‘submit’ value = ‘Submit’>
</form>
```

Prompt Part B- Improving Back-End

Below is part of the code to handle the login page. In a minimum of 300 words, identify and discuss security issues this code could cause and provide a new code that prevents these security issues.

Prompt Code:

```
<?php

$uid = $_GET['userId'];

$upw = $_GET['password'];

$query = "SELECT * FROM users WHERE id = '$uid' and pw = '$upw'"

....

?>
```

To improve the provided back-end code, we need to consider potential security vulnerabilities such as the absence of HTTPS. An absence of HTTPS puts security at a higher risk. HTTPS is the Secure Hypertext Transfer Protocol which uses the Advanced Encryption Standard and will help protect sensitive data such as passwords. Using HTTP does not provide encryption and security. HTTPS will help keep the user credentials and data safe from potential hackers. Additionally, you want to use the POST HTTP method and not the GET HTTP method to enhance security. POST will provide better security as it does not store parameters such as passwords.

Furthermore, a concern is a susceptibility to SQL injection attacks. An SQL injection attack occurs when malicious activity gets into a poorly designed database to access confidential information and make unauthorized changes to the database. Attackers could manipulate the input parameters to execute malicious SQL queries, and possibly access or modify sensitive data in the database. To mitigate this risk, we should implement PHP Data Objects (PDO) prepared statements. A prepared statement is a functionality used to execute the same (or similar) SQL statements multiple times with optimal efficiency. Using prepared statements will reduce the time to parse since the query preparation only occurs once, despite the statement being executed multiple times. The bounded parameters minimize the usage of server bandwidth since only the parameters are sent each time rather than the entire query.

Prepared statements prepare, parse, compile, and then execute. A SQL statement template is created and transmitted to the database with certain values left unspecified. These unspecified values are parameters and they are indicated by a question mark. The database parses, compiles, and optimizes the SQL statement template and then stores the result. Next, the prepared statement executes at a later point. The application connects the values to the parameters and the database executes the statement. The application can execute the statement multiple times with various values.

To use PDO in the code, we need to specify a database as a valid database is required for the statements to connect to. If we do not specify a database, the code will throw an exception. In the improved code, the database is called "sdcBD." The PDO uses a try-catch block that handles problems that occur in the database queries.

An improvement will also be preparing and executing statements for the SQL query with bound parameters. The use of placeholders and binding parameters to them helps eliminate the risk of malicious SQL injection attempts. The proper syntax for executing SQL queries within the PDO framework must be used to ensure compatibility.

These enhancements will ensure the back-end code securely connects to the database and securely executes queries. This helps mitigate potential security risks and vulnerabilities surrounding sensitive user data.

Improved Code:

```
<?php

$uid = $_POST['userId'];

$upw = $_POST['password'];


//Define database connection parameters

$svrname = ‘localhost’;

$uid = ‘userId’;

$upw = ‘password’;


//Use PDO to establish a database connection

Try {

        $connection = new PDO(‘mysql:host = $svrname, dataBaseName = ‘sdcDB’, $uid, $upw);
        $connection-> set Attribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        echo ‘Connected successfully’;

}       catch(PDOException $e) {

        echo ‘Connection failed: ‘ . $e->getMessage();

}


//Prepare and execute the SQL query with bound parameters

$sql = "SELECT * FROM users WHERE id = '$uid' and pw = '$upw';

$stmt = $pdo -> prepare($sql);

$stmt -> bindParam(‘:uid’, $uid, PDO::PARAM_STR);

$stmt -> bindParam(‘:upw’, $upw, PDO::PARAM_STR);

$stmt -> execute();

?>
```

Prompt Part C- Using Content-Security-Policy in HTML

Content-Security-Policy (CSP) is a layer of security designed to prevent certain types of attacks, such as clickjacking, cross-site scripting, and data injection attacks. Below is an example of the CSP code using an HTML meta tag. Provide a CSP code allowing scripts, CSS, and form actions only from your website. Use a meta tag in HTML.

Prompt Code:

```
<meta http-equiv='Content-Security-Policy' content='default-src 'self''>
```

Adding script-src ‘self’ restricts scripts to be loaded only from the same origin to prevent any scripts from being loaded from external domains. The addition of style-src ‘self’ limits the stylesheets to be loaded only from the same document as the HTML to prevent stylesheets from external documents. Stylesheets provide the Cascading Style Sheets or CSS to the webpage. We also add the form-action ‘self’ to allow form submissions that are the same as the HTML document to be used. Finally, we add upgrade-insecure-requests to automatically convert the URL from HTTP to HTTPS to provide security and encryption.

Improved code:

```
<meta http-equiv = “Content-Security-Policy” content = default-src ‘self’; script-src ‘self’; style-src ‘self; form-action ‘self’” upgrade-insecure-requests>
```

References

Aravindan, S. (2015, October 6). *Hypertext Transfer Protocol | HTTP Definition & Process*. Study.com. https://study.com/learn/lesson/what-is-hypertext-transfer-protocol-examples.html

Anonymous (n.d.). *CSP Header Reference.* Content-Security-Policy. https://content-security-policy.com/#:~:text=Content%2DSecurity%2DPolicy%20is%20the,they%20can%20be%20loaded%20from. Accessed 16 March 2024.

Anonymous (n.d.). *HTTP request methods. HTTP Methods GET vs POST*. W3Schools. https://www.w3schools.com/tags/ref_httpmethods.asp. Accessed 14 March 2024.

Anonymous (n.d.). *PHP MySQL Prepared Statements.* https://www.w3schools.com/php/php_mysql_prepared_statements.asp. Accessed 16 March 2024.

Anonymous (n.d.). *PHP Connect to MySQL.* https://www.w3schools.com/php/php_mysql_connect.asp. Accessed 16 March 2024.

Anonymous (n.d.). *PDOStatement::bindParam.* https://www.php.net/manual/en/pdostatement.bindparam.php. Accessed 16 March 2024.