Abigail Davies PHP Assignment

by Abi Davies

Submission date: 05-Dec-2022 12:56PM (UTC+0000)
Submission ID: 191058826
File name: 32723_Abi_Davies_Abigail_Davies_PHP_Assignment_460452_1111604622.docx (49.16K)
Word count: 1237
Character count: 6826

Module Name: Databases and Application Development (Assignment 1)

Module Number: ACCA5026

Student Name: Abigail Davies

Student Number: 2114472

Course: BSC2-PT Computing (Software Engineering)

Contents

Introduction 3
Security Issues 3
SQL Injection 3
Identification and Authentication Failures 5
Conclusion 6
References 7

Introduction

There are many security risks that should be considered when planning and creating web applications. For this reason, organisations and individuals have created websites that review and monitor the hierarchy of risks that websites are subject to. One website would be the Open Web Application Security Project (OWASP) [1]. OWASP assesses the impact of the top ten security risks each year and compares them to previous years [1]. These types of websites allow software engineers to understand what security measures they should implement within their web application and the importance of reviewing security measures on a regular basis.

Security Issues

SQL Injection

Structured Query Language (SQL) is used to control what is stored in a database [2]. It can also be used to find out information that is stored in a database by utilising the commands from the language. Unfortunately, this means that web applications that have a connection to a database can be vulnerable to SQL injection. This is a security risk that software developers should be aware of when creating a web application so that they can minimise the risk [1]. OWASP states that this risk has been within the top three security risks for web applications between 2017 and 2021 [1]. This emphasises the importance of having prevention methods in place, to attempt to safeguard a website and database.

Hackers can use SQL injection in many ways. For example, a hacker could use SQL injection to delete all the data from a database [3]. Alternatively, a hacker could use SQL injection to gain access to a database that contains sensitive and / or confidential information [2].

As SQL injection is a significant risk that software developers need to be aware of, a handful of prevention methods have been identified. One prevention method would be to use prepared statements when executing SQL queries [4]. This removes the risk of hackers attempting to maliciously amend the current SQL query, with a new query. Below is a section of code from the annual leave system, where I have included prepared statements:

```
include "connectioninfo.php";

@$username = $_POST["username"];
@$password = $_POST["password"];

$query = "SELECT * FROM accountinfo WHERE username LIKE ? AND password LIKE ?";

$stmt = mysqli_stmt_init($link);
mysqli_stmt_prepare($stmt,$query);
mysqli_stmt_bind_param($stmt,"ss", $username, $password);
mysqli_stmt_execute($stmt);
$result = mysqli_stmt_get_result($stmt);
$numrows = mysqli_num_rows($result);
mysqli_stmt_close($stmt);
mysqli_close($link);
```

This section of code can be found in logincheck.php, although I have ensured that this technique has been set up for various pages. I believe I have inserted prepared statements where necessary. Initially, I thought that the main priority was to have SQL injection prevention on the log in page. My reason for this was that only authorised users would be able to access the other pages. However, after further consideration, I realised that the system would be vulnerable to disgruntled employees or hackers gaining access fraudulently.

Another suggestion that OWASP had would be to minimise user input [5]. However, it is said that this is not the most recommended prevention method. The annual leave system does not allow for no user input at all, although there are small sections where it can be reduced. OWASP explains that avoiding user input in a web application can reduce the chance of SQL injection occurring, but it is a weak approach compared to other methods [5].

User input has been slightly reduced via dropdown boxes where possible. An example of the code for this from the annual leave system is below:

```
<?
session_start();
include "connectioninfo.php";

@$username = $_SESSION["username"];

$query1 ="SELECT * FROM availabledates where dateid in (SELECT dateid from annualleave where
employeeid in (SELECT employeeid from accountinfo where username LIKE '$username'))";

$resultquery1=mysqli_query($link,$query1);

?>
                <div class="container">
                <h3>Please select the date in which you would no longer like to take leave on:</h3>
                </div>
                        <section id="form">
                        <form action="deleteleave.php" method="POST">
                                Date: <select name="chosendate">
                                <? while ($r1=mysqli_fetch_array($resultquery1)) {
                                        ?>
                                        <option value="<?echo $r1["availabledate"];?>">
                                        <?echo $r1["availabledate"];?></option><?
                                }?>
                                </select><br><br>
                                <input type="submit">
                                </form>
                        </section>
                                        <?
                }
        ?>
        </div>
        </body>
</html>
```

Identification and Authentication Failures

Identification and authentication failures is another security risk that software developers need to assess when creating web applications. The 2021 survey conducted by OWASP, recognised this risk as one of the top ten [1]. Although, the risk has declined over the past five years, and it is believed that it is due to the introduction of available standardised frameworks [1].

It is vital that software engineers implement some validation methods when developing a web application system that involves user authentication [6]. The type of weakness that software engineers should avoid would be weak passwords. For example, the general rule that I've noticed is that a password must be 8 characters or longer and include both upper- / lower-case letters, symbols, and numbers. In the current stage of the annual leave system, it does not fit these criteria. However, this would be implemented prior to release of the web application so that employees can only use secure passwords. In addition, implementation of regular password updates could add to the strength of employee's passwords. This process has been implemented in my workplace.

Another vulnerability that OWASP mentions would be a weak 'forgot password' method [6]. Similarly, to the previous weakness, this has also not been implemented in the current stage as only the admin user has access to change the password. Therefore, I would suggest that a process is set up where a user must contact the administrator to reset password, if required, and they would have to confirm specific questions to allow the administrator to proceed. Additionally, it is suggested that multi-factor authentication should be used [6]. This method could be used to log in to the system and / or to reset password via administrator.

Another recommendation would be to log failed log in attempts [6]. This could recognise a potential hacker attempting to gain access to the system. Multiple failures could generate an alert for the administrator or possibly lock the user's account. Many organisations appear to have the latter introduced on their web applications. These are accompanied by a process to unlock the user's account, involving the user confirming their identity via multiple questions, prior to unlocking the account.

Lastly, the annual leave system sets the session of the user when they log into the web application. OWASP suggests that some web applications do not execute this process efficiently, indicating that an individual could gain access via another's log in if they were not to log off the system correctly [6]. During testing, the session appears to clear once closing down the web browser, resulting in the user 'logging out' of the system without following the official process. That being said, further testing and research would need to take place prior to the release of the web application to ensure that this would be the case on all machines. The section of code below is an example of where this has been introduced:

```
<?
session_start();
include "connectioninfo.php";

@$username = $_SESSION["username"];

?>

<html>
        <link rel="stylesheet" type="text/css" href="StyleSheet.css">

        <head>
```

```
        </head>

        <body>

        <?if ($username == "" OR $username == "admin") {
                include "errorheader.php";
        } else {
                ?>
                <header id="main-header">
                        <div class="container">
                        <h1>Welcome <?echo $username;?></h1>
                        </div>
                </header>
                        <nav id="navbar">
                                <div class ="container">
                                        <ul>
                                                <li><a href="home.php">Home</a></li>
                                                <li><a href="profile.php">Profile</a></li>
                                                <li><a href="myleave.php">My Leave</a></li>
                                                <li><a href="teamleave.php">Team Leave</a></li>
                                                <li><a href="logout.php">Log out</a></li>
                                        </ul>
                                </div>
                        </nav>

        <?}
        ?>

        </body>
</html>
```

Conclusion

To conclude, there are many security risks that software developers need to consider when creating web applications. It is impossible to remove all vulnerabilities, although it is crucial that the available prevention methods are implemented where needed. These can significantly reduce the risks that the web application is subject to.

References

[1] Open Web Application Security Project, "OWASP Top Ten", *OWASP*. [Online]. Available: https://owasp.org/www-project-top-ten/ [Accessed Nov. 28, 2022].

[2] M. A. M. Yunus, M. Z. Brohan, N. M. Nawi, E. S. M. Surin, N. A. M. Najib, and C. W. Liang, "Review of SQL Injection: Problems and Prevention", *JOIV: International Journal on Informatics Visualization*, vol. 2, no. 3-2, June 2018. [Online]. Available: http://joiv.org/index.php/joiv/article/view/144/142 [Accessed Dec. 02, 2022].

[3] Gaynor Thomas, lecture.

[4] Open Web Application Security Project, "A03:2021 – Injection", *OWASP*. [Online]. Available: https://owasp.org/Top10/A03_2021-Injection/ [Accessed Nov. 30, 2022].

[5] Open Web Application Security Project, "SQL Injection Prevention Cheat Sheet", *OWASP*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html [Accessed Dec. 02, 2022].

[6] Open Web Application Security Project, "A07:2021 – Identification and Authentication Failures", *OWASP*. [Online]. Available: https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/ [Accessed Dec. 03, 2022].

Abigail Davies PHP Assignment

GRADEMARK REPORT

FINAL GRADE

74/100

GENERAL COMMENTS

Instructor

WEBSITE

Database structure and design - 6 tables, design stands up well, correctly normalised 10/10

Range of PHP demonstrated -

Login is well-managed and also prevents URL attack

Edit is OK - but it would help if you populated the fields with the existing data. Insert and delete also managed well.

There's something a little odd when booking leave - I typed in a date that wasn't in the database and it went a bit wrong when I then tried to delete it. Might need a pull-down menu with the dates in to prevent this when booking leave.

36/50

Overall integration of website -

It's a bit sparse at times - try to keep a header on each page that's consistent.

It's not always clear where the links are on the page as some are in black text similar to the rest of the text on the page. It would help the user if they stood out a bit.

7/10

REPORT

Investigation/analysis of two security issues - A little bit brief on the SQL injection, would have been nice to see some examples of injection attacks that could be made such as the 1=1 attack from class and others. Authentication failure is covered quite well. 10/15

Mitigation of security issues in your site - Prepared statements have been shown well in the code (could have had a screenshot of it here!) could have had a few more examples of mitigating for authentication failure 11 /15

General Comments: Be careful not to use "I" in academic reports eg "I have inserted prepared statements where necessary" should be in the passive voice - "Prepared statements should be inserted where necessary"

TOTAL: 74/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7