IMPLEMENTATION OF SECURITY FEATURES ON AN EXCURSION BOOKING WEBSITE

ABSTRACT

In this report we reviewed how we mitigated the possibility and severity of attacks on our booking website. Some of the attacks we cover are SQL injection attacks which gives hackers unauthorized access to our database, XSS attacks and unauthorized page access.

INTRODUCTION

The need to secure customer information is increasingly important as unethical hackers are willing to exploit any vulnerabilities in a website. While it may be impossible to completely secure a website, we can try to mitigate these attacks and their consequences. Our excursion website makes use of forms in a number of places such as the sign up, login and booking. These forms can be a pathway for attacks to compromise our website and database by using SQL injection attacks and XSS attacks. We also don't want restricted pages to be accessed by any user who isn't logged in or in our system. In this report we discuss how we addressed these issues.

BACKGROUND

SQL injection works by tricking the server to carry out dangerous commands by adding SQL statements to the query string of a Web form's submission, or input domain. An attacker may be able to entirely take over the database on the remote server by using SQL injection to get around the authentication process. Ma. L et al (2019) studied these attacks and identified the types of SQL attacks we may encounter which include incorrect type handling, escape character is not filtered correctly and bind SQL injection attack. A specific kind of online vulnerability called Cross-Site Scripting (XSS) enables an attacker to insert malicious code into a website. When additional users access the website and run the injected code, the attacker may be able to steal confidential data or carry out other nefarious deeds. Dayal. M et al (2016) discussed how XSS works and offered solutions. The XSS attacker uses different approaches to encrypt the malicious code so that it seems genuine to the user but is actually fake. Some of the methods an attacker encrypts bad code to appear legitimate to the user include session hijacking through the use of malicious script, The goal of XSS is to manage client-side scripts of a web application so they run in the malicious manipulator's chosen sequence. These sorts of modifications can include a script that will run every time the website loads or whenever a connected event takes place (Shrivastava A et al, 2016). An example of XSS attack that we need to consider for our site is the XSS attack in input field. When ready to attack, the attacker modifies the input field with the malicious script after using the HTML web page's tag to change the data. When the user sees the remark and activates it, it is then carried out on their browser (Dayal.M et al, 2016). Shrivastava A et al(2016) recommended the use of a signature mechanism to detect javascript and any scripting language that can be used to insert XSS attacks, as one of the ways to prevent it from happening.

<u>METHODOLOGY</u>

To prevent SQL injection and XSS we implemented some php functions which we would now discuss.

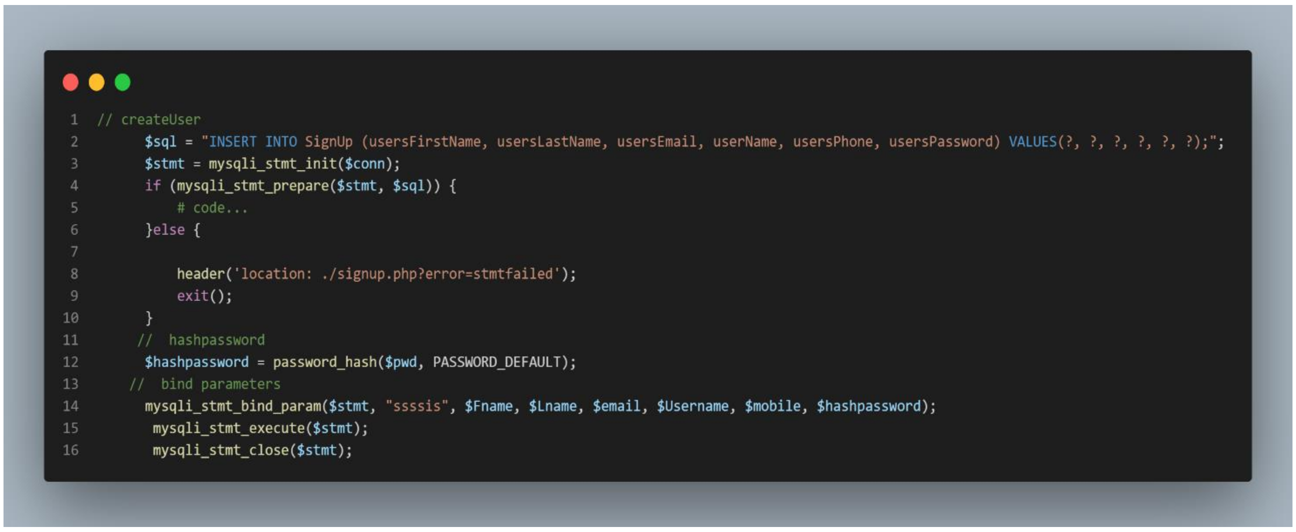
i. SQL INJECTION

One way to stop SQL injection attacks is to use prepared statements with confined arguments (Security Journey, 2020).

A prepared statement is a SQL statement that has already been pre-compiled by the database server. The prepared statement can then be executed many times with different parameter values.

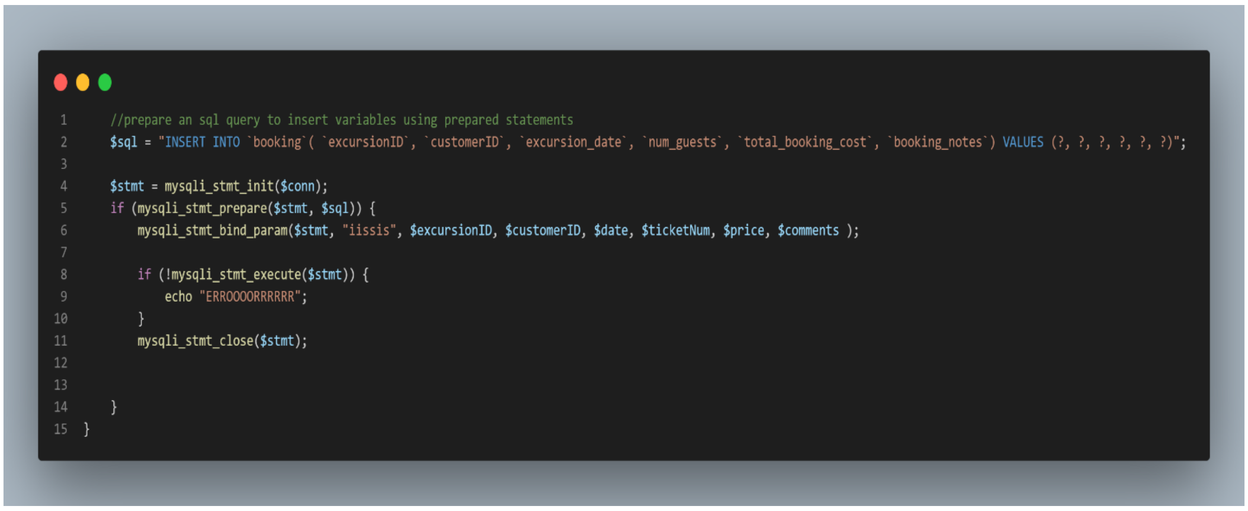
When utilising bound parameters with a prepared statement, you can declare the parameter values separately from the rest of the SQL query. This reduces the risk of SQL injection attacks since the parameter values are treated as distinct data rather than as a part of the SQL syntax.

We utilised this in our code in our pages that took in user input such as Sign up and details page.



```php
// createUser
      $sql = "INSERT INTO SignUp (usersFirstName, usersLastName, usersEmail, userName, usersPhone, usersPassword) VALUES(?, ?, ?, ?, ?, ?);";
      $stmt = mysqli_stmt_init($conn);
      if (mysqli_stmt_prepare($stmt, $sql)) {
          # code...
      }else {

          header('location: ./signup.php?error=stmtfailed');
          exit();
      }
     //  hashpassword
      $hashpassword = password_hash($pwd, PASSWORD_DEFAULT);
    //  bind parameters
      mysqli_stmt_bind_param($stmt, "ssssis", $Fname, $Lname, $email, $Username, $mobile, $hashpassword);
       mysqli_stmt_execute($stmt);
       mysqli_stmt_close($stmt);
```

FIG 1. SQL Prepared statements in our sign-up page.



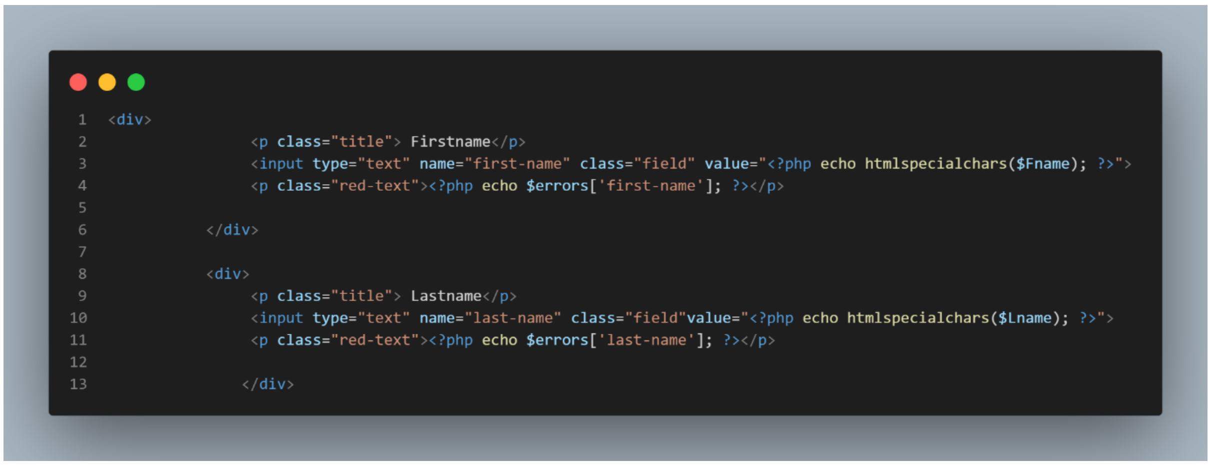
```php
    //prepare an sql query to insert variables using prepared statements
    $sql = "INSERT INTO `booking`( `excursionID`, `customerID`, `excursion_date`, `num_guests`, `total_booking_cost`, `booking_notes`) VALUES (?, ?, ?, ?, ?, ?)";

    $stmt = mysqli_stmt_init($conn);
    if (mysqli_stmt_prepare($stmt, $sql)) {
        mysqli_stmt_bind_param($stmt, "iissis", $excursionID, $customerID, $date, $ticketNum, $price, $comments );

        if (!mysqli_stmt_execute($stmt)) {
            echo "ERROOOORRRRRR";
        }
        mysqli_stmt_close($stmt);


    }
}
```

FIG 2. SQL Prepared statements in our details page.

This use of prepared statements also prevents escape character not filtered correctly attack, incorrect type handling and bind SQL injection attack.

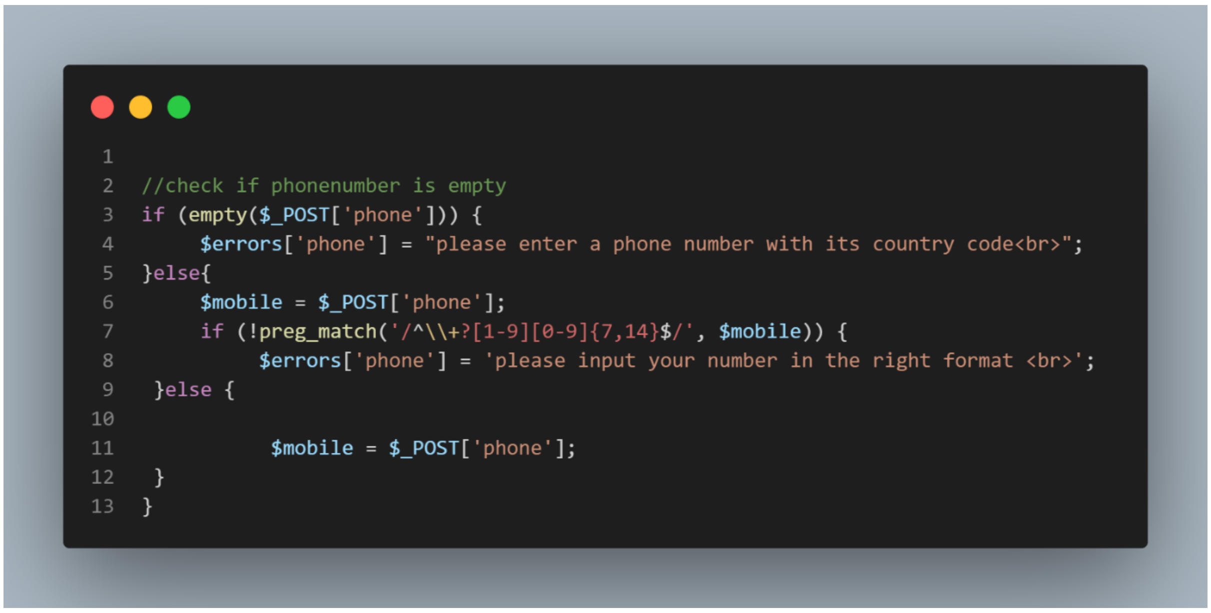
ii. XSS ATTACK

We can prevent XSS attack by validating and sanitizing the user input. This is to make sure that data entered is in the right format and there are no special characters that can be used for scripting. A way we sanitized our inputs was by using some php built in functions such as filter_var() for our email input to make sure scripting tags aren't inserted, use of RegExr for our mobile number to make sure only valid phone numbers are inputted and using htmlspecialchars to convert any special character to html entities. We used this in our forms that require inputs.



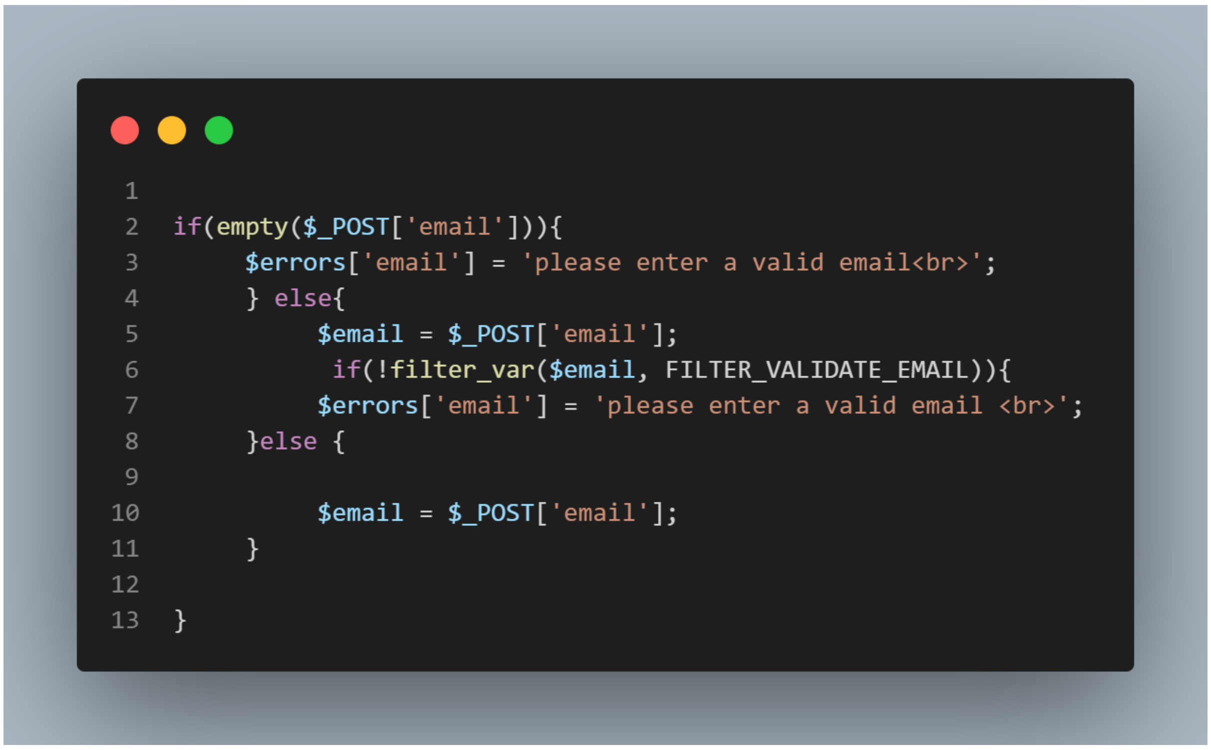
```
<div>
                <p class="title"> Firstname</p>
                <input type="text" name="first-name" class="field" value="<?php echo htmlspecialchars($Fname); ?>">
                <p class="red-text"><?php echo $errors['first-name']; ?></p>

           </div>

           <div>
                <p class="title"> Lastname</p>
                <input type="text" name="last-name" class="field"value="<?php echo htmlspecialchars($Lname); ?>">
                <p class="red-text"><?php echo $errors['last-name']; ?></p>

               </div>
```

FIG 3. htmlspecialchars in input fields.



```

//check if phonenumber is empty
if (empty($_POST['phone'])) {
     $errors['phone'] = "please enter a phone number with its country code<br>";
}else{
     $mobile = $_POST['phone'];
     if (!preg_match('/^\\+?[1-9][0-9]{7,14}$/', $mobile)) {
          $errors['phone'] = 'please input your number in the right format <br>';
 }else {

           $mobile = $_POST['phone'];
 }
}
```

FIG 4. Regular expressions to check phone number is valid.



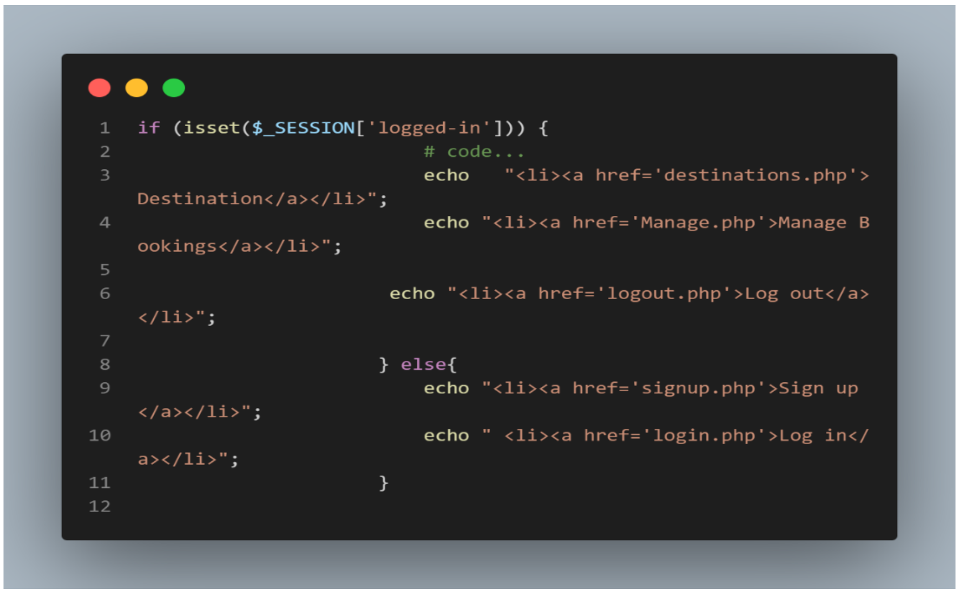
```php

if(empty($_POST['email'])){
      $errors['email'] = 'please enter a valid email<br>';
      } else{
           $email = $_POST['email'];
            if(!filter_var($email, FILTER_VALIDATE_EMAIL)){
           $errors['email'] = 'please enter a valid email <br>';
      }else {

           $email = $_POST['email'];
      }

}
```

FIG 5. Use of filter_var() to validate email.

iii. RESTRICTED PAGES

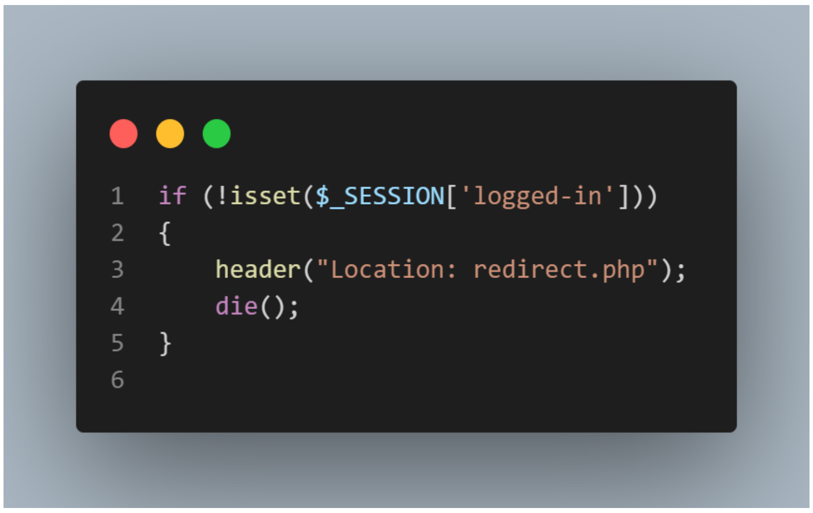
We do not want someone to attempt to access a page via that url that is only accessible to logged in users without logging in. To achieve this we used the following steps.

- Track user login using a php session variable "logged-in".
- Create a check in the navbar to display the hidden pages if user is logged in.
- Create a check to see if the user is coming to that page after being loggd in, if not, send him to an error page.



```
if (isset($_SESSION['logged-in'])) {
                        # code...
                        echo   "<li><a href='destinations.php'>Destination</a></li>";
                        echo "<li><a href='Manage.php'>Manage Bookings</a></li>";

                     echo "<li><a href='logout.php'>Log out</a></li>";

                    } else{
                        echo "<li><a href='signup.php'>Sign up</a></li>";
                        echo " <li><a href='login.php'>Log in</a></li>";
                    }

```

FIG 6. Dynamic display of nav contents.



```
if (!isset($_SESSION['logged-in']))
{
    header("Location: redirect.php");
    die();
}

```

FIG 7. A check for if the user is accessing the page properly.

<u>CONCLUSION</u>

Our excursion website tires to secure our database and user information by using a variety of security techniques to prevent SQL injection attacks, XSS and secure logins. However, there are things we can implement moving forward to improve security. To prevent unwanted access to our website, secure authentication and authorisation procedures can be used. This can entail employing two-factor authentication, creating strong passwords, tracking every log on the website, and storing authentication data in safe cookies. SSL should also be used to secure the connection between client and sever.

<u>REFRENCES</u>

Dayal Ambedkar, Mohit, et al. *A Comprehensive Inspection of Cross Site Scripting Attack*. Apr. 2016, 10.1109/CCAA.2016.7813770. Accessed 9 Jan. 2023.

Ma, Limei, et al. "Research on SQL Injection Attack and Prevention Technology Based on Web." *2019 International Conference on Computer Network, Electronic and Automation (ICCNEA)*, Sept. 2019, 10.1109/iccnea.2019.00042. Accessed 9 Jan. 2023.

Shrivastava, Ankit, et al. *XSS Vulnerability Assessment and Prevention in Web Application*. 2016 2nd International Conference on Next Generation Computing Technologies (NGCT), Oct. 2016.