BACHELOR OF CYBER SECURITY

SCHOOL OF COMPUTER SCIENCE

BINA NUSANTARA UNIVERSITY

JAKARTA

ASSESSMENT FORM

Course: COMP6695001 - Secure Programming

Method of Assessment: Case Study

Semester/Academic Year : 5/2023-2024

Name of Lecturer : **D6729 – Chrisando Ryan Pardomuan Siahaan, S.Kom., M.Kom.**

Date : **January 5th, 2024**

Class : **LA07**

Topic : **LO3: Apply Web Security Method**

Group Members :	1. 2501962870 – Vivian Dwitama 2. 2501966206 – Jaysie Lestari 3. 2540129193 – Febri Fransisca 4. 2540131292 – Wiryahadi Gunawan 5. 2540134944 – Rika Zakiyyah

Student Outcomes:

(SO 8) Mampu mengimplementasi sistem yang aman dengan metode *secure software engineering.*
Able to implement secure system with secure software engineering methodology.

Learning Objectives;
(LObj 8.2) Mampu mendesain fungsi sistem keamanan dengan metode yang sesuai dan sistematis;
Able to design security system functions with the appropriate method and systematic;

Learning Outcomes:

LO 1: Describe Web Technology Environment
LO 2: Construct Web Application
LO 3: Apply Web Security Method

No	Related LO-LOBJ-SO	Assessment criteria	Weight	Excellent (85 - 100)	Good (75-84)	Average (65-74)	Poor (0 - 64)	Score	(Score x Weight)
1	LO3 LObj8.2 SO8	Able to demonstrate web programming in detail and develop a functioning web application	**30%**	Able to describe describe web technology environment in detail and develop a fully functioning web application with rich and sophisticated features	Able to describe describe web technology environment in detail and develop a fully functioning web application with basic features	Able to describe describe web technology environment in detail and develop a functioning web application, although with some bugs / errors	Lack of ability to describe describe web technology environment nor develop a functioning web application		
2	LO3 LObj8.2 SO8	Able to secure dangerous methods and functions in web technology with	**20%**	Able to apply web security method on every feature developed for the web application	Able to apply web security method on half of the feature	Able to apply web security method on not suitable case	Lack of ability to apply web security method on suitable case		

No	Related LO-LOBJ-SO	Assessment criteria	Weight	Excellent (85 - 100)	Good (75-84)	Average (65-74)	Poor (0 - 64)	Score	(Score x Weight)
		proper mitigation strategy			developed on the web application				
3	LO3 LObj8.2 SO8	Able to perform a thorough code review and identify vulnerable code in a web applications	**25%**	Able to perform a thorough code review and identify every vulnerable code, then create an elaborate and clear report	Able to perform a thorough code review and identify some of the vulnerable code, then create an clear report	Able to perform a thorough code review and identify every vulnerable code, but not able to create an elaborate and clear report	Lack of ability to perform a thorough code review and identify every vulnerable code nor create an elaborate and clear report		
4	LO3 LObj8.2 SO8	Able to provide a correct security mitigation strategy and recommendation to prevent vulnerability on web application	**25%**	Able to provide a robust (*unbypassable*) and clear recommendation to fix all of the found security vulnerability after code review	Able to provide a robust (*unbypassable*) and clear recommendation to fix most of the found security vulnerability after code review	Able to provide a quick fix recommendation to fix the found security vulnerability after code review, but the recommendation does not fix the problem entirely	Lack of ability to provide a robust (*unbypassable*) and clear recommendation to fix the found security vulnerability after code review		
		Total Score: ∑(Score x Weight)							

Remarks:

ASSESSMENT METHOD
Case Study

Instructions

1. Students are divided into groups of 3-5 people.
2. [LO3, Lobj8.2, SO8] The students are expected to develop a secure web application (the features and themes of the application are unconstrained) according to the web security standards learned in this course
3. [LO3, Lobj8.2, SO8] The students in a group must pair with other group in a circular manner (group A and B cannot pair with the same group C). Both groups must assess the code written by each other and identify the potential security vulnerability / security countermeasures (if there is no vulnerability anymore) in the application (white-box, source code should be shared between the two groups)
4. Each group must then submit two reports:
    a. [LO3, Lobj8.2, SO8] A documentation report containing elaborative explanation of the features and functionalities of the web application they developed, including security countermeasures they have put in place.
    b. [LO3, Lobj8.2, SO8] A code review report containing the identified vulnerabilities / countermeasures of the other group's application, along with recommendations / feedback on them. The report must be written with code snippet / screenshot attached on each explanation.

Case Study Output

- [LO3, Lobj8.2, SO8] Application
- [LO3, Lobj8.2, SO8] Application Report
- [LO3, Lobj8.2, SO8] Code Review Report

Note for Lecturers:

1. Emphasize to the student that the project will be evaluated using the version of the source code that is used in the code review activity (to prevent cheating because there will be code sharing between groups). This way, any modification / addition into the code after the code review activity will not be evaluated.
2. Code review activity will begin on 10th session of the lecture, thereby students are expected to finish their application by then. Inability to submit a functioning application for the code review will affect their final score.
3. Students must collect and present their project, along with their code assessment report of the other groups, in the last week of lecturing sessions

Secure Programming Report

Group 11

Febri Fransisca – 2540129193

Jaysie Lestari – 2501966206

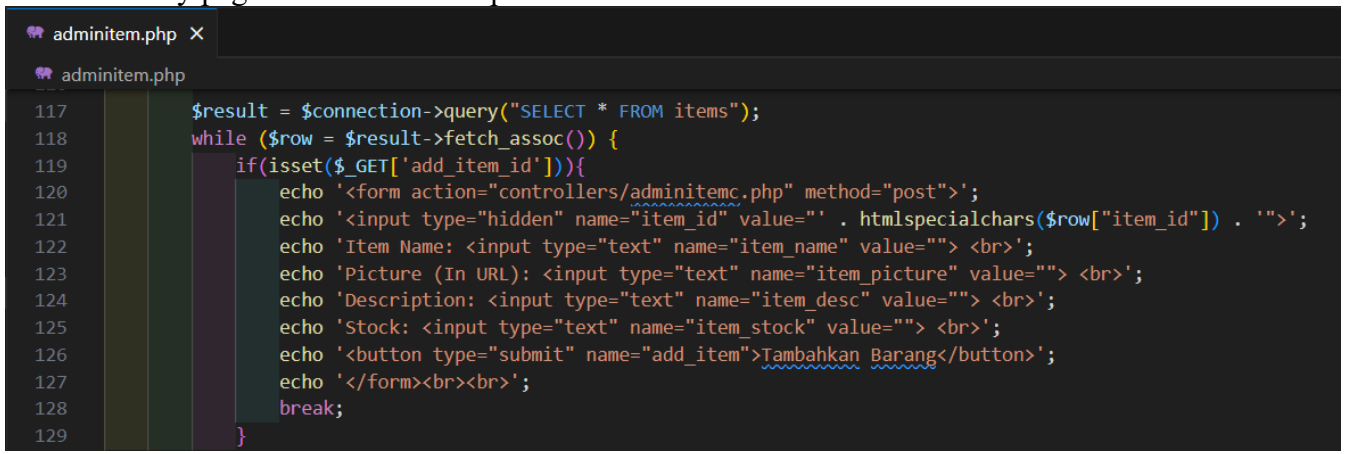
Rika Zakiyyah – 2540134944

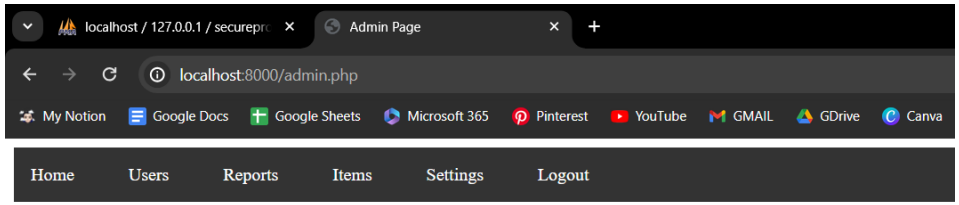
Vivian Dwitama – 2501962870

Wiryahadi Gunawan – 2540131292

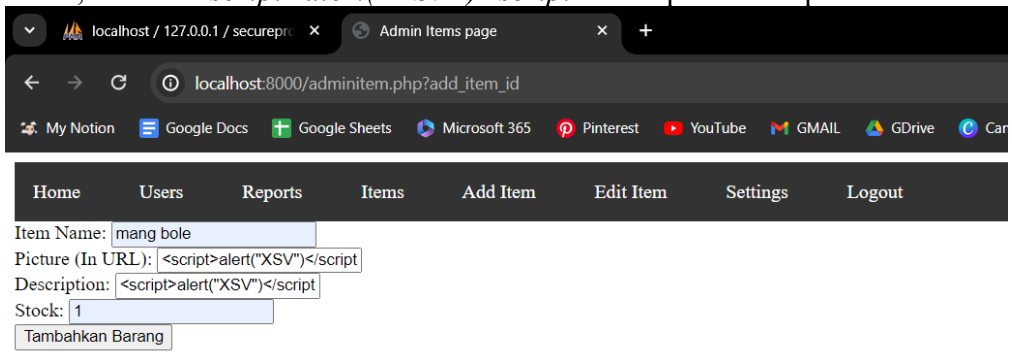
I. Code Audit (Code Auditor POV)

1) Stored XSS

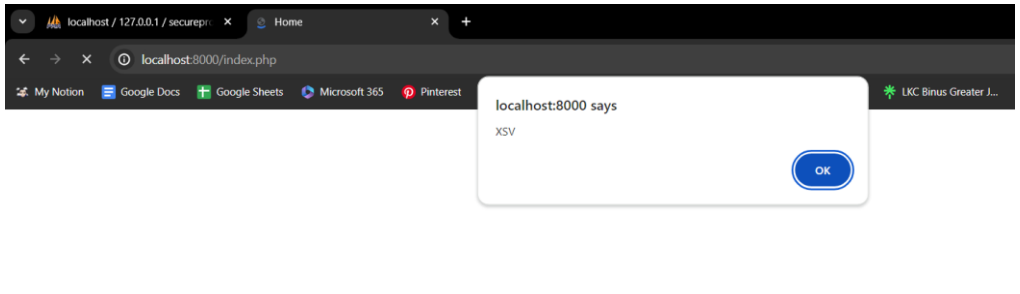
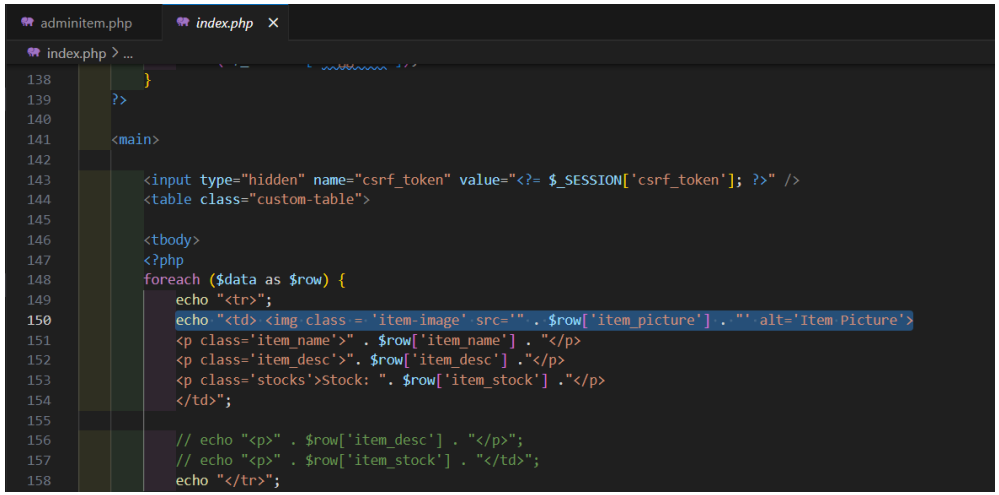
Vulnerability	*Stored XSS (Cross Site Scripting)*
Executive Summary	All pages input for user activity were sanitized from HTML tag except for admin's pages.
POC	First, check HTML tag validation in every page. We found there is no validation in *adminitem.php* where this is the only page that admin can input item.  Second, we need to check dynamically by inject simple script as the input. Before that, we need to login as admin.



Third, insert *"<script>alert("XSV")<script>"* script to the input.

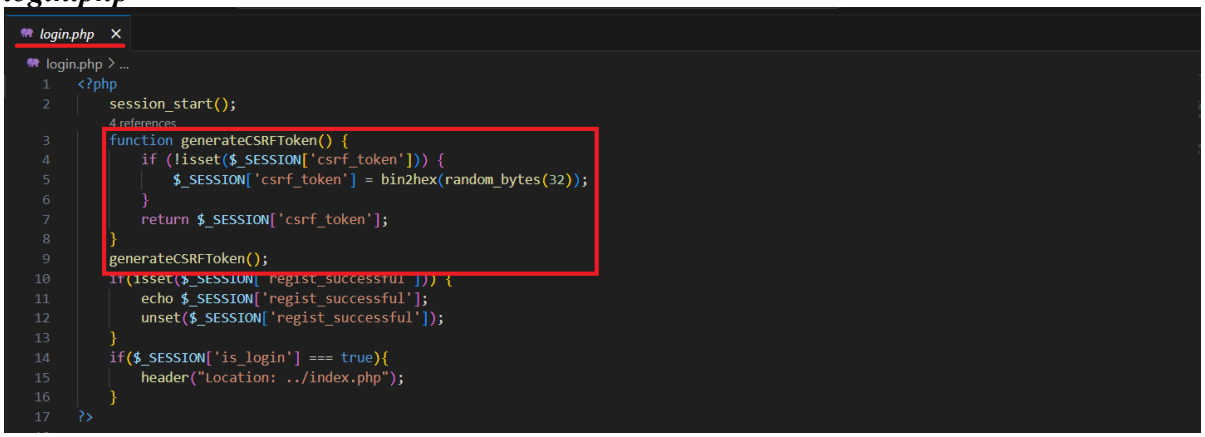


Fourth, it showed that the script is executed whenever we open the home page. Based on *index.php* code, we found out that the image resource is taken from the database so the script that stored in database run altogether.

	 
Recommendation	Give HTML tag sanitation in every input in both user and admin side.

2) CSRF

Vulnerability	*CSRF (Cross Site Request Forgery)*
Executive Summary	Although some pages have CSRF Token generators, **there aren't any on the important pages** where users can change, add, or remove data.

POC	First, let's see the pages that have CSRF Token generators. **1. *login.php***  **2. *register.php*** 

```php
<?php
    session_start();
    function generateCSRFToken() {
        if (!isset($_SESSION['csrf_token'])) {
            $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
        }
        return $_SESSION['csrf_token'];
    }
    generateCSRFToken();
    if(isset($_SESSION['regist_successful'])) {
        echo $_SESSION['regist_successful'];
        unset($_SESSION['regist_successful']);
    }
    if($_SESSION['is_login'] === true){
        header("Location: ../index.php");
    }
?>
```

```php
<?php
    session_start();
    function generateCSRFToken() {
        if (!isset($_SESSION['csrf_token'])) {
            $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
        }
        return $_SESSION['csrf_token'];
    }
    generateCSRFToken();
?>
```

3. *index.php*

index.php

index.php > ...

```php
<?php
    session_start();
    function generateCSRFToken() {
        if (!isset($_SESSION['csrf_token'])) {
            $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
        }
        return $_SESSION['csrf_token'];
    }

    function loginstatus(){
        if(!isset($_SESSION['is_login'])) {
            $_SESSION['is_login'] = false;
        }
        return $_SESSION['is_login'];
    }
    function adminstatus(){
        if(!isset($_SESSION['is_admin'])) {
            $_SESSION['is_admin'] = false;
        }
        return $_SESSION['is_admin'];
    }
    generateCSRFToken();
    loginstatus();
    adminstatus();
    if(isset($_SESSION['regist_successful'])) {
        echo $_SESSION['regist_successful'];
        unset($_SESSION['regist_successful']);
    }
    require(__DIR__ . '/controllers/indexConnection.php');
?>
```

4. ***about-us.php***

```php
<?php
    session_start();
    4 references
    function generateCSRFToken() {
        if (!isset($_SESSION['csrf_token'])) {
            $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
        }
        return $_SESSION['csrf_token'];
    }
    2 references
    function loginstatus(){
        if(!isset($_SESSION['is_login'])) {
            $_SESSION['is_login'] = false;
        }
        return $_SESSION['is_login'];
    }
    2 references
    function adminstatus(){
        if(!isset($_SESSION['is_admin'])) {
            $_SESSION['is_admin'] = false;
        }
        return $_SESSION['is_admin'];
    }
    generateCSRFToken()
    loginstatus();
    adminstatus();
?>
```

As we can see, the pages above do implement a CSRF Token generator. Now, let's see the crucial pages that lacks it.

1. ***report.php*** – used for Add report

```php
<?php
    session_start();
    if (!isset($_SESSION['is_login']) || $_SESSION['is_login'] !== true) {
        header("Location: login.php");
    }
?>
```

2. ***adminitem.php*** – used for Add, Update, and Delete item

adminitem.php

```php
<?php
    session_start();
    require_once(__DIR__ . '/controllers/connection.php');
    if($_SESSION['is_admin'] !== true){
        header("Location: login.php");
    }
?>
```

3. ***settings.php*** – used for Update profile

settings.php

```php
<?php

session_start();
require_once(__DIR__ . '/controllers/connection.php');

if ($connection->error) {
    die($connection->error);
}

if (!isset($_SESSION['is_login']) || $_SESSION['is_login'] !== true) {
    header("Location: login.php");
    exit;
}
?>
```

4. ***admin.php*** – used for Delete report

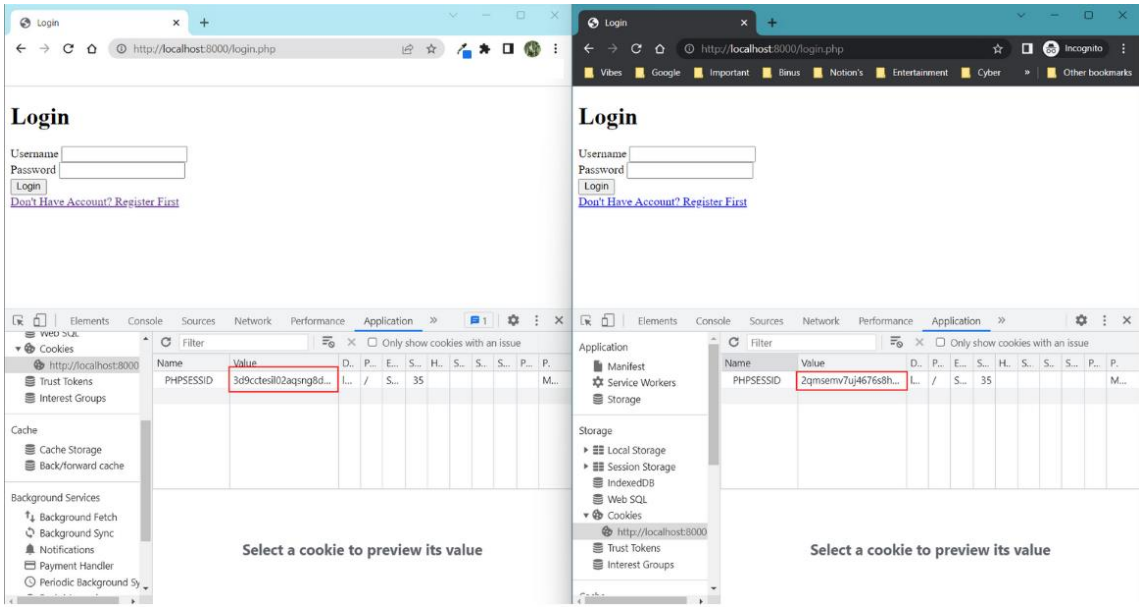
admin.php

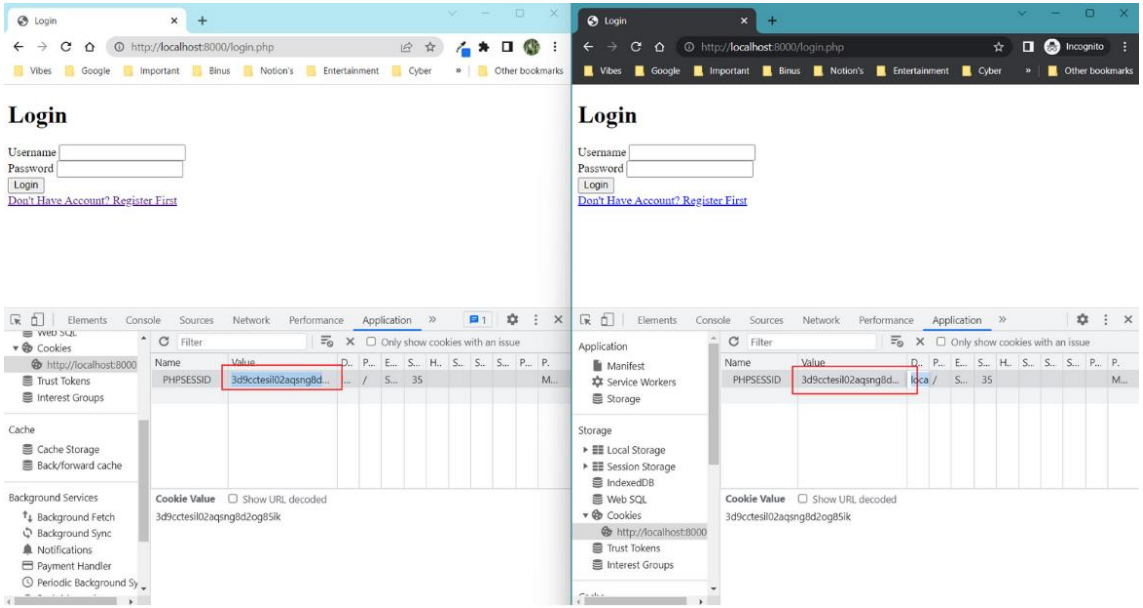
```php
<?php
    session_start();
    require_once(__DIR__ . '/controllers/connection.php');
    if ($_SESSION['is_admin'] !== true) {
        header("Location: login.php");
    }

    $report_typelist = array(
        "1" => "Kritik dan Saran",
        "2" => "Pengajuan Keluhan",
        "3" => "Lainnya"
    );

    ?>
```

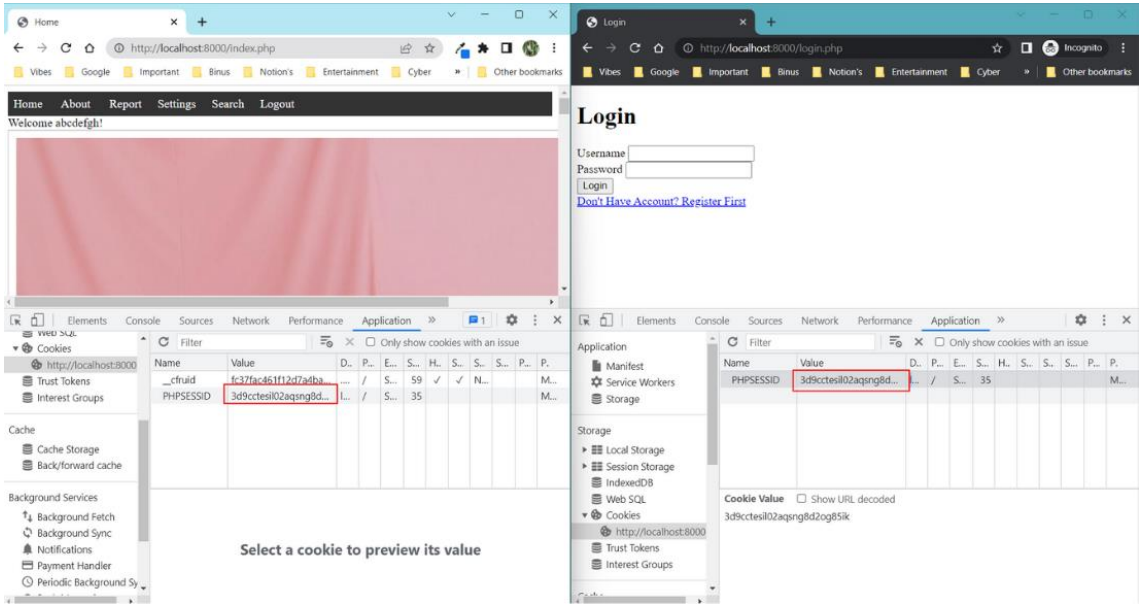
	5. ***adminuser.php*** – used for Delete user  The picture above shows that there is no function that generates a CSRF Token.
Recommendation	Use CSRF Token in every file, especially when there is Create (Add), Update, or Delete operation.

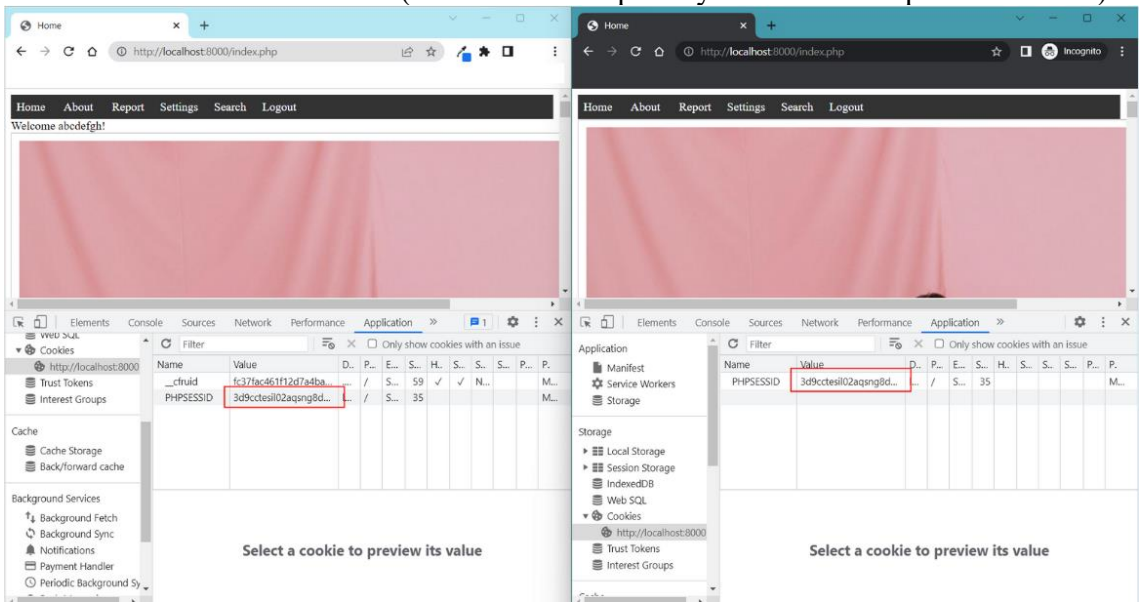
3) Insecure Session Management

Vulnerability	*Session Hijacking*
Executive Summary	The session token does not change when the user login and it also doesn't expire when the user terminates the session (logout). This way, the session token can stolen before the user login and when the user login, the attacker can refresh the page and automatically login using the stolen session token.
POC	1. The left window is User A, meanwhile the right window (the incognito one) is User B. At first, we can see that both users has different session token.

	 Login http://localhost:8000/login.php Login Username Password Login Don't Have Account? Register First Elements Console Sources Network Performance Application Cookies http://localhost:8000 Trust Tokens Interest Groups Cache Cache Storage Back/forward cache Background Services Background Fetch Background Sync Notifications Payment Handler Periodic Background Sy Filter Only show cookies with an issue Name Value PHPSESSID 3d9cctesil02aqsng8d... Select a cookie to preview its value Login http://localhost:8000/login.php Incognito Vibes Google Important Binus Notion's Entertainment Cyber Other bookmarks Login Username Password Login Don't Have Account? Register First Elements Console Sources Network Performance Application Application Manifest Service Workers Storage Storage Local Storage Session Storage IndexedDB Web SQL Cookies http://localhost:8000 Trust Tokens Interest Groups Filter Only show cookies with an issue Name Value PHPSESSID 2qmsemv7uj4676s8h... Select a cookie to preview its value

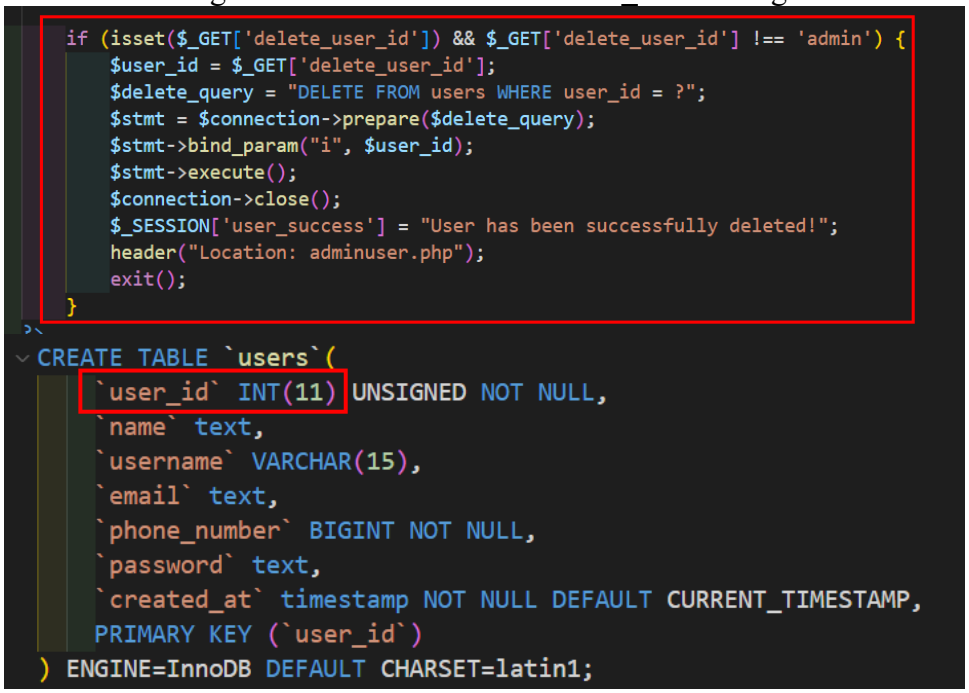
	2. Here, the User A's session token is copied and pasted into the User B's session token. We can see both the users have the same session token now. 

	3. Here, User A login using his/ her Username and Password and redirected to the home page (*index.php*) and both the users still have the same session token. 

	4. After that, the User B refresh the webpage and automatically authenticated using User A's credential due to the same session token. (User B didn't input any username and password at all) 
Recommendation	Regenerate new session token every time user login and logout. To do so, the session_regenerate_id PHP function can be used. This function will automatically regenerate new session token. `session_regenerate_id(true);`

4) IDOR

Vulnerability	*IDOR (Insecure Direct Object Reference)*
Executive Summary	As part of mistake from developer code for failing to validate the request payload is causing the attacker able to do action for admin user only (CRUD) as normal user and able to update other user information

	(email and password) from their own update information page. Hence, as long the attacker knows the endpoint for all actions, they can exploit it as normal user.
POC	There is no validation for request payload that being sent to the backend for admin action 1. Only check if the parameter **"delete_user_id"** has value and not **"admin",** meanwhile their database design for table **"users"** store **"user_id"** as integer. 

```php
if (isset($_GET['delete_user_id']) && $_GET['delete_user_id'] !== 'admin') {
    $user_id = $_GET['delete_user_id'];
    $delete_query = "DELETE FROM users WHERE user_id = ?";
    $stmt = $connection->prepare($delete_query);
    $stmt->bind_param("i", $user_id);
    $stmt->execute();
    $connection->close();
    $_SESSION['user_success'] = "User has been successfully deleted!";
    header("Location: adminuser.php");
    exit();
}
```

```sql
CREATE TABLE `users`(
    `user_id` INT(11) UNSIGNED NOT NULL,
    `name` text,
    `username` VARCHAR(15),
    `email` text,
    `phone_number` BIGINT NOT NULL,
    `password` text,
    `created_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2. There is no validation for checking only the owner of the email able to update their email.

```php
function updateEmail($user_id, $email, $new_email, $connection) {
    if (strlen($new_email) == 0){
        return;
    }else if($new_email === $email){
        return "Email can't be the same!";
    }else if(!filter_var($new_email, FILTER_VALIDATE_EMAIL)){
        return 'Email is not valid!';
    }
    $update_email_sql = $connection->prepare("UPDATE users SET email = ? WHERE user_id = ?");
    $update_email_sql->bind_param("si", $new_email, $user_id);
    if ($update_email_sql->execute()) {
        $update_email_sql->close();
        $_SESSION['email'] = $new_email;
        return "Email berhasil diubah.";
    } else {
        return "Gagal mengubah email.";
    }
}
```

3. Only validate item stock is numeric when add new item and there is no admin validation.

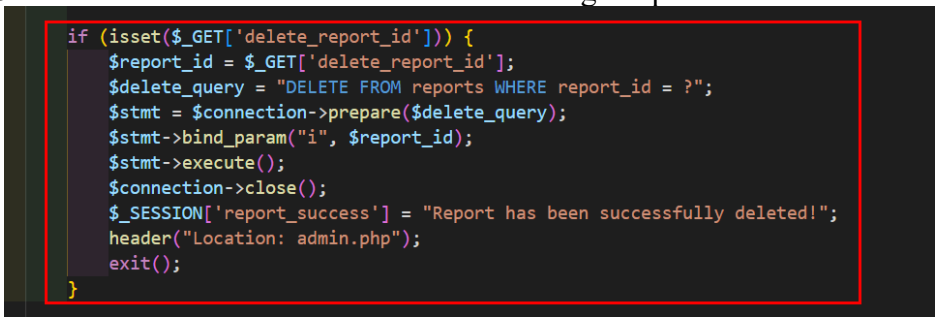
```php
if(isset($_POST['add_item'])){
    $item_id = $_POST['item_id'];
    $item_name = $_POST['item_name'];
    $item_picture = $_POST['item_picture'];
    $item_desc = $_POST['item_desc'];
    $item_stock = $_POST['item_stock'];

    if(is_numeric($item_stock) == 0){
        $_SESSION['item_failed'] = "Stock must be a number!";
        header("Location: ../adminitem.php");
    }

    $insert_query = "INSERT INTO items VALUES (NULL, ?, ?, ?, ?)";
    $stmt = $connection->prepare($insert_query);
    $stmt->bind_param("sssi", $item_name, $item_picture, $item_desc, $item_stock);
    $stmt->execute();
    $connection->close();
    $_SESSION['item_success'] = "Item has been successfully added!";
    header("Location: ../adminitem.php");
}
```

4. Only validate item stock is numeric when update item and there is no admin validation.

```php
if (isset($_POST['edit_item'])) {
    $item_id = $_POST['item_id'];
    $item_name = $_POST['item_name'];
    $item_picture = $_POST['item_picture'];
    $item_desc = $_POST['item_desc'];
    $item_stock = $_POST['item_stock'];

    if(is_numeric($item_stock) == 0){
        $_SESSION['item_failed'] = "Stock must be a number!";
        header("Location: ../adminitem.php");
    }

    $edit_query = "UPDATE items SET item_name = ?, item_picture = ?, item_desc = ?, item_stock = ? WHERE item_id = ?";
    $stmt = $connection->prepare($edit_query);
    $stmt->bind_param("sssii", $item_name, $item_picture, $item_desc, $item_stock, $item_id);
    $stmt->execute();
    $connection->close();
    $_SESSION['item_success'] = "Item has been successfully updated!";
    header("Location: ../adminitem.php");
}
```

5. There is no admin validation when deleting an item.

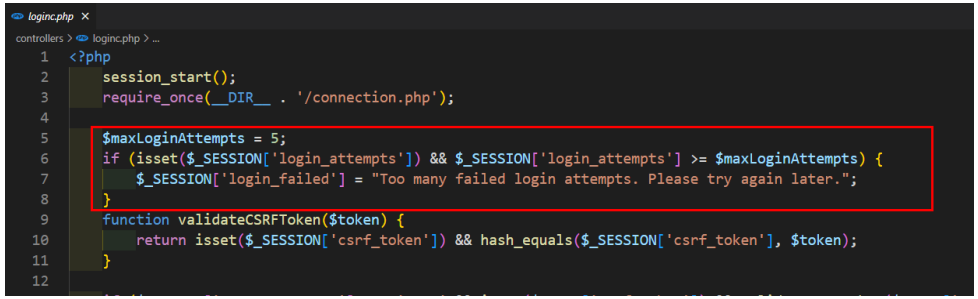
```php
if (isset($_GET['delete_item_id'])) {
    $item_id = $_GET['delete_item_id'];
    $delete_query = "DELETE FROM items WHERE item_id = ?";
    $stmt = $connection->prepare($delete_query);
    $stmt->bind_param("i", $item_id);
    $stmt->execute();
    $connection->close();
    $_SESSION['item_success'] = "Item has been successfully deleted!";
    header("Location: adminitem.php");
}
?>
```

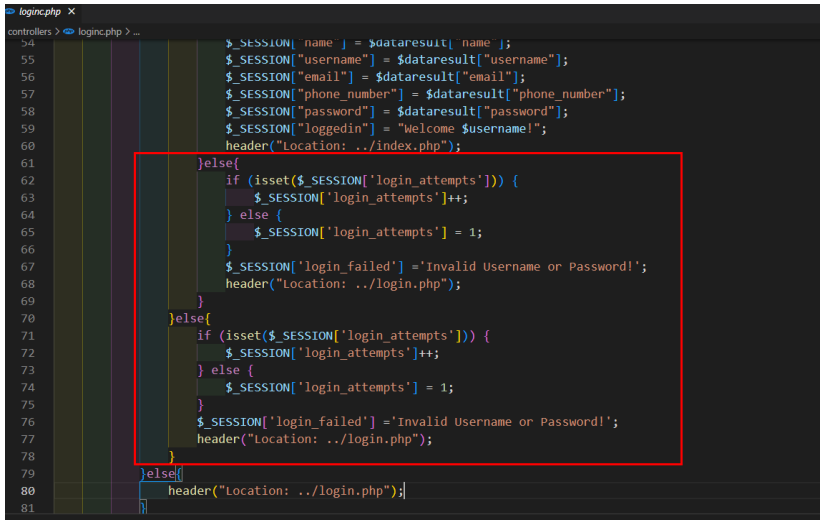
	6. There is no admin validation when deleting a report.
	 
	`if (isset($_GET['delete_report_id'])) {`
	`  $report_id = $_GET['delete_report_id'];`
	`  $delete_query = "DELETE FROM reports WHERE report_id = ?";`
	`  $stmt = $connection->prepare($delete_query);`
	`  $stmt->bind_param("i", $report_id);`
	`  $stmt->execute();`
	`  $connection->close();`
	`  $_SESSION['report_success'] = "Report has been successfully deleted!";`
	`  header("Location: admin.php");`
	`  exit();`
	`}`

Recommendation	Add backend validation to validate all data from the request payload.

5) No Rate Limit

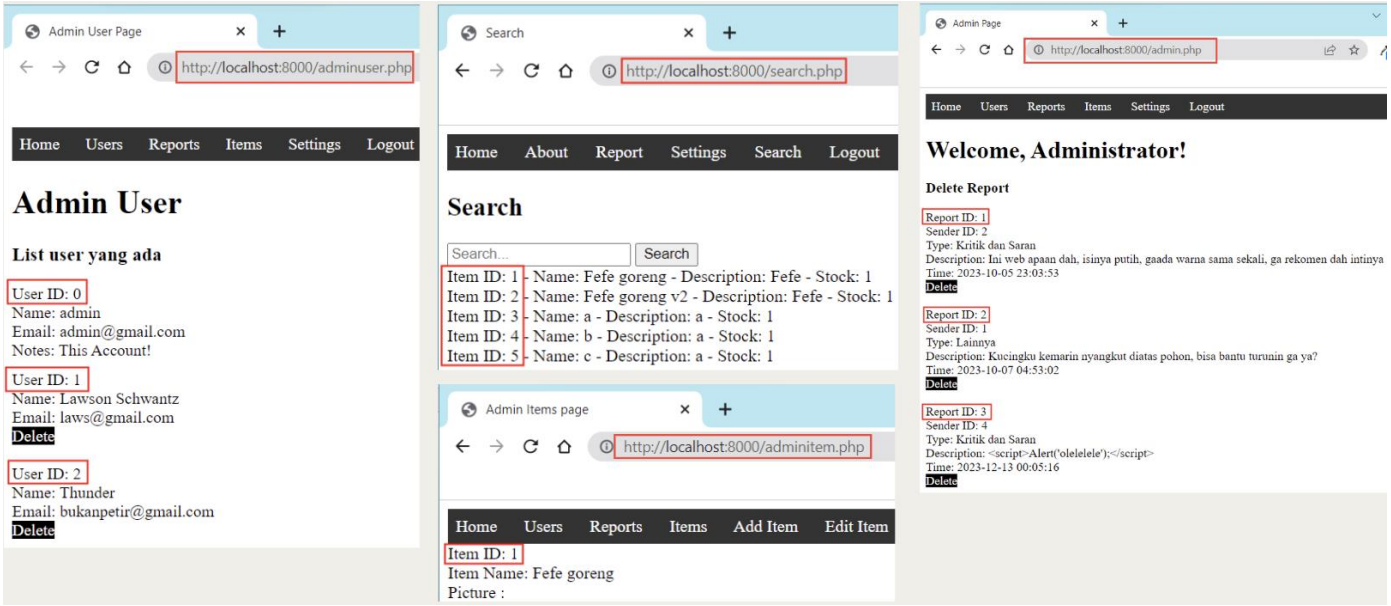
Vulnerability	*Brute-Force Attacks*
Executive Summary	The attacker can brute force login page to guess the correct credential since the developer only counts the attempt in browser session. Hence, if the attacker attempts to brute force using a different browser for each attempt or changing the browser session token, the counter will always start from 0.

POC	
	loginc.php
	controllers > loginc.php > ...
	`<?php`
	`  session_start();`
	`  require_once(__DIR__ . '/connection.php');`
	`  $maxLoginAttempts = 5;`
	`  if (isset($_SESSION['login_attempts']) && $_SESSION['login_attempts'] >= $maxLoginAttempts) {`
	`  $_SESSION['login_failed'] = "Too many failed login attempts. Please try again later.";`
	`  }`
	`  function validateCSRFToken($token) {`
	`  return isset($_SESSION['csrf_token']) && hash_equals($_SESSION['csrf_token'], $token);`
	`  }`

	
Recommendation	Instead of setting/saving the number of user attempts in the session, it is better to save the attempts to the database.

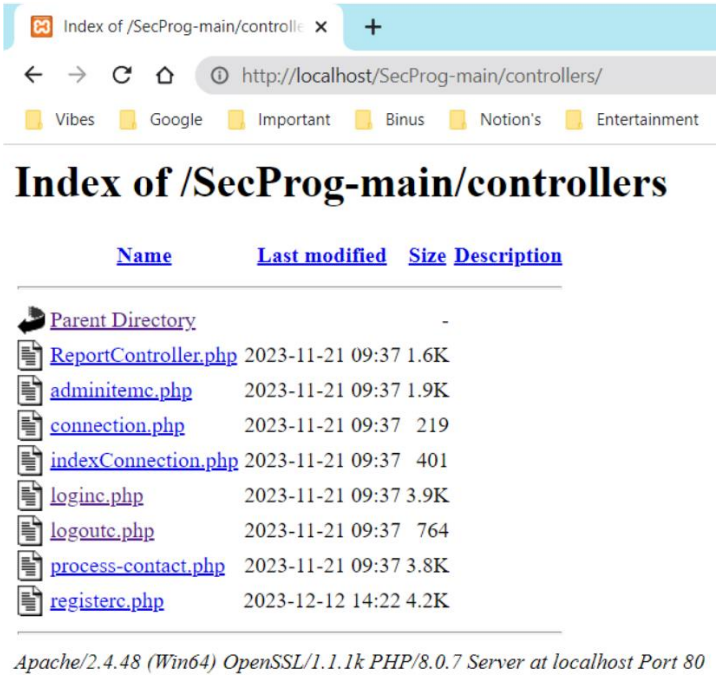
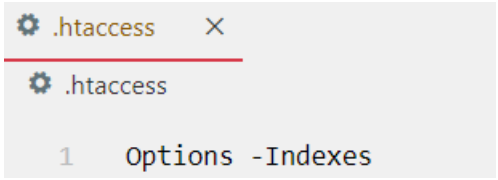
6) Information Disclosure (ID was Shown)

Vulnerability	*Information Disclosure*
Executive Summary	The user identification (ID) is openly displayed on the *adminuser.php*, *search.php*, *admin.php*, and *adminitem.php* pages, which is not appropriate. This could result in the disclosure of information and create a possible security threat. Attackers might misuse this ID information for unauthorized purposes.

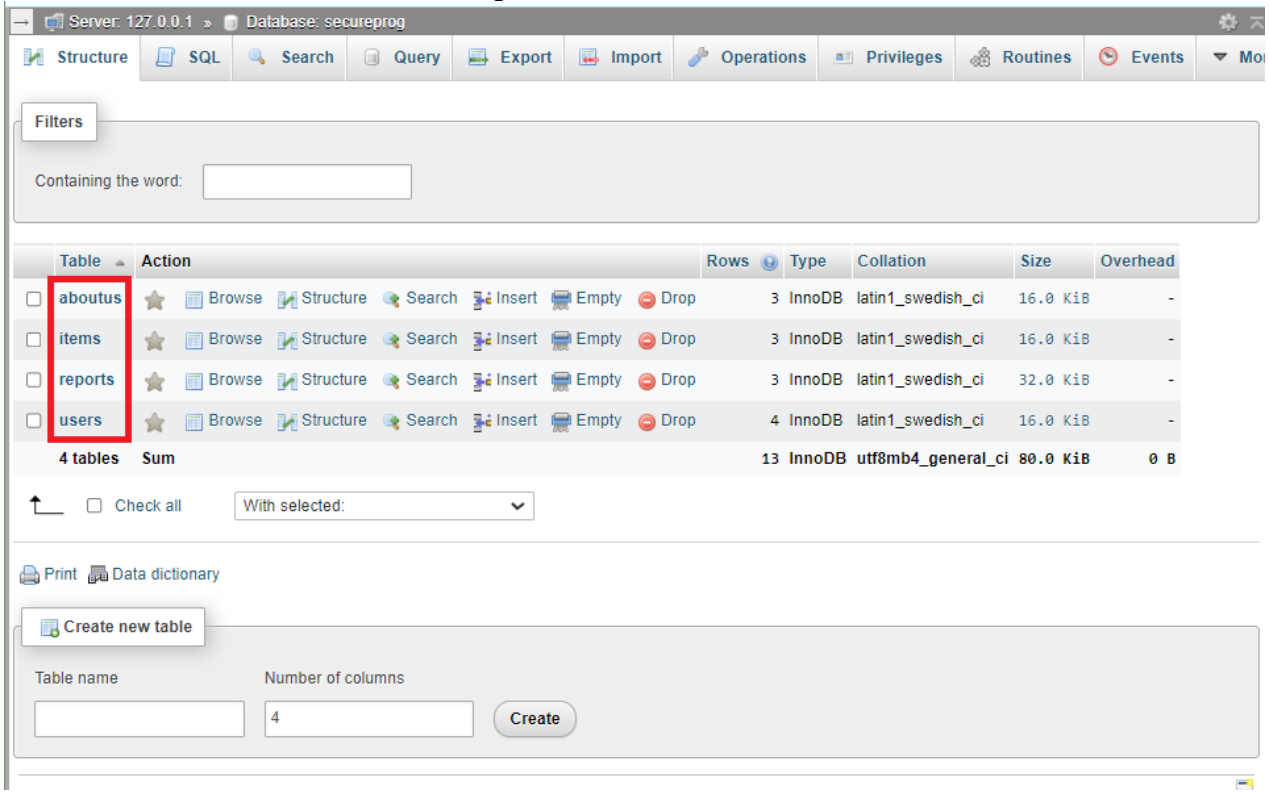
POC	
Recommendation	Never show unnecessary information, such as user identification (ID), that can lead to potential threats.

7) Directory Listing

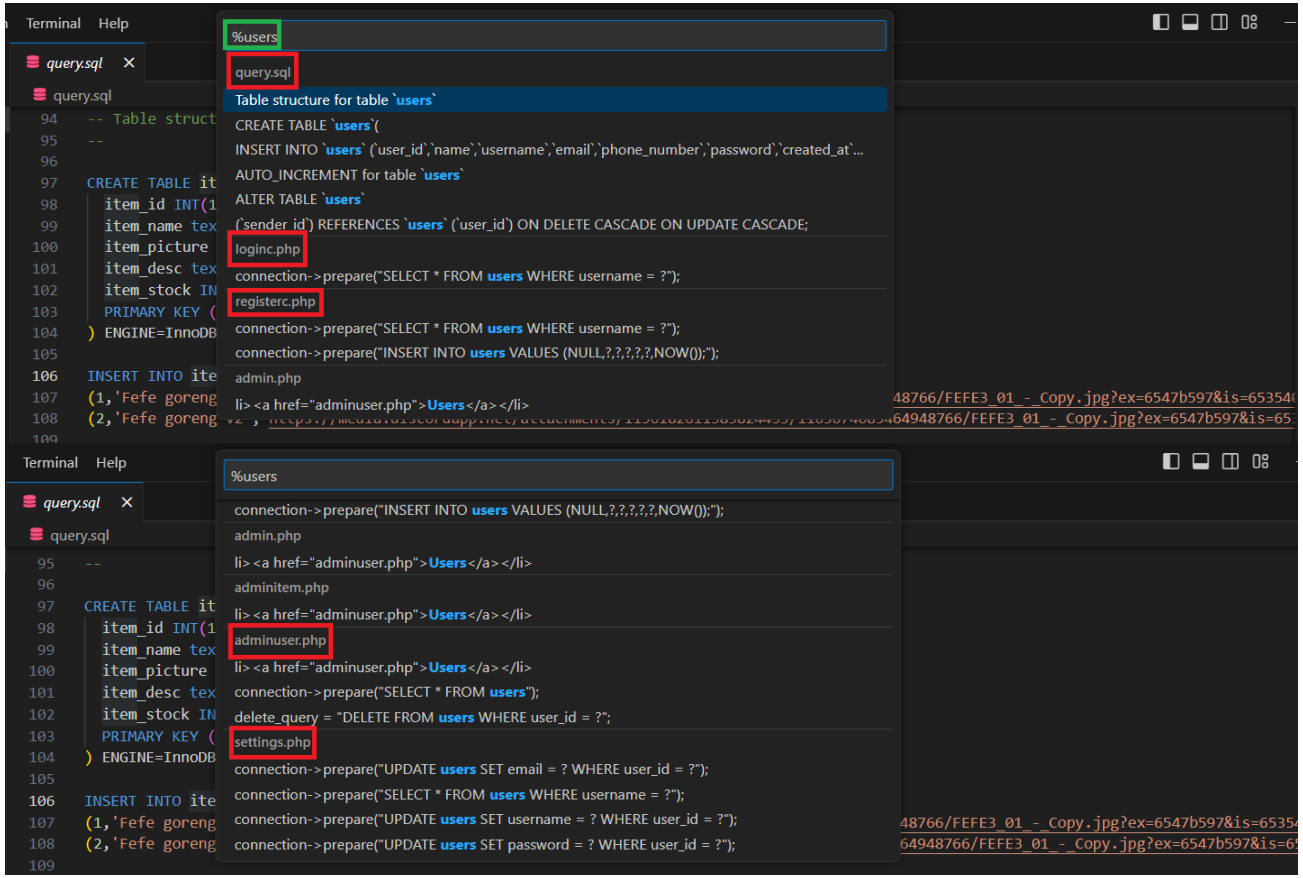
Vulnerability	*Directory Listing*
Executive Summary	When opened using Web Server, in this case, opened using Apache from xampp, all the user and admin's directories and files is shown regardless whether the person login is admin or user. This can lead to the exposure of sensitive files that are not intended to be accessible (***information disclosure***).

POC	
Recommendation	Although this is an auditor-induced vulnerability because the web server is not configured to prevent directory listing, it is better to take precautions using a .htaccess file to prevent directory logging regardless of whether the web server is configured or not. To do so, the “Option -Indexes” command can be used in the .htaccess file. 

8) Unnecessary and Unused Function

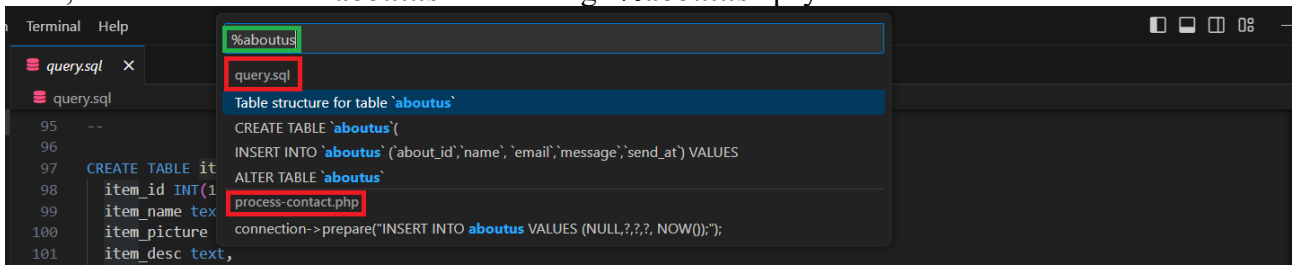
Vulnerability	*Unnecessary and Unused Function*
Executive Summary	The '***aboutus***' table is only used for insertion and **no other operations** are performed on it, this results in an unnecessary and unused function.
POC	To spot the difference, let's see another table first from the database. Altogether there are 4 tables, which are '***aboutus***', '***items***', '***users***', and '***reports***'. 

For comparison, let's try searching for '***users***' table in the source code using "***%users***" payload in the search bar on visual studio code. This is done to see where this table is used.



We can see from the images above, users table is used in multiple pages like in:

- '*login.php*', used to select username data from the table
- '*registerc.php*', used to select and insert data into the table
- '*adminuser.php*', used to select and delete data from the table
- '*settings.php*', used to select and update data from the table

	Now, let's search for the '***aboutus***' table using "***%aboutus***" payload in the search bar in visual studio code.  As we can see, the '***aboutus***' table is only used for insertion in '*process-contact.php*' and it is not used anywhere else.
Recommendation	If there is no use for it, it is better not to create the database and do insertion on it.

II. Link Remediation GitHub Developer's

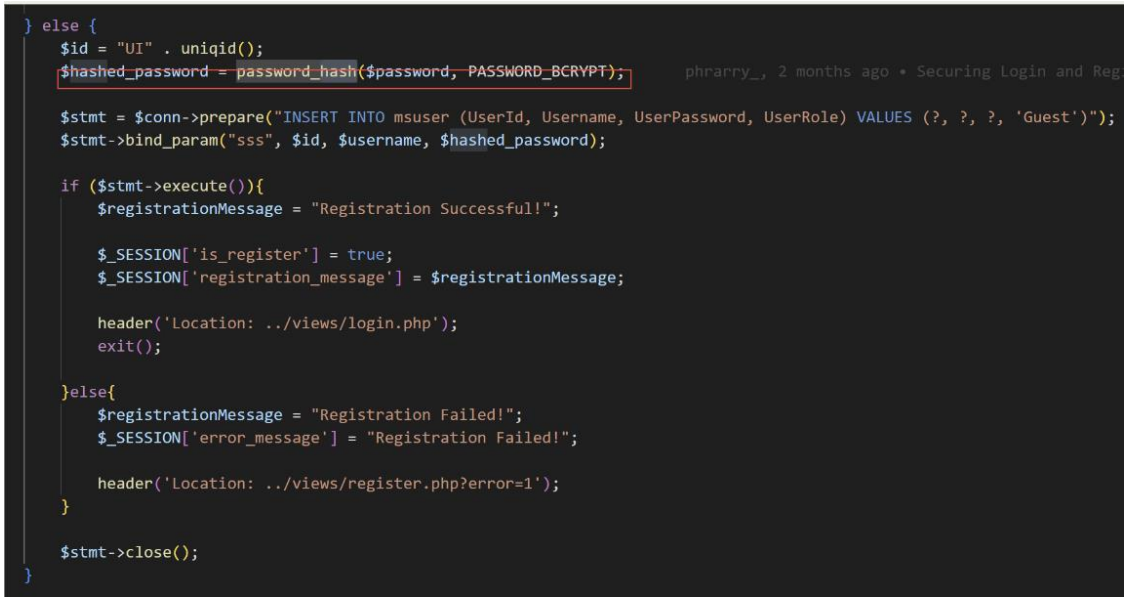
https://github.com/jayjayjay-jj/Secure_Programming

III. Developer's Remediation Explanation

3.1 Before Remediation

1) Hashed Password

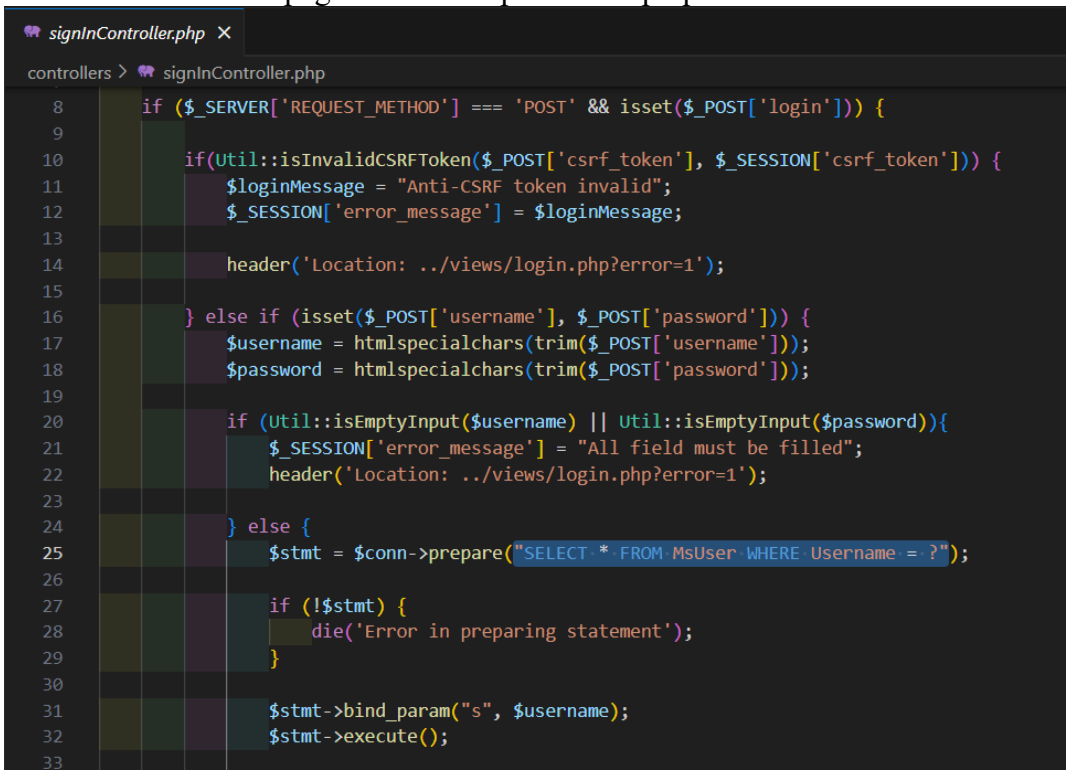
Prevented Vulnerability	*Data Breaches*
Executive Summary	Hashing passwords to ensure that actual passwords cannot be stored in plain text makes it difficult for attackers to retrieve them and can prevent unauthorized access and data breaches.

POC	We've implemented password hashing in the sign-up process. *$hashed_password = password_hash($password, PASSWORD_BCRYPT);*  Where a unique user ID is created by adding "UI" and a unique date using the function uniqid(). And the password entered by the user is encrypted using the password_hash function and the PASSWORD_BCRYPT algorithm. The hash value is then stored in the database along with the user's ID and name.

```php
} else {
    $id = "UI" . uniqid();
    $hashed_password = password_hash($password, PASSWORD_BCRYPT);      phrarry_, 2 months ago • Securing Login and Reg

    $stmt = $conn->prepare("INSERT INTO msuser (UserId, Username, UserPassword, UserRole) VALUES (?, ?, ?, 'Guest')");
    $stmt->bind_param("sss", $id, $username, $hashed_password);

    if ($stmt->execute()){
        $registrationMessage = "Registration Successful!";

        $_SESSION['is_register'] = true;
        $_SESSION['registration_message'] = $registrationMessage;

        header('Location: ../views/login.php');
        exit();

    }else{
        $registrationMessage = "Registration Failed!";
        $_SESSION['error_message'] = "Registration Failed!";

        header('Location: ../views/register.php?error=1');
    }

    $stmt->close();
}
```

2) Prepared Statement

Prevented Vulnerability	*SQLi (SQL Injection)*
Executive Summary	Unsanitized input can lead into database manipulation such as access, modify, or delete data.

POC	To prevent SQL injection, we use prepared statement. We specify statement template such as "SELECT * FROM MsUser WHERE Username = ?" (as we can see in signInController.php) for later use. Here is the pages that we implemented prepared statement :  ***signInController.php***

```php
signInController.php
controllers > signInController.php
8   if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['login'])) {
9
10      if(Util::isInvalidCSRFToken($_POST['csrf_token'], $_SESSION['csrf_token'])) {
11          $loginMessage = "Anti-CSRF token invalid";
12          $_SESSION['error_message'] = $loginMessage;
13
14          header('Location: ../views/login.php?error=1');
15
16      } else if (isset($_POST['username'], $_POST['password'])) {
17          $username = htmlspecialchars(trim($_POST['username']));
18          $password = htmlspecialchars(trim($_POST['password']));
19
20          if (Util::isEmptyInput($username) || Util::isEmptyInput($password)){
21              $_SESSION['error_message'] = "All field must be filled";
22              header('Location: ../views/login.php?error=1');
23
24          } else {
25              $stmt = $conn->prepare("SELECT * FROM MsUser WHERE Username = ?");
26
27              if (!$stmt) {
28                  die('Error in preparing statement');
29              }
30
31              $stmt->bind_param("s", $username);
32              $stmt->execute();
33
```

signUpController.php

controllers > signUpController.php > PHP > checkUsername()

```php
1 reference
function checkUsername($username) {
    global $conn;

    $query = "SELECT * FROM MsUser WHERE Username = ?";
    $stmt = $conn->prepare($query);
    $stmt->bind_param("s", $username);
    $stmt->execute();

    $result = $stmt->get_result();

    if($result->num_rows == 1) {
        return false;
    }

    return true;
}
```

signUpController.php

signUpController.php | *listMedicineController.php*

controllers > listMedicineController.php > PHP > getAllMedicine()

```php
<?php
    require "connection.php";

    1 reference
    function getAllMedicine(){
        global $conn;
        $q = "select * from msmedicine;";
        $d = $conn->query($q);
        return $d;
    }
?>
```

listMedicineController.php

signUpController.php | *adminController.php* ×

controllers > adminController.php > PHP > Admin > GetAllMedicine()

```
2 references | 0 overrides
7   static function GetAllMedicine(){
8       global $conn;
9       $q = "select * from msmedicine;";
10      $d = $conn->query($q);
11      return $d;
12  }
```

signUpController.php | adminController.php ×

controllers > adminController.php > PHP > Admin > GetMedicineByID()

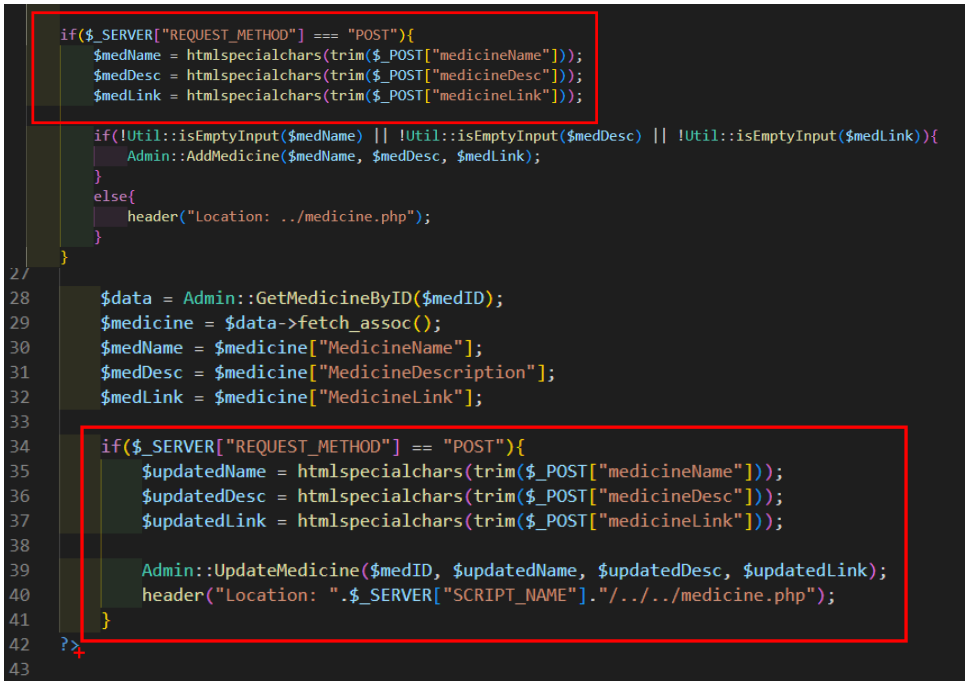
```
13
    1 reference | 0 overrides
14  static function GetMedicineByID($medID){
15      global $conn;
16      $stmt = $conn->prepare("select * from msmedicine where MedicineID = ?");
17      $stmt->bind_param("s", $medID);
18      $stmt->execute();
19      $res = $stmt->get_result();
20      $stmt->close();
21      return $res;
22  }
23
```

adminController.php

3) htmlspecialchars() and trim()

Prevented Vulnerability	*XSS (Cross Site Scripting)*
Executive Summary	We've implemented *htmlspecialchars*() and *trim*() before we process all data from the request payload on our website to prevent XSS attack, where *htmlspecialchars*() will convert every html special character (<, >, etc.) on the input into html entities (&lt, &gt, etc.), and *trim*() to remove leading and trailing space.
POC	

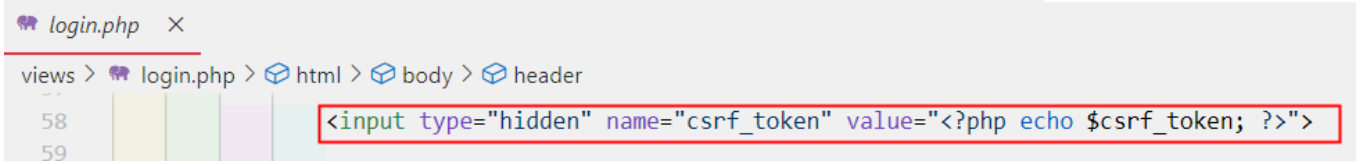
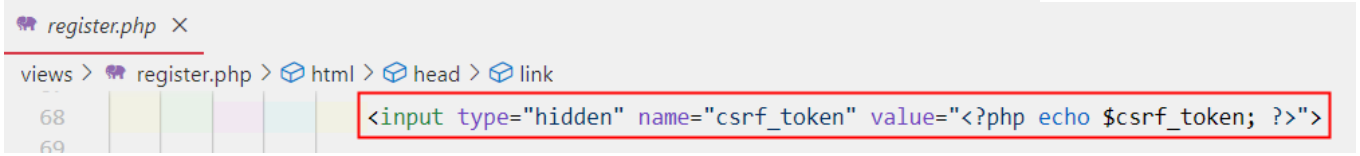
```php
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['login'])) {
    $userKey = md5($_SERVER['REMOTE_ADDR']);

    if(Util::isInvalidCSRFToken($_POST['csrf_token'], $_SESSION['csrf_token'])) {
        $loginMessage = "Anti-CSRF token invalid";
        $_SESSION['error_message'] = $loginMessage;

        header('Location: ../views/login.php?error=1');

    } else if (isset($_POST['username'], $_POST['password'])) {
        $username = htmlspecialchars(trim($_POST['username']));
        $password = htmlspecialchars(trim($_POST['password']));

        // rate Limit (5 attempts, 15 minutes)
        if (rateLimit(5, 900)) {
            if (Util::isEmptyInput($username) || Util::isEmptyInput($password)){
                $_SESSION['error_message'] = "All field must be filled";
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['register'])) {

    if (isset($_POST['username'], $_POST['password'])) {
        $username = htmlspecialchars(trim($_POST['username']));
        $password = htmlspecialchars(trim($_POST['password']));
        $confPassword = htmlspecialchars(trim($_POST['conf-password']));

        if(Util::isInvalidCSRFToken($_POST['csrf_token'], $_SESSION['csrf_token'])) {
            $registrationMessage = "Anti-CSRF token invalid";
            $_SESSION['error_message'] = $registrationMessage;

            header('Location: ../views/register.php?error=1');
```



```php
if($_SERVER["REQUEST_METHOD"] === "POST"){
    $medName = htmlspecialchars(trim($_POST["medicineName"]));
    $medDesc = htmlspecialchars(trim($_POST["medicineDesc"]));
    $medLink = htmlspecialchars(trim($_POST["medicineLink"]));

    if(!Util::isEmptyInput($medName) || !Util::isEmptyInput($medDesc) || !Util::isEmptyInput($medLink)){
        Admin::AddMedicine($medName, $medDesc, $medLink);
    }
    else{
        header("Location: ../medicine.php");
    }
}

    $data = Admin::GetMedicineByID($medID);
    $medicine = $data->fetch_assoc();
    $medName = $medicine["MedicineName"];
    $medDesc = $medicine["MedicineDescription"];
    $medLink = $medicine["MedicineLink"];

    if($_SERVER["REQUEST_METHOD"] == "POST"){
        $updatedName = htmlspecialchars(trim($_POST["medicineName"]));
        $updatedDesc = htmlspecialchars(trim($_POST["medicineDesc"]));
        $updatedLink = htmlspecialchars(trim($_POST["medicineLink"]));

        Admin::UpdateMedicine($medID, $updatedName, $updatedDesc, $updatedLink);
        header("Location: ".$_SERVER["SCRIPT_NAME"]."/../../medicine.php");
    }
?>
```

4) CSRF Token

Prevented Vulnerability	*CSRF (Cross Site Request Forgery)*
Executive Summary	CSRF, a very common web attack where the web can be exploited with unauthorized requests via other unverified websites because it allows the attacker to avoid the same origin policy and cause the website to interfere with other websites. With the fact that the browsers automatically give the attacker any relevant cookie or session token when the attacker has no session token, the attacker can easily perform this CSRF attack. That's why, this CSRF Token is used as a unique token that is included in the forms that will also be sent along with every request. The server will verify whether the request payload is from a legitimate source or not using this CSRF Token.

POC	We implement the CSRF Token in every pages and CSRF Token validation in every controller pages that is used to do Create (Add), Update, and Delete.  *login.php* page   *register.php* page 

medicine.php page

medicine.php ×

views > medicine.php > ...

```
15  if(!isset($_SESSION['csrf_token'])) {
16      $_SESSION['csrf_token'] = uniqid('token', TRUE);
17  }
18
19  $csrf_token = $_SESSION['csrf_token'];
```

medicine.php ×

views > medicine.php > ...

```
80  <input type="hidden" name="csrf_token" value="<?php echo $csrf_token; ?>">
81
```

addMedicine.php page

addMedicine.php ×

views > medicine > addMedicine.php > ...

```
14
15  if(!isset($_SESSION['csrf_token'])) {
16      $_SESSION['csrf_token'] = uniqid('token', TRUE);
17  }
18
19  $csrf_token = $_SESSION['csrf_token'];
20
```

addMedicine.php ×

views > medicine > addMedicine.php > ...

```
94  <input type="hidden" name="csrf_token" value="<?php echo $csrf_token; ?>">
95
```

updateMedicine.php

views > medicine > updateMedicine.php > ...

```php
14
15      if(!isset($_SESSION['csrf_token'])) {
16          $_SESSION['csrf_token'] = uniqid('token', TRUE);
17      }
18
19      $csrf_token = $_SESSION['csrf_token'];
20
```

updateMedicine.php page

updateMedicine.php

views > medicine > updateMedicine.php > ...

```php
106     <input type="hidden" name="csrf_token" value="<?php echo $csrf_token; ?>">
107
```

signInController.php

controllers > signInController.php > ...

```php
48
49      if(Util::isInvalidCSRFToken($_POST['csrf_token'], $_SESSION['csrf_token'])) {
50          $loginMessage = "Anti-CSRF token invalid";
51          $_SESSION['error_message'] = $loginMessage;
52
53          header('Location: ../views/login.php?error=1');
```

signInController.php page for the login controller

signUpController.php

controllers > signUpController.php > checkUsername

```php
32
33      if(Util::isInvalidCSRFToken($_POST['csrf_token'], $_SESSION['csrf_token'])) {
34          $registrationMessage = "Anti-CSRF token invalid";
35          $_SESSION['error_message'] = $registrationMessage;
36
37          header('Location: ../views/register.php?error=1');
```

signUpController.php page for the register controller

adminController.php page for add medicine controller

```
adminController.php
controllers > adminController.php > ...
36      static function AddMedicine($medName, $medDesc, $medLink){
37          global $conn;
38
39          if(Util::isInvalidCSRFToken($_POST['csrf_token'], $_SESSION['csrf_token'])) {
40              $errorMessage = "Anti-CSRF token invalid";
41              $_SESSION['error_message'] = $errorMessage;
42
43              header('Location: ../views/medicine/addMedicine?error=1');
```

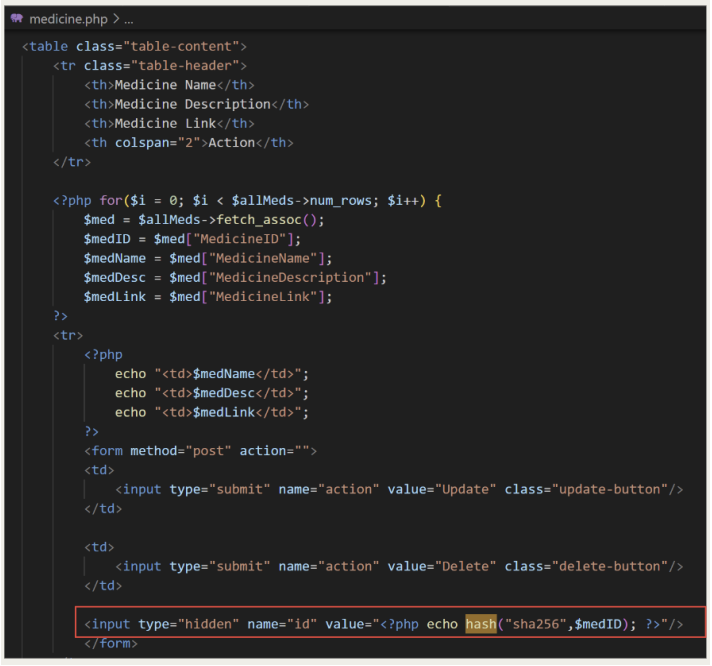
adminController.php page for update medicine controller

```
adminController.php
controllers > adminController.php > ...
61      static function UpdateMedicine($medID, $medName, $medDesc, $medLink){
62          global $conn;
63
64          if(Util::isInvalidCSRFToken($_POST['csrf_token'], $_SESSION['csrf_token'])) {
65              $errorMessage = "Anti-CSRF token invalid";
66              $_SESSION['error_message'] = $errorMessage;
67
68              header('Location: ../views/medicine/updateMedicine?error=1');
69          }
```

adminController.php page for delete medicine controller

5) Hashed MedicineID in Update and Delete Payload

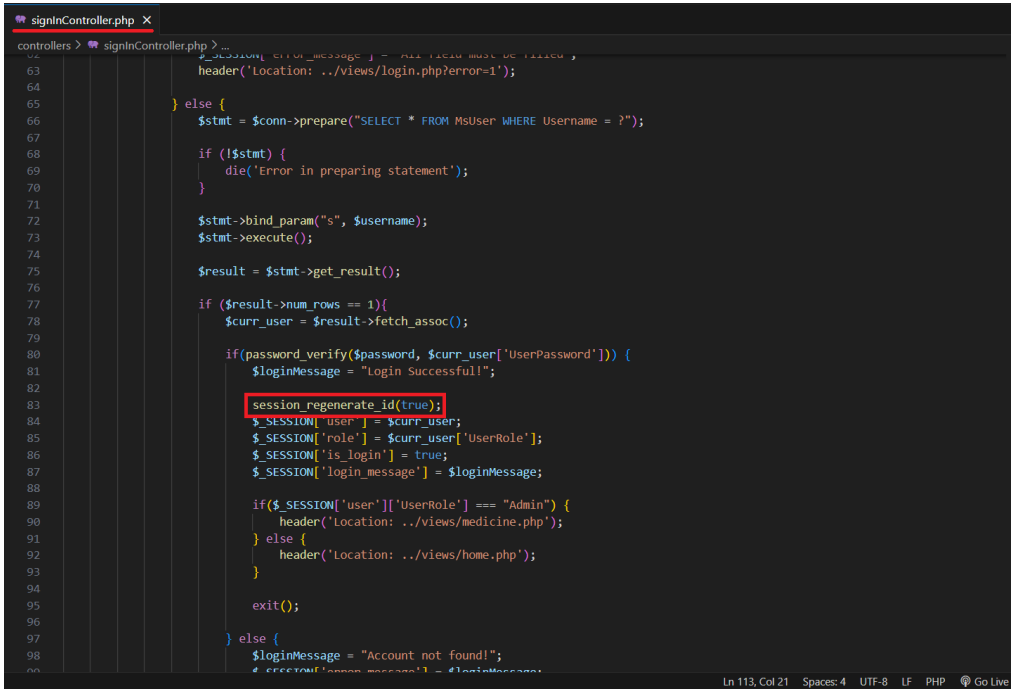
Prevented Vulnerability	*IDOR (Insecure Direct Object Reference)*
Executive Summary	To prevent IDOR (Insecure Direct Object Reference) on the medicine page, we have implemented a hash function using SHA-256 for the MedicineID in both the Update and Delete payloads so that the MedicineID is not exposed as a plain text and it'll be harder for the attacker to guess the MedicineID format that is being used.

POC	We had implemented a hash function using SHA-256 for the MedicineID in both the Update and Delete payloads. *<input type="hidden" name="id" value="<?php echo hash("sha256",$medID); ?>"/>*  The purpose is to secure the MedicineID before passing it as the payload for both update or delete medicine. The SHA-256 hash algorithm turns the original MedicineID into a unique and seemingly random string of characters.

```
medicine.php > ...
<table class="table-content">
    <tr class="table-header">
        <th>Medicine Name</th>
        <th>Medicine Description</th>
        <th>Medicine Link</th>
        <th colspan="2">Action</th>
    </tr>

    <?php for($i = 0; $i < $allMeds->num_rows; $i++) {
        $med = $allMeds->fetch_assoc();
        $medID = $med["MedicineID"];
        $medName = $med["MedicineName"];
        $medDesc = $med["MedicineDescription"];
        $medLink = $med["MedicineLink"];
    ?>
    <tr>
        <?php
            echo "<td>$medName</td>";
            echo "<td>$medDesc</td>";
            echo "<td>$medLink</td>";
        ?>
        <form method="post" action="">
        <td>
            <input type="submit" name="action" value="Update" class="update-button"/>
        </td>

        <td>
            <input type="submit" name="action" value="Delete" class="delete-button"/>
        </td>

        <input type="hidden" name="id" value="<?php echo hash("sha256",$medID); ?>"/>
        </form>
```

6) Session ID Regeneration

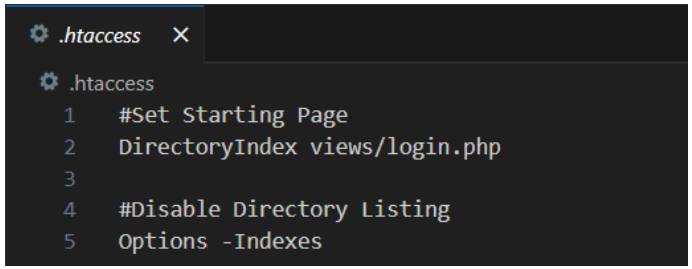
Prevented Vulnerability	*Session Hijacking*

Executive Summary	To prevent session hijacking, we implement a PHP function that regenerates the session id (‘***session_regenerate_id()***’) whenever the user is login in (‘*signInController.php*’) and logout in (‘*logoutController.php*’).
POC	First, after signing in, **if all the credentials are valid and true**, then it will generate a new session id for the user.  Once the user is done and decides to logout, it will check if the user’s session still exists. If it still exists, it will unset() or destroy the value of the variable, which means the user’s existing session id.

	logoutController.php
	controllers > logoutController.php
	`<?php`
	`session_start();`
	
	`if(isset($_SESSION['user'])) {`
	`unset($_SESSION['user']);`
	`}`
	
	`session_regenerate_id(true);`
	`header("Location: ../views/login.php");`
	`?>`
	
	We pass "**true**" as a parameter to delete the old session data immediately after regenerating the ID. This also applies to the one in '*signInController.php*'.

7) Htaccess

Prevented Vulnerability	*Directory Listing*
Executive Summary	Web server directory that displayed publicly can pose significant security risk if the configuration not properly managed, such as information disclosure.
POC	We disabled directory listing with "Options -Indexes" commands in the .htaccess file so user cannot browse content of the website.

3.2 After Remediation

1) Password Policy

Prevented Vulnerability	*Unauthorized Access*
Executive Summary	Previously, we only created a password validation with a minimum length of 8 characters on the controller page for the register page (*signUpController.php*). However, after being audited and reported by another group auditor, we realize that a proper password policy is important for overall security. Thus, we make a password policy that validates the minimum length of 8 characters, consisting of minimum of 1 uppercase letter, 1 lowercase letter, 1 digit, and 1 special character.

POC	register.php signUpController.php M controllers > signUpController.php > ...

```php
} else if (strlen($password) < 8) {
    $_SESSION['error_message'] = "Password must be at least 8 characters";
    header('Location: ../views/register.php?error=1');

} else if(!preg_match('/[A-Z]/', $password)) {
    $_SESSION['error_message'] = "Password must contain at least one uppercase letter.";
    header('Location: ../views/register.php?error=1');

} else if (!preg_match('/[a-z]/', $password)) {
    $_SESSION['error_message'] = "Password must contain at least one lowercase letter.";
    header('Location: ../views/register.php?error=1');

} else if (!preg_match('/\d/', $password)) {
    $_SESSION['error_message'] = "Password must contain at least one digit.";
    header('Location: ../views/register.php?error=1');

} else if (!preg_match('/[^a-zA-Z\d]/', $password)) {
    $_SESSION['error_message'] = "Password must contain at least one special character.";
```

2) Rate Limit

Prevented Vulnerability	*Brute-Force Attacks*
Executive Summary	To prevent brute-force attacks on the login page, we have implemented a rate-limiting function (*rateLimit*()) that keeps track of login attempts and enforces a limit of 5 attempts within a 15-minute time interval.
POC	We have implemented a rate-limiting function to the login page. The process involves the following steps: • The variable *$userKey*, which will store the hashed value of the remote address of host to keep track of the user.

- Checking Attempt Time (*$attemptTime*) : Calculates the amount of time since the last login attempt. If the interval time of 15 minutes is already exceeded, the attempt count is reset, and the login time is updated.
- Checking Attempt Limit: Check if the current login attempts have reached 5 login attemps. If so, it returns false to reject further attempts.
- Updating Attempt Count and Login Time: If the attempt limit has not reached (5 times of login attempts), the attempt count is incremented, and the login time is updated.

3) Error Message

Prevented Vulnerability	*Information Disclosure*
Executive Summary	Previously, in the login page controller (*signInController.php*), we created the error message "User not found" when the username input is wrong and the error message "Password is wrong" when the password is wrong. However, when audited by another group auditor, there is a finding that these error messages caused Information Disclosure because the user can know that either the username or password is wrong. If the password is wrong, the password can be brute-forced. Thus, we fixed both error messages to be the same "Account not found" when either the username input or the password input is wrong.
POC	

4) Session Timeout

Prevented Vulnerability	*Session Hijacking*
Executive Summary	Previously, we only implemented the session token regeneration whenever a user is logged in, but no session timeout for the login user. We realize that if there's no session timeout although the session token is regenerated when the user logs in, there is still risk of session hijacking. That's why the session timeout is implemented so that users can be automatically logged out one hour after inactivity by setting the maximum lifetime of the session and cookie to 3600 seconds (1 hour). If the session is inactive for more than 1 hour, both the session and the session cookie will be expired.
POC	

5) X-Frame-Options

Prevented Vulnerability	*Clickjacking*
Executive Summary	To prevent clickjacking attack, we implement X-Frame-Options header with **DENY** and **SAMEORIGIN** mode, which means: - **DENY**: prevents the page from being displayed in frames on any other domain

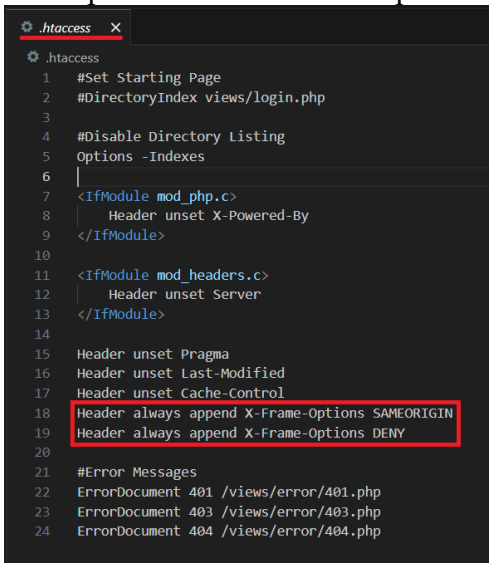
<table>
<tr><td></td><td>

- **SAMEORIGIN**: allows the page to be displayed in frames only if the request is from the same origin (same domain)

By controlling the framing of web content, the **X-Frame-Options** header restrict how a web page can be embedded in frames or iframes on other websites.

</td></tr>
<tr><td>POC</td><td>

We implement the X-Frame-Options header in every page.



```
.htaccess
1  #Set Starting Page
2  #DirectoryIndex views/login.php
3
4  #Disable Directory Listing
5  Options -Indexes
6
7  <IfModule mod_php.c>
8      Header unset X-Powered-By
9  </IfModule>
10
11 <IfModule mod_headers.c>
12     Header unset Server
13 </IfModule>
14
15 Header unset Pragma
16 Header unset Last-Modified
17 Header unset Cache-Control
18 Header always append X-Frame-Options SAMEORIGIN
19 Header always append X-Frame-Options DENY
20
21 #Error Messages
22 ErrorDocument 401 /views/error/401.php
23 ErrorDocument 403 /views/error/403.php
24 ErrorDocument 404 /views/error/404.php
```

.htaccess

```
login.php
views > login.php
1  <?php
2      session_start();
3      header_remove("X-Powered-By");
4      header('X-Frame-Options: DENY, SAMEORIGIN');
5      unset($_SESSION['login_message']);
6
7      if(isset($_SESSION['user']) && $_SESSION['user']['UserRole'] === "Admin") {
8          header("Location: medicine.php");
9      } else if(isset($_SESSION['user']) && $_SESSION['user']['UserRole'] === "Guest") {
10         header("Location: home.php");
11     }
12
13     if(!isset($_SESSION['csrf_token'])) {
14         $_SESSION['csrf_token'] = uniqid('token', TRUE);
15     }
16
17     $csrf_token = $_SESSION['csrf_token'];
18 ?>
```

login.php

```
register.php
views > register.php
1  <?php
2      session_start();
3      header_remove("X-Powered-By");
4      header('X-Frame-Options: DENY, SAMEORIGIN');
5
6      unset($_SESSION['registration_message']);
7
8      if(isset($_SESSION['user']) && $_SESSION['user']['UserRole'] === "Admin") {
9          header("Location: medicine.php");
10     } else if(isset($_SESSION['user']) && $_SESSION['user']['UserRole'] === "Guest") {
11         header("Location: home.php");
12     }
13
14     if(!isset($_SESSION['csrf_token'])) {
15         $_SESSION['csrf_token'] = uniqid('token', TRUE);
16     }
17
18     $csrf_token = $_SESSION['csrf_token'];
19 ?>
```

register.php

</td></tr>
</table>

home.php

views > home.php

```php
<?php
    session_start();
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
    require '../controllers/listMedicineController.php';

    if(!isset($_SESSION['user'])){
        header("Location: login.php");
    }
?>
```

home.php

medicine.php

views > medicine.php

```php
<?php
    session_start();
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
    require '../controllers/adminController.php';

    if(!isset($_SESSION['user'])){
        header("Location: login.php");
    }

    if($_SESSION['user']['UserRole'] !== "Admin") {
        header("Location: ./error/401.php");
    }

    if(!isset($_SESSION['csrf_token'])) {
        $_SESSION['csrf_token'] = uniqid('token', TRUE);
    }

    $csrf_token = $_SESSION['csrf_token'];

    if($_SERVER["REQUEST_METHOD"] === "GET"){
        if(isset($_SERVER["PATH_INFO"])){
            if($_SERVER["PATH_INFO"] === "/" || substr($_SERVER['PATH_INFO'], 1)){
                header("Location: ".$_SERVER["SCRIPT_NAME"]);
            }
        }

        $allMeds = Admin::GetAllMedicine();
    }
    else if($_SERVER["REQUEST_METHOD"] === "POST"){
        if ($_POST["action"] && $_POST["id"]) {
            if ($_POST["action"] == "Update") {
                header("Location: medicine/updateMedicine.php/".$_POST["id"]);
            }
            else if($_POST["action"] == "Delete"){
                Admin::DeleteMedicine($_POST["id"]);
                header("Location: ".$_SERVER["SCRIPT_NAME"]);
```

medicine.php

updateMedicine.php

views > medicine > updateMedicine.php

```php
<?php
    session_start();
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
    require("../../controllers/adminController.php");

    if(!isset($_SESSION['user'])){
        header("Location: ../login.php");
    }

    if($_SESSION["user"]["UserRole"] != "Admin") {
        header("Location: ../../error/401.php");
    }

    if(!isset($_SESSION['csrf_token'])) {
        $_SESSION['csrf_token'] = uniqid('token', TRUE);
    }

    $hashID = substr($_SERVER['PATH_INFO'], 1);
    $medID = Admin::GetMedicineIDByHashID($hashID);

    if($medID == null){
        header("Location: ".$_SERVER["SCRIPT_NAME"]."/../../error/404.php");
    }

    $data = Admin::GetMedicineByID($medID);
    $medicine = $data->fetch_assoc();
    $medName = $medicine["MedicineName"];
    $medDesc = $medicine["MedicineDescription"];
    $medLink = $medicine["MedicineLink"];

    if($_SERVER["REQUEST_METHOD"] == "POST"){
        $updatedName = htmlspecialchars(trim($_POST["medicineName"]));
        $updatedDesc = htmlspecialchars(trim($_POST["medicineDesc"]));
        $updatedLink = htmlspecialchars(trim($_POST["medicineLink"]));

        Admin::UpdateMedicine($medID, $updatedName, $updatedDesc, $updatedLink);
```

updateMedicine.php

index.php

```php
<?php
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
    header("Location: ./views/login.php");
?>
```

index.php

401.php

```php
<?php
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
?>
```

401.php

addMedicine.php

```php
<?php
    session_start();
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
    require("../../controllers/adminController.php");

    if(!isset($_SESSION['user'])){
        header("Location: ../login.php");
    }

    if($_SESSION["user"]["UserRole"] != "Admin") {
        header("Location: ../error/401.php");
    }

    if(!isset($_SESSION['csrf_token'])) {
        $_SESSION['csrf_token'] = uniqid('token', TRUE);
    }

    $csrf_token = $_SESSION['csrf_token'];

    if($_SERVER["REQUEST_METHOD"] === "POST"){
        $medName = htmlspecialchars(trim($_POST["medicineName"]));
        $medDesc = htmlspecialchars(trim($_POST["medicineDesc"]));
        $medLink = htmlspecialchars(trim($_POST["medicineLink"]));

        if(!Util::isEmptyInput($medName) || !Util::isEmptyInput($medDesc)
            Admin::AddMedicine($medName, $medDesc, $medLink);
        }
        else{
            header("Location: ../medicine.php");
        }
    }
?>
```

addMedicine.php

403.php

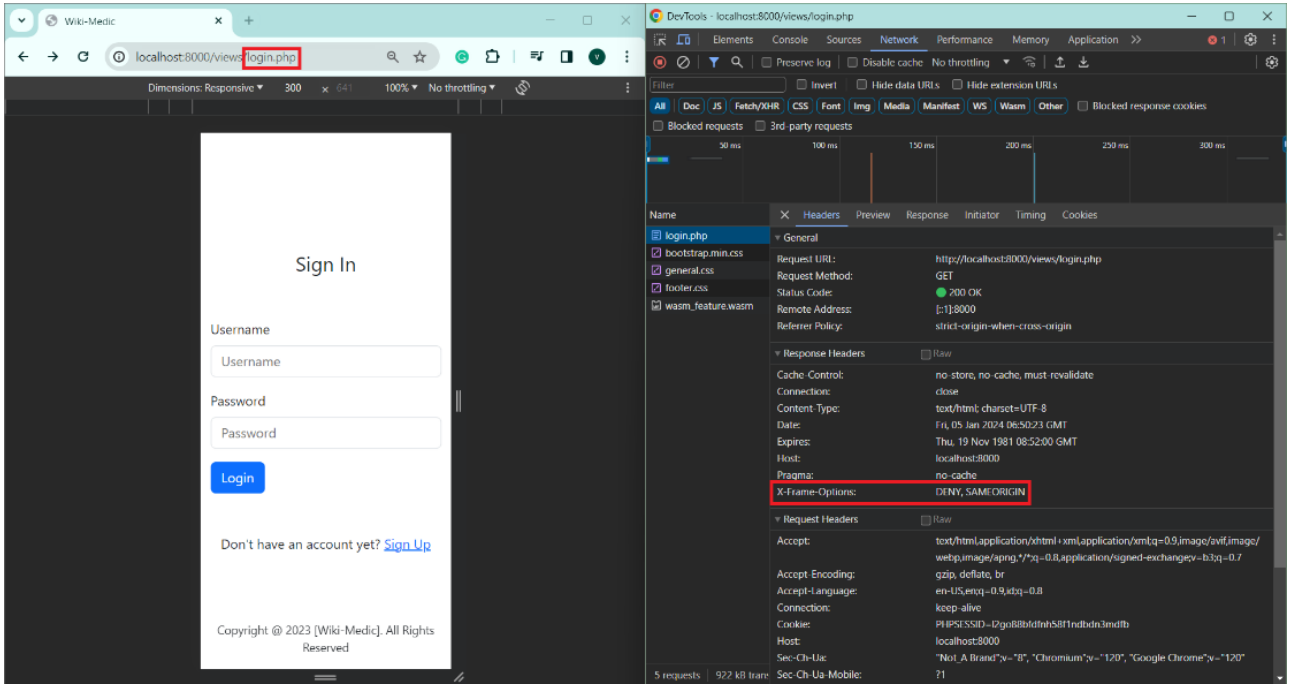
```
views > error > 403.php
1  <?php
2      header_remove("X-Powered-By");
3      header('X-Frame-Options: DENY, SAMEORIGIN');
4  ?>
5
```

404.php

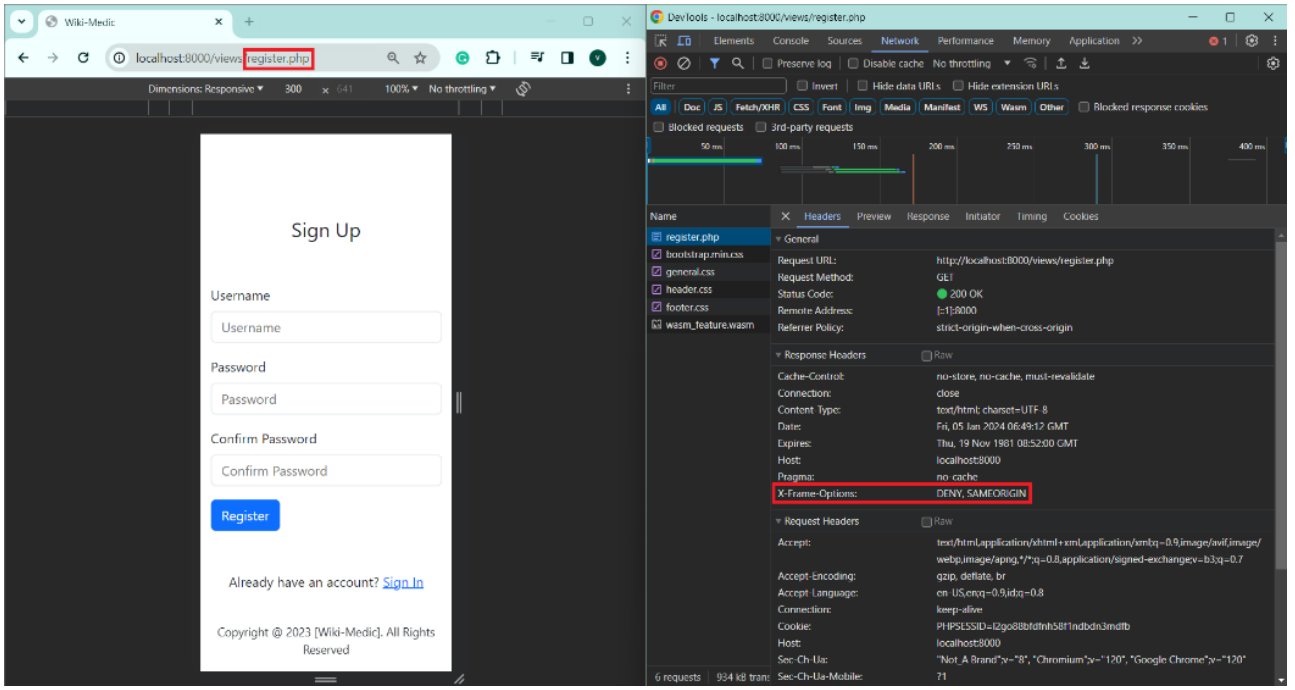
```
views > error > 404.php
1  <?php
2      header_remove("X-Powered-By");
3      header('X-Frame-Options: DENY, SAMEORIGIN');
4  ?>
5
```

403.php ***404.php***

The header is proven to be implemented. This can be seen from the *Header* section when we inspect the website. Let's take two examples: ***login.php*** and ***register.php***.



First, let's inspect ***'login.php'***. If we look into the *Header* section, there is an X-Frame-Options with **DENY** and **SAMEORIGIN** mode.

	 Another example is ***'register.php'***. Same like ***'login.php'***, in the *Header* section there is an X-Frame-Options with **DENY** and **SAMEORIGIN** mode. Same goes for the other pages.

6) Unset PHP and Server version from HTTP header

Prevented Vulnerability	*Information Disclosure*
Executive Summary	Displaying PHP and server version from HTTP header may lead to information disclosure to attacker.
POC	1. Disable PHP version header in every php pages by removing X-Powered-By from the header

401.php

views > error > 401.php > html > body > div.d-flex.align-ite

You, 3 days ago | 2 authors (Druxturd and others)

```php
<?php
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
?>
```

401.php

403.php

views > error > 403.php > ...

You, 3 days ago | 1 author (You)

```php
<?php
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
?>
```

403.php

404.php

views > error > 404.php > html > head > meta

You, 3 days ago | 2 authors (Druxturd and others)

```php
<?php
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
?>
```

404.php

addMedicine.php

views > medicine > addMedicine.php > ...

You, 3 days ago | 2 authors (You and others)

```php
<?php
    session_start();
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
    require("../../controllers/adminController.php");
```

addMedicine.php

updateMedicine.php

views > medicine > updateMedicine.php > html > body > main >

You, yesterday | 2 authors (You and others)

```php
<?php
    session_start();
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
    require("../../controllers/adminController.php");
```

updateMedicine.php

home.php

views > home.php > html > body > header

You, 3 days ago | 3 authors (You and others)

```php
<?php
    session_start();
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
    require '../controllers/listMedicineController.php';
```

home.php

login.php

views > login.php > html > body > header

```php
<?php
    session_start();
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
    unset($_SESSION['login_message']);
```

login.php

medicine.php

views > medicine.php > ...

You, 3 days ago | 2 authors (You and others)

```php
<?php
    session_start();
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
    require '../controllers/adminController.php';
```

medicine.php

register.php

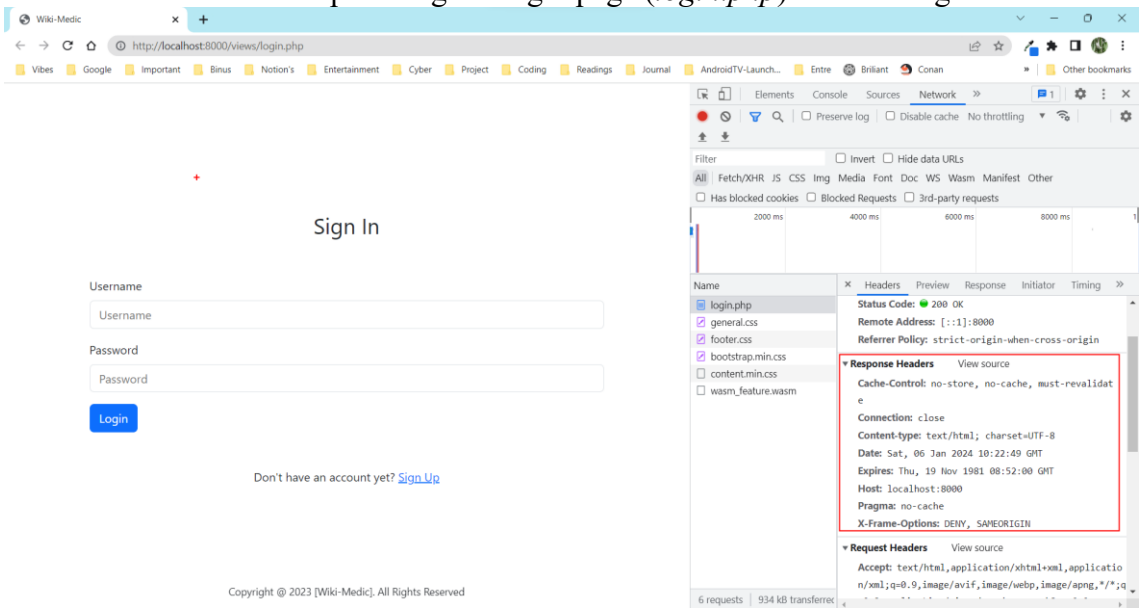
views > register.php > html > head > link

You, yesterday | 1 author (You)

```php
<?php
    session_start();
    header_remove("X-Powered-By");
    header('X-Frame-Options: DENY, SAMEORIGIN');
```

register.php

As the result, it won't display the PHP X-Powered-By version while opened using PHP Development Server. Below is an example using the login page (*login.php*). The same goes for all other pages.

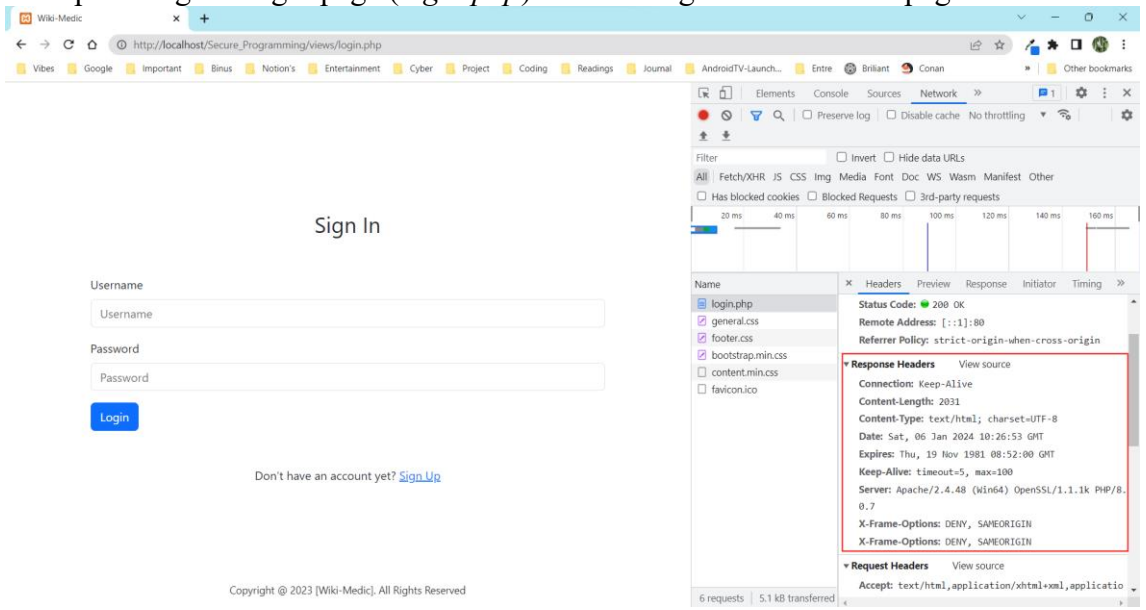


2. Unset all the PHP and Server version in the .htaccess file. For additional, we also unset the Pragma, Last-Modified and Cache-Control.

.htaccess

```
<IfModule mod_php.c>
    Header unset X-Powered-By
</IfModule>

<IfModule mod_headers.c>
    Header unset Server
</IfModule>

Header unset Pragma
Header unset Last-Modified
Header unset Cache-Control
Header always append X-Frame-Options DENY
Header always append X-Frame-Options SAMEORIGIN
```

As the result, it won't display the PHP X-Powered-By version, Server, Pragma, Last-Modified, and Cache-Control while opened apache web server (In this case, apache from xampp). Below is an example using the login page (*login.php*). The same goes for all other pages.



7) Username Validation

Prevented Vulnerability	*Unwanted payload stored in database*
Executive Summary	We've implemented length validation for username input when user register on our website, where the user only limited to make username within range 5 to 12 characters. Hence, it is impossible to register new account where the username is malicious payload (SQL injection payload, path traversal payload, RCE payload, etc.)
POC