Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student: Ian Cannon

Email: mailto:cannoni1@udayton.edu

Short-bio: Ian Cannon interests in Reinforcement Learning for Autonomous Control.

Ian's headshot

Project Overview

This project involved developing a secure, full-stack web application using PHP and MySQL. The primary goal was to implement a complete user authentication system, including user registration, login, profile management, and password updates, while adhering to modern web security principles. Key learning outcomes included implementing defense mechanisms against common vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), Session Hijacking, and Cross-Site Request Forgery (CSRF), as well as correctly configuring a web server for secure communication over HTTPS.

Repository Information

Respository's URL: https://github.com/Spiph/waph-spiph/tree/main/projects/project2

This is a public repository for Ian Cannon to store all code from the course.

Functional Requirements and Implementation

1. User Registration

Requirement: Allow new users to create an account by providing a username, password, name, and email.

Implementation: The user registration process begins with the registrationform.php page, which presents a form for new users.

This form includes client-side validation using HTML5 required and pattern attributes to ensure that data is entered in the correct format before submission. For example, the password field requires a mix of uppercase letters, lowercase letters, numbers, and a minimum length of 8 characters.

Upon submission, the form data is sent to register.php. This script performs crucial server-side validation:

It first checks for a valid anti-CSRF token to prevent forgery attacks.

It sanitizes all inputs using htmlspecialchars() and trim() to prevent XSS.

It validates the data against the same rules as the front-end (e.g., email format, password complexity).

If validation passes, it securely hashes the user's password using password_hash().

Finally, it uses a prepared SQL statement to insert the new user's data into the users table, preventing SQL injection.

2. User Login and Profile Viewing

Requirement: Implement a secure login system that authenticates users and allows them to view their profile.

Implementation: The login process starts at form.php. Users enter their credentials, which are submitted to index.php.

The index.php script handles the login attempt by calling the checklogin_with_password_hash() function. This function securely retrieves the user's hashed password from the database and uses password_verify() to check if the submitted password is correct.

If authentication is successful, a session is established, and the user is granted access to their profile page. The profile page (index.php) then retrieves the logged-in user's name and email from the database and displays it.

3. Profile Management

Requirement: Enable authenticated users to edit their profile information (name and email).

Implementation: From their profile page, a logged-in user can click a link to editprofileform.php. This form allows them to update their name and email. Like all other state-changing forms, it includes an anti-CSRF token.

The form submits to updateprofile.php, which validates the CSRF token and the inputs. It then uses a prepared UPDATE statement to securely modify the user's data in the database, identifying the user by the username stored in the session.

4. Password Update

Requirement: Allow users to change their passwords securely.

Implementation: The password update process is similar to profile management. The user navigates to changepasswordform.php and submits a new password. The form is protected by an anti-CSRF token and client-side validation.

The updatepassword.php script receives the new password, validates the CSRF token, hashes the new password, and uses a prepared UPDATE statement to save it to the database for the currently logged-in user.

Security and Non-Technical Requirements

1. HTTPS and Secure Server Configuration

Requirement: The application must be deployed over HTTPS.

Implementation: The Apache web server was configured to handle HTTPS traffic. This involved creating a self-signed SSL certificate and a dedicated virtual host configuration file, project2-ssl.conf. This file instructs Apache to listen on port 443, enables the SSL engine, and specifies the paths to the SSL certificate and private key. A block was included to grant Apache the necessary permissions to serve files from the project directory over a secure connection.

2. Secure Session Management

Requirement: Protect against session hijacking and fixation attacks.

Implementation: A centralized session_auth.php file was created and is included at the top of every page that requires authentication. This script implements two key defenses:

Secure Cookie Parameters: It uses session_set_cookie_params() to set the session cookie as HttpOnly and Secure, which prevents it from being accessed by client-side scripts and ensures it is only sent over HTTPS.

User Agent Verification: Upon login, the user's browser HTTP_USER_AGENT string is stored in the session. On every subsequent page load, the script verifies that the current user agent matches the one stored in the session. A mismatch indicates a potential session hijacking attack, causing the session to be destroyed immediately.

3. Input Validation and XSS Prevention

Requirement: Implement comprehensive input validation on both client and server sides.

Implementation: All user inputs are treated as untrusted.

Client-Side: HTML5 pattern attributes are used on forms for immediate user feedback.

Server-Side: In scripts like register.php and updateprofile.php, all incoming data is sanitized with htmlspecialchars() to neutralize any embedded HTML or script tags, preventing XSS attacks. The data is also validated to ensure it meets expected formats (e.g., valid email, password complexity) before being used.

4. SQL Injection Prevention

Requirement: Ensure all SQL operations use prepared statements.

Implementation: Every single database query in the application that involves user-supplied data (INSERT, SELECT, UPDATE) is executed using prepared statements. This is visible in register.php, index.php, updateprofile.php, and updatepassword.php. This practice ensures that user input is never directly concatenated into SQL queries, completely mitigating the risk of SQL injection.

5. CSRF Protection

Requirement: Protect against Cross-Site Request Forgery attacks.

Implementation: All forms that perform state-changing actions (registration, profile updates, password changes) are protected against CSRF.

When a form page like editprofileform.php is loaded, a cryptographically random token is generated and stored in the user's session ($_SESSION['csrf_token']).

This token is also embedded as a hidden input field in the form.

When the form is submitted, the corresponding action script (e.g., updateprofile.php) verifies that the token from the form submission matches the token stored in the session. If they do not match, the request is aborted, preventing the attack.

6. Password Hashing

Requirement: Passwords must be hashed before being stored in the database.

Implementation: The application uses PHP's modern and secure password_hash() function to create a strong, salted hash of a user's password during registration. When a user logs in, the password_verify()

function is used to compare their submitted password against the stored hash. This is a significant security improvement over older methods like MD5.

Video Demo

https://github.com/Spiph/waph-spiph/blob/main/projects/project2/2025-07-27%2016-26-50.mkv

My Code

changepasswordform

database-setup

editprofileform

form

index

logout

register

registrationform

session_auth

sessiontest

updatepassword

updateprofile