Project Portfolio

Tyler D. Wallingford

CST-452 Senior Project II

Grand Canyon University

Instructor: Professor Jeff Stucker

Date: 09/29/24

**Project Overview**

The Secure Messaging Application is designed to provide users with a platform for secure, real-time messaging. The application emphasizes end-to-end encryption using AES-128 encryption to ensure message confidentiality and security. The project integrates security measures such as CSRF token protection, Content Security Policy headers, and validation processes to guard against common web vulnerabilities, including Cross-Site Scripting and CSRF attacks. The platform offers a user-friendly interface with functionalities such as message sending, contact management, theme customization, and robust encryption. This project was implemented using PHP for the backend, MySQL for database management, and is deployed on Microsoft Azure.

Design Diagrams:

**Wireframe Diagram:**

A screenshot of a computer screen

Description automatically generated

Flow Chart Diagram:

A diagram of a computer program

Description automatically generated

**Code Snippets and Explanations:**

1. **AES Encryption and decryption:**

A screen shot of a computer program

Description automatically generated

* **Encryption Process**:The encryptMessage function uses AES-128-CBC encryption with a defined key and initialization vector. It encrypts the provided plaintext message and converts the encrypted data into a hexadecimal format. If encryption fails, an error message is logged.
* **Decryption Process**: The decryptMessage function first converts the hexadecimal encrypted message back to binary. It then decrypts the binary message using AES-128-CBC with the same key and IV. If decryption fails, an error message is logged, and null is returned.

1. **CSRF Token Generation:**

A screen shot of a computer program

Description automatically generated

* **Session Initialization:** The code checks if a session is already active. If not, it starts a session using session\_start(). This ensures that each user has a unique session for tracking purposes.
* **CSRF Token Generation:** If there isn't an existing CSRF token in the session, a new token is generated using bin2hex(random\_bytes(32)). This creates a 32-byte random string, ensuring a high level of security and uniqueness for each user session.
* **CSRF Token Functions**: The generateCsrfToken function retrieves the current CSRF token from the session for use in forms. The verifyCsrfToken function compares the provided token with the session's token using hash\_equals. This function is crucial for preventing CSRF attacks by ensuring that incoming form submissions are genuine and originate from the authenticated user’s session.

1. **User Registration Password Hashing:**



* **Password Hashing**: The code uses password\_hash($password, PASSWORD\_DEFAULT) to hash the user's password during registration. PASSWORD\_DEFAULT is a built-in constant that ensures the use of a strong, up-to-date hashing algorithm (currently bcrypt). It also automatically handles the creation of a salt, which further strengthens the hash.
* **Security Benefits**: By hashing the password, the application stores an irreversible representation of the password in the database. Even if the database were compromised, attackers wouldn't have access to the original password. This hashing approach ensures that passwords are protected against common attacks like rainbow table and brute-force attacks.

1. **Content Security Policy Implementation:**



* **CSP Header Implementation:** This line sets a Content Security Policy header to help mitigate Cross-Site Scripting and data injection attacks.
* **Security Benefits:** This CSP header provides a security layer that helps prevent XSS attacks by restricting the sources from which content can be loaded.It ensures that only trusted content from the application's domain is executed, reducing the risk of malicious script injections.

1. **Message Sending Function:**

A computer screen shot of a program code

Description automatically generated

**Code Functionality:**

* The sendMessage() function handles the process of sending a message from one user to another in the messaging application.

**Key Steps:**

* Fetching the Message and CSRF Token:var messageText = document.getElementById('message').value;: Retrieves the message typed by the user. var csrfToken = document.querySelector('input[name="csrf\_token"]').value;: Fetches the CSRF token from the hidden input field to protect against CSRF attacks.
* Form Data Preparation: A FormData object is created, and necessary data (action, receiver\_id, message, csrf\_token) are appended to it.
* Sending the AJAX Request: fetch('index.php', { method: 'POST', body: formData }): Sends a POST request to index.php with the form data using the Fetch API.
* Handling the Response:If the message is sent successfully (data.success), the input field is cleared, and the conversation is refreshed (fetchMessages(currentConversationId)). If an error occurs, an alert displays the error message.

**Security Aspects:**

* CSRF Protection: Including the CSRF token (csrf\_token) ensures that the request is verified, preventing unauthorized users from performing actions on behalf of authenticated users.
* Client-Side Validation: Checks whether the messageText is not empty and whether currentReceiverId is selected, ensuring that valid data is sent to the server.

**What Significance does this function give?**

* This function is crucial for enabling secure message exchange between users while incorporating essential security checks like CSRF token verification.

**GitHub Portfolio Link:**

https://github.com/EqWallyT/Secure-Messaging-App-public

**Video Presentations**

Part 1: Code Walkthrough: https://www.loom.com/share/7b81ec2b97b5403c8b0c28126ce29fd3?sid=a6b6fce9-5614-480f-807c-efb91704f794

Part 2: Presentation:

https://www.loom.com/share/d99fe2a49ec4490c81d156c5e3da10b5?sid=62d06b75-e425-4bba-beda-d949034dc6d3