#### **Step 1: Security Logging and Monitoring Failures**

#### **Objective:**

1. Log critical activities (e.g., logins, failed attempts, file uploads).
2. Securely store logs and prevent unauthorized access.

**Schema Overview**:

* **Artist Table**: Holds artist information, including username, ID, and contact details.
* **Art\_submission Table**: Contains art details, including the artist's ID and contact information.
* **Exhibition Table**: Tracks event information like art name, username, and location.
* **Gallery Table**: Stores gallery-related data, including art name and pricing.

**Password Security**:

* The password column in the Artist table is of type VARCHAR(100), which indicates plaintext storage. This is a critical vulnerability. Passwords should be hashed (e.g., using password\_hash() in PHP) before storage.

**Foreign Key Relationships**:

* The Art\_submission table references the Artist table by username.
* The Artist table references the Art\_submission table by artist\_id. However, this dependency could create cyclic or redundant relationships.

**Data Types**:

* The use of VARCHAR and INT seems appropriate for most fields, but the size of VARCHAR(20) for the art column in Art\_submission and contact fields may need a review based on the expected data.

**Indexes**:

* Primary keys are defined for all tables, which is good. However, you might consider additional indexes for columns frequently used in lookups or joins, such as username in the Art\_submission table.

 **Normalization**:

* While the database is relatively normalized, there may be opportunities to simplify relationships to reduce redundancy.

Loginphp

**Key Improvements**

1. **Secure Password Handling**:
   * Uses password\_verify() for password verification.
   * Assumes that the passwords are hashed during registration with password\_hash().
2. **Input Validation**:
   * Trims and checks inputs to ensure they are not empty before proceeding.
3. **Prepared Statements**:
   * Prevents SQL injection by using prepared statements ($conn->prepare and bind\_param).
4. **Session Security**:
   * Uses session\_regenerate\_id(true) to mitigate session fixation attacks.
   * Stores only the necessary session information (e.g., username).
5. **Logging**:
   * Logs successful and failed login attempts with details like IP address and reason.
6. **Error Messages**:
   * Provides generic error messages to avoid leaking information about valid usernames.
7. **UTF-8 Compliance**:
   * Assumes the database and application use UTF-8 encoding for better compatibility.

Signup.php

**Key Enhancements**

1. **Password Hashing**:
   * User passwords are securely hashed using password\_hash() with the default algorithm (currently bcrypt).
2. **Input Validation**:
   * Sanitized and validated username, email, and password.
   * Checked for email format and ensured passwords match.
3. **Unique Username and Email**:
   * Checked the database to prevent duplicate usernames or emails.
4. **Event Logging**:
   * Logged successful registration events, including the user's IP address.
5. **Session Handling**:
   * Automatically logs in the user upon successful registration and redirects them.
6. **Secure SQL Queries**:
   * Used prepared statements to prevent SQL injection.

2. Identification and Authentication Failures (A07)

**Objective:**

1. Strengthen authentication mechanisms.
2. Prevent brute force attacks.
3. Enhance session security

Against brute force

// Constants for brute force mitigation

define('MAX\_ATTEMPTS', 5);

define('LOCKOUT\_TIME', 15 \* 60); // 15 minutes

### ****What's Been Added****

1. **Track Failed Attempts:**
   * After every failed login attempt, failed\_attempts in the database is incremented.
   * When the number of attempts exceeds MAX\_ATTEMPTS, the current time is stored in lockout\_time.
2. **Check Lockout:**
   * Before verifying the password, the code checks if the account is locked by comparing failed\_attempts and lockout\_time.
3. **Reset After Successful Login:**
   * Upon successful login, failed\_attempts is reset to 0, and lockout\_time is cleared.
4. **Log Events:**
   * Logs all critical actions such as failed logins, locked account access, and successful logins

### ****Web Application Security Documentation****

This documentation covers the steps taken to secure the web application against **Security Logging and Monitoring Failures** and **Identification and Authentication Failures** (A07) as per the OWASP Top Ten.

## **1. Security Logging and Monitoring Failures**

### ****Objective****

The goal is to ensure that all significant security events, such as login attempts (both successful and failed), are logged in a secure, accessible, and tamper-proof manner to allow for proper monitoring and alerting in the event of suspicious activity.

### ****Implementation****

* **Secure Logging Function**: A custom log\_event() function was created to log significant events like successful and failed login attempts, account lockouts, and system errors.
* **Log File Location**: The logs are stored in the /app\_logs/ directory. The log file used is app.log, which is located in the same directory as the PHP script (\_\_DIR\_\_).
  + If the directory does not exist, it is dynamically created with secure permissions (0750), ensuring that only the owner and the application have access.
* **Log Entries**: Each log entry includes a timestamp (e.g., 2024-11-28 14:30:00), a message describing the event, and the IP address of the user triggering the event. The format is:

csharp

Copy code

[Y-m-d H:i:s] Event message IP=REMOTE\_ADDR

* **Log Events**:
  + **Successful Login**: Logged when a user successfully logs in, with their username and IP address.
  + **Failed Login**: Logged for incorrect passwords or unknown usernames. It also includes the IP address for traceability.
  + **Account Lockout**: Logged when a user tries to log in after their account has been locked due to too many failed attempts.

### ****Security Considerations****

* **Access Control for Logs**: The /app\_logs/ directory has restricted permissions (0750) to prevent unauthorized access to the logs.
* **Log Integrity**: The application writes logs using file\_put\_contents() with the FILE\_APPEND flag to ensure new entries are added to the end of the log file without overwriting existing data.
* **Log Rotation and Retention**: For scalability and security, a log rotation strategy (e.g., using logrotate) should be considered for handling large log files.

### ****Further Enhancements**** (Optional):

* **Centralized Logging**: Consider implementing centralized logging (e.g., using services like ELK Stack, Splunk, or a cloud-based solution) to manage logs more effectively and integrate them with monitoring/alerting systems.

## **2. Identification and Authentication Failures (A07)**

### ****Objective****

To mitigate **Identification and Authentication Failures** (A07) by implementing secure and resilient mechanisms for user authentication and session management, reducing the risk of unauthorized access.

### ****Implementation****

* **Input Validation**: Ensured that both username and password inputs are validated for presence before processing the login attempt. If either is empty, the user is notified with a generic error message.
* **SQL Injection Prevention**: Used prepared statements ($stmt->bind\_param()) to prevent SQL injection attacks by safely embedding user input into the SQL query.
* **Password Verification**: Instead of storing passwords as plain text, the application uses password\_verify() to compare the entered password with the hashed password stored in the database. This ensures that even if the database is compromised, attackers cannot easily retrieve user passwords.
* **Brute Force Mitigation**:
  + Implemented a mechanism to **limit login attempts** to prevent brute-force attacks. After a predefined number of failed login attempts (MAX\_ATTEMPTS), the account is temporarily locked for a specified period (LOCKOUT\_TIME).
  + **Lockout Time**: Accounts are locked for 15 minutes after 5 failed login attempts.
  + The system checks for locked accounts before allowing further login attempts. If an account is locked, the user is informed that they should try again later.
  + Failed login attempts and lockout information are tracked in the database in the failed\_attempts and lockout\_time fields for each user.
* **Session Fixation Protection**: When a user successfully logs in, the session ID is regenerated using session\_regenerate\_id(true) to prevent session fixation attacks.
* **Secure Error Messages**: To prevent information leakage that could aid an attacker in identifying valid usernames or accounts, the application provides a **generic error message** ("Invalid username or password.") for all failed login attempts. The application does not disclose whether the username or password is incorrect.

### ****Database Changes****

The following changes were made to the Artist table to support brute-force mitigation:

* **failed\_attempts** (INT): Tracks the number of consecutive failed login attempts.
* **lockout\_time** (INT, nullable): Stores the timestamp when an account is locked, allowing the application to enforce the lockout period.

sql

Copy code

ALTER TABLE Artist ADD failed\_attempts INT DEFAULT 0;

ALTER TABLE Artist ADD lockout\_time INT NULL;

### ****Session Management****

* Sessions are managed using PHP’s native $\_SESSION functionality, and session data is stored securely on the server.
* A new session ID is generated on login to prevent session fixation.
* **Session Security**:
  + Cookies should be set with HttpOnly and Secure flags for better session security.
  + Implementing stricter session handling policies (e.g., limiting session timeouts) could be considered.

### ****Further Enhancements**** (Optional):

* **Rate Limiting**: For more robust brute force protection, implement IP-based rate limiting (e.g., using .htaccess rules or an external service like Fail2Ban) to block multiple login attempts from a single IP address in a short time.
* **Multi-Factor Authentication (MFA)**: Implementing an additional layer of authentication (e.g., TOTP or SMS) could further strengthen authentication security.

### ****Summary****

* **Security Logging and Monitoring**: Ensured all critical security events (logins, failed logins, account lockouts) are securely logged with proper access controls.
* **Identification and Authentication Failures**: Secured the login system with input validation, prepared statements, password hashing, brute-force mitigation, and session fixation protection, reducing the risk of unauthorized access.

Sure! Here's a **documentation** for the **first security measure** — **Security Logging and Monitoring Failures (A09)** — based on the work you've already done, using the files you've provided.

## **1. Security Measure Documentation: Security Logging and Monitoring Failures (A09)**

### ****Objective****:

The goal of this security measure is to ensure that all security-related events (such as failed login attempts, successful logins, and account lockouts) are logged and monitored. This allows for early detection of suspicious activity, such as brute force attacks or credential stuffing, and provides the means for investigating potential breaches.

### ****Approach Implemented****:

1. **Logging Mechanism**:  
   The system logs authentication-related events into a secure log file (app\_logs/app.log). This includes:
   * Successful login attempts.
   * Failed login attempts.
   * Account lockouts (due to too many failed login attempts).
   * Attempts to access restricted areas or perform unauthorized actions.

The logs are written with the following details:

* + Timestamp of the event.
  + Type of event (e.g., login success, failed login attempt).
  + The username involved.
  + The IP address of the user attempting the action.

1. **Log File Path and Permissions**:
   * The log file is located in the app\_logs/ directory, with the file named app.log.
   * The permissions for the log file ensure that only the web server has read and write access to it.
   * Access to the log file through the web is denied using .htaccess rules.
2. **Code Implementation**:
   * **Logging Function**: A custom logging function (log\_event()) is used to write events to the log file. The function ensures that logs are timestamped and written to the file in a structured format.

php

Copy code

// Logging function to log events

function log\_event($message) {

$log\_file = \_\_DIR\_\_ . '/app\_logs/app.log'; // Define log path

if (!file\_exists(dirname($log\_file))) {

mkdir(dirname($log\_file), 0777, true); // Ensure directory exists

}

$log\_message = date('[Y-m-d H:i:s]') . " " . $message . PHP\_EOL;

file\_put\_contents($log\_file, $log\_message, FILE\_APPEND); // Append to log file

}

* + - **Event Logging**: The log\_event() function is called when a user logs in, fails to log in, or when an account lockout occurs.

Example of **successful login** logging:

php

Copy code

log\_event("Successful login: Username={$username}, IP={$\_SERVER['REMOTE\_ADDR']}");

Example of **failed login attempt** logging:

php

Copy code

log\_event("Failed login attempt: Username={$username}, IP={$\_SERVER['REMOTE\_ADDR']}");

**Failed login attempts** are tracked and logged every time a login attempt fails. After a set number of failed attempts, the system locks the account for a specific duration to prevent brute-force attacks.

1. **Log Format**: The log entries are structured as follows:

ruby

Copy code

[YYYY-MM-DD HH:MM:SS] Event Message

For example:

yaml

Copy code

[2024-12-09 14:30:01] Failed login attempt: Username=artist1, IP=192.168.0.1

[2024-12-09 14:35:12] Successful login: Username=artist1, IP=192.168.0.1

[2024-12-09 14:45:23] Account locked due to too many failed login attempts: Username=artist1, IP=192.168.0.1

1. **Account Lockout Monitoring**:
   * After a specified number of failed login attempts (e.g., 5), the user account is locked for a defined duration (e.g., 15 minutes). During this lockout period, further login attempts are blocked, and the event is logged.
   * Once the lockout period expires, the failed attempt count is reset, and the user can attempt to log in again.
2. **Access Restrictions**:
   * The log files are protected against unauthorized access by setting file permissions on the log directory and by using .htaccess to block web access.
   * The log file has the permission set to 600 (read/write for the owner, no permissions for others) to prevent unauthorized access.
3. **Log Storage**:
   * Logs are stored in plain text files for easy access and auditing.
   * The log directory and file are backed up periodically to prevent loss of critical data.

### ****Security Considerations****:

1. **Avoid Logging Sensitive Information**:  
   The logging system has been implemented to avoid logging sensitive information such as passwords or credit card numbers. Only relevant details such as username and IP address are logged.
2. **Log File Integrity**:  
   Logs must be monitored for tampering or unauthorized deletion. Consider implementing measures such as log rotation and more advanced logging systems (e.g., centralized logging or SIEM) for larger-scale applications.
3. **Log File Monitoring**:  
   Logs should be regularly reviewed to detect suspicious activity, such as repeated failed login attempts, which may indicate a brute force attack.

### ****Future Considerations****:

1. **Centralized Logging**:  
   Consider integrating a centralized logging solution like **ELK Stack (Elasticsearch, Logstash, Kibana)** or **Splunk** for better visibility and analysis of log data, especially for large applications.
2. **Log Rotation**:  
   Implement log rotation to ensure that log files do not grow too large and affect server performance. Tools like **logrotate** can help automate this.
3. **Alerting on Suspicious Activity**:  
   Set up automated alerting based on log entries. For example, alerting the system administrator when there are a high number of failed login attempts within a short period.

This documentation describes the **implementation of the first security measure** focused on **Security Logging and Monitoring Failures (A09)** in your login system. It covers the logging mechanism used, the protection of log files, and the process for monitoring failed login attempts, including account lockout functionality