Vulnerability Analysis Report

File Analyzed: upload_1742834926.php

Total Vulnerabilities: 32

Vulnerability Summary:

SQL Injection: 11

Command Injection: 1

Shell Injection Vulnerability: 2

Cross-Site Scripting (XSS): 7

File Upload Vulnerability: 8

Code Injection: 3

Detailed Vulnerabilities:

Type: Code Injection
Pattern: eval($_GET["cmd"])
Line: 212

Type: Code Injection
Pattern: eval($_GET["cmd"])
Line: 212

Type: Code Injection
Pattern: eval($_GET["cmd"])
Line: 212

Type: SQL Injection
Pattern: SELECT * FROM
Line: 77

Type: SQL Injection
Pattern: DELETE FROM
Line: 87

Type: SQL Injection
Pattern: OR '1'='1"
Line: 73

Type: SQL Injection
Pattern: $maliciousUsername = "' OR '1'='1"
Line: 73

Type: SQL Injection
Pattern: $maliciousInput = "'; alert('xss'); '"
Line: 100

```
Type: SQL Injection
Pattern: $_POST['username']
Line: 199

Type: SQL Injection
Pattern: $_POST['password']
Line: 200

Type: SQL Injection
Pattern: $_POST['username']
Line: 202

Type: SQL Injection
Pattern: $_POST['password']
Line: 202

Type: SQL Injection
Pattern: $_POST['password']
Line: 203

Type: SQL Injection
Pattern: $_GET["cmd"]
Line: 212

Type: Cross-Site Scripting (XSS)
Pattern: <script>alert("xss")</script>
Line: 92

Type: Cross-Site Scripting (XSS)
Pattern: $_POST['username']
Line: 199

Type: Cross-Site Scripting (XSS)
Pattern: $_POST['password']
Line: 200

Type: Cross-Site Scripting (XSS)
Pattern: $_POST['username']
Line: 202

Type: Cross-Site Scripting (XSS)
Pattern: $_POST['password']
Line: 202

Type: Cross-Site Scripting (XSS)
Pattern: $_POST['password']
Line: 203
```

Type: Cross-Site Scripting (XSS)
Pattern: $_GET["cmd"]
Line: 212

Type: Command Injection
Pattern: eval($_GET["cmd"])
Line: 212
Mitigation: Validate and sanitize user inputs. Use allowlists instead of executing raw input commands.

Type: Shell Injection Vulnerability
Pattern: eval($_GET["cmd"])
Line: 212

Type: Shell Injection Vulnerability
Pattern: eval($_GET["cmd"])
Line: 212

Type: File Upload Vulnerability
Pattern: ../
Line: 2

Type: File Upload Vulnerability
Pattern: ../
Line: 3

Type: File Upload Vulnerability
Pattern: ../
Line: 145

Type: File Upload Vulnerability
Pattern: ../
Line: 153

Type: File Upload Vulnerability
Pattern: '/../vendor/autoload.php'
Line: 2

Type: File Upload Vulnerability
Pattern: '/../uploads/secure_version.php'
Line: 3

Type: File Upload Vulnerability
Pattern: '../config.php'
Line: 145

Type: File Upload Vulnerability
Pattern: '/../includes/header.php'
Line: 153

Mitigations:

Type: SQL Injection

SQL Injection mitigations below).**

* **Sanitize and validate all user input.** This is crucial for preventin
* **Implement the principle of least privilege.** Run your web server wit
* **Use a Web Application Firewall (WAF).** A WAF can detect and block ma
* **Regularly scan your code for vulnerabilities.**
* **Implement Content Security Policy (CSP).** CSP can help mitigate some

Vulnerability Type: SQL Injection

* **Description:** SQL Injection occurs when user-supplied data is inserted i

* **Mitigation Strategy:** **Use parameterized queries (also known as prepare

* **Corrected Code Example (using PDO):**

  Vulnerable Code (example from $_POST['username']):

```php
$username = $_POST['username'];
$password = $_POST['password'];
$query = "SELECT * FROM users WHERE username = '$username' AND password = '$
// Execute query (vulnerable)
```

  Corrected Code (using PDO and parameterized queries):

```php
<?php
$db_host = 'localhost';
$db_name = 'your_database_name';
$db_user = 'your_database_user';
$db_pass = 'your_database_password';

try {
    $pdo = new PDO("mysql:host=$db_host;dbname=$db_name", $db_user, $db_pass
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); // Enable

    $username = $_POST['username'];
    $password = $_POST['password'];

    $stmt = $pdo->prepare("SELECT * FROM users WHERE username = :username AN
    $stmt->bindParam(':username', $username);
    $stmt->bindParam(':password', $password);
    $stmt->execute();

    $results = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

```
        if ($results) {
            // User found
            print_r($results); //Do something with the results
        } else {
            // User not found
            echo "Invalid username or password.";
        }

    } catch (PDOException $e) {
        echo "Database Error: " . $e->getMessage();
    }
    ?>
```

*   **Best Practices:**

    *   **Always use parameterized queries (prepared statements).** This is the
    *   **Use an ORM (Object-Relational Mapper).** ORMs often provide built-in
    *   **Principle of Least Privilege:** Grant database users only the necessar
    *   **Input Validation:** Validate and sanitize user input, but **do not rel
    *   **Escape special characters.** If you absolutely cannot use parameterize
    *   **Regularly audit your database queries.**
    *   **Use a Web Application Firewall (WAF).**

**Vulnerability Type:
Type: Command Injection
Command Injection and Shell Injection)**

*   **Description:** The `eval($_GET["cmd"])` construct allows arbitrary code ex
Type: Shell Injection Vulnerability
Mitigation for Shell Injection Vulnerability not provided by the API.
Type: Cross-Site Scripting (XSS)
Cross-Site Scripting (XSS)**

*   **Description:** XSS vulnerabilities allow attackers to inject malicious scr

*   **Mitigation Strategy:** **Escape all user-supplied data before displaying

*   **Corrected Code Example:**

    Vulnerable Code:

    ```php
    echo "Welcome, " . $_POST['username'];
    ```

Corrected Code:

```php
echo "Welcome, " . htmlspecialchars($_POST['username'], ENT_QUOTES, 'UTF-8')
```

Explanation:

*   `htmlspecialchars()` converts special characters (like `<`, `>`, `&`, `"
*   `ENT_QUOTES` handles both single and double quotes.
*   `UTF-8` specifies the character encoding.

*   **Best Practices:**

    *   **Context-Aware Output Encoding:** Use the correct escaping function for
        *   `htmlspecialchars()` for HTML output.
        *   `urlencode()` for URLs.
        *   `json_encode()` for JavaScript.
        *   `CSS` escaping functions for CSS.
    *   **Input Validation:** Validate input to ensure it conforms to expected f
    *   **Content Security Policy (CSP):** Use CSP to control the sources from
    *   **Escape early, escape often:** Escape data as close to the output as po
    *   **Use a templating engine with auto-escaping features.**
    *   **Sanitize HTML input (if necessary) with a library like HTML Purifier.*

**Vulnerability Type:
Type: File Upload Vulnerability
File Upload Vulnerability**

*   **Description:** File upload vulnerabilities occur when the application all

*   **Mitigation Strategy:** Implement a multi-layered approach to file upload

*   **Corrected Code Example (Illustrative):**

    Vulnerable Code (example):

```php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file); //Vul
```

Corrected Code (example):

```php
<?php
```

```
// Set allowed file types
$allowed_types = array("jpg", "jpeg", "png", "gif");

// Set upload directory
$upload_dir = "uploads/";

// Generate a unique filename
$filename = uniqid() . "_" . time();

// Get file extension
$file_ext = strtolower(pathinfo($_FILES["fileToUpload"]["name"], PATHINFO_EX

// Check if file type is allowed
if (!in_array($file_ext, $allowed_types)) {
    echo "Error: Only JPG, JPEG, PNG & GIF files are allowed.";
    exit();
}

// Check file size (example: 2MB limit)
if ($_FILES["fileToUpload"]["size"] > 2000000) {
    echo "Error: File size must be less than 2MB.";
    exit();
}

// Check for errors during upload
if ($_FILES["fileToUpload"]["error"] !== UPLOAD_ERR_OK) {
    echo "Error: File upload failed with error code: " . $_FILES["fileToUplo
    exit();
}

// Generate full filename with extension
$full_filename = $filename . "." . $file_ext;

// Set full upload path
$upload_path = $upload_dir . $full_filename;

// Move uploaded file
if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $upload_path)) {
    echo "File uploaded successfully.";
} else {
    echo "Error: Failed to move uploaded file.";
}
?>
```

*   **Best Practices:**

1. **Whitelist Allowed File Types:** Only allow specific file types (e.g.,
2. **Validate File Extension:** Check the file extension against a whiteli
3. **Content-Type Validation:** While not sufficient alone, verify the Cont
4. **File Size Limits:** Set maximum file size limits to prevent denial-of
5. **Unique Filenames:** Generate unique filenames for uploaded files to p
6. **Secure Upload Directory:** Store uploaded files outside of the webroo
7. **Directory Traversal Prevention:** Sanitize filenames to prevent direc
8. **File Content Scanning:** Scan uploaded files for malware and other ma
9. **Permissions:** Set appropriate file permissions on the uploaded files
10. **Input Validation:** Validate the file name and size on the client-side
11. **Store Metadata:** Store metadata about the uploaded file (e.g., origin
12. **Regular Security Audits:** Regularly audit your file upload functional
13. **Consider a CDN:** For serving uploaded files, especially images and ot

These mitigations provide a strong foundation for improving the security of your
Type: Code Injection
Code Injection (including