Secure Code Review & Vulnerability Assessment Report

Prepared By: Youssef Hamada Alaraky

1.Executive Summary

This report outlines the results of a Manual Secure Code Review conducted on the provided PHP source code archive.

The objective of this assessment was to identify security vulnerabilities within the application's core modules, specifically focusing on User Management, File Uploads, and Diagnostic Tools.

During the review, multiple critical and high-severity vulnerabilities were discovered. These include OS Command Injection, SQL Injection, Insecure File Upload, and Local File Inclusion (LFI).

If exploited, these vulnerabilities could allow an attacker to bypass authentication mechanisms, read sensitive server files, or achieve full Remote Code Execution on the host server.

Immediate remediation of the findings detailed in this report is strongly advised to secure the application environment.

2.Findings Summary

Finding	Severity	Location
SQL Injection (Authentication)	Critical	Login module, query construction
OS Command Injection	Critical	Ping/Diagnostics utility
Local File Inclusion / Path Traversal	Critical	Log export module
Insecure File Upload (extension blacklist bypass)	High	Avatar upload module
Weak password hashing (unsalted MD5)	Medium	Login module
Hard-coded database credentials	Low	Database connection setup
Reflected Cross-Site Scripting (XSS)	Medium	Login module

3.Detailed Findings for Vulnerabilities in order

Vulnerability 1:

- Vulnerability Name: SQL Injection
- CVSS Score: 9.8 (Critical)
- **Location:** User Authentication Module (Line 14)

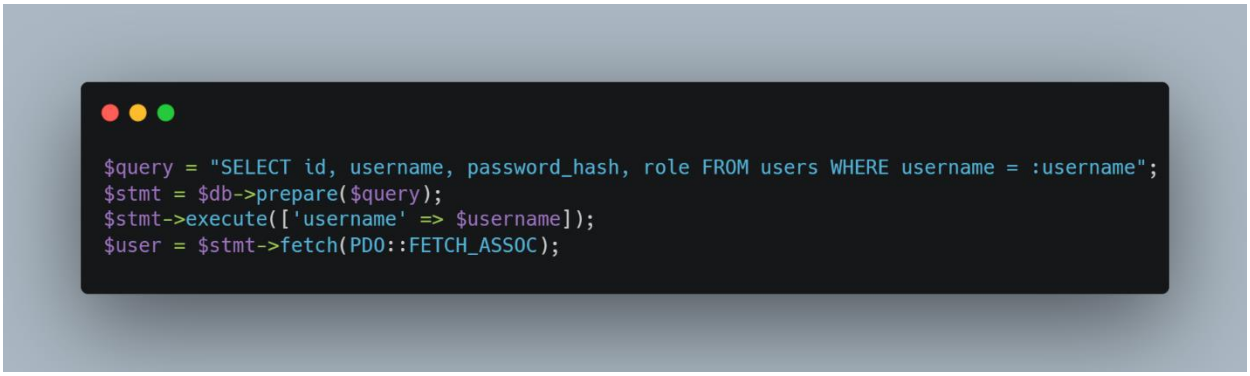
Description: The application constructs SQL queries by directly concatenating user input ($username) without proper sanitization or the use of Prepared Statements. This allows an attacker to manipulate the query logic by injecting malicious SQL payloads (e.g., ' OR 1=1 --).

Evidence:

```
$query = "SELECT id, username, password_hash, role FROM users WHERE username = '{$username}'";
$stmt = $db->query($query);
$user = $stmt->fetch(PDO::FETCH_ASSOC);
```

Impact: Authentication bypass, allowing unauthorized access to any account, including administrators. It can also lead to severe data exfiltration.

Remediation: Implement Prepared Statements (Parameterized Queries) using PDO to ensure the database treats the input strictly as data, neutralizing any injected code.



Vulnerability 2:

- Vulnerability Name: OS Command Injection
- CVSS Score: 9.1 (Critical)
- **Location:** Ping/Diagnostics Utility (Line 62)

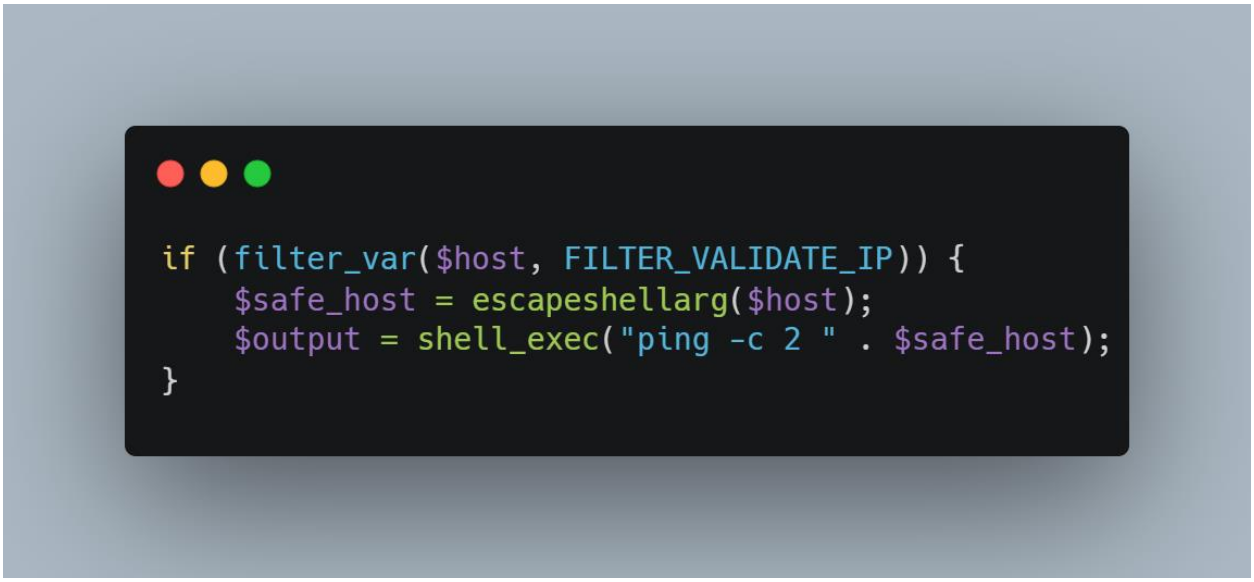
Description: The application takes user input from the $_POST['ping_host'] parameter and passes it directly to the shell_exec() function. An attacker can append arbitrary OS commands using separators like ;, &&, or |.

Evidence:

```
$output = shell_exec("ping -c 2 " . $host);
echo "<pre>" . $output . "</pre>";
```

Impact: Complete compromise of the host operating system, leading to Remote Code Execution.

Remediation: If ping functionality is strictly required, accept only valid IP addresses (e.g., using filter_var). Avoid shell execution entirely if possible; if a shell command is unavoidable, use escapeshellarg() as an additional defense layer.



```
if (filter_var($host, FILTER_VALIDATE_IP)) {
    $safe_host = escapeshellarg($host);
    $output = shell_exec("ping -c 2 " . $safe_host);
}
```

Vulnerability 3:

- Vulnerability Name: Local File Inclusion / Path Traversal
- CVSS Score: 9.8 (Critical)
- **Location:** User Export / Backup Module (Lines 51-53)

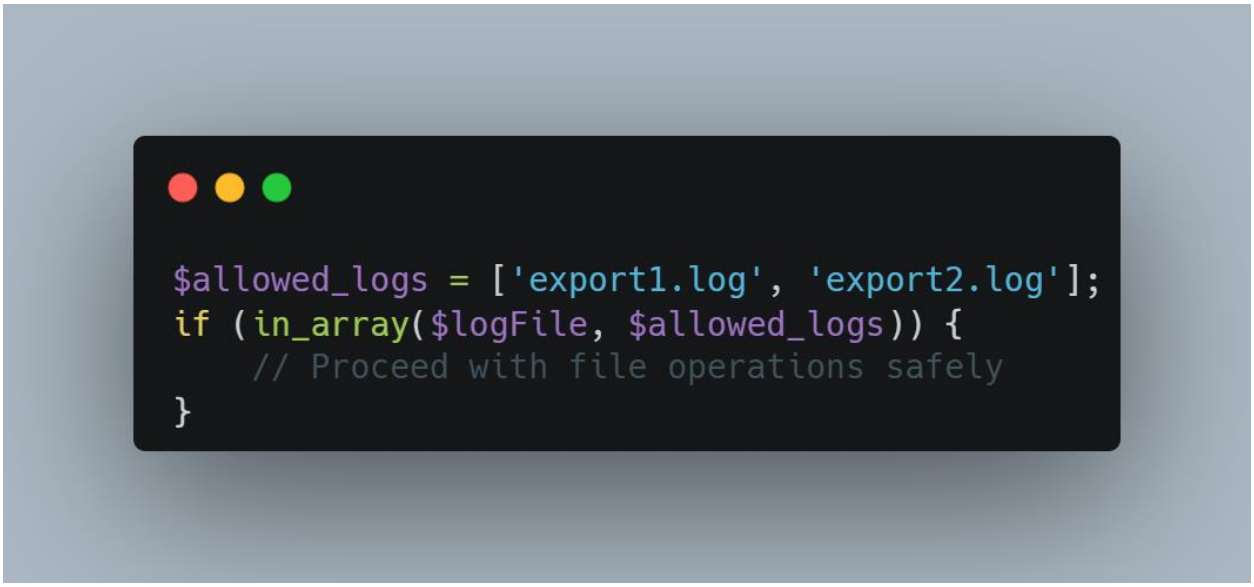
Description: The $_GET['download_log'] parameter is appended to a directory path and executed via the include() function without sanitization against directory traversal sequences (../).

Evidence:

```
51      $filePath = "/var/www/html/logs/" . $logFile;
52      if (file_exists($filePath)) {
53          include($filePath);
```

Impact: Attackers can read sensitive local files on the server (e.g.,/etc/passwd). Furthermore, because the application uses include(), exploitation may also result in direct PHP Code Execution (RCE) if an attacker can control the contents of an included file (e.g., via log poisoning).

Remediation: Avoid passing user input directly into file-inclusion functions. Use a strict whitelist of allowed filenames instead.



```php
$allowed_logs = ['export1.log', 'export2.log'];
if (in_array($logFile, $allowed_logs)) {
    // Proceed with file operations safely
}
```

Vulnerability 4:

- Vulnerability Name: Insecure File Upload
- CVSS Score: 8.8 (High)
- **Location:** Profile Avatar Upload Module (Lines 34-41)

Description: The file upload validation relies on an insecure blacklist approach (blocking .php, .exe, .asp). The blacklist can potentially be bypassed. Depending on the server configuration, alternative PHP-related extensions such as .phtml or .phar may be treated as executable scripts.

Evidence:

```
$fileExt = strtolower(pathinfo($fileName, PATHINFO_EXTENSION));
$forbiddenExts = array("php", "exe", "asp");
```

Impact: Uploading a malicious Web Shell leads to Remote Code Execution and full server takeover.

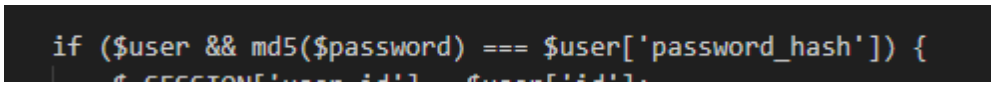
Remediation: Implement a strict whitelist approach for file extensions (e.g., allowing only .jpg, .png). Check the actual file type, rename uploaded files to random names, and block any code from running in the uploads folder.

Vulnerability 5:

- Vulnerability Name: Weak Password Hashing
- CVSS Score: 5.9 (Medium)
- **Location:** User Authentication Module (Line 18)

Description: Passwords are hashed using md5(). MD5 is unsuitable for password storage because it is a fast, unsalted hashing algorithm, making offline brute-force and dictionary attacks significantly easier and more practical.

Evidence:



Impact: Exposed MD5 hashes can be easily cracked using rainbow tables, exposing plaintext passwords.

Remediation: Replace md5() with PHP's native password_hash() function, which utilizes robust algorithms like Bcrypt with automatic salting.

Vulnerability 6:

- Vulnerability Name: Hardcoded Database Credentials
- CVSS Score: 3.7 (Low)
- **Location:** Database Connection (Line 6)

Description: The database connection string contains a hardcoded plaintext password.

Evidence:

```
$db = new PDO('mysql:host=localhost;dbname=corporate_db', 'db_user', 'SecretPass123!');
```

Impact: Anyone who gains access to the source code (e.g., via the LFI vulnerability) can extract the database credentials and gain unauthorized database access.

Remediation: Store sensitive credentials in secure environment variables or configuration files located outside the public web root directory.

Vulnerability 7:

- Vulnerability Name: Potential Reflected Cross-Site Scripting (XSS)
- CVSS Score: 6.1 (Medium)
- **Location:** User Authentication Module (Line 23)

Description: The application reflects unescaped user-controlled output from $_POST['username']. While traditional exploitation via a malicious link is harder for POST requests, it remains exploitable depending on how the endpoint is invoked and the browser context.

Evidence:

```
    echo "Invalid credentials for user: " . $_POST['username'];
}
```

Impact: Attackers can inject malicious JavaScript payloads. If executed within a victim's browser session, it can lead to session cookie theft, forced redirects, or unauthorized actions performed on behalf of the user.

Remediation: Always encode user-supplied data before rendering it in the HTML context. Use PHP's native htmlspecialchars() function with the ENT_QUOTES flag to convert special characters to HTML entities, neutralizing any injected scripts.