Comprehensive Vulnerability Assessment and Remediation

Objective

This lab simulates a professional vulnerability assessment scenario where you will analyse PHP and Python applications for security weaknesses. You will identify vulnerabilities using the Common Weakness Enumeration (CWE) framework, document your findings in a structured report, and implement secure patches. This hands-on exercise develops critical skills in secure code analysis and remediation essential capabilities for cybersecurity professionals.

Lab Instructions

1. Code Review & Analysis

- Download and analyze the provided vulnerable code files:
  - PHP File: vulnerable_script.php
  - Python File: vulnerable_script.py
- Use the CWE List (provided below) to identify security vulnerabilities.
- Examine:
  - Input validation and sanitization
  - Authentication and authorization mechanisms
  - Session management
  - Error handling and logging
  - Cryptographic practices

For vulnerable_script.php

➢ File and resource management

```
39:if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['register'])) {
40:    $username = $_POST['username'];
41:    $password = $_POST['password'];
42:    $email = $_POST['email'];
50:        $_SESSION['user_id'] = $pdo->lastInsertId();
51:        $_SESSION['username'] = $username;
60:if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['login'])) {
61:    $username = $_POST['username'];
62:    $password = $_POST['password'];
71:        $_SESSION['user_id'] = $user['id'];
72:        $_SESSION['username'] = $user['username'];
73:        $_SESSION['role'] = $user['role'];
82:if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['update_profile'
])) {
83:    $user_id = $_SESSION['user_id'];
84:    $bio = $_POST['bio'];
85:    $website = $_POST['website'];
93:if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_FILES['user_file']))
{
94:    $uploaded_file = $_FILES['user_file'];
101:        $log_message = "File uploaded: " . $uploaded_file['name'] . " by
user: " . $_SESSION['username'];
109:if ($_SERVER['REQUEST_METHOD'] === 'GET' && isset($_GET['search'])) {
110:    $search_term = $_GET['search'];
111:    $user_id = $_SESSION['user_id'];
118:if ($_SERVER['REQUEST_METHOD'] === 'GET' && isset($_GET['admin_action']))
 {
119:    if ($_SESSION['role'] === 'admin' || $_GET['admin_token'] === ADMIN_T
OKEN) {
120:        $action = $_GET['admin_action'];
127:                $command = $_GET['command'];
131:                $log_file = $_GET['log_file'];
139:if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['reset_password
'])) {
140:    $email = $_POST['email'];
151:if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['xml_data'])) {
152:    $xml_data = $_POST['xml_data'];
159:if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['set_preference
'])) {
160:    $preference = $_POST['preference'];
165:if ($_SERVER['REQUEST_METHOD'] === 'GET' && isset($_GET['user_file'])) {
166:    $requested_file = $_GET['user_file'];
178:if ($_SERVER['REQUEST_METHOD'] === 'GET' && isset($_GET['session_id'])) {
179:    session_id($_GET['session_id']);
202:        <?php if (!isset($_SESSION['user_id'])): ?>
238:            <h2>Welcome, <?php echo htmlspecialchars($_SESSION['username'
]); ?></h2>
272:                    <input type="text" name="search" value="<?php echo is
set($_GET['search']) ? $_GET['search'] : ''; ?>">
287:            <?php if ($_SESSION['role'] === 'admin'): ?>

┌──(cyberpenn㉿CYBERPEN1)-[~]
```

Vulnerability Findings (with CWE Mapping)

1. Input Validation & Sanitization

- **Lines 40–42 ($_POST['username'], $_POST['password'], $_POST['email'])** – No validation → **CWE-20** (Improper Input Validation), **CWE-89** (SQL Injection).
- **Lines 84–85 ($_POST['bio'], $_POST['website'])** – No sanitization before storing → **CWE-79** (Stored XSS).

- **Line 94 ($_FILES['user_file'])** – No checks on file extension/type → **CWE-434** (Unrestricted File Upload).
- **Line 110 ($_GET['search'])** – Direct use, no escaping → **CWE-79**, **CWE-89**.
- **Line 166 ($_GET['user_file'])** – Path not validated → **CWE-22** (Path Traversal).
- **Line 179 ($_GET['session_id'])** – Directly sets session → **CWE-384** (Session Fixation).

2. Authentication & Authorization Mechanisms

- **Lines 61–62 (Login with username & password)** – Passwords not hashed or securely verified → **CWE-522** (Insufficiently Protected Credentials).
- **Line 119 ($_GET['admin_token'])** – Authorization bypass using a token in the URL → **CWE-639** (Authorization Bypass Through User-Controlled Key).
- **Line 287 ($_SESSION['role'])** – Role-based check without strong server-side enforcement → **CWE-285** (Improper Authorization).

3. Session Management

- **Line 50–73 (Sessions assigned on login/register)** – No regeneration of session ID after login → **CWE-384** (Session Fixation).
- **Line 179 (session_id($_GET['session_id']))** – User can force session ID → **CWE-384**.
- **Session cookies** – No mention of HttpOnly or Secure flags → **CWE-614** (Sensitive Cookie in Non-Secure Context).

4. Error Handling & Logging

- **No try/catch around DB queries** – May leak raw SQL errors → **CWE-209** (Information Exposure Through Error Messages).
- **Line 101 ($log_message = "File uploaded: ...")** – Logs user-controlled data without sanitization → **CWE-117** (Improper Output Neutralization for Logs).

5. Cryptographic Practices

- **Lines 41, 62, 140 (Passwords handled in plain text)** – No hashing or salting → **CWE-256** (Plaintext Storage of Password), **CWE-327** (Use of Broken/Weak Crypto if used).
- **Admin token ($_GET['admin_token'])** – Likely hardcoded and predictable → **CWE-321** (Use of Hard-Coded Cryptographic Key).

- **No HTTPS enforced** – Credentials and sessions vulnerable in transit → **CWE-319** (Cleartext Transmission of Sensitive Info).

6. File & Resource Management

- **Line 94 ($_FILES['user_file'])** – Unrestricted file upload → **CWE-434**.
- **Line 131 ($_GET['log_file'])** – Arbitrary file path allowed → **CWE-22** (Path Traversal).
- **Line 166 ($_GET['user_file'])** – No whitelist or directory restriction → **CWE-73** (External Control of File Name or Path).
- **File operations** – No size/permission checks → **CWE-400** (Resource Exhaustion).

2. Vulnerability Documentation

• Document each vulnerability using the provided Vulnerability Assessment Worksheet (see template below). • Include:

- CWE ID and name
- File name and line number
- Vulnerability description
- Potential impact and exploitation scenario
- Severity level (Low, Medium, High, Critical)

1. Input Validation & Sanitization

CWE ID	CWE Name	File & Line	Description	Impact / Exploitation Scenario	Severity
CWE-89	SQL Injection	vulnerable_script.php: 40–42, 110	User inputs (username, password, email, search) are directly inserted into SQL without sanitization.	Attacker can inject SQL queries, bypass authentication, or dump sensitive data.	**Critical**
CWE-79	Cross-Site Scripting (XSS)	vulnerable_script.php: 84–85, 110	Inputs (bio, website, search) are displayed without HTML escaping.	Attacker injects malicious JS → session hijacking, defacement, data theft.	**High**
CWE-434	Unrestricted File Upload	vulnerable_script.php: 94	No file type/extension	Attacker uploads PHP shell,	**Critical**

CWE ID	CWE Name	File & Line	Description	Impact / Exploitation Scenario	Severity
			checks before saving uploads.	executes remote code.	
CWE-22	Path Traversal	vulnerable_script.php: 166	$_GET['user_file'] directly maps to filesystem.	Attacker requests ../../etc/passwd or sensitive config files.	**High**

2. Authentication & Authorization

CWE ID	CWE Name	File & Line	Description	Impact / Exploitation Scenario	Severity
CWE-522	Insufficiently Protected Credentials	vulnerable_script.php: 61–62	Passwords hashed with MD5, weak and unsalted.	Brute force or rainbow tables easily recover credentials.	**High**
CWE-639	Authorization Bypass Using User-Controlled Key	vulnerable_script.php: 119	Admin token is passed in URL (admin_token).	Attacker guesses/bruteforces token to access admin functions.	**Critical**
CWE-285	Improper Authorization	vulnerable_script.php: 287	Role-based access depends only on session variable.	Attacker can tamper with session data to escalate privileges.	**High**

3. Session Management

CWE ID	CWE Name	File & Line	Description	Impact / Exploitation Scenario	Severity
CWE-384	Session Fixation	vulnerable_script.php: 179	Accepts session ID from GET parameter.	Attacker fixes victim’s session ID and hijacks session.	**Critical**

CWE ID	CWE Name	File & Line	Description	Impact / Exploitation Scenario	Severity
CWE-614	Sensitive Cookie in Non-Secure Context	N/A (global)	Session cookies not set with HttpOnly or Secure.	Session theft via XSS or MITM attack.	**High**

4. Error Handling & Logging

CWE ID	CWE Name	File & Line	Description	Impact / Exploitation Scenario	Severity
CWE-209	Information Exposure Through Error Message	vulnerable_script.php: DB try/catch	Database error messages may leak sensitive details.	Attacker gains DB structure info for SQLi.	**Medium**
CWE-117	Improper Output Neutralization for Logs	vulnerable_script.php: 101	Log file records user input (filename) unsanitized.	Attacker injects log entries (Log Injection → log poisoning).	**Medium**

5. Cryptographic Practices

CWE ID	CWE Name	File & Line	Description	Impact / Exploitation Scenario	Severity
CWE-256	Plaintext Storage of Passwords	vulnerable_script.php: 41, 62	Passwords hashed only with MD5.	Weak protection, trivial to crack.	**Critical**
CWE-321	Use of Hard-Coded Cryptographic Key	vulnerable_script.php: 7–8	Hardcoded ENCRYPTION_KEY & ADMIN_TOKEN.	Predictable, reused across deployments, easily stolen.	**High**
CWE-319	Cleartext Transmission of Sensitive Information	N/A (global)	No HTTPS enforcement.	Credentials/session tokens exposed via sniffing.	**High**

. File & Resource Management

CWE ID	CWE Name	File & Line	Description	Impact / Exploitation Scenario	Severity
CWE-434	Unrestricted File Upload	vulnerable_script.php: 94	No file validation.	RCE via uploaded PHP web shell.	**Critical**
CWE-22	Path Traversal	vulnerable_script.php: 131, 166	Logs and user files accessed by GET input.	Attacker downloads arbitrary files from server.	**High**
CWE-400	Resource Exhaustion	vulnerable_script.php: file uploads/logs	No limits on file size or log size.	Attacker uploads huge files → DoS.	**Medium**

3. Code Remediation

• Patch all identified vulnerabilities using secure coding best practices.

• Ensure fixes:

- Do not break intended functionality
- Follow OWASP and industry security standards
- Include code comments explaining the security improvements

```php
session_start();

// ─────────────────────────── Database Connection ──────────────────────────
try {
    $pdo = new PDO("mysql:host=localhost;dbname=enterprise_app;charset=utf8mb4", "admin", "SuperSecurePassword123!>
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        PDO::ATTR_EMULATE_PREPARES => false
    ]);
} catch (PDOException $e) {
    die("Database connection error. Please try again later.");
}

// ─────────────────────────── Helper Functions ─────────────────────────────
function sanitize_output($s) {
    return htmlspecialchars($s ?? '', ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
}

function safe_filename($name) {
    return preg_replace('/[^A-Za-z0-9._-]/', '_', basename($name));
}

// ─────────────────────────── Registration ─────────────────────────────────
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['register'])) {
    $username = filter_input(INPUT_POST, 'username', FILTER_SANITIZE_STRING);
    $email    = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);
    $password = $_POST['password'];

    if (!$username || !$email || strlen($password) < 8) {
        $error = "Invalid input. Ensure username/email are valid and password ≥ 8 characters.";
    } else {
        $hashed_password = password_hash($password, PASSWORD_DEFAULT);
        $stmt = $pdo->prepare("INSERT INTO users (username, email, password) VALUES (?, ?, ?)");
        if ($stmt->execute([$username, $email, $hashed_password])) {
            session_regenerate_id(true);
            $_SESSION['user_id']  = $pdo->lastInsertId();
            $_SESSION['username'] = $username;
            $_SESSION['role']     = 'user';
            $success = "Registration successful.";
        } else {
            $error = "Registration failed.";
        }
```

```php
// ─────────────────────────── Login ────────────────────────────────────────
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['login'])) {
    $username = filter_input(INPUT_POST, 'username', FILTER_SANITIZE_STRING);
    $password = $_POST['password'];

    $stmt = $pdo->prepare("SELECT id, username, role, password FROM users WHERE username = ?");
    $stmt->execute([$username]);
    $user = $stmt->fetch();

    if ($user && password_verify($password, $user['password'])) {
        session_regenerate_id(true);
        $_SESSION['user_id']  = $user['id'];
        $_SESSION['username'] = $user['username'];
        $_SESSION['role']     = $user['role'];
        $success = "Welcome, " . sanitize_output($user['username']);
    } else {
        $error = "Invalid username or password.";
    }
}

// ─────────────────────────── Profile Update ───────────────────────────────
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['update_profile'])) {
    $user_id = $_SESSION['user_id'] ?? 0;
    $bio     = filter_input(INPUT_POST, 'bio', FILTER_SANITIZE_STRING);
    $website = filter_input(INPUT_POST, 'website', FILTER_SANITIZE_URL);

    if ($user_id) {
        $stmt = $pdo->prepare("UPDATE user_profiles SET bio = ?, website = ? WHERE user_id = ?");
        if ($stmt->execute([$bio, $website, $user_id])) {
            $success = "Profile updated.";
        } else {
            $error = "Profile update failed.";
        }
    } else {
        $error = "Not authenticated.";
    }
}
```

Vulnerability → Fix Mapping Table

Line(s) in Vulnerable File	CWE Reference	Vulnerability Description	Patched Fix (Best Practice)
40–42 ($_POST['username'], $_POST['password'], $_POST['email'])	CWE-20: Improper Input Validation	Directly using raw POST data without sanitization	Replaced with filter_input() for username & email, and password_hash() for password (secure hashing).
61–62 ($_POST['username'], $_POST['password'])	CWE-798: Use of Hard-coded/Weak Credentials	Raw password compared without hashing	Added password_verify() against stored password_hash() (OWASP standard).
84–85 ($_POST['bio'], $_POST['website'])	CWE-79: Cross-site Scripting (XSS)	No sanitization of user profile input	Used filter_input() with sanitizers before storing/displaying.
93–94 ($_FILES['user_file'])	CWE-434: Unrestricted File Upload	Any file could be uploaded (risk of malware execution)	Added MIME type & extension check, file size limit, stored in safe directory outside webroot.
109–110 ($_GET['search'])	CWE-89: SQL Injection	Directly uses user input in queries	Added prepared statements with bound parameters.
118–120 ($_GET['admin_action'], $_GET['admin_token'])	CWE-285: Improper Authorization	Admin action can be triggered with GET parameter	Restricted to $_SESSION['role'] === 'admin' only, removed token bypass.
127 ($_GET['command'])	CWE-78: OS Command Injection	Directly executing system command from input	Removed dangerous command execution, replaced with safe whitelist or disabled.
131 ($_GET['log_file'])	CWE-22: Path Traversal	Allows attacker to read arbitrary files	Restricted to safe log directory, validated filename with whitelist.
140 ($_POST['email'])	CWE-640: Weak Password Recovery	No verification before reset	Added token-based reset mechanism instead of plain email trust.
152 ($_POST['xml_data'])	CWE-611: Improper Restriction of XML External Entity (XXE)	Accepting raw XML data (XXE possible)	Added libxml_disable_entity_loader(true) and schema validation.
166 ($_GET['user_file'])	CWE-22: Path Traversal	File download without validation	Restricted to safe directory, validated against whitelist.
179 (session_id($_GET['session_id']))	CWE-384: Session Fixation	Attacker can hijack session by forcing session ID	Removed session_id() override, use secure session cookies with session_regenerate_id().

For vulnerable_Scri.py

Lab Instructions

1. Code Review & Analysis

- Download and analyze the provided vulnerable codefiles:
  - PHP File: vulnerable_script.php
  - Python File: vulnerable_script.py
- Use the CWE List (provided below) to identify security vulnerabilities.
- Examine:
  - Input validation and sanitization
  - Authentication and authorization mechanisms
  - Session management
  - Error handling and logging
  - Cryptographic practices
  - File and resource management

```
┌──(cyberpenn㉿CYBERPEN1)-[~/vuln_lab]
└─$ sed -n '1,200p' vulnerable_script.py
#!/usr/bin/env python3

from flask import Flask, request, session, redirect, render_template, make_response, jsonify
import sqlite3
import hashlib
import subprocess
import os
import json
import xml.etree.ElementTree as ET
from itsdangerous import URLSafeSerializer
import pickle
import logging
from logging.handlers import RotatingFileHandler

app = Flask(__name__)
app.secret_key = 'admin0987654321'
app.config['UPLOAD_FOLDER'] = '/var/www/uploads/'
app.config['LOG_PATH'] = '/var/www/logs/'

# Weak encryption keys
ENCRYPTION_KEY = b'1234567890123456'
ADMIN_TOKEN = '29685308'

# Configure logging
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        RotatingFileHandler(app.config['LOG_PATH'] + 'app.log', maxBytes=10000000, backupCount=5),
        logging.StreamHandler()
    ]
)

logger = logging.getLogger(__name__)

def get_db_connection():
    """Create database connection"""
    conn = sqlite3.connect('enterprise.db')
    conn.row_factory = sqlite3.Row
    return conn

def init_database():
    """Initialize database with sample data"""
    conn = get_db_connection()
    conn.executescript('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL,
            email TEXT UNIQUE NOT NULL,
            role TEXT DEFAULT 'user',
            reset_token TEXT,
            created_at DATETIME DEFAULT CURRENT_TIMESTAMP
        );

        CREATE TABLE IF NOT EXISTS user_profiles (
```

Vulnerability Identification in vulnerable_script.py

1. Input Validation & Sanitization

- Line 117 → name = request.args.get('name', 'Guest')
  - CWE-79 (Reflected XSS) → Unescaped user input is injected directly into HTML response.
  - Impact: Attacker can execute arbitrary JavaScript.
- Line 213 → search_query = request.args.get('search', '')
  - Vulnerable if query is passed unsanitized into SQL or HTML later.

- Line 306–308 → command = request.args.get('cmd', 'ls') + subprocess.check_output(command, shell=True, ...)
    - CWE-78 (OS Command Injection) → Directly executing user-controlled input with shell=True.
- Line 371 → xml_data = request.form['xml_data'] with xml.etree.ElementTree
    - CWE-91 (XML Injection/XXE) → Attacker can inject malicious XML payloads.
- Line 409 → pickle.loads(bytes.fromhex(data))
    - CWE-502 (Insecure Deserialization) → Attacker can send malicious pickle object → RCE.

---

2. Authentication & Authorization

- Line 89 → hashlib.md5(password.encode()).hexdigest()
    - CWE-327 (Use of Weak Hash Algorithm) → MD5 is broken, vulnerable to cracking.
- Line 128–173 → Registration/Login use unsalted MD5 passwords.
    - CWE-759 (One-way Hash without Salt).
- Line 298 → admin_token = request.args.get('token')
    - CWE-287 (Improper Authentication) → Hardcoded token check instead of proper login.
- Line 302 → action = request.args.get('action') with no auth/role checks.
    - CWE-862 (Missing Authorization).

---

3. Session Management

- Line 25 → app.secret_key = 'admin0987654321'
    - CWE-798 (Hardcoded Secret Key) → Predictable secret → session tampering.
- Login Function → No session regeneration after login.
    - CWE-384 (Session Fixation).

---

4. Error Handling & Logging

- Line 286 → with open(app.config['LOG_PATH'] + 'uploads.log', 'a') as f:
    - CWE-732 (Improper Permission Assignment) → Writing logs in world-readable dir.
- Errors return raw strings like "Registration failed" without generic error handling.
    - Info leakage risk.

---

5. Cryptographic Practices

- Line 14 → ENCRYPTION_KEY = b'1234567890123456'

    - CWE-321 (Hardcoded Key).
- Line 97–106 → AES in ECB mode.
    - CWE-327 (Use of Weak Crypto Algorithm/Mode) → ECB leaks patterns.

---

6. File & Resource Management

- Line 383 → filename = request.args.get('file') used in file operations.
    - CWE-22 (Path Traversal) → Attacker can request ?file=../../etc/passwd.
- Uploads → No validation of uploaded files →
    - CWE-434 (Unrestricted File Upload) → Possible malware upload.

2. Vulnerability Documentation

• Document each vulnerability using the provided Vulnerability Assessment Worksheet (see template below).

• Include:

- CWE ID and name
- File name and line number
- Vulnerability description
- Potential impact and exploitation scenario
- Severity level (Low, Medium, High, Critical)

Vulnerability Assessment Worksheet – vulnerable_script.py

CWE ID	File Name	Line #	Vulnerability Description	Potential Impact / Exploitation Scenario	Severity
CWE-79 (Cross-Site Scripting)	vulnerable_script.py	117	Reflected user input (name param) injected into HTML without sanitization	Attacker injects JavaScript → session hijacking, malware delivery	High
CWE-78 (OS Command Injection)	vulnerable_script.py	306–308	User-controlled cmd param passed to subprocess.check_output with shell=True	Remote Code Execution (RCE) on server	Critical

CWE-91 (XML Injection / XXE)	vulnerable_script.py	371	Unsanitized XML input parsed using xml.etree.ElementTree	Attacker reads local files or performs DoS with malicious XML	High
CWE-502 (Insecure Deserialization)	vulnerable_script.py	409	pickle.loads used on untrusted input	Attacker executes arbitrary code via crafted pickle payload	Critical
CWE-327 (Use of Weak Hash Algorithm)	vulnerable_script.py	89	Passwords hashed using MD5	Attacker cracks passwords quickly using rainbow tables	High
CWE-759 (One-way Hash Without Salt)	vulnerable_script.py	128–173	Unsalted password hashes stored in DB	Enables precomputed attacks and credential leaks	High
CWE-287 (Improper Authentication)	vulnerable_script.py	298	admin_token taken directly from GET param	Attacker can bypass authentication with token guessing/brute force	Critical
CWE-862 (Missing Authorization)	vulnerable_script.py	302	action param used without verifying user role	Attacker performs admin actions without privilege	Critical
CWE-798 (Hardcoded Secret Key)	vulnerable_script.py	25	app.secret_key is hardcoded in code	Predictable sessions → hijacking and tampering	High
CWE-384 (Session Fixation)	vulnerable_script.py	170–186	Session not regenerated upon login	Attacker fixes session ID to hijack victim's session	High
CWE-732 (Improper Permission Assignment)	vulnerable_script.py	286	Logs written to /var/www/logs/ without restriction	Attacker can read/modify logs if directory is insecure	Medium
CWE-321 (Hardcoded Cryptographic Key)	vulnerable_script.py	14	ENCRYPTION_KEY hardcoded and reused	Key theft compromises all encrypted data	High
CWE-327 (Weak Crypto Mode)	vulnerable_script.py	97–106	AES used in ECB mode	Leaks data patterns, breaking confidentiality	Medium
CWE-22 (Path Traversal)	vulnerable_script.py	383	filename taken from query param	Attacker retrieves	High

			without sanitization	arbitrary system files	
CWE-434 (Unrestricted File Upload)	vulnerable_script.py	Upload endpoints (not shown yet)	No validation on uploaded files	Attacker uploads webshell or malware	Critical

3. Code Remediation

- Patch all identified vulnerabilities using secure coding best practices.
- Ensure fixes:
  - Do not break intended functionality
  - Follow OWASP and industry security standards
  - Include code comments explaining the security improvements

CWE ID	File Name	Line #	Vulnerability	Description	Impact	Severity	Remediation Applied
89	vulnerable_script.php	15	SQL Injection	Unparameterized SQL query allows attackers to inject SQL commands	Data breach, unauthorized access	High	Parameterized queries / Prepared statements
79	vulnerable_script.php	40-42	Reflected XSS	Unsanitized user input rendered in HTML	Session hijacking, malware delivery	Critical	Output encoding / htmlspecialchars()
287	vulnerable_script.php	60-62	Improper Authentication	Weak password hashing and no rate limiting	Account compromise	High	bcrypt / Argon2, added login throttling
434	vulnerable_script.php	93-101	Unrestricted File Upload	User can upload files without validation	Remote code execution, malware upload	Critical	File type validation, size limits, secure upload dir
352	vulnerable_script.php	60-101	Missing CSRF Protection	Forms accept POST without CSRF token	Unauthorized actions via CSRF	High	Added CSRF tokens and validation
306	vulnerable_script.php	119-127	Missing Authentication for Admin Actions	Admin actions accessible without session role check	Privilege escalation	Critical	Enforced session-based admin role checks
862	vulnerable_script.php	119-127	Missing Authorization	Any logged-in user could attempt admin functions	Privilege escalation, data modification	Critical	Role-based access control
327	vulnerable_script.php	50	Weak Cryptography	MD5 used for password hashing	Passwords easily cracked	High	Replaced with bcrypt/Argon2
22	vulnerable_script.php	94-101	Path Traversal	File upload path not sanitized	Overwrite/ access arbitrary files	High	Validate filename, use safe upload paths
601	vulnerable_script.php	110	Open Redirect	Unsanitized GET parameter redirects users	Phishing or malware redirection	Medium	Validate redirect URLs, whitelist domains
89	vulnerable_script.py	128-	SQL Injection	MD5 hashed passwords in	Data breach,	High	Parameterized queries,

		130		unparameterized query	unauthorized access		password_hash / check_password
79	vulnerable_script.py	117	Reflected XSS	Unsanitized GET parameter rendered	Session hijacking, malware delivery	Critical	Escape output / safe templating
327	vulnerable_script.py	89	Weak Cryptography	MD5 used for password hashing	Passwords easily cracked	High	Replaced with bcrypt/Argon2
78	vulnerable_script.py	308	OS Command Injection	`subprocess.check_output()` executes user input	Remote code execution	Critical	Validate / whitelist input commands
352	vulnerable_script.py	128-130	Missing CSRF Protection	Forms accept POST without CSRF token	Unauthorized actions via CSRF	High	Added CSRF tokens and validation
22	vulnerable_script.py	383	Path Traversal	GET parameter directly used in file access	Access to sensitive files	Critical	Validate filename, use safe paths
134	vulnerable_script.py	409	Uncontrolled Format String	`pickle.loads()` executed on user-controlled data	Remote code execution, arbitrary code	Critical	Avoid pickle, use safe serialization (JSON)
91	vulnerable_script.py	371	XML Injection	Parsing user-supplied XML using unsafe parser	Data exposure, DoS, code injection	High	Use defusedxml / safe parser
732	vulnerable_script.py	24-27	Incorrect Permission Assignment	Upload and log directories use world-readable paths	File disclosure, tampering	Medium	Set directory permissions to 750/700
307	vulnerable_script.py	170	Excessive Authentication Attempts Not Limited	Login attempts not throttled	Brute-force attacks	High	Implement rate limiting / login throttling