E-commerce Website Security Assessment & Remediation Plan

Arpan Patel
Date: 07/09/2025

1. Introduction

This report checks the security of a small e-commerce website.

The site uses a LAMP stack: Linux (OS), Apache (web server), MySQL (database), and PHP (code).

The goal is to find common problems and suggest clear fixes.

The work links to Cyber Essentials Plus (a basic UK security standard).

The focus is the website: login, search, and server settings.

2. Top 10 Common Web Vulnerabilities (OWASP)

Short list in simple words:

1) Broken Access Control – users can see or change data they should not. Impact: privacy loss.

2) Cryptographic Failures – weak or no encryption. Impact: stolen personal data.

3) Injection (SQLi) – unsafe input goes into database. Impact: database theft.

4) Insecure Design – design does not think about security early. Impact: easy to chain attacks.

5) Security Misconfiguration – bad defaults and verbose errors. Impact: info leaks.

6) Vulnerable/Outdated Components – old software. Impact: known bugs exploited.

7) Identification/Authentication Failures – weak login. Impact: account takeover.

8) Software/Data Integrity Failures – unverified updates. Impact: supply-chain risk.

9) Logging/Monitoring Failures – missing or weak logs. Impact: slow detection.

10) Server-Side Request Forgery – server fetches attacker links. Impact: internal exposure.

3. Issues in Given Code and Configuration

A) PHP Login Script

- SQL Injection (unsafe string in SQL).

- Passwords not hashed (plain text compare).

- DB username/password hardcoded in code.

- No lockout/MFA; message helps attackers.

Fix (short):

- Use prepared statements and password hashing.

- Move DB creds to environment variables.

- Add lockout and MFA; use generic error messages.

B) JavaScript Search

- Uses innerHTML with user input → XSS risk.

Fix (short):

- Use textContent, validate/encode, add CSP header.

C) Apache Log Settings

- Default permissions may be too open.

- Logs only on one server (easy to delete).

- No alerts for strange events.

Fix (short):

- Set perms (640), keep logs outside web-root.

- Send logs to central store; keep 90 days.

- Add alerts for failed logins/5xx spikes/public ACLs.

4. Prioritization of Fixes (what to do first)

1) SQL Injection + plain text password handling.

2) XSS in search feature.

3) Weak login (no lockout/MFA; weak sessions).

4) Log security + monitoring.

5) Patch and harden LAMP (Linux, Apache, MySQL, PHP).

5. Detailed Fix Plan for Top 3 Issues

1) SQL Injection + Password Security

- Code: prepared statements (PDO/MySQLi); use password_hash()/password_verify().

- Config: DB creds in env vars; least-privilege DB user.

- Headers/WAF: enable ModSecurity + OWASP CRS.

- Ops: re-hash old passwords; code review for auth.

2) XSS in Search

- Code: replace innerHTML → textContent.

- Server: encode output; validate input.

- Headers: add CSP, X-Content-Type-Options.

- Ops: tests to prevent unsafe DOM updates.

3) Authentication/Brute Force

- Code: lockout after 5 fails; generic errors.

- Config: MFA for admin; session cookie Secure/HttpOnly/SameSite.

- Rate limit login endpoint.

- Ops: alert on failed login bursts; phishing training.

6. Breaking Things Attack Scenario (chained)

Story: attacker injects script via search (XSS). Admin opens analytics page that shows user queries. Script steals admin session. Attacker uploads a PHP web shell and steals the database.

Why it fails after fixes:

- XSS blocked (textContent + CSP).

- HttpOnly cookies stop JS from reading session.

- Uploads do not allow .php; stored outside web-root.

- Logs + alerts show strange admin actions fast.

7. AI Help and My Own Work

AI helped make a fast list of common risks and first ideas.

But AI was generic.

My own work was to choose the top risks for this LAMP shop, write step-by-step fixes, and design a chained attack that our defenses stop.

I also linked the plan to Cyber Essentials Plus and kept the language simple.

8. Tools Used

AI: ChatGPT for quick ideas and OWASP list.

Non-AI: OWASP docs, PHP manual (password_hash/verify), Apache docs, Word editor.

(We did not run scans on real sites.)

9. Cyber Essentials Plus Basics (how this plan helps)

- Firewalls/Ports: only 80/443 open for the site.

- Secure Config: disable verbose errors; safe permissions; add CSP.

- Access Control: strong passwords, MFA, least privilege.

- Malware Protection: scan uploads; do not execute uploads.

- Patch Management: keep Linux/Apache/MySQL/PHP up to date.

10. The Crucial 10% (My Personal Input)

This part is my own thinking and choices.

A) Why I picked these priorities

- Rule used: high impact + easy to exploit → fix first.

- SQL Injection + plain passwords can give full DB access.

- XSS can steal sessions and payment data.

- Weak login can give admin access to attackers.

- Logs/patching help detect and prevent future attacks.

B) Extra, specific steps beyond AI

1) SQL Injection + Passwords

  - SQL: avoid SELECT *; fetch only needed fields.

  - Hashing: migrate old passwords to password_hash(); force reset if needed.

  - DB rights: no DROP/ALTER for app user; only SELECT/UPDATE as required.

  - Header line (Apache example):

- Content-Security-Policy: "default-src 'self'; script-src 'self'; object-src 'none'"

- Test: add unit tests that fail if raw user input reaches SQL.

2) XSS

- PHP output: use htmlspecialchars($value, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8').

- JS: forbid innerHTML in code review unless wrapped safely.

- Template engines that auto-escape by default.

- Keep a small allow-list for search input (letters, numbers, spaces).

3) Authentication

- Back-off: after 5 fails, wait 15 minutes; increase delay on more fails.

- PHP sessions:
session_set_cookie_params(['secure'=>true,'httponly'=>true,'samesite'=>'Strict']).

- Rate-limit by IP and by account name.

- Prefer SSO + MFA for admins.

C) Attack with a twist

- Twist: a “support ticket” link tricks admin to open a page that renders user search terms.

- Chain: social engineering → reflected XSS → session theft → web shell → DB dump.

- Blocked because: XSS blocked, session unreadable by JS, uploads cannot run PHP, alerts catch new admin actions.

D) Trade-offs I accepted

- Small login delays to stop brute force.

- Simpler logging/alerts instead of a full SIEM for a small team.

- Free tools (ModSecurity CRS) instead of paid WAF.

E) How I would check it works

- Unit tests for auth and encoding.

- Try SQLi/XSS inputs on a test site.

- Ensure alerts fire for failed login bursts and many 5xx errors.

- Review configs after updates.