**Mock Shop – Security Hardening & Vulnerability Assessment Report**

**1. Project Overview**

**Objective:** Deploy and secure an e-commerce platform (www.mock-shop.com) built on **Django, PostgreSQL, and Docker**.  
**Scope:** Address key web security risks (SQL Injection, XSS, CSRF), enforce security best practices, and validate through penetration testing.  
**Deliverables:**

* Secured live website (https://mockshop-1.onrender.com)
* Security Hardening Checklist

**2. Security Testing Approach**

1. **HTTPS Setup** – TLS/SSL configuration using Let’s Encrypt.
2. **WAF Rules** – ModSecurity baseline with OWASP Core Rule Set (CRS).
3. **Penetration Testing** – Conducted using OWASP ZAP 2.16.1.
4. **Hardening Guide** – Aligned with OWASP Top 10 and Django security guidelines.

**3. OWASP ZAP Scan Summary**

**Date:** 30 Aug 2025  
**Target:** https://mockshop-1.onrender.com  
**Tool:** OWASP ZAP by Checkmarx

**3.1 Alert Breakdown**

| **Risk Level** | **Count** | **Example Finding** |
| --- | --- | --- |
| High | 1 | SQL Injection (SQLite) |
| Medium | 1 | Missing CSP Header |
| Low | 4 | Cookie flags, HSTS missing |
| Informational | 6 | Cache issues, suspicious comments |

**Total Alerts:** 12

**3.2 High-Risk Findings**

**SQL Injection – SQLite**

* **Location:** POST /cart/add/2/ (parameter: quantity)
* **Description:** Possible time-based SQL injection payload detected.
* **Analysis:** Likely a false positive. The application uses Django ORM with strict integer validation. No raw SQL queries are present.
* **Mitigation:**
  + Enforced strict input validation via Django forms (IntegerField(min\_value=1, max\_value=10)).
  + Verified all queries use ORM parameter binding.
  + No evidence of exploitability in manual testing.

**3.3 Medium-Risk Findings**

**Content Security Policy (CSP) Header Not Set**

* **Impact:** Increases risk of Cross-Site Scripting (XSS) and unsafe resource loading.
* **Mitigation:** Configured strict CSP headers in settings.py using django-csp:
* CSP\_DEFAULT\_SRC = ("'self'",)
* CSP\_SCRIPT\_SRC = ("'self'", 'cdn.jsdelivr.net')
* CSP\_STYLE\_SRC = ("'self'", 'cdn.jsdelivr.net')

**3.4 Low-Risk Findings**

* **Strict-Transport-Security (HSTS) Header Missing**
  + Mitigated by adding SECURE\_HSTS\_SECONDS = 31536000 in settings.
* **Cookies Without Security Flags**
  + Mitigated by enabling:
  + SESSION\_COOKIE\_SECURE = True
  + SESSION\_COOKIE\_HTTPONLY = True
  + CSRF\_COOKIE\_SECURE = True
  + CSRF\_COOKIE\_HTTPONLY = True
* **Cross-Domain JavaScript Inclusion**
  + External JavaScript sources restricted to trusted CDNs only.

**3.5 Informational Findings**

* **Suspicious Comments:** Developer comments reviewed and removed.
* **Cache Control:** Updated headers to disable sensitive caching:
* SECURE\_BROWSER\_XSS\_FILTER = True
* SECURE\_CONTENT\_TYPE\_NOSNIFF = True
* **Session Management:** Verified Django generates and enforces secure session IDs.

**4. Security Hardening Checklist**

* HTTPS enforced with HSTS
* CSRF protection enabled through Django middleware
* SQL Injection prevention (Django ORM with strict input validation)
* XSS protection (CSP headers, template auto-escaping)
* Secure cookies with Secure and HttpOnly flags
* WAF with ModSecurity CRS rules applied
* Debug mode disabled, sensitive comments removed
* Dependencies monitored with pip-audit and safety

**5. Conclusion & Recommendations**

The e-commerce platform has been deployed and hardened against common web application threats.

* The only high-risk finding (SQL Injection) was determined to be a false positive.
* Medium and low-risk issues were resolved by strengthening Django settings and HTTP response headers.

**Recommendations:**

1. Perform periodic OWASP ZAP scans (monthly).
2. Maintain active monitoring through the WAF.
3. Apply regular dependency and Django security updates.

**6. Deliverables Completed**

* Secured live website at https://mockshop-1.onrender.com
* Security hardening checklist prepared and implemented

A screenshot of a computer

AI-generated content may be incorrect.

