Web Application Security Assessment – OWASP Top 10

Scope & Methodology

The assessment simulates a black-box security test of intentionally vulnerable web applications (PortSwigger Web Security Academy and OWASP Juice Shop). Manual testing techniques aligned with the OWASP Top 10 were used to identify common, high-impact web application security risks and evaluate their potential business impact.

Executive Summary

A web application security assessment was performed to identify vulnerabilities aligned with the OWASP Top 10. Several high-risk issues were identified, including SQL Injection, Broken Access Control (IDOR), and Cross-Site Scripting (XSS). Successful exploitation of these vulnerabilities could allow attackers to bypass authentication, access or manipulate sensitive user data, and execute malicious scripts in user browsers. Addressing these issues will significantly reduce the application's attack surface and overall security risk.

Finding 1 – SQL Injection (High Risk)

OWASP Category: A03:2021 – Injection
Severity: High (Direct database compromise possible)

Description

SQL Injection occurs when untrusted user input is incorporated into database queries without proper handling. This allows attackers to manipulate query logic and interact directly with the backend database in unintended ways.

Affected Functionality

- User authentication (login)
- Product search and filtering features

Impact

An attacker could:

- Bypass authentication controls

- Access or modify sensitive application data
- Extract user credentials
- Compromise the entire database
- Disrupt data integrity and confidentiality

Root Cause

- Use of dynamic SQL queries with user-controlled input
- Absence of parameterized queries or prepared statements
- Insufficient server-side input validation
- Over-privileged database accounts

Recommendation

- Use parameterized queries or prepared statements for all database access
- Validate and sanitize input on the server side
- Avoid constructing SQL queries using raw user input
- Enforce least-privilege permissions for database users
- Implement logging and monitoring for abnormal query behavior

---

Finding 2 – Broken Access Control (IDOR) (High Risk)

OWASP Category: A01:2021 – Broken Access Control
Severity: High (Horizontal privilege escalation)

Description

Insecure Direct Object Reference (IDOR) occurs when an application exposes internal object identifiers and fails to verify whether the authenticated user is authorized to access the requested resource. Attackers can manipulate object identifiers to access other users' data.

Affected Functionality

- Basket and cart management
- User-specific resources accessed via URL parameters

Impact

An attacker could:

- Access other users' cart or order data

- Modify or delete resources they do not own
- Perform unauthorized actions on behalf of other users
- Compromise data confidentiality and integrity

Root Cause

- Missing server-side authorization checks
- Direct reliance on client-supplied object identifiers
- Trust in user-controlled request parameters

Recommendation

- Enforce object-level authorization on every request
- Validate that the authenticated user owns or is permitted to access the requested resource
- Avoid exposing predictable object identifiers
- Log and monitor unauthorized access attempts

Finding 3 – Cross-Site Scripting (XSS) (Medium–High Risk)

OWASP Category: A07:2021 – Cross-Site Scripting
Severity: Medium–High (User compromise possible)

Description

Cross-Site Scripting (XSS) occurs when untrusted user input is rendered in application responses without proper output encoding. This allows attackers to execute malicious JavaScript in the context of a victim's browser.

Affected Functionality

- Search input fields
- Comment or review features

Impact

An attacker could:

- Hijack user sessions
- Steal authentication tokens or credentials
- Perform unauthorized actions as victims
- Deliver phishing or malicious client-side payloads

Root Cause

- Lack of context-aware output encoding
- Rendering of untrusted input in HTML responses
- Absence of a restrictive Content Security Policy (CSP)

Recommendation

- Apply context-specific output encoding (HTML, JavaScript, URL)
- Treat all user input as untrusted
- Implement a strong Content Security Policy (CSP)
- Use secure frameworks or templating engines with automatic escaping

---

Final Recommendations

- Adopt secure coding practices aligned with OWASP guidelines
- Perform regular security testing throughout development
- Integrate security checks into CI/CD pipelines
- Provide secure coding training for developers