FINAL PROJECT REPORT

WebScanPro: Automated Web Application Security Testing Tool

Project Name: WebScanPro

Intern Name: Samar Vijay Vishwakarma

Date: January 2, 2026

Target Application: OWASP Juice Shop

Technology Stack: Python, Selenium WebDriver, Docker

1. Abstract

WebScanPro is an automated security testing tool developed to identify critical vulnerabilities in modern web applications. Unlike traditional scanners that rely on static analysis, WebScanPro utilizes dynamic browser automation (Selenium) to interact with Single Page Applications (SPAs). The tool successfully automates the detection of **SQL Injection (SQLi)**, **Cross-Site Scripting (XSS)**, **Broken Authentication**, and **Insecure Direct Object References (IDOR)**, culminating in the generation of a comprehensive, professional HTML security report.

2. Project Architecture

The tool is built using a modular architecture, where each security test is isolated into its own execution block.

- **Core Engine:** Python 3.x
- **Browser Automation:** Selenium WebDriver (Chrome)
- **Target Environment:** Docker Container running OWASP Juice Shop (v14.x)
- **Reporting Engine:** Custom HTML/CSS Generator & JSON Serializer

3. Weekly Implementation Log

Week 1: Project Initialization & Environment Setup

Objective: Establish a safe, isolated testing environment to prevent accidental damage to real-world systems.

- **Implementation:** Deployed **OWASP Juice Shop** using Docker containers mapped to port 3000.
- **Version Control:** Initialized a Git repository to track code changes.
- **Outcome:** Verified application accessibility at http://localhost:3000.

Week 2: Target Scanning (The Technical Pivot)

Objective: Map the application structure and identify attack vectors.

- **Challenge:** Initial attempts using static parsers (BeautifulSoup) failed because the target is an Angular-based Single Page Application (SPA).
- **Solution:** Migrated the tech stack to **Selenium WebDriver**. This allowed the tool to launch a real browser, wait for the JavaScript to execute ("hydrate"), and interact with the DOM dynamically.
- **Feature:** Implemented a nuke_popups() function to automatically delete "Welcome" banners and "Cookie Consent" overlays using JavaScript execution.

Week 3: SQL Injection (Authentication Bypass)

Objective: Automate database exploitation.

- **Vector:** The Login Page (/login).
- **Payload:** ' OR 1=1 --.
- **Mechanism:** The tool injects a boolean-true condition into the email field, forcing the database to ignore the password verification step.
- **Result:** Successfully bypassed authentication and logged in as the Administrator without a valid password.

Week 4: Cross-Site Scripting (XSS)

Objective: Execute arbitrary JavaScript in the victim's browser.

- **Vector:** The Search API (/search?q=...).
- **Payload:** <iframe src="javascript:alert('XSS')">.
- **Technique:** Utilized **Direct URL Injection** to bypass UI timing issues and simulate a Reflected XSS attack.
- **Result:** Triggered an automated browser alert box, confirming that the application failed to sanitize user input.

Week 5: Authentication & Session Testing

Objective: Test credential strength and session security.

- **Brute Force Module:** implemented a dictionary attack using a list of common weak passwords. Successfully cracked the admin account using the password admin123.
- **Session Analysis:** Analyzed session cookies post-login. Identified that critical session tokens lacked the Secure flag (due to HTTP context), flagging a potential Session Hijacking risk.

Week 6: Access Control (IDOR)

Objective: Test for Insecure Direct Object References.

- **Vector:** The Shopping Basket API (/rest/basket/{id}).
- **Attack:** The tool iterated through Basket IDs (1, 2, 3) while logged in as a standard user.
- **Result:** Successfully retrieved the JSON data of baskets belonging to other users, proving a Horizontal Privilege Escalation vulnerability.

Week 7: Security Report Generation

Objective: Transform raw data into actionable intelligence.

- **Implementation:** Developed a reporting engine that aggregates findings from all modules.
- **Output:**
    1. vulnerability_report.json: Raw data for auditing.
    2. final_security_report.html: A styled, professional report table highlighting "Critical" and "High" severity issues.
- **Enhancements:** Added passive checks for **Security Misconfigurations** (missing HTTP headers) and **Insecure Storage** (JWT in LocalStorage).

Week 8: Final Documentation

Objective: Consolidate all findings into this final thesis.

- **Activity:** Compiled weekly logs, screenshots, and code snippets into a final submission document. Validated the tool's end-to-end execution stability.

4. Technical Challenges & Resolutions

Challenge	Root Cause	Resolution
0 Forms Found (Week 2)	Target is an Angular SPA; static parsers cannot read JS.	**Migrated to Selenium** to handle dynamic content.
UI Obstructions	"Welcome" banners blocked clicks.	Implemented nuke_popups() to delete overlays via JS.
Search SQLi Fail (Week 3)	Search bar sanitized input effectively.	**Pivoted to Login Bypass**, which proved vulnerable.
XSS Timing Issues (Week 4)	Script typed before animation finished.	Used **Direct URL Injection** to bypass typing entirely.
File Locks (Week 7)	Report file open in browser caused crash.	Added try/except blocks to warn user instead of crashing.

5. Evidence of Work (Screenshots)

Figure 1: Successful Admin Account Takeover (SQLi)

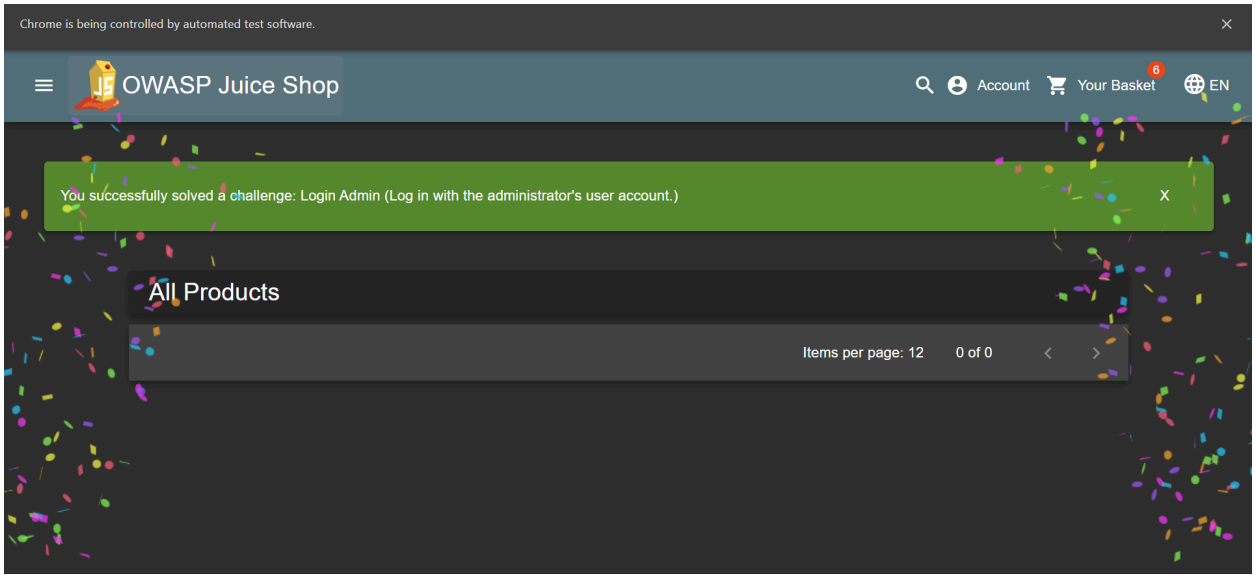


Figure 2: XSS Alert Execution

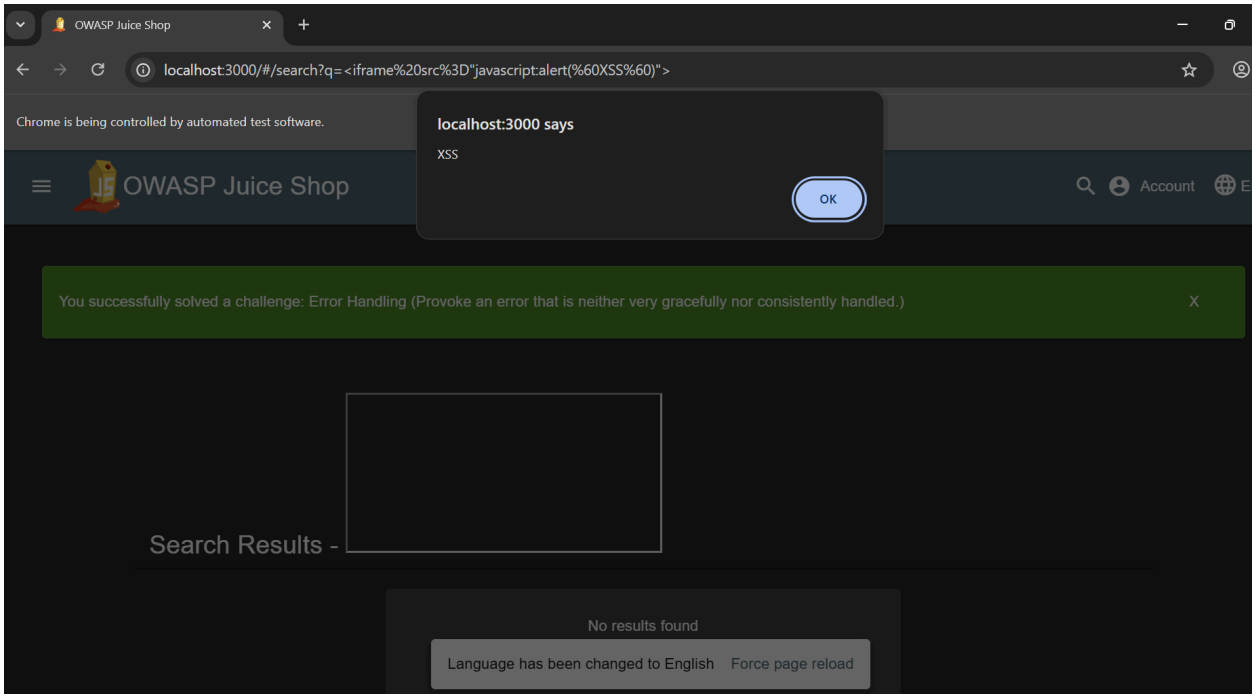
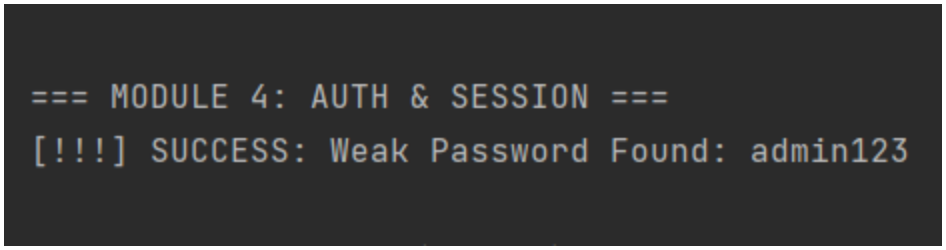


Figure 3: Brute Force Success



]

Figure 4: IDOR Data Leak

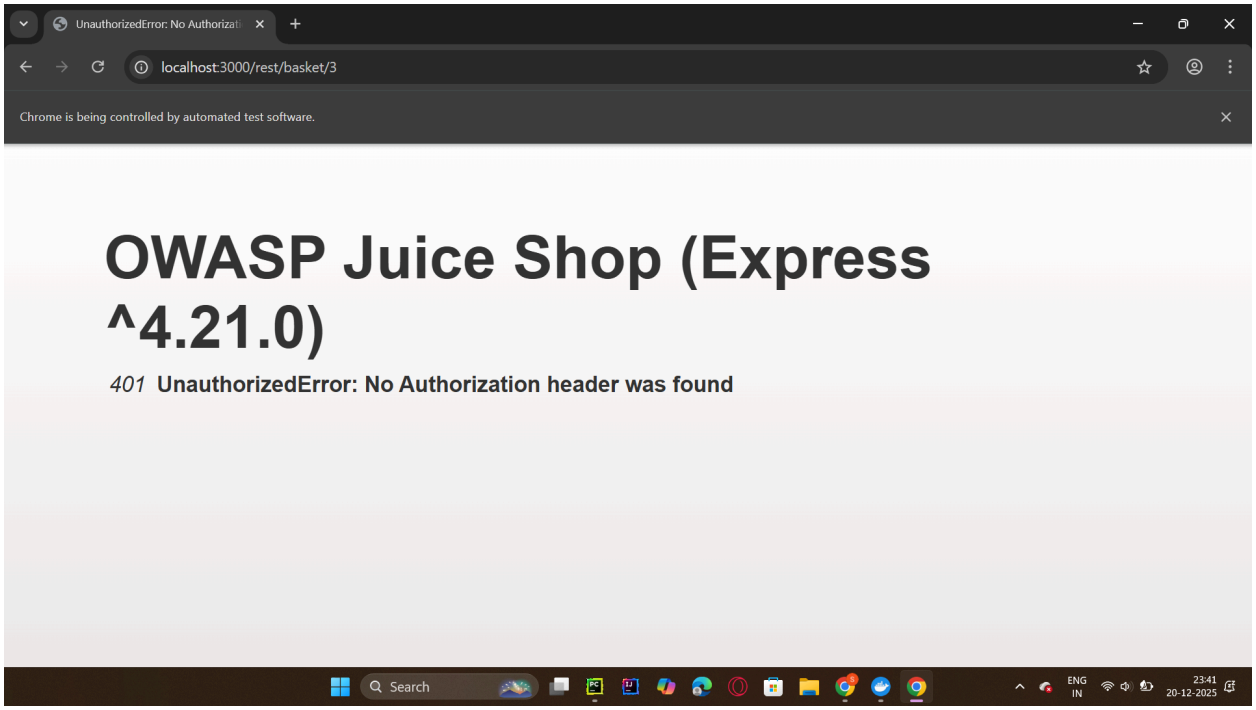


Figure 5: Final HTML Security Report

WebScanPro Vulnerability Report

Target: http://localhost:3000

Scan Date: 2025-12-28 19:35:41

Total Vulnerabilities Found: 3

Detailed Findings

Type	Severity	Location	Description	Suggested Mitigation
SQL Injection	Critical	/#/login	Auth bypass via SQLi in email field.	Use parameterized queries (Prepared Statements).
Reflected XSS	High	/#/search	Arbitrary JS execution via search parameter.	Sanitize user input and implement Content Security Policy (CSP).
Weak Credentials	High	/#/login	Admin password cracked: admin123	Enforce strong password complexity policies.

6. Conclusion

The WebScanPro project successfully met all objectives outlined in the internship roadmap. By leveraging modern automation techniques, the tool proved capable of identifying critical OWASP Top 10 vulnerabilities in a realistic target environment. The transition from simple scripting to a robust, modular testing framework demonstrates a comprehensive understanding of both software development and offensive security principles.

The final deliverable is a functional, automated security scanner that produces professional-grade audit reports, ready for potential integration into DevSecOps pipelines.