Full-Spectrum Web App Hardening: Simulating Cyber Attacks and Training an AI Model for Self-Defense

Project Authors: MOHANAD GABER (2305391), AHMED MOHAMED (2305136)

Course: Secure Software Development – Cybersecurity Program **Faculty:** Faculty of Computers and Data Science **Semester:** Fall 2025

1. Project Description

This project demonstrates a complete security lifecycle for a web application, moving from vulnerability discovery to code-level hardening and evidence-based validation. The target is a deliberately vulnerable Node.js/Express application.

The project's methodology is based on a full-spectrum security assessment:

1. **Dynamic Application Security Testing (DAST):** Used **OWASP ZAP** and manual testing to find vulnerabilities in the running application.
2. **Static Application Security Testing (SAST):** Employed **Semgrep** with custom rules to pinpoint the insecure code patterns responsible for the vulnerabilities.
3. **Fix and Validation:** Applied secure coding practices to remediate the flaws, followed by re-testing with both DAST and SAST to prove the fixes were effective.
4. **Log Analysis:** Captured network traffic and analyzed it in **Splunk Enterprise** to demonstrate how real-world attacks are detected in a security monitoring environment.

2. Technologies Used

Category	Technology	Purpose
Backend	Node.js, Express.js	Core application runtime and web framework.
Security Testing	OWASP ZAP, Postman, `curl`	DAST (Automated and Manual) for vulnerability discovery.
Code Analysis	Semgrep (CLI)	SAST for code-level pattern matching and custom rule creation.

Log Analysis	Splunk Enterprise, `tcpdump`	SIEM for traffic ingestion and attack detection via SPL queries.
Security Libraries	`xss` , `bcrypt`	Used for server-side input sanitization and secure password hashing.

3. Project Structure

The repository structure is organized to include the application code, security configurations, and evidence:

Plain Text

```
.
├── src/                    # Application source code (controllers, models,
routes)
├── semgrep-rules/          # Custom Semgrep rules (YAML files)
├── traffic-capture/        # Network traffic captures (.pcap files)
├── .env.example            # Template for environment variables
├── package.json            # Node.js dependencies and scripts
└── README.md               # This document
```

- `src/` : Contains the vulnerable and subsequently hardened application code.
- `semgrep-rules/` : Holds the **five custom Semgrep rules** developed to detect project-specific insecure patterns (e.g., Command Injection, SSRF, Broken Authorization).

4. Setup & Installation

This guide assumes you have **Node.js, npm, and git** installed.

Prerequisites

1. **Clone the Repository:**

   Bash

   ```
   git clone https://github.com/SirAppSec/vuln-node.js-express.js-app.git
   cd vuln-node.js-express.js-app
   ```

2. **Install Dependencies:**

```
Bash

npm install
```

3. **Set Environment Variables:** Create a `.env` file based on the `.env.example` and set required variables.
4. **Install Semgrep:**

```
Bash

pip install semgrep
```

Running the Application

```
Bash

npm run dev
# Application will be running on http://localhost:5000
```

5. Vulnerabilities Overview

The project successfully identified and addressed **8 distinct vulnerabilities (V1-V8)** covering **5 OWASP Top 10 categories** in the initial codebase.

ID	Type	OWASP Category	Affected Endpoint	Short Impact Description
V1	Reflected XSS	A03: Injection	`GET /?message=`	Executes arbitrary JavaScript in the victim's browser via unsanitized URL parameter.
V2	IDOR	A01: Broken Access Control	`GET /profile?id={id}`	Unauthorized access to other users' profile data by manipulating the ID parameter.

V3	Broken Function Auth	A01: Broken Access Control	`DELETE /v1/user/{user_id}`	Any authenticated user can delete any other user's account.
V4	SSRF	A10: Server-Side Request Forgery	`GET /v1/test?url=`	Server makes requests to attacker-controlled internal/external URLs.
V5	Excessive Session Lifetime	A05: Security Misconfiguration	Session Cookie	Increases risk of session hijacking due to a 120-day session expiration.
V6	RCE	A03: Injection (Command Injection)	`GET /v1/status/{brand}`	Enables remote execution of OS commands via unsanitized input.
V7	User Enumeration	A07: Identification and Authentication Failures	`POST /v1/user/token`	Different error messages allow attackers to enumerate valid user emails.
V8	Weak Password Hashing	A02: Cryptographic Failures	User Authentication	Passwords stored using unsalted MD5, making them vulnerable to cracking.

6. DAST Testing (OWASP ZAP)

DAST was performed to confirm the existence of the vulnerabilities and create Proof-of-Concepts (PoCs).

Example PoCs (Before Fix)

Vulnerability	Example Payload (`curl`)	Confirmation Method
V1 (XSS)	`curl "http://localhost:5000/?message=<script>alert(1)</script>"`	Script was reflected in the HTML response, confirming XSS.
V2 (IDOR)	`curl "http://localhost:5000/profile?id=3" -H "Cookie: sessionID=VALID_SESSION"`	Server returned profile data for ID 3, despite the session belonging to a different user.
V6 (RCE)	`curl "http://localhost:5000/v1/status/google.com;id"`	Server response included the output of the `id` command, confirming RCE.

7. SAST Testing (Semgrep)

Semgrep was used to perform static analysis, detecting **16 issues** with default rulesets and **7 findings** with custom rules before the fix.

Custom Rules Explanation

Five custom Semgrep rules were developed and placed in the `semgrep-rules/` directory to target specific insecure patterns that were not covered by default rules, including:

- **Command Injection:** Rule to detect unsanitized input passed to system execution functions.
- **SSRF:** Rule to flag external URL input used in server-side requests.
- **Broken Function-Level Authorization:** Rule to identify routes missing proper role-based checks.

Mapping Vulnerabilities to Code and Rules

The SAST phase successfully mapped the DAST findings to their code-level root causes:

ID	Vulnerability	Semgrep Rule (Built-in or Custom)
V1	Reflected XSS	Built-in rule for unvalidated user input.

V6	RCE	Custom rule for Command Injection.
V8	Weak Password Hashing	Built-in rule for weak cryptographic hashing (MD5).
V2, V3, V4	IDOR, Broken Auth, SSRF	Custom rules were essential to detect these business logic and configuration flaws.

8. Fixes and Validation

All 8 vulnerabilities were successfully mitigated, and the effectiveness of the fixes was validated by re-running both DAST and SAST.

Summary of Fixes

ID	Vulnerability	Fix Applied	Verification (Post-Fix)
V1 (XSS)	Reflected XSS	Server-side input sanitization using the `xss` library.	Injected scripts are rendered as plain text; no JavaScript execution occurs.
V2 (IDOR)	IDOR	Added authorization checks to ensure users can only access their own resources.	Accessing another user’s ID returns **403 Forbidden**.
V3 (Auth)	Broken Function Auth	Enforced Role-Based Access Control (RBAC) requiring an admin role for sensitive endpoints.	Non-admin users receive **403 Forbidden** on admin actions.
V4 (SSRF)	SSRF	Implemented a strict allow-list for URLs and removed dynamic template rendering with untrusted input.	Template injection payloads no longer work; external requests are blocked.
V5 (Session)	Excessive Session Lifetime	Reduced session lifetime and enforced secure	Session cookies expire after a

		cookie attributes (`HttpOnly` , `SameSite`).	reasonable, configured timeout.
V6 (RCE)	RCE	Removed paths for system-level execution functions and implemented strict allow-lists for input.	Command injection payloads (e.g., `;id`) no longer execute.
V7 (Enum)	User Enumeration	Implemented generic error messages for all authentication failures.	Login failures always return the same error message, preventing enumeration.
V8 (Hash)	Weak Password Hashing	Switched to **bcrypt** with a secure salt factor for all password storage.	Stored passwords appear as secure bcrypt hashes (`$2b$`) in the database.

Post-Fix SAST Validation

After applying the code fixes, Semgrep was re-run with both the default and custom rulesets. The result was **0 findings**, confirming that all previously identified insecure patterns were successfully removed or mitigated from the source code.

9. Splunk Log Analysis

Network traffic captured during the DAST phase was ingested into the **`vulnapp_index`** in Splunk Enterprise to demonstrate attack detection capabilities.

SPL Queries for Attack Detection

The following Search Processing Language (SPL) queries were used to successfully detect evidence of the simulated attacks within the logs:

Vulnerability	SPL Query	Validation
V1 (XSS)	`index=vulnapp_index ("<script>" OR "alert(" OR "onerror=")`	Matched HTTP requests containing JavaScript payloads.

V2 (IDOR)	`index=vulnapp_index "/profile? id="`	Detected requests attempting to access user profiles by directly modifying the `id` parameter.
V4 (SSRF)	`index=vulnapp_index "/v1/test" ("127.0.0.1" OR "localhost" OR "169.254")`	Identified requests attempting to force the server to access internal IP addresses.
V6 (RCE)	`index=vulnapp_index "/v1/status" ("id" OR "whoami" OR "cat /etc/passwd")`	Matched suspicious requests targeting command execution endpoints with command-like input.
V7 (Enum)	`` `index=vulnapp_index "/v1/user/login" ``	stats count by email

10. Deliverables

The final deliverables for this project include:

- **Source Code:** The final, hardened Node.js/Express application code.
- **Semgrep Rules:** The `semgrep-rules/` directory containing the five custom YAML rules.
- **PCAP Files:** Network traffic captures (`.pcap`) recorded during the DAST phase.
- **Final Report PDF:** The `Security_Project_Report.pdf` detailing the findings, fixes, and evidence.
- **Video Demonstration:** A short video demonstrating an exploit before the fix, the code fix, and the exploit failing after the fix.