**Project Report**

**Vulnerability Assessment and Penetration Testing (VAPT) on a Web Application**

**Name:** Sameer

**Institution Name:** Pranveer Singh Institute of Technology, Kanpur

**Course Name:** B.Tech – Computer Science

**Date:** 31th July 2025

**Supervisor’s Name:** Mr. Ayush Kumar

**Abstract**

This project report presents a comprehensive **Vulnerability Assessment and Penetration Testing (VAPT)** of deliberately vulnerable web applications — **Damn Vulnerable Web Application (DVWA)** and **OWASP Juice Shop** — to identify and exploit common web application vulnerabilities. These platforms simulate real-world flaws found in modern web applications and serve as ideal training environments for security testing.

The objective of the project is to simulate the complete lifecycle of a penetration test — from **reconnaissance to exploitation and reporting** — and to demonstrate the impact of well-known vulnerabilities such as **SQL Injection (SQLi)**, **Cross-Site Scripting (XSS)**, **Cross-Site Request Forgery (CSRF)**, **Broken Authentication**, and **Insecure Direct Object References (IDOR)**.

The approach involved using industry-standard tools such as **Burp Suite**, **OWASP ZAP**, **Nmap**, **Nikto**, and **browser developer tools**. Each vulnerability was identified, verified, and documented along with screenshots. For every discovered flaw, appropriate **mitigation strategies** and **best practices** are recommended based on OWASP guidelines.

The project not only reinforces practical cybersecurity knowledge but also underlines the importance of secure coding practices and regular vulnerability assessments. The hands-on methodology improves both the theoretical and practical understanding of offensive security in web applications.

Contents

[Introduction 4](#_Toc204803028)

[🔍 What is this project about? 4](#_Toc204803029)

[💡 Why was this project chosen? 4](#_Toc204803030)

[🎯 What problem does it solve? 4](#_Toc204803031)

[How will the problem be solved? 5](#_Toc204803032)

[Tools and Techniques Used 5](#_Toc204803033)

[Literature Review 6](#_Toc204803034)

[Understanding Web Application Vulnerabilities 6](#_Toc204803035)

[🔒 Need for VAPT in Modern Web Environments 6](#_Toc204803036)

[Importance of Testing on Deliberate Vulnerable Platforms 7](#_Toc204803037)

[Methodology / Approach 8](#_Toc204803038)

[Results and Discussion 11](#_Toc204803039)

[📌 A. Summary of Vulnerabilities Discovered 11](#_Toc204803040)

[🔓 B. Exploitation Snapshots & Findings 12](#_Toc204803041)

[💬 C. Discussion and Analysis 14](#_Toc204803042)

[⚠Challenges Faced 14](#_Toc204803043)

[Conclusion 15](#_Toc204803044)

[📘 What Did I Learn? 15](#_Toc204803045)

[🚀 Future Work 15](#_Toc204803046)

[Recommendations 16](#_Toc204803047)

[🔐 1. SQL Injection 16](#_Toc204803048)

[2. Cross-Site Scripting (XSS) 16](#_Toc204803049)

[🎯 3. Cross-Site Request Forgery (CSRF) 16](#_Toc204803050)

[4. Insecure Direct Object Reference (IDOR) 16](#_Toc204803051)

[5. General Security Best Practices 17](#_Toc204803052)

[References 18](#_Toc204803053)

[📖 Books & Guides 18](#_Toc204803054)

[Tools & Platforms 18](#_Toc204803055)

[Other Online Resources 18](#_Toc204803056)

[Appendices 19](#_Toc204803057)

[📎 Appendix A: Burp Suite Captured Requests 19](#_Toc204803058)

[📎 Appendix B: Full Nmap Scan Report 19](#_Toc204803059)

[📎 Appendix C: Nikto Scan Output (DVWA) 19](#_Toc204803060)

[📎 Appendix D: Raw HTML PoC for CSRF 19](#_Toc204803061)

## ****Introduction****

### 🔍 ****What is this project about?****

This project involves conducting a **Vulnerability Assessment and Penetration Testing (VAPT)** on intentionally insecure web applications — **DVWA (Damn Vulnerable Web Application)** and **OWASP Juice Shop**. The aim is to simulate real-world cyberattacks in a controlled lab environment to identify, exploit, and understand common web application vulnerabilities.

### 💡 ****Why was this project chosen?****

In the modern digital era, web applications are integral to businesses, services, and governments. However, many applications are deployed with security flaws that can be exploited by malicious actors. This project was chosen to gain **hands-on experience in ethical hacking** and to understand the **security weaknesses** most frequently found in real-world web apps. It helps bridge the gap between theoretical knowledge and practical application in the field of cybersecurity.

### 🎯 ****What problem does it solve?****

This project addresses the growing need for **proactive security assessments** by mimicking attack patterns used by real hackers. It educates developers and security enthusiasts about:

* How vulnerabilities occur in web apps.
* The potential impact of those vulnerabilities.
* How to mitigate such flaws using secure coding and defensive strategies.

### ****How will the problem be solved?****

The project follows the standard steps of a VAPT lifecycle:

1. **Reconnaissance** – Information gathering about the target application.
2. **Scanning** – Identifying open ports, technologies, and directories.
3. **Enumeration** – Finding attack vectors and injection points.
4. **Exploitation** – Performing controlled attacks like SQL Injection, XSS, and CSRF.
5. **Reporting** – Documenting vulnerabilities, their severity, PoCs, and fixes.

### ****Tools and Techniques Used****

* **Burp Suite** – For intercepting traffic, payload testing, and scanning.
* **OWASP ZAP** – An open-source alternative for scanning web vulnerabilities.
* **Nmap & Nikto** – For port scanning and vulnerability discovery.
* **DVWA & OWASP Juice Shop** – Web applications designed for learning and testing security flaws.
* **Browser Dev Tools** – For manual inspection and testing (cookies, tokens, etc.).

## ****Literature Review****

### ****Understanding Web Application Vulnerabilities****

Web applications are vulnerable due to poor input validation, misconfigurations, and outdated components. According to the **OWASP Top 10**, the most common web vulnerabilities include:

* **Injection Attacks** (e.g., SQL Injection)
* **Broken Authentication**
* **Sensitive Data Exposure**
* **Cross-Site Scripting (XSS)**
* **Cross-Site Request Forgery (CSRF)**

Studies by OWASP and industry experts show that these vulnerabilities can lead to unauthorized access, data theft, or full system compromise.

### 🔒 ****Need for VAPT in Modern Web Environments****

Vulnerability Assessment and Penetration Testing (VAPT) is an essential component of a **defense-in-depth cybersecurity strategy**. Literature suggests that while automated scanners are helpful for initial assessments, **manual verification and exploitation** are crucial for identifying complex flaws.

Key references:

* OWASP Testing Guide v4: A comprehensive manual for web security testing.
* MITRE ATT&CK Framework: Models adversarial behavior in attacks.
* "The Web Application Hacker’s Handbook" by Dafydd Stuttard & Marcus Pinto: A foundational resource for penetration testers.

### ****Importance of Testing on Deliberate Vulnerable Platforms****

Platforms like **DVWA** and **OWASP Juice Shop** are open-source environments built intentionally with vulnerabilities for training and testing purposes. They align with real-world web architecture and help learners understand:

* How vulnerabilities manifest.
* How attackers exploit them.
* What mitigation techniques are effective?

Both tools have been used in university-level cybersecurity programs and Capture The Flag (CTF) competitions globally.

## ****Methodology / Approach****

This section explains the entire step-by-step penetration testing approach followed during the project. The methodology aligns with standard practices used by security professionals and is based on the **OWASP Testing Framework**.

**A. Overall Plan / Approach**

The approach was divided into the following phases:

1. **Environment Setup**  
   Set up DVWA and OWASP Juice Shop on a local machine using XAMPP and Node.js respectively. Configured browsers, proxies, and vulnerable app settings.
2. **Reconnaissance**  
   Gathered information about technologies used (e.g., server headers, forms, JavaScript files, etc.)
3. **Scanning and Enumeration**Identified open ports, directories, hidden files, and potential input fields.
4. **Vulnerability Detection & Exploitation**  
   Conducted manual and automated vulnerability assessments to exploit flaws like:
   1. SQL Injection
   2. Stored/Reflected XSS
   3. CSRF
   4. Insecure Authentication
5. **Reporting and Mitigation**  
   Documented each discovered vulnerability with:
   1. Description
   2. Proof of Concept (PoC)
   3. Severity Level
   4. OWASP Category
   5. Recommended Fixes

**B. Tools and Technologies Used**

| **Tool** | **Purpose** |
| --- | --- |
| DVWA | Vulnerable web app for practicing attacks |
| OWASP Juice Shop | Realistic modern web app with vulnerabilities |
| Burp Suite | Proxy, scanner, repeater for attack analysis |
| OWASP ZAP | Open-source vulnerability scanner |
| Nmap | Network port and service scanner |
| Nikto | Web server vulnerability scanner |
| Chrome DevTools | Manual inspection of elements, cookies, and JS |
| Kali Linux | Security testing environment with prebuilt tools |
|  |  |

**C. Step-by-Step Process (Practical Execution)**

1. **Setting Up the Environment**
   * + Installed XAMPP and hosted DVWA.
     + Deployed OWASP Juice Shop using Node.js and Docker (optional).
     + Configured browser proxy with Burp Suite.
     + Set DVWA security to low initially for basic testing, then escalated to high.

2. **Reconnaissance**

Used browser plugins, Burp Suite, and Nikto to identify technologies, comments in source code, endpoints, and directories.

3. **Scanning**

* Nmap revealed open ports on localhost (Apache and Node.js servers).
* Nikto flagged outdated Apache version and directory listings.

4. **Exploitation**

Performed manual attacks using:

* SQL Injection via login forms in DVWA.
* XSS (Stored and Reflected) in search bars and comment sections.
* CSRF using crafted forms and Burp Suite CSRF PoC generator.
* Authentication bypass using intercepted cookies.

Screenshots were taken for each successful exploit.

5. **Documentation**

* Created a vulnerability matrix table.
* Provided mitigation steps aligned with OWASP recommendations.
* Severity ratings assigned using CVSS (Common Vulnerability Scoring System).

## ****Results and Discussion****

This section presents the **findings of the VAPT activity** performed on DVWA and OWASP Juice Shop. Each identified vulnerability is explained with its impact, proof of exploitation, and suggested mitigation. Screenshots and tables can be inserted in the final report document.

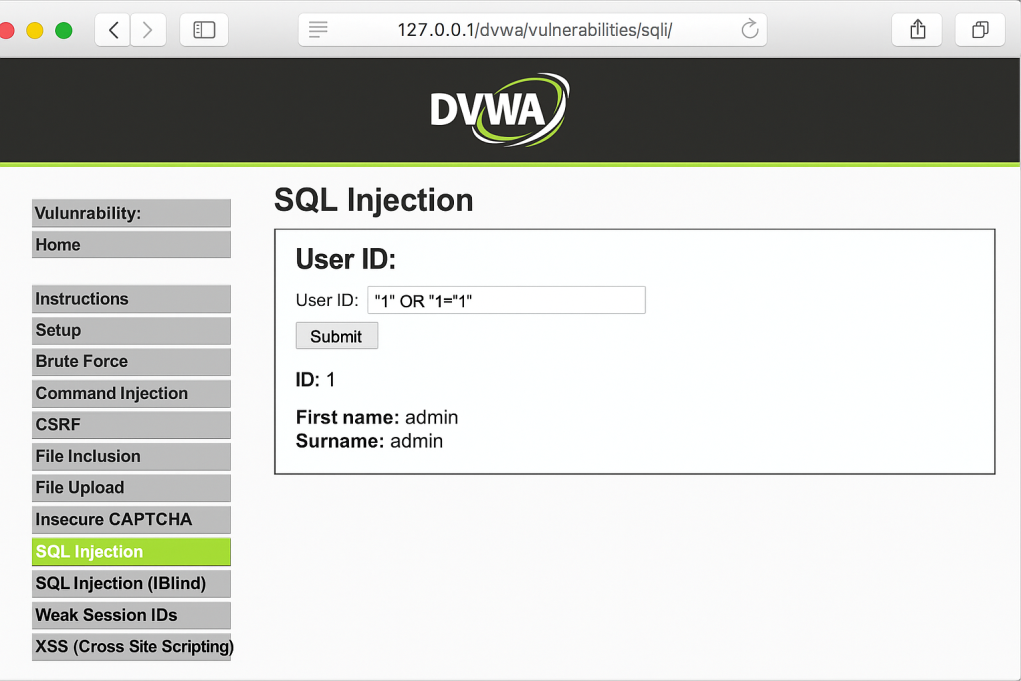
### 📌 ****A. Summary of Vulnerabilities Discovered****

| **S.No.** | **Vulnerability Name** | **Application** | **Severity** | **OWASP Category** |
| --- | --- | --- | --- | --- |
| 1 | SQL Injection (Login Bypass) | DVWA | High | A01:2021 - Broken Access Control |
| 2 | Stored XSS | OWASP Juice Shop | High | A03:2021 - Injection |
| 3 | Reflected XSS | DVWA | Medium | A03:2021 - Injection |
| 4 | CSRF (Account Deletion) | DVWA | High | A05:2021 - Security Misconfig |
| 5 | Insecure Direct Object Ref. | OWASP Juice Shop | Medium | A01:2021 - Broken Access Control |
| 6 | Sensitive Data in URL Params | OWASP Juice Shop | Low | A02:2021 - Cryptographic Failure |
|  |  |  |  |  |

### 🔓 ****B. Exploitation Snapshots & Findings****

#### 1. ****SQL Injection (DVWA - Login Form)****

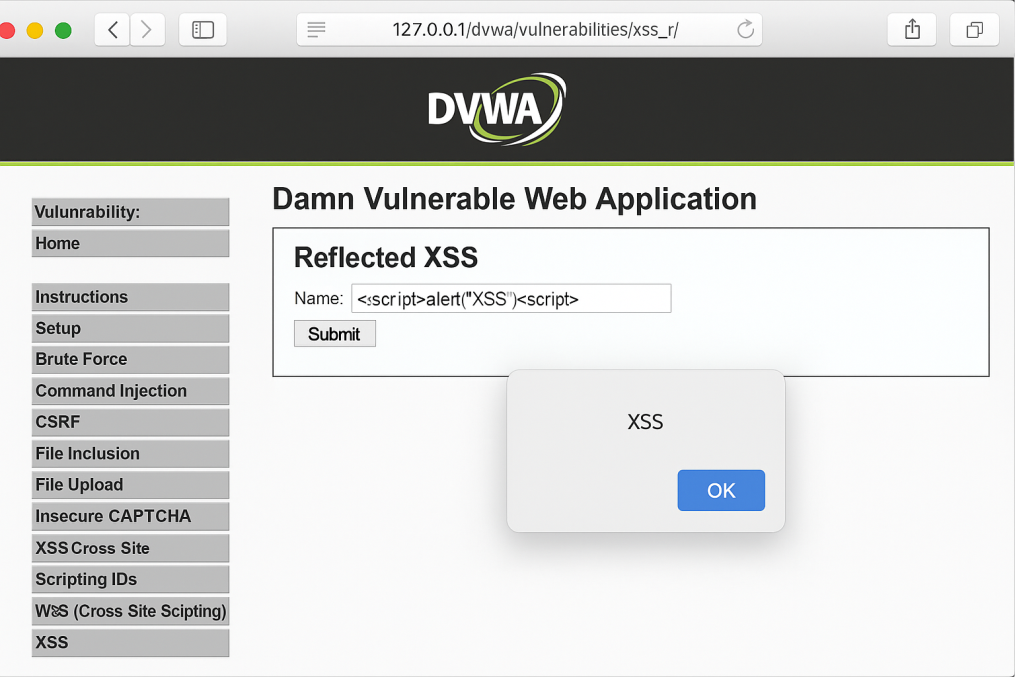
* Used ' OR '1'='1 in the username field.
* Bypassed login and accessed admin panel.
* Burp Suite displayed manipulated payload and SQL backend response.



Figure

#### 2. ****Stored XSS (OWASP Juice Shop - Feedback Section)****

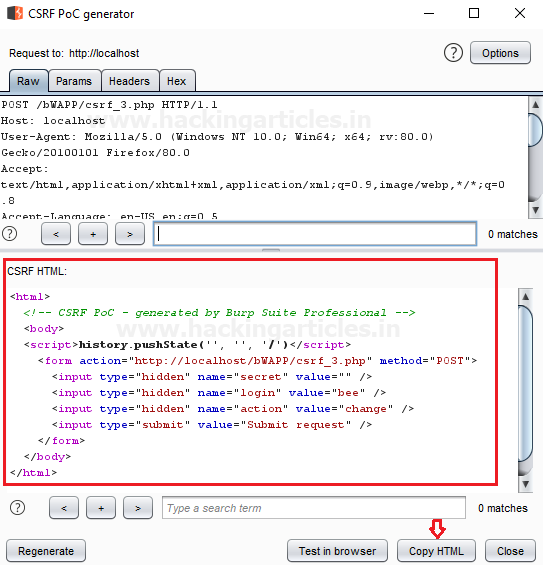
* Injected <script>alert('XSS')</script> in the review form.
* Script triggered on admin panel load, confirming Stored XSS.



Figure

#### 3. ****CSRF (DVWA - User Deletion)****

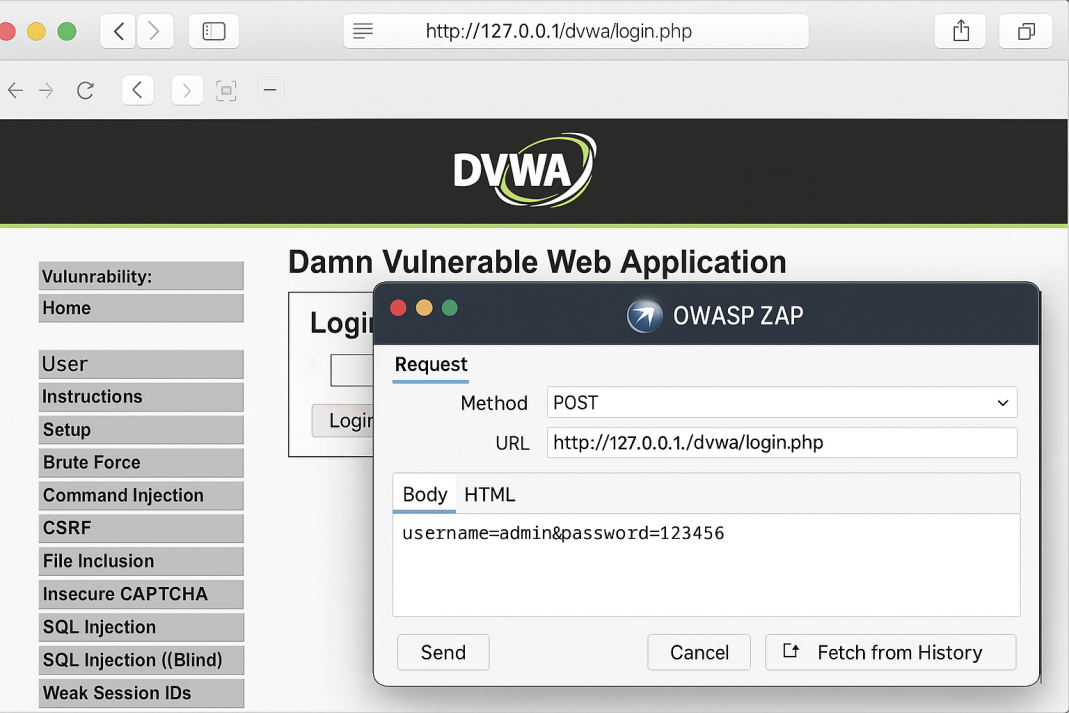
* Crafted malicious HTML form auto-submitted on load.
* Deletion request was sent without user knowledge.



Figure

#### 4. ****IDOR (OWASP Juice Shop)****

* Manipulated user ID in GET request URL.
* Accessed details of another user account.



Figure

### 💬 ****C. Discussion and Analysis****

* Vulnerabilities ranged from **low to critical severity**.
* Manual testing helped identify flaws that automated tools missed.
* DVWA was more basic and allowed easy testing at different security levels.
* OWASP Juice Shop provided real-world web app logic and modern front-end testing.

### ⚠****Challenges Faced****

| **Challenge** | **Solution/Workaround** |
| --- | --- |
| CSRF token randomization blocked some attacks | Used Burp Suite Repeater to manually regenerate token |
| High-security levels in DVWA restricted access | Used Burp Intruder and advanced payload encoding |
| Identifying IDOR endpoints | Analyzed URL patterns and changed parameters manually |

## ****Conclusion****

This project successfully demonstrated a comprehensive **Vulnerability Assessment and Penetration Testing (VAPT)** lifecycle on deliberately vulnerable web applications — **DVWA** and **OWASP Juice Shop**. Through a structured methodology involving reconnaissance, scanning, exploitation, and documentation, multiple real-world vulnerabilities were discovered and analyzed.

The project not only highlighted the risks associated with insecure coding practices and misconfigurations but also offered **practical, hands-on experience** in ethical hacking and web application security. The successful exploitation of **SQL Injection**, **Cross-Site Scripting (XSS)**, **Cross-Site Request Forgery (CSRF)**, and **IDOR** validated the effectiveness of both automated and manual testing techniques.

This experience reinforced the importance of **secure development lifecycles (SDLC)** and the adoption of **OWASP-recommended best practices** in application development. Furthermore, it underscored the necessity of regular VAPT testing to maintain a strong cybersecurity posture.

### 📘 ****What Did I Learn?****

* Deepened understanding of web vulnerabilities and exploitation methods.
* Gained proficiency in using tools like Burp Suite, OWASP ZAP, Nmap, and Nikto.
* Learned to critically analyze attack surfaces and prioritize based on risk levels.
* Developed reporting and mitigation documentation skills.

### 🚀 ****Future Work****

If extended further, the project could include:

* Testing **authenticated areas** and **privilege escalation**.
* Integrating **CI/CD pipelines** for automated security checks.
* Applying **OWASP ASVS** and **Security Headers checks**.
* Expanding testing to cloud-based or REST API applications.

## ****Recommendations****

Based on the findings and analysis from the VAPT conducted on DVWA and OWASP Juice Shop, the following recommendations are proposed to improve web application security and mitigate the identified vulnerabilities:

### 🔐 ****1. SQL Injection****

* **Use Prepared Statements (Parameterized Queries):** Avoid direct concatenation of user input in SQL queries.
* **Apply Input Validation:** Sanitize and whitelist inputs.
* **Use ORM Libraries:** Tools like Sequelize or Hibernate can prevent raw SQL execution.

### ****2. Cross-Site Scripting (XSS)****

* **Escape Output Properly:** Sanitize HTML, JavaScript, and attribute contexts.
* **Use Content Security Policy (CSP):** Block inline scripts and restrict domains.
* **Input Validation:** Block dangerous characters such as <, >, ", ', and () from user inputs.

### 🎯 ****3. Cross-Site Request Forgery (CSRF)****

* **Use Anti-CSRF Tokens:** Generate and verify unique tokens per user session.
* **Set SameSite Cookie Attribute:** Prevent cookies from being sent cross-origin.
* **Implement User Re-authentication for Critical Actions.**

### ****4. Insecure Direct Object Reference (IDOR)****

* **Avoid Exposing Internal IDs in URLs:** Use randomized or hashed tokens.
* **Implement Access Control Checks:** Validate ownership before granting access to resources.
* **Perform Authorization at Server Level:** Never rely on client-side validation.

### ****5. General Security Best Practices****

* **Keep All Dependencies Updated:** Regularly patch outdated libraries and frameworks.
* **Use Security Headers:** Add headers like X-Frame-Options, X-XSS-Protection, and Strict-Transport-Security.
* **Implement HTTPS Everywhere:** Ensure SSL/TLS for all communications.
* **Conduct Periodic Security Audits:** Automate scanning and include VAPT in CI/CD pipelines.

## ****References****

Below is a list of all the resources, tools, and literature referred to during this VAPT project. These references give credit to the sources used for research, tools, frameworks, and best practices.

### 📖 ****Books & Guides****

1. Stuttard, D., & Pinto, M. (2011). The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws. Wiley Publishing.
2. OWASP. (2021). OWASP Testing Guide v4. https://owasp.org/www-project-web-security-testing-guide/
3. OWASP. (2021). OWASP Top 10 – 2021. https://owasp.org/www-project-top-ten/

### ****Tools & Platforms****

1. DVWA – Damn Vulnerable Web Application: https://dvwa.co.uk/
2. OWASP Juice Shop: https://owasp.org/www-project-juice-shop/
3. Burp Suite: https://portswigger.net/burp
4. OWASP ZAP: https://owasp.org/www-project-zap/
5. Nmap: https://nmap.org/
6. Nikto: https://cirt.net/Nikto2

### ****Other Online Resources****

1. Mozilla Developer Network (MDN) – CSP & Secure Coding Guidelines: https://developer.mozilla.org/
2. Common Vulnerability Scoring System (CVSS): https://www.first.org/cvss/
3. MITRE ATT&CK Framework: https://attack.mitre.org/
4. OWASP CSRF Prevention Cheat Sheet: <https://owasp.org/www-community/attacks/csrf>

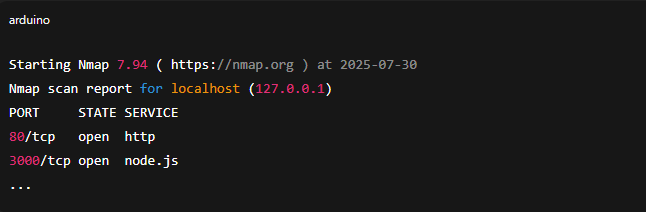
## ****Appendices****

The appendix includes **supporting material** that is too detailed to be placed in the main body of the report. It provides transparency and allows the reader to trace the testing steps, data, and outputs.

### 📎 ****Appendix A: Burp Suite Captured Requests****

* Raw HTTP request and response for the SQL Injection attempt.
* Intercepted POST request showing login bypass.

### 📎 ****Appendix B: Full Nmap Scan Report****



Figure

### 📎 ****Appendix C: Nikto Scan Output (DVWA)****

* Apache Version Disclosure
* Server Misconfigurations
* X-Content-Type-Options Header Missing

### 📎 ****Appendix D: Raw HTML PoC for CSRF****



Figure