Task 1: Web Application Security Testing Using DVWA

Name: Yonatan Melaku

CIN ID: FIT/DEC25/CS5443

Submitted to: Future Intern

Submission Date: 29th Jan, 2026

Contents

Introduction

Web applications are the foundation of many modern digital services including e-commerce platforms, online banking systems, and educational portals. These systems enable businesses and users to perform critical tasks online, but their widespread adoption also makes them attractive targets for cyber attackers, as threat actors constantly seek insecure entry points that can be exploited to steal data, disrupt services, or damage reputation (OWASP, 2025; Ethical Hacking Institute, 2025). The most prevalent vulnerabilities include SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Command Injection, and insecure file uploads, all of which remain among the most exploited weaknesses in real-world web applications because they arise from common coding mistakes and insufficient input validation (Ethical Hacking Institute, 2025; OWASP, 2025).

Learning web application security through attacks on live, production systems is both illegal and unethical; unauthorized exploitation of vulnerabilities can lead to data loss, system outages, and legal consequences. To safely gain practical experience, cybersecurity educators and ethical hackers use intentionally vulnerable applications deployed in isolated lab environments where tests can be performed legally and without risk to real users (Principles of Cyber, 2025). One of the most widely used platforms for this purpose is the Damn Vulnerable Web Application (DVWA).

DVWA is a deliberately insecure PHP/MySQL-based web application designed as a hands-on learning resource that simulates common web security flaws so that learners can understand how vulnerabilities work and how attackers exploit them (EdgeNexus, 2025; Principles of Cyber, 2025). It includes modules that replicate numerous real-world vulnerabilities, making it ideal for beginners and advanced students alike. This project demonstrates practical vulnerability testing using DVWA in a controlled lab environment.

About DVWA

Damn Vulnerable Web Application (DVWA) is an intentionally weak web application that provides a safe environment to practice vulnerability testing and security assessment techniques. Its simple design and clear structure allow learners to explore how common attacks function and how mitigations can be applied.

Key Features

- Contains multiple modules representing common web vulnerabilities used in real-world attacks.
- Offers **adjustable security levels** *Low, Medium, High*, and *Impossible*, that help learners progress from basic exploitation to understanding how hardened defenses work.
- Lightweight and beginner-friendly, making it ideal for educational environments.
- Built around the **OWASP Top 10** vulnerabilities, aligning practical exercises with industry standards (EdgeNexus, 2025; OWASP, 2025).

Vulnerabilities Tested

In this project, the following vulnerabilities were explored and tested through DVWA:

- **SQL Injection (SQLi)** – where malicious input manipulates database queries.
- **Cross-Site Scripting (Reflected & Stored)** – which allows attackers to inject scripts into web pages viewed by others.
- **Command Injection** – executing arbitrary system commands via user input.
- **File Upload Vulnerability** – where insecure upload handling allows malicious files to be stored and executed (Ethical Hacking Institute, 2025; EdgeNexus, 2025).

⚙️ **Environment Setup**

To create the controlled lab environment required for this project, the following system configuration was used:

System Configuration

- **Operating System:** Kali Linux (I used it as a virtual machine) - a security-focused distribution used by professionals to perform penetration testing and security research.
- **Web Server:** Apache- a widely deployed open-source web server.
- **Database:** MySQL- used by DVWA to store application data and replicate real database interactions.
- **Application:** DVWA installed on the server with default configurations for testing.
- **Security Level:** Set to *Low* to clearly observe vulnerabilities in their most exposed form.

Tools Used

- **Web Browser:** for manual interaction with test modules.
- **Burp Suite (Community Edition):** to intercept and analyze web traffic during attack simulations.
- **Terminal:** to execute Linux commands and manage services during setup.

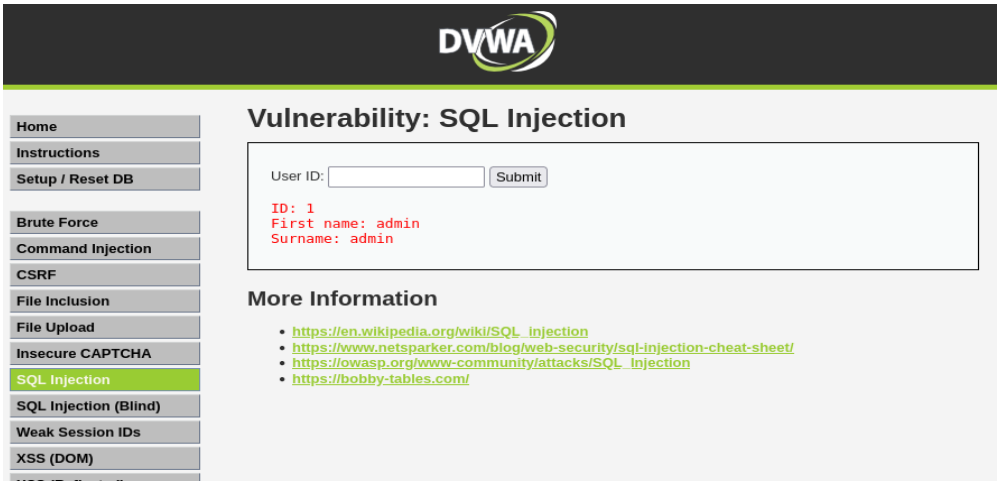
SQL Injection (SQLi)

SQL Injection occurs when user-supplied data is directly embedded into database queries without proper validation or parameterization. Attackers exploit this to manipulate backend SQL statements, which can allow them to view, modify, or delete sensitive information stored in databases — a major security concern in many applications (StackHawk, 2025; Ethical Hacking Institute, 2025).

Testing Steps:

1. Navigated to the **SQL Injection module** within DVWA.

2. Entered a normal user ID (e.g., “1”) to observe the application’s expected behavior and establish a baseline for normal responses.

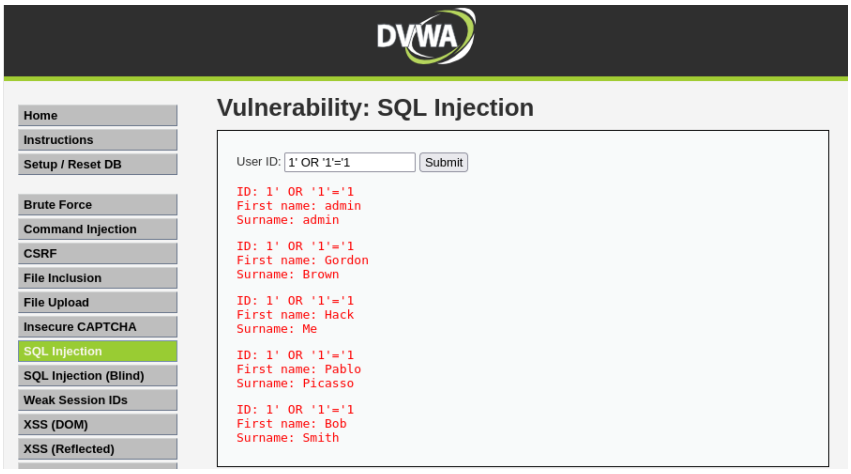


Displays user details (First name, Surname)

Purpose: Confirms the application behaves normally before attack.

Step 3:- Injected the payloads:

1' OR '1'='1



- The application returns multiple user records
- Authentication logic is bypassed

Explanation:

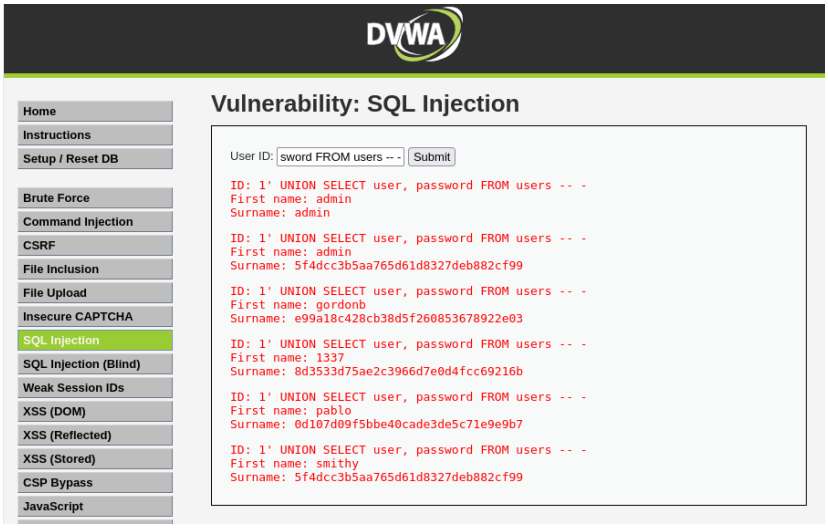
'1'='1 is always true, forcing the SQL query to return all rows.

1' UNION SELECT null, database() -- -

Displays the database name

This proves: Attacker can extract database metadata.

1' UNION SELECT user, password FROM users -- -



- Displays usernames and hashed passwords
- Sensitive credentials exposed due to insecure query handling.

Impact and Mitigation

The attack resulted in the exposure of usernames and hashed passwords, demonstrating how insecure SQL query handling can leak sensitive credentials. Such vulnerabilities allow attackers to gain

unauthorized access to database data, bypass authentication mechanisms, and potentially compromise the entire database if exploited further (OWASP, 2025).

Mitigation Measures:

To prevent SQL Injection attacks, applications should implement prepared statements to separate SQL code from user input, along with proper input sanitization and validation. Additionally, enforcing least-privilege access for database accounts limits the damage even if an attack succeeds (OWASP, 2025; StackHawk, 2025).

Cross-Site Scripting (XSS)

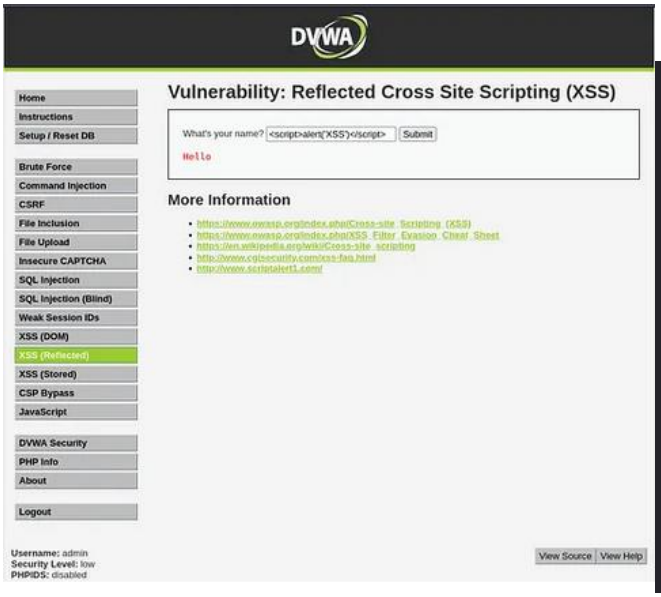
Cross-Site Scripting (XSS) is a web application vulnerability that allows attackers to inject malicious JavaScript code into web pages viewed by other users. When executed in a victim's browser, these scripts can steal session cookies, capture user input, or manipulate page content, leading to serious security and privacy risks (OWASP, n.d.).

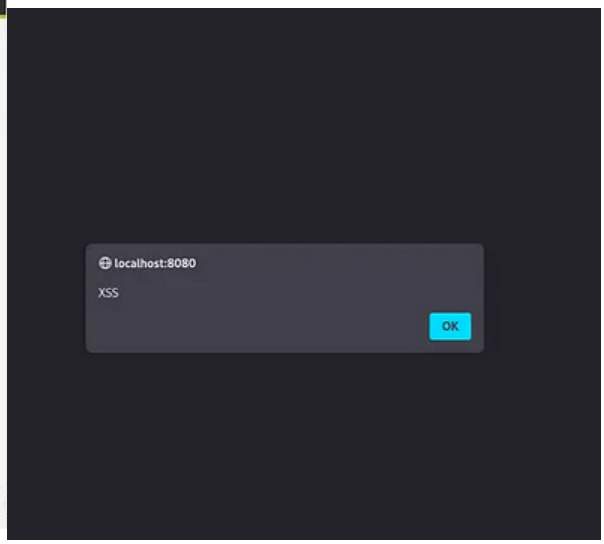
Reflected XSS

Payload used:

```
<script>alert('xss')</script>
```

The script executed immediately, displaying an alert.





Stored XSS

Payload used:

<script>alert('xss')</script>

The script was stored and executed on every page load.

Impact:

Cross-Site Scripting (XSS) can lead to session hijacking, cookie theft, and phishing attacks by allowing malicious scripts to execute in a user's browser. These attacks may compromise user accounts and damage trust in the application (OWASP, n.d.).

Mitigation:

XSS risks can be reduced by encoding output before rendering it in the browser, properly sanitizing and validating user input, and enforcing a Content Security Policy (CSP) to restrict the execution of unauthorized scripts (OWASP, n.d.).

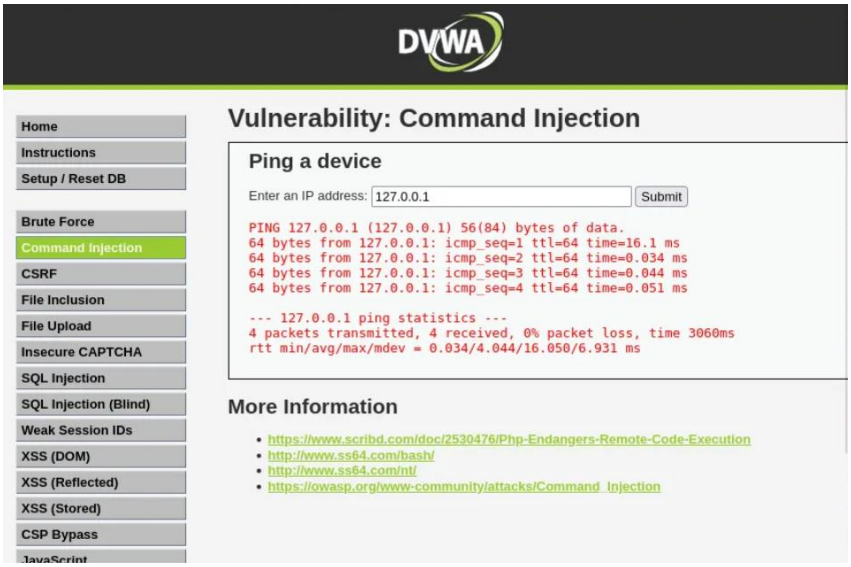
Command Injection

Command Injection is a critical security vulnerability that occurs when untrusted user input is passed directly to system-level commands without proper validation. This allows attackers to execute arbitrary operating system commands on the server, potentially leading to data theft, system compromise, or complete server takeover (OWASP, n.d.).

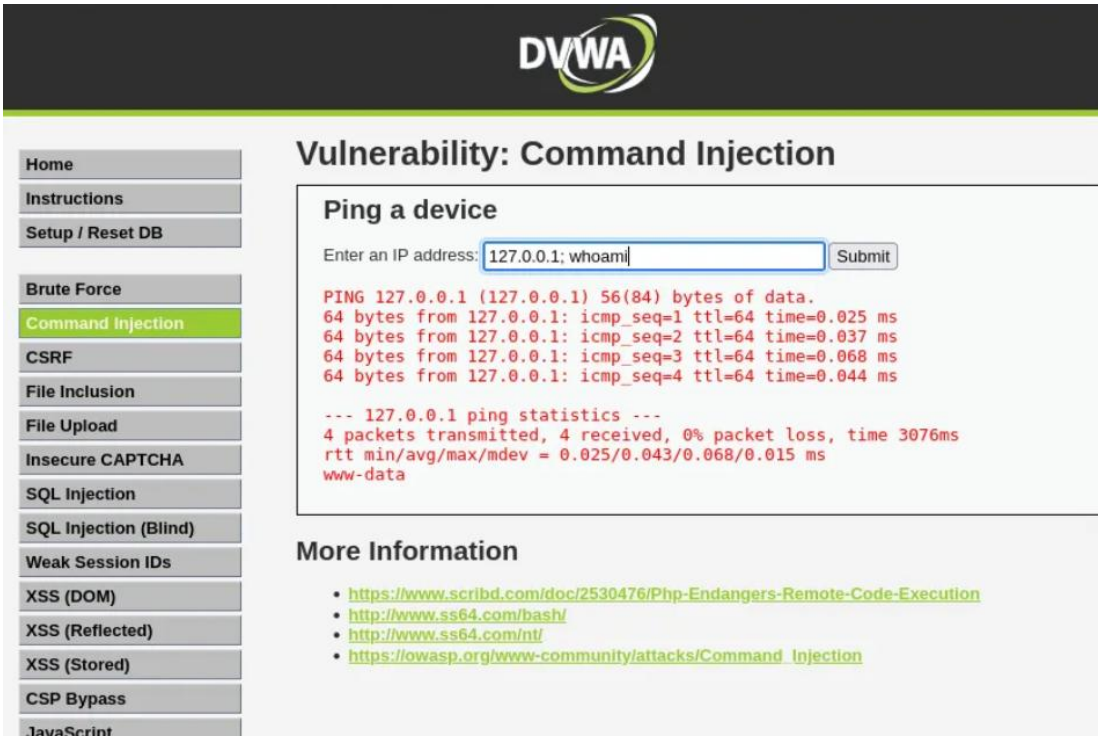
Payload Used

```
127.0.0.1
```

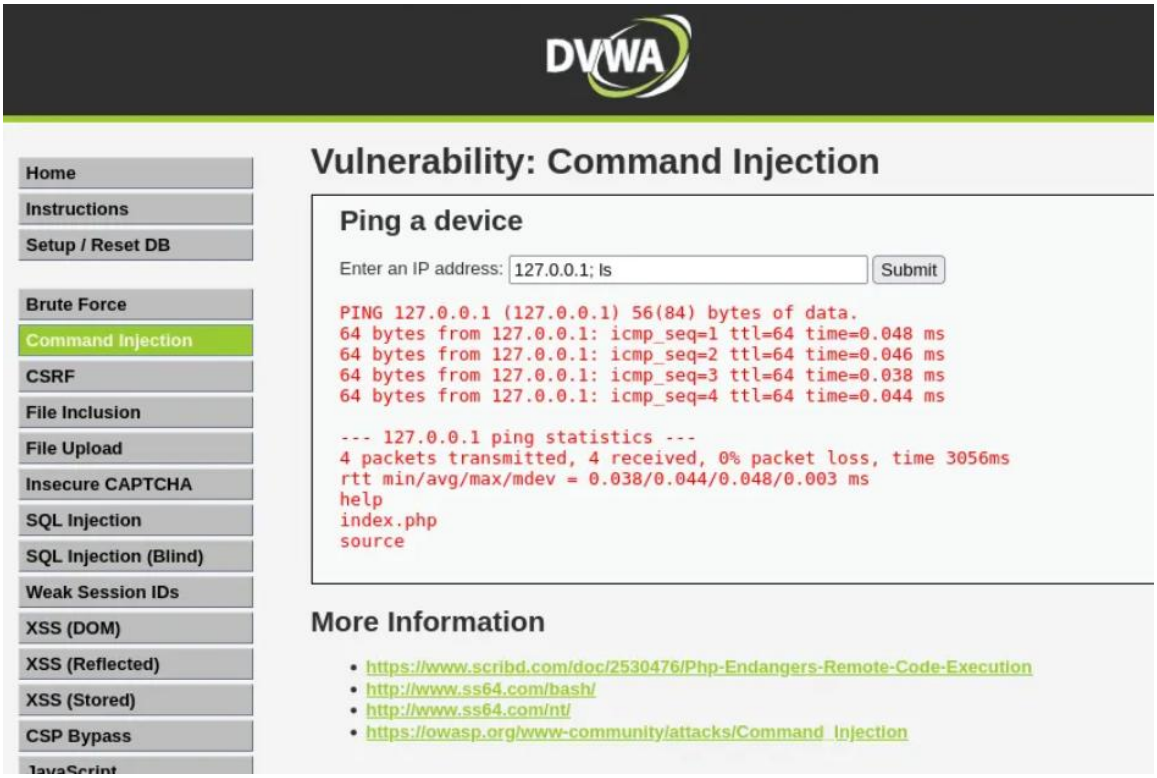
The injected command executed successfully and returned the server user.



127.0.0.1; whoami



127.0.0.1; ls



Impact:

Command Injection can result in arbitrary command execution, disclosure of sensitive system information, and potential full server compromise, allowing attackers to gain significant control over the affected system (OWASP, n.d.).

Mitigation:

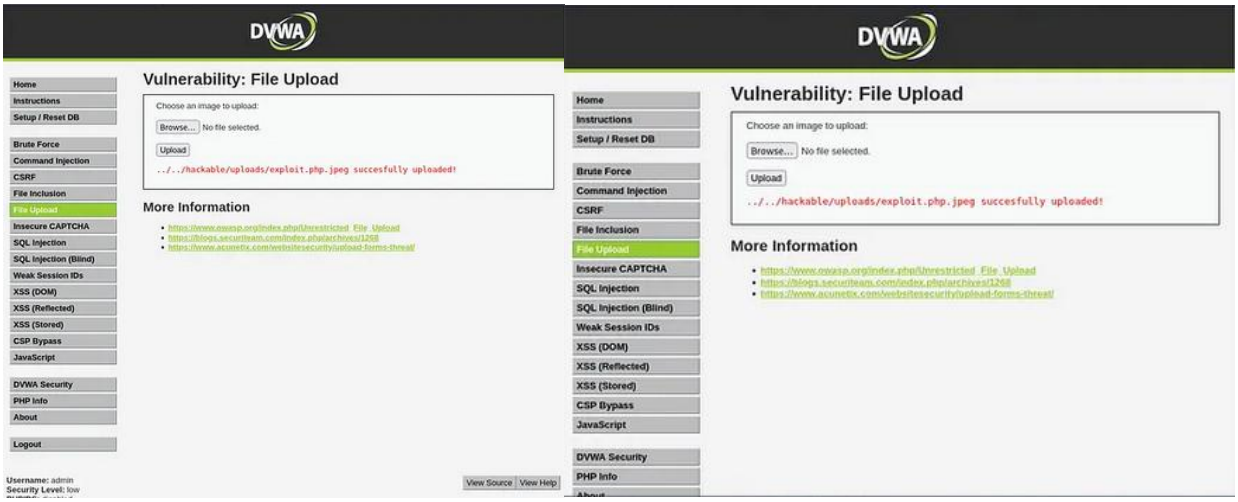
To prevent Command Injection, applications should avoid direct shell command execution, strictly validate and sanitize all user input, and use allow-listed commands to ensure only approved operations can be performed (OWASP, n.d.).

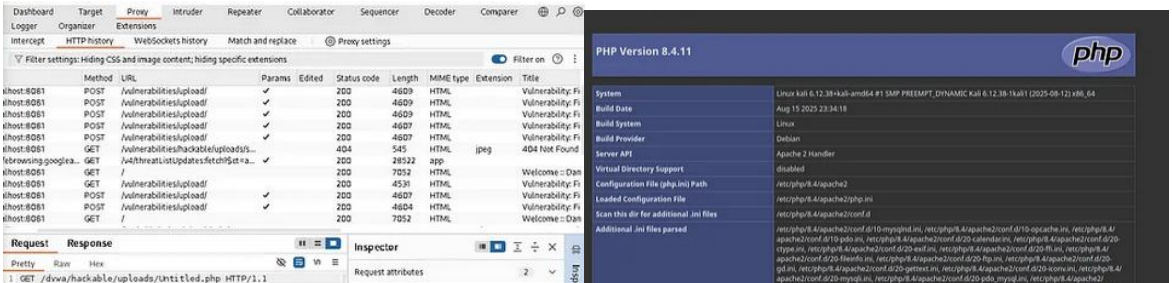
File Upload Vulnerability

Insecure File Upload allows attackers to upload and execute malicious files.

Testing Steps

1. Uploaded a malicious PHP file via the File Upload module
2. Accessed the uploaded file through the browser
3. Achieved remote code execution





Impact:

Insecure file upload vulnerabilities can lead to remote code execution, backdoor installation, and even full server compromise, allowing attackers to gain persistent and unauthorized control over the server (OWASP, n.d.).

Mitigation:

Such risks can be minimized by restricting allowed file extensions, validating MIME types on the server side, and storing uploaded files outside the web root to prevent direct execution (OWASP, n.d.).

Conclusion

This project provided me with hands-on experience in identifying and exploiting common web application vulnerabilities using DVWA. Through this practical assessment, I successfully demonstrated vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Command Injection, and insecure File Upload flaws in a controlled environment.

Overall, this assessment reinforced my understanding of the OWASP Top 10 security risks and significantly enhanced my practical skills in vulnerability assessment and ethical hacking, highlighting the importance of secure coding practices in modern web applications.

References

- EdgeNexus. (2025). *Damn Vulnerable Web Application (DVWA) overview*. Retrieved from https://www.edge-nexus.com/dvwa
- Ethical Hacking Institute. (2025). *Common web application vulnerabilities and attacks*. Retrieved from https://www.ethicalhackinginstitute.com/resources/web-vulnerabilities
- MIT Web Security. (2025). *Introduction to SQL injection and prevention techniques*. Retrieved from https://ocw.mit.edu/courses/web-security
- OWASP. (n.d.). *OWASP Top 10 Web Application Security Risks*. Open Web Application Security Project. Retrieved January 27, 2026, from https://owasp.org/www-project-top-ten/
- Principles of Cyber. (2025). *Ethical hacking and penetration testing: Safe learning practices*. Retrieved from https://www.principlesofcyber.org/ethical-hacking
- StackHawk. (2025). *Understanding SQL injection and its impact on web applications*. Retrieved from https://www.stackhawk.com/resources/sql-injection/