

Web app exploitation - 4geeks

Carlos Duclaud

Objective:

Conduct exploitation of web application DVWA to assess the vulnerability level of the app.

Scope:

This assessment will focus on the exploitation of one specific vulnerability:

- SQL injection

The objective is to leverage these vulnerabilities to obtain access credentials of the target app.

Target Application:

- Application Name: Damn Vulnerable Web Application (DVWA)
- Vulnerability Identified: SQL Injection (via User Input)
- Vulnerable Parameter: User input field (e.g., login, search)

Executive Summary:

During the penetration test of the DVWA application, an SQL Injection vulnerability was identified in a user input field. The injection point was found to be susceptible to basic SQL injection payloads, which allowed an attacker to manipulate SQL queries and potentially retrieve or manipulate sensitive data from the underlying database. The vulnerability was triggered using a common payload:

1' OR '1' = '1

Vulnerability Details:

Vulnerability Type: SQL Injection (Classic)

Impact Level: High

Exploitable: Yes (Remote exploitation possible)

Attack Vector: User input (e.g., login, search forms)

Description:

The vulnerable SQL query used in the application does not properly sanitize user input, allowing an attacker to inject arbitrary SQL statements into the backend database. Specifically, the payload: 1' OR '1' = '1 can manipulate the SQL query to always return true, potentially bypassing authentication or leaking sensitive data.

Example Request:

When inputting the following payload into the login form (or similar input fields):

Username: 1' OR '1' = '1

Password: (anything)

The resulting SQL query would look like this:

```
SELECT * FROM users WHERE username = '1' OR '1' = '1' AND password = '(anything)' ;
```

This query would always evaluate to true ('1' = '1'), potentially bypassing authentication altogether and granting unauthorized access to the application.

Steps to Reproduce:

- Navigate to the login page of the DVWA application (or any form where user input is accepted).
- Enter the following payload in the username field:

1' OR '1' = '1
- Leave the password field blank or input anything (it does not matter).
- Submit the form.

Expected Outcome:

The system should validate the username and password properly and either grant access to authorized users or return an error if authentication fails.

Actual Outcome:

The system successfully logs in the user, bypassing the authentication mechanism due to the injected SQL query returning TRUE.

Risk Assessment:

Likelihood of Exploitation: High

SQL injection vulnerabilities are widely known and easily exploitable, particularly in web applications that fail to sanitize user input.

Impact on the System: High

If exploited, an attacker could bypass authentication, retrieve sensitive data (e.g., usernames, passwords), or execute arbitrary SQL queries on the database.

Confidentiality Impact: High

An attacker could gain unauthorized access to the system, potentially exposing user data and sensitive application details.

Integrity Impact: Moderate to High

Depending on the permissions of the application user, an attacker could modify or delete data in the database.

Availability Impact: Low

No direct impact on availability, though data integrity could be compromised.

Recommendations:

Input Validation & Parameterized Queries:

- Implement input validation techniques to sanitize user input (e.g., escaping special characters).
- Use parameterized queries (prepared statements) for all database interactions to prevent SQL injection attacks.

Web Application Firewall (WAF):

- Deploy a Web Application Firewall to detect and block common SQL injection attacks.

Least Privilege Principle:

- Ensure the database account used by the application has the minimum privileges required to perform necessary actions.

Security Headers:

- Implement HTTP security headers (e.g., X-Content-Type-Options, Content-Security-Policy) to improve overall security.

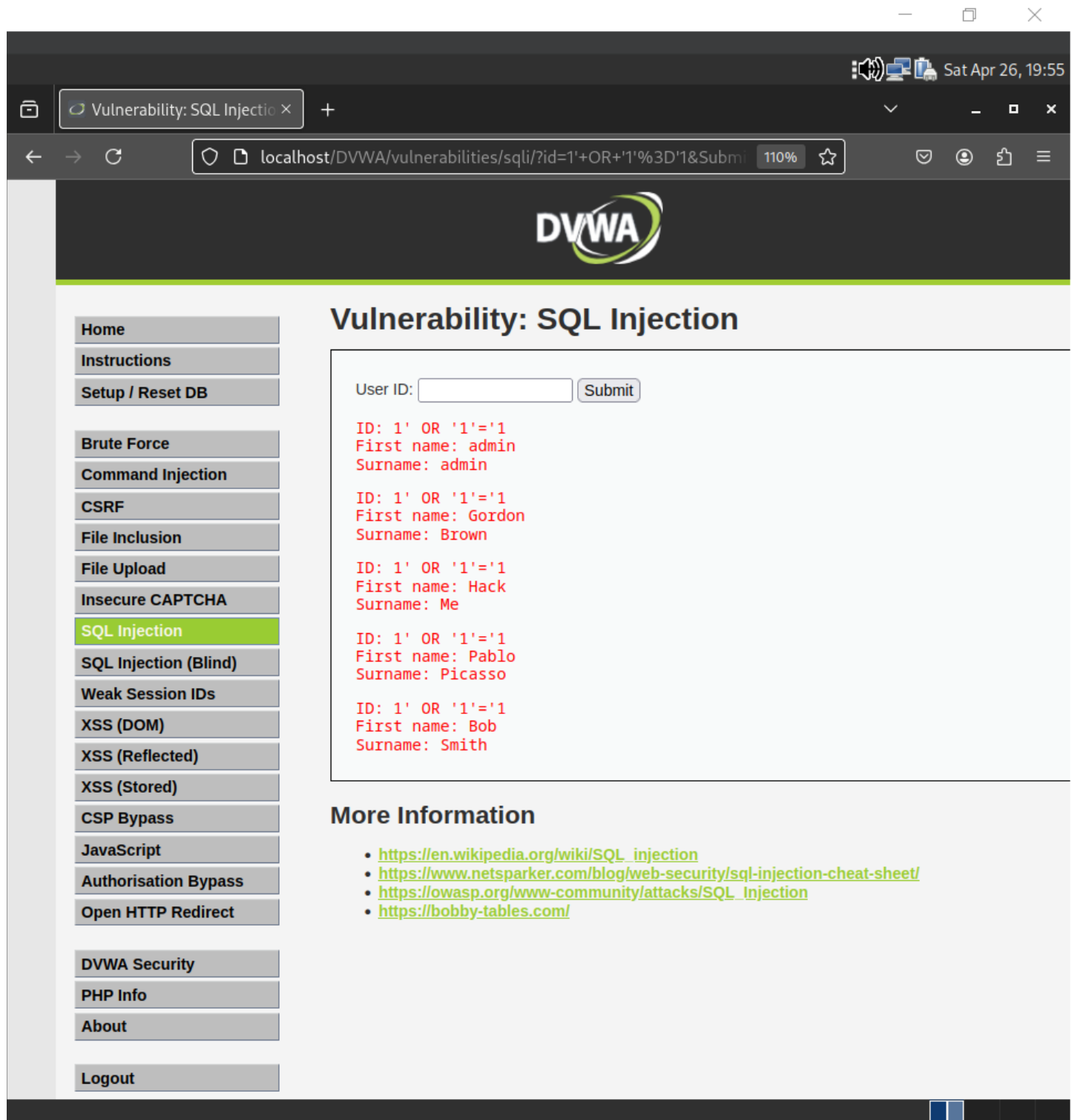
Continuous Testing:

- Regularly perform security testing (including automated vulnerability scans) to identify and mitigate new vulnerabilities.

Conclusion:

The SQL injection vulnerability identified in the DVWA application is a critical security issue that allows an attacker to bypass authentication and potentially access sensitive data. Immediate remediation is required to mitigate the risk of exploitation. By applying the recommended mitigations, the application's security posture can be significantly improved.

Evidence:



The screenshot shows a web browser window displaying the DVWA (Damn Vulnerable Web Application) interface. The browser's address bar shows the URL: `localhost/DVWA/vulnerabilities/sql/?id=1'+OR+'1'%3D'1&Submit`. The page title is "Vulnerability: SQL Injection".

On the left side, there is a navigation menu with various options. The "SQL Injection" option is highlighted in green. Other options include Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, Open HTTP Redirect, DVWA Security, PHP Info, About, and Logout.

The main content area displays the "Vulnerability: SQL Injection" page. It features a form with a "User ID:" label and a "Submit" button. Below the form, there are five lines of red text, each representing a successful SQL injection attack:

```
ID: 1' OR '1'='1
First name: admin
Surname: admin

ID: 1' OR '1'='1
First name: Gordon
Surname: Brown

ID: 1' OR '1'='1
First name: Hack
Surname: Me

ID: 1' OR '1'='1
First name: Pablo
Surname: Picasso

ID: 1' OR '1'='1
First name: Bob
Surname: Smith
```

Below the form, there is a section titled "More Information" with a list of links:

- https://en.wikipedia.org/wiki/SQL_injection
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- https://owasp.org/www-community/attacks/SQL_injection
- <https://bobby-tables.com/>