

# SQL Injection Attack on DVWA

## Objective :-

Demonstrate SQL Injection on DVWA by extracting data and suggesting security fixes.

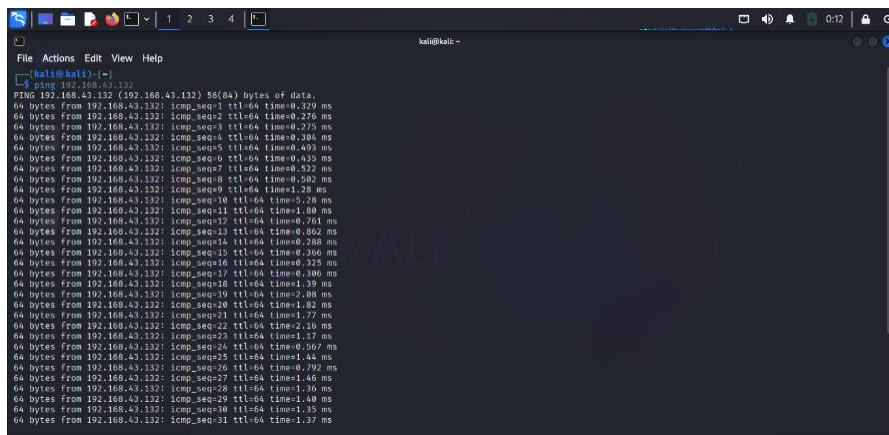
## Requirements :-

- Kali Linux (Attacker)
- Metasploitable 2 with DVWA installed (Target)
- Both VMs on Host-Only Network
- Browser – Firefox

## Task1 :- Network Setup and Access DVWA

1. Confirm network connectivity between Kali and DVWA host using “ping”

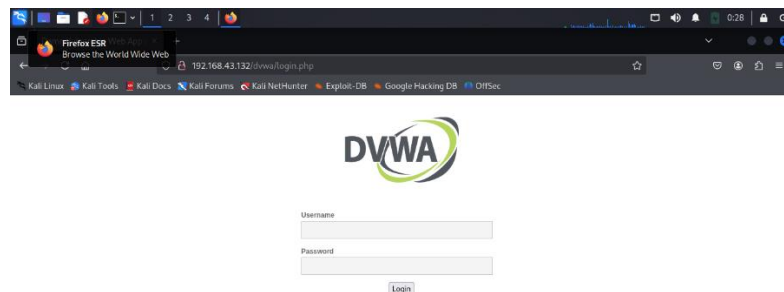
In Metasploitable – ifconfig IP – 192.168.43.132



```
kali@kali:~$ ping 192.168.43.132
PING 192.168.43.132 (192.168.43.132) 56(84) bytes of data.
64 bytes from 192.168.43.132: icmp_seq=1 ttl=64 time=0.229 ms
64 bytes from 192.168.43.132: icmp_seq=2 ttl=64 time=0.275 ms
64 bytes from 192.168.43.132: icmp_seq=3 ttl=64 time=0.275 ms
64 bytes from 192.168.43.132: icmp_seq=4 ttl=64 time=0.284 ms
64 bytes from 192.168.43.132: icmp_seq=5 ttl=64 time=0.403 ms
64 bytes from 192.168.43.132: icmp_seq=6 ttl=64 time=0.435 ms
64 bytes from 192.168.43.132: icmp_seq=7 ttl=64 time=0.522 ms
64 bytes from 192.168.43.132: icmp_seq=8 ttl=64 time=0.502 ms
64 bytes from 192.168.43.132: icmp_seq=9 ttl=64 time=1.28 ms
64 bytes from 192.168.43.132: icmp_seq=10 ttl=64 time=0.528 ms
64 bytes from 192.168.43.132: icmp_seq=11 ttl=64 time=1.08 ms
64 bytes from 192.168.43.132: icmp_seq=12 ttl=64 time=0.761 ms
64 bytes from 192.168.43.132: icmp_seq=13 ttl=64 time=0.892 ms
64 bytes from 192.168.43.132: icmp_seq=14 ttl=64 time=0.288 ms
64 bytes from 192.168.43.132: icmp_seq=15 ttl=64 time=0.366 ms
64 bytes from 192.168.43.132: icmp_seq=16 ttl=64 time=0.223 ms
64 bytes from 192.168.43.132: icmp_seq=17 ttl=64 time=0.386 ms
64 bytes from 192.168.43.132: icmp_seq=18 ttl=64 time=0.288 ms
64 bytes from 192.168.43.132: icmp_seq=19 ttl=64 time=1.02 ms
64 bytes from 192.168.43.132: icmp_seq=20 ttl=64 time=1.77 ms
64 bytes from 192.168.43.132: icmp_seq=21 ttl=64 time=2.16 ms
64 bytes from 192.168.43.132: icmp_seq=22 ttl=64 time=1.17 ms
64 bytes from 192.168.43.132: icmp_seq=23 ttl=64 time=1.44 ms
64 bytes from 192.168.43.132: icmp_seq=24 ttl=64 time=0.792 ms
64 bytes from 192.168.43.132: icmp_seq=25 ttl=64 time=1.46 ms
64 bytes from 192.168.43.132: icmp_seq=26 ttl=64 time=1.36 ms
64 bytes from 192.168.43.132: icmp_seq=27 ttl=64 time=1.40 ms
64 bytes from 192.168.43.132: icmp_seq=28 ttl=64 time=1.35 ms
64 bytes from 192.168.43.132: icmp_seq=29 ttl=64 time=1.37 ms
```

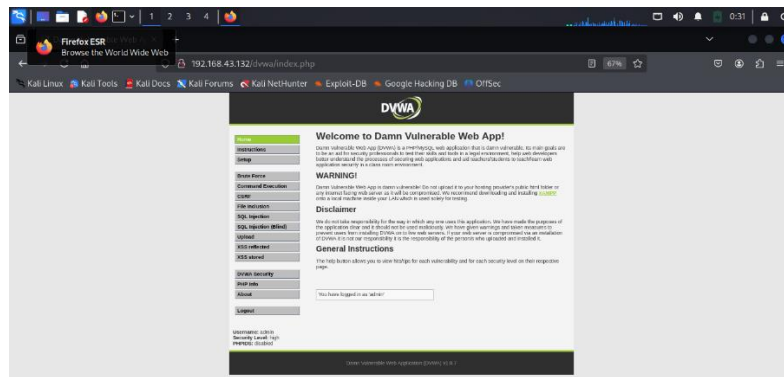
2. Open DVWA in the Kali browser using the target's IP address.

Open Firefox & search <http://192.168.43.132/dvwa>



3. Login to DVWA using default credentials: admin/password

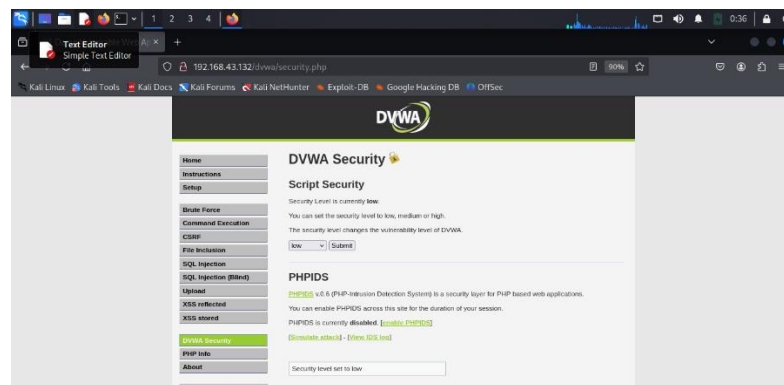
Login with username: admin & password: password



## Task2 :- Set DVWA Security Level

Set the DVWA security level to “Low” under DVWA Security settings.

Go to DVWA Settings & set the level to “low”

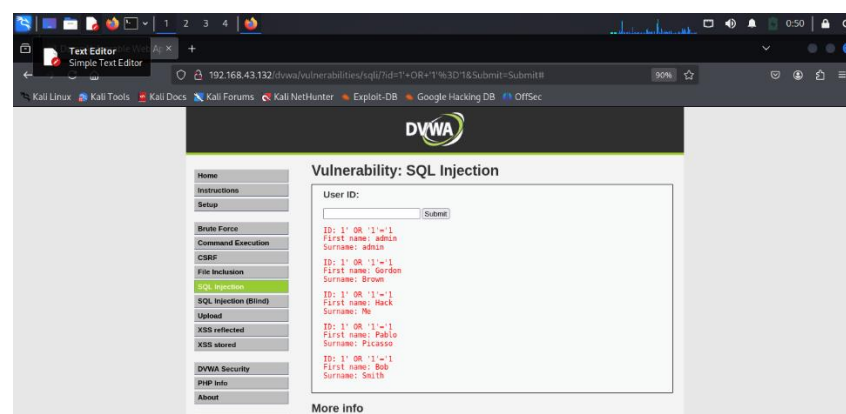


This process disables the built-in protections for our experiment usage.

## Task3 :- Perform Basic SQL Injection

Inject “1’ OR ‘1’ = ‘1” in the SQL Injection module's user ID field.

Click Submit and observe the bypass of login logic.

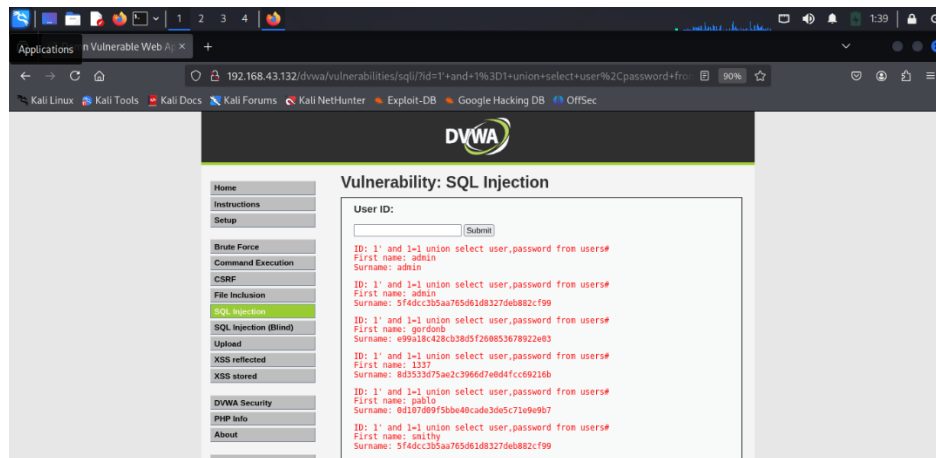


This confirms a successful injection by bypassing logic in the SQL Query.

## Task4 :- Extract User Credentials Using UNION

Use UNION-based injection to extract user and password :

“ 1' and 1=1 union select user,password from users# “



admin

5f4dcc3b5aa765d61d8327deb882cf99

This is the MD5 hash of the password (“password”).

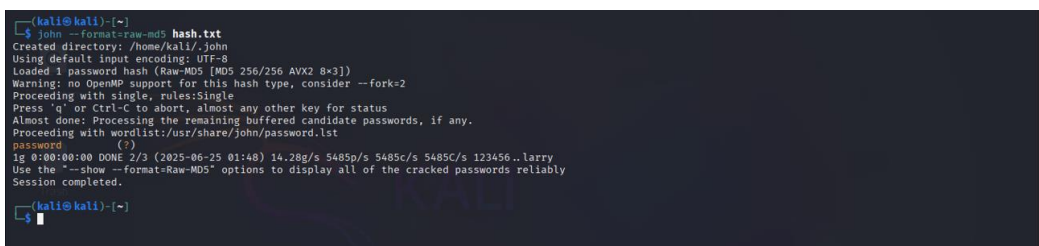
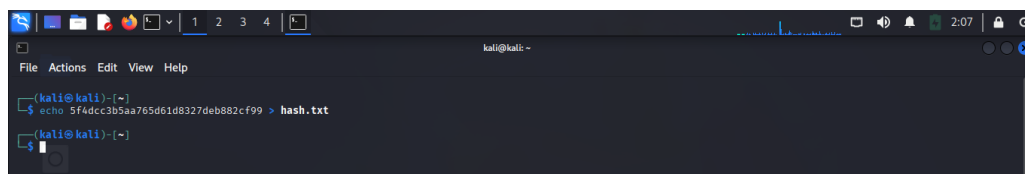
## Task5 :- Crack the Password Hash

Save hash to a file :

echo 5f4dcc3b5aa765d61d8327deb882cf99 > hash.txt

Use John the Ripper to crack the extracted MD5 password hash :

e.g., “john --format=raw-md5 hash.txt”



Result :- password

## Task6 :- Analyze and Report Findings

Document the SQL injection point, payloads used, and results.

Create a summary table with the target URL, data extracted, and cracked password.

| Field              | Value   |
|--------------------|---|
| Target URL :       | <a href="http://192.168.43.132/dvwa/vulnerabilities/sqli/">http://192.168.43.132/dvwa/vulnerabilities/sqli/</a> |
| Injection Used :   | 1' OR '1'='1  |
| Data Extracted:    | Username & hashed password  |
| Password Cracked : | Yes – “password”  |

## Task7 :- Recommend Mitigation Strategies

Provide security recommendations to prevent SQL injection :

1. Implement Input Validation and Sanitization  
By validating and sanitizing user input, applications can ensure that only expected data formats and types are accepted, thereby mitigating the risk of malicious SQL commands being injected into SQL queries
2. Adopt the Least Privilege Principle  
Limiting permissions to the minimum necessary reduces the impact of a successful SQL injection attack. Granting only specific privileges required for the application decreases the potential damage.
3. Deploy Web Application Firewalls (WAF)  
**Web Application Firewall (WAF)** monitors and filters incoming HTTP traffic, detecting and blocking SQL injection attempts and other malicious traffic. It can catch and neutralize known attack patterns before they reach the application
4. Utilize Parameterized Statements (Prepared Statements)  
Parameterized statements separate user inputs from the SQL query, eliminating the need for manual escaping. This ensures that user inputs are treated as data, preventing the execution of malicious code. The database system recognizes placeholders and binds user inputs securely during execution.
5. Use Escaping for User Input  
SQL injection mitigation technique involves modifying user inputs to neutralize special characters that could be used for malicious SQL injection.
6. Incorporate Stored Procedures  
Stored procedures encapsulate SQL code within the database. The injection risk is minimized by defining parameterized procedures, as these procedures are executed without directly incorporating user inputs.

## Conclusion :-

This project showed how insecure web input leads to full SQL data extraction through simple injection techniques. It reinforces why developers must sanitize inputs & follow secure database queries.

