**Breaking Bank - CTF Walkthrough**

**Challenge Name:** Breaking Bank (Solo Leveling Edition)

**Category:** Web Exploitation

**Description:** The challenge presents a login page with username and password fields. The objective is to bypass authentication and obtain the flag.

### ****Step 1: Identifying the Vulnerability****

The source code reveals the following SQL query handling login authentication:

$query = "SELECT \* FROM users WHERE username = '$username' AND password = '$password' LIMIT 1";

This query suggests an SQL injection vulnerability since user input is directly concatenated into the query without sanitization.

### ****Step 2: Testing for SQL Injection****

A simple payload can confirm SQL injection vulnerability:

Username: ' OR '1'='1

Password: anything

This works because the SQL statement effectively becomes:

SELECT \* FROM users WHERE username = '' OR '1'='1' AND password = 'anything' LIMIT 1;

Since '1'='1' is always true, it bypasses authentication.

### ****Step 3: Extracting Hashed Password****

Since the challenge requires a hashed SQL injection, we need to obtain hashed values from the database. A typical UNION-based payload:

Username: ' UNION SELECT 1,2,3,4,5 -- -

This helps determine the number of columns. Adjust the query accordingly:

Username: ' UNION SELECT 1,username,password,4,5 FROM users -- -

This dumps hashed passwords in the response.

### ****Step 4: Cracking the Hash****

Once we retrieve the password hash, we use tools like John the Ripper or Hashcat:

john --wordlist=rockyou.txt hashfile.txt

If the hash is cracked successfully, we can log in using the obtained password.

### ****Step 5: Obtaining the Flag****

Once authenticated, the application displays the flag:

G8KEY{FIRST\_LEVEL\_UP}

### ****Conclusion****

* The challenge demonstrates SQL injection with hashed password retrieval.
* Using UNION queries allows extracting hashed credentials.
* Cracking the hash enables access to the admin panel where the flag is revealed.

**Flag:** G8KEY{FIRST\_LEVEL\_UP}

**Mitigation Measures:**

1. Use prepared statements to prevent SQL injection.
2. Hash and salt passwords securely with bcrypt or Argon2.
3. Validate and sanitize user inputs to prevent malicious payloads.
4. Implement least privilege access for database users.

**End of Walkthrough** ✅