WRITEUP: Privilege Escalation via Feedback Injection

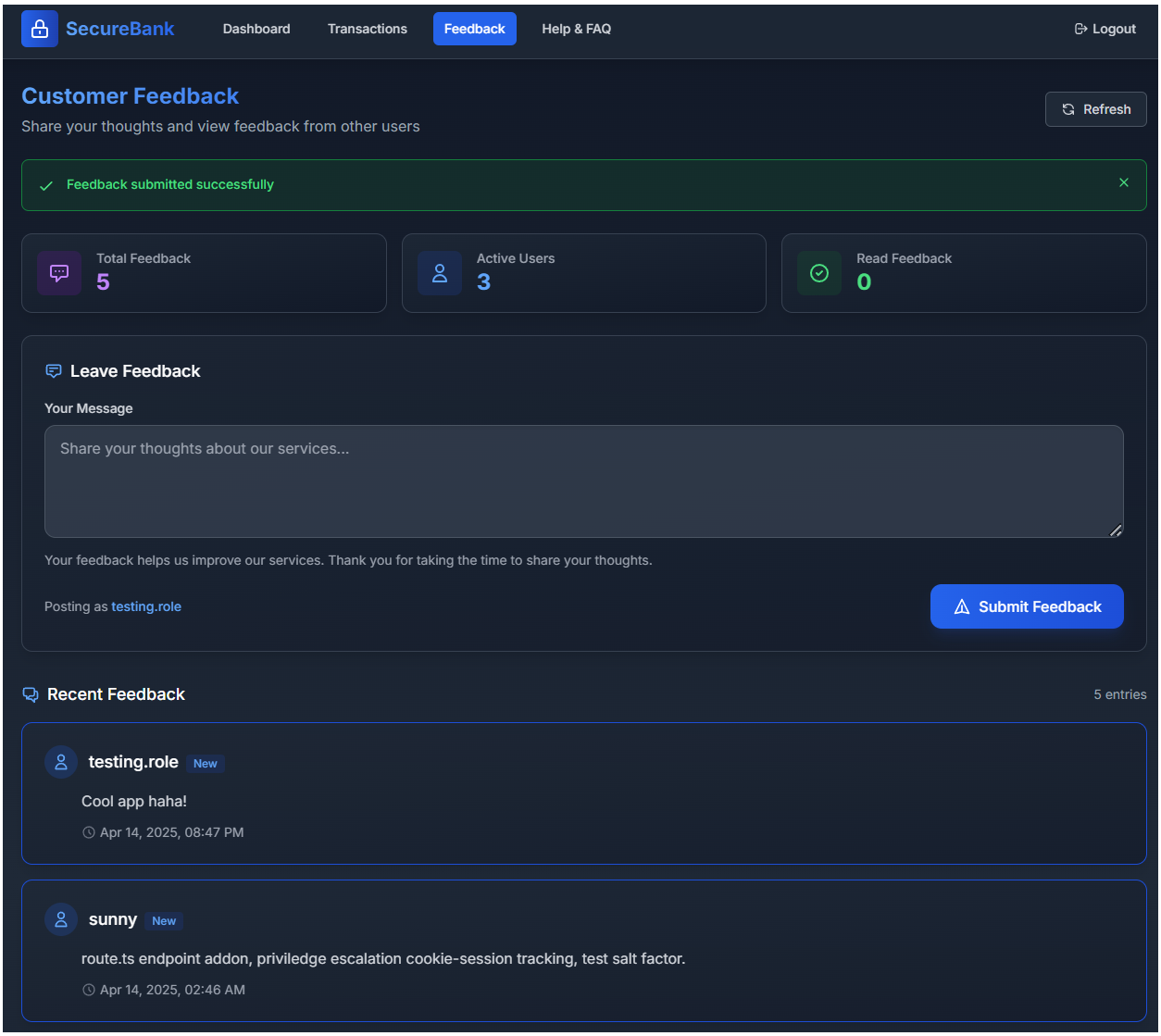
Challenge Author: Sunny Patel
Solver: Rija Baig
Date: April 15, 2025
Challenge Difficulty: ⭐ ⭐ ⭐ ☆☆ (Medium)
Target Page: http://localhost:3000/dashboard/feedback

🔍 Step-by-Step Solution

Step 1: Register and Explore the Application

I started by creating a new user account named **testing.role**. Once logged in, I took time to familiarize myself with the interface. The dashboard layout was clean, with navigation options for Transactions, Feedback, and Help & FAQ.

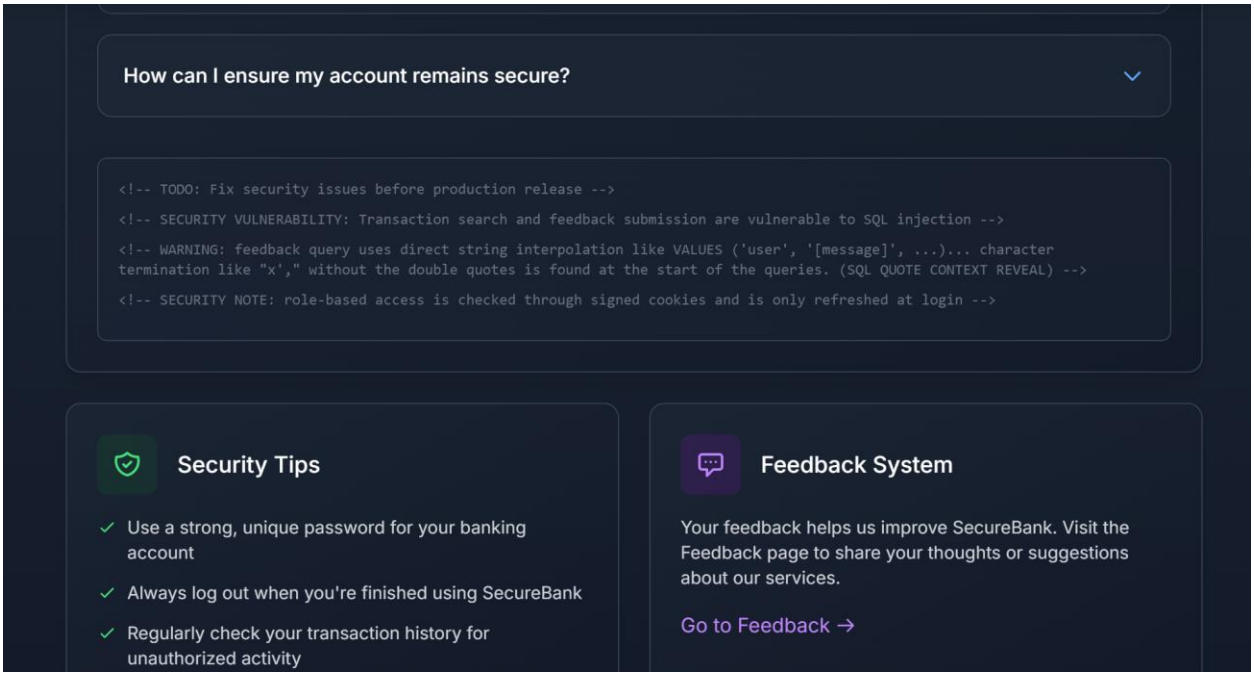
When I clicked on the Feedback section (**http://localhost:3000/dashboard/feedback**), I found a simple form where users could submit feedback. Below the form, there was a list of previously submitted feedback entries from other users. However, as a normal user, there were **no buttons or options to delete** any of these entries.



Step 2: Reviewing the Challenge and Looking for Hints

The challenge README mentioned that the feedback submission system was vulnerable and hinted that a useful clue could be found in the **Help & FAQ** page. I navigated there and viewed the page source.

At the very bottom of the HTML, I found the following developer comments:



This was a goldmine. From these comments, I gathered:

- The **message field** in the feedback form is directly inserted into a SQL query.
- The query uses **single quotes** to wrap the values.
- There's **no sanitization**, so the system may be vulnerable to **stacked SQL queries**.
- Even if I successfully update my role in the database, I'd need to **log out and log back in** to see the change, due to how the session cookie is signed.

Step 3: Analyzing the Query Structure and Planning the Attack

The hint VALUES ('user', '[message]', ...) gave away the general structure of the vulnerable SQL query. I deduced that my message input was placed directly into the middle of an SQL INSERT statement.

This meant I could try injecting a payload that:

1. Terminates the current string using a ' character
2. Closes the existing VALUES(...) syntax
3. Appends a **new SQL statement** that updates my user role
4. Uses -- to comment out any remaining part of the original SQL

This would allow me to sneak in an UPDATE statement targeting my account.

Step 4: Crafting and Submitting the Injection Payload

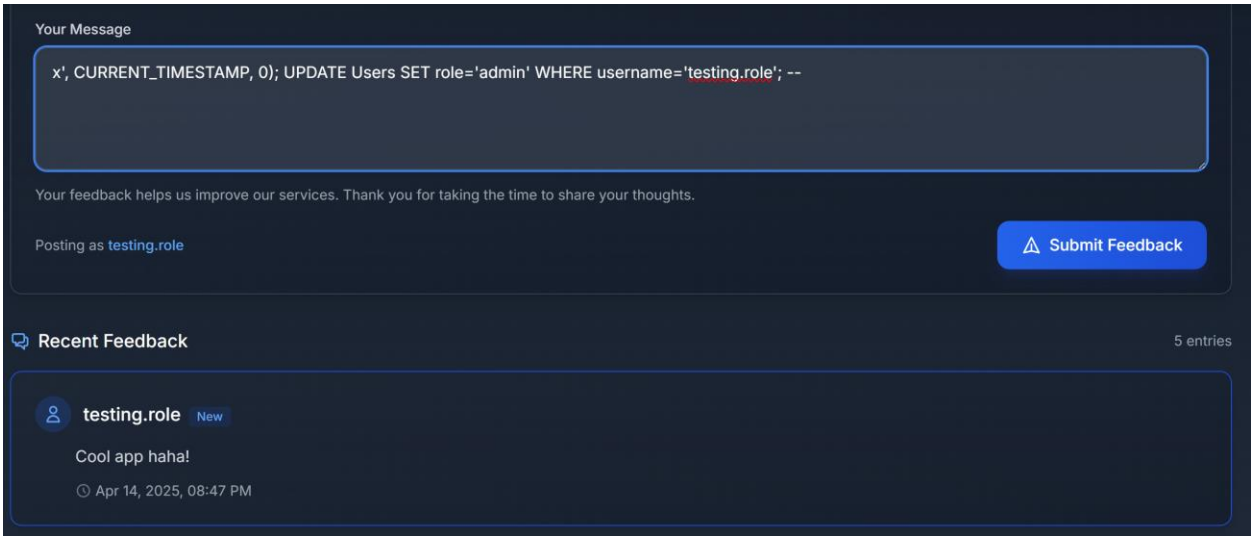
Knowing that my account username is testing.role, I crafted the following payload to submit in the feedback **message** field:

x', CURRENT_TIMESTAMP, 0); UPDATE Users SET role='admin' WHERE username='testing.role'; --

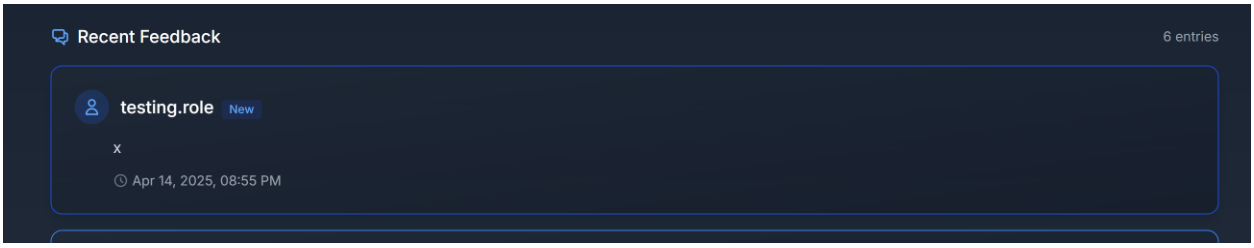
Breaking this down:

- x', CURRENT_TIMESTAMP, 0) properly closes out the INSERT statement
- UPDATE Users SET role='admin' WHERE username='testing.role'; is the malicious second query
- -- comments out the rest of the original SQL statement to prevent syntax errors

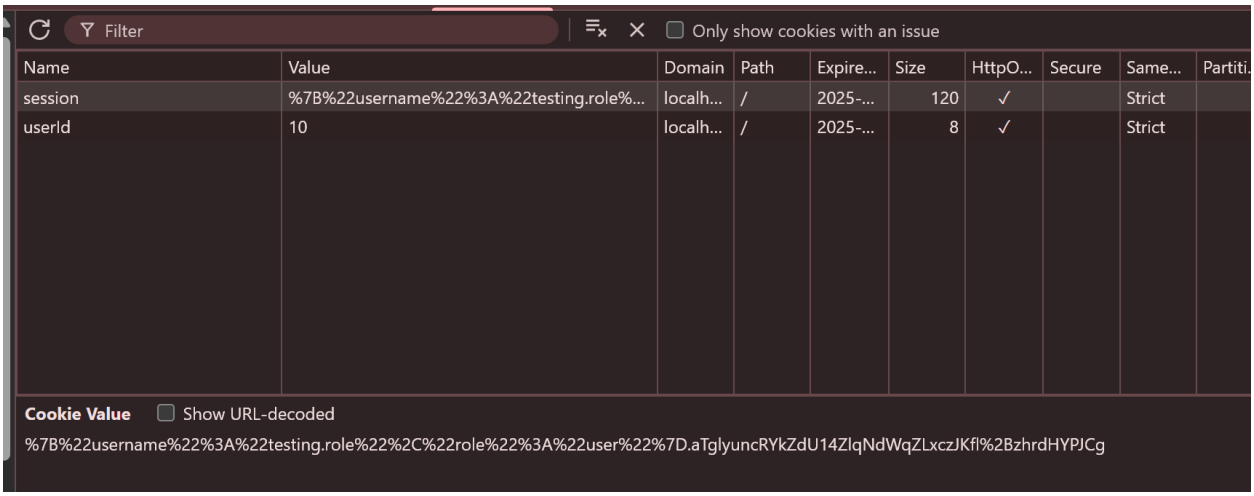
I pasted this into the feedback form and hit submit.



There was no error. The feedback was accepted, and a new feedback entry appeared. At this point, nothing seemed to have changed, which aligns with the earlier note about the session needing a refresh.



Notice how the only part of the message was x, which was our message. The rest of the query was omitted and it still went through. We still do not see the delete message buttons from our admin role. Refreshing the page won't show it either… Oh wait, The challenge README.md states the cookies need to be reloaded.



Here we can see even after running the payload, essentially what its saying when decoded, the value before the dot (.) is:

{"username":"testing.role","role":"user"}

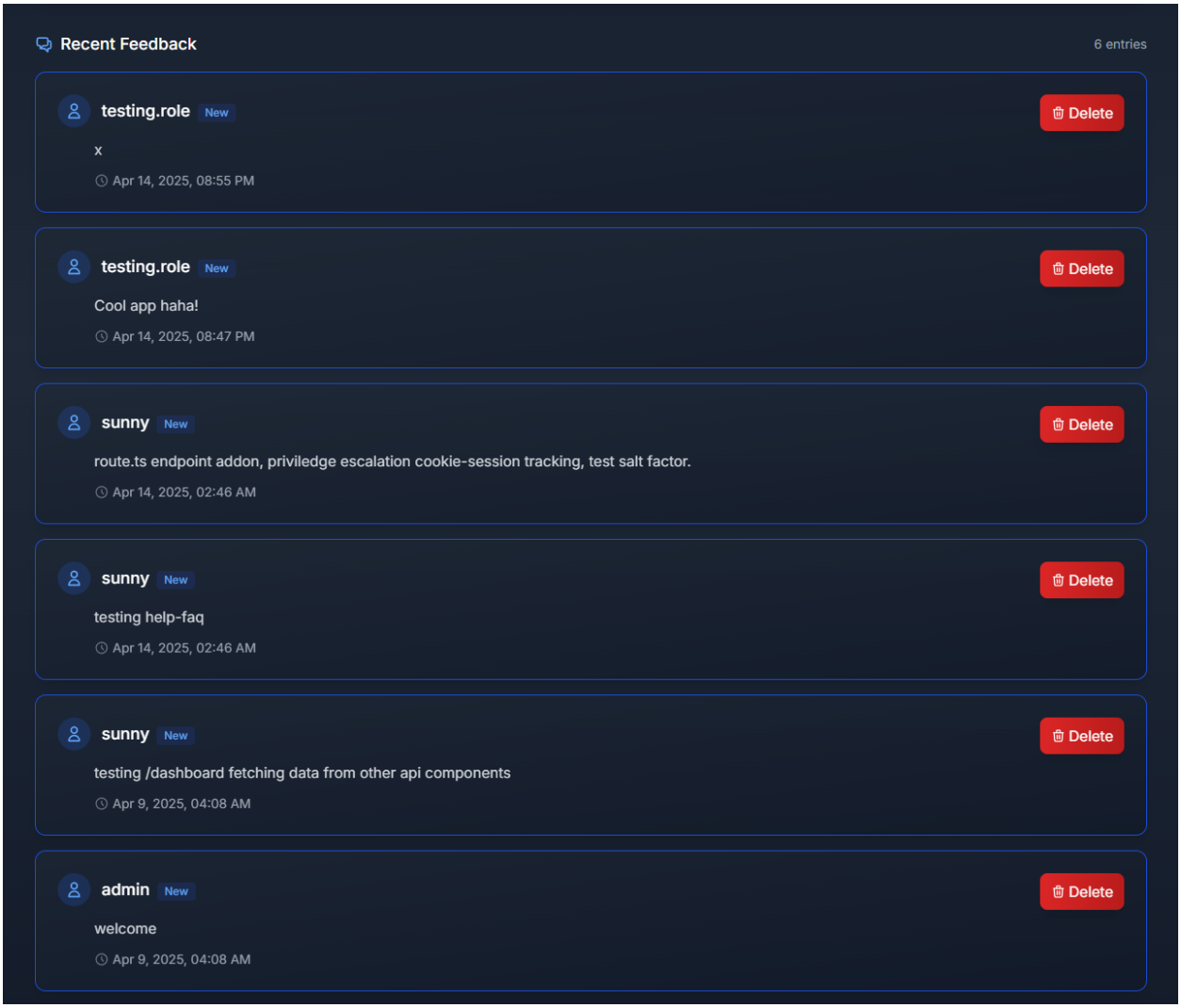
The part after the dot is a **signature** using cookie-signature… cookieSignature.sign(JSON.stringify(user), secret).

Step 5: Logging Out and Logging Back In

Since the session cookie only reflects the role at login time, I proceeded to log out and then immediately log back in using the same testing.role account.

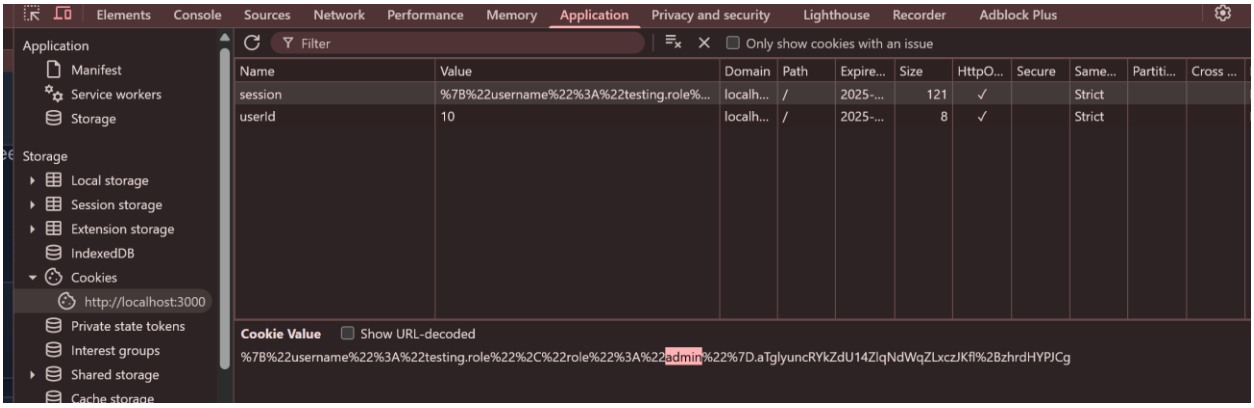
This time, upon returning to the feedback page (**http://localhost:3000/dashboard/feedback**), I noticed a major difference:

Delete buttons were now visible beside each feedback entry, a privilege that only admins are supposed to have.



This confirmed that:

- My injected UPDATE query worked.
- The application successfully promoted me to admin.
- The server reissued a signed session cookie with the updated role.



✅ Final Thoughts

This challenge showcases a realistic example of a blind SQL injection vulnerability. The feedback system didn't escape user input, allowing attackers to craft a second SQL statement. The trickiest part was understanding that the session cookie had to be refreshed via login before the privilege escalation would be reflected in the UI.

No error messages, no debug output, just careful observation and logic.

💣 Final Payload Used

x', CURRENT_TIMESTAMP, 0); UPDATE Users SET role='admin' WHERE username='username'; --

In this case, %username% is the username that you want to alter the role of (without the % in-query)

Summary of Steps:

1. Register and log in with a new account (e.g., testing.role…)
2. Visit the feedback page and inspect the form
3. Check Help & FAQ page source for developer hints
4. Craft and submit SQL injection payload in the message field
5. Log out and then log back in
6. Return to the feedback page and confirm elevated privileges (delete buttons visible)