

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.

The screenshot shows the 'Customer Feedback' section of the SecureBank application. At the top, there are navigation links: 'SecureBank' (with a lock icon), 'Dashboard', 'Transactions', 'Feedback' (which is highlighted in blue), and 'Help & FAQ'. On the far right is a 'Logout' link. Below the navigation, the title 'Customer Feedback' is displayed, followed by the sub-instruction 'Share your thoughts and view feedback from other users'. A green success message box contains the text '✓ Feedback submitted successfully'. To the right of this message is a small 'X' icon. Below the message are three summary cards: 'Total Feedback' (5), 'Active Users' (3), and 'Read Feedback' (0). Underneath these cards is a form titled 'Leave Feedback' with a placeholder 'Your Message' and a text area for input. The text area contains the instruction 'Share your thoughts about our services...'. Below the text area is a note: 'Your feedback helps us improve our services. Thank you for taking the time to share your thoughts.' To the left of the note is the text 'Posting as testing.role'. To the right of the note is a blue button with a white triangle icon and the text 'Submit Feedback'. Below this section is another card titled 'Recent Feedback' with a count of '5 entries'. It lists two recent messages: one from 'testing.role' (New) posted on April 14, 2025, at 08:47 PM, and one from 'sunny' (New) posted on April 14, 2025, at 02:46 AM.

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:

The screenshot shows a dark-themed web application interface. At the top, a question is displayed: "How can I ensure my account remains secure?". Below it, a code block contains several TODO and WARNING annotations related to SQL injection and role-based access:

```
<!-- TODO: Fix security issues before production release -->
<!-- SECURITY VULNERABILITY: Transaction search and feedback submission are vulnerable to SQL injection -->
<!-- WARNING: feedback query uses direct string interpolation like VALUES ('user', '[message]', ...)... character
termination like "x," without the double quotes is found at the start of the queries. (SQL QUOTE CONTEXT REVEAL) -->
<!-- SECURITY NOTE: role-based access is checked through signed cookies and is only refreshed at login -->
```

The interface is divided into two main sections: "Security Tips" and "Feedback System".

- Security Tips:** Contains three items:
 - ✓ Use a strong, unique password for your banking account
 - ✓ Always log out when you're finished using SecureBank
 - ✓ Regularly check your transaction history for unauthorized activity
- Feedback System:** Contains a message and a link:

Your feedback helps us improve SecureBank. Visit the Feedback page to share your thoughts or suggestions about our services.

[Go to Feedback →](#)

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.

Your Message

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

Your feedback helps us improve our services. Thank you for taking the time to share your thoughts.

Posting as testing.role

Submit Feedback

Recent Feedback

5 entries

testing.role **New**

Cool app haha!

Apr 14, 2025, 08:47 PM

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.

Recent Feedback

6 entries

testing.role **New**

x

Apr 14, 2025, 08:55 PM

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.

C Filter | Only show cookies with an issue

Name	Value	Domain	Path	Expire...	Size	HttpO...	Secure	Same...	Partiti...
session	%7B%22username%22%3A%22testing.role%...	localh...	/	2025-...	120	✓		Strict	
userId	10	localh...	/	2025-...	8	✓		Strict	

Cookie Value Show URL-decoded
%7B%22username%22%3A%22testing.role%22%2C%22role%22%3A%22user%22%7D.aTglyuncRYkZdU14ZlqNdWqZLxczJKfl%2BzhrdHYPJCg

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.

Recent Feedback

6 entries

- testing.role** [New]
- X
- ⌚ Apr 14, 2025, 08:55 PM
- Delete**

- testing.role** [New]
- Cool app haha!
- ⌚ Apr 14, 2025, 08:47 PM
- Delete**

- sunny** [New]
- route.ts endpoint addon, privilege escalation cookie-session tracking, test salt factor.
- ⌚ Apr 14, 2025, 02:46 AM
- Delete**

- sunny** [New]
- testing help-faq
- ⌚ Apr 14, 2025, 02:46 AM
- Delete**

- sunny** [New]
- testing /dashboard fetching data from other api components
- ⌚ Apr 9, 2025, 04:08 AM
- Delete**

- admin** [New]
- welcome
- ⌚ Apr 9, 2025, 04:08 AM
- Delete**

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.

Elements Console Sources Network Performance Memory Application Privacy and security Lighthouse Recorder Adblock Plus

Application

- Manifest
- Service workers
- Storage

Storage

- Local storage
- Session storage
- Extension storage
- IndexedDB
- Cookies

http://localhost:3000

Name	Value	Domain	Path	Expires	Size	HttpOnly	Secure	SameSite	Partition	Cross Site
session	%7B%22username%22%3A%22testing.role%22%7D	localhost	/	2025-04-14T08:55:00Z	121	✓		Strict		
userId	10	localhost	/	2025-04-14T08:55:00Z	8	✓		Strict		

Cookie Value Show URL-decoded

%7B%22username%22%3A%22testing.role%22%2C%22role%22%3A%22admin%22%7D.aTglyuncRYkZdU14ZlqNdWqZLxczJKf%2BzhrdHYPIcG

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)