**SCENARIO**

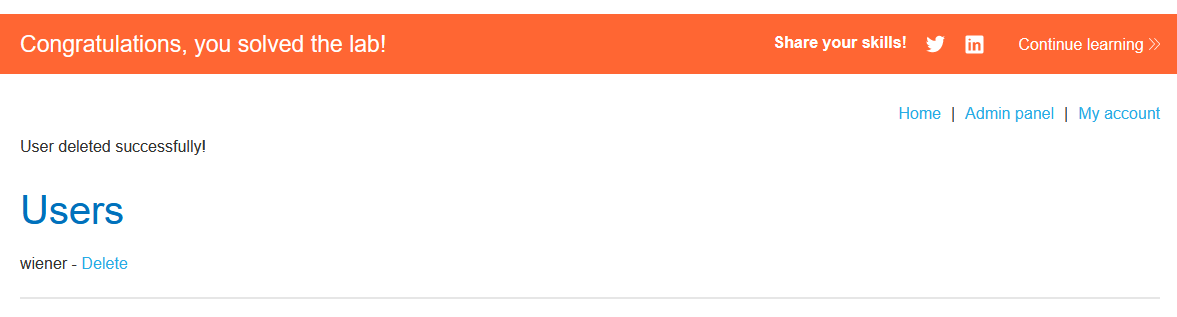
Our online blog platform has integrated a feature allowing users to connect their accounts with a social media profile for easier OAuth login. However, there appears to be an insecure implementation of the OAuth flow, which could be exploited by attackers to gain unauthorized access to any user's account, including that of administrators.

**PROCEDURE**

1. Begin by directing our traffic through a proxy tool like Burp Suite, then click on "My account" on the blog platform. Though there's an option to log in using a social media profile, we first choose the standard login form and use the provided credentials to access our account.
2. Once logged in, we observe an option to link our social media profile with the blog account.
3. Click on "Attach a social profile". This action reroutes us to the associated social media website. Here, we log in using our social media credentials to finalize the OAuth process. Upon successful linking, we are redirected back to the original blog platform.
4. To validate the linkage, log out of the blog website, then click "My account" again. This time, select "Log in with social media" and note that we are automatically logged in, showcasing that the OAuth flow has been correctly established.
5. To understand the underlying OAuth flow, we inspect our proxy history and identify the sequence of requests made when attaching a social profile. Within these requests, particularly in the GET /auth?client\_id[...] request, we notice the redirect\_uri sends the authorization code to /oauth-linking. Interestingly, this request does not incorporate a state parameter, leaving the process vulnerable to CSRF attacks.
6. With our proxy tool's interception on, we reinitiate the "Attach a social profile" process.
7. While navigating through the Burp Proxy, we continue forwarding requests until intercepting the one targeting GET /oauth-linking?code=[...]. We then copy this request URL for later use.
8. It's crucial to drop the intercepted request, ensuring the captured code remains valid and unused.
9. After deactivating the proxy interception, log out from the blog platform to reset our session.
10. On the exploit server, we generate an iframe, setting the src attribute to the previously copied URL. The iframe structure should resemble: <iframe src="https://LAB-ID.web-security-academy.net/oauth-linking?code=CAPTURED-CODE"></iframe>
11. By delivering this exploit to a victim (in this case, the admin), their browser will unknowingly complete the OAuth process using our social media profile, linking it to the admin's blog account.
12. Return to the blog website and choose the "Log in with social media" option. Now, we observe that we've gained access as the admin user. Access the admin dashboard and delete the "carlos" user to confirm our elevated privileges.

**PAYLOAD**

<iframe src="https://LAB-ID.web-security-academy.net/oauth-linking?code=CAPTURED-CODE"></iframe>

**PROOF OF CONCEPT**

**REMEDIATION**

1. Ensure that the OAuth 2.0 state parameter is used and verified for each OAuth request, preventing potential CSRF attacks.
2. Implement strict referer checks during the OAuth linking process.
3. Introduce regular security reviews and penetration testing, especially when implementing third-party login services.
4. Provide user notifications whenever a new login or linking action occurs, allowing them to promptly identify and address any unauthorized activities.
5. Use multi-factor authentication (MFA) as an additional security layer, ensuring even if an account linkage is abused, the attacker cannot gain full control.