SCENARIO

The application consists of an email change functionality which is vulnerable to CSRF. We will try to mount an exploit and deliver it to the target with the help of an exploit server in order to change the email address of the target.

**PROCEDURE**

1. Go the vulnerable web applications and log in with the username and password of Wiener’s account provided to act as an user.
2. Then change the email once to study the HTTP request and response and we see that there are no unpredictable tokens so it may be vulnerable to CSRF if we bypass the SameSite restriction.
3. Check if there are any loopholes by changing the request type, CSRF Key, CSRF Token, Session Token, etc.
4. As there are no SameSite restrictions, the browser will follow Lax restriction level.
5. So, the session cookie will be sent in cross-site GET requests, as long as they involve a top-level navigation.
6. Send the change email request to BurpSuite’s Repeater, change the request method to GET and we see that the endpoint only allows POST requests.
7. Now try overriding the method by injecting the Payload 1 in the URL and we see that we managed to get through the request restriction as our email address was changed.
8. Go to the exploit server and paste the Payload 2 into the body tag of the exploit and click store.
9. At the end, click the button to deliver the exploit.

**PAYLOAD**

1. GET /my-account/change-email?email=foo%40web-security-academy.net&\_method=POST HTTP/1.1
2. <script>

document.location = "https://YOUR-LAB-ID.web-security-academy.net/my-account/change-email?email=pwned@web-security-academy.net&\_method=POST";

</script>

**REMEDIATION**

1. **Implement Strong CSRF Tokens:** Introduce unpredictable CSRF tokens for every session and tie them to that particular session. By doing so, even if an attacker tries to forge a request, they would need a valid CSRF token, which would be almost impossible to predict. This unpredictability can shield applications from CSRF attacks.
2. **Ensure Strict HTTP Methods:** Restrict each endpoint to its designated HTTP method. By strictly defining what HTTP methods are allowed for each route (GET, POST, PUT, DELETE, etc.), you prevent attackers from exploiting lax configurations. In the scenario described, the application was exploited by overriding the HTTP method. Strictly enforcing HTTP methods will prevent such attacks.
3. **Use the SameSite Cookie Attribute Appropriately:** Update the cookie settings to use the SameSite=Strict attribute. With this attribute in place, browsers will not send the cookie in any cross-site requests. This offers significant protection against CSRF attacks, as attackers will be unable to force a victim's browser to make an authenticated cross-site request. If "Strict" interferes with legitimate cross-site interactions, SameSite=Lax could be considered, but it's essential to understand the security trade-offs.