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 but there exists an explicitly defined **SameSite** parameter in the **login request** as **Strict**.
3. Check if there are any loopholes by changing the request type, CSRF Key, CSRF Token, Session Token, etc.
4. Because of SameSite restriction as Strict the application won’t include any cookies in the cross-origin request.
5. Now go and comment on any blog post and we see that after a second we are redirected back to some other page and the redirection is from some other page named **commentConfirmationRedirect.js.**
6. If we check that page then we see that the application injects the **postID** dynamically into the URL.
7. We will try to inject a dynamic path into that postID parameter to change the email using the POST request to change the email by converting it to a GET request.
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**

<script>

document.location = "https://YOUR-LAB-ID.web-security-academy.net/post/comment/confirmation?postId=1/../../my-account/change-email?email=pwned%40web-security-academy.net%26submit=1";

</script>

**REMEDIATION**

1. **Implement Strong CSRF Tokens:** Introduce unpredictable CSRF tokens for every session and tie them to that particular session. This unpredictability can shield applications from CSRF attacks, and since tokens are tied to specific sessions, attackers cannot reuse them across different user sessions.
2. **Enforce Strict Path Restrictions:** Implement path validation to prevent path traversal attacks. Avoid using user-controlled input as a part of the path, especially without proper validation. In this scenario, the "postId" parameter was manipulated, and by putting strict validation, such attempts can be thwarted.
3. **Regularly Audit and Review Code:** Conduct periodic reviews of the application codebase, particularly areas that manage user input and redirections. This can help in identifying potential vulnerabilities like the one with the postId parameter, ensuring they're addressed before malicious exploitation.