SCENARIO

The application consists of an email change functionality which is vulnerable to CSRF even though some security controls are implemented but those tokens are not tied to any session. 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.
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. Now, login as the Carlos’s/attacker’s account and inspect the source of the page to get the CSRF token of the attacker and copy it.
6. Go to the exploit server and paste the request into the body tag of the exploit and replace the original token with the attacker’s token, then click store.
7. 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.

**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>

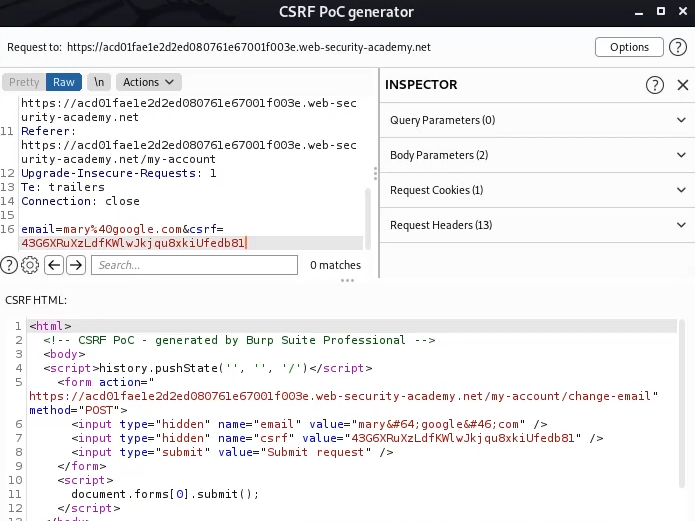
<script>

document.forms[0].submit();

</script>

</body>

</html>

**PROOF OF CONCEPT**

****

**REMEDIATION**

1. **Bind CSRF Tokens to Sessions:** It's crucial that each CSRF token is tied directly to a user's session. This binding ensures that even if an attacker manages to get their hands on a CSRF token, they can't employ it within the scope of a different user's session. This approach ensures the user-specificity of tokens, thereby preventing their reuse across varied sessions.
2. **Token Unpredictability:** Make sure that the CSRF tokens in use are of significant length, random, and difficult to predict. To achieve this, employ a robust cryptographic algorithm for their generation. Such tokens are robust against potential prediction or brute-force attacks.
3. **Implement Referrer Header Checks:** For each incoming request, validate the 'Referer' header. The aim is to ascertain that the request originates from a genuine page of your application and isn't coming from an external or potentially malicious source. This validation acts as another protective layer against CSRF assaults, even though it shouldn't be the only protective mechanism in place.