Bypass of CSRF Middleware in Astro Writeup

Challenge Information

CVE Link, Reference Link

Difficulty: Easy-Medium

Category: Cross-Site Request Forgery.

Author: Hunter Stout

Description

The Official CVE Description States “Astro is a web framework for content-driven websites. In affected versions a bug in Astro’s CSRF-protection middleware allows requests to bypass CSRF checks. When the `security.checkOrigin` configuration option is set to `true`, Astro middleware will perform a CSRF check. However, a vulnerability exists that can bypass this security. A semicolon-delimited parameter is allowed after the type in `Content-Type`. Web browsers will treat a `Content-Type` such as `application/x-www-form-urlencoded; abc` as a `simple request` and will not perform preflight validation. In this case, CSRF is not blocked as expected. Additionally, the `Content-Type` header is not required for a request. This issue has been addressed in version 4.16.17 and all users are advised to upgrade. There are no known workarounds for this vulnerability.”

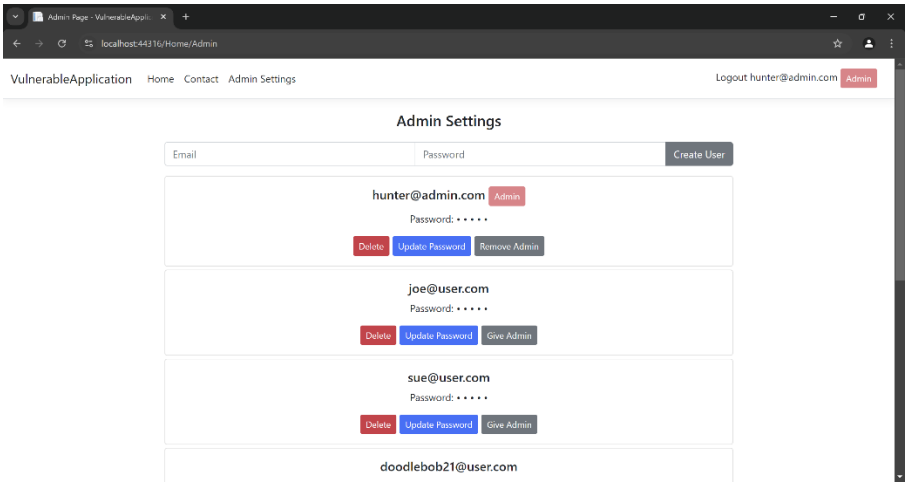
To further break down the vulnerability here in my own words. Let’s start with a CSRF attack, these are attacks that trick “authenticated” users in into performing unwanted actions. They do this through social engineering such as a malicious phishing email. Once the user clicks on the malicious link, assuming they are authenticated to the website at hand, sends a request that could cause harmful actions. The best way to protect against these attacks, which most software development tools have built in protection against this, is sending a cookie token. This token basically states to an endpoint that the request is coming from the same application. Outside sources would not be able to generate this session token. The problem with this CVE specifically is that astro’s built in middleware CSRF protection has a chink in their armor for version 4.16.16. With the correct settings you can bypass certain requests under specific conditions. The challenge I’ve created shows how dangerous this could truly be for someone lacking the proper protection.

This challenge requires a “hacker” and “user” role. The hacker has created a malicious astro application that can send fake authenticated requests to a vulnerable application. The point of this CTF is to create discrete but effective social engineering to manipulate the user into giving the hacker admin access. There was a hint released on accident showing the potential for a vulnerable endpoint. The hacker must exploit the user into falling for the trap, then punish the user & developers for their mistake.

Hints

Hint 1: Your work email is undercoveruser@suspicious.com and your password is “dummy”.

Hint 2: The company has a very strict naming policy. What you see on the screen will most likely reflect the code.




Hint 3: It’s important to cover your tracks. Maybe manipulating the astro page to appear more believable. Maybe after the damage has been done, remove your admin access to hide the trace. Half of the solution is the discreteness of how the outcome is achieved.

Solution

1. Inside of the Astro page of the malicious application, change the url of the fetch request to be “localhost:<YourLocalPortNumber>/Home/Admin/GiveAdmin”. The body needs to be your work email.
2. Change the JavaScript of the Astro Page to interact with the user. Maybe reload or close on submission to draw less attention to the fake page.
3. Send a fake work email containing the link to your malicious page. You will be given admin access once the user clicks on the link & interacts with the page.
4. On the vulnerable application, remove everyone besides you from admin access. Then change around some posts & passwords.
5. Once the damage has been done, remove your admin access to cover up your tracks.