SCENARIO

The application contains a DOM based client-side cookie manipulation vulnerability which redirects the user to our exploit server. We’ll try to exploit this vulnerability by forcing the user to inject a cookie that will cause XSS on a different page and call the print() function.

**PROCEDURE**

1. Access the application and open any blog to inspect its source code.
2. Upon inspecting we see that there is a button for going back to the home page which contains a cookie parameter named lastViewedProduct:

document.cookie = 'lastViewedProduct=' + window.location + '; SameSite=None; Secure'

1. We notice that the URL parameter is the actual value of that cookie after using cookie inspector.
2. So, according to that we will create a payload and navigate to it by injecting it into our URL which will change the URL parameter to force the application to go back to a different page.

**PAYLOAD**

<iframe src="https://0a62004303cc6e3b82f84273003500a2.web-security-academy.net/product?productId=1&'><script>print()</script>" onload="if(!window.x)this.src='https://0a62004303cc6e3b82f84273003500a2.web-security-academy.net';window.x=1;">

**REMEDIATION**

1. **Strict Validation of Input Data:** Always validate, sanitize, and escape data that will be processed. In this case, the value that's being stored in the cookie should be strictly validated against a set pattern to ensure it doesn't contain any malicious characters or scripts.
2. **Encoding of Cookie Values:** Values stored in cookies should be URL encoded, making it difficult to insert executable JavaScript code. Also, consider encrypting the cookie content so that they can't be tampered with easily by an attacker.
3. **Use HttpOnly Attribute:** By setting the HttpOnly attribute for cookies, you ensure that they cannot be accessed or modified by JavaScript running in the browser. This prevents XSS attacks from stealing or tampering with cookies.
4. **Content Security Policy (CSP):** Implement a strict CSP header that restricts where the content can be loaded from and what type of scripts can be executed. This will reduce the risk of any injected script from executing.
5. **SameSite Attribute:** Use the SameSite attribute for cookies, which prevents cross-site request forgery (CSRF) attacks. Although it's primarily for CSRF, it adds an extra layer of security against potential misuse of cookies.
6. **Limit Cookie Scope:** Restrict the scope of the cookie to specific paths or domains using the Path and Domain attributes. This ensures that the cookie is only sent to the server when requested from a specific path or domain.