SCENARIO

The application contains a DOM based client-side simple message-based vulnerability which posts a message. We’ll try to exploit this vulnerability by forcing the user to inject a cookie that will cause XSS on the 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 an event listener active for the ads:

window.addEventListener('message', function(e) {

document.getElementById('ads').innerHTML = e.data;

})

1. When the iframe loads, the postMessage() method sends a web message to the home page. The event listener, which is intended to serve ads, takes the content of the web message and inserts it into the div with the ID ads. However, in this case it inserts our img tag, which contains an invalid src attribute. This throws an error, which causes the onerror event handler to execute our payload..
2. So, according to that we will create a payload and inject it into the application using our exploit server to deliver it to the victim.

**PAYLOAD**

<iframe src="https://YOUR-LAB-ID.web-security-academy.net/" onload="this.contentWindow.postMessage('<img src=1 onerror=print()>','\*')">

**REMEDIATION**

1. **Validation and Sanitization:** Always validate and sanitize data that will be processed. In this scenario, the data from the event listener should be strictly validated and sanitized before being injected into the DOM. Only allow safe and expected message structures.
2. **Use of document.createElement:** Instead of directly inserting HTML content using innerHTML, create DOM elements programmatically using methods like document.createElement(). This way, you can safely set attributes and text without risk of injecting malicious scripts.
3. **White-listed Origins:** When using postMessage(), always specify the target origin to be more restrictive than a wildcard "\*". Specify exactly which origins are allowed to receive the message.
4. **Avoiding Direct DOM Manipulation:** Instead of directly manipulating the DOM using innerHTML, consider using safer alternatives like innerText or textContent which don't parse their input as HTML.
5. **Monitoring and Logging:** Regularly monitor and log messages being posted to detect any malicious activities.