SCENARIO

The application contains a DOM based client-side redirection vulnerability which is triggered by web messaging. 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 to inspect its source code.
2. Upon inspecting we see that there is an event listener active that listens for a web message. The JavaScript contains a flawed indexOf() check that looks for the strings "http:" or "https:" anywhere within the web message. It also contains the sink location.href.:

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

var url = e.data;

if (url.indexOf('http:') > -1 || url.indexOf('https:') > -1) {

location.href = url;

}

}, false);

1. This script sends a web message containing an arbitrary JavaScript payload, along with the string "http:". The second argument specifies that any targetOrigin is allowed for the web message. When the iframe loads, the postMessage() method sends the JavaScript payload to the main page. The event listener spots the "http:" string and proceeds to send the payload to the location.href sink, where the print() function is called.
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('javascript:print()//http:','\*')">

**REMEDIATION**

1. **Validate URL Before Redirection:** Always validate URLs before redirection. In this case, the application should validate that the URL received from the web message matches a list of safe and expected URLs. Any URL that doesn't match the whitelist should not trigger redirection.
2. **Strict Matching with Protocol:** Instead of checking if the strings "http:" or "https:" are just present anywhere within the message, the application should check if the string starts with these protocols.
3. Safeguard against Arbitrary Protocols: If checking for "http:" or "https:", ensure that other harmful protocols like "javascript:" are strictly disallowed.
4. **White-listed Origins for Web Messaging:** When using the postMessage() method, specify a target origin other than a wildcard "\*". Indicate exactly which origins are allowed to receive the message. This prevents cross-site messaging attacks.
5. **Avoiding Dynamic Redirections:** If possible, avoid dynamic client-side redirections based on user input or web messages. Instead, use server-side logic to handle redirections, which is generally safer and more controllable.