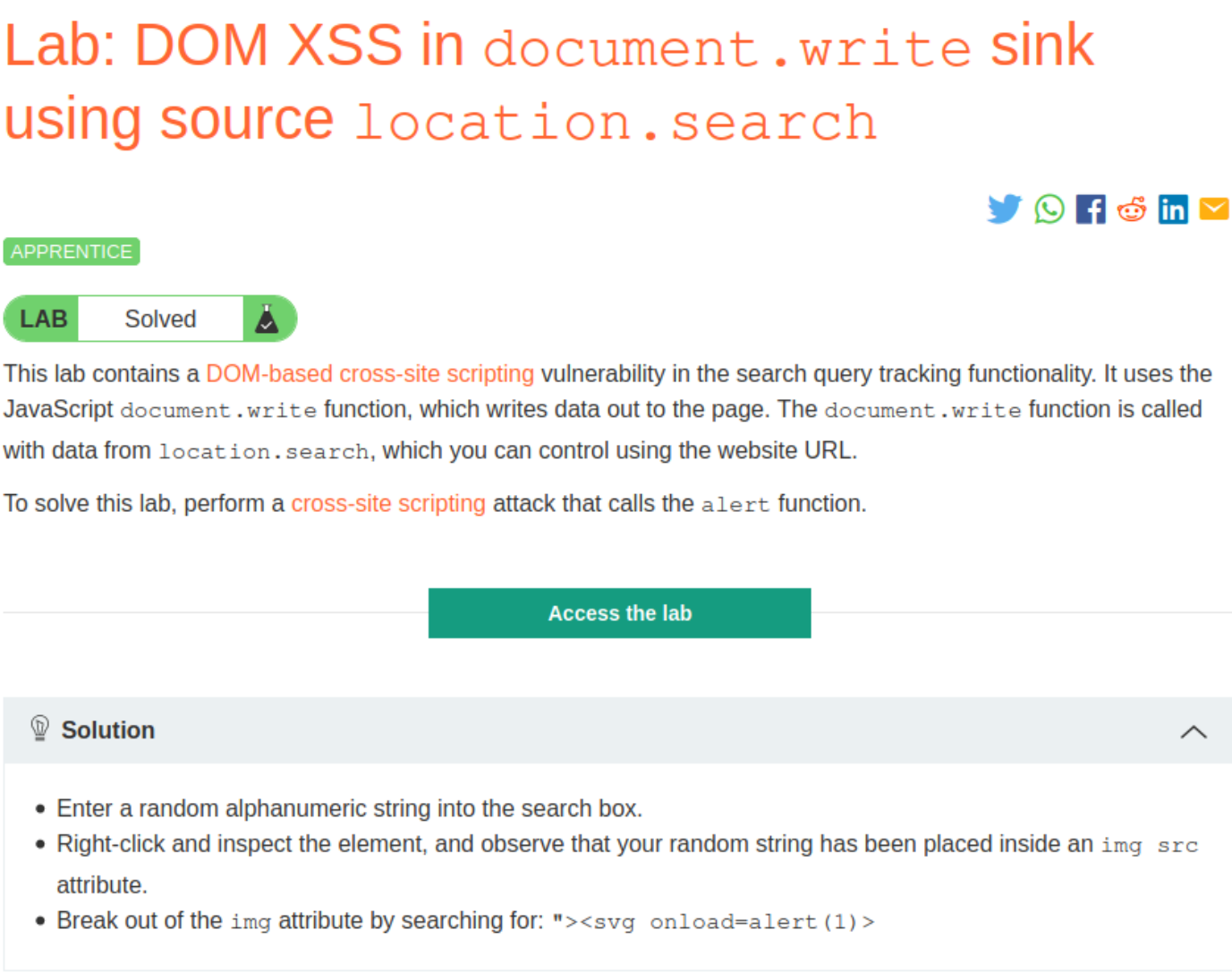
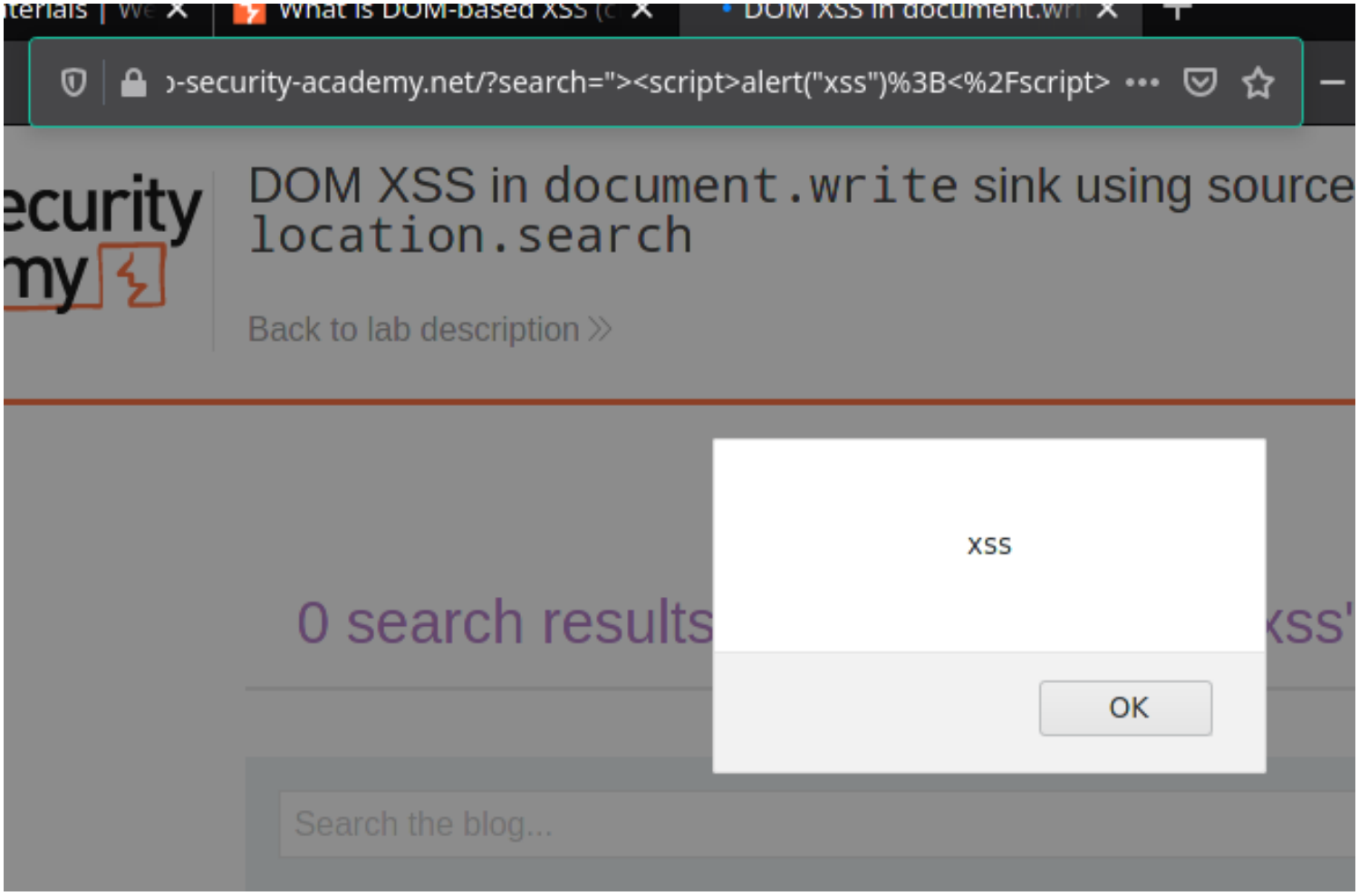
Exploiting DOM XSS with different sources and sinks

1.




-security-academy.net/?search="><script>alert("xss")%3B<%2Fscript>
DOM XSS in document.write sink using source location.search
Back to lab description
xss
OK
0 search results
Search the blog...

2.

Lab: DOM XSS in `document.write` sink using source `location.search` inside a select element

PRACTITIONER

LAB Solved

This lab contains a DOM-based cross-site scripting vulnerability in the stock checker functionality. It uses the JavaScript `document.write` function, which writes data out to the page. The `document.write` function is called with data from `location.search` which you can control using the website URL. The data is enclosed within a select element.

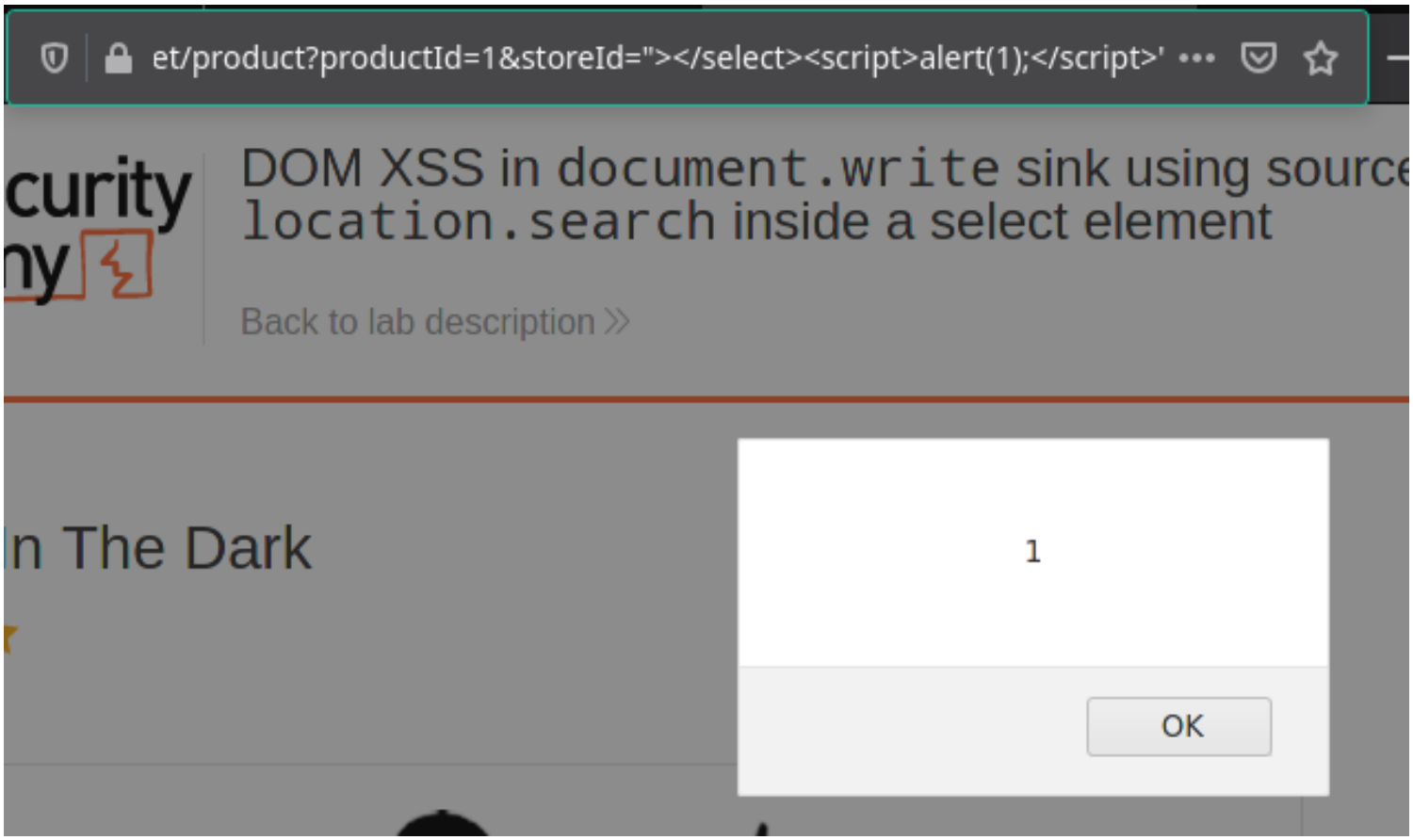
To solve this lab, perform a cross-site scripting attack that breaks out of the select element and calls the `alert` function.

Access the lab

Solution

- On the product pages, notice that the dangerous JavaScript extracts a `storeId` parameter from the `location.search` source. It then uses `document.write` to create a new option in the select element for the stock checker functionality.
- Add a `storeId` query parameter to the URL and enter a random alphanumeric string as its value. Request this modified URL.
- In your browser, notice that your random string is now listed as one of the options in the drop-down list.
- Right-click and inspect the drop-down list to confirm that the value of your `storeId` parameter has been placed inside a select element.
- Change the URL to include a suitable XSS payload inside the `storeId` parameter as follows:

```
product?productId=1&storeId="></select><img%20src=1%20onerror=alert(1)>
```



3.

The `innerHTML` sink doesn't accept `script` elements on any modern browser, nor will `svg onload` events fire. This means you will need to use alternative elements like `img` or `iframe`. Event handlers such as `onload` and `onerror` can be used in conjunction with these elements. For example:

```
element.innerHTML='... <img src=1 onerror=alert(document.domain)> ...'
```

Lab: DOM XSS in `innerHTML` sink using source `location.search`

APPRENTICE

LAB Solved

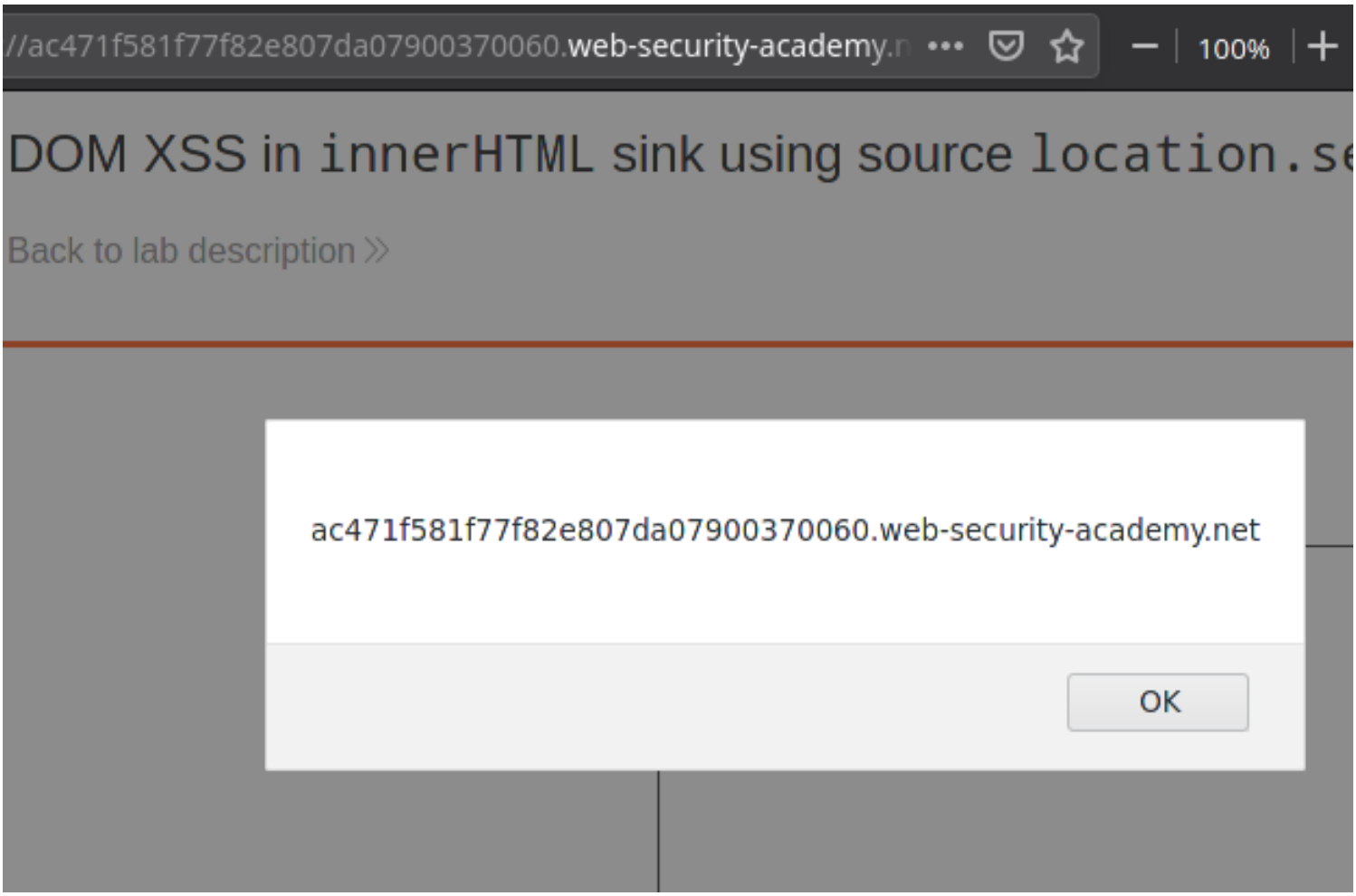
This lab contains a DOM-based cross-site scripting vulnerability in the search blog functionality. It uses an `innerHTML` assignment, which changes the HTML contents of a `div` element, using data from `location.search`.

To solve this lab, perform a cross-site scripting attack that calls the `alert` function.

Access the lab

Solution

- Enter the following into the into the search box: `<img src=1 onerror=alert(1)>`
- Click "Search".



4.

DOM XSS combined with reflected and stored data

Some pure DOM-based vulnerabilities are self-contained within a single page. If a script reads some data from the URL and writes it to a dangerous sink, then the vulnerability is entirely client-side.

However, sources aren't limited to data that is directly exposed by browsers - they can also originate from the website. For example, websites often reflect URL parameters in the HTML response from the server. This is commonly associated with normal XSS, but it can also lead to so-called reflected+DOM vulnerabilities.

In a reflected+DOM vulnerability, the server processes data from the request, and echoes the data into the response. The reflected data might be placed into a JavaScript string literal, or a data item within the DOM, such as a form field. A script on the page then processes the reflected data in an unsafe way, ultimately writing it to a dangerous sink.

```
eval('var data = "reflected string"');
```

_ab: Reflected DOM XSS

PRACTITIONER

LAB Solved

This lab demonstrates a reflected DOM vulnerability. Reflected DOM vulnerabilities occur when the server-side application processes data from a request and echoes the data in the response. A script on the page then processe the reflected data in an unsafe way, ultimately writing it to a dangerous sink.

To solve this lab, create an injection that calls the `alert()` function.

Access the lab

Solution

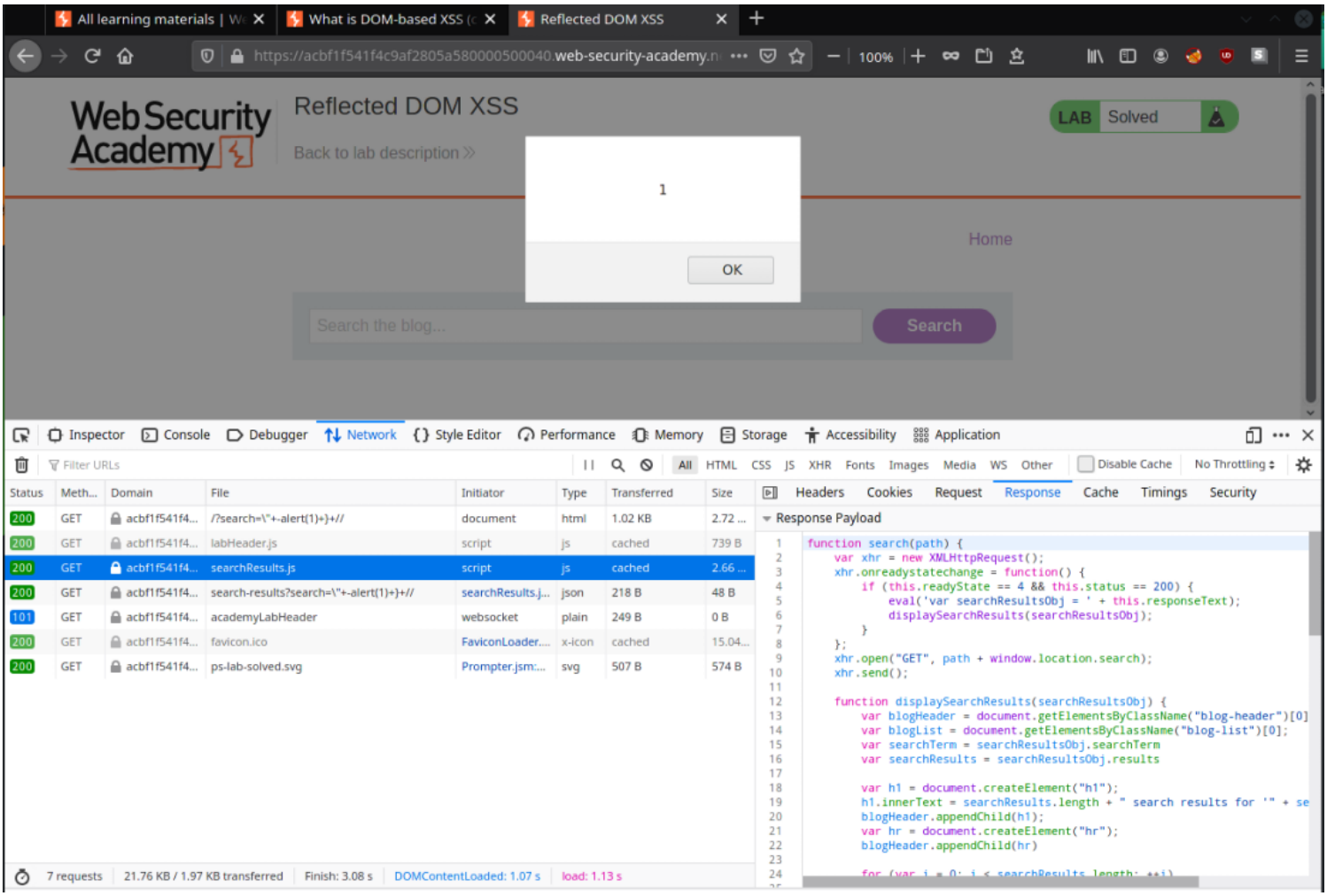
1. In Burp Suite, go to the Proxy tool and make sure that the Intercept feature is switched on.
2. Back in the lab, go to the target website and use the search bar to search for a random test string, such as `"XSS"`.
3. Return to the Proxy tool in Burp Suite and forward the request.
4. On the Intercept tab, notice that the string is reflected in a JSON response called `search-results`.
5. From the Site Map, open the `searchResults.js` file and notice that the JSON response is used with an `eval()` function call.
6. By experimenting with different search strings, you can identify that the JSON response is escaping quotation marks. However, backslash is not being escaped.
7. To solve this lab, enter the following search term: `\"-alert(1)}//`

As you have injected a backslash and the site isn't escaping them, when the JSON response attempts to escape the opening double-quotes character, it adds a second backslash. The resulting double-backslash causes the escaping to be effectively canceled out. This means that the double-quotes are processed unescaped, which closes the string that should contain the search term.

An arithmetic operator (in this case the subtraction operator) is then used to separate the expressions before the `alert()` function is called. Finally, a closing curly bracket and two forward slashes close the JSON object early and comment out what would have been the rest of the object. As a result, the response is generated as follows:

```
{"searchTerm":"\\"-alert(1)}//", "results":[]}
```

manipulating through some try and error:



5.

Websites may also store data on the server and reflect it elsewhere. In a stored+DOM vulnerability, the server receives data from one request, stores it, and then includes the data in a later response. A script within the later response contains a sink which then processes the data in an unsafe way.

```
element.innerHTML = comment.author
```

Lab: Stored DOM XSS

PRACTITIONER

LAB Solved

This lab demonstrates a stored DOM vulnerability in the blog comment functionality. To solve this lab, exploit this vulnerability to call the `alert()` function.

Access the lab

Solution

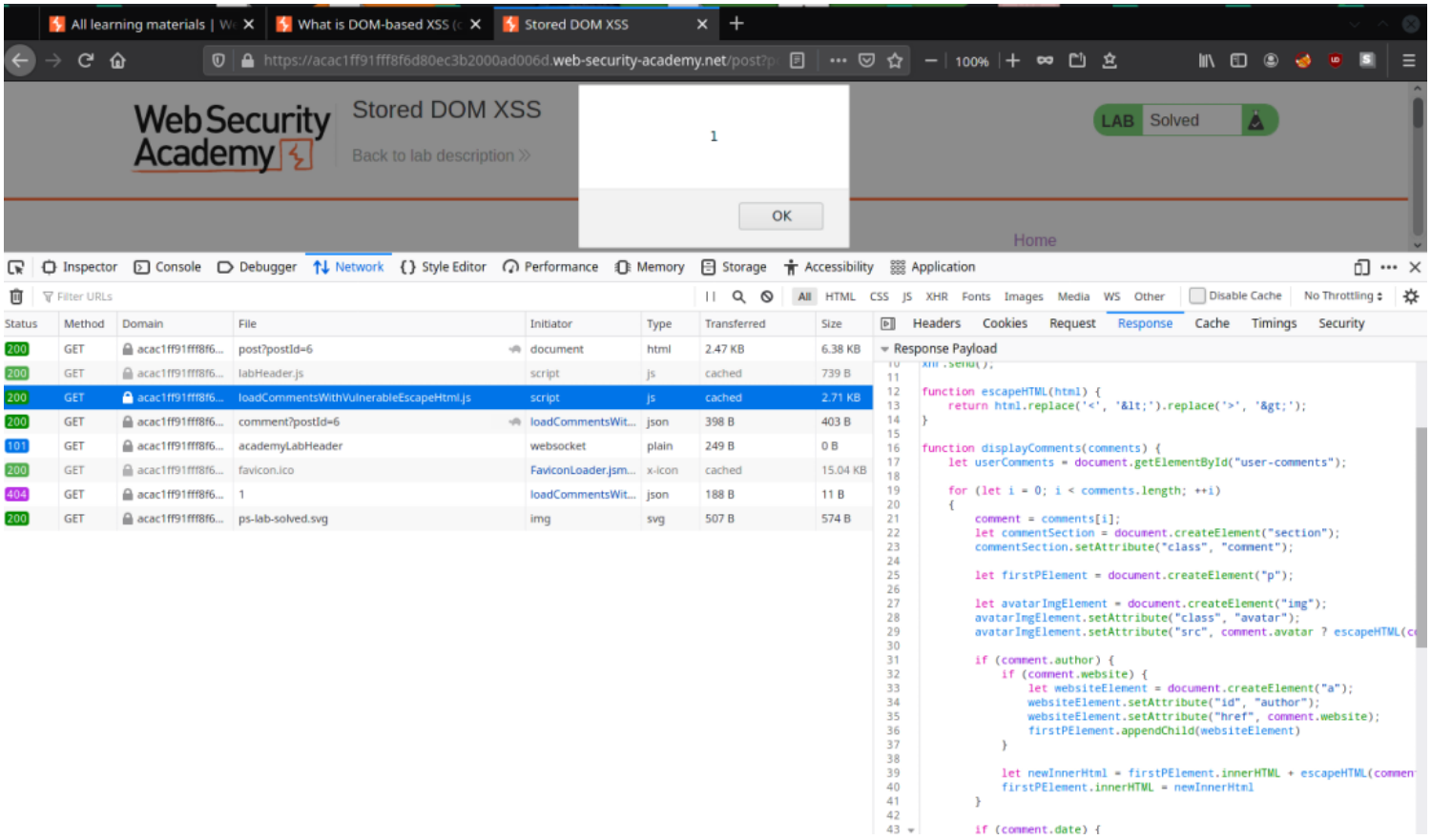
To solve this lab, create a comment with the following vector:

```
<><img src=1 onerror=alert(1)>
```

In an attempt to prevent XSS, the website uses the JavaScript `replace()` function to encode angle brackets. However, when the first argument is a string, the function only replaces the first occurrence. We exploit this vulnerability by simply including an extra set of angle brackets at the beginning of the comment. These angle brackets will be encoded, but any subsequent angle brackets will be unaffected, enabling us to effectively bypass the filter and inject HTML.

The value of the `src` attribute is invalid and throws an error. This triggers the `onerror` event handler, which then calls the `alert()` function. As a result, the payload is executed whenever the user's browser attempts to load the page containing your malicious post.

here , when atttempting it seems “<script>” tag is prohibited, so using "<img> tag



6.

If a JavaScript library such as jQuery is being used, look out for sinks that can alter DOM elements on the page. For instance, the `attr()` function in jQuery can change attributes on DOM elements. If data is read from a user-controlled source like the URL and then passed to the `attr()` function, then it may be possible to manipulate the value sent to cause XSS. For example, here we have some JavaScript that changes an anchor element's `href` attribute using data from the URL:

```
$(function(){
$('#backLink').attr("href",(new
URLSearchParams(window.location.search)).get('returnUrl'));
});
```

You can exploit this by modifying the URL so that the `location.search` source contains a malicious JavaScript URL. After the page's JavaScript applies this malicious URL to the back link's `href`, clicking on the back link will execute it:

```
?returnUrl=javascript:alert(document.domain)
```

Lab: DOM XSS in jQuery anchor `href` attribute sink using `location.search` source

APPRENTICE

LAB Solved

This lab contains a DOM-based cross-site scripting vulnerability in the submit feedback page. It uses the jQuery library's `$` selector function to find an anchor element, and changes its `href` attribute using data from `location.search`.

To solve this lab, make the "back" link alert `document.cookie`.

Access the lab

Solution

- On the Submit feedback page, change the query parameter `returnPath` to `/` followed by a random alphanumeric string.
- Right-click and inspect the element, and observe that your random string has been placed inside an a `href` attribute.
- Change `returnPath` to `javascript:alert(document.cookie)`, then hit enter and click "back".