**Homework – XSS**

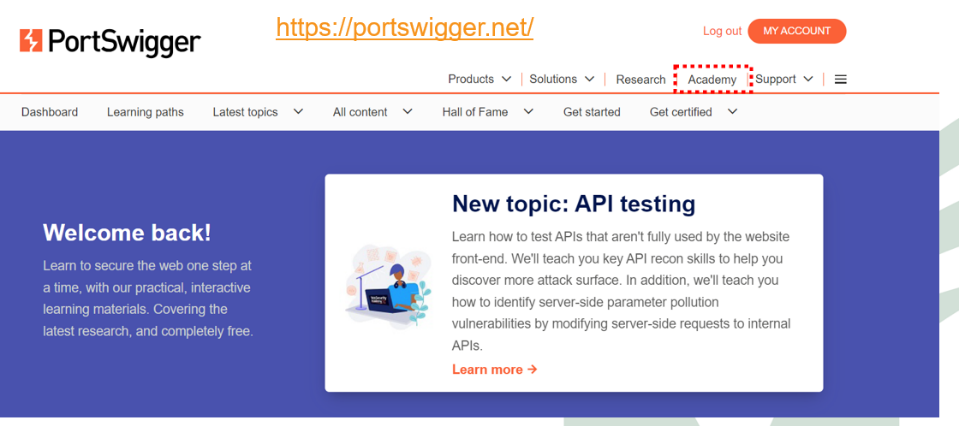
**Lab Goals:** This lab aims to provide students with a basic understanding of exploiting several XSS injection attacks using the PortSwigger Academy and Burp Suite Community.

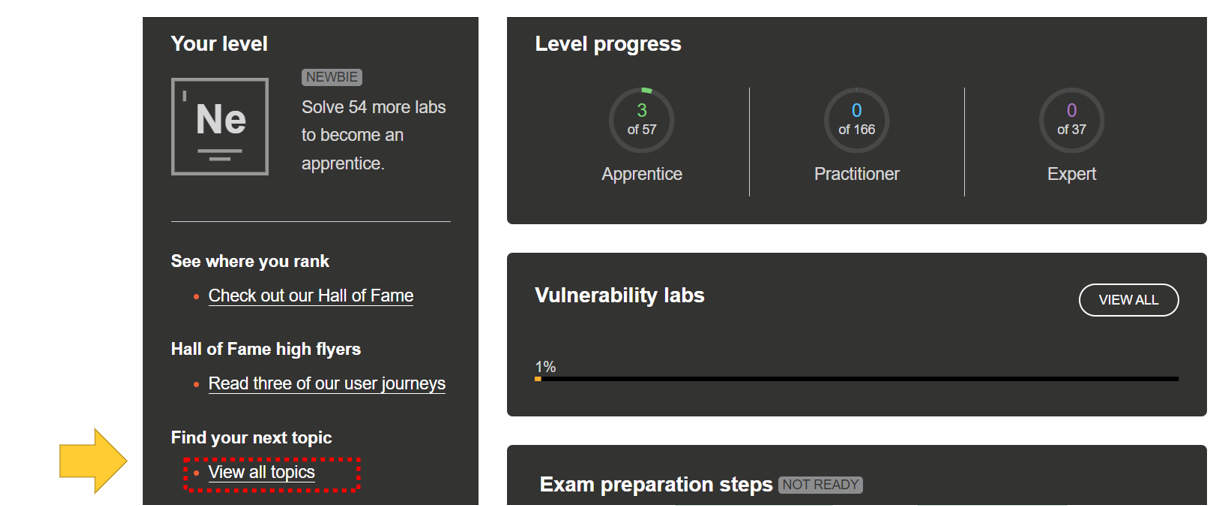
**Submission: There is no submission.**

**Rubric:** It is a non-graded activity.

**Task 1: Create an account and prepare the environment.**

**Goal:** Create an account in the PortSwigger Academy, which is required to access and execute the rest of the lab.

1. Create an account and log in to the Academy.
2. Scroll down and click on the link “View all topics”.



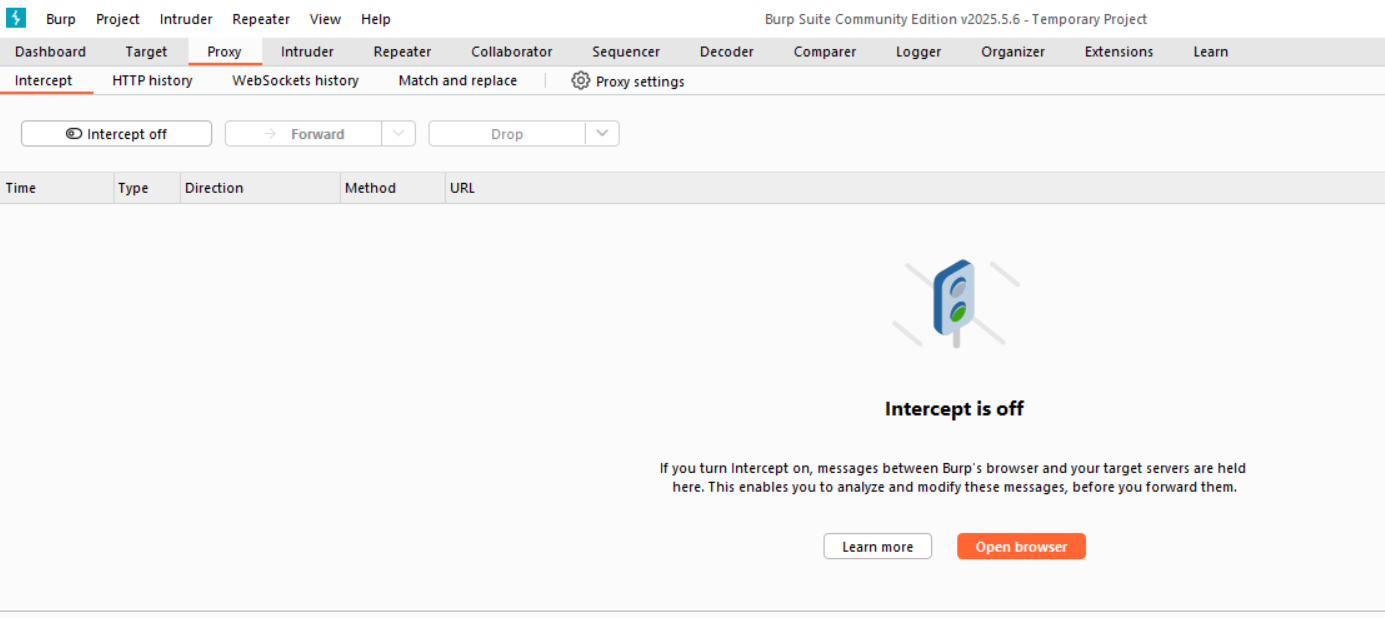
1. Download and install the **Burp Suite Community Edition**: <https://portswigger.net/burp/communitydownload>.
2. In the PortSwigger Academy, take this course: <https://portswigger.net/burp/documentation/desktop/getting-started>

**Task 2 - Reflected XSS into HTML context with nothing encoded.**

**Goal:** The lab aims to teach how to inject a simple reflected XSS vulnerability in the search page.

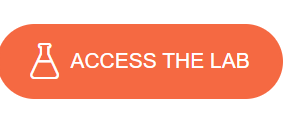
After you are logged in to PortSwigger Academy, start the Burp Suite and complete these tasks.

* In the Proxy tab, click on **Open Browser**.

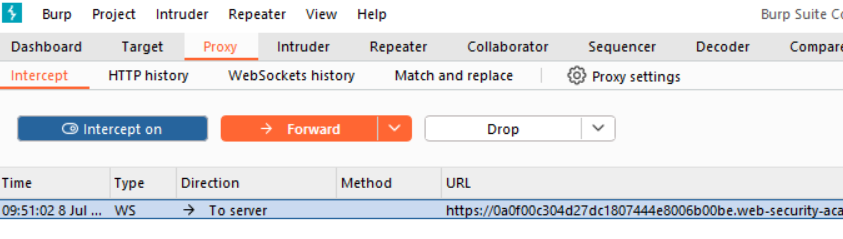


After you are logged in to PortSwigger Academy, start the Burp Suite and complete these tasks.

* In the Proxy tab, open this webpage using the embedded browser:
  + <https://portswigger.net/web-security/cross-site-scripting/reflected/lab-html-context-nothing-encoded>
  + On the page, click on the bottom to access the Lab to open the Exercise.



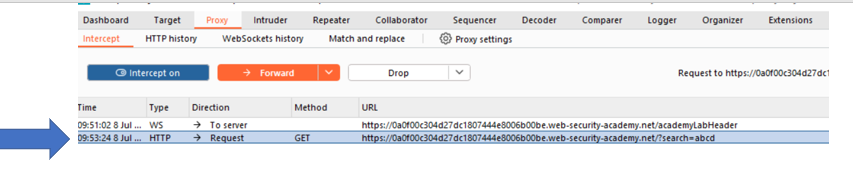
* Activate the interception in the Proxy tab.

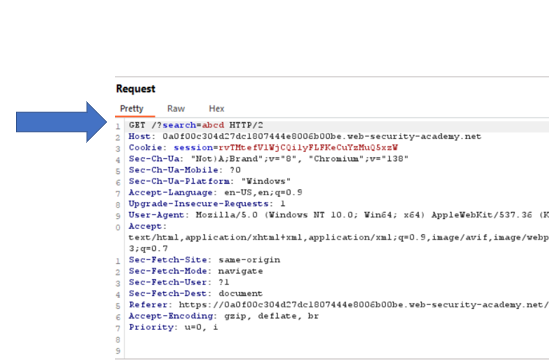


* Set a random string in the Search Bar and **press search**.

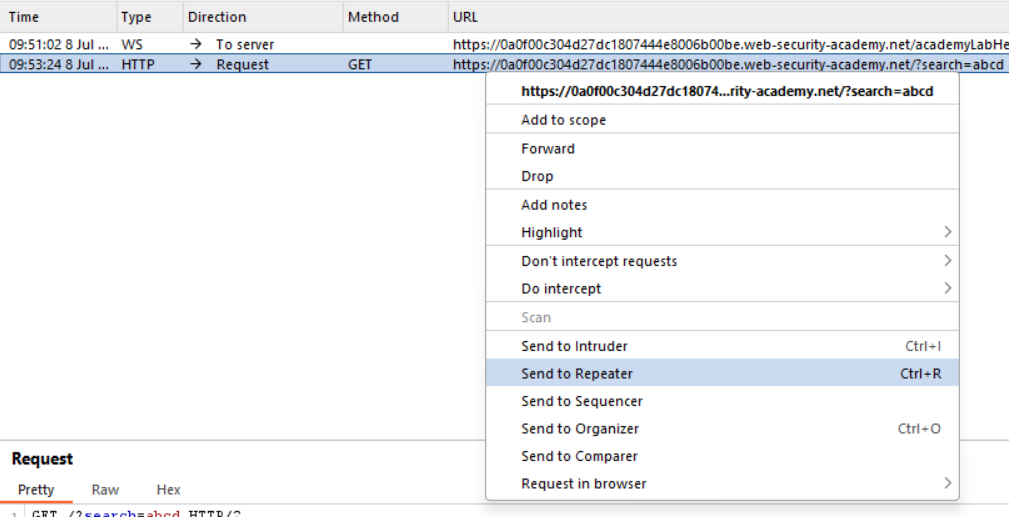


* Analyze the request in the Proxy (Check the request and its content).

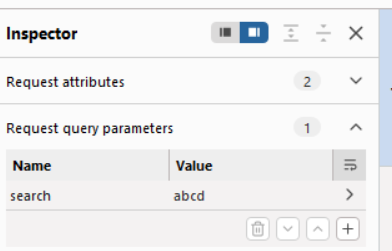




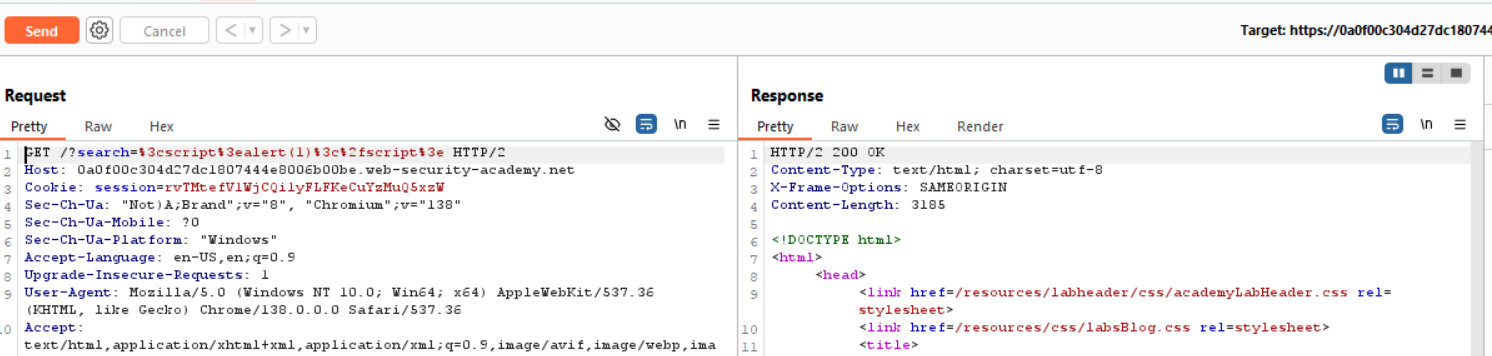
* + What is the type of HTTP request? GET
  + Does the message have a body? Why? No, usually a GET request does not have a body.
  + How does the message pass the parameter (content of the search field) to the server? They pass as a request parameter.
* Send the request to Repeater.



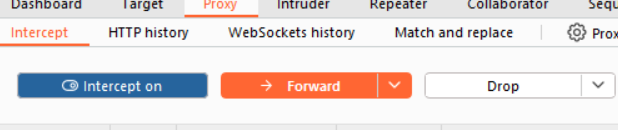
* In the Inspector inside the Repeater tab, find the request query parameter.



* Change this values for *<script>alert(1)</script>*.
* Press the **Sender** button and check the response.



* + What is the request error code? 200 OK
  + What does it mean? It means that “probably” this exploitation will work (the server does not detect anything wrong).
* Now, inside the Proxy tab, we will change the parameter in the **Inspector** by inserting the previous script and pressing the **Forward** button.



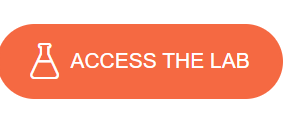


**Task 3 - Reflected XSS into HTML context with most tags and attributes blocked.**

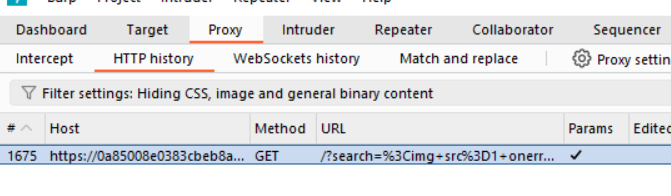
**Goal:** This lab aims to train students on how to bypass web application firewalls (WAFs) designed to prevent common Cross-Site Scripting (XSS) attacks.

After you are logged in to PortSwigger Academy, start the Burp Suite and complete these tasks.

* In the Proxy tab, open this webpage using the embedded browser:
  + <https://portswigger.net/web-security/cross-site-scripting/contexts/lab-html-context-with-most-tags-and-attributes-blocked>
  + On the page, click on the bottom to access the Lab to open the Exercise.



* **Activate the interception in the Proxy tab.**
* In the search tab type this: **<img src=1 onerror=alert(document.cookie)>,** click on **Search button,** and **forward** the message to see the result**.**
  + What is the response? “Tag is not allowed”.
* In the **HTTP history**, locate the request.



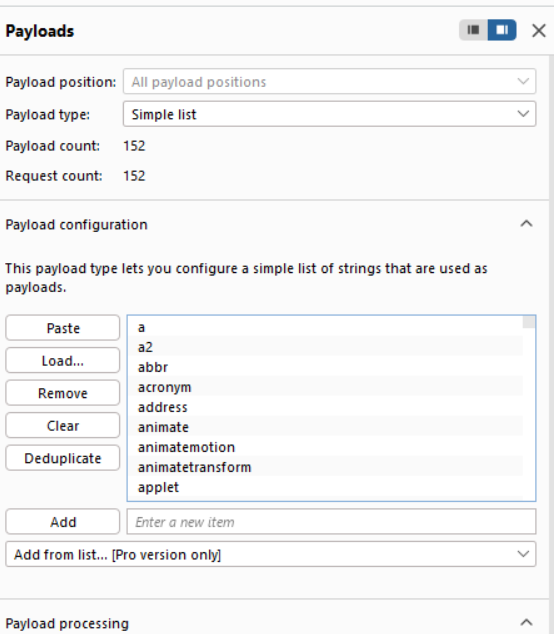
* Send it to Intruder and open the **Intruder** tab.
  + What is the Intruder in Burp Suite? Burp Intruder is a tool for automating customized attacks against web applications. It enables you to configure attacks that send the same HTTP request repeatedly, inserting different payloads into predefined positions each time.



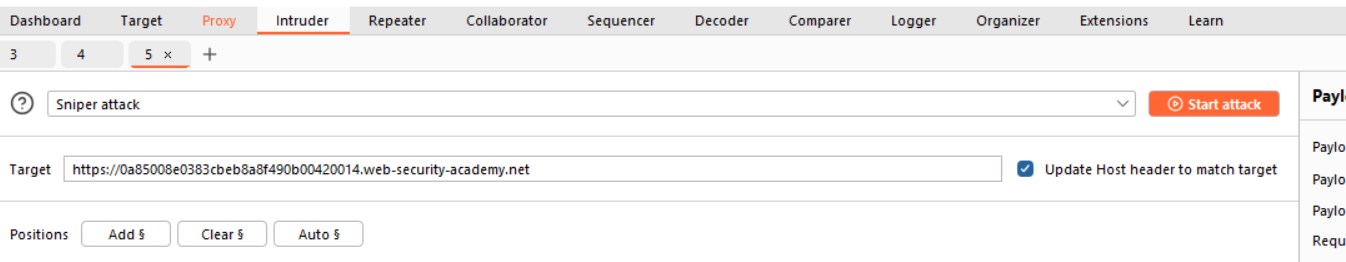
* Clean the parameters in the URL that you send to the server and click on the Add button[[1]](#footnote-1).



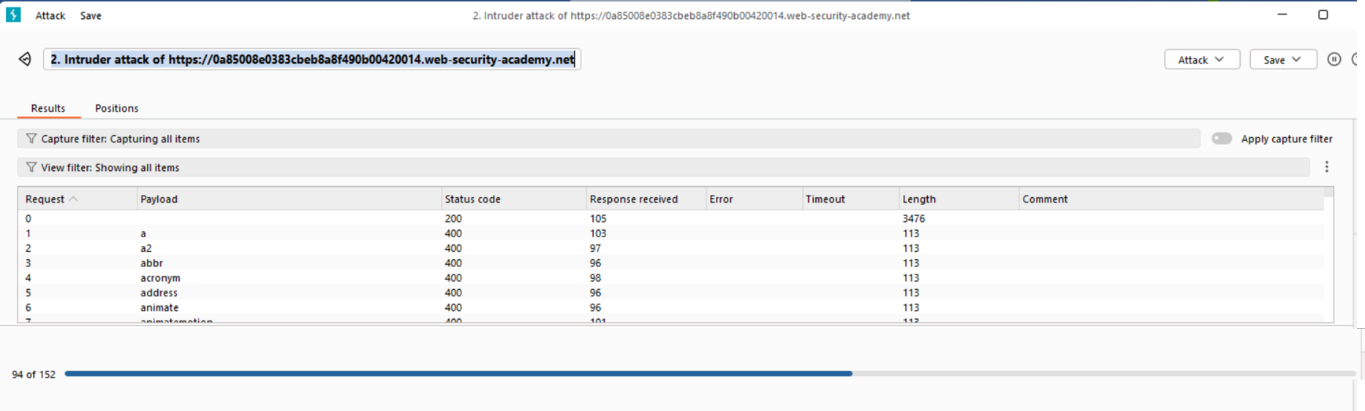
* Open the XSS cheat sheet, and click **Copy tags** to clipboard.
  + What is the Port Swigger cheat sheet? This cheat sheet contains many vectors that can help you bypass WAFs and filters. You can select vectors by the event, tag, or browser, and a proof of concept is included for every vector.
* In the **Payloads** section, paste the clipboard content inside the **Payload Configuration**.



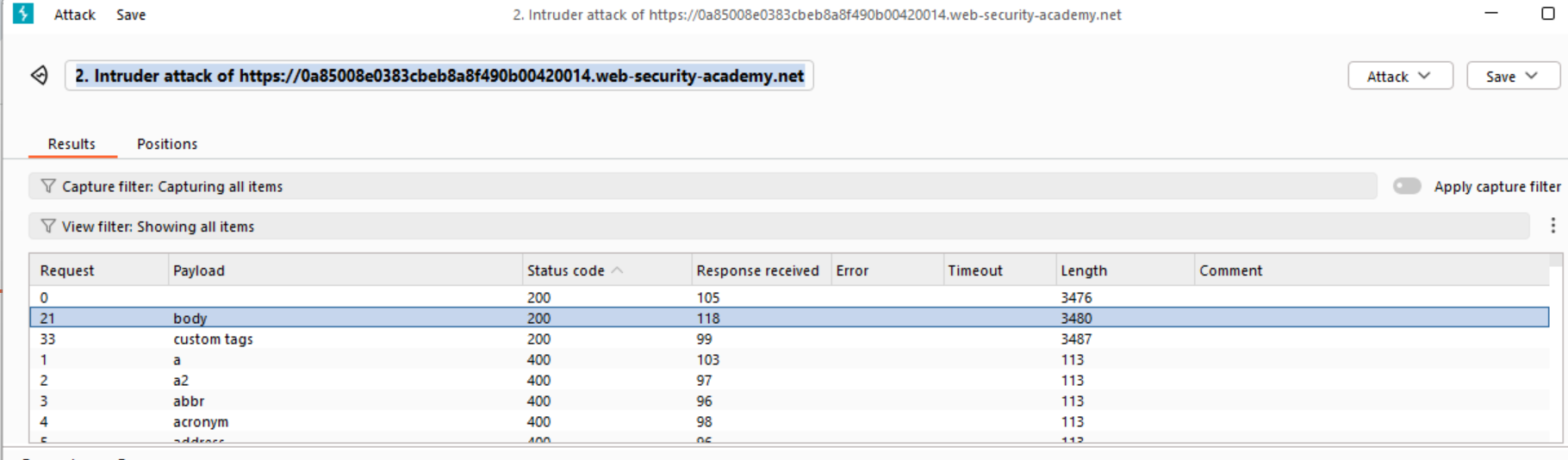
* **Start the attack**.



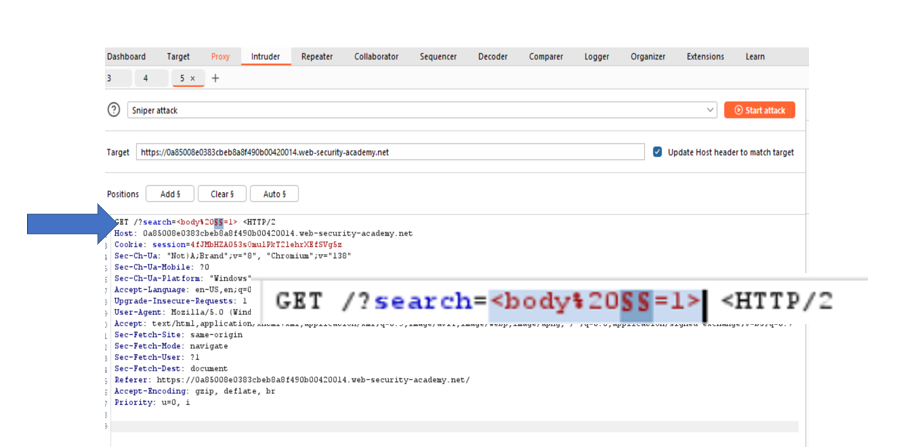
* + What does the start attack button do? It initiates the configured attack against the target web application.
    - Automated Request Sending
    - Payload Insertion
  + The Intruder version in the Community tools is a demo.



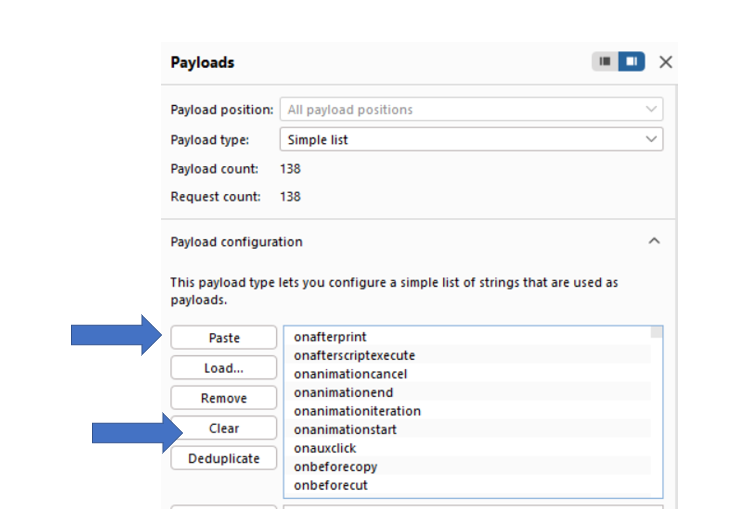
* After the command finishes its execution, we will analyze the payloads for which the response is 200 OK.
  + When the attack is finished, review the results. Note that most payloads caused a 400 response, but the body payload caused a 200 response.



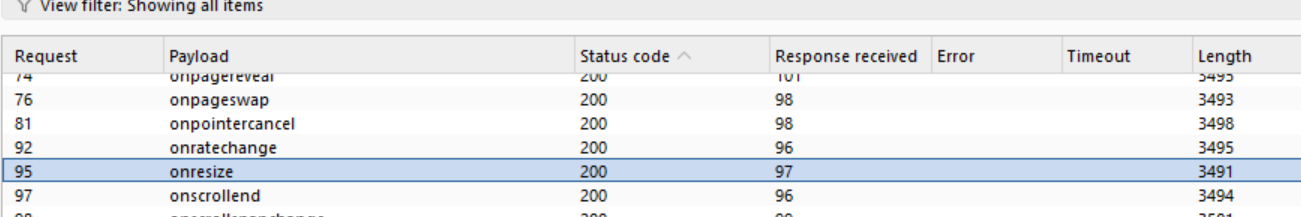
* Back to the Intruder, and in the same URL, add the following string.



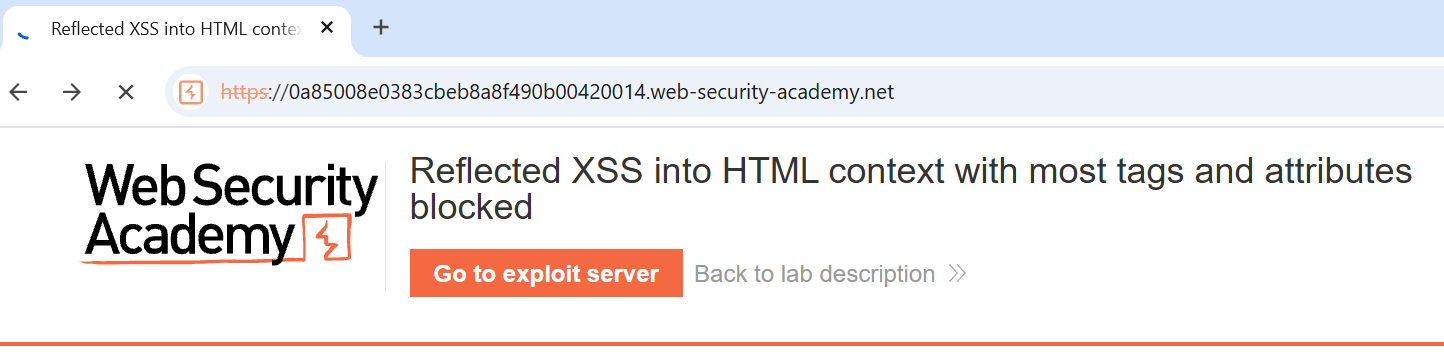
* Return to the cheat sheet and click on 'Copy events to clipboard.'
* In the **Payload Configuration**, clear all the information and paste the latest one. 🡪 **Start the attack again**.



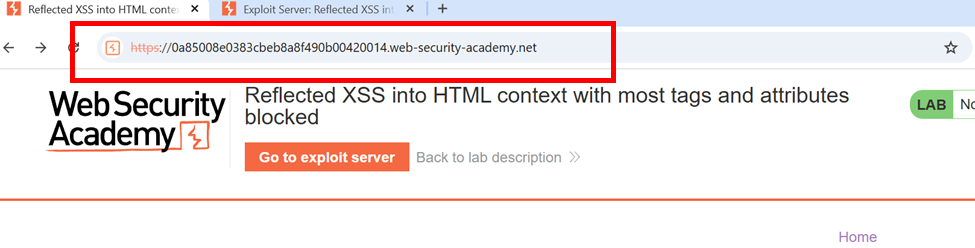
* When the attack is finished, review the results. Note that most payloads resulted in a 400 response, but the **onresize** payload yielded a 200 response.



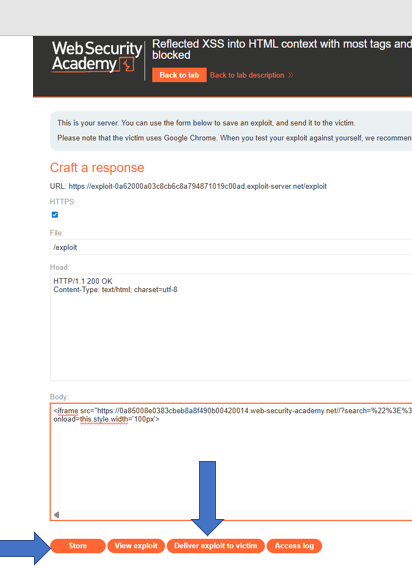
* Stop the Interceptor in the Proxy tab.
* Go to the website server and click on the **Go to Exploit server button**.
  + Exploit Server: This server is provided with a controlled and safe environment, so you do not need to deploy multiple requirements to exploit the compromised server.
    - The need to set up your own public-facing web server.
    - The need to configure your own logging mechanism to capture data from victims.
    - The hassle of acquiring a domain name and setting up DNS.



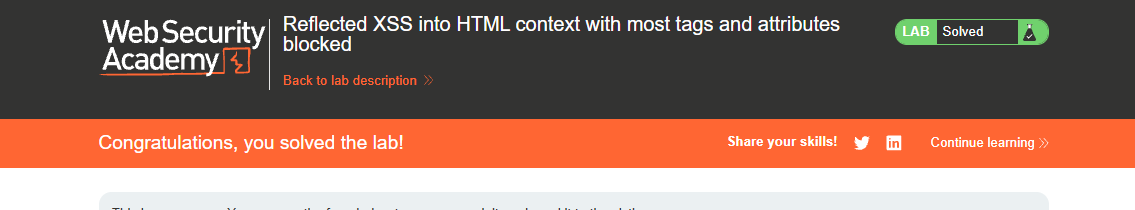
* In the body of the message, type this string: <iframe src="https://YOUR-LAB-ID.web-security-academy.net/?search=%22%3E%3Cbody%20onresize=print()%3E" onload=this.style.width='100px'>
  + The IP address of your lab.



* + Remember that this string decoded is:
    - <iframe src="https://0a85008e0383cbeb8a8f490b00420014.web-security-academy.net//?search="><body onresize=print()>" onload=this.style.width='100px'>



* Store the exploit payload (for future use) and display it.
* Finally, send it to the Victim. You will see that you finish the activity.



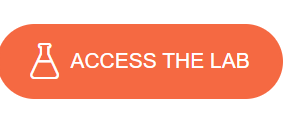
|  |
| --- |
| **Important Observation**  When you click the "Send to the victim" button on the PortSwigger exploit server (like the one you linked: https://exploit-0a62000a03c8cb6c8a794871019c00ad.exploit-server.net), the following usually happens in the context of their web security labs:   1. **Simulated Victim Interaction:** The lab environment simulates a "victim user" (e.g., a browser session belonging to an administrator or another user). 2. **Delivering the Exploit:** The exploit server constructs a URL, often incorporating the malicious payload you've designed (e.g., your XSS payload hosted on the exploit server itself, or a URL that triggers the vulnerability in the main lab application). This crafted URL is then "sent" to the simulated victim. 3. **Victim Browses the URL:** The simulated victim's browser is made to navigate to this crafted URL. 4. **Exploit Execution:**    * If the URL points back to the vulnerable lab application with your XSS payload, the victim's browser will render the page, and if the vulnerability is successfully exploited, your injected script will execute in the context of the victim's browser.    * If your exploit involves fetching resources from the exploit server (e.g., a JavaScript file you've stored there), the victim's browser will request that resource, and your exploit code will run. |

**Task 3: DOM XSS in document.write sink using source location.search.**

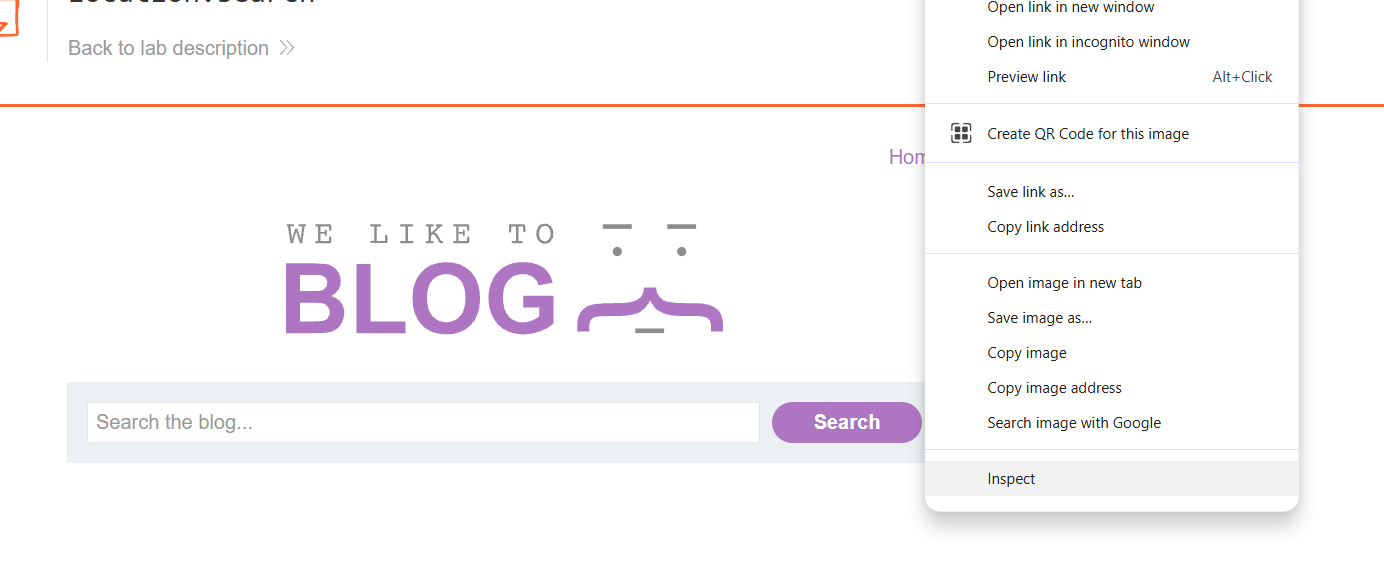
**Goal:** The goal of the lab is to perform a Cross-Site Scripting (XSS) attack that successfully calls the alert() function. This is achieved by exploiting a DOM-based XSS vulnerability within the website's search query tracking functionality. Specifically, the lab demonstrates how the JavaScript document.write function can be insecurely used with data directly taken from location.search (the query string of the URL), allowing for the injection of malicious scripts.

After you are logged in to PortSwigger Academy, start the Burp Suite and complete these tasks.

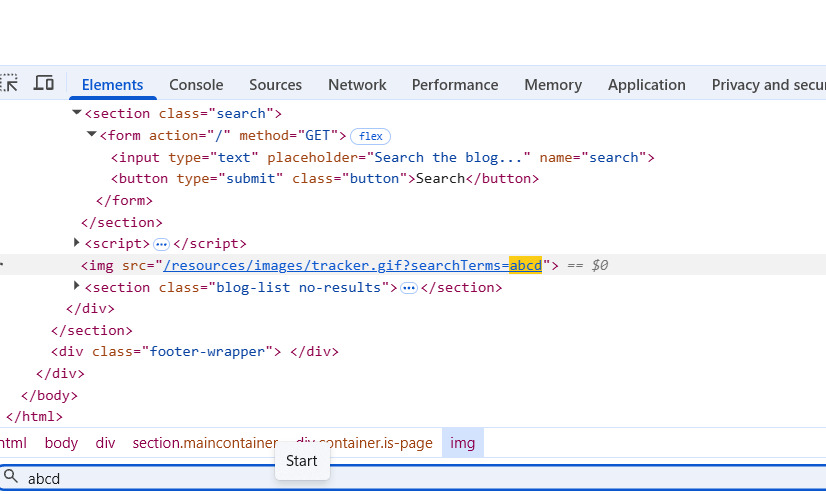
* In the Proxy tab, open this webpage using the embedded browser:
  + <https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-document-write-sink>
  + On the page, click on the bottom to access the Lab to open the Exercise.



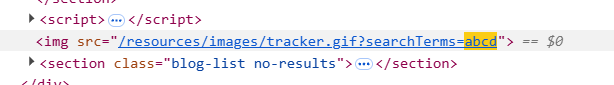
* Open the browser's Inspector.



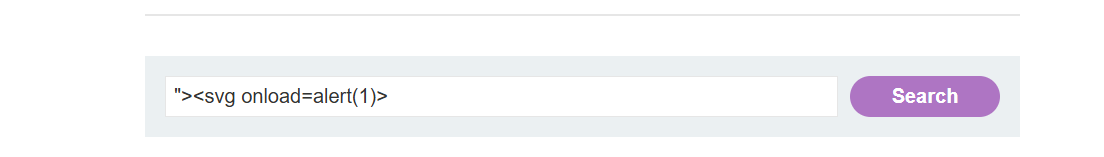
* We will analyze the page and try to find how this component works. Type any random string, like “abcd”, and in the Elements tab, find it in the inspector.

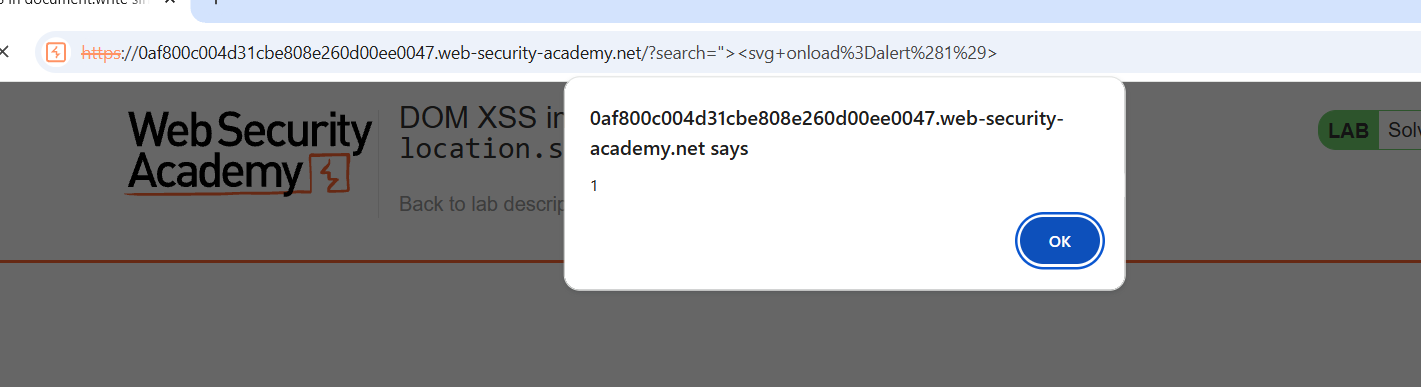


* We will see that the page inserts the query in an image component.



* Now, we can try to inject malicious code into the component, changing the string for a malicious script.
  + *"><svg onload=alert(1)>*
  + What does this script perform? *This script is an XSS (Cross-Site Scripting) payload. It's designed to be injected into a web page where it will break out of an existing HTML context and then execute JavaScript.*
  + Why don't you send the full script tag? *You will start closing the searchTerms parameters and inject the malicious script immediately after.*

**

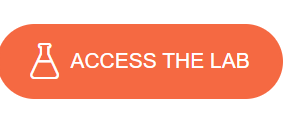
**

**Task 4: DOM XSS in document.write sink using source location.search inside a select element.**

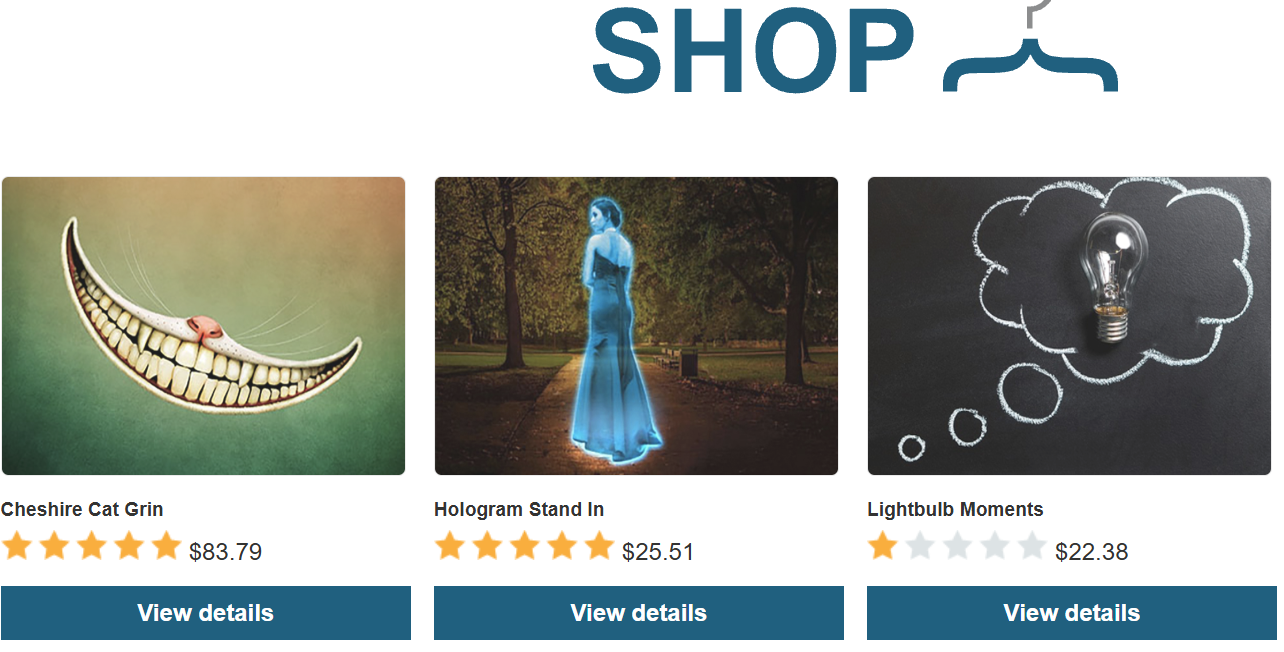
**Goal:** The goal of the lab is to perform a Cross-Site Scripting (XSS) attack that successfully calls the alert() function. This particular lab presents a challenge: the DOM-based XSS vulnerability is within the stock checker functionality, and the injected data (from location.search via document.write) is placed inside a <select> HTML element. Therefore, the specific objective is to break out of the <select> element to execute your XSS payload and trigger the alert() function.

After you are logged in to PortSwigger Academy, start the Burp Suite and complete these tasks.

* In the Proxy tab, open this webpage using the embedded browser:
  + <https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-document-write-sink-inside-select-element>
  + On the page, click on the bottom to access the Lab to open the Exercise.



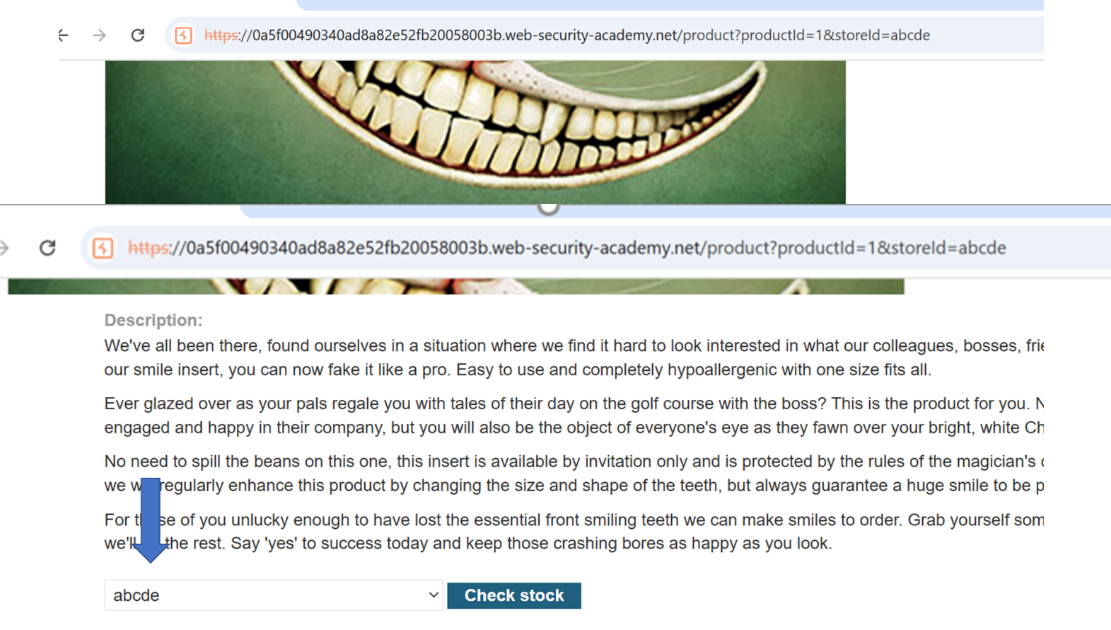
* Select any product and open it (**View Details**).



* On the detail page, verify that a button labeled **'Check Stock**' is present. Click the right button to inspect the element. We can see that there is a form, and it is what we will analyze.



* What is the problem with this form?
  + document.write('<option selected>'+store+'</option>');
    - Untrusted Input from location.search: The store variable directly takes its value from window.location.search (specifically, the storeId parameter in the URL query string) using URLSearchParams. This means an attacker can control the value of store by manipulating the URL.
    - document.write() Sink: The document.write() function is a common XSS "sink." It takes a string as an argument and directly writes it into the HTML document. When untrusted input is passed to document.write(), it creates a high risk of injection.
    - No Input Sanitization/Encoding: The store variable's value is concatenated directly into the HTML string without any form of sanitization, encoding, or escaping.
* Now, we will exploit the web server. We will return to the page and change the parameter, including the following script after the product ID: **&storeId=abcde**.

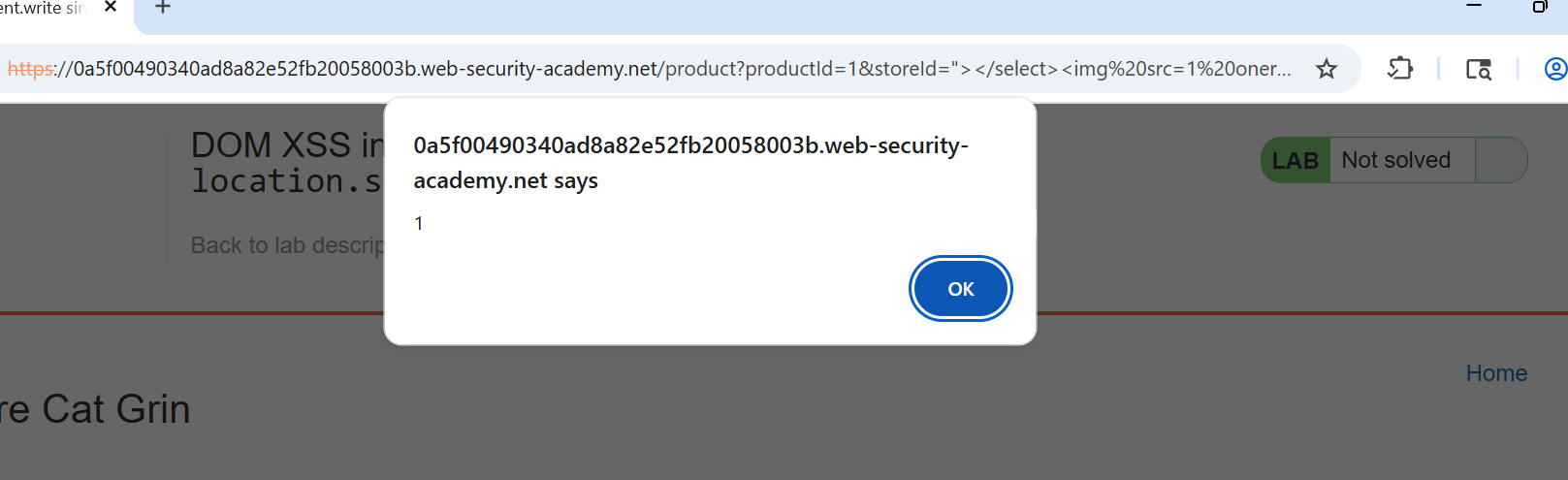


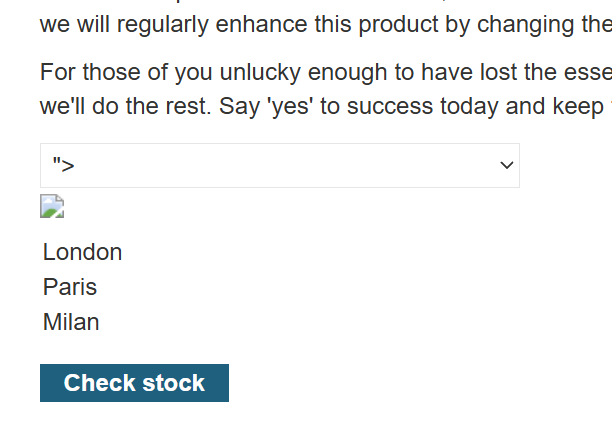
* We can also see that we change the page information. Using the Inspect, we can make a double check of this change.



* After this confirmation, we can inject the malicious code.
  + First, we need to encode the script: <img src=1 onerror=alert(1)>, generating <img%20src=1%20onerror=alert(1)>
  + Next, we concatenate this string with the following string: "></select>, generating this string: "></select><img%20src=1%20onerror=alert(1)>
* Now, we need to replace the ID for this string: product?productId=1&storeId="></select><img%20src=1%20onerror=alert(1)>







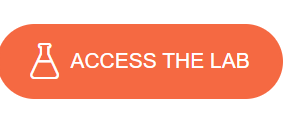
**EXTRA EXERCISE**

**Task 5: Stored DOM XSS**

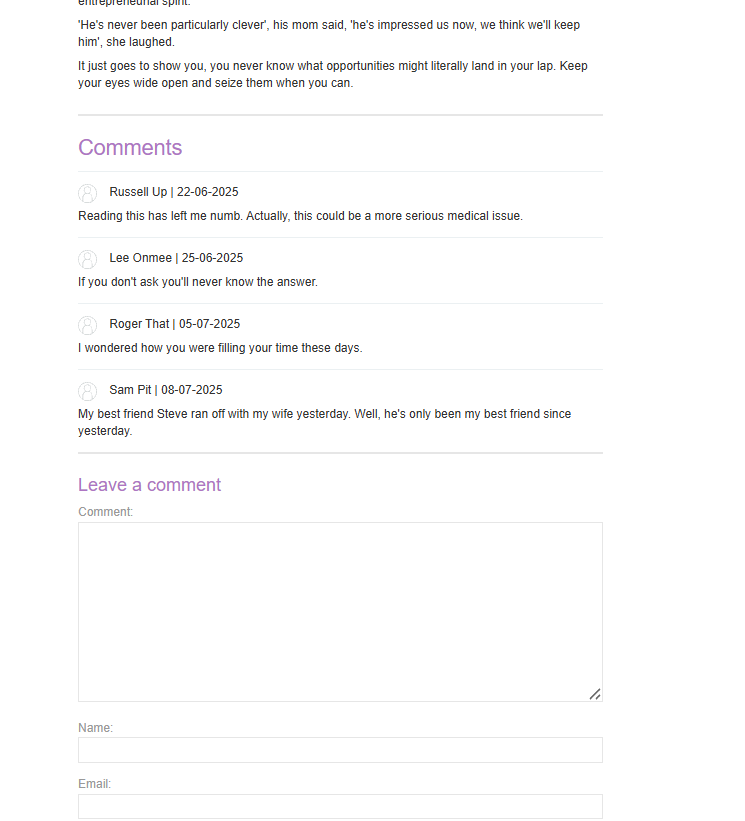
**Goal:** The goal of the lab is to exploit a stored DOM-based Cross-Site Scripting (XSS) vulnerability in the blog comment functionality to invoke the alert() function successfully. Unlike reflected XSS where the payload is immediately bounced back, "stored" XSS means your malicious input will be saved by the application (in this case, in a blog comment) and then rendered to other users (or yourself on a subsequent visit), triggering the XSS when their browser executes the vulnerable client-side code.

After you are logged in to PortSwigger Academy, start the Burp Suite and complete these tasks.

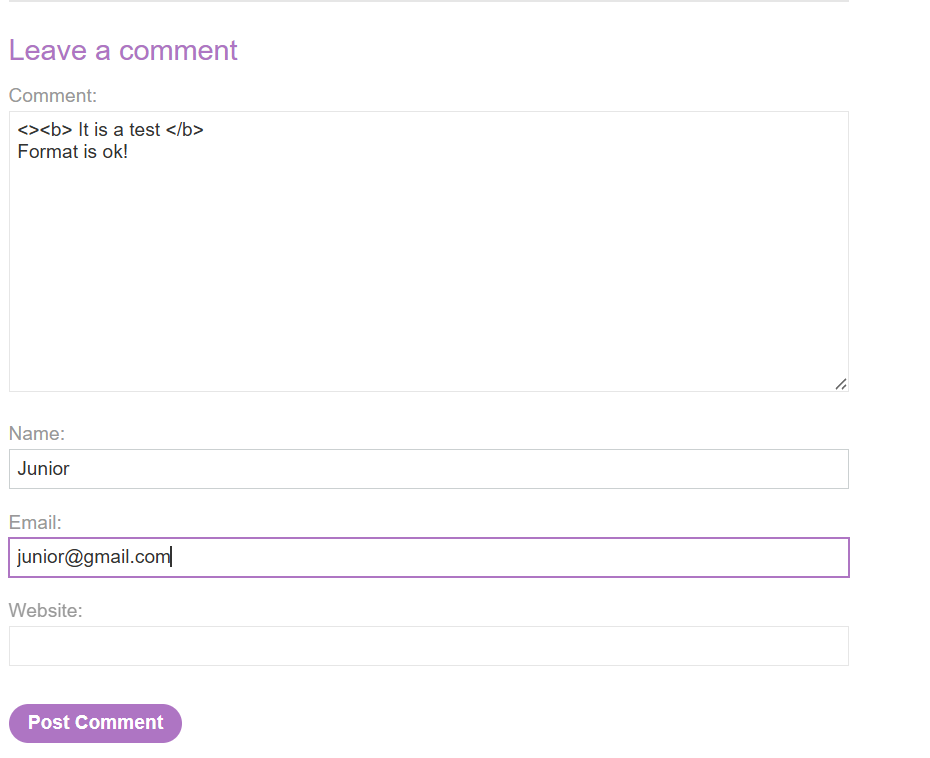
* In the Proxy tab, open this webpage using the embedded browser:
  + <https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-dom-xss-stored>
  + On the page, click on the bottom to access the Lab to open the Exercise.



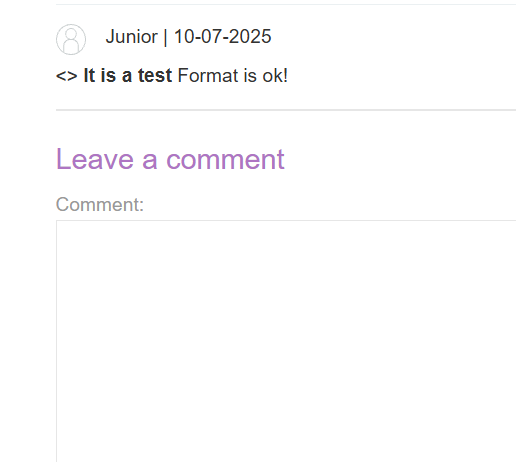
* In the exercise page, open any post.



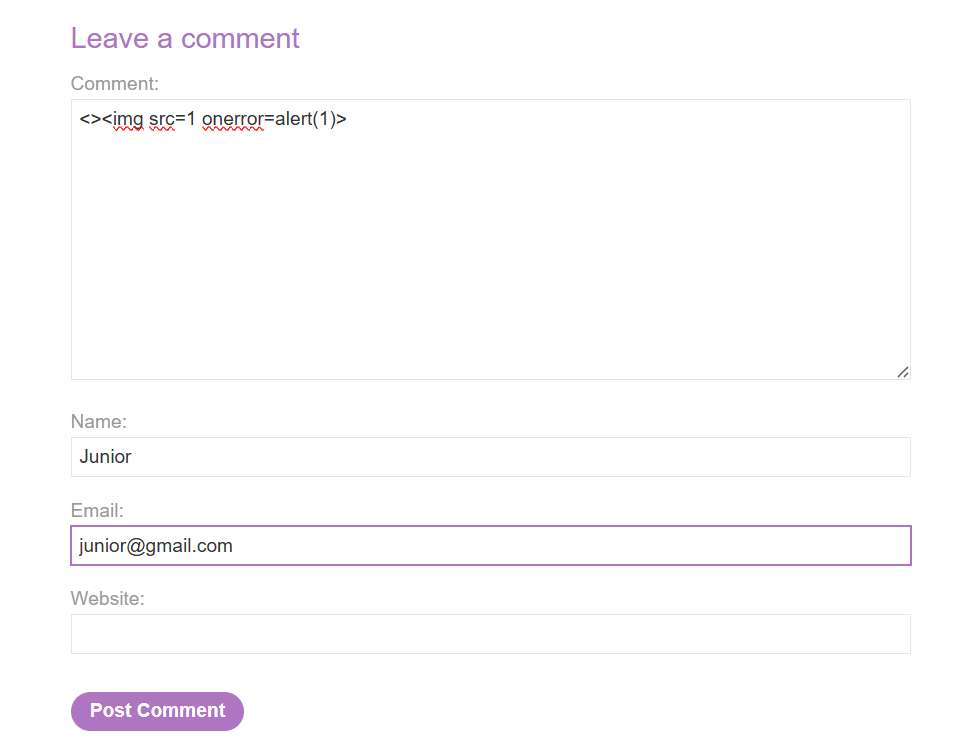
* Our goal is to exploit a server that is vulnerable to a stored XSS attack. A good test (**doesn't always work**) is to check if the page processes a special format, indicating that the **page does not parse the input**. We can check it, creating a comment to test it, including the tag <b>; as for example: **<b> format HTML test**.
  + The first <>< is a trick — it closes any open tag and ensures valid HTML parsing. It's used to:
    - Break out of improperly escaped HTML.
    - Ensure the browser starts fresh with a valid tag like <img>.
  + **What <>… Does?**
    - The first < starts a tag (even if it's junk).
    - The second > closes any unclosed tag or attribute.



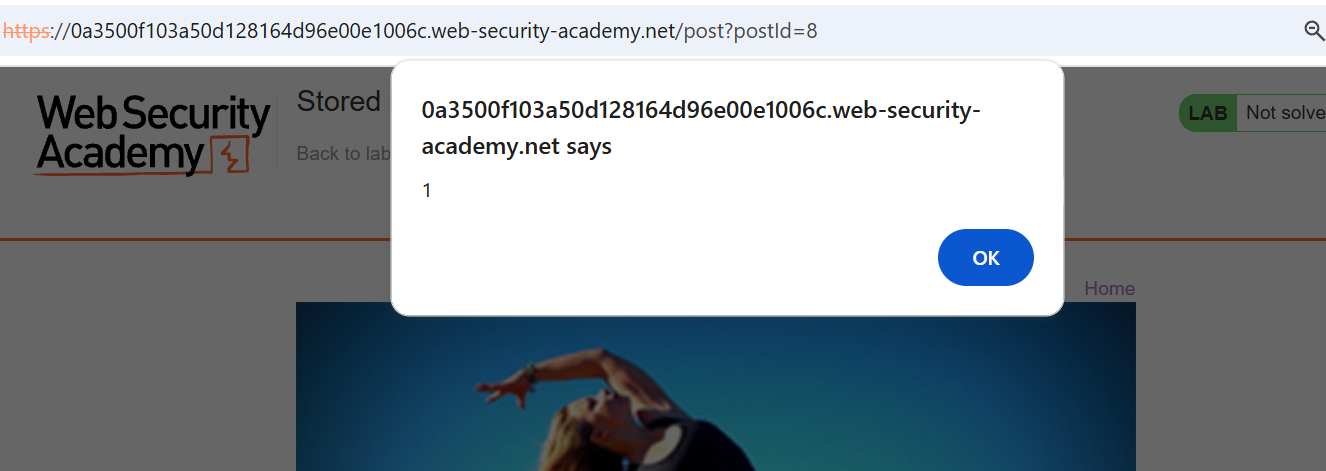
* If our test works, the post will be in bold format.



* **Good news; it is a good indication that the server does not handle the users' inputs well.**
* Now, we need to test the malicious script: **<><img src=1 onerror=alert(1)>**, posting it using the interface.



* You see, when you open the blog’s main page again, a pop-up is fired.



* Close the pop-up, and you will see that the Lab is finished.



1. In the context of Burp Suite's Intruder tool, the § (section sign) character is used to mark payload positions within your base request. [↑](#footnote-ref-1)