TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

WEB SECURITY

Content

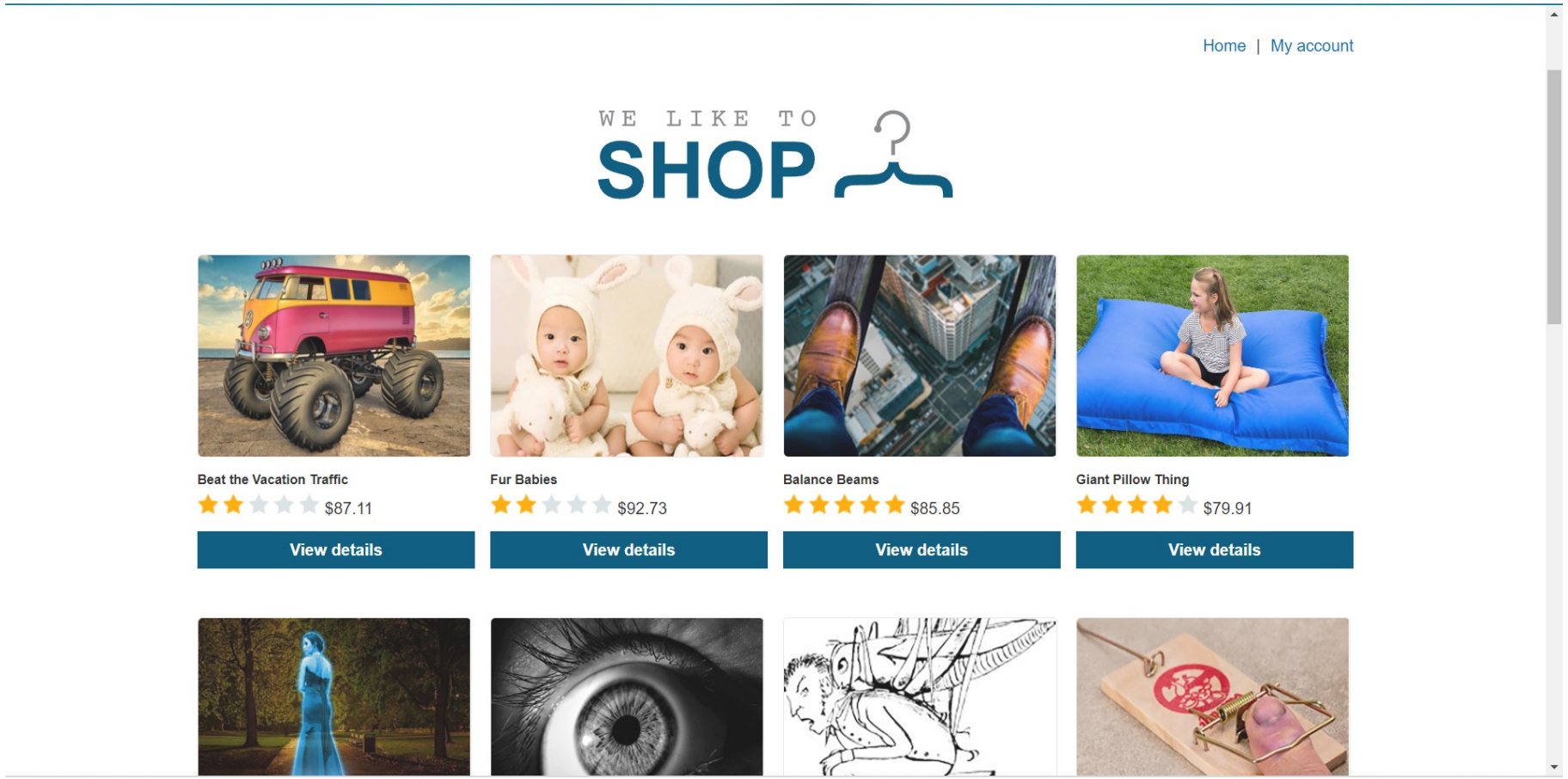
1. SQL Injection
2. XSS

Content

1. **SQL Injection**
2. XSS

SQL Injection

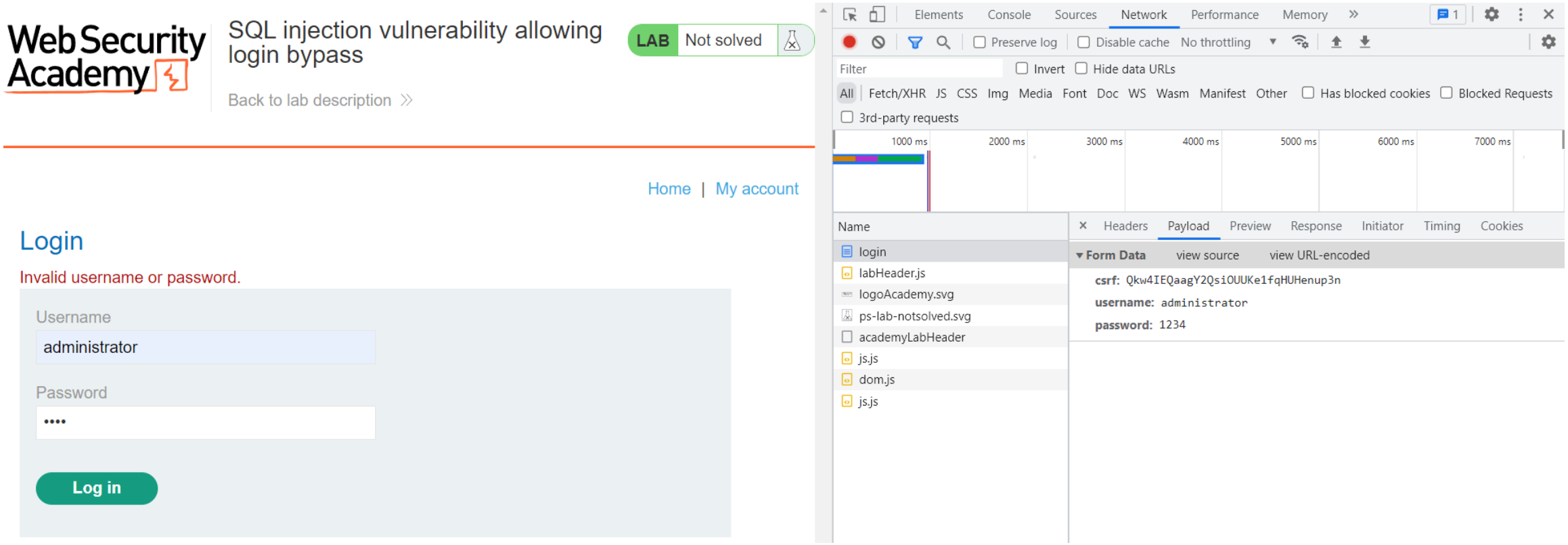
Eg: Perform an SQL injection attack that logs in to the application as the `administrator` user.



SQL Injection

Step 1: Open Devtools → Network to intercept http request, then try login with username is `administrator` and password is anything.

👉 We can see the browser send a http request with payload have 2 fields that we just enters.



SQL Injection

Step 2: From http request, we can guess that SQL query will be like:

```
SELECT * FROM USER WHERE username='administrator' AND password='1234'
```

Step 3: So if we enter username with value `administrator'--` the query will be like:

```
SELECT * FROM USER WHERE username='administrator' -- 'and password='1234'
```

👉 In SQL, after "--" is comment so the query will select the account with username is `administrator` without any password

SQL Injection

Step 4: We can test our predictions:

Web Security Academy

SQL injection vulnerability allowing login bypass

LAB Solved

Back to lab description

Congratulations, you solved the lab!

Share your skills!

Continue learning

Home | My account | Log out

My Account

Your username is: administrator

Email

Update email

Elements Console Sources Network Performance Memory

Preserve log Disable cache No throttling

Filter Invert Hide data URLs

All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other Has blocked cookies Blocked Requests

3rd-party requests

1000 ms 2000 ms 3000 ms 4000 ms 5000 ms 6000 ms 7000 ms 8000 ms 9000 ms

Name

login

my-account

labHeader.js

logoAcademy.svg

ps-lab-notsolved.svg

academyLabHeader

ps-lab-solved.svg

js.js

dom.js

js.js

Headers Payload Preview Response Initiator Timing Cookies

Form Data view source view URL-encoded

csrf: kt29am4qeFKsQkFG4vKu0HpWabHYSYHn

username: administrator'--

password: 1234

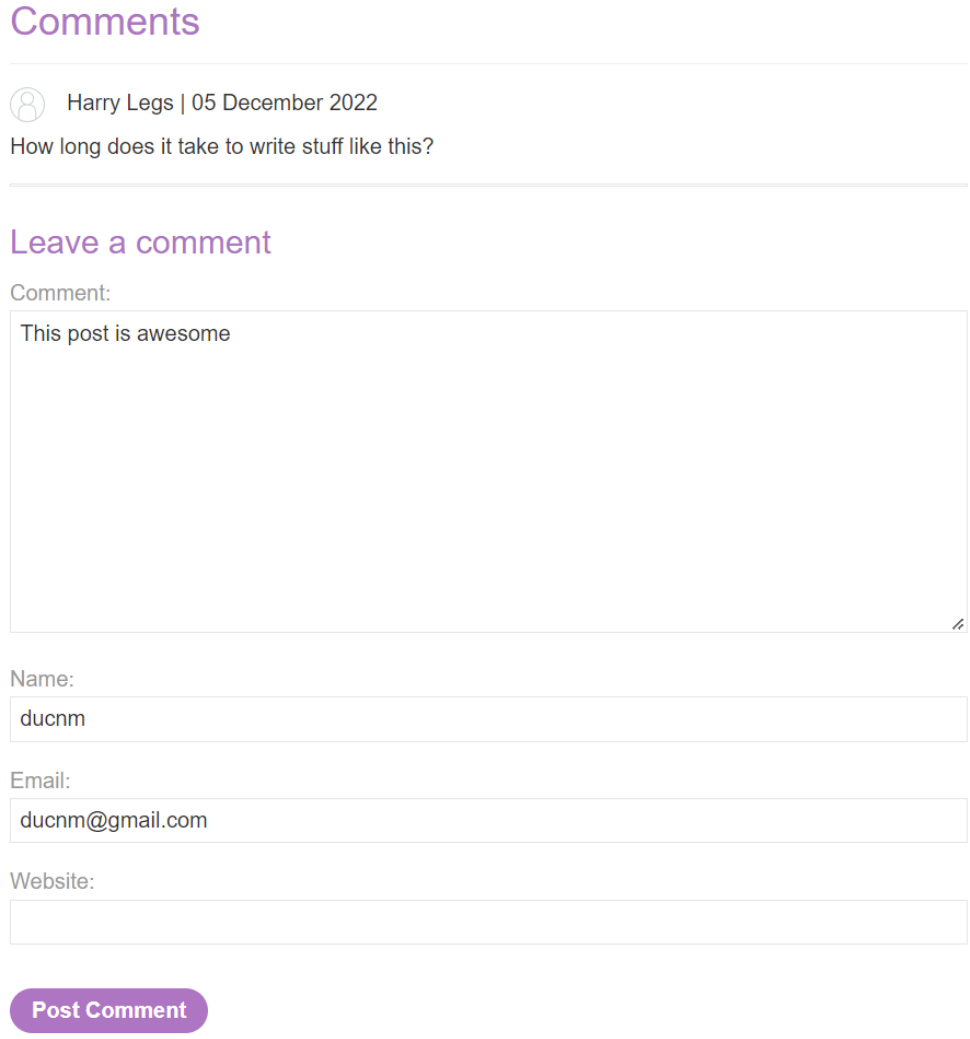
Content

1. SQL Injection
2. XSS

XSS

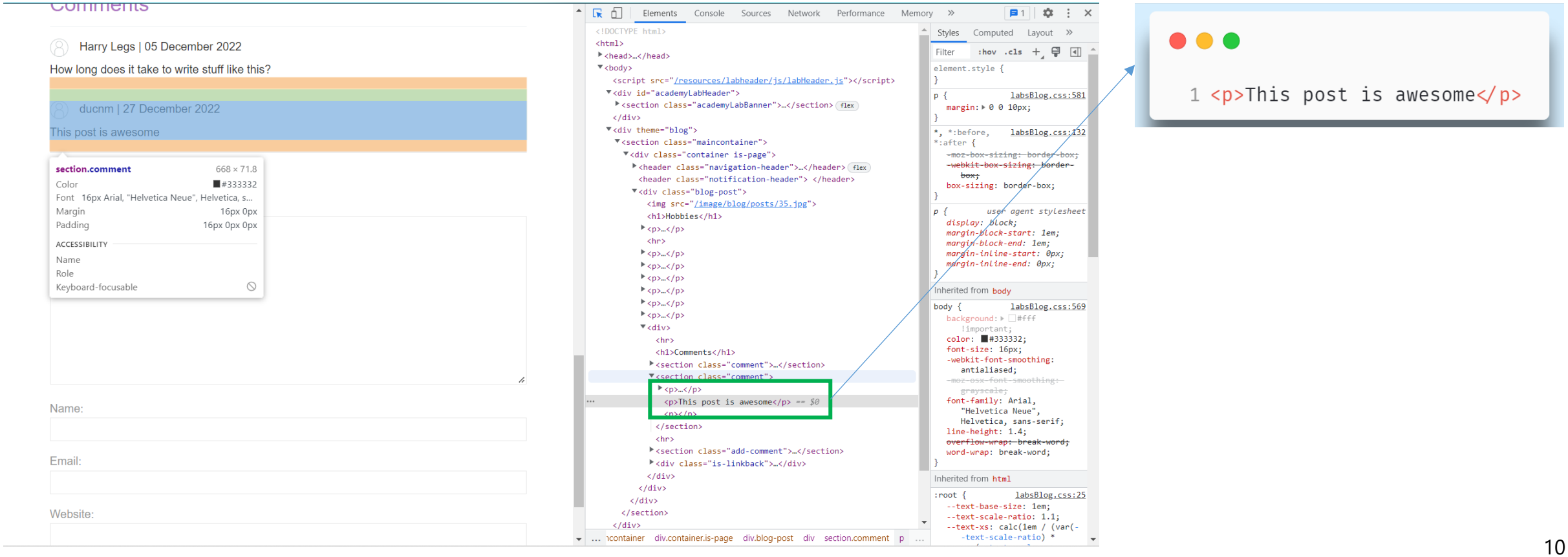
Eg: This sites contains a stored cross-site scripting vulnerability in the comment functionality. We will submit a comment that calls the alert function when the blog post is viewed.

Step 1: We will try to submit a comment:



XSS

Step 2: After submit, we reload the post and see our new comment, we can use Devtools → Elements to see our comment in raw HTML:



XSS

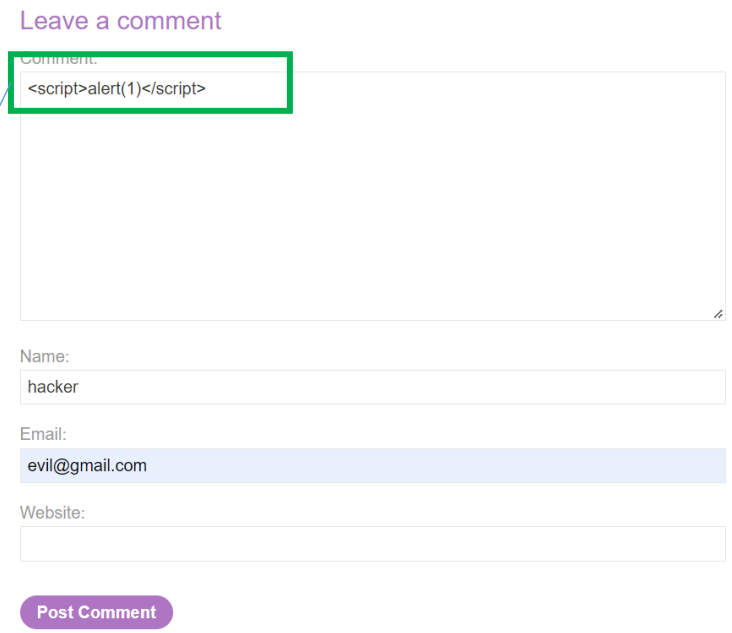
Step 3: We can guess that the database store comments in raw text, so if we comment a script tag, it will trigger when any user open the post, like this:

```
<script> /** bad stuff... </script>
```

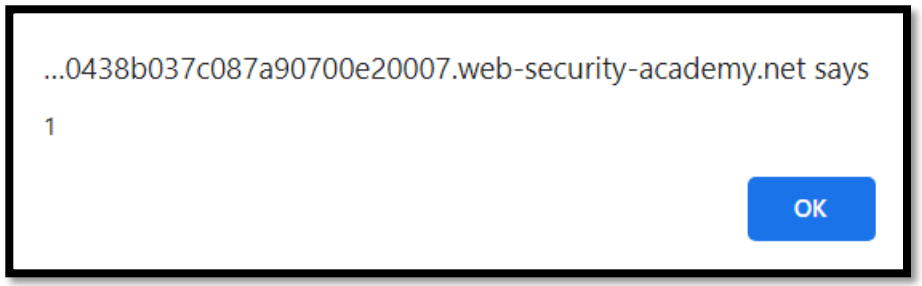
XSS

Step 4: We will try our predictions to comment a script which will alert 1 when any user open the post:

```
1 <script> alert(1) </script>
```

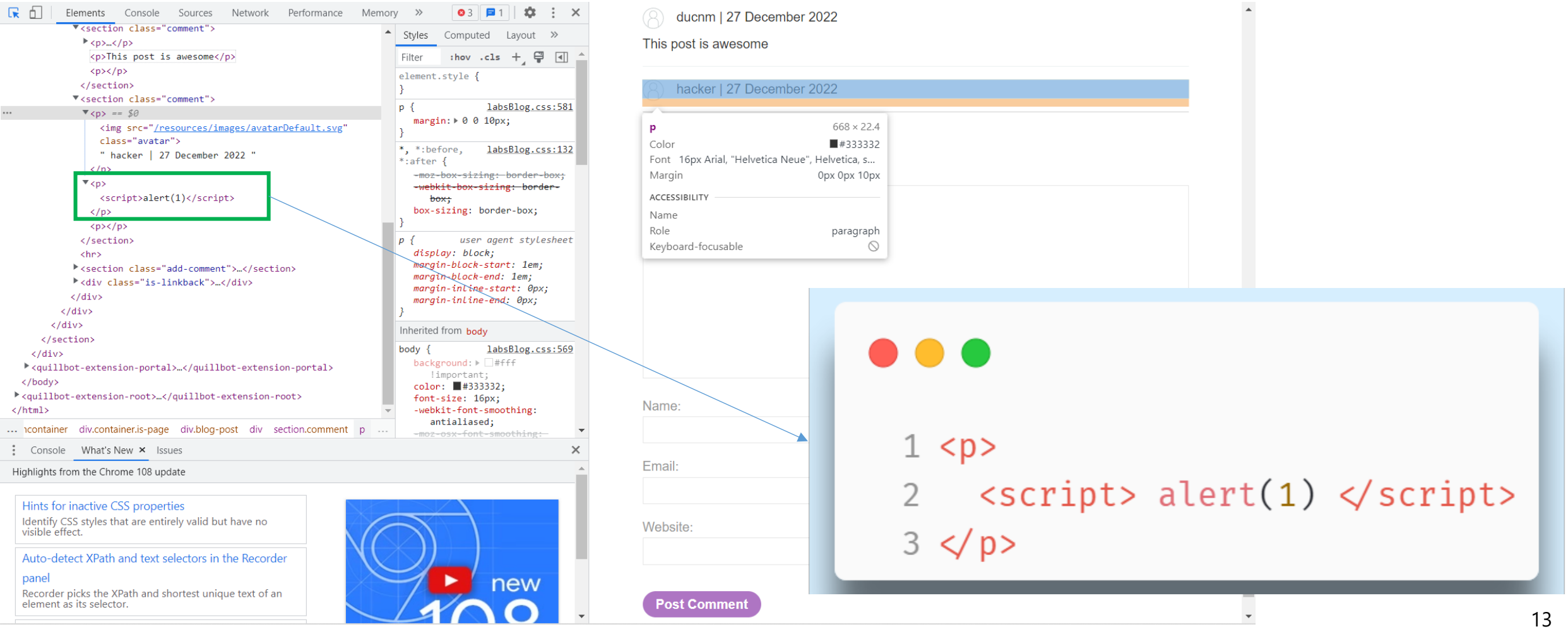


👉After submit, when back to blog post, we will see an alert



XSS

With the Devtools, we can see raw HTML:



Excercise

? Lab 01: Retrieve hidden data

This lab contains an SQL injection vulnerability in the product category filter. You must perform an SQL injection attack that causes the application to display details of all products in any category, both released and unreleased.

Excercise

? Lab 02: Determine the number of columns

This lab contains an SQL injection vulnerability in the product category filter. You must determine the number of columns returned by the query by performing an SQL injection UNION attack that returns an additional row containing null values.

Excercise

? **Lab 03:** Reflected XSS into attribute with angle brackets HTML-encoded

This lab contains a reflected cross-site scripting vulnerability in the search blog functionality where angle brackets are HTML-encoded. You must perform a cross-site scripting attack that injects an attribute and calls the alert function.

Excercise

? **Lab 04:** Stored XSS into anchor href attribute with double quotes HTML-encoded

This lab contains a stored cross-site scripting vulnerability in the comment functionality. You must submit a comment that calls the alert function when the comment author name is clicked.