

LA CTF — web/mutation **mutation** Writeup

Challenge: web/mutation mutation

Event: LA CTF / <https://lac.tf/>

Category: Web

Challenge Author: burturt

Writeup Author: Sunny Nguyen

Tools Used: Chrome DevTools (Mac)

Difficulty: Easy (but tricky if you're new to DevTools)

TL;DR

The page rapidly mutated its displayed content to make the flag hard to read manually. I used Chrome DevTools to **pause execution** from the **Sources** tab, then switched back to **Elements** and copied the flag while the DOM was frozen.

Challenge Description

“It’s a free flag! You just gotta inspect the page to get it. Just be quick though... the flag is constantly mutating. Can you catch it before it changes? 🎉”

The site elements constantly changed, making it hard to visually catch the flag before it mutated again.

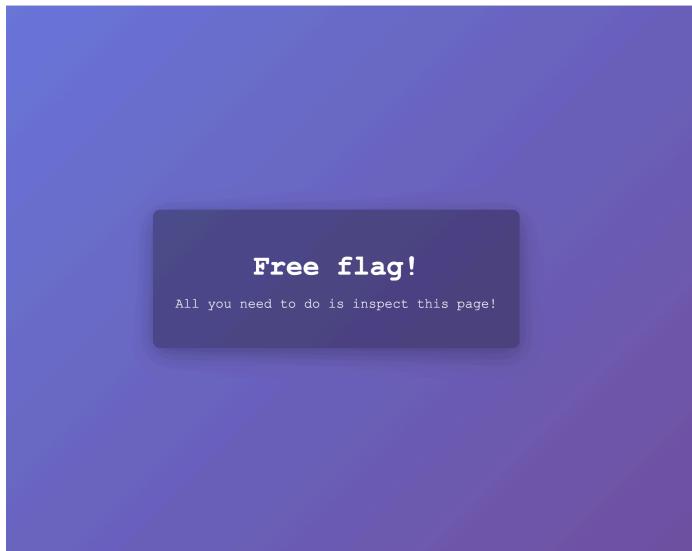
Objective

Capture the flag from a webpage where the DOM content changes rapidly.

Initial Analysis

When I first arrived at the challenge webpage, I noticed two things immediately:

1. **Right-click was disabled**, so I couldn't inspect the page normally.



Since I was on Mac, I opened Chrome DevTools manually using:

- **Cmd + Option + I**

At this point, I could see the page content changing constantly in the Elements panel.

A screenshot of the Chrome DevTools Elements panel. The top navigation bar shows "Elements" selected. The main area displays a long list of element IDs and their corresponding class names, all of which are preceded by the prefix "<!-- lactf{". The list includes various identifiers such as "wrong_flag", "fake_flag", "not_here", "bait", "decoy", "wrong_one", "nice_try", "wrong_flag", "not_the_flag", "decoy", "bait", "keep_looking", and "wrong_flag". The entire list is scrollable, indicating many more elements than shown. The bottom of the panel shows the standard DevTools footer with "Elements", "Sources", and "Network" tabs.

First idea (not ideal 😅)

My first thought was:

“Maybe I can screenshot the screen and get lucky.”

This would technically work, but it’s unreliable and not really a good technique.

Investigation

I explored Chrome DevTools for a bit (I wasn’t very experienced with it yet), and found the page’s `index` source file.

However, the JavaScript logic was **obfuscated**, so reverse engineering it would’ve taken longer than necessary.

The screenshot shows the Chrome DevTools interface with the 'Sources' tab selected. The file '(index)' is open, displaying obfuscated JavaScript code. A specific line of code is highlighted with a yellow background: `const _0xb9dca4 = _0x51e4`. The code is heavily minified, using underscores and numbers for variable names. The status bar at the bottom indicates 'Line 45, Column 13318' and 'Coverage: n/a'.

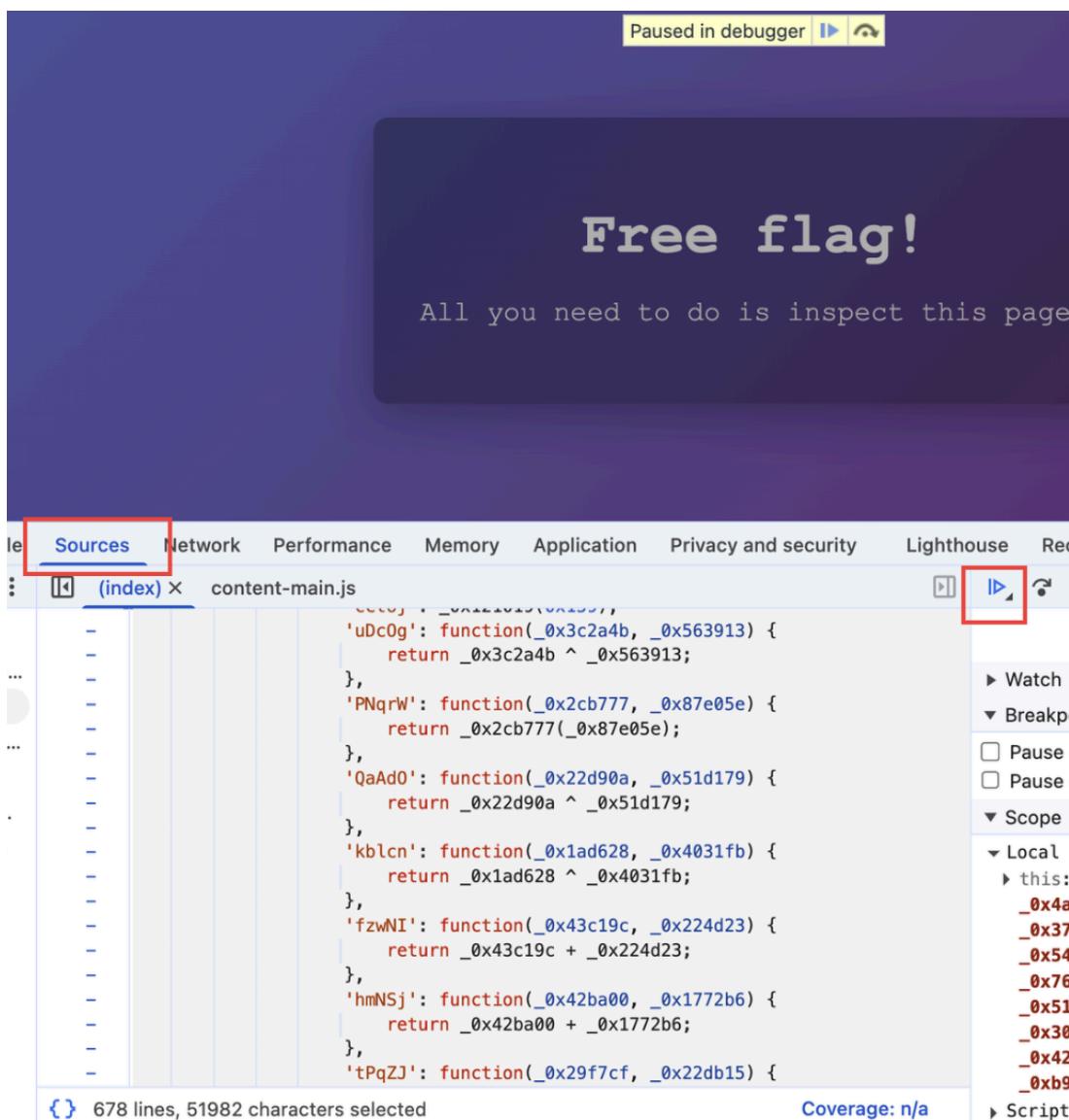
Instead of trying to deobfuscate the script, I looked for a faster approach.

Solution (What Worked)

The simplest solution was to **freeze** the page while the flag was visible.

Steps

1. Open **DevTools**
2. Go to the **Sources** tab
3. Click the **Pause** button (■) to pause JavaScript execution

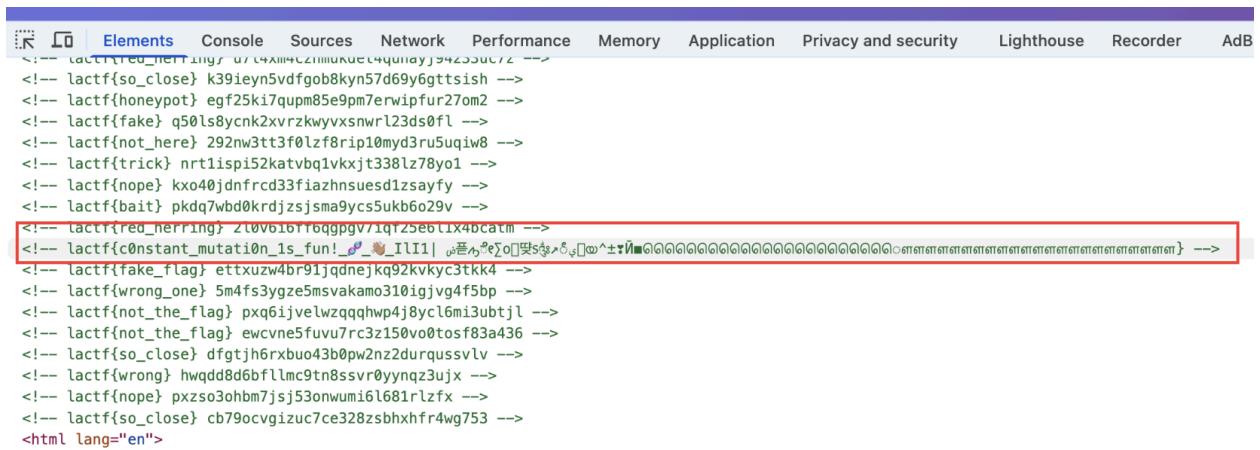


4. Switch back to the **Elements** tab
5. Scroll/look through the text output until finding a string matching a flag format (ex: `lactf{...}`)
6. Copy and paste the flag

Because the DOM was frozen while execution was paused, the flag stopped mutating long enough to copy it cleanly.

Result

I was able to successfully copy the flag directly from the frozen DOM.



The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. The page content is displayed as a series of obfuscated JavaScript strings. A red box highlights a specific string: `<!-- lactf{red_nerring} zt0vvbi0rr0qgppgv/1qfz5eb0lx40catm -->`. This string is part of a larger sequence of obfuscated code, including other lactf{} blocks and various identifiers like 'fake_flag', 'wrong_one', etc. The entire page content is wrapped in a `<html lang="en">` tag at the bottom.

What I Learned

- Even if right-click is disabled, you can still open DevTools with keyboard shortcuts.
 - You don't always need to reverse engineer obfuscated JavaScript — sometimes there's a simpler way.
 - **Pausing execution in the Sources tab** is a powerful trick for CTF web challenges where:
 - the DOM is changing rapidly
 - the flag flashes briefly
 - scripts are trying to hide content
-

Defensive Notes (Real-World Takeaway)

This kind of “security” is not real security — it’s just **client-side obfuscation**.

If a flag (or secret) exists in the client-side DOM at any point, a user can usually:

- freeze the DOM
- intercept responses
- view source
- dump memory/state
- or scrape the page