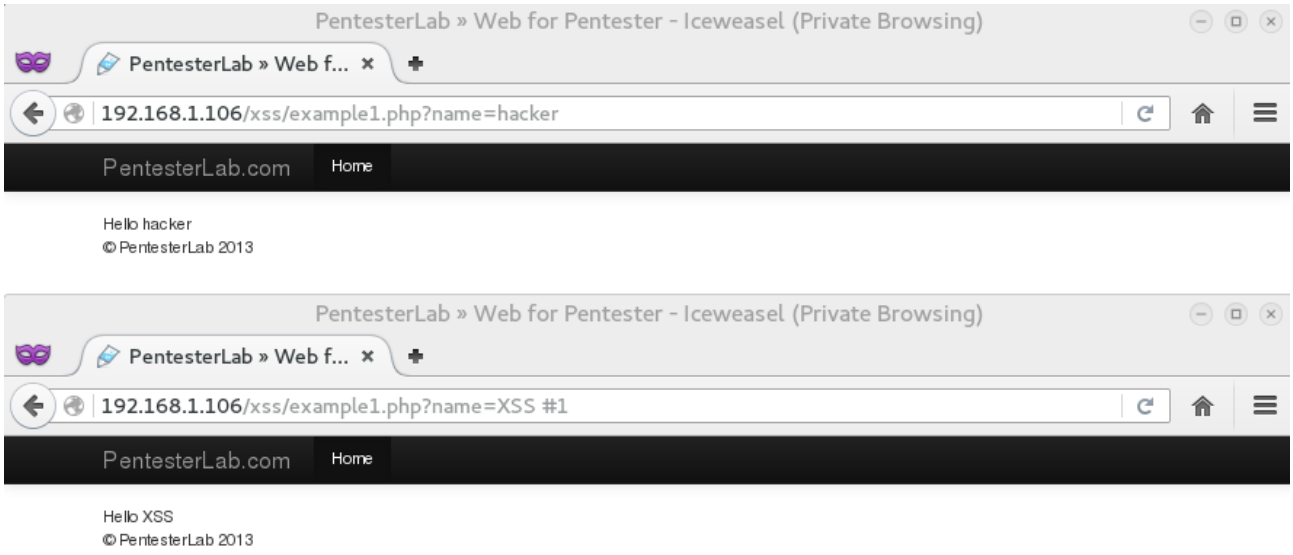
XSS Solutions for ‘Web for Pentester I’

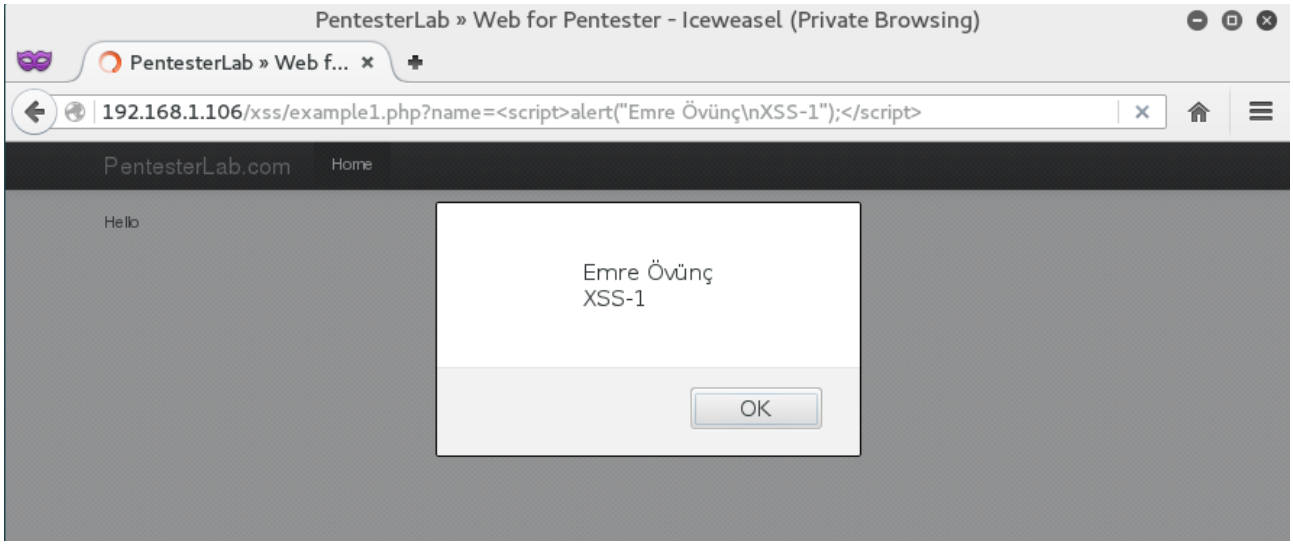
Emre ÖVÜNÇ
Intern – İnnovera
info@emreovunc.com

Example #1:

Firstly, I should look at the URL to understand web pages which may contain type of XSS attacks. It is the first sign that is changing word in the url.

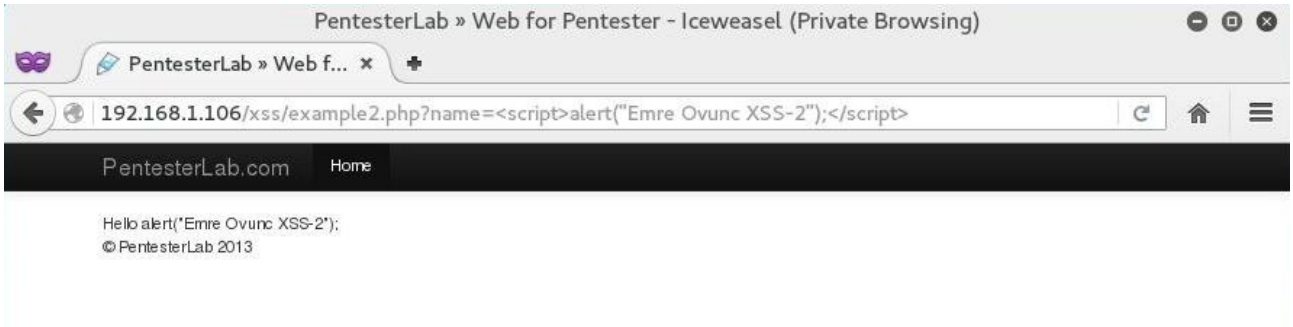


In this example, I try to use the basic payload , if I succeed , web page shows me an alert box. Only thing that you can do is adding script parameter at the end of the URL.
(e.g. **<script>alert(“Emre Ovunc XSS”);</script>**)

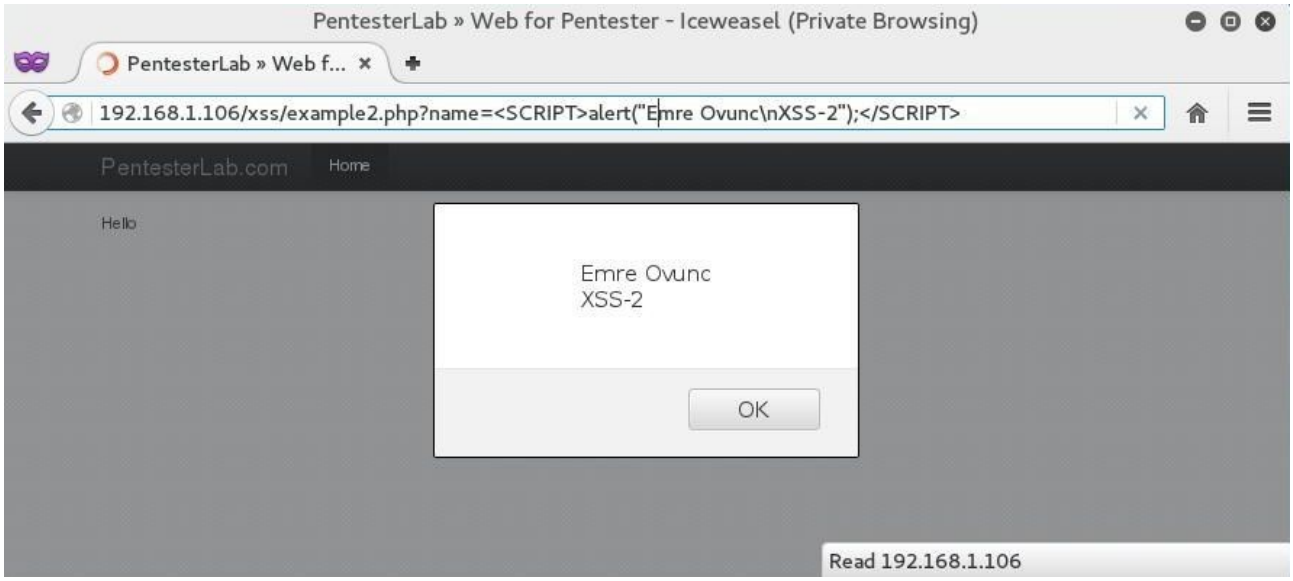


Example #2:

As you can see from the picture, I try to use same payload as Example #1, but “<script>” is filtered by the system. This approach prevents only the basic type of XSS attacks.

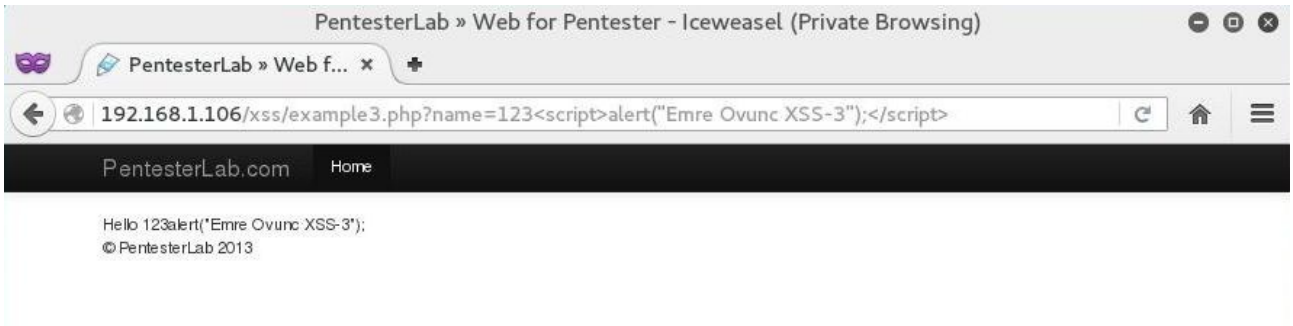


Then, I change <script> to **<SCRIPT>** because of prevention.

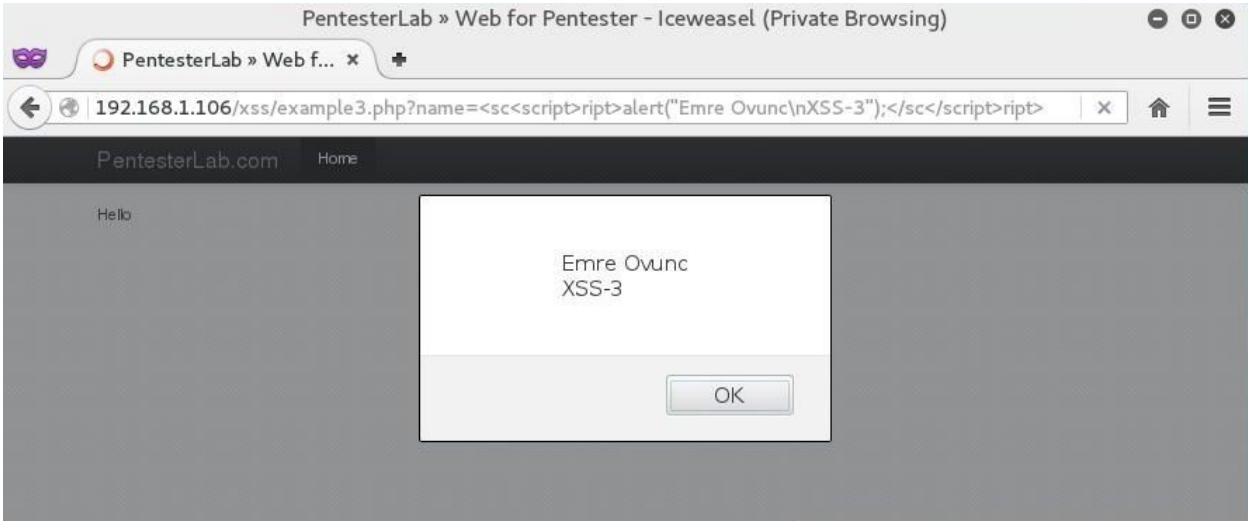


Example #3:

Now, the developer make effort to avoid type of XSS attacks, but nothing can not intimidate us. As you can see from the picture, when I write **123<script>alert(“Emre Ovunc XSS-3”);</script>**, I can see **123alert(“Emre Ovunc XSS-3”);** on the web page. It means that, the developer filter only <script> words. In addition, 123 and alert are combined each other.

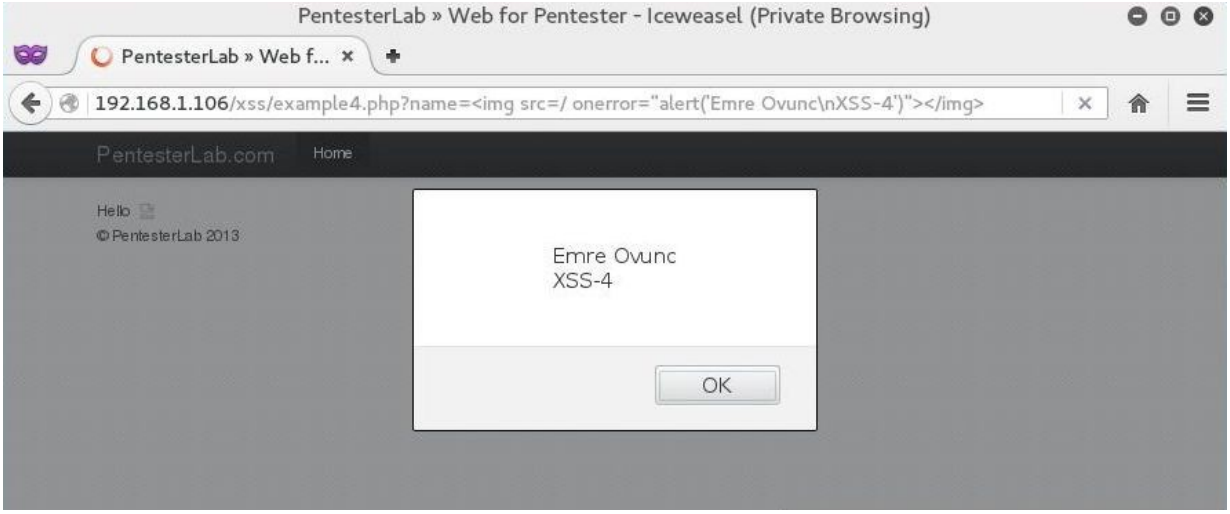


To bypass that kind of filter, I decide to divide “<script>” word into smaller parts. I mean, if I write **<sc<script>ript>** instead of “**<script>**”, it will pass filters and the result is “<script>”.



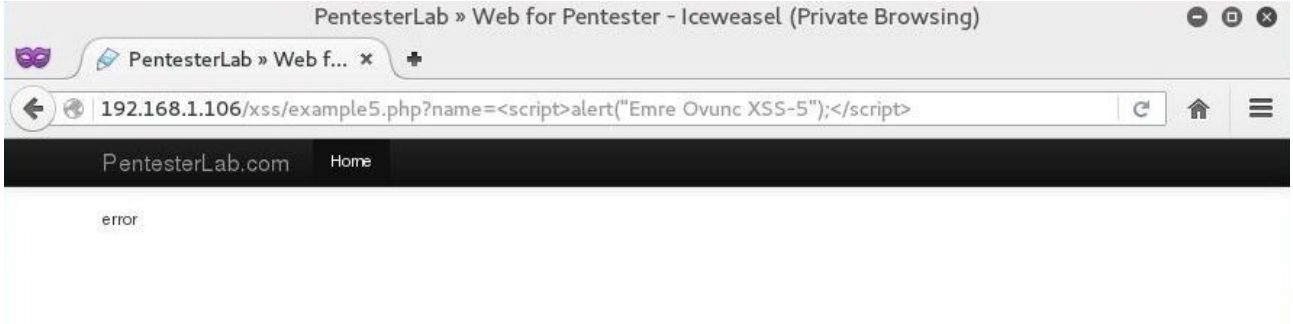
Example #4:

In this case, the developer blocks all requests which matches the word “script”. So that, I have to find another way to apply XSS attacks. There are many ways to run JavaScript , so I choose “**<img**” tag. Let’s do it.

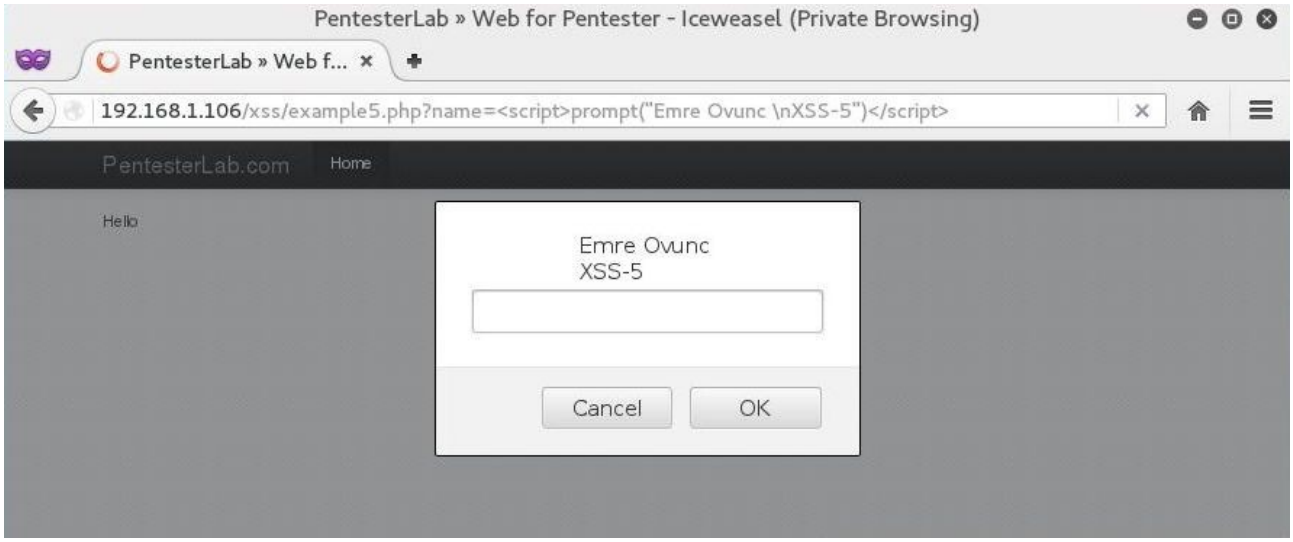


Example #5:

When I use a simple payload (e.g. **<script>alert(“Emre Ovunc XSS-5”);</script>**, I see that page returns me an error.



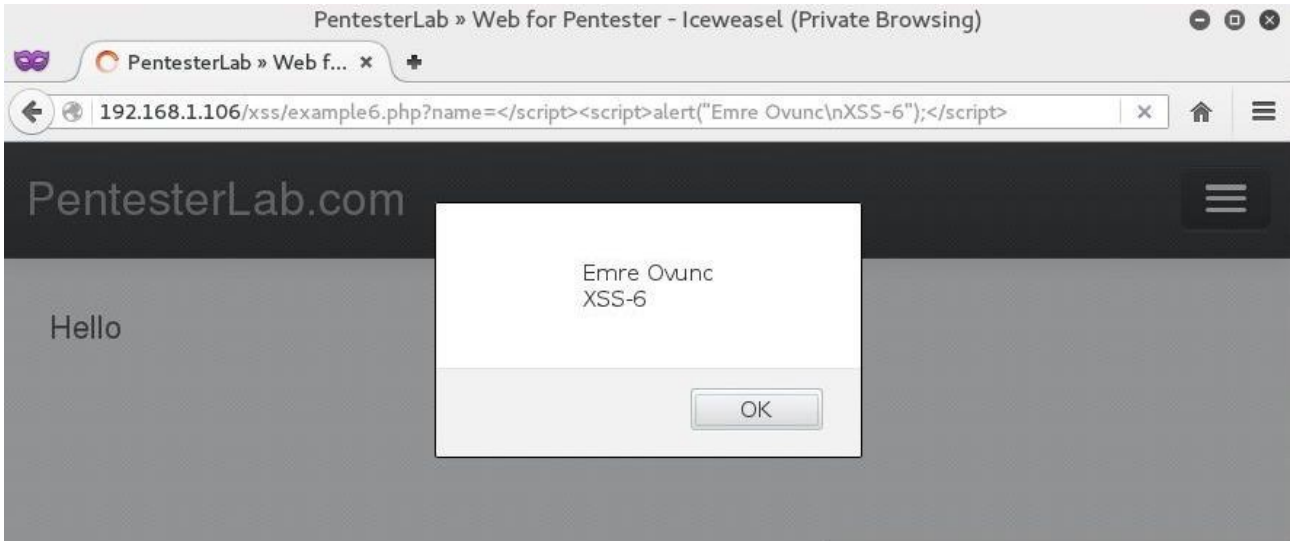
Then, I started to try another versions of payloads, I realize that “alert” word does not pass the filter. Thus, I change **“alert” to “prompt”.**



Example #6:

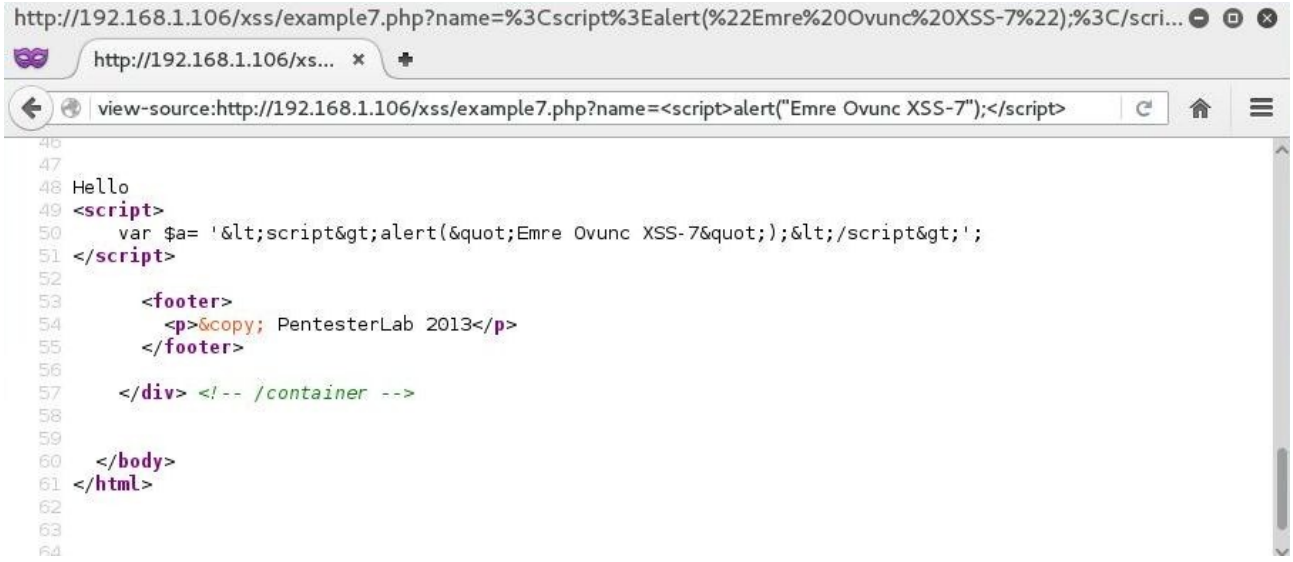
Again, I use a simple payload like Example #1, but web page prints only **";** characters on the screen. I think, I have to view the source code.

As you can see from last picture, I add my codes into **<script> … </script>** , then only thing that I want web page to create 2 different <script> and implement XSS attacks. To do that, I started XSS payload with **</script>** to deactivate first one

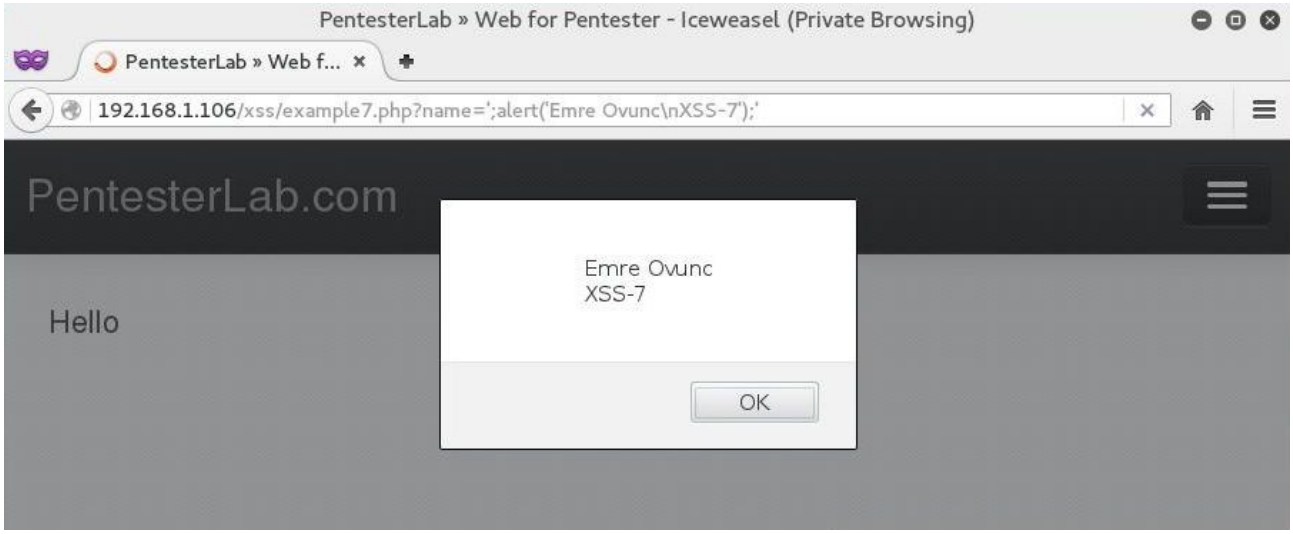


Example #7:

In this case, I use similar payload like Example #6 and I started to analyze source codes. I see that, all special characters are encoded (called HTML-Encode).

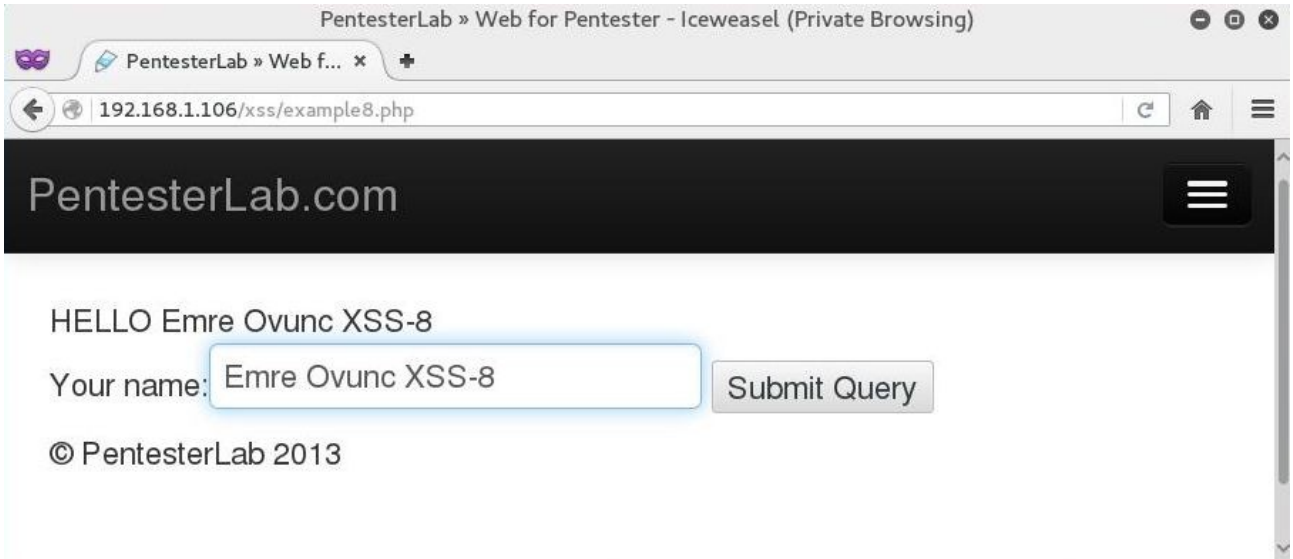


That's why, I add my codes into **'; …… ' .**



Example #8:

Now, this case is a slightly different example. When you enter the web page, a simple form says “Hello” to you. Of course I start to look at source codes immediately.



I started to find a way to inject my XSS codes. The one of my favorite way is closing old forms and place my codes into it.