Here we see a web page that is vulnerable to Cross Site Scripting. Specifically, this is vulnerable to Reflected Cross Site Scripting. In the URL we notice the parameter 'e' with the value "john". Let's change this to four As just to confirm that the value of this parameter is actually reflected to the web page.

Now, to check if this application is indeed vulnerable to XSS, we need to see if meta-characters, such as angle brackets, are being filtered or encoded. Let's modify the URL to include the simplest XSS payload, that is a script tag which shows an alert box. Reloading the page causes the alert box to pop up, which confirms our script tags are being executed.

By checking the page source for our modified error message, we can see that the script tags have indeed been added into the page unaltered. As far as the web browser if concerned, the injected JavaScript came from the developer of the application and not the attacker.

To finish this demo, let's now take a look at a more realistic payload to keylog a victim's password. The details of the payload are not important, but in a nutshell, we register an event handler that gets executed every time the user presses a key. We then read the value of the key pressed and send it to the attackers' server, using an image tag.

Here is a one-liner, compact version of this payload, that we can copy in the URL, after substituting the attackers' IP address. We are now ready to copy the final payload in the vulnerable URL. Before we do that, let's start a webserver, so that we can capture the key presses.

Ok, time to see this in action. Let's reload the page with the payload in the 'e' parameter and try lo go in. As you can see, we are now capturing the users' login credentials successfully.