# PHP Lab 7 Cross Site Scripting (XSS)

This lab is based on slides 9-11 of the CyberAces module, Basic Web Security.

The XSS vulnerability deserves a bit more attention, as it is a common problem and often underestimated. First of all, why is XSS "Critical"? It doesn't allow direct data access like SQL Injection, and it doesn't allow code execution on the server. Or does it?

XSS allows an attacker to modify the HTML of the site that is presented in the victim’s browser. The attacker could modify a login form, and change the URL so that the form submits the username and password to the attacker’s site. Or the attacker could send the site’s session cookie’s to the attacker’s site. The attacker can run scripts of his choosing in the victim’s browser to perform reconnaissance of the victim’s network, or even own the victim’s computer if the victim is foolish enough to run the browser with admin privileges. XSS is important.

## Install a vulnerable script

The script, guess.php, is a guessing game that was written by a previous class. It asks the user to guess a number between 1 and 10, displays the number the user entered, and then tells the user whether the guess was correct, high, or low. The problem is the part, “displays the number the user entered.” It takes the user input and puts it back into the HTML that executes on the user’s browser. Therefore, if the input is a script, it will be executed by the browser. Since the script runs in the browser and not in the server, it must be written in JavaScript instead of PHP.

Copy guess.php to your /var/www/html directory and view it in a browser.  
<http://localhost/guess.php>, or <http://[IP> address]/guess.php.  
Check to see that the script works properly. If it doesn’t work check the privileges of /var/www/html/guess.php with ls -l. You may have to add read privileges for all users with chmod +r. Play the game and see if you can win—since you can see the code in guess.php, it shouldn’t be very hard. Look at how the guess is passed to the server in the URL.

## Why is the script vulnerable?

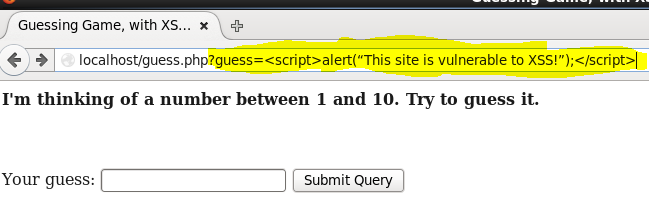
Examine the guess.php script in a text editor and find the line where the script takes the user’s guess as an input. Then find the line where the user’s guess is echoed back to their web browser. What checking is done to ensure that the input is a valid guess and not something else, like evil attacker code?

## Demonstrate XSS

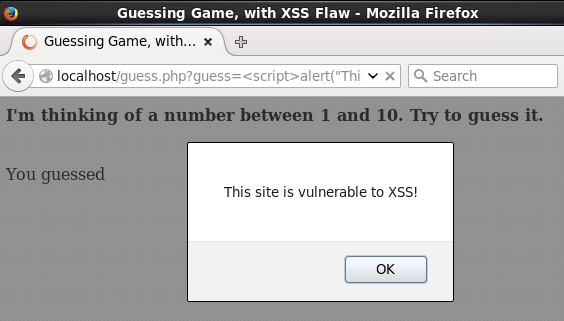
Now enter a small JavaScript into the URL bar, after localhost/guess.php.  
?guess=<script>alert('This site is vulnerable to XSS!');</script>  
  
Or, you can enter  
<script>alert('This site is vulnerable to XSS!');</script  
in the ‘Your guess:’ window.

Or, you can try both!

Use the Firefox browser in your Ubuntu VM, as I’ve tested that to make sure the XSS will work. If you use different browser, it may detect this simple XSS and block it. (This does not mean that XSS is not a risk. Browsers can detect simple XSS, but there are many ways to avoid detection with more complex XSS. The key is to prevent XSS in the server code! See this site for more information. <https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)>



In HTML, the script tag ( <script> ) tells the browser it will run JavaScript commands. The command is  
alert(“This site is vulnerable to XSS!”);  
If the site really is vulnerable to XSS, it will pass the script tags and the command back to the browser unchanged. The browser will see the script in the HTML and run the command. In this case, alert puts up a message box with the enclosed text.

When you go to that URL, you should see this: 

The message is harmless, but what it signifies is not. If an attacker can get you to click on a link they create that goes to a site with an XSS vulnerability, they can run their code on your browser. If the site with the XSS vulnerability is important, like a bank or a store, they can steal your connection and do anything that you could do on that site.

## Fixing XSS

Go back to guess.php on your CentOS VM and identify the code that caused the XSS problem. Remember, it takes input from the user, and then echoes it directly to the user’s browser. PHP includes a function called html\_entities that can fix the problem. It changes symbols that can cause problems, like /, into their ASCII counterparts like %2F. The browser will display them as text and not execute the script. Research the html\_entities function and use it to fix the XSS vulnerability in guess.php.

An even better fix would be to only accept valid inputs to begin with. In our case, those are integers between 1 and 10. Anything else should be blocked.

The first fix is an example of blacklisting. It attempts to remove anything that could be bad from the user input. Attackers are ingenious and can sometimes find ways to evade blacklists, even good ones like html\_entities. The second fix is an example of whitelisting. If you can narrow the range of user inputs to the absolute minimum, it is harder for the attacker to exploit your code. In general, whitelisting is more effective than whitelisting.

# Hand In

1. Hand in the changes you made to guess.php so that it is not vulnerable to XSS.
2. Read this page. <https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting> What type of XSS did we do in this lab?