PENTESTER LAB - WEB FOR PENTESTER

Lab Exercises

Name : Pathima Shahama Tamber

Student Number : MS15903020

Subject : Penetration Testing

Table of Contents

1. Introduction

Follow the URL https://pentesterlab.com/exercises/ and select your course as you wish.

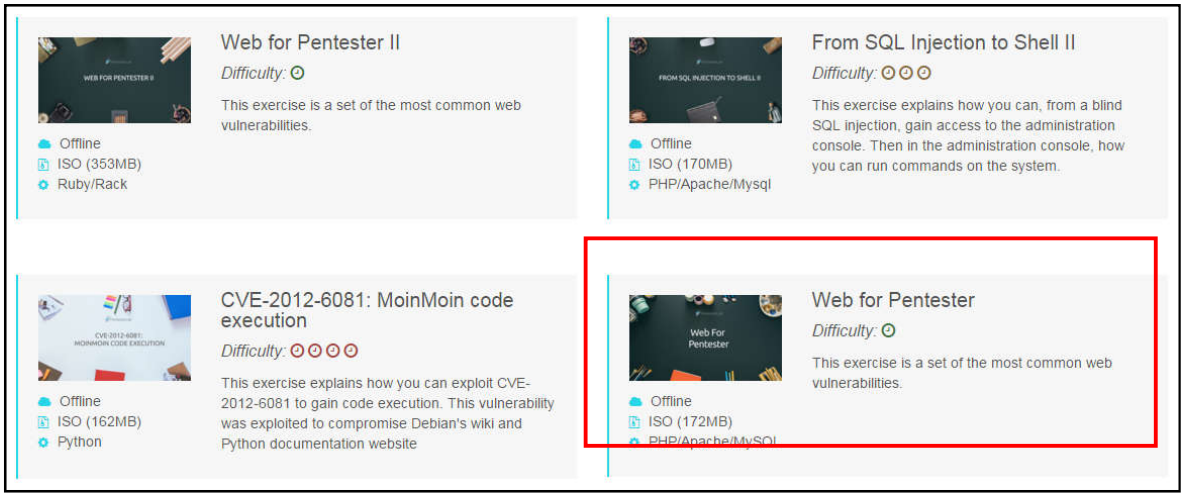

Figure 1 : Select the particular course

Download the ISO from the website (URL : https://pentesterlab.com/exercises/web_for_ pentester) and course content.

Install the ISO in the virtual Environment such as VMware or Virtual Box.

After installing the ISO power on it and get the IP address of it by typing ifconfig.

```
user@debian:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:1a:74:1f
          inet addr:192.168.42.132  Bcast:192.168.42.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe1a:741f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:463 errors:0 dropped:0 overruns:0 frame:0
          TX packets:23 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:48340 (47.2 KiB)  TX bytes:3602 (3.5 KiB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:528 (528.0 B)  TX bytes:528 (528.0 B)

user@debian:~$ _
```

Figure 2 : Find the IP Address of the Server

Type the ip address 192.168.42.132 shown above in the address bar of the browser such as Chrome. Then you will redirect to web for pentester exercises. In that page you can find the set of most common vulnerabilities. You have to refer Web For Pentester course from the Penterserlab.com and do the exercises as given there.

Character	URL encoded value
\r	%0d
\n	%0a
	%20 or `+`
?	%3f
&	%26
=	%3d
;	%3b
#	%23
%	%25

Figure 3 : Encoding Table

2. Example 1

The first vulnerable example is just here to get you started with what is going on when you find a XSS. Using the basic payload, you should be able to get an alert box.

In the testing page explained previously, go to XSS example 1 and replace the URL http://192.168.42.132/xss/example1.php?name=hacker into http://192.168.42.132/xss/example1.php?name= <script>alert(1);</script>

Once you send your payload, you should get something like:

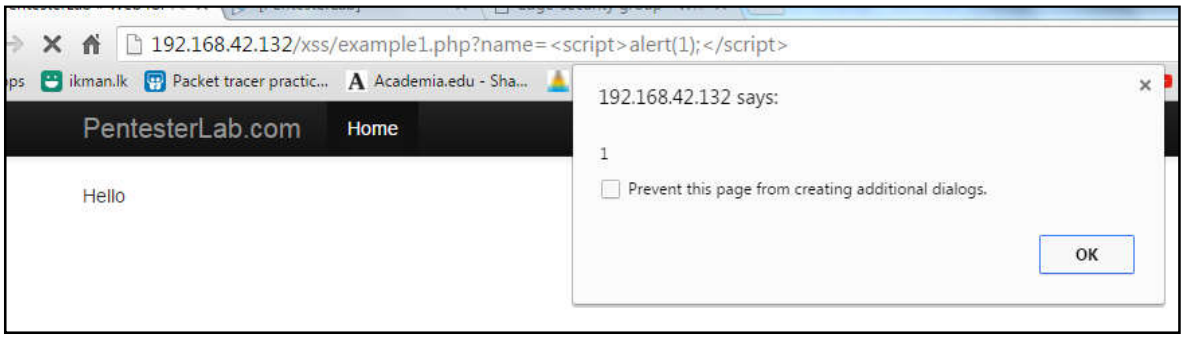

Figure 4

Make sure that you check the source code of the HTML page to see that the information you sent as part of the request is echoed back without any HTML encoding.

3. Example 2

Move to XSS example 2. In the second example, a bit of filtering is involved. The web developer added some regular expressions, to prevent the simple XSS payload from working.

If you play around, you can see that <script> and </script> are filtered. One of the most basic ways to bypass these types of filters is to play with the case: if you try <sCript> and </sCRIpt for example, you should be able to get the alert box.

http://192.168.42.132/xss/example1.php?name= <SCript>alert(1);</sCript>

4. Example 3

You notified the developer about your bypass. He has added more filtering, which now seems to prevent your previous payload. However, he is making a terrible mistake in his code (which was also present in the previous code)...

If you keep playing around, you will realise that if you use Pentest<script>erLab for payload, you can see PentesterLab in the page. You can probably use that to get <script> in the page, and your alert box to pop up.

Figure 5

You have to change the URL into http://192.168.42.132/xss/example3.php?name=Scr<script>ipt to get the result.

5. Example 4

In this example, the developer decided to completely blacklist the word script: if the request matches script, the execution stops.

Fortunately (or unfortunately depending on what side you are on), there are a lot of ways to get JavaScript to be run (non-exhaustive list):

- with the <a tag and for the following events: onmouseover (you will need to pass your mouse over the link), onmouseout,onmousemove, onclick ...
- with the <a tag directly in the URL: <a href='javascript:alert(1)'... (you will need to click the link to trigger the JavaScript code and remember that this won't work since you cannot use script in this example).
- with the <img tag directly with the event onerror: .
- with the <div tag and for the following events: onmouseover (you will need to pass your mouse over the link), onmouseout,onmousemove, onclick...

You can use any of these techniques to get the alert box to pop-up :

For example you can tryout <img onmouseover="bigImg(this)" onmouseout="normalImg(this)" border="0" src="smiley.gif" alt="Smiley" width="32" height="32">

http://pentesterlab/xss/example4.php?name=

6. Example 5

In this example, the <script> tag is accepted and gets echoed back. But as soon as you try to inject a call to alert, the PHP script stops its execution. The problem seems to come from a filter on the word alert.

Using JavaScript's eval and String.fromCharCode(), you should be able to get an alert box without using the word alert directly. String.fromCharCode() will decode an integer (decimal value) to the corresponding character.

You can write a small tool to transform your payload to this format using your favorite scripting language.

Using this trick and the ascii table, you can easily generate the string: alert(1) and call eval on it.

Another easier bypass is to use the functions prompt or confirm in Javascript. They are less-known, but will give you the same result. In order to have our payload run we should use a different javascript function called "eval(String.fromCharCode()" what this does is convert decimal to ascii allowing us to bypass the preg_match function.

http://pentesterlab/xss/example5.php?name=<script>eval(String.fromCharCode(97, 108, 101, 114, 116, 40, 39, 120, 115, 115, 39, 41))</script>

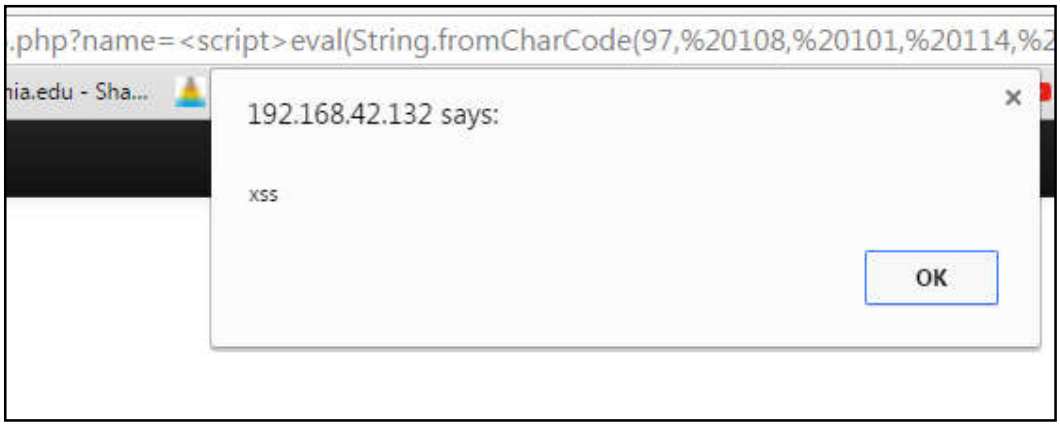

Figure 6

7. Example 6

Here, the source code of the HTML page is a bit different. If you read it, you will see that the value you are sending is echoed back inside JavaScript code. To get your alert box, you will not need to inject a script tag, you will just need to correctly complete the pre-existing JavaScript code and add your own payload, then you will need to get rid of the code after your injection point by commenting it out (using //) or by adding some dummy code (var $dummy = '') to close it correctly.

In this task you have to modify the Javascript code by adding your dummy code.

http://192.168.42.132/xss/example6.php?name=hacker%22;alert(%271%27);%22

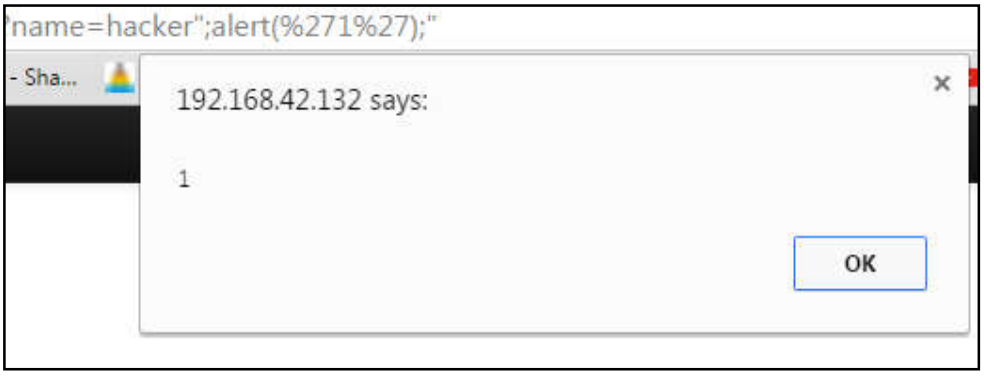

Figure 7

8. Example 7

This example is similar to the one before. This time, you won't be able to use special characters, since they will be HTML-encoded. As you will see, you don't really need any of these characters.

This issue is common in PHP web applications, because the well-known function used to HTML-encode characters (htmlentities) does not encode single quotes ('), unless you told it to do so, using the ENT_QUOTES flag.

In this example html encoding on special characters is added in which means you cannot use a double quote ‘“’.. However, this string should work..

http://pentesterlab/xss/example7.php?name=hacker';alert('1');'
observe the result.

9. Example 8

Here, the value echoed back in the page is correctly encoded. However, there is still a XSS vulnerability in this page. To build the form, the developer used and trusted PHP_SELF which is the path provided by the user. It's possible to manipulate the path of the application in order to:

- call the current page (however you will get an HTTP 404 page);
- get a XSS payload in the page.

This can be done because the current configuration of the server will call /xss/example8.php when any URL matching/xss/example8.php/... is accessed. You can simply get your payload inside the page by accessing/xss/example8.php/[XSS_PAYLOAD]. Now that you know where to inject your payload, you will need to adapt it to get it to work and get the famous alert box.

Trusting the path provided by users is a common mistake, and it can often be used to trigger XSS, as well as other issues. This is pretty common in pages with forms, and in error pages (404 and 500 pages).

10. Example 9

This example is a DOM-based XSS. This page could actually be completely static and still be vulnerable.

In this example, you will need to read the code of the page to understand what is happening. When the page is rendered, the JavaScript code uses the current URL to retrieve the anchor portion of the URL (#...) and dynamically (on the client side) write it inside the page. This can be used to trigger a XSS vulnerability, if you use the payload as part of the URL.

For the last challenge the existing javascript is looking for the anchor # to write the url inside the page. We can exploit this by putting the xss payload inside the url after the anchor.

http://pentestlab/xss/example9.php#<script>alert('xss')</alert>

There we have all the XSS examples from Web for Pentester I. Tune in next time for some of the other examples.

This example uses the PHP_SELF function which allows us to insert malicious javascript by closing off their code and executing our own..

http://pentesterlab/xss/example8.php/"><script>alert('xss')</script>