xss跳转代码_XSS跨站脚本攻击-靶场writeup&总结

weixin_39963819 于 2020-11-21 10:59:29 发布 169 收藏
文章标签： xss跳转代码

XSS简介

XSS(跨站脚本攻击)是指攻击者在网页中嵌入客户端脚本，通常是Javascript编写的恶意代码，当用户使用浏览器浏览被嵌入恶意代码的网页时，恶意代码将在用户的浏览器上被解析执行。

XSS Education

环境搭建

```
docker search xss
# 搜索docker镜像，找到xss education对应的image
docker run -d -p {p}:80 {sha}
# {p}为开放服务的端口号，{sha}为镜像的hash值
```

Basic XSS

复旦内网访问： http://6.fdujwc.cn:84/basicxss/

黑盒测试

尝试：

```
<
```

X进去了。

源码分析

漏洞产生于如下代码段：

```
echo
```

直接将用户输入插入了html页面，没有任何过滤。

JavaScript XSS

复旦内网访问： http://6.fdujwc.cn:84/javascriptxss/

黑盒测试

尝试：

```
<
```

查看回显页面的代码：

```
<
```

与basic xss将输入插入到html中不同，这里把输入未经过滤插入了JavaScript代码段。因为在字符串中，需要闭合两边的引号，构造如下payload：

```
a";alert(1);var q="a
```

X进去了。

源码分析

漏洞产生于如下代码段：

```
$q
```

1-3 Filtered XSS

复旦内网访问： http://6.fdujwc.cn:84/filteredxss/

黑盒测试

尝试：

```
<
```

查看回显html页面代码：

```
<!DOCTYPE html>
```

有两处值得我们注意： 1、我们的输入alert(1);被过滤得只剩下 t(1) ，可见后端有一个比较凶狠的正则过滤，也许可以被绕过； 2、html后半部分有一个JavaScript代码段，其中有一个变量q，它先被赋值为123，然后被用注释符包裹，最后写入到到页面中，如果我们可以控变量q的值，闭合注释，就能利用document.write()函数向页面中插入js代码。

再回到前端看看有没有可能控q，发现：

```
<
```

有一个默认值为123的变量id，它的属性被设置为hidden，不难想到这里的id就是后端的q，我们将hidden改为text，页面上就会多出一个id的输入框。

值得注意的是，因为是插入到js代码断中，而html又是一种从前往后遇到匹配的标签就解析的语言，所以不能使用</script>标签，否则会使该代码段的<script>标签提前被闭合而产生混乱，就像下面这样：

```
<
```

那要如何插入js代码呢？这里就要用到一个伪协议：

```
javascript:[code]
```

直观上来说，这个伪协议可以让一个放url链接的地方，执行js代码，比如：

```
<
```

点这里跳转链接

点这里执行js代码

（知乎这里在文件导入时出现了异常，可以进入我的博客查看代码效果

https://procodeus.github.io/）

现在，我们运用这个伪协议，构造payload：

```
<
```

X进去了。

另外，题目要求alert出creditcard的值，所以还要把alert内的东西完善一下，最终payload：

```
<
```

源码分析

核心代码：

```
$id
```

从源码中可以看出，过滤creditcard的正则几乎能把它过滤得就剩点渣，而对变量id只是转义了一下双引号（所以用双引号闭合是X不进去的）。这里体现了木桶原理，只要有一个可控输入位置成为X入点，其它输入的过滤再严格也没有意义。（泪目

1-4 Chained XSS

复旦内网访问： http://6.fdujwc.cn:84/chainedxss/

这道题有点绕，偷懒留个坑.....

XSS Game

https://xss-game.appspot.com/

Level 1: Hello, World of XSS

没有过滤。

Level 2: Persistence is key

尝试：

```
<
```

被过滤得空格都不剩。

猜测可能存在对关键词的替换，尝试双写绕过：

```
<
```

还剩下：

```
ipt>alert(1)ipt>
```

尝试javascript:[code]伪协议，成功X入：

```
<
```

Level 3: That sinking feeling... （*）

这道题没有任何输入端口，我们可控的只有url。多点几次后不难发现，url末尾的数字决定了展示哪一张图片，而这个过程是前端js代码处理的。换句话说，url末尾其实相当于一个我们可控的参数，我们找到处理它的代码段,发现：

```
// Dynamically load the appropriate image.
```

变量num，也就是最后那个数字，未经过滤就被拿去拼接js代码了！利用此处可以构造payload：

```
#
```

Level 4: Context matters （*）

查看前端传参的代码：

```
<
```

显然，参数timer的值是可控的，尝试在这里X入：

```
</
```

结果发现特殊字符都被转义了：

```
<
```

输入正常的数字来查看这一行：

```
<
```

发现参数timer是不经过滤就被onload使用的，而onload又会运行后面的js代码，所以我们可以在这里做文章,payload:

```
3');alert('fdujwc
```

X进去了。 来看看此时的前端html：

```
<
```

值得一提的是这一题的hint2：

```
When browsers parse tag attributes, they HTML-decode their values first. <foo bar='z'> is the same as <foo
```

这也就是为什么我们的payload部分被编码后js代码依然能被正常执行。

Level 5: Breaking protocol （*）

综合前端：

```
<
```

和url：

```
https://xss-game.appspot.com/level5/frame/signup?next=confirm
```

发现只要控next参数就可以实现js代码注入，比如传入：

```
next=javascript:alert(1);
```

Level 6: Follow the （*）

查看前端js代码，有两个值得我们注意的地方： 获得gadget文件名：

```
// Take the value after # and use it as the gadget filename.
```

包含并加载gadget：

```
function
```

所以这道题的思路还是比较明确的，就是要让加载gadget中包含我们想X入的js代码。至少有下面两种做法：

远程包含：

```
#//www.google.com/jsapi?callback=alert
```

这里使用的是google的一个api，也可以直接把js文件放在自己的服务器上，但注意这里一定要用支持https的服务器，因为题目环境是https的，不能用http引用外部文件。

data:// 伪协议

```
#data:text/javascript,alert('behindthefirewalls')
```

XSS Challenges

https://xss.haozi.me/

0x00

```
<
```

0x01

```
</
```

0x02

```
<
```

0x03

括号被过滤，用反引号绕过

```
<
```

0x04

```
function
```

括号和反引号都被过滤，所以要想办法运行编码后的js代码，下列几个场景可以达到这个目的

利用异常处理：

```
<
```

html标签的属性

```
<
```

<svg>

```
<
```

0x05

```
<
```

0x06

要绕过一个正则将js代码插入到html标签中

```
function
```

中间用一个换行符分隔即可：

```
type=image src onerror
=alert(1)
```

0x07

把所有尖括号对包裹的东西过滤了：

```
function
```

但其实html中很多时候少个>并不影响解析，比如这样：

```
<
```

0x08

```
function
```

</style>标签被替换成了注释..... 直接双写绕过：

```
</style
```

0x09

```
function
```

要求url开头必须是https://www.segmentfault.com，然而后面可以随便加：

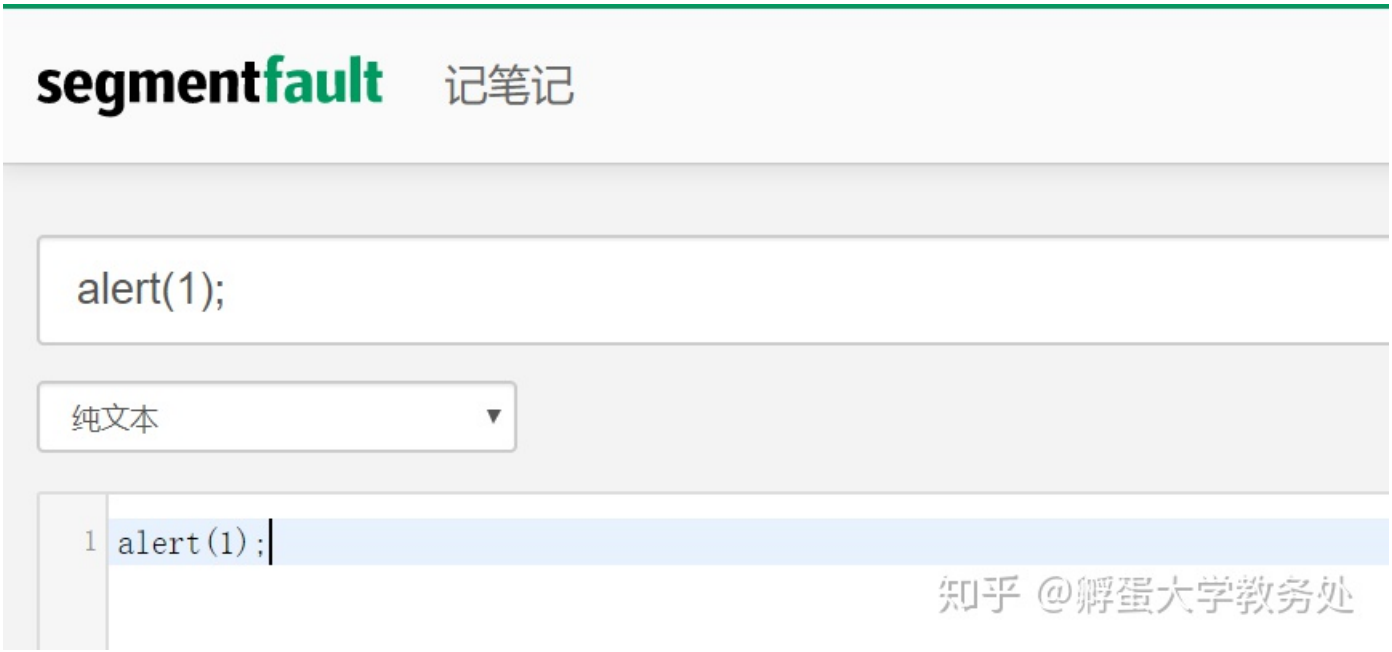
```
</
```

0x0A （*）

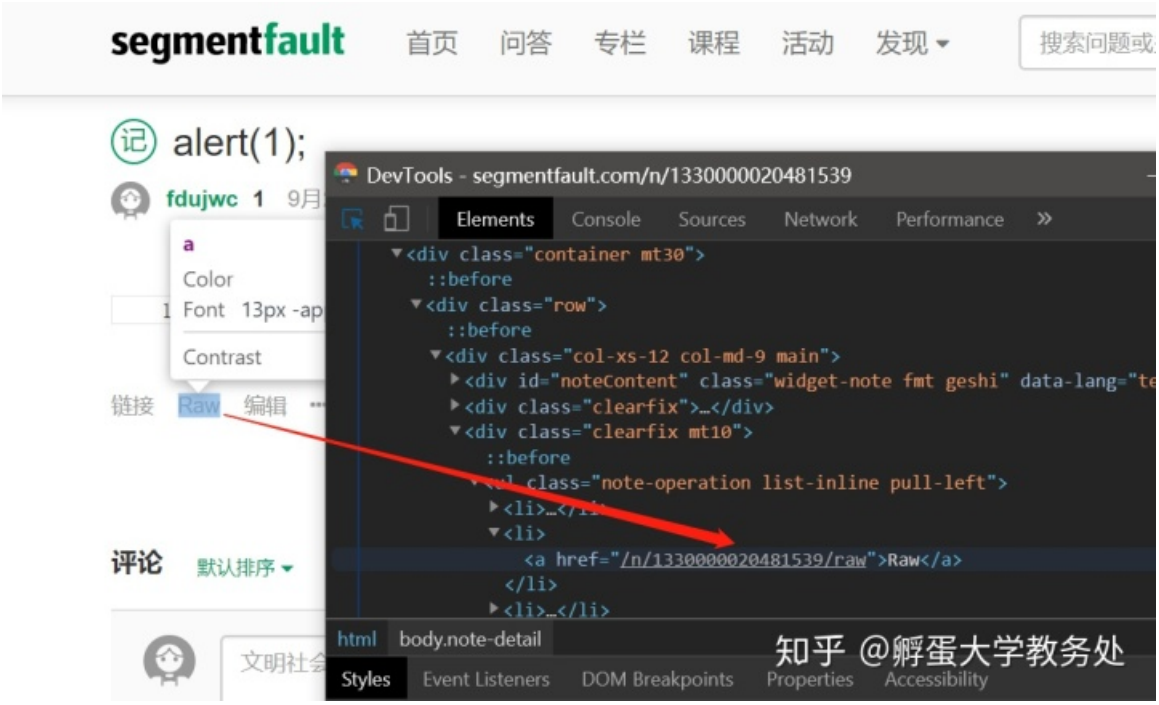
```
function
```

在上一题的基础上，对输入进行了一些转义，目前没有想到绕过的方法，但是可以尝试直接在https://www.segmentfault.com 上构造js文件（不一定是.js文件，只要内容被<script></script>包裹后能产生js代码执行效果即可），这个过程十分有趣。

我们先去这个网站注册一个账号，然后寻找可以插入内容的地方。这个网站有一个记笔记的功能，如下图：



以纯文本的方式写入一个内容为alert(1);的笔记，然后进入这个笔记分析这个笔记在服务端的存储结构发现：



这个笔记目录下有一个叫作raw的文件，点进去看看，发现内容就是我们笔记的内容：

```
alert(1);
```

由此尝试构造payload：

```
https://www.segmentfault.com/n/1330000020470115/raw
```

payload被转义成了：

```
https:&#x2f&#x2fwww.segmentfault.com&#x2fn&#x2f1330000020470115&#x2fraw
```

但此时有一个问题，#x2f1330000020470115被解析为url时会全部被解析（我们只希望2f被解析为/），好在url被解析是会无视换行符，所以我们可以用换行符来分割payload以解决这个问题：

```
https://www.segmentfault.com/n/
1330000020470115/raw
```

X进去了。

0x0B

这道题把所有字母变成了大写：

```
function
```

在服务器上放一个文件名大写的js脚本让其引用即可，因为域名不区分大小写，这道题可以直接用它提供的js资源

```
<
```

注意这里一定要用支持https的服务器，因为题目环境是https的，不能用http引用外部文件。

0x0C

在上一题的基础上，script字符串被正则替换掉了：

```
function
```

可以双写绕过，剩下的和上一题一样。

0x0D

```
function
```

这里乍一看没有看出这个正则过滤的目的，先尝试输入

```
//使用换行符绕过前面的单行注释
```

得到：

```
<
```

由于后面跟着的 ') 违反了语法规则，alert(1);不会被执行，又发现上面的正则过滤过滤掉了 <!-- 和 // 两种可能的注释，但其实 --> 可以注释掉所在行的内容，所以：

```
//换行，绕过前面的注释
```

得到：

```
<
```

X进去了。

0x0E

```
function
```

破坏了所有的html标签，并将字母转为大写。 payload：

```
<
```

'ſ'是古英语中的s的写法, 转成大写是正常的S。

0x0F

```
function
```

这个转义过滤看似很严格，但是对html标签内部字符的转义是没有意义的（html会先把它们转义回来再解析），直接当这个过滤不存在就行。

payload：

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

0x10

```
function
```

暂时没搞清楚这个window.data的机制，这题输入alert(1);就直接过了.....

0x11

```
//
```

0x12

```
//
```

" 被转义成 " 经过html 解析后 里面变成 console.log(""), 再补个 即可

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

XSS防御

留个坑.....