

安全防范知识点

XSS

涉及面试题：什么是 XSS 攻击？如何防范 XSS 攻击？什么是 CSP？

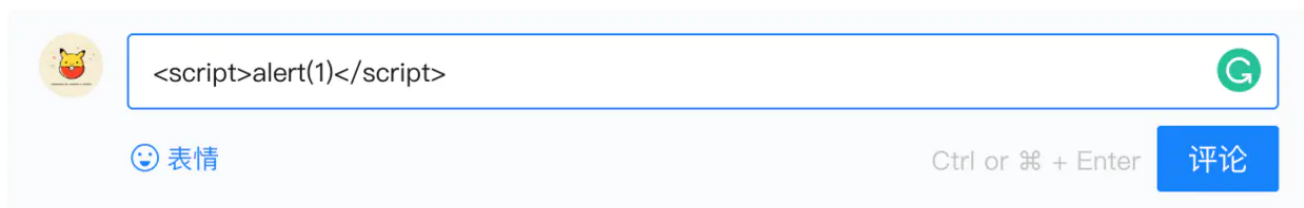
XSS 简单点来说，就是攻击者想尽一切办法将可以执行的代码注入到网页中。

XSS 可以分为多种类型，但是总体上我认为分为两类：**持久型**和**非持久型**。

持久型也就是攻击的代码被服务端写入进**数据库**中，这种攻击危害性很大，因为如果网站访问量很大的话，就会导致大量正常访问页面的用户都受到攻击。

举个例子，对于评论功能来说，就得防范持久型 XSS 攻击，因为我可以在评论中输入以下内容

评论



The image shows a comment form interface. On the left is a circular profile picture of a cat. Next to it is a text input field containing the code `<script>alert(1)</script>`. To the right of the input field is a green circular icon with a white 'G'. Below the input field is a blue button with a smiley face icon and the text '表情'. To the right of this button is the text 'Ctrl or ⌘ + Enter'. On the far right is a blue button with the text '评论'.

这种情况如果前后端没有做好防御的话，这段评论就会被存储到数据库中，这样每个打开该页面的用户都会被攻击到。

非持久型相比于前者危害就小的多了，一般通过**修改 URL 参数**的方式加入攻击代码，诱导用户访问链接从而进行攻击。

举个例子，如果页面需要从 URL 中获取某些参数作为内容的话，不经过过滤就会导致攻击代码被执行

```
<!-- http://www.domain.com?name=<script>alert(1)</script> -->
<div>{{name}}</div>
```

但是对于这种攻击方式来说，如果用户使用 Chrome 这类浏览器的话，浏览器就能自动帮助用户防御攻击。但是我们不能因此就不防御此类攻击了，因为我不能确保用户都使用了该类浏览器。



该网页无法正常工作

Chrome 在此网页上检测到了异常代码。为保护您的个人信息（例如密码、电话号码和信用卡信息），Chrome 已将该网页拦截。

请尝试[访问该网站的首页](#)。

ERR_BLOCKED_BY_XSS_AUDITOR

对于 XSS 攻击来说，通常有两种方式可以用来防御。

转义字符

首先，对于用户的输入应该是永远不信任的。最普遍的做法就是转义输入输出的内容，对于引号、尖括号、斜杠进行转义

```
function escape(str) {
  str = str.replace(/&/g, '&amp;');
  str = str.replace(/</g, '&lt;');
  str = str.replace(/>/g, '&gt;');
  str = str.replace(/"/g, '&quot;');
  str = str.replace(/'/g, '&#39;');
  str = str.replace(/`/g, '&#96;');
  str = str.replace(/\\/g, '&#x2F;');
  return str
}
```

通过转义可以将攻击代码 `<script>alert(1)</script>` 变成

```
// -> &lt;script&gt;alert(1)&lt;&#x2F;script&gt;
escape('<script>alert(1)</script>')
```

但是对于显示富文本来讲，显然不能通过上面的办法来转义所有字符，因为这样会把需要的格式也过滤掉。对于这种情况，通常采用白名单过滤的办法，当然也可以通过黑名单过滤，但是考虑到需要过滤的标签和标签属性实在太多，更加推荐使用白名单的方式。

```
const xss = require('xss')
let html = xss('<h1 id="title">XSS Demo</h1><script>alert("xss");</script>')
// -> <h1>XSS Demo</h1>&lt;script&gt;alert("xss");&lt;/script&gt;
console.log(html)
```

以上示例使用了 `js-xss` 来实现，可以看到在输出中保留了 `h1` 标签且过滤了 `script` 标签。

CSP

CSP 本质上就是建立白名单，开发者明确告诉浏览器哪些外部资源可以加载和执行。我们只需要配置规则，如何拦截是由浏览器自己实现的。我们可以通过这种方式来尽量减少 XSS 攻击。

通常可以通过两种方式来开启 CSP：

1. 设置 HTTP Header 中的 `Content-Security-Policy`
2. 设置 `meta` 标签的方式 `<meta http-equiv="Content-Security-Policy">`

这里以设置 HTTP Header 来举例

- 只允许加载本站资源

```
Content-Security-Policy: default-src 'self'
```

- 只允许加载 HTTPS 协议图片

```
Content-Security-Policy: img-src https://*
```

- 允许加载任何来源框架

```
Content-Security-Policy: child-src 'none'
```

当然可以设置的属性远不止这些，你可以通过查阅 [文档](#) 的方式来学习，这里就不过多赘述其他的属性了。

对于这种方式来说，只要开发者配置了正确的规则，那么即使网站存在漏洞，攻击者也不能执行它的攻击代码，并且 CSP 的兼容性也不错。

Mitigate cross-site scripting attacks by whitelisting allowed sources of script, style, and other resources.

Current aligned	Usage relative	Date relative	Apply filters	Show all	?										
IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Blackberry Browser	Opera Mobile *	Chrome for Android	Firefox for Android	IE Mobile	UC Browser for Android	Samsu Interr
				3.1 - 5		3.2 - 4.3									
		2 - 3.6	4 - 13	5.1		5.1									
6 - 9		4 - 22	14 - 24	6 - 6.1	10 - 12.1	6.1		2.1 - 4.3							
10	12 - 17	23 - 62	25 - 69	7 - 11.1	15 - 55	7 - 11.4		4.4 - 4.4.4	7	12 - 12.1			10		4 - 6
11	18	63	70	12	56	12	all	67	10	46	70	63	11	11.8	7.2
		64 - 65	71 - 73	TP											

CSRF

涉及面试题：什么是 CSRF 攻击？如何防范 CSRF 攻击？

CSRF 中文名为跨站请求伪造。原理就是攻击者构造出一个后端请求地址，诱导用户点击或者通过某些途径自动发起请求。如果用户是在登录状态下的话，后端就以为是用户在操作，从而进行相应的逻辑。

举个例子，假设网站中有一个通过 `GET` 请求提交用户评论的接口，那么攻击者就可以在钓鱼网站中加入一个图片，图片的地址就是评论接口

```

```

那么你是否会想到使用 `POST` 方式提交请求是不是就没有这个问题了呢？其实并不是，使用这种方式也不是百分百安全的，攻击者同样可以诱导用户进入某个页面，在页面中通过表单提交 `POST` 请求。

如何防御

防范 CSRF 攻击可以遵循以下几种规则：

1. Get 请求不对数据进行修改
2. 不让第三方网站访问到用户 Cookie
3. 阻止第三方网站请求接口
4. 请求时附带验证信息，比如验证码或者 Token

SameSite

可以对 Cookie 设置 `SameSite` 属性。该属性表示 Cookie 不随着跨域请求发送，可以很大程度减少 CSRF 的攻击，但是该属性目前并不是所有浏览器都兼容。

验证 Referer

对于需要防范 CSRF 的请求，我们可以通过验证 Referer 来判断该请求是否为第三方网站发起的。

Token

服务器下发一个随机 Token，每次发起请求时将 Token 携带上，服务器验证 Token 是否有效。

点击劫持

涉及面试题：什么是点击劫持？如何防范点击劫持？

点击劫持是一种视觉欺骗的攻击手段。攻击者将需要攻击的网站通过 `iframe` 嵌套的方式嵌入自己的网页中，并将 `iframe` 设置为透明，在页面中透出一个按钮诱导用户点击。



对于这种攻击方式，推荐防御的方法有两种。

X-FRAME-OPTIONS

`X-FRAME-OPTIONS` 是一个 HTTP 响应头，在现代浏览器有一个很好的支持。这个 HTTP 响应头 就是为了防御用 `iframe` 嵌套的点击劫持攻击。

该响应头有三个值可选，分别是

- `DENY`，表示页面不允许通过 `iframe` 的方式展示
- `SAMEORIGIN`，表示页面可以在相同域名下通过 `iframe` 的方式展示

- `ALLOW-FROM`，表示页面可以在指定来源的 `iframe` 中展示

JS 防御

对于某些远古浏览器来说，并不能支持上面的这种方式，那我们只有通过 JS 的方式来防御点击劫持了。

```
<head>
  <style id="click-jack">
    html {
      display: none !important;
    }
  </style>
</head>
<body>
  <script>
    if (self == top) {
      var style = document.getElementById('click-jack')
      document.body.removeChild(style)
    } else {
      top.location = self.location
    }
  </script>
</body>
```

以上代码的作用就是当通过 `iframe` 的方式加载页面时，攻击者的网页直接不显示所有内容了。

中间人攻击

涉及面试题：什么是中间人攻击？如何防范中间人攻击？

中间人攻击是攻击方同时与服务端和客户端建立起了连接，并让对方认为连接是安全的，但是实际上整个通信过程都被攻击者控制了。攻击者不仅能获得双方的通信信息，还能修改通信信息。

通常来说不建议使用公共的 Wi-Fi，因为很可能就会发生中间人攻击的情况。如果你在通信的过程中涉及到了某些敏感信息，就完全暴露给攻击方了。

当然防御中间人攻击其实并不难，只需要增加一个安全通道来传输信息。HTTPS 就可以用来防御中间人攻击，但是并不是说使用了 HTTPS 就可以高枕无忧了，因为如果你没有完全关闭 HTTP 访问的话，攻击方可以通过某些方式将 HTTPS 降级为 HTTP 从而实现中间人攻击。