[交换机] 学习笔记 - 5分24秒
12 浏览器基础知识点及常考面试题 学习时长：3分20秒
13 浏览器缓存机制 学习时长：4分17秒
14 浏览器重排重绘 学习时长：1分20秒
15 安全防范知识点 学习时长：1分20秒
16 从 V8 中看 JS 性能优化 学习时长：1分24秒
17 性能优化精讲事 学习时长：5分14秒
18 Webpack 性能优化 学习时长：1分24秒
19 实现小型打包工具 学习时长：1分24秒
20 React 和 Vue 两大框架之间的重要区别 学习时长：1分24秒

安全防范知识点

这一章我打算来学习安全防范这一块的知识点。总的来说安全是很复杂的一个领域，不可能通过一个章节就学习完整的安全知识。但是，一章节中，我打算学习四常见的一些安全问题及如何防范的内容，在为了其实安全问题解决提供基础，只经家用内容所关会最的技能了。

XSS

```
浏览器安全 | 什么是 XSS 攻击？如何防范 XSS 攻击？什么是 CSP？
```

XSS 简单点来说，就是攻击者想法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，但是总体上我们大分为两类：**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**数据库**中，这种危害性较大，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，就像现在很流行的一种做法：评论区中插入以下内容

评论

<script>alert(1)</script>

登录 Ctrl or ⌘ + Enter 评论

这种情况如果成功被攻击者植入代码的话，这段评论就会被存储到数据库中，这样每个打开页面的用户都会被攻击到。

非持久型相比于前者危害面小的多了，一般通过**钓鱼 URL 事件**的方式植入攻击代码，请每用户访问链接从而进行攻击。

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

```
GET /search?keyword=<script>alert(1)</script> HTTP/1.1
```

但是对于这种攻击方式来说，现在用户使用 Chrome 浏览器浏览网页的话，浏览器就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，因为用不能确保用户都使用了浏览器呢。

谷狗页无法正常运作

Chrome 无法阻止上此网页了来吧不此。为保护您的个人信息（例如密码、电话号码和信用卡信息），Chrome 已经阻止了此页。

请尝试访问此网页稍后再试。

ERR_BLOCKED_BY_XSS_AUDITOR 详细信息隐藏不此页

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，对于用户的输入这类基本法不信任的，最最最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, '&gt;')
  str = str.replace(/<g, '&lt;')
  str = str.replace(/"/g, '&quot;')
  str = str.replace(/'/g, '&#39;')
  str = str.replace(/`/g, '&#96;')
  return str
}
```

通过转义可以将简单的用 `<script>alert(1)</script>` 变成

```
// => alert(1)</script>&lt;/script&gt;
escape('<script>alert(1)</script>')
```

但是对于复杂复杂来说，虽然不能通过上面的方法来对这种有等字符，因为这样会把整个格式也过滤掉，对于这种情况，通常来说要用一个白名单的方式，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')
var html = xss('<script>alert("xss");</script>');
// => a[0-9]|b|br|div|h[1-6]|i|p|span|ul|li|ol|br|strong|
console.log(html)
```

以上示例使用了 `jo-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 其原上就是建立白名单，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很减少 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还不止这些，你可以通过查阅 文档 的方式来学习。这里简不过多赘述我的属性了。

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所攻击代码，并且 CSP 的兼容性也不错：