# 前端面试指南 — 攻击篇

# ****安全方面的面面试部分面试官喜欢问，但较少，但我们需要做好充分准备来应对，下文介绍了了一些常见的前端安全方面的知识及如何防御这些攻击，应对面试的话，基本上也算够用了。****

目录

[前端面试指南 — 攻击篇 1](#_Toc11920)

[安全方面的面面试部分面试官喜欢问，但较少，但我们需要做好充分准备来应对，下文介绍了了一些常见的前端安全方面的知识及如何防御这些攻击，应对面试的话，基本上也算够用了。 1](#_Toc13279)

[首先简单介绍几种常见的攻击方式：SQL注入 1](#_Toc23810)

[XSS 1](#_Toc16278)

[CSRF 1](#_Toc8782)

[点击劫持 1](#_Toc11640)

[中间人攻击 1](#_Toc14839)

[SQL 注入 1](#_Toc29366)

[什么是XSS， 如何防范 2](#_Toc13624)

[「如何防御XSS 」 3](#_Toc20774)

[字符转译    3](#_Toc5615)

[CSP    4](#_Toc29958)

[CSRF 4](#_Toc10710)

[「如何防御CSRF」 5](#_Toc17881)

[点击劫持 5](#_Toc28778)

[「如何防御点击劫持 」 5](#_Toc11039)

[1. 使用 HTTP 头防御 6](#_Toc26932)

[2. 使用 Javascript 防御 6](#_Toc19460)

[中间人攻击 6](#_Toc21852)

[「 如何防御中间人攻击 」 6](#_Toc18733)

# 首先简单介绍几种常见的攻击方式：

# SQL注入

# XSS

# CSRF

# 点击劫持

# 中间人攻击

# SQL 注入

这是一种比较简单的攻击方式。

如果后台人员使用用户输入的数据来组装SQL查询语句的时候不做防范， 遇到一些恶意的输入， 最后生成的SQL就会有问题。

举个例子

比如地址栏输入的是：

articlrs/index.php?id=1

发送一个get请求， 调用的查询语句是：

sql = "SELECT \* FROM articles WHERE id =", $id

正常情况下， 返回 id = 1 的文章。

如果攻击者想获得所有的文章，语句就可以改成：

articlrs/index.php?id=-1 OR 1 = 1

这样就可以了， 为什么呢？

这是因为，id = -1 永远是 false，1=1 永远是true，所有整个where语句永远是ture.

所以 where 条件相当于没有加where条件，那么查询的结果相当于整张表的内容，攻击者就达到了目的。

现在的系统一般都会加入 过滤 和  验证 机制， 可以有效预防SQL注入问题。

# 什么是XSS， 如何防范

XSS 全称是跨站脚本攻击。

通过代码注入的方式来达到攻击的目的。

举个例子

我们有个社交网站，允许大家相互访问空间，网站可能是这样做的：

<form action="" method="POST"><input type="text" name="text"><input type="submit" value="submit">input> form><h2>你输入的内容: {{{text}}}h2>

如果你用的是Chrome浏览器， 会得到来自浏览器的警告：

Chrome 这类浏览器能自动帮助用户防御攻击， 很贴心。

但是也不是说， 只要我用Chrome 就万事大吉了，该防御， 还得防御。

# 「如何防御XSS 」

对于 XSS 攻击，通常来说，有两种方式可以防御：

 字符转译

CSP(Content Security Policy)

# **字符转译**

做法就是转义输入输出的内容，对于引号、尖括号、斜杠等字符进行转义。

& 替换为：&< 替换为：&lt;> 替换为：&gt;” 替换为："‘ 替换为：&#x27;/ 替换为：&#x2f;

通过转义可以将攻击代码

<script>alert('1')script>

转译成:

&lt;script&gt;alert(1)&lt;&#x2F;script>

替换了这些字符之后，恶意代码就会失效，XSS 攻击将不会轻易发生。

对于一般的输入， 可以像上面那样粗暴的转译。

有些情况， 光转译也是不够的，比如：

<a href="{{xss}}">点我a>链接中如果存在 javacript: 开头的协议，点击链接时浏览器会执行后面的代码。这个时候光转义是没有用的，需要对 url 协议进行白名控制，只允许 http, https, http, mailto 等安全协议。包括图片 src 属性 img src="{{xss}}", iframe iframe src="{{xss}}" 都会存在这样的问题，都需要白名单处理。但是别的一些情况， 比如，富文本，这时候就不能这么干了。

 如此粗暴的转译会破坏掉原有的格式。

 这种情况， 比较合适的策略是使用白名单进行过滤标签和属性。简单总结一下：

说完字符转译， 我们再看看CSP。

## CSP

CSP , Content Security Policy 。

本质也是白名单，通过设置白名单， 我们可以设置允许浏览器加载哪些外部资源。

要开启CSP可以通过两种方式：

设置 HTTP Header 中的 Content-Security-Policy

设置 meta 标签的方式

只要配置了正确的规则，那么即使网站存在漏洞，恶意代码也不会被执行。

CSP兼容性：

CSP 文档地址：

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy

## CSRF

CSRF  全称是跨站请求伪造( Cross Site Request Forgery)本质上， 说白了就是借用用户的身份或权限偷偷的完成某些操作。CSRF 的发生其实是借助了 cookie 的特性。

举个例子

我们登录了某个 http://tao.com 购物网站之后，cookie 就会有登录过的标记了。

此时请求http://tao.com/pay?id=123&money=1000， 是会带着 cookie 的，server 端就知道已经登录了。

如果在http://tao.com去请求其他域名的 API ,  例如http://tx.com/api时，是不会带 cookie 的，这是浏览器同源策略的限制。

但是此时在其他域名的页面中，请求http://tao.com/pay?id=123&money=1000，就会带着tao.com的 cookie 。

这是发生 CSRF 攻击的理论基础。

# 「如何防御CSRF」

防御CSRF 有效的手段就是加入各个层级的权限验证.

例如现在的购物网站，只要涉及现金交易，肯定要输入密码或者扫码验证才行。

除此之外，敏感的接口要使用POST请求而不是GET.

## 点击劫持

click-jacking，也被称为UI-覆盖攻击。

这也是比较常问到的一个问题，也是我们最常见的一种情况。

攻击方式就是在某些操作的按钮上加一层透明的iframe。

点击一下， 就入坑了。

# 「如何防御点击劫持 」

常用的两种方式：

1.使用 JS 防范

判断顶层视口的域名是否是和本页面的域名一致，若是不一致就让恶意网页自动跳转到我方的网页。固然你还能够恶心一下这些恶意网站，好比说弹窗十几回，或者跳转到某些404页面。

2.使用 HTTP 头防范

经过配置 nginx 发送 X-Frame-Options 响应头，这样浏览器就会阻止嵌入网页的渲染。

## 1. 使用 HTTP 头防御

通过配置 nginx 发送 X-Frame-Options 响应头，这个 HTTP 响应头 就是为了防御用 iframe 嵌套的点击劫持攻击。 这样浏览器就会阻止嵌入网页的渲染。

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。

更详细的可以查阅 MDN 上关于 X-Frame-Options 响应头的内容。

## 2. 使用 Javascript 防御

判断顶层视口的域名是不是和本页面的域名一致，如果不一致就让恶意网页自动跳转到我方的网页。

if (top.location.hostname !== self.location.hostname) { alert("您正在访问不安全的页面，即将跳转到安全页面！") top.location.href = self.location.href; }

# 中间人攻击

中间人攻击（Man-in-the-Middle Attack, MITM）是一种由来已久的网络入侵手段，如 SMB 会话劫持、DNS 欺骗等攻击都是典型的MITM攻击。

简而言之，所谓的MITM攻击就是通过拦截正常的网络通信数据，并进行数据篡改和嗅探来达到攻击的目的，而通信的双方却毫不知情。

## 「 如何防御中间人攻击 」

以下是针对防止中间人攻击的一些建议：

确保当前你所访问的网站使用了HTTPS

如果你是一个网站管理员，你应当执行HSTS协议

不要在公共Wi-Fi上发送敏感数据

如果你的网站使用了SSL，确保你禁用了不安全的SSL/TLS协议。

不要点击恶意链接或电子邮件。

还有一个容易被忽略的漏洞

window.opener

带有 target="\_blank" 跳转的网页拥有了浏览器 window.opener 对象赋予的对原网页的跳转权限，这可能会被恶意网站利用，例如一个恶意网站在某 UGC 网站 Po 了其恶意网址，该 UGC 网站用户在新窗口打开页面时，恶意网站利用该漏洞将原 UGC 网站跳转到伪造的钓鱼页面，用户返回到原窗口时可能会忽视浏览器 URL 已发生了变化，伪造页面即可进一步进行钓鱼或其他恶意行为:代码如下

<script language="javascript">window.opener.location = 'https://example.com'script>想象一下，你在浏览淘宝的时候，点击了网页聊天窗口的一条外链，出去看了一眼，回来之后淘宝网已经变成了另一个域名的高仿网站，而你却未曾发觉，继续买东西把自己的钱直接打到骗子手里。如何修复

为 target="\_blank" 加上 rel="noopener noreferrer" 属性。

<a href="外跳的地址" rel="noopener noreferrer">外跳的地址a>缺点:  为禁止了跳转带上 referrer，目标网址没办法检测来源地址。还有一种方法是，所有的外部链接都替换为内部的跳转连接服务，点击连接时，先跳到内部地址，再由服务器 redirect 到外部网址。现在很多站点都是这么做的, 不仅可以规避风险，还可以控制非法站点的打开：

"/redirect?target=http%3A%2F%2Fxxx.yyy.com"> http://xxx.yyy.com</a>