# **web安全**

###### **编码方式**

askii码，主要收录英语和欧洲一些国家的语言，是单字节编码

Unicode编码，可以有很多不同的编码方式，旨在收录全球所有字符

utf-16是Unicode的一种使用方式，为双字节编码

utf-8是Unicode的一种使用方式，为传输而设计的编码，采用变长的编码方式，可使用1~4个字节来表示一个字符

gbk是最早的一版汉字编码，注意的是UTF-8一个中文字符占3个字节）

###### **前端中的编码**

base64

**URL编码规则**：需要编码的字符转换为UTF-8编码，然后在每个字节前面加上%。

**encodeURI**：encodeURI不编码的82个字符：!#$&’()\*+,/:;=?@-.\_~0-9a-zA-Z，从中可以看不会对url中的保留字符进行编码，所以适合url整体编码**encodeURIComponent**：对url中的保留字进行的编码，所以当传递的参数中包含这些url中的保留字（@，&,=)，就可以通过这个方法编码后传输

# **csrf（Cross-site request forgery）**

scrf,简称跨站请求伪造。

一、背景：

由于http是无状态的，每次服务器都需要收到请求参数后才能确定用户的行为，服务器并不清楚在请求前发生了什么，所以这是一个无状态协议。

但是在现实情况下很多都需要有状态，比如用户态，因为网页上很多的操作都是需要登录以后才能判断是否能够操作，那如何去判断这个用户态呢，主要就是http请求的时候都把用户态的信息带过去传给后台服务器，后台服务再根据带过来的信息判断用户合法性再进行各种操作。

那让每次http请求都带上用户信息的方式就是cookie。

cookie的特征：

* 浏览器自动携带本次域名请求的cookie
* 读写cookie有跨域的限制
* 有生命周期

二、攻击过程

因为登录态cookie的信息是浏览器默认携带的，用户访问了网站a，只要浏览器不关闭，这个网站a产生的cookie就会一直存在，此时用户去访问一个黑客网站b，黑客网站b发送http请求到网站a，那浏览器就会自动给他带上用户态cookie信息，就可以模拟用户进行一系列增删改查的操作

读操作是否会被攻击到呢？默认大部分情况下是拿不到的，除非以下两种情况：

后台返回的数据是jsonp格式的，并且是get请求，此时黑客可以拿到数据

后台通过cors处理了跨域，并且没有对请求图的origin做来源的限制

三、csrf防护

防护方式很简单，能够识别请求源是来自用户的页面还是黑客的页面就可以

1. referrer

referrer是浏览器默认带上的，含义是发送请求的网站的地址，可识别发送请求的地址是不是白名单预期的地址

1.1 referrer会不会被伪造或者篡改

* 在浏览器环境下，referrer字段是浏览器自带的，注入的js脚本是无法修改该字段的
* 浏览器自带的插件可以修改referrer字段，但这不属于csrf的攻击范畴，但是如果都已经在浏览器上装上了插件，那就有更简单的方式来进行攻击
* 通过网关或者抓包的方式可以修改refferer，但是这个是中间人攻击，需要使用https来解决
* 黑客通过自己后台代理，把请求先发到自己的后台服务，然后修改refferer再转发到需要攻击的后台，这种方式拿不到cookie，所以行不通

1.2 referrer防csrf是安全的吗

处理好以下几种情况，referrer是安全的

* 读操作：读操作不能用jsonp，cors要设置白名单，这样读操作就安全
* 写操作：referrer要求不能为空，使用白名单。不能用get请求，比如用户写日志可自定义输入图片链接，而图片的img的src或者a标签的href就指向了写操作的url，这样只要打开这篇日志，就会发送get请求，此时的请求就自动带有referrer
* 什么情况下referrer为空？
  + 地址栏直接输入url的时候，第一次请求referrer为空，一般是html页面，这种读操作不用防csrf
  + referrer-policy设置no-referrer时
  + 一些iframe的特殊使用也可能导致为空

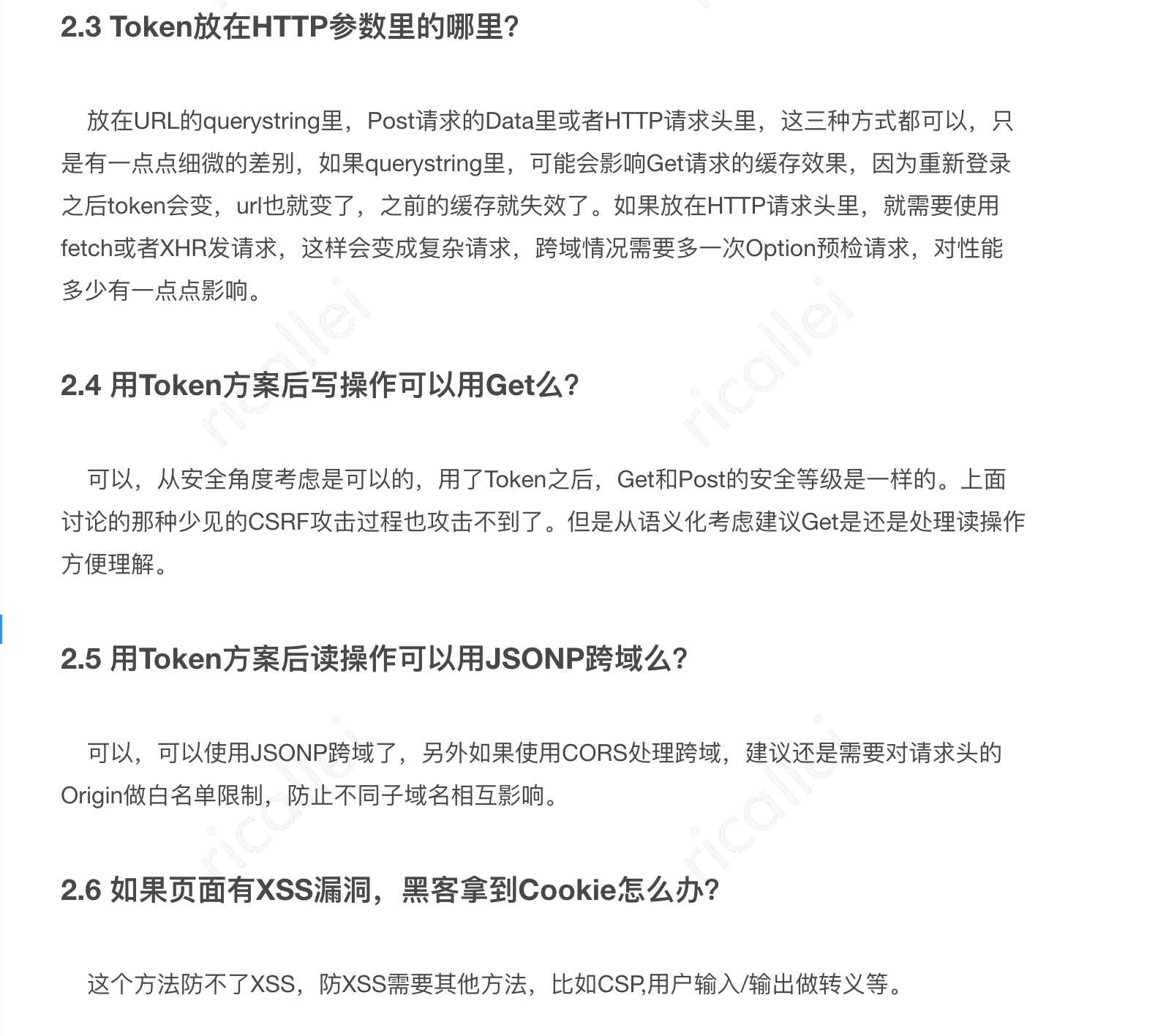
1.3用origin可以吗？

可以，和referrer原理一样，他是xhr2.0增加的，含义是发送请求页面的域名，主要是为了解决跨域问题。用他同样要设置白名单，不允许为空。

还可以自定义http请求头，升级为复杂请求，这样跨域的情况下会先发一个option请求预检

2、token

上面讲到Cookie的一些特性的第二条，读写Cookie有跨域限制(作用域，Domain,Path)，所以我们可以用这个特性来区分是自己页面还是黑客页面。只要页面能读（或者写）[www.photo.com](http://www.photo.com/)域名Cookie,就证明是自己的页面。懂了原理，方案就很简单，比如服务器通过cookie下发一个token，token值是随机数，页面发请求的时候从cookie取出token通过HTTP请求参数传给后台，后台比对参数里的token和cookie里的token是否一致，如果一致就证明是自己页面发的请求，如果不一致就返回失败。**防CSRF的方案就是这么简单，这种方法能够100%防CSRF**，但是可能会有几个变种，下面探讨几种情况。



# **xss(Cross-site scripting)**

攻击者通过网站注入点注入恶意客户端可执行的脚本，当用户访问网站时，可执行的恶意脚本通过网站执行点执行来达到某些目的，比如获取用户权限，恶意传播，钓鱼

1. 常见的xss分类

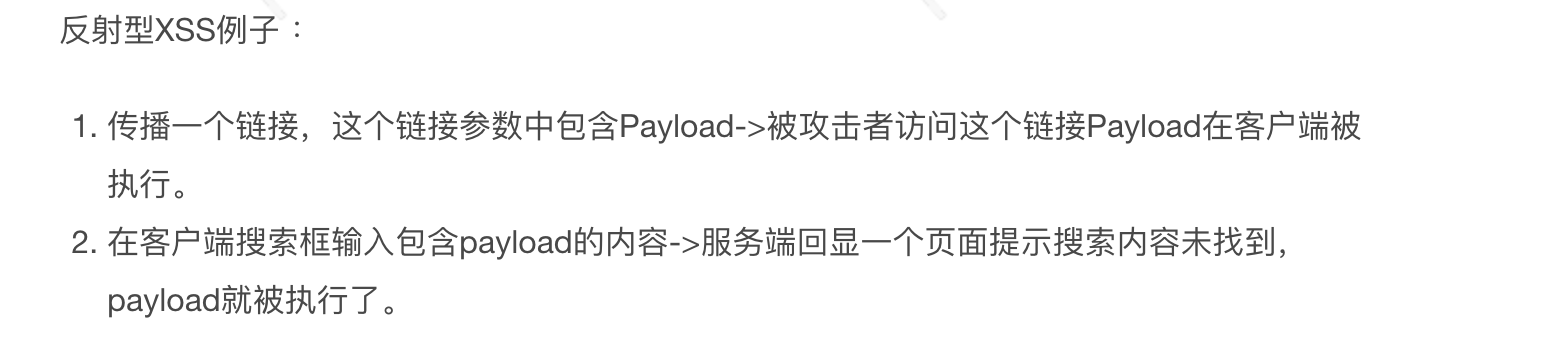
1、按照恶意代码来源划分

* 存储型xss

恶意脚本被永久存储到服务器，又称持久型xss，当浏览器请求数据时，带有恶意脚本的数据被传回并执行

* 反射型xss

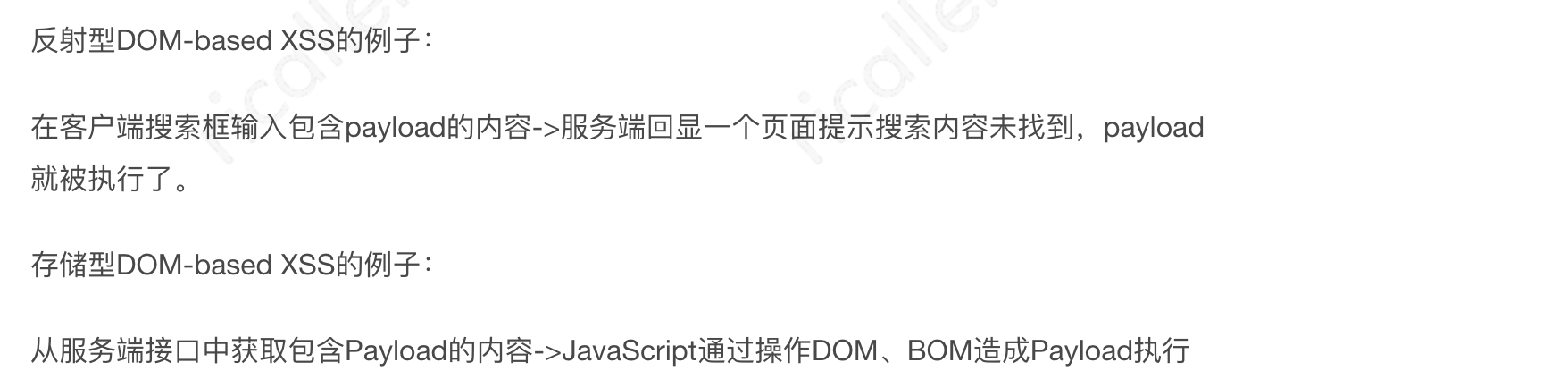
又称非持久型xss，有两种情况，一种是恶意脚本来源在客户端然后又被客户端直接执行，第二种是客户端传回服务端临时数据，再直接回显客户端立即执行

* 修改url的参数值，嵌入恶意可执行脚本，用户点击时就会触发脚本
* 在搜索框中输入恶意脚本点击搜索，脚本被执行 

2、按照恶意代码位置划分

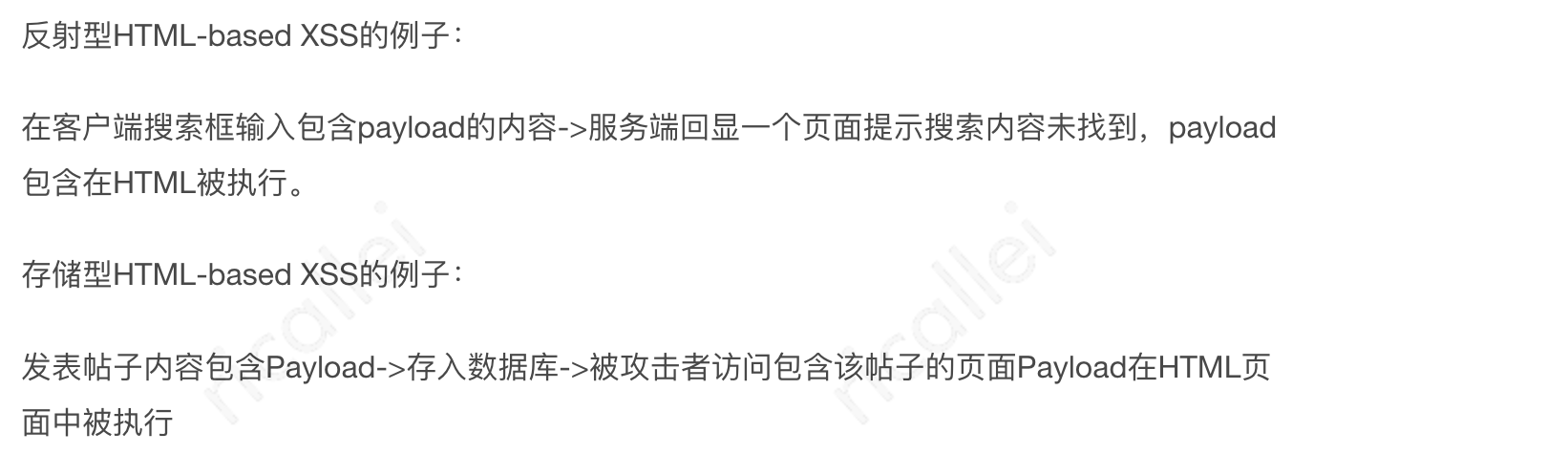
* dom-based xss

操作dom/bom导致的恶意代码被执行



* html-based xss

恶意代码包含在服务端返回的html中。在浏览器解析html的时候被执行



二、xss攻击模型分析

1. xss漏洞执行点

* 页面直出的dom
* 客户端跳转的链接location.href / location.replace() / location.assign()
* 动态脚本的执行：alert，setTimeout，setInterval
* 取值写入页面，innerHtml，document.write
* 不安全的属性设置：setAttribute。不安全属性前面见过：a标签的href、iframe的src、object的data
* postmessage来自不安全域的数据
* 不安全的第三方库

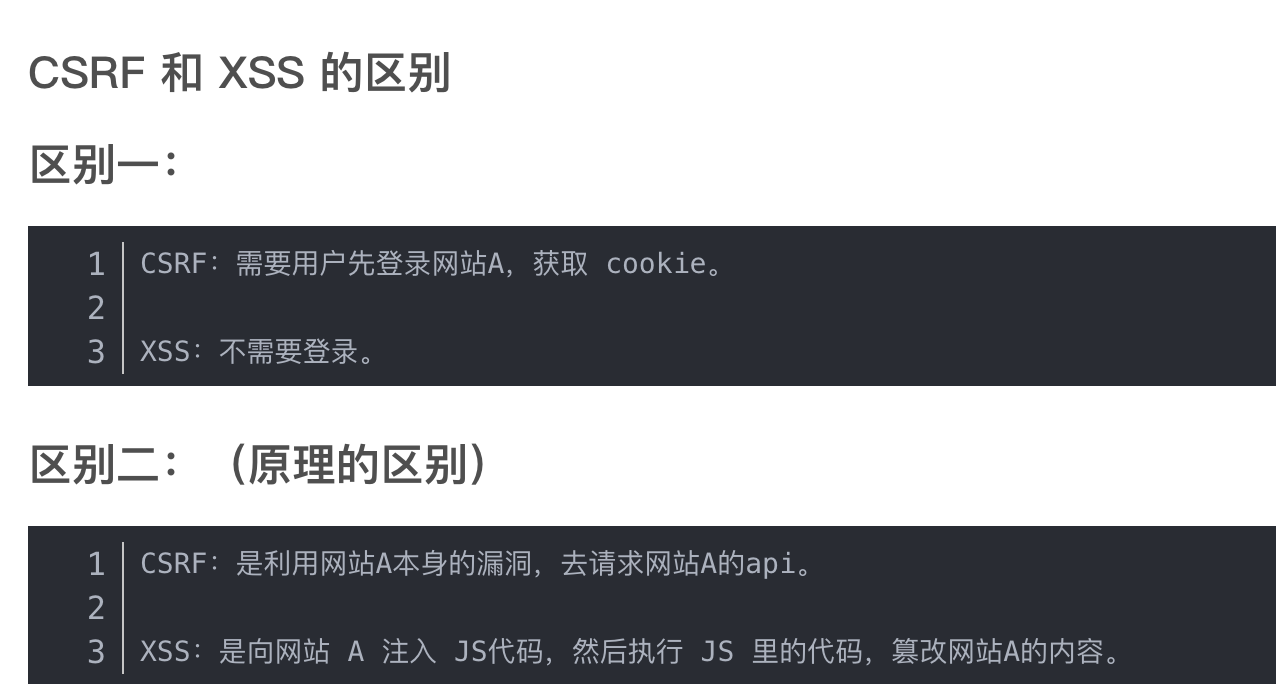
1. xss漏洞注入点

* 服务端返回数据
* 用户输入数据
* 访问链接带的参数
* 客户端存储
* 跨域调用

防范：对内容进行编码过滤矫正



xss和csrf区别



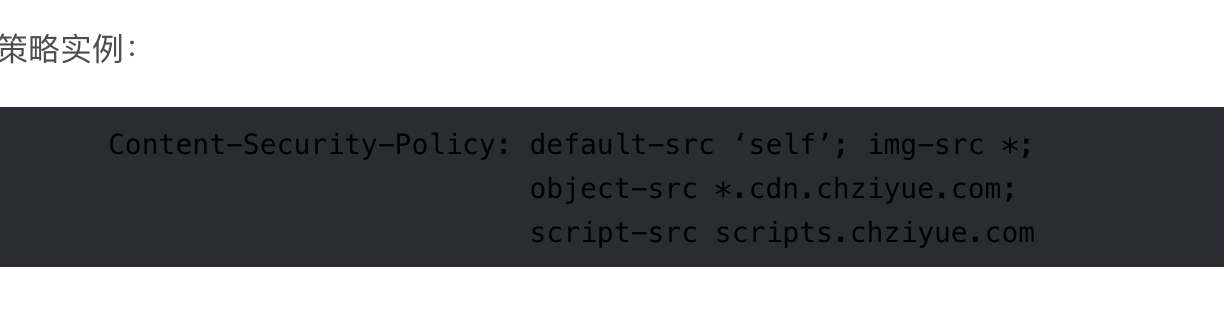
# **csp**内容安全策略（Content Security Policy）

用于帮忙检测和缓解xss的攻击

csp的本质更像是白名单制度，开发者明确告诉客户端，哪些资源可以加载哪些资源不可以加载，等同于提供白名单。它的实现和执行全部由浏览器完成，开发者只需要提供配置

通过server返回conntent-sercurity-policy头来启动不同程度的防护措施，，CSP可以完全禁止内联的JavaScript，并且控制外部代码从哪里加载。它也可以禁止动态代码执行。禁用了所有的攻击源，XSS攻击变得更加困难。

实现上可以在meta标签也可以在header头



该策略默认只允许资源从同一来源加载，但允许图像从任何URI加载，插件内容只从指定的CDN地址加载，外部脚本仅从scripts.chziyue.com加载。

**策略测试**

有关CSP的一个好处是，能够在执行一个策略的同时并行测试其他策略。这意味着，可以在比开发环境要复杂得多的生产环境中部署测试策略。

Content-Security-Policy-Report-Only响应头用于创建一个仅用于测试的策略：

Content-Security-Policy-Report-Only: default-src ‘self’

如果只启用报告的策略出错，并不会阻塞请求，但可以配置报告使得故障可被传递回起初的网站。

# **sql注入**

SQL 注入就是在用户输入的字符串中加入 SQL 语句，如果在设计不良的程序中忽略了检查，那么这些注入进去的 SQL 语句就会被数据库服务器误认为是正常的 SQL 语句而运行，攻击者就可以执行计划外的命令或访问未被授权的数据。

不能相信用户的任何输入，如果一个 sql 语句里包含了用户输入的内容，那我们要对内容做 sql 安全相关的过滤检查。同时，不使用拼凑字符串型的语句执行方式，改用占位符预处理的参数化语句。

# **中间人劫持**（英语：**M**an-**i**n-**t**he-**m**iddle attack，缩写：**MITM**）

攻击者与通信双端分别建立独立的连接，并交换其所收到的数据，使通信双端认为他们正处在一个私密的链接中通信，但事实上整个会话已经被攻击者完全控制

常见的是dns劫持，目标将其dns请求发送到了攻击者伪造的dns服务上，攻击者伪造dns响应，替换正确的ip，之后登录的就是攻击者指定的ip，而该ip上攻击者就可以实行一系列攻击

http劫持：就是在用户和服务器的信息传输之间加入其他信息，比如广告。用户发送http请求，运营商的路由器收到请求，识别到为http请求，路由器的旁路设备标记此次tcp链接为http协议后，直接返回修改后端html代码，导致浏览器被插入广告

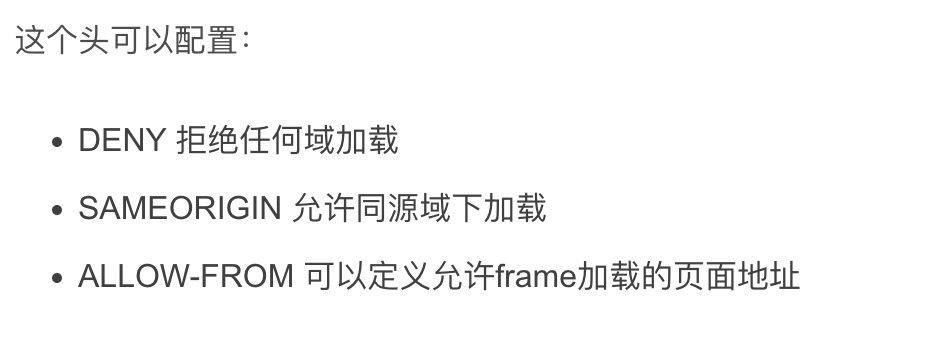
预防措施：

## descript**点击劫持（ClickJacking）**

通过网页的课件控件上覆盖不可见的框，使用户误认为在操作可见性的空间，从而实现攻击（主要利用css中的opacity、position、z-index。以及iframe）

防范：避免嵌入iframe框

* 利用js代码，禁止iframe嵌套（frame busting）
* 设置http头x-frame-options为DENY。来防御利用frame嵌套的点击劫持攻击



* csp也可以防止点劫持