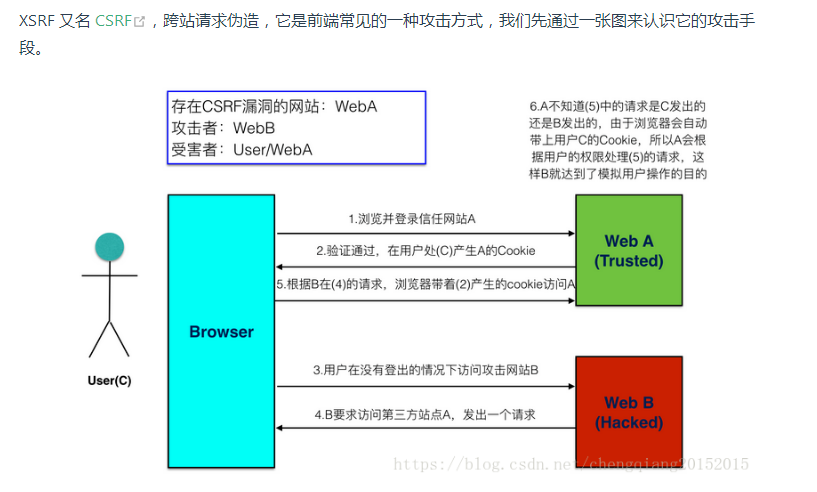
**1. xsrf**: 跨站请求伪造(Cross-site request forgery)



防御：token，Referer

CSRF 的防御手段有很多，比如验证请求的 referer，但是 referer 也是可以伪造的，所以杜绝此类攻击的一种方式是服务器端要求每次请求都包含一个 token，这个 token 不在前端生成，而是在我们每次访问站点的时候生成，并通过 set-cookie 的方式种到客户端，然后客户端发送请求的时候，从 cookie 中对应的字段读取出 token，然后添加到请求 headers 中。这样服务端就可以从请求 headers 中读取这个 token 并验证，由于这个 token 是很难伪造的，所以就能区分这个请求是否是用户正常发起的。

攻击细节

跨站请求攻击，简单地说，是攻击者通过一些技术手段欺骗用户的浏览器去访问一个自己曾经认证过的网站并运行一些操作（如发邮件，发消息，甚至财产操作如转账和购买商品）。由于浏览器曾经认证过，所以被访问的网站会认为是真正的用户操作而去运行。这利用了web中用户身份验证的一个漏洞：**简单的身份验证只能保证请求发自某个用户的浏览器，却不能保证请求本身是用户自愿发出的**。

**例子**

假如一家银行用以运行转账操作的URL地址如下：http://www.examplebank.com/withdraw?account=AccoutName&amount=1000&for=PayeeName

那么，一个恶意攻击者可以在另一个网站上放置如下代码： <img src="http://www.examplebank.com/withdraw?account=Alice&amount=1000&for=Badman">

如果有账户名为Alice的用户访问了恶意站点，而她之前刚访问过银行不久，登录信息尚未过期，那么她就会损失1000资金。

这种恶意的网址可以有很多种形式，藏身于网页中的许多地方。此外，攻击者也不需要控制放置恶意网址的网站。例如他可以将这种地址藏在论坛，博客等任何[用户生成内容](https://baike.baidu.com/item/%E7%94%A8%E6%88%B7%E7%94%9F%E6%88%90%E5%86%85%E5%AE%B9" \t "_blank)的网站中。这意味着**如果服务端没有合适的防御措施的话，用户即使访问熟悉的可信网站也有受攻击的危险**。

透过例子能够看出，攻击者并不能通过CSRF攻击来直接获取用户的账户控制权，也不能直接窃取用户的任何信息。他们能做到的，是**欺骗用户浏览器，让其以用户的名义运行操作**。

2. XSS: 跨站脚本攻击（Cross Site Scripting）

跨站脚本攻击（XSS），是最普遍的Web应用安全漏洞。这类漏洞能够使得攻击者嵌入恶意脚本代码到正常用户会访问到的页面中，当正常用户访问该页面时，则可导致嵌入的恶意脚本代码的执行，从而达到恶意攻击用户的目的。-------------恶意的代码注入

HTML是一种超文本标记语言，通过将一些字符特殊地对待来区别文本和标记，例如，小于符号（<）被看作是HTML标签的开始，<title>与</title>之间的字符是页面的标题等等。当动态页面中插入的内容含有这些特殊字符（如<）时，用户浏览器会将其误认为是插入了HTML标签，当这些HTML标签引入了一段JavaScript脚本时，这些脚本程序就将会在用户浏览器中执行。所以，当这些特殊字符不能被动态页面检查或检查出现失误时，就将会产生XSS漏洞。

**原理：恶意代码注入。**

**类型：反射型(url)、存储型(评论区)、Dom型(前端，url)。**

**防御： cookie中设置 httpOnly ，输入内容的特定字符编码。**

防御方法:

（1）不信任用户提交的任何内容，对所有用户提交内容进行可靠的输入验证，包括对URL、查询关键字、HTTP头、REFER、POST数据等，仅接受指定长度范围内、采用适当格式、采用所预期的字符的内容提交，对其他的一律过滤。尽量采用POST而非GET提交表单；对“<”，“>”，“；”，“””等字符做过滤；任何内容输出到页面之前都必须加以en-code，避免不小心把htmltag显示出来。

（2）实现Session 标记（session tokens）、CAPTCHA（验证码）系统或者HTTP引用头检查，以防功能被第三方网站所执行，对于用户提交信息的中的img等link，检查是否有重定向回本站、不是真的图片等可疑操作。

（3）cookie 防盗。避免直接在cookie中泄露用户隐私，例如email、密码，等等；通过使cookie和系统IP绑定来降低cookie泄露后的危险。这样攻击者得到的cookie没有实际价值，很难拿来直接进行重放攻击。

（4）确认接收的内容被妥善地规范化，仅包含最小的、安全的Tag（没有JavaScript），去掉任何对远程内容的引用（尤其是样式表和JavaScript），使用HTTPonly的cookie。HttpOnly是加在cookies上的一个标识，用于告诉浏览器不要向客户端脚本（document.cookie或其他）暴露cookie。**HttpOnly背后的相关议题是：当网站存在跨站脚本攻击（XSS）漏洞时,黑客通过执行脚本获得cookie时被阻止，从而在根本上杜绝这种类型的攻击。**

3. withCredientials

有些时候我们会发一些跨域请求，比如 http://domain-a.com 站点发送一个 http://api.domain-b.com/get 的请求，默认情况下，浏览器会根据同源策略限制这种跨域请求，但是可以通过 [CORS](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS" \t "_blank) 技术解决跨域问题。

在同域的情况下，我们发送请求会默认携带当前域下的 cookie，但是在跨域的情况下，默认是不会携带请求域下的 cookie 的，比如 http://domain-a.com 站点发送一个 http://api.domain-b.com/get 的请求，默认是不会携带 api.domain-b.com 域下的 cookie，如果我们想携带（很多情况下是需要的），只需要设置请求的 xhr 对象的 withCredentials 为 true 即可。

举个例子，A 网站请求了 B 网站的接口(B网站域名下的接口)，并设置 withCredientials 为 true，那么携带的是 B网站下的 cookie，但是 B 的 cookie 你是无法通过 document.cookie 或者从请求头部拿到的，这就是享受同源策略的意义。