CSRF攻击-CRLF注入-XSS攻击-XEE攻击-XML注入-sql注入-java反序列化漏洞-DOS攻击-OOM内存溢出

1. CSRF攻击：
2. CSRF概念：CSRF(Cross-site request forgery)跨站请求伪造，也被称为"One Click Attack"或者Session Riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。它与XSS非常不同，XSS利用站点内的信任用户，而CSRF则通过伪装来自受信任用户的请求（比如一个含有跨站请求的图片）来利用受信任的网站。
3. CSRF原理：假如银行站点A存在跨站漏洞
4. 用户John登录某银行站点A，进行转账操作，银行站点将用户信息通过cookie或者session等保存在客户端浏览器中，并设置了过期时间。
5. 转账完成后，John未清理cookie或者session，并打开了同一客户端浏览器另一个标签窗口查看另一网站B，而在网站B上有一个图片非常吸引John的注意（伪装），John就点击了该图片查看。
6. 而该图片中包含了John请求银行站点A转账的链接或者返回了恶意代码，如果此时包含John银行站点信息的cookie或者session未过期，客户端浏览器则携带此cookie或session发送请求到银行站点A。
7. 银行站点A误认为是John本人操作，并通过了验证，于是在John不知情的情况下发生了转账行为，在John发现钱少了后到银行查看，确实存在本人自己的转账操作。
8. CSRF的常见特性:
9. 依靠用户标识危害网站。
10. 利用网站对用户标识的信任。
11. 欺骗用户的浏览器发送HTTP请求给目标站点。
12. 另外可以通过IMG标签会触发一个GET请求，可以利用它来实现CSRF攻击。
13. 其他场景：
14. 贴图只是GET的方式，很多时候我们需要伪造POST的请求。一个办法是利用跨站，当然目标站点可能不存在跨站，这个时候我们可以从第三方网站发动攻击。
15. 比如我要攻击一个存在问题的blog，那就先去目标blog留言，留下一个网址，诱其主人点击过来(这个就要看你的忽悠本事咯:p)，然后构造个HTML表单提交些数据过去。
16. 多窗口浏览器就帮了一点忙。
17. 多窗口浏览器(firefox、遨游、MyIE……)便捷的同时也带来了一些问题，因为多窗口浏览器新开的窗口是具有当前所有会话的。即我用IE登陆了我的Blog，然后我想看新闻了，又运行一个IE进程，这个时候两个IE窗口的会话是彼此独立的，从看新闻的IE发送请求到Blog不会有我登录的cookie;但是多窗口浏览器永远都只有一个进程，各窗口的会话是通用的，即看新闻的窗口发请求到Blog是会带上我在blog登录的cookie。
18. 想一想，当我们用鼠标在Blog/BBS/WebMail点击别人留下的链接的时候，说不定一场精心准备的CSRF攻击正等着我们。
19. CSRF漏洞检测：
20. 检测CSRF漏洞是一项比较繁琐的工作，最简单的方法就是抓取一个正常请求的数据包，去掉Referer字段后再重新提交，如果该提交还有效，那么基本上可以确定存在CSRF漏洞。
21. 随着对CSRF漏洞研究的不断深入，不断涌现出一些专门针对CSRF漏洞进行检测的工具，如CSRFTester，CSRF Request Builder等。
22. 以CSRFTester工具为例，CSRF漏洞检测工具的测试原理如下：使用CSRFTester进行测试时，首先需要抓取我们在浏览器中访问过的所有链接以及所有的表单等信息，然后通过在CSRFTester中修改相应的表单等信息，重新提交，这相当于一次伪造客户端请求。如果修改后的测试请求成功被网站服务器接受，则说明存在CSRF漏洞，当然此款工具也可以被用来进行CSRF攻击。

（摘自：<https://blog.csdn.net/stpeace/article/details/53512283>）

1. CSRF防御：

  目前防御 CSRF 攻击主要有三种策略：验证 HTTP Referer 字段；在请求地址中添加 token 并验证；在 HTTP 头中自定义属性并验证。

1. **验证 HTTP Referer 字段**

根据 HTTP 协议，在 HTTP 头中有一个字段叫 Referer，它记录了该 HTTP 请求的来源地址。在通常情况下，访问一个安全受限页面的请求来自于同一个网站，比如需要访问 http://bank.example/withdraw?account=bob&amount=1000000&for=Mallory，用户必须先登陆 bank.example，然后通过点击页面上的按钮来触发转账事件。这时，该转帐请求的 Referer 值就会是转账按钮所在的页面的 URL，通常是以 bank.example 域名开头的地址。而如果黑客要对银行网站实施 CSRF 攻击，他只能在他自己的网站构造请求，当用户通过黑客的网站发送请求到银行时，该请求的 Referer 是指向黑客自己的网站。因此，要防御 CSRF 攻击，银行网站只需要对于每一个转账请求验证其 Referer 值，如果是以 bank.example 开头的域名，则说明该请求是来自银行网站自己的请求，是合法的。如果 Referer 是其他网站的话，则有可能是黑客的 CSRF 攻击，拒绝该请求。

这种方法的显而易见的好处就是简单易行，网站的普通开发人员不需要操心 CSRF 的漏洞，只需要在最后给所有安全敏感的请求统一增加一个拦截器来检查 Referer 的值就可以。特别是对于当前现有的系统，不需要改变当前系统的任何已有代码和逻辑，没有风险，非常便捷。

然而，这种方法并非万无一失。Referer 的值是由浏览器提供的，虽然 HTTP 协议上有明确的要求，但是每个浏览器对于 Referer 的具体实现可能有差别，并不能保证浏览器自身没有安全漏洞。使用验证 Referer 值的方法，就是把安全性都依赖于第三方（即浏览器）来保障，从理论上来讲，这样并不安全。事实上，对于某些浏览器，比如 IE6 或 FF2，目前已经有一些方法可以篡改 Referer 值。如果 bank.example 网站支持 IE6 浏览器，黑客完全可以把用户浏览器的 Referer 值设为以 bank.example 域名开头的地址，这样就可以通过验证，从而进行 CSRF 攻击。

即便是使用最新的浏览器，黑客无法篡改 Referer 值，这种方法仍然有问题。因为 Referer 值会记录下用户的访问来源，有些用户认为这样会侵犯到他们自己的隐私权，特别是有些组织担心 Referer 值会把组织内网中的某些信息泄露到外网中。因此，用户自己可以设置浏览器使其在发送请求时不再提供 Referer。当他们正常访问银行网站时，网站会因为请求没有 Referer 值而认为是 CSRF 攻击，拒绝合法用户的访问。

1. **在请求地址中添加 token 并验证**

 CSRF 攻击之所以能够成功，是因为黑客可以完全伪造用户的请求，该请求中所有的用户验证信息都是存在于 cookie 中，因此黑客可以在不知道这些验证信息的情况下直接利用用户自己的 cookie 来通过安全验证。要抵御 CSRF，关键在于在请求中放入黑客所不能伪造的信息，并且该信息不存在于 cookie 之中。可以在 HTTP 请求中以参数的形式加入一个随机产生的 token，并在服务器端建立一个拦截器来验证这个 token，如果请求中没有 token 或者 token 内容不正确，则认为可能是 CSRF 攻击而拒绝该请求。

这种方法要比检查 Referer 要安全一些，token 可以在用户登陆后产生并放于 session 之中，然后在每次请求时把 token 从 session 中拿出，与请求中的 token 进行比对，但这种方法的难点在于如何把 token 以参数的形式加入请求。对于 GET 请求，token 将附在请求地址之后，这样 URL 就变成 http://url?csrftoken=tokenvalue。 而对于 POST 请求来说，要在 form 的最后加上 <input type=”hidden” name=”csrftoken” value=”tokenvalue”/>，这样就把 token 以参数的形式加入请求了。但是，在一个网站中，可以接受请求的地方非常多，要对于每一个请求都加上 token 是很麻烦的，并且很容易漏掉，通常使用的方法就是在每次页面加载时，使用 javascript 遍历整个 dom 树，对于 dom 中所有的 a 和 form 标签后加入 token。这样可以解决大部分的请求，但是对于在页面加载之后动态生成的 html 代码，这种方法就没有作用，还需要程序员在编码时手动添加 token。

  该方法还有一个缺点是难以保证 token 本身的安全。特别是在一些论坛之类支持用户自己发表内容的网站，黑客可以在上面发布自己个人网站的地址。由于系统也会在这个地址后面加上 token，黑客可以在自己的网站上得到这个 token，并马上就可以发动 CSRF 攻击。为了避免这一点，系统可以在添加 token 的时候增加一个判断，如果这个链接是链到自己本站的，就在后面添加 token，如果是通向外网则不加。不过，即使这个 csrftoken 不以参数的形式附加在请求之中，黑客的网站也同样可以通过 Referer 来得到这个 token 值以发动 CSRF 攻击。这也是一些用户喜欢手动关闭浏览器 Referer 功能的原因。

1. **在 HTTP 头中自定义属性并验证**

这种方法也是使用 token 并进行验证，和上一种方法不同的是，这里并不是把 token 以参数的形式置于 HTTP 请求之中，而是把它放到 HTTP 头中自定义的属性里。通过 XMLHttpRequest 这个类，可以一次性给所有该类请求加上 csrftoken 这个 HTTP 头属性，并把 token 值放入其中。这样解决了上种方法在请求中加入 token 的不便，同时，通过 XMLHttpRequest 请求的地址不会被记录到浏览器的地址栏，也不用担心 token 会透过 Referer 泄露到其他网站中去。

然而这种方法的局限性非常大。XMLHttpRequest 请求通常用于 Ajax 方法中对于页面局部的异步刷新，并非所有的请求都适合用这个类来发起，而且通过该类请求得到的页面不能被浏览器所记录下，从而进行前进，后退，刷新，收藏等操作，给用户带来不便。另外，对于没有进行 CSRF 防护的遗留系统来说，要采用这种方法来进行防护，要把所有请求都改为 XMLHttpRequest 请求，这样几乎是要重写整个网站，这代价无疑是不能接受的。

（摘自：<https://blog.csdn.net/stpeace/article/details/53512283>）

1. CRLF注入：
2. CRLF概念：CRLF是Carriage-Return Line-Feed的缩写，意思是回车换行，就是回车(CR, ASCII 13, \r) 换行(LF, ASCII 10, \n)。换行在有的[ASCII](https://baike.so.com/doc/7103239-7326232.html" \t "_blank)码表也用newline(简nl)来进行表示,这里的lf是line feed的概念，意思是一样的。可以导致日志注入
3. CRLF注入原理：
4. CRLF原理：
5. XSS：跨站脚本攻击

Html页面漏洞，导致特殊字符的输入，而构建出脚本攻击，可以通过输入校验来预防

1. XEE(xml entity extend):XML实体扩展

限制实体大小，禁止实体解析，

1. XXE：XML外部实体注入

限制实体大小，禁止实体解析，

1. XPATH:

通过构建XML节点而获取权限，

1. SQL注入：
2. DOS攻击和reDOS：
3. 正则注入：
4. OOM内存溢出：
5. OS注入