【漏洞利用】CSRF跨站请求伪造-详解

原创

白丁Gorilla 于 2021-03-03 21:46:43 发布 229 收藏 1
分类专栏： 网络安全 文章标签： 安全 网络安全

参考文章

- CSRF攻击与防御
- CSRF & CORS
- 读取型CSRF-需要交互的内容劫持
- Tag：
- Ref：

本片文章仅供学习使用，切勿触犯法律！
未写完，待补充

概述

总结

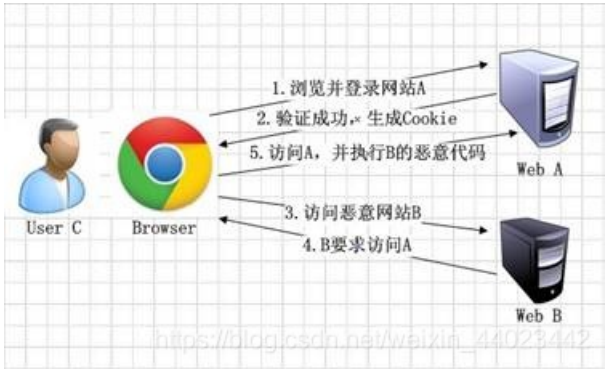
一、漏洞介绍

CSRF(Cross-site request forgery，跨站请求伪造)也被称为One Click Attack或者Session Riding，是一种对网站的恶意利用。==它与XSS非常不同，XSS利用站点内信任用户，而CSRF则通过伪装成受信任用户请求受信任的网站。==与XSS攻击相比，CSRF攻击往往不大流行，也难以防范，所以被认为比XSS更具有危险性。

二、漏洞原理

攻击者利用目标用户的身份，以目标用户的名义执行某些非法操作。

一般流程



1. 用户C打开浏览器，访问受信任网站A，输入用户名和密码请求登录网站A；
2. 在用户信息通过验证后，网站A产生Cookie信息并返回给浏览器，此时用户登录网站A成功，可以正常发送请求到网站A；
3. 用户未退出网站A之前，在同一浏览器中，打开一个TAB页访问网站B；
4. 网站B接收到用户请求后，返回一些攻击性代码，并发出一个请求要求访问第三方站点A；
5. 浏览器在接收到这些攻击性代码后，根据网站 B 的请求，在用户不知情的情况下携带 Cookie 信息，向网站 A 发出请求。网站 A 并不知道该请求其实是由 B 发起的，所以会根据用户 C 的 Cookie 信息以 C 的权限处理该请求，导致来自网站 B 的恶意代码被执行。

三、漏洞危害

- 用来制作蠕虫攻击、刷SEO流量等
- 冒用目标用户的身份认证，可能导致目标用户的财产损失

四、利用前提

- 目标用户已经登录了网站，能够执行网站的功能
- 目标用户访问了攻击者构造的URL

五、挖掘利用

1、站外型（一般类型）

1.描述

CSRF站外类型的漏洞本质上就是传统意义上的外部提交数据问题。

2.挖掘

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。或使用CSRF检测工具，例如CSRFTester[[016.CSRFTester]]

3.利用

1. 确认存在CSRF漏洞，抓包请求包
2. 右击存在CSRF漏洞的请求包，使用burpsuite的自动构造CSRF PoC功能（右击–Engagement tool–Generate PoC）
   ![[Pasted image 20210303174700.png]]
   burpsuite会生成一段HTML代码，此HTML代码即为CSRF漏洞的测试代码。
   ![[Pasted image 20210303174941.png]]
3. 将CSRF测试代码发布到一个网站中，例如：`http://aaa.com/1.html`
4. 诱导目标用户访问`http://aaa.com/1.html`，如果目标用户处于登录状态，并且在同一个浏览器访问了该网址后，就会以目标用户的身份认证去执行测试HMTL的内容。

2、站内型

1.描述

CSRF站内类型的漏洞在一定程度上是由于程序员滥用REQUEST类变量造成的。在一些敏感的操作中（如修改密码、添加用户等），本来要求用户从表单提交发起POST请求传递参数给程序，但是由于使用了_REQUEST等变量，程序除支持接收POST请求传递的参数外也支持接收GET请求传递的参数，这样就会为攻击者使用CSRF攻击创造条件。一般攻击者只要把预测的请求参数放在站内一个贴子或者留言的图片链接里，受害者浏览了这样的页面就会被强迫发起这些请求。
该漏洞类型不清楚，待进一步研究

2.挖掘

3.利用

六、修复防范

目前防御 CSRF 攻击主要有以下几种策略：

- 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。
- 对敏感信息的操作增加安全的逻辑流程、比如在修改密码时，先校验旧密码

1、验证HTTP Referer 字段

要防御 CSRF 攻击，网站只需要对于每一个请求验证其 Referer 值，如果是以 自己的域名，则说明该请求是来自网站自己的请求，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。

优点
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。

缺点
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。

2、在请求地址中添加 token 并验证

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。

优点

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

缺点

1. 这种方法的难点在于如何把 token 以参数的形式加入请求。
   解决办法：
   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。
2. 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
   解决办法：
   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。
3. 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。
   解决办法：
   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。

3、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。

优点

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。

缺点

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。

七、提出问题

挖掘、利用方式不清晰。