## [浅谈CSRF攻击方式](https://www.cnblogs.com/hyddd/archive/2009/04/09/1432744.html)

2009-04-09 22:44 by hyddd, 267425 阅读, 128 评论, [收藏](http://www.cnblogs.com/hyddd/archive/2009/04/09/1432744.html), [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=1432744)

**一.CSRF是什么？**

　　CSRF（Cross-site request forgery），中文名称：跨站请求伪造，也被称为：one click attack/session riding，缩写为：CSRF/XSRF。

**二.CSRF可以做什么？**

　　你这可以这么理解CSRF攻击：攻击者盗用了你的身份，以你的名义发送恶意请求。CSRF能够做的事情包括：以你名义发送邮件，发消息，盗取你的账号，甚至于购买商品，虚拟货币转账......造成的问题包括：个人隐私泄露以及财产安全。

**三.CSRF漏洞现状**

　　CSRF这种攻击方式在2000年已经被国外的安全人员提出，但在国内，直到06年才开始被关注，08年，国内外的多个大型社区和交互网站分别爆出CSRF漏洞，如：NYTimes.com（纽约时报）、Metafilter（一个大型的BLOG网站），YouTube和百度HI......而现在，互联网上的许多站点仍对此毫无防备，以至于安全业界称CSRF为“沉睡的巨人”。

**四.CSRF的原理**

　　下图简单阐述了CSRF攻击的思想：



　　从上图可以看出，要完成一次CSRF攻击，受害者必须依次完成两个步骤：

　　1.登录受信任网站A，并在本地生成Cookie。

　　2.在不登出A的情况下，访问危险网站B。

　　看到这里，你也许会说：“如果我不满足以上两个条件中的一个，我就不会受到CSRF的攻击”。是的，确实如此，但你不能保证以下情况不会发生：

　　1.你不能保证你登录了一个网站后，不再打开一个tab页面并访问另外的网站。

　　2.你不能保证你关闭浏览器了后，你本地的Cookie立刻过期，你上次的会话已经结束。（事实上，关闭浏览器不能结束一个会话，但大多数人都会错误的认为关闭浏览器就等于退出登录/结束会话了......）

　　3.上图中所谓的攻击网站，可能是一个存在其他漏洞的可信任的经常被人访问的网站。

　　上面大概地讲了一下CSRF攻击的思想，下面我将用几个例子详细说说具体的CSRF攻击，这里我以一个银行转账的操作作为例子（仅仅是例子，真实的银行网站没这么傻:>）

**示例1：**

　　银行网站A，它以GET请求来完成银行转账的操作，如：http://www.mybank.com/Transfer.php?toBankId=11&money=1000

　　危险网站B，它里面有一段HTML的代码如下：

　　<img src=http://www.mybank.com/Transfer.php?toBankId=11&money=1000>

　　首先，你登录了银行网站A，然后访问危险网站B，噢，这时你会发现你的银行账户少了1000块......

　　为什么会这样呢？原因是银行网站A违反了HTTP规范，使用GET请求更新资源。在访问危险网站B的之前，你已经登录了银行网站A，而B中的<img>以GET的方式请求第三方资源（这里的第三方就是指银行网站了，原本这是一个合法的请求，但这里被不法分子利用了），所以你的浏览器会带上你的银行网站A的Cookie发出Get请求，去获取资源“http://www.mybank.com/Transfer.php?toBankId=11&money=1000”，结果银行网站服务器收到请求后，认为这是一个更新资源操作（转账操作），所以就立刻进行转账操作......

**示例2：**

　　为了杜绝上面的问题，银行决定改用POST请求完成转账操作。

　　银行网站A的WEB表单如下：

　　<form action="Transfer.php" method="POST">  
　　　　<p>ToBankId: <input type="text" name="toBankId" /></p>  
　　　　<p>Money: <input type="text" name="money" /></p>  
　　　　<p><input type="submit" value="Transfer" /></p>  
　　</form>

　　后台处理页面Transfer.php如下：

[IMG_257](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

　　<?php  
　　　　session\_start();  
　　　　if (isset($\_REQUEST['toBankId'] &&　isset($\_REQUEST['money']))  
　　　　{  
　　　　    buy\_stocks($\_REQUEST['toBankId'],　$\_REQUEST['money']);  
　　　　}  
　　?>

[IMG_258](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

　　危险网站B，仍然只是包含那句HTML代码：

　　<img src=http://www.mybank.com/Transfer.php?toBankId=11&money=1000>

　　和示例1中的操作一样，你首先登录了银行网站A，然后访问危险网站B，结果.....和示例1一样，你再次没了1000块～T\_T，这次事故的原因是：银行后台使用了$\_REQUEST去获取请求的数据，而$\_REQUEST既可以获取GET请求的数据，也可以获取POST请求的数据，这就造成了在后台处理程序无法区分这到底是GET请求的数据还是POST请求的数据。在PHP中，可以使用$\_GET和$\_POST分别获取GET请求和POST请求的数据。在JAVA中，用于获取请求数据request一样存在不能区分GET请求数据和POST数据的问题。

**示例3：**

　　经过前面2个惨痛的教训，银行决定把获取请求数据的方法也改了，改用$\_POST，只获取POST请求的数据，后台处理页面Transfer.php代码如下：

[IMG_259](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

　　<?php  
　　　　session\_start();  
　　　　if (isset($\_POST['toBankId'] &&　isset($\_POST['money']))  
　　　　{  
　　　　    buy\_stocks($\_POST['toBankId'],　$\_POST['money']);  
　　　　}  
　　?>

[IMG_260](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

　　然而，危险网站B与时俱进，它改了一下代码：

[IMG_261](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

<html>  
　　<head>  
　　　　<script type="text/javascript">  
　　　　　　function steal()  
　　　　　　{  
          　　　　 iframe = document.frames["steal"];  
　　     　　      iframe.document.Submit("transfer");  
　　　　　　}  
　　　　</script>  
　　</head>  
  
　　<body onload="steal()">  
　　　　<iframe name="steal" display="none">  
　　　　　　<form method="POST" name="transfer"　action="http://www.myBank.com/Transfer.php">  
　　　　　　　　<input type="hidden" name="toBankId" value="11">  
　　　　　　　　<input type="hidden" name="money" value="1000">  
　　　　　　</form>  
　　　　</iframe>  
　　</body>  
</html>

[IMG_262](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

如果用户仍是继续上面的操作，很不幸，结果将会是再次不见1000块......因为这里危险网站B暗地里发送了POST请求到银行!

　　总结一下上面3个例子，CSRF主要的攻击模式基本上是以上的3种，其中以第1,2种最为严重，因为触发条件很简单，一个<img>就可以了，而第3种比较麻烦，需要使用JavaScript，所以使用的机会会比前面的少很多，但无论是哪种情况，只要触发了CSRF攻击，后果都有可能很严重。

　　理解上面的3种攻击模式，其实可以看出，CSRF攻击是源于WEB的隐式身份验证机制！WEB的身份验证机制虽然可以保证一个请求是来自于某个用户的浏览器，但却无法保证该请求是用户批准发送的！

**五.CSRF的防御**

　　我总结了一下看到的资料，CSRF的防御可以从服务端和客户端两方面着手，防御效果是从服务端着手效果比较好，现在一般的CSRF防御也都在服务端进行。

**1.服务端进行CSRF防御**

　　服务端的CSRF方式方法很多样，但总的思想都是一致的，就是在客户端页面增加伪随机数。

　　(1).Cookie Hashing(所有表单都包含同一个伪随机值)：

　　这可能是最简单的解决方案了，因为攻击者不能获得第三方的Cookie(理论上)，所以表单中的数据也就构造失败了:>

　　<?php  
　　　　//构造加密的Cookie信息  
　　　　$value = “DefenseSCRF”;  
　　　　setcookie(”cookie”, $value, time()+3600);  
　　?>

　　在表单里增加Hash值，以认证这确实是用户发送的请求。

[IMG_263](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

　　<?php  
　　　　$hash = md5($\_COOKIE['cookie']);  
　　?>  
　　<form method=”POST” action=”transfer.php”>  
　　　　<input type=”text” name=”toBankId”>  
　　　　<input type=”text” name=”money”>  
　　　　<input type=”hidden” name=”hash” value=”<?=$hash;?>”>  
　　　　<input type=”submit” name=”submit” value=”Submit”>  
　　</form>

[IMG_264](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

　　然后在服务器端进行Hash值验证

[IMG_265](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

      <?php  
　　      if(isset($\_POST['check'])) {  
     　　      $hash = md5($\_COOKIE['cookie']);  
          　　 if($\_POST['check'] == $hash) {  
               　　 doJob();  
　　           } else {  
　　　　　　　　//...  
          　　 }  
　　      } else {  
　　　　　　//...  
　　      }  
      ?>

[IMG_266](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

　　这个方法个人觉得已经可以杜绝99%的CSRF攻击了，那还有1%呢....由于用户的Cookie很容易由于网站的XSS漏洞而被盗取，这就另外的1%。一般的攻击者看到有需要算Hash值，基本都会放弃了，某些除外，所以如果需要100%的杜绝，这个不是最好的方法。  
　　(2).验证码

　　这个方案的思路是：每次的用户提交都需要用户在表单中填写一个图片上的随机字符串，厄....这个方案可以完全解决CSRF，但个人觉得在易用性方面似乎不是太好，还有听闻是验证码图片的使用涉及了一个被称为MHTML的Bug，可能在某些版本的微软IE中受影响。

　　(3).One-Time Tokens(不同的表单包含一个不同的伪随机值)

　　在实现One-Time Tokens时，需要注意一点：就是“并行会话的兼容”。如果用户在一个站点上同时打开了两个不同的表单，CSRF保护措施不应该影响到他对任何表单的提交。考虑一下如果每次表单被装入时站点生成一个伪随机值来覆盖以前的伪随机值将会发生什么情况：用户只能成功地提交他最后打开的表单，因为所有其他的表单都含有非法的伪随机值。必须小心操作以确保CSRF保护措施不会影响选项卡式的浏览或者利用多个浏览器窗口浏览一个站点。

　　以下我的实现:

　　1).先是令牌生成函数(gen\_token())：

[IMG_267](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

     <?php  
     function gen\_token() {  
　　　　//这里我是贪方便，实际上单使用Rand()得出的随机数作为令牌，也是不安全的。  
　　　　//这个可以参考我写的Findbugs笔记中的[《Random object created and used only once》](http://www.cnblogs.com/hyddd/articles/1391737.html)  
          $token = md5(uniqid(rand(), true));  
          return $token;  
     }

[IMG_268](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

　　2).然后是Session令牌生成函数(gen\_stoken())：

[IMG_269](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

     <?php  
     　　function gen\_stoken() {  
　　　　　　$pToken = "";  
　　　　　　if($\_SESSION[STOKEN\_NAME]  == $pToken){  
　　　　　　　　//没有值，赋新值  
　　　　　　　　$\_SESSION[STOKEN\_NAME] = gen\_token();  
　　　　　　}      
　　　　　　else{  
　　　　　　　　//继续使用旧的值  
　　　　　　}  
     　　}  
     ?>

[IMG_270](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

　　3).WEB表单生成隐藏输入域的函数：

[IMG_271](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

     <?php  
　　     function gen\_input() {  
     　　     gen\_stoken();  
　　          echo “<input type=\”hidden\” name=\”" . FTOKEN\_NAME . “\”  
          　　     value=\”" . $\_SESSION[STOKEN\_NAME] . “\”> “;  
     　　}  
     ?>

[IMG_272](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

　　4).WEB表单结构：

[IMG_273](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

     <?php  
          session\_start();  
          include(”functions.php”);  
     ?>  
     <form method=”POST” action=”transfer.php”>  
          <input type=”text” name=”toBankId”>  
          <input type=”text” name=”money”>  
          <? gen\_input(); ?>  
          <input type=”submit” name=”submit” value=”Submit”>  
     </FORM>

[IMG_274](http://www.cnblogs.com/hyddd/archive/2009/04/09/javascript:void(0);)

　　5).服务端核对令牌：

　　这个很简单，这里就不再啰嗦了。

　　上面这个其实不完全符合“并行会话的兼容”的规则，大家可以在此基础上修改。

　　其实还有很多想写，无奈精力有限，暂且打住，日后补充，如果错漏，请指出:>

　　PS：今天下午写这篇文档的时候FF崩溃了一次，写了一半文章的全没了，郁闷好久T\_T.......

　　转载请说明出处，谢谢[hyddd(http://www.cnblogs.com/hyddd/)]

**六.参考文献**

[1].[Preventing CSRF](http://www.playhack.net/view.php?id=31)

[2].[Security Corner: Cross-Site Request Forgeries](http://shiflett.org/articles/cross-site-request-forgeries)

[3].[《深入解析跨站请求伪造漏洞：原理剖析》](http://netsecurity.51cto.com/art/200812/102951.htm)

[4].[《Web安全测试之跨站请求伪造（CSRF）》](http://netsecurity.51cto.com/art/200811/97281.htm)

[5].[《深入解析跨站请求伪造漏洞：实例讲解》](http://netsecurity.51cto.com/art/200812/102925.htm)

[6].http://baike.baidu.com/view/1609487.htm