Django-CSRF

转载

weixin_33804582 于 2017-02-04 09:50:00 发布 53 收藏
文章标签： python xhtml
原文链接：https://my.oschina.net/yuyang/blog/831445

最近新接手一个项目，第一个任务就是解决安全检查发现的问题，其中一个棘手的是CSRF处理。CSRF之前只是听过，了解过一些皮毛，却未实际处理过。使用Django时，要么直接绕过，去掉中间件不用，要么就是加@csrf_exempt屏蔽掉。 因为不理解，所以心里犯怵，故尔避之。 可是，这次是真逃不掉了。。。。而且，这次还要麻烦些，是Django + Jinja2结合的环境。。。。。 没办法，那就上吧，还能难上天，哼哼！ ☺

首先，先来理解下什么是CSRF？

文章目录

理解CSRF

以下内容摘抄自 http://www.django-china.cn/topic/580/ （文字直白，内容浅显易懂，是我喜欢的类型^_^）

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，是一种对网站的恶意利用。尽管听起来像跨站脚本（XSS），但它与XSS非常不同，并且攻击方式几乎相左。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来利用受信任的网站。与XSS攻击相比，CSRF攻击往往不大流行（因此对其进行防范的资源也相当稀少）和难以防范，所以被认为比XSS更具危险性。

CSRF可以做什么？

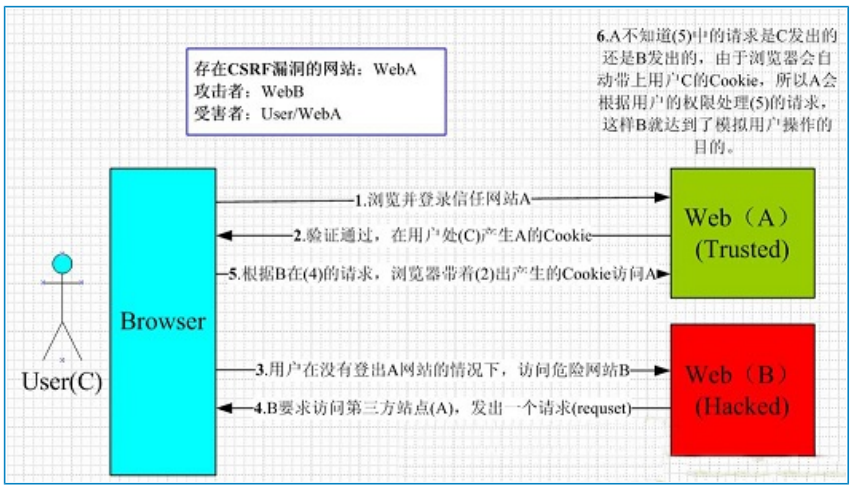
你这可以这么理解CSRF攻击：攻击者盗用了你的身份，以你的名义发送恶意请求。CSRF能够做的事情包括：以你名义发送邮件，发消息，盗取你的账号，甚至于购买商品，虚拟货币转账……造成的问题包括：个人隐私泄露以及财产安全。

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”。

CSRF的原理

下图简单阐述了CSRF攻击的思想：



CSRF攻击

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。

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！

CSRF的防御

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。
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。
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。

Django解决办法

原理

Django使用专门的中间件（CsrfMiddleware）来进行CSRF防护。具体的原理如下：

> 它修改当前处理的请求，向所有的 POST 表单增添一个隐藏的表单字段，使用名称是 csrfmiddlewaretoken，值为当前会话 ID 加上一个密钥的散列值。 如果未设置会话 ID，该中间件将不会修改响应结果，因此对于未使用会话的请求来说性能损失是可以忽略的。
>
> 对于所有含会话 cookie 集合的传入 POST 请求，它将检查是否存在 csrfmiddlewaretoken 及其是否正确。 如果不是的话，用户将会收到一个 403 HTTP 错误。 403 错误页面的内容是检测到了跨域请求伪装。 终止请求。
> 该步骤确保只有源自你的站点的表单才能将数据 POST 回来。

另外要说明的是，未使用会话 cookie 的 POST 请求无法受到保护，但它们也不需要受到保护，因为恶意网站可用任意方法来制造这种请求。为了避免转换非 HTML 请求，中间件在编辑响应结果之前对它的 Content-Type 头标进行检查。 只有标记为 text/html 或 application/xml+xhtml 的页面才会被修改。

具体操作

form表单提交POST请求

1、 settings.py中在MIDDLEWARE_CLASSES中把 ‘django.middleware.csrf.CsrfViewMiddleware’ 中间件加上，加上后即可全局使用csrf防护（默认已经添加）； 如果不加，也可以在视图函数上加 @csrf_protect 进行单view控制，不过这不是推荐的方法，因为这样可能会有遗漏。

注：该中间件必须在 SessionMiddleware 之后执行，因此在列表中 CsrfMiddleware 必须出现在 SessionMiddleware 之前 （因为响应中间件是自后向前执行的）。 同时，它也必须在响应被压缩或解压之前对响应结果进行处理，因此CsrfMiddleware必须在GZipMiddleware之后执行。

2、在html的表单中form标签后加上{% csrf_token %}， 如：

```
1
2
```

```
&lt;form action="" method="post"&gt;{% csrf_token %}
```

3、在视图函数中不想进行csrf保护的view可以加上@csrf_exempt

4、把django.core.context_processors.csrf上下文处理器加到配置文件的TEMPLATE_CONTEXT_PROCESSORS；或者手动生成csrftoken并加到template context，如：

```
1  from django.core.context_processors import csrf
2  from django.shortcuts import render_to_response
3
4  def my_view(request):
5  c = {}
6  c.update(csrf(request))
7  # ... view code here
8  return render_to_response("a_template.html", c)
9
```

ajax提交post请求

1. settings设置同上节；
2. 在视图中使用 render （而不要使用 render_to_response），如：

views.py

```python
from django.shortcuts import render
from django.http import HttpResponse

from .models import BioDrug

def select_drug(request):
    if request.method == 'POST':
        # print request.POST.items()
        drugs_id = request.POST.get('drugs')
        drugs_id_list = drugs_id.strip().split(' ')
        return HttpResponse(','.join(drugs_id_list))
    else:
        drugs = BioDrug.objects.filter(inputer__isnull=True)
        return render(request,'exam2014/index.html',{'drugs': drugs})
```

render

在html模板script标签中加（注：不能 .js文件中加以下代码）：

$.ajaxSetup({ data: {csrfmiddlewaretoken: '{{ csrf_token }}' }, });

示例：

```
<!DOCTYPE html>
<html>
<head>
<script src="/static/jquery.min.js"></script>
<script>
    $.ajaxSetup({
      data: {csrfmiddlewaretoken: '{{ csrf_token }}' },
    });

    function select_drug() {
        var drug=document.forms[0].drug;
        var drugs="";
        var i;
        for (i=0;i<drug.length;i++) {
        if (drug[i].checked)
          {
          drugs=drugs + drug[i].value + " ";
          }
        }

        $.post('{% url "exam2014" %}', {'drugs': drugs}, function(ret){
            $('#result').html(ret);
        })
    }
</script>
</head>
<body>
    <p>请选择两个药名：</p>
    <form>
    {% for drug in drugs %}
        <input type="checkbox" name="drug" value="{{ drug.id }}">{{ drug.name }}<br>
    {% endfor %}
    <br><input type="button" onclick="select_drug()" value="Submit"><br>
    </form>

    <div id="result"></div>
</body>
</html>
```

ajaxSetup

其它模板引擎

如与Jinja2结合的解决办法：

1、settings.py中在 MIDDLEWARE_CLASSES 中加上 ‘django.middleware.csrf.CsrfViewMiddleware’ 这个中间件

2、如果需要校验cookie中的csrf值，则在views.py中导入

```
from django.core.context_processors import csrf
```

上下文参数改为形如：

```
context = { 'args': args, 'condition': condition }
context.update(csrf(request))
return render(context, 'login.html')
```

同时在模板HTML文件中的对应的Form表单里加

```
1
2
```
```
&lt;input type="hidden" id="csrfmiddlewaretoken" name="csrfmiddlewaretoken" value="{{ csrf_token }}"&gt;
```

3、如果不需要校验cookie中的csrf值，则在views.py中导入

```
1
2
```
```
from django.views.decorators.csrf import csrf_exempt
```

对应的函数加 @csrf_exempt 装饰器

最后

俗话说：难者不会，会者不难。把它理解透了，发现也没那么难，以后看见CSRF我不怕不怕了~^o^

转载于:https://my.oschina.net/yuyang/blog/831445