# 再谈CSRF漏洞防御

作者: LANDGREY ● 创建时间 2018年6月19日 19:50 ● 更新时间 2018年6月20日 23:16
浏览: 1132 次. ● 标签: #渗透测试, #Web安全备忘录
您的IP地址: 140.207.23.83


再谈CSRF漏洞防御 正文

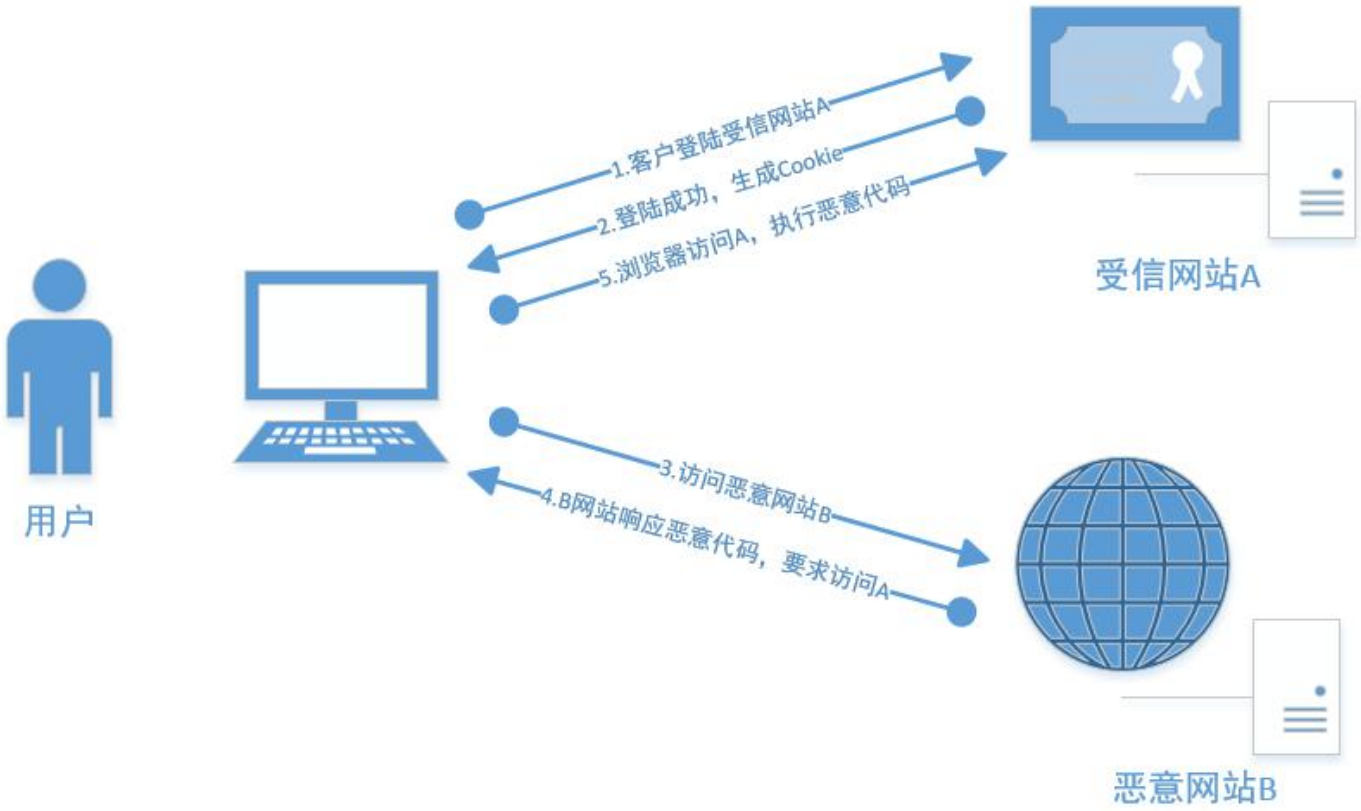
## 漏洞简述

> 用户认证成功后访问或点击了黑客控制的网站链接，浏览器根据黑客网页代码指示，使用用户的身份凭证(Cookie)自动请求服务器，冒充用户身份完成恶意操作。

## 根本原因

> HTTP协议是无状态的协议，所以引入了Cookie机制来辨识和跟踪用户，但Cookie最初被设计成允许在第三方网站发起的请求中携带，从而成为CSRF漏洞滋生的土壤。

# 利用流程




# 测试思路

下面的英文描述更清楚简洁一点

1. Perform the request without modifying the parameters, see what the result is
2. Remove the CSRF token completely
3. Modify one of the characters in the token (keep the length same)
4. Remove the value of the token (leave the parameter in place)
5. POST request cnvert to a GET request
6. Confirm whether they have the same response after modify the request

# 漏洞验证

a. 攻击者必须要知悉请求所需要的所有必须字段名和值的格式，以便构造漏洞验证页面；

b. 用户已打开浏览器，完成了目标系统的身份认证；

c. 诱使用户使用同一浏览器请求(打开)构造好的漏洞验证链接地址

**一个基本的验证步骤参考如下**：

1. 将以下 `index.html` 页面放置在某服务器上，如 `http://hacker.com`
2. 引导已登录 http://target.com 网站的受害者访问 http://hacker.com/index.html 页面
3. 浏览器会在受害者察觉不到的情况下，利用受害者当前在http://target.com的Cookie，自动用 `index.html` 中的代码指令，请求存在CSRF漏洞的地址 http://target.com/add-admin-page.php ，添加新管理员用户
4. 然后浏览器自动跳转到 http://target.com/normal-page.php 正常页面
5. 对于受害者来说"第3步"用时很短，并且操作完成后自动跳转到"第4步"页面，整个过程是完全察觉不到的

```
<html>
<head>
<script>
    function auto(){
        document.getElementById('LandGrey').submit();
        window.location.assign("http://target.com/normal-page.php")
    }
</script>
</head>

<body onload=auto()>
    <form action="http://target.com/add-admin-page.php" method="post" id="LandGrey">
        <input type='hidden' name="username" value="hacker">
        <input type='hidden' name="password" value="hackyou">
        <input type='hidden' name="submit" value="addUser">
    </form>
</body>
</html>
```

## 防御思想

> 提供攻击者不能伪造、借用，不在Cookie中的信息，并加以验证。

## 防御技术

01: 检查Referer

**HTTP Header中的Referer字段，记录了 HTTP 请求的来源地址链接。**

检查Referer值的域名是否和当前域名一致，如不一致，则拒绝该请求。

但存在 `Referer值为空` 的情况、 `Referer检查代码实现有误` 等原因，存在较多被绕过的情况，所以只能作为一个辅助预防CSRF漏洞的方法。

02: 检查Origin

**HTTP Header中的Origin字段，标明了跨域请求中发起跨域请求的源域名。**

但存在 `IE 11浏览器在跨域请求中不会添加Origin头` 、 `302跳转不携带Origin` 等原因，在发起的请求中存在Origin字段时，检查跨域请求的源地址实际上只能辅助预防CSRF漏洞。

03: 同步Token

即**服务端比对Token**，这是目前较成熟和推荐的一种防御CSRF漏洞的方案，使用Session存储CSRF防御token。

对于每一次敏感操作请求，当请求方法为GET时，生成的CSRF token会被放在URL参数值中一起提交；请求方法为POST时，生成的CSRF token被放在表单中一起提交。提交后，服务端校验表单中的token值和Session中的token值是否一致，不一致则为CSRF攻击。

但当请求方法为GET时，token可能会随着 Referer泄漏 到其它网站；网站 存在XSS漏洞时会被绕过 。

04: 加密Token

即**服务端解密Token**，与**服务端比对Token**不同，服务端解密token防御方式会根据用户ID、时间戳、只被使用一次的随机数值，用一个唯一的key加密生成token值，存储到**Cookie**中，并赋值到表单中。

用户请求时携带表单中的token值，服务端使用key解密token，解密成功后获得用户ID、时间戳，如果解密不成功或者当前用户ID不符或当前时间距时间戳超过一定阀值，那么判断是CSRF攻击。

这种防御方法的时间差阀值很难控制，过低影响用户体验，过高还是会造成CSRF攻击成功，所以不推荐使用。

05: 基于Cookie的Token值双校验

即**Double Submit Cookie**，**安全性基于攻击脚本无法跨域读取Cookie值。**如果服务端没有提供Session机制，可以在用户认证完成时，同时设置一个随机token存储在Cookie中。

当用户提交表单时，读取Cookie中的token，附加在表单中一起发送，由服务端验证Cookie中的token和附加的参数值是否相同，不同则说明是CSRF攻击。

由于要取用Cookie中的token值附加在表单中，所以使用此种防御方法时，需要读取的 cookie不能标识为httponly ； 当攻击者结合 XSS漏洞 、 子域名XSS 、 CRLF漏洞 、 畸形Cookie 、 中间人攻击 ，Cookie值可以被注入或固定时，此种防御方法就会被绕过。

06: 自定义请求头

**安全性基于SOP和CORS**，只有同源的JavaScript代码才能增加自定义请求头。

例如一个常用的表示Ajax异步请求的请求头为 **X-Requested-With: XMLHttpRequest** ，在保护REST服务Ajax请求时作为的CSRF防御方案，较为便捷。但对于修复CSRF漏洞来说，此种方法意味着可能要重写整个网站，是不切实际的。

07: 设置同站Cookie

Google为了从根源上解决第三方Cookie导致的CSRF问题，提出SameSite Cookie(禁用第三方Cookie)防御方案。

即网站响应头中用Set-Cookie为Cookie值新增SameSite 属性，如：

```
Set-Cookie: bar=xxxx; SameSite=Strict
```

表明bar Cookie值只作为第一方Cookie使用，第三方网站hacker.com向第一方网站target.com发送请求时，浏览器不会将该Cookie一起发往target.com网站，攻击者也就无法仿冒用户身份，就不存在CSRF漏洞了。

但是到目前为止，这个HTTP协议提案也不是每个浏览器都实现了，所以方案的**安全性基于浏览器对SameSite Cookie协议的实现。**

08: 强制用户交互

在重要操作步骤前，让用户进一步确认该操作请求是自己自愿发起的，如：

1. 生成图形验证码并让用户正确填写
2. 向用户发送短信验证码并让正确填写
3. 弹出窗口让用户点击确认
4. 让用户再次输入密码确认

虽然能够防御住CSRF漏洞，但是对用户体验十分不友好，只有特别重要的用户操作才会使用。



<