Uma **implementação contra ataques CSRF (Cross-Site Request Forgery)** em PHP é uma técnica de segurança usada para proteger aplicações web contra um tipo específico de ataque onde um usuário autenticado é induzido a executar ações não intencionais em uma aplicação na qual ele está logado.

**🔐 O que é um ataque CSRF?**

**CSRF (Cross-Site Request Forgery)**, ou "falsificação de solicitação entre sites", é um ataque em que um site malicioso força o navegador de um usuário autenticado a enviar uma requisição não autorizada a outro site (geralmente o sistema legítimo), explorando a **sessão ativa do usuário**.

**🔎 Exemplo de ataque:**

1. O usuário está logado em meubanco.com.
2. Ele visita um site malicioso.
3. Esse site executa (via JavaScript, imagem, formulário oculto, etc.) uma requisição como:

<img src="https://meubanco.com/transferir.php?para=123&valor=1000" />

1. Como o navegador já tem um **cookie de sessão** válido, o site do banco processa a requisição — **sem o consentimento do usuário**.

**🛡️ Como uma implementação contra CSRF protege?**

A **implementação CSRF Token** impede isso por exigir um **token secreto e único**, enviado junto com o formulário ou requisição AJAX. Esse token:

* **É armazenado na sessão do usuário**
* **É verificado pelo servidor ao receber a requisição**

Se o token não for enviado ou for inválido, a requisição é rejeitada.

**✅ Etapas da Implementação CSRF em PHP**

**1. Gerar o token CSRF (em form.php, por exemplo):**

session\_start();

if (empty($\_SESSION['csrf\_token'])) {

$\_SESSION['csrf\_token'] = bin2hex(random\_bytes(32)); // token seguro

}

$csrf\_token = $\_SESSION['csrf\_token'];

**2. Incluir o token no formulário:**

<form method="post" action="processa.php">

<input type="hidden" name="csrf\_token" value="<?= htmlspecialchars($csrf\_token) ?>">

<!-- outros campos do formulário -->

<button type="submit">Enviar</button>

</form>

**3. Verificar o token na requisição (em processa.php):**

session\_start();

if (!isset($\_POST['csrf\_token'], $\_SESSION['csrf\_token']) ||

!hash\_equals($\_SESSION['csrf\_token'], $\_POST['csrf\_token'])) {

die('Erro: Requisição inválida ou token CSRF incorreto.');

}

**// Aqui segue o processamento seguro**

**🧠 Explicações técnicas importantes:**

* bin2hex(random\_bytes(32)): gera um token forte, de 64 caracteres hexadecimais.
* hash\_equals(): faz a comparação de forma segura contra ataques de tempo de execução.
* O token pode ser renovado após cada submissão, dependendo da política de segurança.

**🎯 Benefícios da proteção CSRF:**

* **Impedir ações maliciosas** de serem executadas por scripts externos.
* **Reforçar a integridade** de ações sensíveis (envio de dados, transferências, exclusões).
* **Complementar outras medidas de segurança** como autenticação e controle de sessão.

**🛑 Importante:**

* Essa proteção **não substitui** outras práticas de segurança (como validação de entrada ou proteção contra XSS).
* Em requisições **AJAX**, o token pode ser incluído em um **cabeçalho personalizado**.
* Ferramentas como Laravel, Symfony, CodeIgniter já implementam CSRF automaticamente.

**Implementação de Segurança na Aplicação PHP**

**Resumo das Implementações**

Este projeto recebeu melhorias importantes para aumentar a segurança da aplicação, especialmente focadas em proteção contra-ataques CSRF e controle de acesso para usuários autenticados.

**Funcionalidades Implementadas**

**1. Proteção contra-ataques CSRF**

* Criação do helper Funcoes/csrf\_helper.php para geração e validação de tokens CSRF.
* Inclusão do token CSRF em todos os formulários importantes, como:
  + Formulário de login (index.php)
  + Formulário de cadastro/edição de usuários (View/usuario/usuario.php)
* Validação do token CSRF no backend para todas as requisições que alteram dados:
  + Funcoes/process\_login.php (login)
  + Funcoes/usuario.php (cadastro/edição de usuários)

**2. Restrição de Acesso para Usuários Não Autenticados**

* Criação do helper Funcoes/auth\_helper.php para centralizar a verificação de autenticação.
* Inclusão da verificação de login em arquivos sensíveis para garantir que apenas usuários autenticados possam acessá-los:
  + Funcoes/usuario.php
  + View/usuario/listarUsuarios\_partial.php
  + View/usuario/usuario.php
* Redirecionamento para a página de login (index.php) caso o usuário não esteja autenticado.

**3. Testes Realizados**

* Testes completos cobrindo:
  + Fluxo de login com token CSRF válido e inválido.
  + Acesso restrito a páginas protegidas sem autenticação.
  + Fluxo de cadastro/edição de usuários com validação CSRF.
  + Cenários de erro e sucesso para ambos os formulários.

**Como Utilizar**

* Ao iniciar a aplicação, o usuário deve realizar login para acessar as páginas protegidas.
* Todos os formulários que alteram dados possuem proteção CSRF automática.
* Caso o token CSRF esteja ausente ou inválido, a requisição será rejeitada.
* Usuários não autenticados são redirecionados para a página de login ao tentar acessar páginas protegidas.

**Arquivos Principais Modificados/Adicionados**

* Funcoes/csrf\_helper.php - Helper para geração e validação de tokens CSRF.
* Funcoes/auth\_helper.php - Helper para verificação de autenticação.
* index.php - Formulário de login com token CSRF.
* Funcoes/process\_login.php - Validação do token CSRF no login.
* Funcoes/usuario.php - Validação do token CSRF e verificação de autenticação para cadastro/edição.
* View/usuario/usuario.php - Inclusão do token CSRF no formulário de cadastro/edição.
* View/usuario/listarUsuarios\_partial.php - Restrição de acesso para usuários autenticados.

**Considerações Finais**

Estas melhorias aumentam significativamente a segurança da aplicação, prevenindo ataques CSRF e garantindo que apenas usuários autenticados possam acessar áreas restritas.

Para futuras melhorias, recomenda-se:

* Implementar logout seguro.
* Adicionar controle de permissões por tipo de usuário.
* Monitorar sessões e implementar timeout de sessão.