Curso PHP

Do XLSX ao CMS (aula 14)

Cross-Site Request Forgery — CSRF/XSRF

CSRF/XSRF

A falsificação **Cross-Site Request Forgery (CSRF)** é um tipo de ataque que ocorre quando um site, e-mail, blog, mensagem instantânea ou programa mal-intencionado faz com que o navegador da web de um usuário realize uma ação indesejada em um site confiável quando o usuário é autenticado.

Um ataque CSRF funciona porque as solicitações do navegador incluem automaticamente quaisquer credenciais associadas ao site, como o cookie de sessão do usuário, endereço IP, etc.

Portanto, se o usuário for autenticado no site, o site não poderá distinguir entre o pedido forjado ou legítimo enviado pela vítima. **Precisamos de um token/identificador que não seja acessível ao atacante** e não seja enviado (como cookies) com solicitações forjadas que o invasor inicia.

(OWASP)

Synchronizer Token Pattern

Qualquer operação de mudança de estado requer um token aleatório seguro (por exemplo, token CSRF) para evitar ataques CSRF.

Um token CSRF deve ser exclusivo por sessão de usuário, com “grande valor aleatório” e também gerado por um gerador de números aleatórios criptograficamente seguro.

O token CSRF é adicionado como um campo oculto para formulários, cabeçalhos/parâmetros para chamadas AJAX e dentro do URL se a operação de alteração do estado ocorrer através de um GET.

O servidor rejeita a ação solicitada se o token CSRF falhar na validação.

(OWASP)

```
<head>
    <meta charset="utf-8">
    <meta name="color-scheme" content="light dark">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="csrf-token" content="{{ CSRF_TOKEN }}">
    <link rel="icon" href="/favicon.ico" type="image/x-icon">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@picocss/pico@2/css/pico.min.css">
    <title>Plano de aulas</title>
</head>
```

Tag adicionada ao cabeçalho do documento

```
<form action="{{ url('/disciplinas') }}" method="post">
    {% include "disciplinas/fieldset.twig" %}
    <input type="hidden" name="_csrf_token" value="{{ CSRF_TOKEN }}" />
    <input type="submit" value="&#128190; Cadastrar">
</form>
```

Campo oculto adicionado ao formulário

```php
if (in_array($this->request->getMethod(), [
    BaseRequest::METHOD_DELETE,
    BaseRequest::METHOD_POST,
    BaseRequest::METHOD_PUT,
]) && ! $this->request->verifyCsrfToken()) {
    return response('erro_403', status: Response::HTTP_FORBIDDEN);
}
```

Verificando se o token é válido para os métdos HTTP especificados

Mitos do CSRF

CORS é um cabeçalho projetado para relaxar a política de Same-Origin quando é necessária a comunicação de origem cruzada entre os locais. Não é projetado e nem impede ataques CSRF.

Usar HTTPS não tem nada a ver com a proteção contra ataques CSRF. Os recursos que estão sob HTTPS ainda são vulneráveis ao CSRF se as mitigações CSRF.

(OWASP)

Testes

composer require "ext-dom:*"

Requisita extensão como dependência

```php
public static function getCsrfToken(string $path): ?string
{
    $response = self::get($path);
    $responseContent = $response->getBody()->getContents();
    $dom = new DOMDocument;

    // Suprime erros de HTML mal formado
    libxml_use_internal_errors(true);

    // Carrega o conteúdo HTML da resposta para o DOMDocument
    $dom->loadHTML($responseContent);

    // Limpa os erros de parsing
    libxml_clear_errors();

    foreach ($dom->getElementsByTagName('input') as $input) {
        if ($input->getAttribute('name') === '_csrf_token') {
            return $input->getAttribute('value');
        }
    }

    return null;
}
```

Método para extrair o token CSRF da página requisitada

Testando

Agora que temos a verificação no backend, precisamos testar nossa aplicação.

Para isso, a classe **Http** implementa o comportamento *IsSingleton*, para que mantenha o token CSRF durante a testagem.

```
test('cadastrar atividade', function () {
    $nome = faker()->word();
    $cor = faker()->hexColor();
    $_csrf_token = Http::getCsrfToken('/atividades');
    $options = [
        'form_params' => compact('nome', 'cor', '_csrf_token'),
    ];

    $response = Http::post('/atividades', $options);
    $responseContent = $response->getBody()->getContents();

    expect($responseContent)->toMatch('/total: 1 atividades/i');
    expect($responseContent)->toMatch("/{$nome}/i");
});
```

Estabelecendo o público e o restrito. Autenticando usuários. Definindo papéis. Nos próximos episódios.