Desenvolvimento Web II

Cookies, Sessões, Login, Ataques CSRF

Prof. Dr. Daniel A. Furtado - FACOM/UFU

Introdução

- Em algumas situações a aplicação web pode precisar manter o **contexto de acesso** entre várias requisições do usuário;
- Por exemplo, em um sistema web de acesso restrito, após efetuar o login do usuário, deve haver um mecanismo que autorize o acesso às páginas restritas **enquanto** o usuário estiver logado;
  - Portanto, depois do login, o sistema deve ser capaz de dar **continuidade** ao acesso do usuário, ou seja, deve ser capaz de manter sua sessão de acesso;
- Igualmente importante, o sistema não deve permitir o acesso às páginas restritas sem que o usuário tenha feito login;
- As linguagens de programação server-side geralmente possuem mecanismos para criação e manutenção da sessão do usuário. Em PHP, esse controle pode ser feito utilizando cookies e variáveis de sessão.

Tipos de Requisições HTTP no Navegador

- Requisição de Navegação de Nível Superior (Top-Level)
  - Ocorre quando a URL na barra de endereços do navegador é alterada;
  - O conteúdo retornado pela requisição substitui completamente o conteúdo da página atual (a página inteira da aba é recarregada);
  - **Exemplos:** digitar uma nova URL, clicar em um link externo ou em um link `<a>` ou submeter um formulário que redireciona o usuário.
  - O navegador assume que é uma ação intencional do usuário (ele quis sair de um site e ir para outro). Por isso, as regras de cookies são mais permissivas para manter o usuário logado;
  - Não são restringidas pela Política de Mesma Origem (SOP) nem pelo mecanismo de CORS.

Tipos de Requisições HTTP no Navegador

- Requisição de Sub-Recurso
  - Busca partes de uma página já carregada no navegador;
  - O contexto da página principal não muda; apenas dados ou elementos específicos são buscados.
  - **Exemplos**: requisições decorrentes de imagens (<img src='url'>), iframes (<iframe src='url'>) e scripts com a API Fetch ou o XMLHttpRequest.
  - Podem ser disparadas silenciosamente por scripts maliciosos sem que o usuário perceba. Por isso, os navegadores modernos são mais rígidos quanto ao envio de cookies.

Cookies

- Um **cookie** é um pequeno bloco de dados do tipo **chave-valor** que o navegador armazena para um domínio específico;
- O cookie pode ser criado tanto pelo servidor (via cabeçalhos de resposta HTTP) quanto pelo cliente (via JavaScript);
- Uma vez armazenado, o navegador reenvia automaticamente o cookie ao servidor nas requisições subsequentes, até que sua validade expire;
- Dependendo das configurações, os cookies podem ser acessados pelo website durante os próximos minutos, dias ou até meses:
  - Cookies de sessão são normalmente apagados ao fechar o navegador;
  - Cookies persistentes têm data de validade definida e permanecem no computador até expirarem ou serem apagados manualmente.

Cookies podem ser criados com vários atributos

- Expires/Max-Age: este atributo define o tempo de permanência do cookie. Se omitido, o cookie será tratado como de sessão;
- Path: define um caminho no servidor. O navegador só enviará o cookie em requisições utilizando tal caminho. O padrão é '/';
- HttpOnly: indica que o cookie não pode ser acessado por código JavaScript, prevenindo roubo de cookie por ataques XSS. Cookies com o atributo HttpOnly podem ser acessados apenas pelo próprio navegador.
- Secure: com esse atributo, o cookie só é enviado em conexões seguras (HTTPS);

Cookies podem ser criados com vários atributos

- SameSite: atributo que controla se o cookie deve ser enviado em requisições iniciadas de sites externos (cross-site). Há três valores possíveis:
  - SameSite=Strict: o cookie só é enviado em requisições que partem do próprio site;
  - SameSite=Lax (padrão): o cookie é enviado em requisições de navegação de nível superior com métodos seguros* (como o GET), mas não em sub-requisições (<img>, <iframes>, <fetch> etc.) originadas de outros sites;
  - SameSite=None: o cookie é enviado até mesmo em requisições de sub-recursos como aquelas provenientes de elementos HTML (<img>, <iframe>, <video>, etc.) originadas de outros sites, desde que o atributo Secure também esteja presente. São enviados também em requisições fetch com a opção credentials: 'include' ou com o XMLHttpRequest com a opção xhr.withCredentials=true, inclusive utilizando métodos HTTP não seguros (como o POST).

*Métodos seguros são os métodos que não têm a intenção de alterar o estado da aplicação no servidor, como GET, HEAD e OPTIONS.

Criação de Cookies pelo Servidor

- Para criar um cookie no navegador do usuário, o servidor só precisa enviar uma resposta HTTP ao navegador com o cabeçalho set-cookie detalhando o nome, o valor e os atributos do cookie a ser criado;
- Por exemplo, uma resposta HTTP contendo o cabeçalho:

  Set-Cookie: PHPSESSID=123456; path=/; Secure; HttpOnly; SameSite=Strict

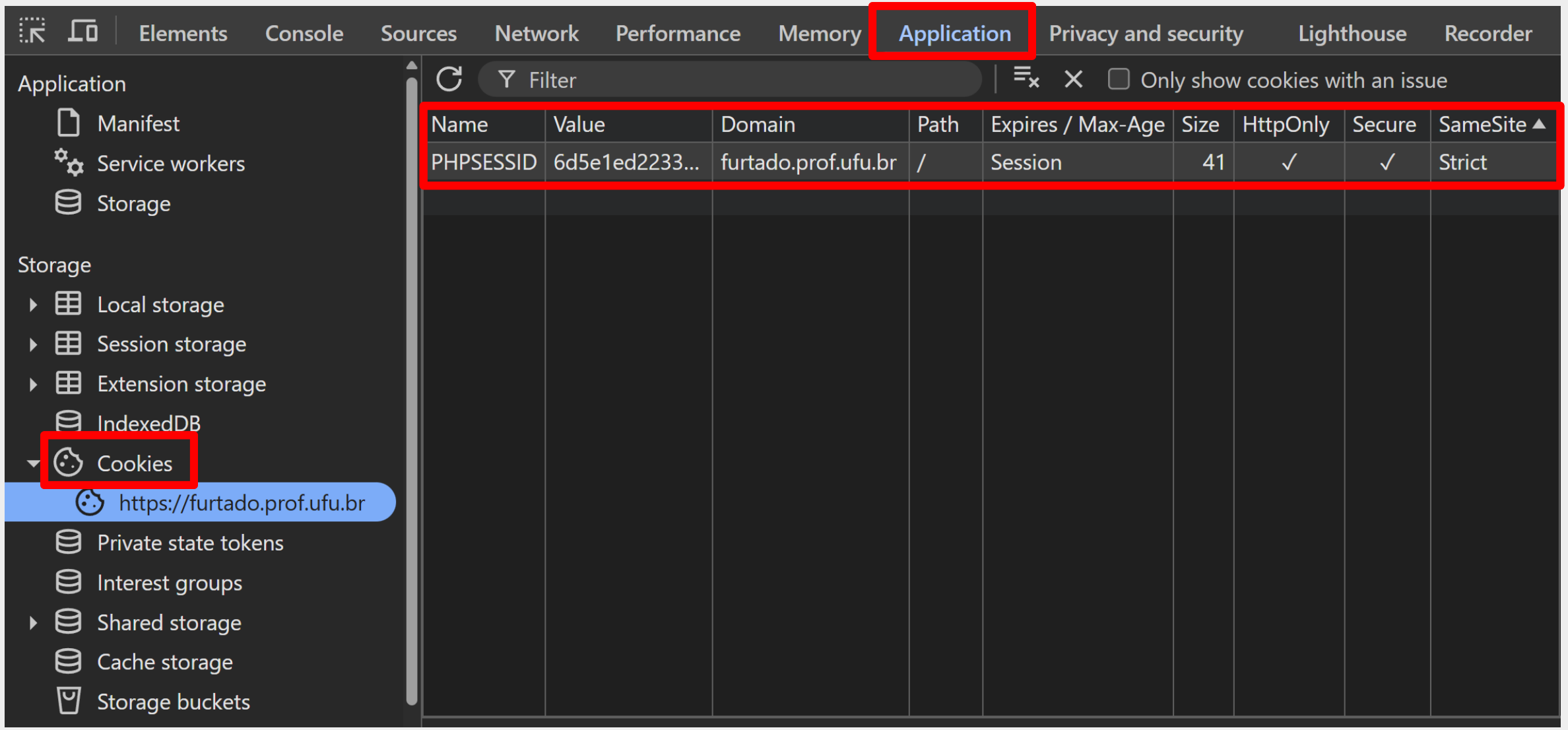
  diz ao navegador para criar um cookie de nome PHPSESSID com o valor 123456, que só deve ser enviado em HTTPS, que não pode ser acessado via JavaScript, e que só deve ser enviado em requisições originadas do próprio site.

Envio do Cookie pelo Navegador ao Servidor

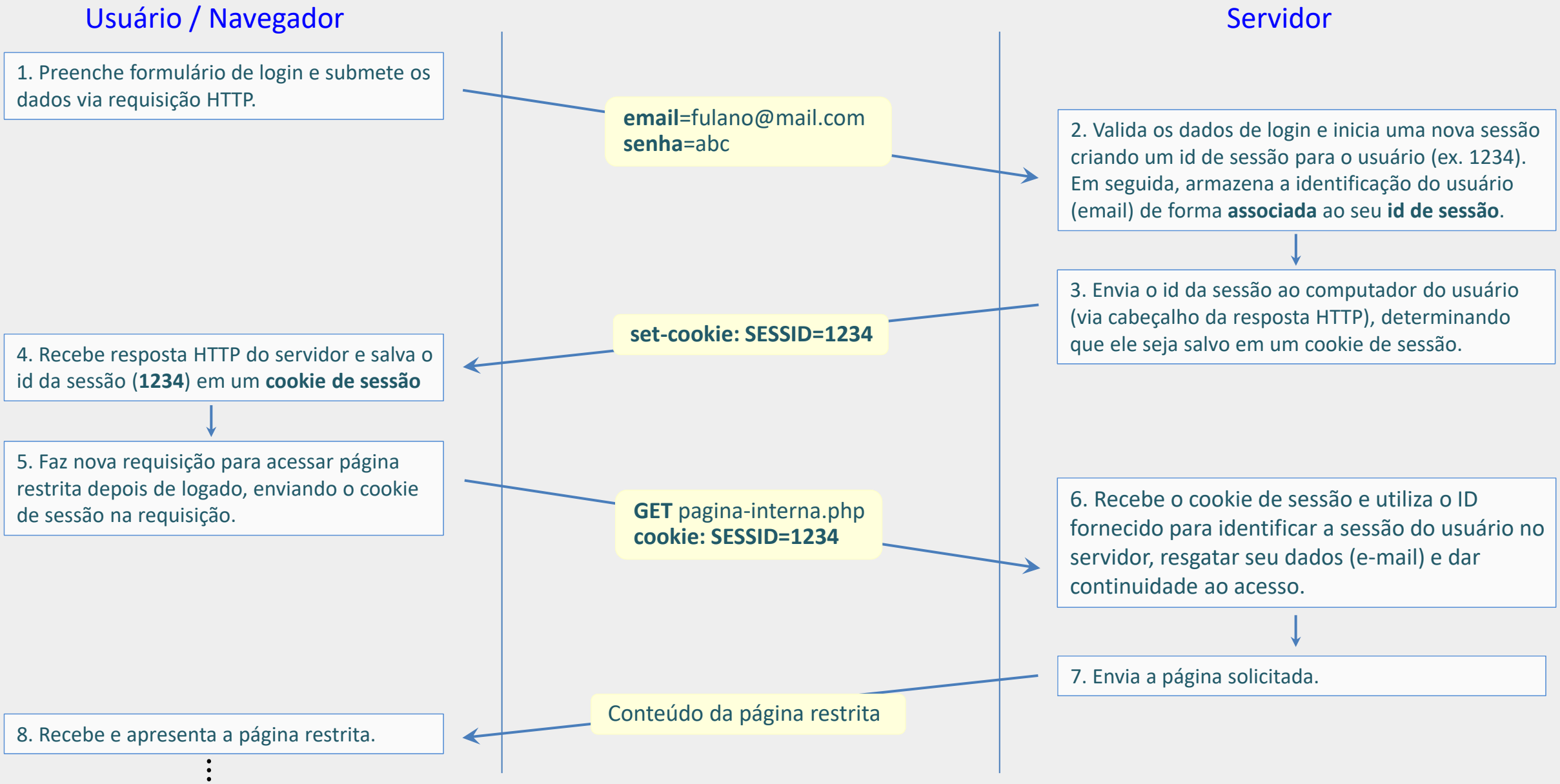
- Quando novas requisições são enviadas ao website, o cookie é enviado por meio do cabeçalho cookie da requisição HTTP;
- Exemplo:

```
GET /saat/professor/index.html HTTP/1.1
Accept-Language: pt-BR
Cookie: PHPSESSID=cd43560b219d6ba1a
Host: furtado.prof.ufu.br
```

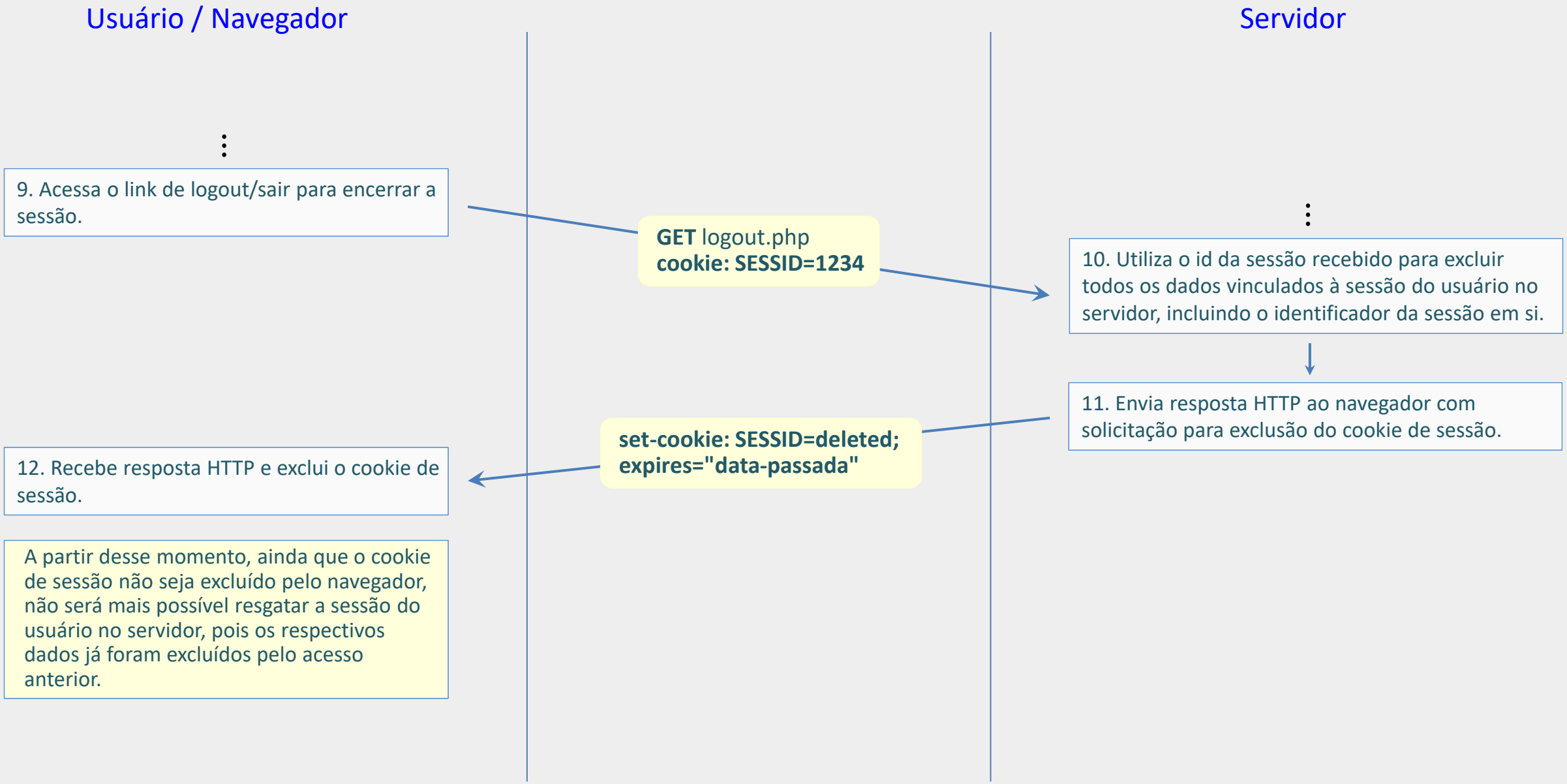
Verificando os Cookies do Site no Chrome DevTools (F12)



Mecanismo de Login com Cookie de Sessão



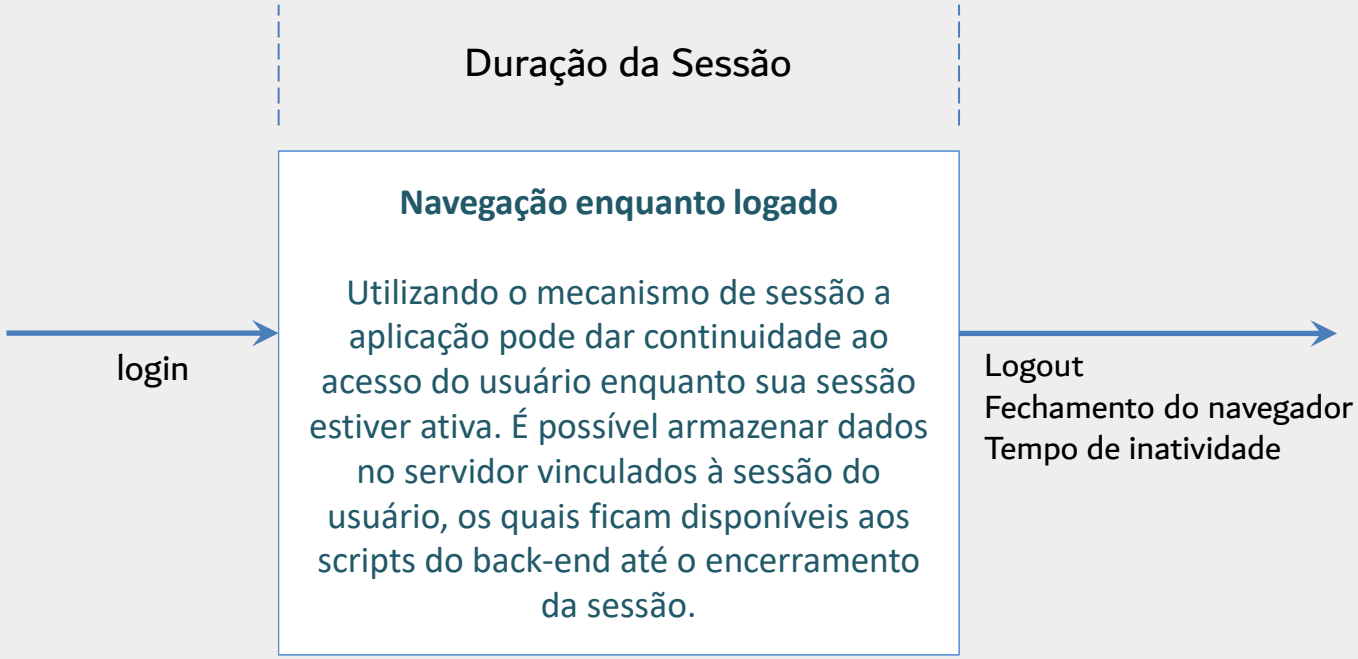
Mecanismo de Login com Cookie de Sessão



Mecanismo de Login com Cookie de Sessão - Resumo

- No momento da criação de uma nova sessão (login), o servidor gera um código identificador (ID) para a sessão e o armazena em um arquivo;
- Em seguida, o código da sessão é enviado ao navegador e armazenado em um cookie de sessão no computador do usuário;
- Quando o usuário fizer novas requisições à aplicação (enquanto logado), o cookie de sessão será enviado de volta ao servidor;
- O servidor, portanto, terá condições de dar continuidade ao acesso do usuário, uma vez que receberá, nas novas requisições, o ID da sessão iniciada no momento do login;
- Os dados da sessão são armazenados no servidor e ficam vinculados ao ID da sessão do usuário.

Manutenção de Sessão de Usuário Logado



Criação e Manipulação de Sessão no PHP

- Para criar uma nova sessão, no momento do login, basta chamar a função session_start() antes que o script produza qualquer saída.
- Para armazenar dados **no servidor** vinculados à sessão aberta, basta utilizar $_SESSION['nome-da-variavel'] = 'valor'.
- Para finalizar manualmente a sessão e remover os dados vinculados (no logout), deve-se chamar a função session_destroy().
- Para acessar um dado de sessão criado anteriormente, deve-se abrir/retomar a sessão corrente com session_start() e acessar a variável com $_SESSION['nome-da-variavel'].

Repare que a função session_start **cria** uma nova sessão ou **retoma** a sessão atual com base no identificador de sessão passado por meio da requisição HTTP ou pelo cookie.

Exemplo Simples

Script login.php

```php
$email = $_POST['email'] ?? '';
$senha = $_POST['senha'] ?? '';

$pdo = mysqlConnect();
if (checkUserCredentials($pdo, $email, $senha)) {
  session_set_cookie_params([
    'secure' => true,
    'httponly' => true,
    'samesite' => 'Lax'
  ]);

  session_start();
  $_SESSION['loggedIn'] = true;
  $_SESSION['user'] = $email;
  header('Location: home.php');
} else
  header('Location: index.html');
```

Script home.php

```php
<?php
session_start();
if (!isset($_SESSION['loggedIn'])) {
  header("Location: index.html");
  exit();
}
?>
<!DOCTYPE html>
<html>
<head><title>Home</title></head>
<body>
  Olá, <?php echo $_SESSION['user'] ?>!
</body>
</html>
```

Tempo de Permanência do Cookie de Sessão

- Por padrão, o cookie de sessão enviado pelo PHP permanecerá no computador do usuário até que a sessão seja finalizada no navegador, ou seja, até o navegador ser fechado*;
- A exclusão do cookie de sessão causará a perda de sessão do usuário, uma vez que o ID da sessão não será mais enviado ao servidor para resgate das respectivas variáveis;
- Porém é possível alterar o tempo de expiração do cookie de sessão utilizando a função session_set_cookie_params do PHP:
  - Exemplo: session_set_cookie_params(3600) – o cookie de sessão irá expirar em 3600 segundos (1 hora). Neste caso, mesmo que o navegador seja fechado, durante esse período o cookie de sessão continuará presente.

*Porém os navegadores geralmente disponibilizam uma funcionalidade que permite ao usuário salvar a navegação de todas as abas abertas para restauração posterior, mesmo que o navegador seja fechado. Nesses casos, os cookies de sessão também serão restaurados, como se o navegador não tivesse sido fechado.

Tempo de Permanência dos Dados da Sessão no Servidor

- No servidor, os dados da sessão permanecerão armazenados em arquivo até que uma das situações a seguir ocorra:
  1. O tempo de inatividade da sessão for maior que o parâmetro ***session.gc_maxlifetime***. Neste caso, o *garbage collector* do PHP tratará o conteúdo das variáveis de sessão como lixo, e poderá excluí-las;
  2. O desenvolvedor encerra explicitamente a sessão chamando os métodos **session_unset**() e **session_destroy**();

O parâmetro session.gc_maxlifetime do PHP tem o valor padrão de 1440 segundos (24 minutos). Portanto, a sessão do usuário poderá ser perdida caso seu tempo de inatividade seja superior a esse valor. Entretanto, esse parâmetro pode ser ajustado no arquivo de configuração do PHP (php.ini) ou pela função ini_set (Exemplo: ini_set('session.gc_maxlifetime', 3600), para que o servidor mantenha os dados da sessão por pelo menos 1 hora)

Ataques CSRF - Introdução

- CSRF (Cross-Site Request Forgery - Falsificação de Requisição Entre Sites) é um tipo de ataque onde um invasor engana o navegador da vítima para que ele execute ações indesejadas em um site no qual a vítima já está autenticada;
- O ataque se baseia no fato de que o navegador envia automaticamente os cookies de sessão nas requisições feitas para o próprio domínio do site, mesmo que essa requisição tenha sido iniciada **por outro site;**
- A vítima é levada a realizar uma ação sem seu consentimento explícito, como alteração de senha e dados, compra de produtos etc.

Como geralmente ocorre um ataque CSRF

1. O usuário faz login no site vulnerável (loja.com);
2. O navegador do usuário armazena o cookie de sessão (identificando-o como autenticado);
3. O usuário abre outra aba no navegador e visita um site malicioso (seja por um link do e-mail ou de qualquer outra forma);
4. O site malicioso contém um código HTML oculto que força o navegador do usuário a enviar uma requisição para loja.com:

```
<body onload="document.forms[0].submit()">
<form action="https://loja.com/altera-senha.php">
   <input type="text" name="nova-senha" value="123456">
</form></body>
```

5. O navegador do usuário, ao enviar a requisição para loja.com, automaticamente anexa o cookie de sessão válido;
6. O website da loja recebe a requisição com o cookie do usuário e a executa, achando que o usuário realmente a solicitou.

Prevenção de Ataques CSRF com SameSite=Strict

- Uma forma simples de prevenir esse tipo de ataque é criando o cookie de sessão com o atributo SameSite=Strict;
- Dessa forma, o navegador nunca irá enviar o cookie de sessão quando a requisição for originada de outro website, como no caso da página visitada pelo link do e-mail;
- Entretanto, essa solução pode ter o efeito colateral de prejudicar a experiência de navegação do usuário, uma vez que um link externo (não malicioso) não poderá levar o usuário até o site mantendo-o logado.

Prevenção de Ataques CSRF com Token

- Outra forma de prevenir esse tipo de ataque, mantendo SameSite=Lax, é utilizando tokens CSRF;
- Um token CSRF é um valor **aleatório, único e imprevisível** gerado pelo servidor para cada sessão de usuário;
- O token deve ser incluído como um campo oculto nos formulários HTML e enviado como parâmetro nas requisições Ajax que alteram dados no servidor.

Prevenção de Ataques CSRF com Token

1. O servidor gera o token CSRF e o armazena na sessão do usuário;
2. O token é inserido em um campo oculto (hidden) do formulário HTML ou passado como parâmetro na requisição Ajax;
3. Quando o formulário é submetido (ou a requisição Ajax é enviada), o servidor compara o token recebido com o token armazenado na sessão;
4. Se os tokens não coincidirem, a requisição é rejeitada como um possível ataque CSRF.

Esse mecanismo previne ataques CSRF porque o atacante não tem como saber o valor do token gerado pelo servidor, pois ele não está acessível externamente e é exclusivo para cada sessão.

Exemplo de Criação do Token CSRF ao Efetuar Login

```php
if ($credenciaisValidas) {
    session_set_cookie_params([
        'secure' => true,
        'httponly' => true,
        'samesite' => 'Lax'
    ]);
    session_start();
    $_SESSION['loggedIn'] = true;
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}
```

Exemplo de Envio do Token na Requisição

Html

```html
<head>
  <meta name="csrf-token" content="<?php echo $_SESSION['csrf_token']; ?>">
</head>
```

Envio com Fetch

```javascript
const csrfToken = document.querySelector('meta[name="csrf-token"]').getAttribute('content');
fetch('altera-senha.php', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'X-CSRF-TOKEN': csrfToken
  },
  body: JSON.stringify({ novaSenha: '123456' })
});
```

Envio com <form>

```html
<form action="altera-senha.php" method="post">
  <input type="hidden" name="csrf_token" value="<?php echo $_SESSION['csrf_token'] ?>">
  <input type="password" name="novaSenha">
  <button>Alterar</button>
</form>
```

Prevenção de Ataques CSRF com Token e Cookie

Caso o front-end seja totalmente estático (SPA, etc.), o servidor pode enviar o Token CSRF em um segundo cookie, sem o atributo HttpOnly:

1. O PHP gera o token e o salva na sessão e em um cookie chamado XSRF-Token;
2. O JavaScript lê esse cookie utilizando document.cookie;
3. O JavaScript envia o valor de volta em um cabeçalho da requisição Ajax;
4. O servidor compara o valor do cabeçalho com aquele salvo na sessão.

Essa técnica protege contra CSRF porque, pela Política de Mesma Origem (SOP/CORS), um site atacante pode enviar o cookie automaticamente, mas não pode lê-lo com JavaScript para coloca-lo no cabeçalho da requisição (desde que não haja vulnerabilidade para ataque XSS). Só o script legítimo da mesma origem pode ler esse cookie.

Códigos de Exemplo

https://furtado.prof.ufu.br/site/teaching/DW2/Exemplos-Sessao-CSRF.zip

Referências

- https://www.php.net/docs.php
- NIXON, R. ***Learning PHP, MySQL & JavaScript:*** With jQuery, CSS & HTML5. 5. ed. O'Reilly Media, 2018