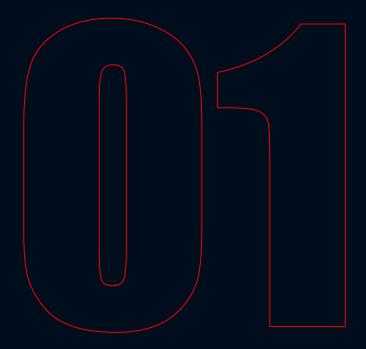
IDENTIDADE SECRETA

Clark Kent e Superman têm identidades distintas: sua sessão também deve ter!



Luiz Vitorio Casagrande



O QUE SÃO FALKAS DE GERENGIAMENTO DE SESSÃO P

O QUE SÃO FALHAS DE GERENCIAMENTO DE SESSÃO?



Imagine que você entra em um sistema, faz login com seu usuário e senha e começa a navegar. Para que a aplicação "lembre" que você está logado a cada nova página, ela cria uma sessão. Essa sessão funciona como uma chave temporária de acesso, que permite a você permanecer autenticado sem precisar digitar sua senha a todo momento. Gerenciar sessões de forma segura é essencial para garantir que usuários legítimos mantenham o controle de suas atividades dentro da aplicação, enquanto invasores são impedidos de se aproveitar de sessões já iniciadas. Sessões mal gerenciadas abrem portas para roubo de identidade, acesso indevido a dados sensíveis e outros riscos críticos.

Impactos das Falhas de Sessão:

- Roubo de identidade (impersonation): o atacante usa a sessão de outra pessoa.
- Execução de ações críticas em nome da vítima (ex: fazer compras, transferências, etc.).
- Comprometimento de todo o sistema, se a sessão pertencer a um administrador.

Causas comuns:

- Sessões que não expiram.
- ID de sessão previsível ou fixado.
- Cookies de sessão mal configurados.
- Exposição em URL ou locais inseguros.
- Reutilização de sessões entre usuários.



CONTROLE DE EXPIRAÇÃO DE SESSÃO

CONTROLE DE EXPIRAÇÃO DE SESSÃO



Quando uma aplicação não destrói adequadamente a sessão após o logout ou após um período de inatividade, ela permite que a mesma sessão continue ativa. Isso pode levar a cenários onde um invasor ou outro usuário acesse a aplicação utilizando uma sessão legítima que permaneceu armazenada no navegador.

Exemplos práticos

Inseguro — PHP

```
php

session_start();

// Nenhum controle de tempo de sessão
$_SESSION['usuario'] = 'joao'; // Sessão ativa indefinidamente
echo "Bem-vindo, {$_SESSION['usuario']}!";
```

Seguro — PHP

```
php

session_start();
// Define um tempo limite personalizado para a sessão (20 minutos)
$_SESSION['limite_sessao'] = 1200;
// Verifica se já existe um carimbo de tempo de acesso anterior
if (isset($_SESSION['ultimo_acesso'])) {
    $inatividade = time() - $_SESSION['ultimo_acesso'];
    if ($inatividade > $_SESSION['limite_sessao']) {
        session_unset();
        session_destroy();
        echo "Sessão expirada. Por favor, faça login novamente.";
        exit;
    }
}
// Atualiza o carimbo de tempo da sessão ativa
$_SESSION['ultimo_acesso'] = time();
$_SESSION['usuario'] = 'joao';
echo "Sessão ativa, usuário: {$_SESSION['usuario']}";
```

Inseguro — Java

```
Java

// LoginServlet.java
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
   HttpSession session = req.getSession();
   session.setAttribute("user", "admin");
   resp.getWriter().write("Usuário autenticado.");
}
```

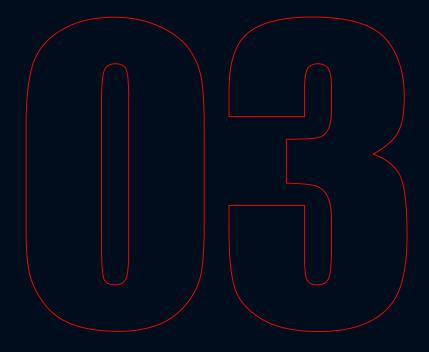
- A sessão é criada sem configurar um tempo de expiração.
- Um invasor que obtenha o JSESSIONID durante esse período pode reutilizá-lo.



Seguro — Java

```
// LoginServlet.java
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
   HttpSession session = req.getSession();
   session.setMaxInactiveInterval(900); // 15 minutos de inatividade
   session.setAttribute("user", "admin");
   resp.getWriter().write("Login seguro com expiração configurada.");
}
```

- Define um limite de inatividade de 15 minutos usando setMaxInactiveInterval.
- Reduz o tempo de exposição a sequestros de sessão em caso de roubo do ID de sessão.



REGERERAÇÃO DO ID DE SESSÃO

REGENERAÇÃO DO ID DE SESSÃO

Quando o ID da sessão (session_id) não é regenerado após o login, um invasor que saiba ou consiga roubar o ID da sessão anterior (por exemplo, antes do login) pode se passar pelo usuário após o login. Cenário de Ataque (Session Fixation):

- 1. O invasor cria uma sessão anônima no site e compartilha o link com a vítima.
- 2. A vítima clica no link e faz login.
- 3. Como o session_id não foi alterado, o atacante pode reutilizar esse mesmo ID e se passar pela vítima.

Exemplos práticos

Inseguro — PHP

```
php

session_start();
// Nenhum controle de tempo de sessão
$_SESSION['usuario'] = 'joao'; // Sessão ativa indefinidamente
echo "Bem-vindo, {$_SESSION['usuario']}!";
```

Seguro — PHP

```
php

session_start();
// Define um tempo limite personalizado para a sessão (20 minutos)
$_SESSION['limite_sessao'] = 1200;
// Verifica se já existe um carimbo de tempo de acesso anterior
if (isset($_SESSION['ultimo_acesso'])) {
    $inatividade = time() - $_SESSION['ultimo_acesso'];

    if ($inatividade > $_SESSION['limite_sessao']) {
        session_unset();
        session_destroy();
        echo "Sessão expirada. Por favor, faça login novamente.";
        exit;
    }
}
// Atualiza o carimbo de tempo da sessão ativa
$_SESSION['ultimo_acesso'] = time();
$_SESSION['ultimo_acesso'] = 'joao';
echo "Sessão ativa, usuário: {$_SESSION['usuario']}";
```

Inseguro — Java

```
Java

// LoginServlet.java
HttpSession session = req.getSession(); // Reaproveita ID existente
session.setAttribute("user", "admin");
```

- Reutiliza a mesma sessão anterior ao login.
- Permite ataque de Session Fixation, onde o invasor força a vítima a usar um ID conhecido.



Seguro — Java

```
Java

// LoginServlet.java
HttpSession oldSession = req.getSession(false);
if (oldSession ≠ null) oldSession.invalidate();

HttpSession newSession = req.getSession(true); // Nova sessão com novo ID newSession.setAttribute("user", "admin");
```

- Invalida a sessão anterior e gera uma nova, com novo JSESSIONID.
- Impede o uso de um ID de sessão previamente conhecido por um invasor.



SEGURANÇA DOS GOOMIES DA SESSÃO

SEGURANÇA DOS COOKIES DE SESSÃO

Falhas na configuração dos cookies de sessão podem permitir que invasores capturem ou reutilizem esses cookies em ataques como XSS – Cross Site Script ou Session Hijacking (sequestro de sessão). Além disso, o uso de conexões não seguras (HTTP em vez de HTTPS) amplia esse risco.

Exemplos práticos

Inseguro — PHP



```
php

session_start();

$_SESSION['usuario'] = 'admin';
echo "Sessão iniciada sem configurações de segurança.";
```

session_start() é chamado sem configurar os parâmetros de cookie. Isso faz com que o PHP use as configurações padrão, o que geralmente não inclui HttpOnly, Secure nem SameSite.

Seguro — PHP

```
ini_set('session.name', 'APPSESS');
ini_set('session.cookie_lifetime', '3600');
ini_set('session.cookie_path', '/');
ini_set('session.cookie_domain', $_SERVER['HTTP_HOST']);
ini_set('session.cookie_secure', '1');
ini_set('session.cookie_httponly', '1');
ini_set('session.cookie_samesite', 'Strict');
session_start();
$_SESSION['usuario'] = 'joao';
echo "Sessão iniciada com segurança via ini_set().";
```

Inseguro — Java

```
Java

// Sem qualquer controle sobre o cookie JSESSIONID
HttpSession session = req.getSession();
session.setAttribute("usuario", "admin");
```

- O cookie JSESSIONID pode ser acessado por scripts (sem HttpOnly).
- Se a aplicação estiver em HTTP, o cookie pode ser roubado via sniffing.

Seguro — Java

```
Cookie cookie = new Cookie("JSESSIONID", req.getSession().getId());
cookie.setHttpOnly(true);
cookie.setSecure(true); // Só via HTTPS
cookie.setPath("/");
cookie.setMaxAge(-1);
resp.addCookie(cookie);
```

- HttpOnly: protege contra XSS, bloqueando acesso ao cookie via JavaScript.
- Secure: garante que o cookie só será enviado por conexões HTTPS.
- Path e MaxAge: definem escopo e tempo de vida do cookie.

