**🧠 RCA – Problemas no Fluxo de Autenticação e Autorização**

**Projeto: FiberGuardian**

**Data: 14/07/2025**

**Autor: Giovanni Rozza**

**🔍 1. Contexto Inicial**

O sistema MPA com backend Spring Boot usava autenticação baseada em sessão (JSESSIONID), CSRF com CookieCsrfTokenRepository, e autenticação via endpoint POST /login. Foi identificado que o endpoint protegido POST /usuarios retornava **HTTP 403 Forbidden**, mesmo após login bem-sucedido.

**⚠️ 2. Problemas Identificados e Causas Raiz**

**🧩 2.1. Ausência de Role no Grant de Autorização**

* **Problema**: Usuário autenticava com sucesso, mas requisições a rotas protegidas por hasRole("ADMIN") eram negadas.
* **Causa**: A classe CustomUserDetailsService retornava UserDetails sem popular corretamente as GrantedAuthorities. O campo role do usuário estava sendo ignorado na criação da autorização.

**✔️ Solução Aplicada**

* Criada a classe UsuarioAutenticado que implementa UserDetails e gera a autoridade com o padrão "ROLE\_" + usuario.getRole().name().
* CustomUserDetailsService foi ajustada para retornar instâncias de UsuarioAutenticado.

**🧩 2.2. Incompatibilidade com XorCsrfTokenRequestAttributeHandler**

* **Problema**: O token CSRF era considerado inválido com a seguinte mensagem de log:
* Not returning the CSRF token since its Base64-decoded length (27) is not equal to (72)
* **Causa**: O handler XorCsrfTokenRequestAttributeHandler realiza ofuscação e espera tokens com tamanho fixo. Como a aplicação usa fetch(...) com envio direto de header "X-XSRF-TOKEN", esse comportamento interferiu.

**✔️ Solução Aplicada**

* Substituído XorCsrfTokenRequestAttributeHandler por CsrfTokenRequestAttributeHandler() padrão, eliminando a exigência de token ofuscado.
* Justificativa: como os cookies são HttpOnly=false e acessíveis via document.cookie, não há ganho real de segurança com Xor nesse contexto.

**🧩 2.3. Falta de persistência do SecurityContext na sessão**

* **Problema**: Após login bem-sucedido, a requisição para /usuarios era tratada como anônima (AnonymousAuthenticationFilter acionado).
* **Causa**: O SecurityContextHolder foi populado manualmente no controller de login, mas **não foi salvo na HttpSession**, como ocorre automaticamente quando se usa formLogin().

**✔️ Solução Aplicada**

* Após autenticação manual no AuthController, foi inserido explicitamente:
* session.setAttribute(
* HttpSessionSecurityContextRepository.SPRING\_SECURITY\_CONTEXT\_KEY,
* SecurityContextHolder.getContext()
* );
* Isso garantiu que a sessão passasse a conter o contexto de segurança necessário para futuras requisições autenticadas via JSESSIONID.

**🧪 3. Testes Realizados**

* Requisição com curl contendo os headers:
  + Cookie: JSESSIONID, XSRF-TOKEN
  + Header X-XSRF-TOKEN
* Fluxo completo validado:
  + Obtenção do token via /csrf-token
  + Login via /login (200 OK)
  + Inclusão de usuário via /usuarios (200 OK, com role ADMIN)

**🛡️ 4. Lições Aprendidas**

* Ao desabilitar formLogin(), é **obrigação do desenvolvedor salvar o SecurityContext** manualmente na sessão.
* **Atenção com handlers CSRF personalizados**, especialmente os que aplicam transformação nos tokens.
* Sempre verificar o conteúdo de SecurityContextHolder e se ele está sendo persistido corretamente para manter o estado autenticado entre requisições.

**✅ 5. Situação Final**

* O problema de autorização nas rotas protegidas foi resolvido.
* O token CSRF está sendo aceito corretamente.
* O SecurityContext é preservado na sessão entre as requisições.

Sistema agora opera corretamente com:

* Login via JSON
* CSRF Token seguro via Cookie + Header
* Sessão mantida via JSESSIONID
* Controle de acesso por role (hasRole("ADMIN"))