Advanced Spring

Spring Security

Boris Fresow, Markus Günther

Adesso eduCamp 2023

Mastichari, Kos

Introduction

- **Authentication** and **Authorization** framework for Java applications
- Highly customizable security framework
  - Integrated with Spring ecosystem
- Most recent version is 6.x

First class support for securing both imperative and reactive applications, it is the de-facto standard for securing Spring-based applications.

Authentication

Authentication

- Verifying user/system identity
- Supports various authentication mechanisms:
  - Username/password
  - OAuth2/OIDC
  - SAML
  - Custom
  - ...

Authentication (cont.)

Main components

- Authentication Manager
- Authentication Provider
- Password Encoder
- User Details Service

Authentication Manager

- Central authority for authentication
- Single method: `authenticate()`

```
public interface AuthenticationManager {
    Authentication authenticate(Authentication authentication) throws AuthenticationException;
}
```

- Return `Authentication` (with `authenticated=true`) if the input represents a valid principal
- Throw an `AuthenticationException` if the input represents an invalid principal
- Return `null` if it cannot decide
- Default implementation is the `ProviderManager` that delegates to a chain of `AuthenticationProvider`

Authentication Provider

- An `AuthenticationProvider` has an extra method to allow the caller to query whether it supports a given `Authentication` type.
- Does **not** extend the `AuthenticationManager` !

```
public interface AuthenticationProvider {

Authentication authenticate(Authentication authentication) throws AuthenticationException;

boolean supports(Class<?> authentication);

}
```

Authentication Provider (cont.)

- Responsible for one specific way of authentication
- Decision is made via the `supports` method if the provider is applicable

```
class OidcAuthenticationRequestChecker implements AuthenticationProvider {

(...)

    public boolean supports(Class<?> authentication) {
        return OAuth2LoginAuthenticationToken.class.isAssignableFrom(authentication);
    }
}
```

Password Encoder

Interface to perform a one-way transformation of a password to let the password be stored securely

- Default is the `DelegatingPasswordEncoder` that understands different password hashes
- The format might look familiar to you:

```
{id}encodedPassword
{bcrypt}`$2a$`10$dXJ3SW6G7P50lGmMkkmwe.20cQQubK3.HZWzG3YB1tlRy.fqvM/BG
{noop}password
{pbkdf2}5d923b44a6d129f3ddf3e3c8d29412723dcbde72445e8ef6bf3b508fbf17fa4ed4d6b99ca763d8dc
{sha256}97cde38028ad898ebc02e690819fa220e88c62e0699403e94fff291cfffaf8410849f27605abcbc0
```

User Detail Service

- Interface for retrieving user details
- Can use various data stores (e.g., database, LDAP, in-memory)
- `UserDetails` is an interface that describes the minimal set of user information

```
public interface UserDetailsService {

    UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;

}
```

Authorization

Authorization

- Determining and validating permissions for resources/actions
- Main components
  - Security Interceptor
  - Authorization Manager (formerly Access Decision Manager)
  - Granted Authority / Role

Security Interceptor

- Intercepts incoming requests
- Checks user permissions
- Commonly used examples: `FilterSecurityInterceptor`, `MethodSecurityInterceptor`

Authorization Manager

- Since Spring Security 6 the `AccessDecisionManager` is deprecated
- Makes final access control decision
- Takes an `Authentication` object and decides for a given resource if the permissions are sufficient
- Can be used with a different set of strategies to make the final decision:
  - `AuthorizationManagers.anyOf` - at least one of the managers grant access
  - `AuthorizationManagers.allOf` - all of the managers grant access
  - Consensus based decision is now longer provided out of the box

GrantedAuthority vs Role

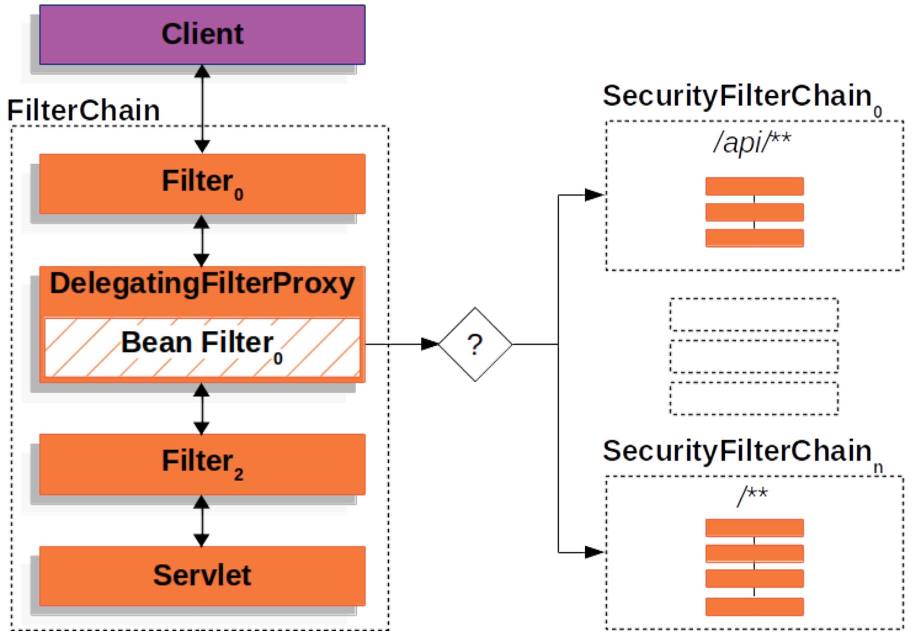
A `GrantedAuthority` and `Role` are technically the same construct with different semantics

- A `GrantedAuthority` is an individual permission with an **arbitrary name**
  - Is checked via `hasAuthority` - e.g. `@PreAuthorize("hasAuthority('WRITE_ENTRY')")`
- A `Role` is a `GrantedAuthority` container with the prefix `ROLE_` (per default)
  - Is checked via `hasRole` - e.g. `@PreAuthorize("hasRole('ROLE_EDITOR')")`

Securing Web Resources

Architecture

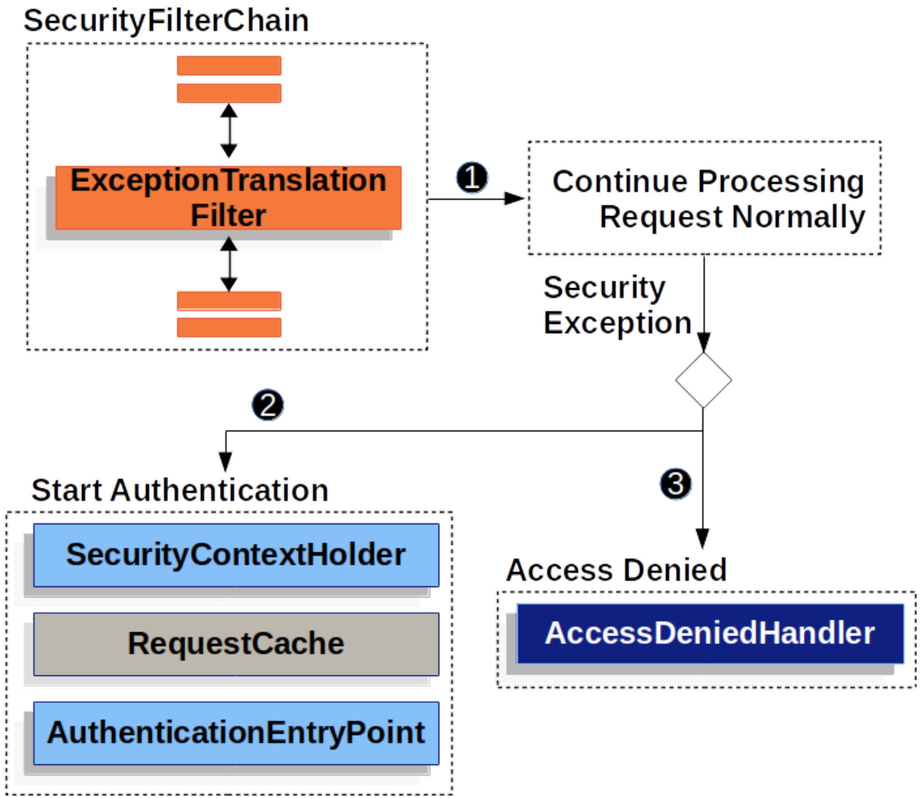
- Spring Security is executed as part of the `FilterChain`
- `SecurityFilter` are a chain again
- There can be 1...n `SecurityFilterChain` - one per context
- Ordering matters - **a lot!**



Exception Handling

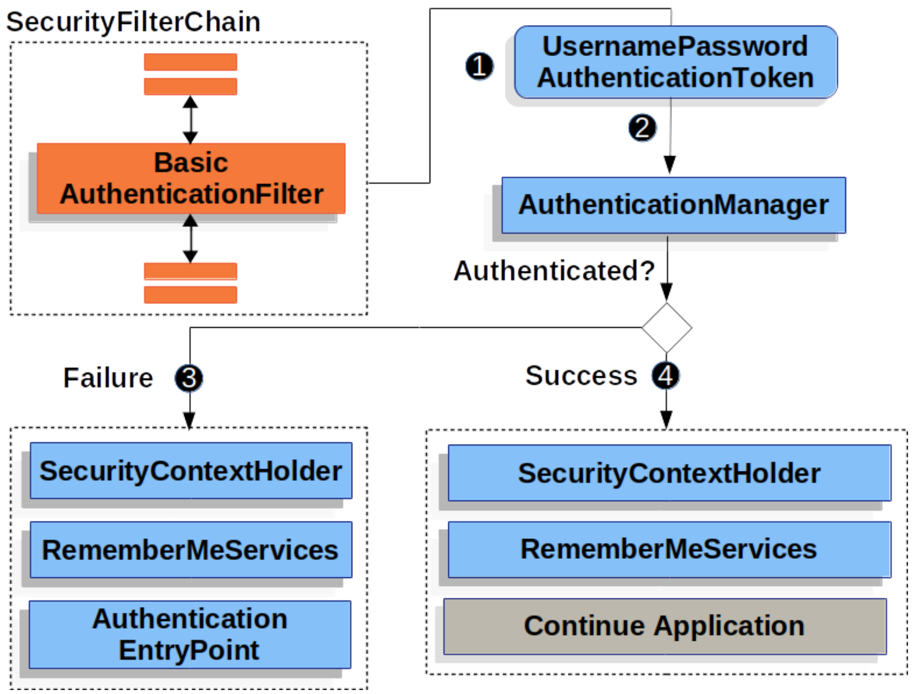
Two types of exception: `AuthenticationException` and `AccessDeniedException`

- `AuthenticationException` - start a new authentication if there is a `AuthenticationEntryPoint`, otherwise `401 - Unauthorized` for HTTP
- `AccessDeniedException` - terminate the request `403 - Forbidden` for HTTP



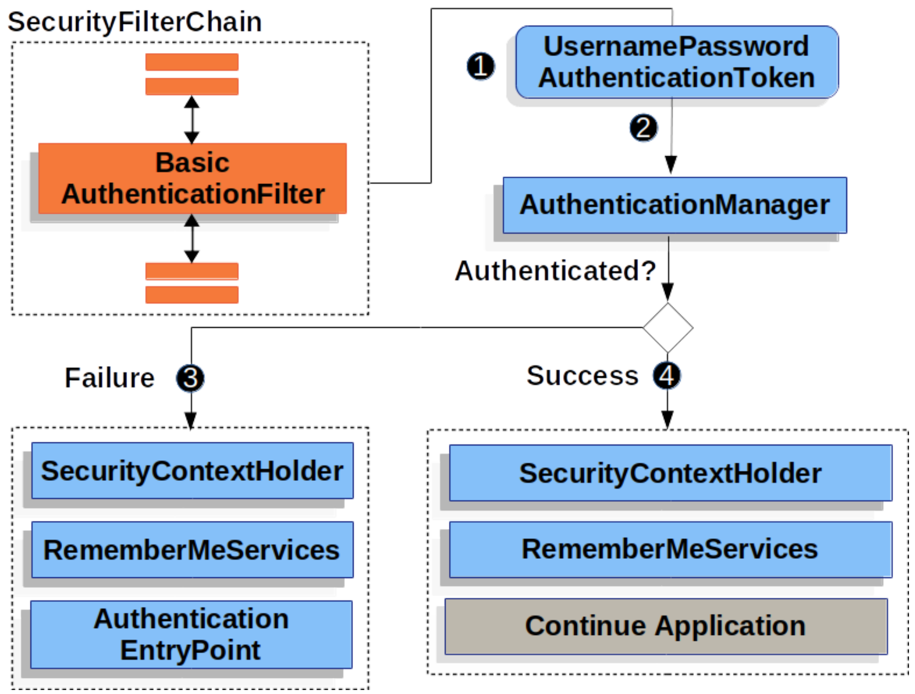
Basic Auth Example

- BasicAuthenticationFilter is triggered as part of the SecurityFilterChain
- UPAToken is a specialized version of AbstractAuthenticationToken
- The appropriate AuthenticationManager validates the token against a UserDetailsService
- The result is saved into the SecurityContextHolder (thread based)



Basic Auth Example (cont.)

- The `SecurityFilterChain` is configured with matchers
- This is the basis for the decision which `Filter` should be run in which Context in which order



Basic Auth Example (cont.)

- `@EnableWebSecurity` on a `@Configuration` class to configure `SecurityFilterChain`
- `@EnableMethodSecurity` to use method-level security annotations
- Matcher to enable Basic Auth for all matching URIs

```java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

  @Bean
  public SecurityFilterChain securityFilterChain
      (HttpSecurity http) throws Exception {

    http.authorizeHttpRequests(requests -> requests
            .requestMatchers("/**")
            .authenticated())
        .httpBasic(Customizer.withDefaults());

    return http.build();
  }
}
```

Basic Auth Example (cont.)

- `@PreAuthorize` to check for a specific role
- `Authentication` object will be injected from the current SecurityContext
  - `Principal` is also possible

```
@RequestMapping("/hello")
@RestController
public class HelloWorldController {

  @GetMapping(produces = MediaType.APPLICATION_JSON_VALUE)
  @PreAuthorize("hasRole('USER')")
  UserPermissions helloUser(Authentication authentication) {
    final var authorities = authentication.getAuthorities().stream()
        .map(GrantedAuthority::getAuthority)
        .toList();
    return new UserPermissions(authentication.getName(), authorities);
  }
}
```

Questions?

Lab

Lab

It's time to use (*some*) of that!

- Open the Spring Security Repository in your IDE
- Let's take a look at the repository and README.md