Spring Security Cheat Sheet

Auth · JWT · OAuth2 · Method Security

SECURITY FILTER CHAIN

```
@Bean SecurityFilterChain
chain(HttpSecurity http){
return http
.authorizeHttpRequests(a ->
a.requestMatchers("/public/**").permitAll()
.anyRequest().authenticated())
.build();
}
```

USERDETAILSSERVICE

```
@Service class UserSvc implements
UserDetailsService{
public UserDetails
loadUserByUsername(String u){
User user = repo.findByEmail(u);
return new
org.springframework.security.core
.userdetails.User(user.email,
user.pwHash,
mapRoles(user.roles));
}
}
```

PASSWORD ENCODING

```
@Bean PasswordEncoder encoder(){
return new BCryptPasswordEncoder();
}
encoder.encode(rawPassword)
encoder.matches(raw, stored)
```

BCrypt strength 1012; use Argon2 for new projects

JWT FILTER

```
class JwtFilter extends
OncePerRequestFilter{
void doFilterInternal(req, res, chain){
String token = extractToken(req);
if(jwtUtil.validate(token)){
var auth =
jwtUtil.getAuthentication(token);
SecurityContextHolder.getContext().setAuthentication(auth);
}
chain.doFilter(req, res);
}
}
```

OAUTH2 / OIDC

```
<dep>spring-boot-starter-oauth2-client</dep>
spring.security.oauth2.client.registration
.google.client-id=...
.google.client-secret=...
http.oauth2Login(withDefaults())
```

OAuth2User / OidcUser principals available

METHOD SECURITY

```
@EnableMethodSecurity
@PreAuthorize("hasRole('ADMIN')")
@PostAuthorize("returnObject.owner ==
authentication.name")
@Secured({"ROLE_USER","ROLE_ADMIN"})
@RolesAllowed("ADMIN") (JSR-250)
```

CORS & CSRF

```
http.cors(cors ->
cors.configurationSource(source))
CorsConfiguration cfg = new
CorsConfiguration();
cfg.addAllowedOrigin("https://myapp.com");
```

CSRF enabled by default; disable for stateless APIs:

```
http.csrf(AbstractHttpConfigurer::disable)
```

RBAC PATTERN

```
.requestMatchers("/admin/**").hasRole("ADMIN")
.requestMatchers("/api/**").hasAuthority("SCOPE_read")
```

Role = ROLE_ prefix; Authority = any string

`GrantedAuthority interface`

Hierarchy: RoleHierarchyImpl for nested roles

EXCEPTION HANDLING

```
http.exceptionHandling(e ->
e.authenticationEntryPoint(myEntryPoint)
.accessDeniedHandler(myHandler))
```

401 Unauthorized vs 403 Forbidden

```
AuthenticationEntryPoint:
commence(req,res,ex)
AccessDeniedHandler: handle(req,res,ex)
```

SECURITY CONTEXT

```
Authentication auth =
SecurityContextHolder
.getContext().getAuthentication();
String name = auth.getName();
Collection<? extends GrantedAuthority>
roles = auth.getAuthorities();
@AuthenticationPrincipal UserDetails
user  in controllers
```

TESTING

```
@WithMockUser(roles="ADMIN")
@WithUserDetails("admin@example.com")
mvc.perform(get("/admin")).andExpect(status().isOk())
@WithSecurityContext for custom auth
SecurityMockMvcRequestPostProcessors.csrf()
for POST
```

COMMON GOTCHAS

CSRF must be disabled for REST APIs with JWT
Roles need ROLE_ prefix in hasRole()
SecurityContextHolder clears after request by default
Don't store sensitive data in JWT it's not encrypted
HTTPS required in prod; use HSTS header