API-Key Authentifizierung

(Spring Boot 3)

Die Security-Konfiguration hat sich in **Spring Security 6** deutlich verändert. Beispielsweise wurde die *WebSecurityConfigurerAdapter-Klasse* entfernt und durch eine *SecurityFilterChain-Bean* ersetzt.

(Diese Notiz als Pdf)

Inhalt

1. Beispielprojekt

Das Projekt realisiert einen Spring Boot Webservice. Der Service bietet frei zugängliche und administrative Resourcen an. Die administrativen Resourcen benötigen einen Api-Key zur Authentifikation des Benutzers.

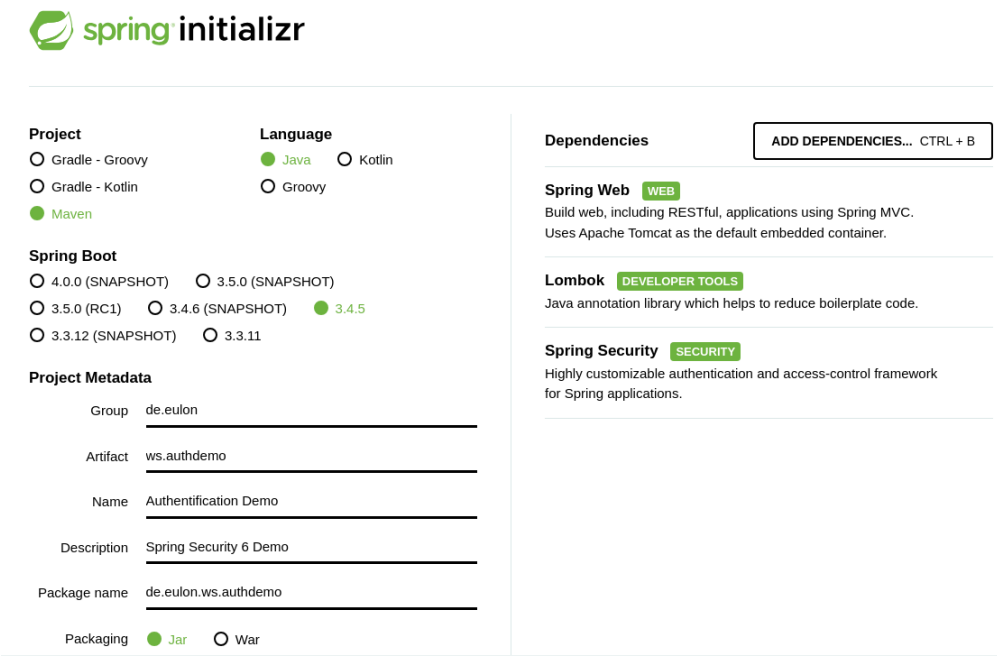

Abbildung 1. spring initializr

Dieses Projekt wird intern verwendet. Das Passwort ist in Form eines Hash in der Konfiguration abgelegt. Grundsätzlich und vor allem bei öffentlich bereitgestellten Resourcen sollten Passwörter in einem Key-Vault, z.B. einem Hashicorp Vault, einem OpenBao-Vault oder etwas ähnlichem vorgehalten werden.

2. Konfiguration

Configuration-SecurityFilterChain (Ausschnitt)

```
:
:
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfiguration
{

   private final ApiKeyAuthenticationFilter authenticationFilter;   ①

   @Bean
   public SecurityFilterChain securityFilterChain( HttpSecurity http ) throws Exception
   {
      return http
            .sessionManagement( session -> session.sessionCreationPolicy( SessionCreationPolicy.STATELESS ) )     ②
            .formLogin(AbstractHttpConfigurer::disable)  ③
            .csrf( AbstractHttpConfigurer :: disable )  ④
            .securityMatcher( "/**" )       ⑤
            .authorizeHttpRequests( registry -> registry
                  .requestMatchers( AntPathRequestMatcher.antMatcher( "/internal/**" ) ).authenticated()        ⑥
                  .anyRequest().permitAll()     ⑦
            )
            .addFilterBefore( authenticationFilter, UsernamePasswordAuthenticationFilter.class )        ⑧
            .build();
   }

}
:
:
```

① ApiKeyAuthenticationFilter ist ein Requestfilter, der den eingehenden Request anhand der Security-Regeln filtert und bei Bedarf die Authentifizierungslogik anwendet. Dazu wird die Methode "*doFilterInternal*" überschrieben, die als Parameter u.a. die FilterChain-Bean und den Request enthält.

② Stateless Sessions stellen sicher, dass jeder Request neu authetifiziert wird, da keine Cookies verwendet werden.

③ Damit wird die Out-of-the-box Authentifizierung über ein Formular, d.h. die Weiterleitung auf eine Default-Loginseite verhindert.

④ Spring Security schützt standardmäßig vor CSRF-Angriffen für unsichere HTTP-Methoden, wie z. B. eine POST-Anfrage, sodass kein zusätzlicher Code erforderlich ist. Um POST-Anfragen durchführen zu können, muss CSRF disabled sein.

⑤ hier kann festgelegt werden, für welche Endpunkte die Sicherheitskonfiguration gültig ist (in diesem Fall für alle Endpunkte).

⑥ Dieser Matcher legt fest, dass alle Resourcen, die "internal" in der Url enthalten authentifiziert werden müssen.

⑦ Dieser Matcher legt fest, dass alle weiteren Resourcen keiner Authentifizierung unterliegen.

⑧ Es wird sichergestellt, dass die individuelle Filterung immer vor der sonst an erster Stelle (default) stehenden User:Password Filterung durchgeführt wird.

3. Controller

Der Controller enthält Resourcen, die keiner Zugangsbeschränkung unterliegen und administrative Resourcen, die mit einem Api-Key abgesichert werden.

Controller-Resourcen

```
    :
    :
    @GetMapping("/doSomething")
    public ResponseEntity<String> getDoSomething() {
       return ResponseEntity.ok("This is a public resource.");
    }

    @GetMapping("/internal/doAdmin")
    public ResponseEntity<String> getDoAdmin() {
       return ResponseEntity.ok("This is a a protected resource.");
    }
    :
    :
```

4. Filterung

ApiKeyAuthenticationFilter erbt von einem Requestfilter, der den eingehenden Request anhand der Security-Regeln filtert und bei Bedarf die Authentifizierungslogik anwendet. Dazu wird die Methode "*doFilterInternal*" überschrieben.

Filterung-ApiKeyAuthenticationFilter

```
:
:
@Component
@RequiredArgsConstructor
public class ApiKeyAuthenticationFilter extends OncePerRequestFilter
{
   private final ApiKeyAuthExtractor extractor;       ①

   @Override
   protected void doFilterInternal( HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain ) throws ServletException, IOException
   {
      extractor.extract( request )
```

```
                .ifPresent( SecurityContextHolder.getContext() :: setAuthentication );        ②

        filterChain.doFilter( request, response );
    }
}
:
:
```

① Der Extractor ist die Klasse, die die Authentifizierung durchführt, d.h. in diesem Fall den übergebenen Api-Key überprüft.

② Abhängig vom Ergebnis der Extractor-Klasse wird ein Request als authentifiziert bzw. nicht authentifiziert angesehen.

5. Authentifizierungs-Handler

ApiKeyAuthExtractor ist ein Handler, der in der Methode "*extract*" in diesem Fall den im Request übergebenen Key mit einem vorhandenen Wert, z.B. aus einem Key-Vault oder Properties-File vergleicht und damit eine erfolgreiche/nicht erfolgreiche Authentifizierung anzeigt.

Authentifizierung-ApiKeyAuthExtractor

```
:
:
    @Value("${security.key}")
    private String apiKey;

    public Optional<Authentication> handle( HttpServletRequest request) {
        String providedKey = request.getHeader("api-key");
        if (providedKey == null || !providedKey.equals(apiKey))
            return Optional.empty();

        return Optional.of(new ApiKeyAuthentication( providedKey, AuthorityUtils.NO_AUTHORITIES));
    }
:
:
```

6. Authentifizierungs-Objekt

ApiKeyAuthentication ist ein Authentifizierungs-Object, d.h. ein Token der bei einer erfolgreichen Authetifizierung erzeugt wird, d.h. wenn im Requestheader ein korrekter API-Schlüssel gefunden wird.

Authentifizierung-ApiKeyAuthentication

```
:
:
public class ApiKeyAuthentication extends AbstractAuthenticationToken
{
    private final String apiKey;

    public ApiKeyAuthentication(String apiKey, Collection<? extends GrantedAuthority> authorities) {
```

```
        super(authorities);
        this.apiKey = apiKey;
        setAuthenticated(true);
    }

    @Override
    public Object getCredentials() {
        return null;
    }

    @Override
    public Object getPrincipal() {
        return apiKey;
    }
}
:
:
```

(Diese Notiz als Pdf)