The Technoblogist

Technology for Technologists

Preauthenticated scenario with spring-boot

Posted on 30/05/2017 by techn142_wp

Today i want to share how to set up spring-security over spring-boot application on a preauthenticated scenario.

Preauthentication is a spring-security scenario supported for quite a long time. It is often used when the authentication is provided using X.509 certificates, or when you relay in an external authentication system.

Imagine a microservice architecture implementing an api-gateway pattern. In this case you might want to avoid implementing authentication in all microservices and rely on a perimetral authentication implemented at the api-gateway component. There are many ways to implement this, and we can talk about that in a different post, but the important thing is that, even when you want to avoid authentication in all microservices you will want to know the principal. Probably to implement authorization of some kind, that can be role based endpoint access, filtering information to be sure principal only access his own data, or any other data access control mechanism you prefer.

There are several handy classes already available in spring security. Under the package:`org.springframework.security.web.authentication.preauth` you may take a look at:

- PreAuthenticatedAuthenticationProvider
- PreauthenticatedGrantedAuthoritiesUserDetailsService
- RequestHeaderAuthenticationFilter

With all those pieces you should have already an idea of what comes here: The plan is api-gateway component will place the result of the authentication as custom headers, and rest of the services will transform those headers in a principal object that will be injected in the securityContext where it could be used aside with `@Secured`, `@PreAuthorize` annotations, or any other authorization method you would like.

Now lets see the code!

All you need to do in order to confugure preauthenticated filtering in spring-boot is a usual configuration bean extending WebSecurityConfigurerAdapter

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true)
public class WebSecurityConfiguration extends WebSecurityConfigurerAdapter
```

Then you need to set the PreAuthenticatedAuthenticationProvider. e.g. like this:

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Except
    auth.authenticationProvider(preauthAuthProvider());
}
```

```
@Bean
public PreAuthenticatedAuthenticationProvider preauthAuthProvider() {
  PreAuthenticatedAuthenticationProvider preauthAuthProvider =
    new PreAuthenticatedAuthenticationProvider();
  preauthAuthProvider.setPreAuthenticatedUserDetailsService(
    new PreAuthenticatedGrantedAuthoritiesUserDetailsService()
  );
  return preauthAuthProvider;
}
```

Now you shall declare a RequestHeaderAuthenticationFilter

```
  @Bean
  public RequestHeaderAuthenticationFilter requestHeaderAuthenticationFilter(
      final AuthenticationManager authenticationManager) {
          RequestHeaderAuthenticationFilter filter =
              new RequestHeaderAuthenticationFilter();
  }
```

There are many configuration options in this class, like customizing the principal header name. Look at the documentation for addtional details. **NOTE the standard header for the principal id is SM_USER** (this is related an external authentication provider called SiteMinder)

Finally you can configure the FilterChainProxy like this:

```
@Override
protected void configure(final HttpSecurity http) throws Exception {
  http.authorizeRequests()
    .antMatchers("/**").authenticated()
    .and()
    .sessionManagement()
    .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
    .and().securityContext()
    .and()
    .addFilterBefore(requestHeaderAuthenticationFilter(authenticationManager()
      LogoutFilter.class);
}
```

With this simple configuration you can retrieve the principal id from standard or custom header and inject it in security context. There are plenty customization options on how the security process must behave depending on your needs, like what to do if header is absent, what HTTP status code should be used, and so on.

In next entry post i will write how to deal with authorization since there are a couple of tricky things to have in mind there.

Happy coding!!

Posted in Technology *Tagged* #coding, #java, #spring, #spring-security

Caveheats performing authorization on spring data rest endpoints

Authorization on a PreAuthenticated scenario (Round 1)

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment