[**Spring Security Pre-Authentication and Authorization using Java Configurations**](http://www.learningthegoodstuff.com/2014/12/spring-security-pre-authentication-and.html)

I've been working extensively with Spring Boot recently and have grown to love Java configurations.  I spend extra time just so I never have to write any **XML**.  
  
Spring Security confused the hell out of me,  and I was tempted to use an existing XML configuration and be done with it.  But **I hate XML**, so I fought that urge.  My findings are below.  If I find more I'll update this blog post.

The configuration below is meant to be argued about.

1. It can be deployed **behind SiteMinder**.
2. It uses **Pre-Authentication** to grab the pre-auth header (SM\_USER by default).
   1. If the header isn't present, you're out!
3. Assuming you have a header we allow you to attempt to access **HTTP endpoints**.
   1. If you're header value is in the **authorized list of users** who have access to the requested resource, you're in! (JSON returned)
   2. Otherwise, **you're out!** (Access Denied)

Things I'd like you to know:

* A **principal**is the representation of the User Spring Security uses during Authentication.
* **Please do not use this blindly**.  Figure out what it is doing and do it better.
* **The code does not comment itself**, but I hope this may help you if you're poking around in the dark like I was.
* This was deployed with Spring Boot using a Tomcat 7 embedded app server.
* If I did something **stupid** leave a comment.
* **CSRF is disabled because it is**.  Please figure out how it works and enable it.
* The configuration below **is meant to be argued about, but should get you going**.

**ThingController.java**

// Comment

@RestController

@RequestMapping("/api")

public class ThingController {

@Autowired

private ThingService thingService;

/\*\*

\* Retrieve a Thing by the id of the Thing.

\*/

@RequestMapping(value = "/thing/{thingId}" , method = RequestMethod.GET)

@PreAuthorize("isAuthenticated() and hasPermission(#thingId, 'path.to.Thing', 'read')")

public ResponseEntity<Thing> requestSectionById(@PathVariable("thingId") String thingId) {

return thingService.requestThingById(thingId);

}

}

**SecurityConfig.java**

@Configuration

@EnableWebMvcSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

private final static Logger logger = LoggerFactory.getLogger(SecurityConfig.class);

@Override

protected void configure(HttpSecurity http) throws Exception {

http.addFilterBefore(ssoFilter(), RequestHeaderAuthenticationFilter.class)

.authenticationProvider(

preauthAuthProvider())

.csrf().disable()

.authorizeRequests().anyRequest().authenticated();

}

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth.authenticationProvider(preauthAuthProvider());

}

@Bean

public UserDetailsByNameServiceWrapper<PreAuthenticatedAuthenticationToken> userDetailsServiceWrapper() {

UserDetailsByNameServiceWrapper<PreAuthenticatedAuthenticationToken> wrapper =

new UserDetailsByNameServiceWrapper<PreAuthenticatedAuthenticationToken>();

wrapper.setUserDetailsService(new CustomUserDetailsService());

return wrapper;

}

@Bean

public PreAuthenticatedAuthenticationProvider preauthAuthProvider() {

PreAuthenticatedAuthenticationProvider preauthAuthProvider =

new PreAuthenticatedAuthenticationProvider();

preauthAuthProvider.setPreAuthenticatedUserDetailsService(userDetailsServiceWrapper());

return preauthAuthProvider;

}

@Bean

public SSORequestHeaderAuthenticationFilter ssoFilter() throws Exception {

SSORequestHeaderAuthenticationFilter filter = new SSORequestHeaderAuthenticationFilter();

filter.setAuthenticationManager(authenticationManager());

return filter;

}

}

**SSORequestHeaderAuthenticationFilter.java**

/\*\*

\* Handles for SSO request headers to create Authorization ids.

\*/

public class SSORequestHeaderAuthenticationFilter extends RequestHeaderAuthenticationFilter {

private boolean allowPreAuthenticatedPrincipals = true;

public SSORequestHeaderAuthenticationFilter() {

super();

//TODO Pull this value from a properties file (application.properties, or localstrings.properties)

//NOTE SM\_USER is the default, but you can change it like this (your company may use some other header)

this.setPrincipalRequestHeader("SM\_USER");

}

/\*\*

\* This is called when a request is made, the returned object identifies the

\* user and will either be {@literal null} or a String. This method will throw an exception if

\* exceptionIfHeaderMissing is set to true (default) and the required header is missing.

\*

\* @param request {@link javax.servlet.http.HttpServletRequest}

\*/

@Override

protected Object getPreAuthenticatedPrincipal(HttpServletRequest request) {

String userName = (String) (super.getPreAuthenticatedPrincipal(request));

if (userName == null || userName.trim().equals("")) {

return userName;

}

return userName;

}

public boolean isAllowPreAuthenticatedPrincipals() {

return allowPreAuthenticatedPrincipals;

}

}

**CustomUserDetailsService.java**

@Component

public class CustomUserDetailsService implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

//NOTE The implementation of this method is for example purposes only.

List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();

SimpleGrantedAuthority authority = new SimpleGrantedAuthority("ROLE\_USER");

authorities.add(authority);

//TODO Why do we need a password here if we are pre-authenticating?

UserDetails user = new User(username, "password", authorities);

return user;

}

}

**CustomPermissionEvaluator.java**

@Component

public class CustomPermissionEvaluator implements PermissionEvaluator {

<GrantedAuthority>

//NOTE - This is for example purposes only.

private static final Map<String, List<String>> testPermissionMap;

static {

Map<String, List<String>> map = new HashMap<String, List<String>>();

ArrayList<String> things = new ArrayList<String>();

things.add("11111");

things.add("22222");

things.add("33333");

things.add("44444");

ArrayList<String> badThings = new ArrayList<String>();

badThings.add("99999");

badThings.add("XXXXX");

map.put("212389921", things);

map.put("999999999", badThings);

testPermissionMap = Collections.unmodifiableMap(map);

}

public boolean hasPermission(Authentication authentication, Object targetDomainObject,

Object permission) {

return true;

};

public boolean hasPermission(Authentication authentication, Serializable targetId,

String targetType, Object permission) {

return authorize((User) authentication.getPrincipal(), (String) targetId);

};

public boolean authorize(User user, String thingId) {

boolean allowed = false;

System.out.println("Authorizing " + user.getUsername() + "...");

if (testPermissionMap.get(user.getUsername()) != null &&

testPermissionMap.get(user.getUsername()).contains(thingId)) {

allowed = true;

System.out.println(user.getUsername() + " authorized!");

}

return allowed;

};

}

**MethodSecurityConfig.java**

@Configuration

@EnableGlobalMethodSecurity(prePostEnabled = true)

public class MethodSecurityConfig extends GlobalMethodSecurityConfiguration {

@Override

protected MethodSecurityExpressionHandler createExpressionHandler() {

DefaultMethodSecurityExpressionHandler expressionHandler = new DefaultMethodSecurityExpressionHandler();

expressionHandler.setPermissionEvaluator(new CustomPermissionEvaluator());

return expressionHandler;

}

}

**User.java**

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.userdetails.UserDetails;

public class User implements UserDetails{

private static final long serialVersionUID = 1924132530173910545L;

private Collection authorities;

private String password;

private String username;

public User() {}

public User(String username, String password) {

this.username = username;

this.password = password;

}

public User(String username, String password, Collection authorities) {

this.username = username;

this.password = password;

this.authorities = authorities;

}

@Override

public Collection getAuthorities() {

return authorities;

}

@Override

public String getPassword() {

return this.password;

}

@Override

public String getUsername() {

return this.username;

}

//NOTE If any of these things can be retrieved from your SSO Authentiation point, you may want to use them.

@Override

public boolean isAccountNonExpired() {

return true;

}

@Override

public boolean isAccountNonLocked() {

return true;

}

@Override

public boolean isCredentialsNonExpired() {

return true;

}

@Override

public boolean isEnabled() {

return true;

}

}