Tapestry5HowToSecureWithSpringAndLDAP

This article describes how you can easily integrate Tapestry5 with Spring Security using LDAP for storing your users and roles.

Setting up

The only additional dependencies you'll need is the tapestry-spring-security integration module that you can find here and the spring-ldap project. If your using maven you can download them by adding the following to your POM:

```
<dependency>
  <groupId>nu.localhost.tapestry</groupId>
  <artifactId>tapestry-spring-security</artifactId>
  <version>2.0.0</version>
</dependency>

<dependency>
  <groupId>org.springframework.ldap</groupId>
  <artifactId>spring-ldap</artifactId>
  <version>1.2.1</version>
</dependency>

...and...

<repository>
   <id>localhost.nu</id>
   <url>http://www.localhost.nu/java/mvn</url>
</repository>
```

Configuring your application

First you'll need to add something to the Tapestry filter mapping in your *web.xml* file. This is because by default when you specify a filter mapping the filter will actually only be invoked when actual request are made to the configured URL, but not when the RequestDispatcher forwards a request to that URL. Spring Security uses FORWARD so you'll have to make you filter mapping look something like this:

```
<filter-mapping>
   <filter-name>app</filter-name>
   <url-pattern>/*</url-pattern>
   <dispatcher>FORWARD</dispatcher>
   <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

Now you'll need to configure your security services. As every configuration in Tapestry, this is done in the application module. I usually create and Security Module submodule so that i don't get my main module filled with security stuff, but you can place the following code in any module you want.

```
public class SecurityModule
{
  public static UserDetailsService buildLdapUserDetailsService(final @Inject LdapUserSearch ldapUserSearch,
                                                                final @Inject @Value("${ldap-user-description-
attribute}") String userDescriptionAttribute,
                                                                final Logger logger)
  {
    return new UserDetailsService()
    {
      /**
       * Finds user details by the username.
       *
       * @param username the username to look for.
       * @return the user details or <code>null</code> if no user is found with the given username.
       */
      public UserDetails loadUserByUsername(String username)
      {
        try
        {
          DirContextOperations user = ldapUserSearch.searchForUser(username);

          Person.Essence person = new Person.Essence(user);
```

```
          person.setDescription(user.getAttributes("").get(userDescriptionAttribute).get().toString());
          person.setUsername(username);

          return person.createUserDetails();
        } catch (UsernameNotFoundException ex)
        {
          logger.info("Couldn't find user with username \"" + username + "\".");

          return null;
        } catch (NamingException ex)
        {
          logger.error("Error finding user with username \"" + username + "\".");

          return null;
        }
      }
    };
  }

  public static SpringSecurityContextSource buildInitialDirContextFactory(@Inject @Value("${ldap-provider-
url}") String providerUrl,
                                                                           @Inject @Value("${ldap-manager-dn}")
String managerDn,
                                                                           @Inject @Value("${ldap-manager-
password}") String managerPassword) throws Exception
  {
    assert providerUrl != null;

    assert managerDn != null;

    assert managerPassword != null;

    // Initialize the context factory
    DefaultSpringSecurityContextSource contextSource = new DefaultSpringSecurityContextSource(providerUrl);
    contextSource.setUserDn(managerDn);
    contextSource.setPassword(managerPassword);
    contextSource.afterPropertiesSet();

    return contextSource;
  }

  public static LdapUserSearch buildFilterBasedLdapUserSearch(SpringSecurityContextSource factory,
                                                              @Inject @Value("${ldap-users-search-base}")
String usersSearchBase)
  {
    FilterBasedLdapUserSearch userSearch = new FilterBasedLdapUserSearch(usersSearchBase, "(cn={0})", factory);

    // search in subtrees
    userSearch.setSearchSubtree(true);

    userSearch.setDerefLinkFlag(true);

    return userSearch;
  }

  public static AuthenticationProvider buildLdapAuthenticationProvider(SpringSecurityContextSource factory,
@Inject LdapUserSearch ldapUserSearch,
                                                                       @Inject @Value("${ldap-roles-search-
base}") String rolesSearchBase)
          throws Exception
  {
    BindAuthenticator authenticator = new BindAuthenticator(factory);
    authenticator.setUserSearch(ldapUserSearch);
    authenticator.afterPropertiesSet();

    DefaultLdapAuthoritiesPopulator populator = new DefaultLdapAuthoritiesPopulator(factory, rolesSearchBase);
    populator.setGroupRoleAttribute("cn");
    populator.setGroupSearchFilter("member={0}");
    populator.setDefaultRole("ROLE_ANONYMOUS");
    populator.setConvertToUpperCase(true);
    populator.setSearchSubtree(true);
```

```
    populator.setRolePrefix("ROLE_");

    return new LdapAuthenticationProvider(authenticator, populator);
  }

  public static void contributeProviderManager(OrderedConfiguration<AuthenticationProvider> configuration,
                                               @InjectService("LdapAuthenticationProvider")
AuthenticationProvider ldapAuthenticationProvider)
  {
    configuration.add("ldapAuthenticationProvider", ldapAuthenticationProvider);
  }

  public static void contributeApplicationDefaults(MappedConfiguration<String, String> configuration)
  {
    // Url redirected to when trying to use a secured class and/or method.
    configuration.add("spring-security.loginform.url", "/login");

    // Url redirected to when fails to login.
    configuration.add("spring-security.failure.url", "/login/failed");

    // If set to other than empty, the request dispatcher will "forward" to this specified error page view.
From Acegi documentation: The error page to use.
    // Must begin with a "/" and is interpreted relative to the current context root.
    configuration.add("spring-security.accessDenied.url", "/accessdenied");

    // Change the default password encoder. Must implement org.acegisecurity.providers.encoding.PasswordEncoder.
    configuration.add("spring-security.password.encoder", "org.springframework.security.providers.encoding.
Md5PasswordEncoder");

    // Page redirected to after a successful logout.
    configuration.add("spring-security.afterlogout.page", "Index");
  }
}
```

I think most of is self explanatory. As you can see there are a lot inject symbols, this is because i wanted to keep this solution as generic as possible so it could be used in other projects without changing any code. You'll find more information on some of the configurations made here.

Lets use it

Using this couldn't be simpler. Secure your application the way its described here. Just add the @Secured annotation on your page classes and methods and it's done.