Tapestry5HowToSpringSecurityAndOpenId

Motivation

OpenID is a service providing single-sign-on features. At our institute we are operating an OpenID provider that delegates authentication to our ActiveDirectory where all our students are registered. For a course planning system I was assessing Tapestrys integration possibilities with this OpenID provider and found out, that the tapestry-spring-security project already has much of the stuff needed for integration with our provider on board. This is a description of what you have to do in order to have OpenID authentication in your Tapestry apps.

Background

Spring Security already comes with everything needed for authenticating users against an OpenID provider (although this is somewhat incomplete regarding additional attributes that can be provided by the OpenID provider). But the tapestry-spring-security lacks the configuration needed to use this out of the box. In particular it lacks the filter configuration for the OpenIDAuthenticationProcessingFilter that processes incoming OpenID authentications. In a "normal" setup this is just another Filter that you configure in your web.xml. In a Tapestry setup though, Tapestry receives all incoming requests and processes them in the HttpServletRequestHandler pipeline. Fortunately this pipeline can be extended by contributing HttpServletRequestFilters. So all we need to do is wrap the OpenIDAuthenticationProcessingFilter and contribute it to the HttpServletRequestHandler pipeline. Such a wrapper is provided by the tapestry-spring-security project. With that in place we can process incoming OpenID authentication requests. Next, we have to look up the authenticated user in our local database where we store the roles assigned to a user. For this we have to implement a UserDetailsService which retrieves a user and his roles from some data store. The user object that is returned has to implement the UserDetails interface and the roles he holds have to implement the GrantedAuthority interface. If you don't want to grant access to some OpenID authenticated user, you have to do that in the UserDetailsService. The implementation I will show you grants accesss to EVERY OpenID authenticated user so don't use this in a "real" environment.

Requirements

tapestry-hibernate and tapestry-spring-security have to be in your pom.xml (or their jars and all of their dependencies have to be in your classpath). Configuring tapestry-hibernate is outside the scope of this article, please consult it's own documentation.

Implementation

Let's begin with the User and Role objects that implement the UserDetails and GrantedAuthority interfaces, respectively. These are used to store users and their roles in a local database using Hibernate. The User is a minimalistic implementation that only stores an id and a username.

```
package org.mygroup.myapp.entities;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.Transient;

import org.springframework.security.GrantedAuthority;
import org.springframework.security.GrantedAuthorityImpl;
import org.springframework.security.userdetails.UserDetails;

/**
 * A minimalistic UserDetails implementation providing a username only. Storing
 * a password is not necessary since the OpenID provider will do the authentication.
 *
 * @author Ulrich St&auml;rk
 */
@Entity
public class User implements UserDetails
{
```

```
private static final long serialVersionUID = 4068206679084877888L;

private int id;

private String username;

private Set<Role> roles;

@Id
@GeneratedValue(strategy = GenerationType.AUTO)
public int getId()
{
    return id;
}

public void setId(int id)
{
    this.id = id;
}

@ManyToMany
public Set<Role> getRoles()
{
    return roles;
}

public void setRoles(Set<Role> roles)
{
    this.roles = roles;
}

@Transient
public GrantedAuthority[] getAuthorities()
{
    Set<GrantedAuthority> authorities = new HashSet<GrantedAuthority>();

    for (Role role : getRoles())
    {
        authorities.add(new GrantedAuthorityImpl(role.getAuthority()));
    }

    return authorities.toArray(new GrantedAuthority[authorities.size()]);
}

@Transient
public String getPassword()
{
    return "notused";
}

public String getUsername()
{
    return username;
}

public void setUsername(String username)
{
    this.username = username;
}

@Transient
public boolean isAccountNonExpired()
{
    return true;
}

@Transient
public boolean isAccountNonLocked()
{
    return true;
}
```

```
    @Transient
    public boolean isCredentialsNonExpired()
    {
        return true;
    }

    @Transient
    public boolean isEnabled()
    {
        return true;
    }
}
```

```
package org.mygroup.myapp.entities;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import org.springframework.security.GrantedAuthority;

/**
 * A GrantedAuthority (aka Role) implementation. hashCode, equals and compareTo are
 * a bit messy (auto-generated), you might want to change this.
 *
 * @author Ulrich St&auml;rk
 */
@Entity
public class Role implements GrantedAuthority
{
    private static final long serialVersionUID = -117212611936641518L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    private String authority;

    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        this.id = id;
    }

    public String getAuthority()
    {
        return authority;
    }

    public void setAuthority(String authority)
    {
        this.authority = authority;
    }

    @Override
    public int hashCode()
    {
        return authority.hashCode();
    }

    @Override
    public boolean equals(Object obj)
```

```
    {
        if (this == obj) return true;
        if (obj == null) return false;
        if (obj instanceof String) return obj.equals(authority);
        if (getClass() != obj.getClass()) return false;
        final Role other = (Role) obj;
        if (authority == null)
        {
            if (other.authority != null) return false;
        }
        else if (!authority.equals(other.authority)) return false;
        return true;
    }

    public int compareTo(Object o)
    {
        if (this == o) return 0;
        if (o == null) return -1;
        if (o.getClass() == String.class) return authority.compareTo((String) o);
        if (getClass() != o.getClass()) return -1;
        final Role other = (Role) o;
        if (authority == null)
        {
            if (other.authority != null) return 1;
        }
        else
            return authority.compareTo(other.authority);
        return -1;
    }
}
```

Next, we need a UserDetailsService implementation, that finds users and their roles in our local database:

```
package org.mygroup.myapp.services.impl;

import java.util.HashSet;

import org.apache.tapestry5.hibernate.HibernateSessionManager;
import org.hibernate.Session;
import org.hibernate.criterion.Restrictions;
import org.slf4j.Logger;
import org.springframework.dao.DataAccessException;
import org.springframework.security.userdetails.UserDetails;
import org.springframework.security.userdetails.UserDetailsService;
import org.springframework.security.userdetails.UsernameNotFoundException;

import de.spielviel.mailadmin.entities.Role;
import de.spielviel.mailadmin.entities.User;

/**
 * @author Ulrich St&auml;rk
 */
public class UserDetailsServiceImpl implements UserDetailsService
{
    private HibernateSessionManager sessionManager;

    private Logger logger;

    public UserDetailsServiceImpl(HibernateSessionManager sessionManager, Logger logger)
    {
        this.sessionManager = sessionManager;
        this.logger = logger;
    }

    /**
     * Try to find the given user in the local database. Since we are using OpenID that user
     * might not exist in our database yet. If it doesn't, create a new user and store it.
     *
     * WARNING: This implementation will permit EVERY OpenID authenticated user to log in. In
```

```
     * a real environment you want to restrict this to trusted OpenID providers or you have
     * to restrict those users to non-sensible information (by means of roles).
     */
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException,
            DataAccessException
    {
        logger.debug("trying to find user " + username);

        Session session = sessionManager.getSession();

        User u = (User) session.createCriteria(User.class).add(
                Restrictions.eq("username", username)).uniqueResult();

        if (u == null)
        {
            logger.debug("user not found, creating");

            Role r = (Role) session.createCriteria(Role.class).add(
                    Restrictions.eq("authority", "ROLE_USER")).uniqueResult();

            if (r == null)
            {
                logger.debug("role not found, creating");

                r = new Role();

                r.setAuthority("ROLE_USER");

                session.saveOrUpdate(r);
            }

            u = new User();

            u.setUsername(username);

            u.setRoles(new HashSet<Role>());

            u.getRoles().add(r);

            session.saveOrUpdate(u);
        }

        logger.debug("returning user " + u.getUsername());

        sessionManager.commit();

        return u;
    }

}
```

That's it. The rest is setting this all up in your AppModule.

Tying it together

Setting up the OpenID authentication is done in two steps. First we need to add an AuthenticationProvider that provides OpenID authentication and contribute this to the ProviderManager service which is defined by the tapestry-spring-security module. The OpenIDAuthenticationProvider comes directly from Spring security and just has to be set up correctly. Basically it is some kind of wrapper around the UserDetailsService. The following code builds the UserDetailsService and the OpenIDAuthenticationProvider and contributes it to the ProviderManager service:

```
public static UserDetailsService buildUserDetailsService(Logger logger,
        @InjectService("HibernateSessionManager")
        HibernateSessionManager session)
{
    return new UserDetailsServiceImpl(session, logger);
}

public static OpenIDAuthenticationProvider buildOpenIDAuthenticationProvider(
        @InjectService("UserDetailsService")
        UserDetailsService userDetailsService) throws Exception
{
    OpenIDAuthenticationProvider provider = new OpenIDAuthenticationProvider();

    provider.setUserDetailsService(userDetailsService);

    provider.afterPropertiesSet();

    return provider;
}

public static void contributeProviderManager(
        OrderedConfiguration<AuthenticationProvider> configuration,

        @InjectService("OpenIDAuthenticationProvider")
        AuthenticationProvider openIdAuthenticationProvider)
{
    configuration.add("openIDAuthenticationProvider", openIdAuthenticationProvider);
}
```

Next we have to define ourselves a URL to which incoming OpenID authentication requests will be send and which the filter intercepts. Background: Every filter is intercepting requests for it's specific URL. If we just used the URL of an existing filter, our OpenID filter would intercept requests for this filter, thus making authentication fail.

```
public static void contributeApplicationDefaults(
        MappedConfiguration<String, String> configuration)
{
    /* here goes your own configuration... */
    ...

    configuration.add("spring-security.openidcheck.url", "/j_spring_openid_security_check");
}
```

The second part is to configure the filter that intercepts incoming OpenID authentication requests and delegates those to the respective services:

```
public static OpenIDAuthenticationProcessingFilter buildRealOpenIDAuthenticationProcessingFilter(
        @SpringSecurityServices final AuthenticationManager manager,

        @SpringSecurityServices final RememberMeServices rememberMeServices,

        @Inject @Value("${spring-security.openidcheck.url}") final String authUrl,

        @Inject @Value("${spring-security.target.url}") final String targetUrl,

        @Inject @Value("${spring-security.failure.url}") final String failureUrl) throws Exception
{
    OpenIDAuthenticationProcessingFilter filter = new OpenIDAuthenticationProcessingFilter();

    filter.setAuthenticationManager(manager);

    filter.setAuthenticationFailureUrl(failureUrl);

    filter.setDefaultTargetUrl(targetUrl);

    filter.setFilterProcessesUrl(authUrl);

    filter.setRememberMeServices(rememberMeServices);

    filter.afterPropertiesSet();

    return filter;
}

public static HttpServletRequestFilter buildOpenIDAuthenticationProcessingFilter(
        final OpenIDAuthenticationProcessingFilter filter)
{
    return new HttpServletRequestFilterWrapper(filter);
}
```

This filter then has to be contributed to the HttpServletRequestHandler pipeline. The order (before: and after🙂 is very important (don't get this wrong or nothing will work):

```
public static void contributeHttpServletRequestHandler(
        OrderedConfiguration<HttpServletRequestFilter> configuration,

        @InjectService("OpenIDAuthenticationProcessingFilter")
        HttpServletRequestFilter openIDAuthenticationProcessingFilter)
{
    configuration.add(
            "openIDAuthenticationProcessingFilter",

            openIDAuthenticationProcessingFilter,

            "before:springSecurityAuthenticationProcessingFilter",

            "after:springSecurityHttpSessionContextIntegrationFilter");
}
```

And that's it. If you now secure your pages/methods with the @Secured annotation and provide a login page, you should be able to login with your OpenID. For the sake of completeness, here is the Login page and it's template:

```
package org.yourgroup.yourapp.pages;

import org.apache.tapestry5.ioc.annotations.Inject;
import org.apache.tapestry5.ioc.annotations.Value;
import org.apache.tapestry5.services.Request;

/**
 * The login page (adapted from the tapestry-spring-security project).
 *
 * @author Ulrich St&auml;rk
 */
public class LoginPage
{
    @Inject
    @Value("${spring-security.openidcheck.url}")
    private String checkUrl;

    @Inject
    private Request request;

    private boolean failed = false;

    public boolean isFailed()
    {
        return failed;
    }

    public String getLoginCheckUrl()
    {
        return request.getContextPath() + checkUrl;
    }

    void onActivate(String extra)
    {
        if (extra.equals("failed"))
        {
            failed = true;
        }
    }

}
```

LoginPage.tml:

```
<html t:type="layout" xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
<form xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd" action="${loginCheckUrl}" method="POST">
    <t:if test="failed">Username and/or password was wrong!<br /></t:if>
    <label class="username" for="j_username">Username:</label>
    <input class="username" name="j_username" type="text" size="30"/>
    <input id="submit" class="submit" type="submit" value="log in"/>
</form>
</html>
```

An example of how a login page for both form-based and OpenID based login could look like:

LoginPage.java:

```
package org.yourgroup.yourapp.pages;

import org.apache.tapestry5.ioc.annotations.Inject;
import org.apache.tapestry5.ioc.annotations.Value;
import org.apache.tapestry5.services.Request;

/**
 * The login page (adapted from the tapestry-spring-security project).
 *
 * @author Ulrich St&auml;rk
 */
public class LoginPage
{
    @Inject
    @Value("${spring-security.check.url}")
    private String checkUrl;

    @Inject
    @Value("${spring-security.openidcheck.url}")
    private String openidCheckUrl;

    @Inject
    private Request request;

    private boolean failed = false;

    public boolean isFailed()
    {
        return failed;
    }

    public String getLoginCheckUrl()
    {
        return request.getContextPath() + checkUrl;
    }

    public String getOpenIdCheckUrl()
    {
        return request.getContextPath() + openidCheckUrl;
    }

    void onActivate(String extra)
    {
        if (extra.equals("failed"))
        {
            failed = true;
        }
    }
}
```

LoginPage.tml:

```
<html t:type="layout" xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
OpenID: <br />
<form xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd" action="${openIdCheckUrl}" method="POST">
    <t:if test="failed">Username and/or password was wrong!<br /></t:if>
    <label class="username" for="j_username">Username:</label>
    <input class="username" name="j_username" type="text" size="30"/>
    <input id="submit" class="submit" type="submit" value="log in"/>
</form>
Form-based: <br />
<form xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd" action="${loginCheckUrl}" method="POST">
    <t:if test="failed">Username and/or password was wrong!<br /></t:if>
    <label class="username" for="j_username">Username:</label>
    <input class="username" name="j_username" type="text" size="30"/>
    <label class="password" for="j_password">Password</label>
    <input class="password" name="j_password" type="password" size="10" maxlength="30"/>
    <input id="submit" class="submit" type="submit" value="log in"/>
</form>
</html>
```