Spring Security and Wicket-auth-roles

Why use Spring Security (Formerly known as Acegi)?

Spring Security is a pretty complete solution for all kinds of security needs. It is very configurable and supports many authentication sources out of the box. These include: database queries, LDAP queries and CAS. It can be a bit complex to set up, but following the how to below should get you started quickly.

The example below shows how you can use Spring Security in combination with Wicket-auth-roles. Spring Security takes care of the authentication, Wicket-auth-roles does authorization. By this I mean that Spring Security looks up the user (including roles, full name, etc.), validates the password, and keeps track of the current user in the session. Wicket-auth-roles validates whether the current user has access to a particular page, or even a particular component.

The advantage of this setup is that you get to use a lot of Spring Security functionality out of the box. There are the mentioned authentication sources, but you can also add security to services that are called by your Wicket application. You do this by adding an Spring Security proxy to your spring beans. Spring Security proxies allow role based access (e.g. access granted if you have role x) but can also filter the results of a service call (e.g. remove data from a list that must not be seen by current user). See the Spring Security manual for more information.

Why not use Spring Security?

A new Wicket project is currently in the works. You can read more about it on http://wicketstuff.org/confluence/display/STUFFWIKI/Wicket-Security. Please investigate whether it will suit your needs better.

For those still wanting to use Spring Security, there is an howto on getting Swarm working with Spring Security http://wicketstuff.org/confluence/display/STUFFWIKI/Swarm+and+Acegi+HowTo

See the examples below for how to setup your project.

1. Wicket 1.3.5, Spring 2.5.5, Spring Security 2.0.4 on JDK 5.0
2. Wicket 1.2.6, Spring 2.0.5, Acegi 1.0.2 on JDK 1.4

Example Wicket 1.3.5

This example uses Maven 2 for dependency management.

Maven 2 pom.xml

```
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-core</artifactId>
        <version>2.0.4</version>
        <exclusions>
                <exclusion>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-core</artifactId>
                </exclusion>
                <exclusion>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-context</artifactId>
                </exclusion>
                <exclusion>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-aop</artifactId>
                </exclusion>
                <exclusion>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-support</artifactId>
                </exclusion>
                <exclusion>
                        <groupId>commons-logging</groupId>
                        <artifactId>commons-logging</artifactId>
                </exclusion>
        </exclusions>
</dependency>
<dependency>
        <groupId>org.apache.wicket</groupId>
        <artifactId>wicket</artifactId>
        <version>${wicket.version}</version>
</dependency>
<dependency>
        <groupId>org.apache.wicket</groupId>
        <artifactId>wicket-spring</artifactId>
        <version>${wicket.version}</version>
        <exclusions>
                <exclusion>
                        <artifactId>spring</artifactId>
                        <groupId>org.springframework</groupId>
                </exclusion>
        </exclusions>
</dependency>
<dependency>
        <groupId>org.apache.wicket</groupId>
        <artifactId>wicket-spring-annot</artifactId>
        <version>${wicket.version}</version>
</dependency>
<dependency>
        <groupId>org.apache.wicket</groupId>
        <artifactId>wicket-extensions</artifactId>
        <version>${wicket.version}</version>
</dependency>
<dependency>
        <groupId>org.apache.wicket</groupId>
        <artifactId>wicket-auth-roles</artifactId>
        <version>${wicket.version}</version>
</dependency>
```

A nasty thing of Spring security is that is brings in its own Spring version 2.0.8. Since we want 2.5.5 we need to exclude it.

Spring Security setup

Since Spring Security has a very exhaustive documentation available I refer to the manual for in depth information about Spring Security. This example merely shows how to configure Spring Security with Wicket.

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
        version="2.4">

        <filter>
                <filter-name>springSecurityFilterChain</filter-name>
                <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
        </filter>

        <filter>
                <filter-name>wicket.filter</filter-name>
                <filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-class>
        </filter>

        <filter-mapping>
                <filter-name>springSecurityFilterChain</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>

        <filter-mapping>
                <filter-name>wicket.filter</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>
</web-app>
```

It is important to add springSecurityFilterChain mapping higher in code than the Wicket filter mappin. Wicket filter is only passing filter call down by filter chain if it is unable to handle request itself.

Spring security version 3 and wicket 1.4

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
        version="2.4">

        <display-name>example</display-name>


        <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/applicationContext-security.xml</param-value>
        </context-param>

        <listener>
                <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
        </listener>

        <filter>
                <filter-name>wicket.example</filter-name>
                <filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-class>
                <init-param>
                        <param-name>applicationClassName</param-name>
                        <param-value>org.wicket.example.WicketApplication
                        </param-value>
                </init-param>
        </filter>


        <filter-mapping>
                <filter-name>wicket.example</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>

</web-app>
```

Spring 2 context

spring-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:security="http://www.springframework.org/schema/security"
        xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema
/beans/spring-beans-2.5.xsd
       http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-
security-2.0.4.xsd">

        <bean id="myApplication" class="com.foo.bar.MyApplication" />

        <bean id="filterChainProxy" class="org.springframework.security.util.FilterChainProxy">
            <property name="filterInvocationDefinitionSource">
                <value>
                    CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
                    PATTERN_TYPE_APACHE_ANT
                    /**=httpSessionContextIntegrationFilter
                </value>
            </property>
        </bean>

        <bean id="httpSessionContextIntegrationFilter"
              class="org.springframework.security.context.HttpSessionContextIntegrationFilter">
            <property name="allowSessionCreation" value="false"/>
        </bean>

        <security:authentication-provider alias="authenticationManager">
                <security:user-service>
                        <security:user password="admin" name="admin" authorities="ROLE_ADMIN" />
                </security:user-service>
        </security:authentication-provider>

</beans>
```

Spring 3 context

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:security="http://www.springframework.org
/schema/security"
	xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema
/beans/spring-beans-3.0.xsd
	http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-
security-3.0.xsd">


	<security:http create-session="never" auto-config="true" >
		<security:remember-me/>
		<security:intercept-url pattern="/**"/>
	</security:http>


	<security:authentication-manager alias="authenticationManager">
		<security:authentication-provider>

			<!--  TODO change this to reference our real user service -->
			<security:user-service>
				<security:user name="admin" password="admin"
					authorities="ROLE_ADMIN, ROLE_USER" />
				<security:user name="user" password="user"
					authorities="ROLE_USER" />

			</security:user-service>
		</security:authentication-provider>

	</security:authentication-manager>

	<security:global-method-security secured-annotations="enabled" />
</beans>
```

The only filter we need defined from Acegi is the HttpSessionContextIntegrationFilter. This filter will ensure that the SecurityContext is transported to and from the HttpSession onto the Thread context. All authorization is delegated to the wicket-auth-roles module which uses Annotations (@AuthorizeInstantiation).
Using the *authentication-provider* XML element we register an AuthenticationManager in the Spring context. In this case we use a simple in-memory user service using the *user-service* element.

Wicket setup

WebSession

```
public class MyAuthenticatedWebSession extends AuthenticatedWebSession {

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

    @SpringBean(name="authenticationManager")
    private AuthenticationManager authenticationManager;

    public MyAuthenticatedWebSession(Request request) {
        super(request);
        injectDependencies();
        ensureDependenciesNotNull();
    }

    private void ensureDependenciesNotNull() {
        if (authenticationManager == null) {
            throw new IllegalStateException("AdminSession requires an authenticationManager.");
        }
    }

    private void injectDependencies() {
        InjectorHolder.getInjector().inject(this);
    }

    @Override
    public boolean authenticate(String username, String password) {
        boolean authenticated = false;
        try {
            Authentication authentication = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(username, password));
            SecurityContextHolder.getContext().setAuthentication(authentication);
            authenticated = authentication.isAuthenticated();
        } catch (AuthenticationException e) {
            logger.warn(format("User '%s' failed to login. Reason: %s", username, e.getMessage()));
            authenticated = false;
        }
        return authenticated;
    }

    @Override
    public Roles getRoles() {
        Roles roles = new Roles();
        getRolesIfSignedIn(roles);
        return roles;
    }

    private void getRolesIfSignedIn(Roles roles) {
        if (isSignedIn()) {
            Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
            addRolesFromAuthentication(roles, authentication);
        }
    }

    private void addRolesFromAuthentication(Roles roles, Authentication authentication) {
        for (GrantedAuthority authority : authentication.getAuthorities()) {
            roles.add(authority.getAuthority());
        }
    }

}
```

Spring 2 Application

MyWebApplication.java

```
public class MyWebApplication extends AuthenticatedWebApplication implements ApplicationContextAware {
    private ApplicationContext context;

    boolean isInitialized = false;

    @Override
    protected void init() {
        if (!isInitialized) {
            super.init();
            setListeners();
            isInitialized = true;
        }
    }

    private void setListeners() {
        addComponentInstantiationListener(new SpringComponentInjector(this, context));
    }

    @Override
    public Class<?> getHomePage() {
        return HomePage.class;
    }

    public void setApplicationContext(ApplicationContext context) throws BeansException {
        this.context = context;
    }

    @Override
    protected Class<? extends WebPage> getSignInPageClass() {
        return LoginPage.class;
    }

    @Override
    protected Class<? extends AuthenticatedWebSession> getWebSessionClass() {
        return MyAuthenticatedWebSession.class;
    }
}
```

Spring 3 Application

MyWebApplicationSpring3.java

```
public class MyWebApplicationSpring3 extends AuthenticatedWebApplication {

    boolean isInitialized = false;

    @Override
    protected void init() {
        if (!isInitialized) {
            super.init();
            setListeners();
            isInitialized = true;
        }
    }

    private void setListeners() {
        addComponentInstantiationListener(new SpringComponentInjector(this));
    }

    @Override
    public Class<?> getHomePage() {
        return HomePage.class;
    }

    @Override
    protected Class<? extends WebPage> getSignInPageClass() {
        return LoginPage.class;
    }

    @Override
    protected Class<? extends AuthenticatedWebSession> getWebSessionClass() {
        return MyAuthenticatedWebSession.class;
    }
}
```

LoginForm.java

```
class LoginForm extends Form {

    private String username;

    private String password;

    public LoginForm(String id) {
        super(id);
        setModel(new CompoundPropertyModel(this));
        add(new RequiredTextField("username"));
        add(new PasswordTextField("password"));
    }

    @Override
    protected void onSubmit() {
        AuthenticatedWebSession session = AuthenticatedWebSession.get();
        if(session.signIn(username, password)) {
            setDefaultResponsePageIfNecessary();
        } else {
            error(getString("login.failed"));
        }
    }

    private void setDefaultResponsePageIfNecessary() {
        if(!continueToOriginalDestination()) {
            setResponsePage(getApplication().getHomePage());
        }
    }

}
```

SecuredPage.java

```
@AuthorizeInstantiation("ROLE_ADMIN")
public class SecuredPage extends WebPage {
    //omitted, since not relevant
}
```

As stated before the above code examples show the very minimal and a basic setup of how to configure Spring Security with Wicket. For more advanced configuration regarding *authentication-provider* mechanisms I refer to Spring Security manual.

Example Wicket 1.2.6

This example is extracted from a production system running on Wicket 1.2.6, Spring 2.0.5 and Acegi 1.0.2 on a 1.4 JVM. Wicket-auth-roles requires Java 5, but is quite simple to port it to Java 1.4 by removing everything related to annotations. If you do not want to port Wicket-auth-roles, this example requires that you use Java 5.

Classpath

The code in this HOWTO works with the following jars. Newer and older versions probably work, but you'll have to try to find out.

- Wicket 1.2.6
- Wicket-auth-roles 1.2.6
- Acegi 1.0.2
- Spring, including spring-support 2.0.5
- Ehcache 1.2.3

In addition, you'll need to inject some Spring beans into your wicket classes. This example assumes (but does not show) you are injecting these into the application. You'll need more jars for this. See here for more information on integrating Wicket with Spring.

Acegi basic setup

First of all you need to set up Acegi. Somewhere in your Spring configs add:

```
<!-- Proxy to a set of filters that enforce authentication and authorization. -->
  <bean id="filterChainProxy" class="org.acegisecurity.util.FilterChainProxy">
    <property name="filterInvocationDefinitionSource">
      <value>
        CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
        PATTERN_TYPE_APACHE_ANT
        /**=httpSessionContextIntegrationFilter
      </value>
    </property>
  </bean>

  <!-- Maintains security context between requests (using the session). -->
  <bean id="httpSessionContextIntegrationFilter"
    class="org.acegisecurity.context.HttpSessionContextIntegrationFilter">
    <property name="forceEagerSessionCreation" value="true"/>
  </bean>

   <!-- Users cache for Acegi (Ehcache). -->
   <bean id="userCache" class="org.acegisecurity.providers.dao.cache.EhCacheBasedUserCache">
      <property name="cache">
         <bean class="org.springframework.cache.ehcache.EhCacheFactoryBean">
            <property name="cacheManager" ref="cacheManager"/>
            <property name="cacheName" value="your.application.name.USER_CACHE"/>
         </bean>
      </property>
   </bean>

   <bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"/>
```

This will configure the following:

- Acegi will keep track of the current user for every page in your application.
- It will do so by storing that user in the session. During a request, the user is available through a ThreadLocal (accessed through the class `AuthenticationStore`).
- User information is stored in a cache. This is very important for performance when you use LDAP or database authentication source. This particular configuration uses a cache from EhCache.

Note that in most types of web applications you would configure Acegi to work on some URLs only, and do authorization from Acegi. However, in Wicket URL have no obvious relation to the content. Instead we let Acegi just track the user on all URLs (authentication) and postphone access checks (authorization) to within the Wicket application.

Add the following to your web.xml:

```
<filter>
        <filter-name>Acegi HTTP Request Security Filter</filter-name>
        <filter-class>org.acegisecurity.util.FilterToBeanProxy</filter-class>
        <init-param>
            <param-name>targetClass</param-name>
            <param-value>org.acegisecurity.util.FilterChainProxy</param-value>
        </init-param>
    </filter>

    <filter-mapping>
        <filter-name>Acegi HTTP Request Security Filter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

Please mind that it is important to put the Acegi-filter-mapping before the wicket-filter-mapping. Otherwise the wicket-authorization would be dependend on values that haven't been set yet.

You now have setup Acegi to put a security token in the 'security context' (which comes down to a thread local variable) during the invocation of each and every request, based on some information in the session. At the end of the request, the security token is removed from the security context.

You will also have to configure an authentication provider. Here is an example that get its data from LDAP.

If Acegi's `org.acegisecurity.userdetails.UserDetails` does not have enough information, you can extend it with your own. Here we add a full name and an e-mail address.

```
public interface YourAppUserDetails extends UserDetails {
    String getEmail();
    String getDisplayname();
}
```

An implementation:

```
public class YourAppUserDetailsLdapImpl implements YourAppUserDetails, org.acegisecurity.userdetails.ldap.
LdapUserDetails {

    private static final String LDAP_ATTRIBUTENAME_MAIL = "mail";
    private static final String LDAP_ATTRIBUTENAME_DISPLAYNAME = "displayname";

    /** The wrapped <code>LdapUserDetails</code> instance. */
    private final LdapUserDetails ldapUserDetails;

    private String email;
    private String displayname;

    /**
     * Wrap the given Acegi <code>LdapUserDetails</code> instance. This class adds the additional properties
     * <code>email</code> and <code>displayname</code> that are fetched from the <code>LdapUserDetails</code>
     * {@link org.acegisecurity.userdetails.ldap.LdapUserDetails#getAttributes() attributes}.
     *
     * Note: an e-mail address is mandatory, full name is not.
     *
     * @param ldapUserDetails the wrapped user details instance
     */
    public YourAppUserDetailsLdapImpl(final LdapUserDetails ldapUserDetails) {
        this.ldapUserDetails = ldapUserDetails;
        Attributes attributes = this.ldapUserDetails.getAttributes();

        // Fetch e-mail address from attributes (required, exception is thrown if not available).
        try {
            Attribute mailAttribute = attributes.get(LDAP_ATTRIBUTENAME_MAIL);
            email = (String) (mailAttribute == null ? null : mailAttribute.get());
            if (email == null) {
                String errorMessage = "No attribute named '" + LDAP_ATTRIBUTENAME_MAIL
                        + "' found for user '" + ldapUserDetails.getUsername() + "'.";
                throw new RuntimeException(errorMessage);
            }

        } catch (NamingException e) {
            String errorMessage = "NamingException while attempting to retrieve value for attribute '"
                    + LDAP_ATTRIBUTENAME_MAIL + "' for user '" + ldapUserDetails.getUsername() + "'.";
            throw new RuntimeException(errorMessage, e);
        }

        // Get Display name
        try {
            Attribute displaynameAttribute = attributes.get(LDAP_ATTRIBUTENAME_DISPLAYNAME);
            displayname = (String) (displaynameAttribute == null ? null : displaynameAttribute.get());

        } catch (NamingException e) {
            LOG.warn("NamingException while attempting to retrieve value for attribute '"
                    + LDAP_ATTRIBUTENAME_DISPLAYNAME
                    + "' for user '" + ldapUserDetails.getUsername() + "'. Setting displayname to null.");
        }
    }

    /** {@inheritDoc} */
    public String getEmail() {
        return email;
    }

    /** {@inheritDoc} */
```

```
    public String getDisplayname() {
        return displayname;
    }

    /**
     * @return this user's authorities (i.e. roles)
     */
    public GrantedAuthority[] getAuthorities() {
        return ldapUserDetails.getAuthorities();
    }

    /**
     * @return this user's password
     */
    public String getPassword() {
        return ldapUserDetails.getPassword();
    }

    /**
     * @return this user's user name
     */
    public String getUsername() {
        return ldapUserDetails.getUsername();
    }

    /** {@inheritDoc} */
    public boolean isAccountNonExpired() {
        return ldapUserDetails.isAccountNonExpired();
    }

    /** {@inheritDoc} */
    public boolean isAccountNonLocked() {
        return ldapUserDetails.isAccountNonLocked();
    }

    /** {@inheritDoc} */
    public boolean isCredentialsNonExpired() {
        return ldapUserDetails.isCredentialsNonExpired();
    }

    /** {@inheritDoc} */
    public boolean isEnabled() {
        return ldapUserDetails.isEnabled();
    }

    /**
     * @return this user's LDAP attributes
     */
    public Attributes getAttributes() {
        return ldapUserDetails.getAttributes();
    }

    /**
     * Returns any LDAP response controls that were part of the user authentication process. See
     * <a href="ftp://ftp.isi.edu/in-notes/rfc2251.txt" >RFC 2251</a> for a description of controls.
     * @return LDAP response controls
     */
    public Control[] getControls() {
        return ldapUserDetails.getControls();
    }

    /**
     * @return this user's distinguished name
     */
    public String getDn() {
        return ldapUserDetails.getDn();
    }
}
```

Its a bit messy, improvements are welcome.

We have to write our own LDAP authenticatoin provider as Acegi does not know about the e-mail address and full name:

```
import org.acegisecurity.providers.ldap.LdapAuthenticationProvider;
import org.acegisecurity.providers.ldap.LdapAuthenticator;
import org.acegisecurity.providers.ldap.LdapAuthoritiesPopulator;

public class YourAppLdapAuthenticationProvider extends LdapAuthenticationProvider {
    public YourAppLdapAuthenticationProvider(LdapAuthenticator authenticator, LdapAuthoritiesPopulator
authoritiesPopulator) {
        super(authenticator, authoritiesPopulator);
    }

    /**
     * Creates a <code>UserDetails</code> instance ({@link YourAppUserDetails}) that provides the additional
properties
     * <code>email</code> and <code>displayname</code>.
     */
    protected UserDetails createUserDetails(LdapUserDetails ldapUser, String username, String password) {
        UserDetails userDetails = super.createUserDetails(ldapUser, username, password);
        return new YourAppUserDetailsLdapImpl((LdapUserDetails) userDetails);
    }
}
```

Finally the configuration of the LDAP authenticatoin provider. Again this is done in a Spring config file:

```
<!-- Authentication manager, configured with one provider that retrieves authentication information
        from an LDAP instance. -->
    <bean id="authenticationManager" class="org.acegisecurity.providers.ProviderManager">
        <property name="providers">
            <list>
                <ref local="ldapAuthenticationProvider"/>
            </list>
        </property>
    </bean>

    <!-- Example query against Active Directory, uses sAMAccountName as username -->
    <bean id="userSearch" class="org.acegisecurity.ldap.search.FilterBasedLdapUserSearch">
        <constructor-arg index="0" value="ou=users,${__ldap.basedn}" />
        <constructor-arg index="1" value="(&amp;(objectclass=person)(sAMAccountName={0}))" />
        <constructor-arg index="2" ref="initialDirContextFactory" />
        <property name="searchSubtree" value="false" />
    </bean>

    <!-- Authentication provider for authentication via LDAP. -->
    <bean id="ldapAuthenticationProvider" class="com.example.app.security.YourAppLdapAuthenticationProvider">
        <constructor-arg>
            <bean class="org.acegisecurity.providers.ldap.authenticator.BindAuthenticator">
                <constructor-arg ref="initialDirContextFactory"/>
                <property name="userSearch" ref="userSearch" />
            </bean>
        </constructor-arg>
        <constructor-arg>
            <bean class="org.acegisecurity.providers.ldap.populator.DefaultLdapAuthoritiesPopulator">
                <constructor-arg ref="initialDirContextFactory"/>
                <constructor-arg>
                    <value>ou=groups,${__ldap.basedn}</value>
                </constructor-arg>
                <property name="groupSearchFilter" value="member={0}"/>
            </bean>
        </constructor-arg>
    </bean>

    <!-- Initial context factory for JNDI queries to LDAP server. -->
    <bean id="initialDirContextFactory" class="org.acegisecurity.ldap.DefaultInitialDirContextFactory">
        <constructor-arg value="ldap://${__ldap.host}:${__ldap.port}/"/>
        <property name="managerDn" value="${__ldap.manager.cn}"/>
        <property name="managerPassword" value="${__ldap.manager.pass}"/>
    </bean>

    <!-- Read LDAP properties from a file. -->
    <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="placeholderPrefix" value="${__"/>
        <property name="location" value="classpath:ldap.properties"/>
    </bean>
```

With ldap.properties something like:

```
ldap.host=192.168.20.123
ldap.port=380
ldap.basedn=ou=Application,ou=Organisation,dc=example,dc=com
ldap.manager.cn=cn=manager,ou=users,ou=Application,ou=Organisation,dc=example,dc=com
ldap.manager.pass=secret
```

⚠ TODO: describe how roles are structured in LDAP tree.

Wicket setup

Your application's session should extend `wicket.authentication.AuthenticatedWebSession` from Wicket-auth-roles instead of Wicket's WebSession.

Notice that YourAppSession below functions as the bridge between Wicket and Acegi. There are 2 methodes that we will later use for signin/signout: `authenticate` and `signout`. Method `getUser` is useful when you want to display the details of the user. Method `getRoles` is used by Wicket-auth-roles to get the roles of the current user.

```
import org.acegisecurity.*;
import org.acegisecurity.context.SecurityContextHolder;
import org.acegisecurity.providers.UsernamePasswordAuthenticationToken;

public class YourAppSession extends AuthenticatedWebSession {
    public YourAppSession(final AuthenticatedWebApplication application) {
        super(application);
    }

    /**
     * Attempts to authenticate a user that has provided the given username and password.
     * @param username current username
     * @param password current password
     * @return <code>true</code> if authentication succeeds, <code>false</code> otherwise
     */
    public boolean authenticate(String username, String password) {
        String u = username == null ? "" : username;
        String p = password == null ? "" : password;

        // Create an Acegi authentication request.
        UsernamePasswordAuthenticationToken authRequest = new UsernamePasswordAuthenticationToken(u, p);

        // Attempt authentication.
        try {
            AuthenticationManager authenticationManager =
                ((YourAppApplication) getApplication()).getAuthenticationManager();
            Authentication authResult = authenticationManager.authenticate(authRequest);
            setAuthentication(authResult);

            LOG.info("Login by user '" + username + "'.");
            return true;

        } catch (BadCredentialsException e) {
            LOG.info("Failed login by user '" + username + "'.");
            setAuthentication(null);
            return false;

        } catch (AuthenticationException e) {
            LOG.error("Could not authenticate a user", e);
            setAuthentication(null);
            throw e;

        } catch (RuntimeException e) {
            LOG.error("Unexpected exception while authenticating a user", e);
            setAuthentication(null);
            throw e;
        }
    }

    /**
     * @return the currently logged in user, or null when no user is logged in
     */
    public YourAppUserDetails getUser() {
        YourAppUserDetails user = null;
        if (isSignedIn()) {
            Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
            user = (YourAppUserDetails) authentication.getPrincipal();
        }
        return user;
    }

    /**
     * Returns the current user roles.
     * @return current user roles
     */
    public Roles getRoles() {
```

```
        if (isSignedIn()) {
            Roles roles = new Roles();
            // Retrieve the granted authorities from the current authentication. These correspond one on
            // one with user roles.
            GrantedAuthority[] authorities = SecurityContextHolder.getContext().getAuthentication().
getAuthorities();
            for (int i = 0; i < authorities.length; i++) {
                GrantedAuthority authority = authorities[i];
                roles.add(authority.getAuthority());
            }
            return roles;
        }
        return null;
    }

    /**
     * Signout, invalidates the session. After a signout, you should redirect the browser to the home page.
     */
    public void signout() {
        YourAppUserDetails user = getUser();
        if (user != null) {
            LOG.info("Logout by user '" + user.getUsername() + "'.");
        }
        setAuthentication(null);
        invalidate();
    }

    /**
     * Sets the acegi authentication.
     * @param authentication the authentication or null to clear
     */
    private void setAuthentication(Authentication authentication) {
        SecurityContextHolder.getContext().setAuthentication(authentication);
    }

    /**
     * @return the current YourApp session
     */
    public static YourAppSession getYourAppSession() {
        return (YourAppSession) Session.get();
    }

}
```

In order to weave the extended session into your Wicket application, your application should extend Wicket-auth-roles's Authenticated AuthenticatedWebApplication. (See method `getWebSessionClass` below.) The other function of the appliction instance is to configure which components need authentication. There are 2 ways to do so. This example runs on Java 1.4 and therefore you can only use the {MetaDataRoleAuthorizationStrategy}}. If you are running on Java 5 or higher, you can also use the annotations based approach. Look at the source code in Wicket-auth-roles-example for inspiration.

Notice that the authentication manager needs to be injected (by Spring).

When no user is currently signed in, and an authorised component (page) is accessed, Wicket-auth-roles will redirect to the page that is given by method g `etSignInPageClass`. We will see the implementation of that page later.

```
import org.acegisecurity.AuthenticationManager;
import wicket.authentication.AuthenticatedWebApplication;
import wicket.authorization.strategies.role.metadata.MetaDataRoleAuthorizationStrategy;

public class YourAppApplication extends AuthenticatedWebApplication {
    // To be injected by Spring
    private AuthenticationManager authenticationManager;

    protected void init() {
        super.init();

        // ... other settings ...

        // Security settings.
        getSecuritySettings().setAuthorizationStrategy(new MetaDataRoleAuthorizationStrategy(this));

        // List every(!) page and component here for which access is forbidden unless
        // the current user has the correct role.
        MetaDataRoleAuthorizationStrategy.authorize(EditPage.class, SecurityConstants.ROLE_EDITOR);
        MetaDataRoleAuthorizationStrategy.authorize(ManagerPage.class, SecurityConstants.ROLE_MANAGER);
    }

    protected Class getWebSessionClass() {
        return YourAppSession.class;
    }

    protected Class getSignInPageClass() {
        return YourAppSignIn.class;
    }

    public AuthenticationManager getAuthenticationManager() {
        return authenticationManager;
    }

    // To be injected by Spring
    public void setAuthenticationManager(final AuthenticationManager authenticationManager) {
        this.authenticationManager = authenticationManager;
    }

}
```

As you can see in method `init` we need to set authorization rules for each component (page) we want to restrict access to. Note that if you are using inheritance, you still need to list every sub-class. The application that this information is extracted from uses a hack to circumvent that. It overrides `wicket.authorization.strategies.role.metadata.InstantiationPermissions` to use the authorisation meta-data of the first higher base-class that has them. See Wicket-XXX for source code.

What now follows is the signin page. In the constructor you can see there is a check whether a user is already signed in. If that is the case, an error is added to the feedback panel.

In the form submit you'll see that `YourAppSession#authenticat(String,String)` is called. Upon success, the user is redirected to the original page the user was trying to see. If the signin page was requested directly there is no such original page, and the user is redirected to the 'home page'. Note that a redirect is always necessary. This will remove the signin page from the browser history, and with it the used password.

```
public class YourAppSignIn extends WebPage {
    public YourAppSignIn() {
        this(null);
    }

    /**
     * Creates a new sign-in page with the given parameters (ignored).
     * @param parameters page parameters (ignored)
     */
    public YourAppSignIn(final PageParameters parameters) {
        add(new SignInForm("loginform", new SimpleUser()));

        YourAppSession YourAppSession = getYourAppSession();
        if (YourAppSession.isSignedIn()) {
            error(getLocalizer().getString("login.errors.alreadysignedin", YourAppSignIn.this));
        }
```

```
    }

    /**
     * The class <code>SignInForm</code> is a subclass of the Wicket
     * {@link Form} class that attempts to authenticate the login request using
     * Wicket auth (which again delegates to Acegi Security).
     */
    public final class SignInForm extends Form {
        public SignInForm(String id, IModel model) {
            super(id, new CompoundPropertyModel(model));
            add(new FeedbackPanel("feedback"));
            add(new TextField("username").setRequired(true));
            add(new PasswordTextField("password").setResetPassword(true));
        }

        /**
         * Called upon form submit. Attempts to authenticate the user.
         */
        protected void onSubmit() {
            if (getYourAppSession().isSignedIn()) {
                // Already logged in, ignore the submit.
                error(getLocalizer().getString("login.errors.alreadysignedin", YourAppSignIn.this));

            } else {
                SimpleUser user = ((SimpleUser) getModel().getObject(null));
                String username = user.getUsername();
                String password = user.getPassword();

                // Attempt to authenticate.
                YourAppSession session = (YourAppSession) Session.get();
                if (session.signIn(username, password)) {

                    Roles roles = session.getRoles();
                    if (!continueToOriginalDestination()) {
                        // Continue to the application home page.
                        setResponsePage(getApplication().getHomePage());

                        if (LOG.isDebugEnabled()) {
                            LOG.debug("User '" + username + "' directed to application"
                                + " homepage (" + getApplication().getHomePage().getName() + ").");
                        }

                    } else {
                        if (LOG.isDebugEnabled()) {
                            LOG.debug("User '" + username + "' continues to original destination.");
                        }
                    }

                } else {
                    LOG.info("Could not authenticate user '" + username + "'. Transferring back to sign-in
page.");
                    error(getLocalizer().getString("login.errors.invalidCredentials", YourAppSignIn.this));
                }
            }

            // ALWAYS do a redirect, no matter where we are going to. The point is that the
            // submitting page should be gone from the browsers history.
            setRedirect(true);
        }
    }

    /**
     * Simple bean that represents the properties for a login attempt (username
     * and clear text password).
     */
    public static class SimpleUser implements Serializable {
        private static final long serialVersionUID = -5617176504597041829L;

        private String username;
        private String password;
```

```
        public String getUsername() { return username; }
        public void setUsername(String username) { this.username = username; }
        public String getPassword() { return password; }
        public void setPassword(String password) { this.password = password; }
    }
}
```

And here is the template (`YourAppSignIn.html`). It is a bit rough as I removed all non-essentials.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:wicket="http://wicket.sourceforge.net/" xml:lang="us_EN" lang="
us_EN">
<body>
    <form wicket:id="loginform">
         <div wicket:id="feedback"></div>
         <div>Username <input wicket:id="username" type="text"/></div>
         <div>Password <input wicket:id="password" type="password"/></div>
         <div><input type="submit" value="Signin"/></div>
    </form>
</body>
</html>
```

The error is read from the properties file `YourAppSignIn.properties`.

```
login.errors.alreadysignedin=You are already singed in, please singout if you which to sign as another user.
login.errors.invalidCredentials=Invalid user name and/or password.
```

Hiding components

We have now seen how to completely disallow access to a page (or any component) and redirect to the signin page as soon as it is accessed. I will now show how to hide components on a page depending on the roles of the user.

> ⚠ TODO: finish this section with a menu as example

Questions?

Please send questions to the wicket user mailing list.