Glassfish 3 – Custom Realm amb eXist DB

1. Introducció

Glassfish ens permet utilitzar autenticació per regnes utilitzant diversos mètodes: Bases de Dades, fitxers, LDAP... I amb una mica de feina també amb eXist.
Un dels problemes que un es troba quan s'utilitza Glassfish conjuntament amb eXist és que Glassfish dona suport a bases de dades relacionals amb SQL i no jeràrquiques com eXist. Per tant, caldrà programar el que s'anomena “Custom Realm”, que no és més que una re-implementació de les classes que utilitza el servidor Glassfish per autentificar els usuaris.
En aquest manual expliquem com programar les classes necessàries per a la autenticació, però no ens parem a explicar les qüestions que tenen a veure amb la configuració del servidor o del projecte; al annex hi podeu trobar una còpia d'un article extret de l'espai web d'Oracle on s'explica el que fa falta.
Com es pot veure l'article, per a la autenticació dels usuaris es pot fer ús, a banda del nom i la clau, del grup de l'usuari. En l'exemple que comentem no farem ús d'aquest atribut i ens limitarem a considerar que tots els usuaris son del grup “user”.

2. Implementació del Custom Realm

2.1. ExistController

Primer de tot, cal crear una classe que serà la responsable de connectar-se a eXist i realitzar consultes necessàries per l'autenticació i l'existència d'un usuari qualsevol. Aquestes operacions són les que demana principalment Glassfish per fer el login. S'ha de dir que tot això depén directament de la informació que tenim dels usuaris a la nostra base de dades eXist.

```
package com.eXistRealm;

import javax.xml.transform.OutputKeys;

import org.apache.xmlrpc.XmlRpcClient;
import org.exist.xmldb.XQueryService;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.base.Collection;
import org.xmldb.api.base.CompiledExpression;
import org.xmldb.api.base.Database;
import org.xmldb.api.base.Resource;
import org.xmldb.api.base.ResourceIterator;
import org.xmldb.api.base.ResourceSet;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.modules.XPathQueryService;

public class ExistController {
 protected static String URI =
   "xmldb:exist://localhost:9090/exist/xmlrpc/db/eXistRealm";
```

```
public static Collection col;
public static XPathQueryService service;
public static XQueryService service2;
public static XmlRpcClient xmlrpc;

public void connect() throws Exception {
	String driver = "org.exist.xmldb.DatabaseImpl";
	// initialize database driver
	Class<?> cl = null;
	try {
		cl = Class.forName(driver);
	} catch (Exception e) {
		throw new Exception("Error en inicialitzacio del driver1");
	}
	Database database = null;
	try {
		database = (Database) cl.newInstance();
	} catch (InstantiationException e) {
		throw new Exception("Error en inicialitzacio del driver2");
	} catch (IllegalAccessException e) {
		throw new Exception("Error en inicialitzacio del driver3");
	}
	try {
		DatabaseManager.registerDatabase(database);
		database.setProperty("create-database", "true");
	} catch (XMLDBException e) {
		throw new Exception("Error en inicialitzacio del driver4");
	}
	try {
		col = DatabaseManager.getCollection(URI,"admin","password");
		col.setProperty(OutputKeys.INDENT, "no");
		service = (XPathQueryService)
		col.getService("XPathQueryService","1.0");
		service.setProperty("pretty", "true");
		service.setProperty("encoding", "8859-1");
		service.setProperty("indent", "yes");
		service2=(XQueryService) col.getService("XQueryService", "1.0");
		service2.setProperty("indent", "yes");
		service2.setProperty("pretty", "true");
		service2.setProperty("encoding", "8859-1");
	} catch (XMLDBException e) {
		throw new Exception("Error en XMLDB");
	}
}
```

```
    //Fa una query del tipus que sigui a la BD
    public String doQuery(String query,boolean XQuery) throws Exception {
        String res = "";
        try {
            if (!XQuery) {
                ResourceSet result = service.query(query);
                ResourceIterator i;
                i = result.getIterator();
                while (i.hasMoreResources()) {
                    Resource r = i.nextResource();
                    res = res.concat((String) r.getContent());
                }
            }
            else {
                CompiledExpression compiled = service2.compile(query);
                ResourceSet result = service2.execute(compiled);
                ResourceIterator i = result.getIterator();
                while(i.hasMoreResources()) {
                    Resource r = i.nextResource();
                    res = res.concat((String) r.getContent());
                }
            }
        } catch (Exception e) {
                throw new Exception("Query Error");
        }
        return res;
    }
}
```

Cal remarcar tres atributs d'aquesta classe que s'han de configurar en cada projecte (remarcats amb negreta). L'atribut URI, que indica la col·lecció a on ens volem connectar, un exemple pot ser: *xmldb:exist://localhost:8088/xmlrpc/db/eXistRealm.* Amb aquesta direcció, ens connectarem a eXist al port 8088, i a la col·lecció eXistRealm. Pel que fa als altres dos seran el nom d'usuari i contrasenya que donen accés al contingut a la nostra col·lecció. Tenim, doncs, una classe que ens connecta a la base de dades i una operació que ens realitza una consulta a la base de dades i que ens retorna el resultat.

2.2. ExistRealm

La segona classe que hem de programar és la que permet als servidor obtenir la informació necessària dels usuaris i grups. Aquesta classe ha de fer un “extend” de *com.sun.appserv.security.AppservRealm*, i ha d' mplementar els mètodes

- public void init (Properties properties) throws BadRealmException, NoSuchRealmException
- *public String getAuthType()* - This method returns a descriptive string representing the type of authentication done by this realm.
- *public Enumeration getGroupNames(String user) throws InvalidOperationException, NoSuchUserException* - This method returns the group names the user belongs to as an Enumeration of Strings.

Podeu veure el codi a continuació:

```
package com.eXistRealm;

import com.sun.appserv.security.AppservRealm;
import com.sun.enterprise.security.auth.realm.BadRealmException;
import com.sun.enterprise.security.auth.realm.InvalidOperationException;
import com.sun.enterprise.security.auth.realm.NoSuchRealmException;
import com.sun.enterprise.security.auth.realm.NoSuchUserException;
import java.util.Enumeration;
import java.util.Properties;
import java.util.ArrayList;
import java.util.Collections;

/**
 * eXistRealm - class extending AppservRealm
 * Sample Custom Realm Class that stores user-group information
 */

public class EXistRealm extends AppservRealm {

   private static final String PARAM_JAAS_CONTEXT = "jaas-context";

   public EXistRealm() { }

   /**
    * Initialization - set the jaas-context property,
    * Set UserA to devGroup and user B to testGroup
    * @param properties - Key-Value pairs defined in the Realm
    * @throws com.sun.enterprise.security.auth.realm.BadRealmException
    * @throws com.sun.enterprise.security.auth.realm.NoSuchRealmException
    */
   public void init(Properties properties)
      throws BadRealmException, NoSuchRealmException {
      log("Init eXistRealm");

      String propJaasContext = properties.getProperty(PARAM_JAAS_CONTEXT);
      if ( propJaasContext != null ) {
         setProperty(PARAM_JAAS_CONTEXT, propJaasContext);
      }
      // A partir d'aqui inicialitzacio (si cal)
   }

   /**
    * @return authType
    */
   public String getAuthType() {
```

```
    return "eXist Realm";
  }

  /**
   * @param user
   * @return Enumeration - List of groups to which user belongs
   * @throws com.sun.enterprise.security.auth.realm.InvalidOperationException
   * @throws com.sun.enterprise.security.auth.realm.NoSuchUserException
   */
  public Enumeration getGroupNames(String user)
    throws InvalidOperationException, NoSuchUserException {
    ArrayList<String> array = new ArrayList<String>();

    if (!existeix(user)) throw new NoSuchUserException("No existeix l'usuari");

    array.add("user");

    // hem de retornar un Enumeraton
    Enumeration e = Collections.enumeration(array);
    return e;
  }

  public boolean existeix(String usermail) {
        String result;
        ExistController exis= new ExistController();
        try {
                exis.connect();
                result = exis.doQuery("/usuaris/usuari[usermail=\""
                        + usermail + "\"]/usermail/text()",false);
                return !result.isEmpty();
        } catch (Exception e) {
                return true;
        }
  }

  private void log(String s) {
    System.out.println((new StringBuilder()).append("eXistRealm::").
          append(s).toString());
  }

}
```

2.3. eXistRealmLoginModule

Per acabar, hem d'implementar la classe que utilitza el servidor per a realitzar la autenticació dels usuaris. Aquesta ha de fer un “extend” de la classe *AppservPasswordLoginModule* i ha d'implementar el mètode

- abstract protected void authenticateUser() throws LoginException.

Aquest mètode disposa de les variables *_username* i *_password* (heretades del servidor) per a poder comprovar si l'usuari té accés o no al regne i per obtenir-ne el grup. En cas que el nom d'usuari o la clau siguin incorrectes s'ha de llançar la excepció pertinent.

Podeu veure el codi a continuació:

```
package com.eXistRealm;

import com.sun.appserv.security.AppservPasswordLoginModule;
import com.sun.enterprise.security.auth.realm.InvalidOperationException;
import com.sun.enterprise.security.auth.realm.NoSuchUserException;
import java.util.Enumeration;
import javax.security.auth.login.LoginException;
import java.util.ArrayList;

/**
 * eXistRealmLoginModule - extends AppservPasswordLoginModule
 * Sample class to be referenced by the eXistRealm class . Overrides the
 * authenticateUser() method that authenticates the user using
 * user-password information and calls commitUserAuthentication
 */

public class eXistRealmLoginModule extends AppservPasswordLoginModule {

    public eXistRealmLoginModule() {
        log("eXistRealmLoginModule: Initialization");
        // inicialitzacio si fa falta
    }

    /**
     * Overrides the authenticateUser() method in AppservPasswordLoginModule
     * Performs authentication of user
     * @throws javax.security.auth.login.LoginException
     */
  protected void authenticateUser() throws LoginException {
        log((new StringBuilder()).append("eXistRealm Auth Info:_username:")
                .append(_username).append(";_password:").append(_password)
                    .append(";_currentrealm:").append(_currentRealm).toString());

        //Check if the given realm is eXistRealm
        if (!(_currentRealm instanceof EXistRealm)) {
            throw new LoginException("Realm not eXistRealm");
        }

        //Authenticate User
        EXistRealmRealm samplerealm = (EXistRealm)_currentRealm;
        if (!authenticate(_username, _password)) {
```

```
        //Login fails
        throw new LoginException((new StringBuilder())
                .append("eXistRealm:Login Failed for user  ")
                    .append(_username).toString());
    }

    //Login succeeds
    log((new StringBuilder()).append("eXistRealm:login succeeded for  ")
        .append(_username).toString());

    //Get group names for the authenticated user from the Realm class
    Enumeration enumeration = null;

    try {
        enumeration = samplerealm.getGroupNames(_username);
    }
    catch(InvalidOperationException invalidoperationexception) {
        throw new LoginException((new StringBuilder())
                .append("An InvalidOperationException was thrown " +
                "   while calling getGroupNames() on the eXistRealm ")
                    .append(invalidoperationexception).toString());
    }
    catch(NoSuchUserException nosuchuserexception) {
        throw new LoginException((new StringBuilder())
                .append("A NoSuchUserException was thrown " +
                "   while calling getGroupNames() on the eXistRealm ")
                    .append(nosuchuserexception).toString());
    }

    ArrayList<String> array = new ArrayList<String>();
    for(int i = 0; enumeration != null && enumeration.hasMoreElements(); i++)
        array.add((String)enumeration.nextElement());

    String authenticatedGroups[] = new String[array.size()];
    for(int i = 0; i < array.size(); i++)
        authenticatedGroups[i] = array.get(i);

    //Call commitUserAuthentication with the groupNames the user belongs to
    commitUserAuthentication(authenticatedGroups);
}

/**
 * Private method to authenticate user from eXist
 */
private boolean authenticate (String usermail, String pass) {
        String result;
        MD5Hash h = new MD5Hash();
```

```
        ExistController exis= new ExistController();
        try {
                exis.connect();
                result=exis.doQuery("for $b in doc(\"usuaris\")//usuari " +
                                "where $b/usermail = \""+usermail+"\" "+
                                "return $b/password/text()",true);
                if (result.isEmpty()) return false;
                if (pass.isEmpty()) return false;
                return h.isEqual(pass, true, result, false);
        } catch (Exception e) {
                return false;
        }
   }

  private void log(String s) {
     System.out.println((new StringBuilder())
           .append("eXistRealmLoginModule::").append(s).toString());
  }
}
```

Com es pot observar, la classe fa ús d'un mètode privat per autentificar amb eXist (*private boolean authenticate*). Pel que fa a aquesta operació, cal destacar un parell de qüestions: primer que s'ha de transformar la clau a la codificació que hem triat per guardar a la nostra base de dades (en el nostre cas fem un HashMD5), i segon que la consulta és completament dependent del l'estructura de la base de dades eXist.

Al annex podeu trobar una implementació de la classe encarregada d'aplicar la funció de Hash.

3. Annex

3.1. Article “Custom Realms and Groups in Glassfish - By Nithya Subramanian”

Article publicat originalment al blog:
http://blogs.oracle.com/nithya/entry/groups_in_custom_realms

Glassfish provides support for Custom Realms and Custom Login Modules that are based on the JAAS framework. This post explains how to write a simple Realm class and its corresponding LoginModule, configure them with an illustration of a simple web application that uses this realm.

Custom Realm
The Custom Realm should extend com.sun.appserv.security.AppservRealm. The Realm class is basically meant to provide user and group-related information. The methods to be implemented are

i) public void init(Properties properties) throws BadRealmException, NoSuchRealmException
- This method is invoked during server startup when the realm is initially loaded. The realm can do any initialization it needs in this method. The Properties is a set of key-value pairs configured while creating the Realm and are present in domain.xml. Among the other custom properties, there is a property *jaas-context* (which is explained later in this post). This property should be set using the call setProperty method implemented in the parent class. If the method returns without throwing an exception, the Enterprise Server assumes that the realm is ready to service authentication requests. If an exception is thrown, the realm is disabled.

ii) public String getAuthType() - This method returns a descriptive string representing the type ofa uthentication done by this realm.

iii) public Enumeration getGroupNames(String user) throws InvalidOperationException, NoSuchUserException -This method returns the group names the user belongs to as an Enumeration of Strings.

Custom LoginModule
The Custom LoginModule should extend com.sun.appserv.security.AppservPasswordLoginModule. This class should override the method

abstract protected void authenticateUser() throws LoginException

This method performs the actual custom authentication, by either using a database, or LDAP or a file or even a simple Hashtable as illustrated in the attached sample code. The custom login module must not implement any of the other methods, such as login(), logout(), abort(), commit(), or initialize(). Default implementations are provided in AppservPasswordLoginModule which hook into the Enterprise Server infrastructure.

The custom login module can access the following protected object fields, which it inherits fromAppservPasswordLoginModule. These contain the user name, password of the user to be authenticated and the currentRealm class.

protected String _username;

protected String _password;

protected com.sun.enterprise.security.auth.realm.Realm _currentRealm;

The *authenticateUser()* method should end with a call to the *commitUserAuthentication(String[] authenticatedGroupList)* method where the *authenticatedGroupList* is the list of groups the user belongs to.

As can be observed, the realm class is isolated from the LoginModule. The Realm is capable of capturing arbitrary configuration information and can help in obtaining the Group information. The Group information from the Realm can be populated into the authenticated JAAS subject during commit() phase following a successful LoginContext.login() call on the authentication module. This populated group information is then used by the container in its authorization policy decisions.

Attached here is the source code of a simple sample realm class and the custom module. In this example, the Realm class stores the user-group information in a hashtable. The LoginModule class stores the user-password information in a hashtable and performs authentication. It obtains the *authenticatedGroupList* from the Realm class*' getGroups(username)* method*.*
To test this sample(it works with both GF v2 and v3), download and install Glassfish v3 from here, drop the binaries of this realm and custom module in <GF-ROOT>/domains/domain1/lib/, start the server and create the realm using the Admin console. The realm classname should be specified as *com.samplerealm.SampleRealm.*

An additional realm property jaas-context should be specified to say sampleRealm. This value should refer to the SampleLoginModule class in the

<GF-ROOT>/domains/domain1/config/login.conf

file as follows:

```
sampleRealm {

        com.samplerealm.SampleLoginModule required;

};
```

where *sampleRealm* refers to the value defined in the jaas-context property.

As can be seen from the source files, the users configured in this realm are userA, userB whose corresponding passwords are abc123, xyz123. userA has been configured in the group devGroup, while userB belongs to testGroup. To test this realm, this web-application can be used.

Observe that the web.xml of the web-app contains the following :

```
<login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>SampleRealm</realm-name>
</login-config>
```

where *SampleRealm* in the <login-conf><realm-name> element points to the name of the configured Realm as can be seen in the server's domain.xml

```
<auth-realm classname="com.samplerealm.SampleRealm" name="SampleRealm">
        <property name="jaas-context" value="sampleRealm"></property>
</auth-realm>
```

To access the web-app using the group names of the user, the following mapping between the role and group is required in sun-web.xml*:*

```
<security-role-mapping>
        <role-name>tester</role-name>
        <group-name>devgroup</group-name>
</security-role-mapping>
```

where the role-name matches the configured role in the auth-constraint of web.xml

```
<auth-constraint>
        <description/>
        <role-name>tester</role-name>
</auth-constraint>
```

and group-name is the corresponding group the user belongs to as defined in the custom realm class. So if the application is accessed using the user/password: userA/abc123, the user is authorized to view the pages, since he belongs to the devGroup, but not userB/xyz123 (who belongs to testGroup).

3.2. MD5Hash

```
package com.eXistRealm;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class MD5Hash {
    private MessageDigest algorithm;
    public MD5Hash() {
          try {
                  this.algorithm=MessageDigest.getInstance("MD5");
          } catch (NoSuchAlgorithmException e) {
          }
    }

    public byte[] StringToByte(String s) {
       int len = s.length();
       byte[] data = new byte[len / 2];
       if (len%2==1) data = new byte[(len/2)+1];
       for (int i = 0; i < len; i += 2) {
          if ((i+1) < len) data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
                          + Character.digit(s.charAt(i+1), 16));
          else data[i / 2] = (byte) (Character.digit(s.charAt(i), 16) << 4);
       }
       return data;
    }

     private String toHexadecimal(byte[] digest){
          String md5val = "";
          StringBuffer hexString = new StringBuffer();
          for (int i = 0; i < digest.length; i++)
          {
             String hex = Integer.toHexString(0xFF & digest[i]);
             if (hex.length() == 1)
             {
                hexString.append('0');
```

```
            }
            hexString.append(hex);
        }
        md5val = hexString.toString();
        return md5val;
    }

    public String hash(String s) {
        algorithm.update(s.getBytes());
        byte[] res=algorithm.digest();
        String l=toHexadecimal(res);
        return l;
    }

    public boolean isEqual(String pass1, boolean hash1,String pass2, boolean hash2) {
        byte[] password1=StringToByte(pass1), password2=StringToByte(pass2);
        if (hash1) password1=StringToByte(hash(pass1));
        if (hash2) password2=StringToByte(hash(pass2));
        return MessageDigest.isEqual(password1, password2);
    }
}
```