OpenEdge Single Point of Authentication

Using OpenEdge SPA with OpenEdge Business Process Server

Overview

OpenEdge Single Point of Authentication is an authentication service for use in realm-based systems. The first OpenEdge product to support this service is the OpenEdge Business Process Server (BP Server). BP Server has built-in support for JDBC, LDAP, and an LDAPHybrid realm that combines LDAP and JDBC. BP Server also supports custom realms, which is how this feature is delivered.

The OEHybrid realm is modeled after the LDAPHybrid realm. An OpenEdge AppServer based service provides support for user authentication and limited user attributes, while the JDBC realm is used for BP Server specific support such as groups, user attributes not supported in SPA, and permissions.

An ABL reference implementation that utilizes the _User table is provided. OpenEdge developers can also create their own implementation by implementing the built-in interface Progress.Security.Realm.IHybridRealm.

Configuring an Authentication Service

Before configuring the Business Process Server to use the new authentication realm, you must create an authentication service running on an OpenEdge AppServer running in State-free mode. The reference implementation uses the _User table of the default database the AppServer is connected to. Here is how you get the reference implementation up and running.

Step 1 – Populate _User table of primary database

Your authentication service needs access to user account information. The reference implementation uses the _User table of an OpenEdge database. The following fields in a _User record are applicable to the reference implementation:

Field Name	BPS Attribute Name	Description
_Userid	userid	The user id someone uses to log into the Business Process Portal. Must be unique.
_Password	password	Password of the user stored in clear text.
_User_number	Used internally by realm	The _User_number field contains a unique number assigned to the user. This number is used by realm APIs once a userid has been validated.
_Given_name	firstname	First name of the user.
_Surname	lastname	Last name of the user.
_Telephone	phone	Telephone number of the user.
_Email	email	Email address of the user.
_Description	description	Descriptive text of the user.

Step 2 – Deploy reference implementation to a state-free AppServer

The reference implementation source code can be found in $DLC/src/samples/security. It consists of an implementation class – OpenEdge.Security.Realm.HybridRealm.cls and a utility class for property file support – OpenEdge.Security.Util.Properties.cls. Using the fully qualified package name, copy these classes to the proper subdirectory of your AppServer's $WRKDIR.

Step 3 – Configure reference implementation properties

The reference implementation utilizes a property file that contains information used to lock down the authentication service. A SPA client will read a sealed client-principal file from disk and send it when communication with the authentication service. The authentication service will validate the seal by using the domain password, and then examine the role of the client-principal to see if it matches. A default property file is provided in $DLC/src/samples/security/spaservice.properties. It should be copied to $WRKDIR. The default property file contains these values:

```
Password=oech1::20333c34252a2137
Role=SpaClient
DebugMsg=false
```

Password is the encoded domain password of the sealed client-principal.

Role is the role that the client-principal role is compared to.

DebugMsg will enable log messages in the AppServer server log file for the service.

The default client-principal found in BPSHOME/conf/spadefault.cp uses a password of 'password', which corresponds to the encoded value above, and the role 'SpaClient'. See below for information on creating your own client-principal file using the genspacp utility.

Configuring SPA in OpenEdge Business Process Server

These instructions assume a fresh installation of BP Server and a newly provisioned repository. An existing system would require migration of existing user accounts and is beyond the scope of this document. These instructions also assume that you have installed the SPA reference implementation, or developed your own on an OpenEdge AppServer version 11.3 or later running in state-free mode.

Step 1 – Create Admin BP Server User

When you installed BP Server, you specified a default user and password, usually admin/admin. Since the SPA service only authenticates users while BP Server manages permission, not having this user will prohibit you from administering BP Server.

Option 1

Log into the portal with as admin and go to the user administration page. Create a new user that exists in your authentication server and give that user all permissions and options.

Option 2

Create the user 'admin' on your authentication server. When you switch over to use the custom authentication realm, the admin user will be authenticated and you can then manage permissions for the rest of your users that exist in your realm.

Step 2 – Specify custom authentication realm in BP Server

You specify the attributes of your custom authentication realm in BPSHOME/conf/umacl.conf as shown below.

```
###### User Manager Properties ######
# <param name="usermgr.realm.type">
#    <visible>true</visible>
#    <alias>User Management realm type</alias>
#    <description>Specifies the realm type for user management</description>
#    <legalvalues>jdbc|ldap|ldaphybrid|custom</legalvalues>
# </param>
usermgr.realm.type=custom

# <param name="usermgr.realm.provider">
#    <alias>User manager realm provider</alias>
#    <description>This property is read only if property 'usermgr.realm.type'
#        has value 'custom'. Specify the fully-qualified class name of user
#       management realm implementation class. More information can be found
#       in "Customization Guide" under chapter "Customizing User Management
#       Java interface".
#    </description>
#    <legalvalues>ANY</legalvalues>
# </param>
usermgr.realm.provider=com.savvion.usermanager.OERealm

###### OpenEdge Single Point of Authentication Properties ###########
# <param name="oeauth.server.location">
#    <alias>OpenEdge AppServer URL</alias>
#    <description>URL to an AppServer providing the SPA service</description>
#    <legalvalues>Any valid AppServer URL</legalvalues>
#    <group>OESPA</group>
# </param>
oeauth.server.location=AppServer://localhost/sbmbroker1

# <param name="oeauth.server.provider">
#    <alias>OpenEdge Class Name</alias>
#    <description>Class name of implrementation providing SPA
#        service</description>
#    <legalvalues>Fully qualified class name that implements the
#    Progress.Security.Realm.IHybridRealm interface </legalvalues>
#    <group>OESPA</group>
# </param>
oeauth.server.provider=OpenEdge.Security.Realm.HybridRealm

# <param name="oeauth.server.authmethod">
#    <alias>Method used to validate password</alias>
#    <description>Basic authorization will send password over the wire.
#        Digest authorization will send a cryptographic has of the
#        password.</description>
#    <legalvalues>basic|digest</legalvalues>
#    <group>OESPA</group>
# </param>
oeauth.server.authmethod=digest

# <param name="oeauth.server.clientprincipal">
#    <alias>Location of sealed clientprincipal file</alias>
```

```
#    <description>File name of the client principal file used to authorize
SPA clients</description>
#    <legalvalues>filename</legalvalues>
#    <group>OESPA</group>
# </param>
oeauth.server.clientprincipal=spadefault.cp
```

Step 3 – Enable group administration in the portal

Because this is a hybrid realm, users are authenticated externally but groups are managed in the BP Server repository. By default, group management is disabled when using a custom realm. Open BPSHOME/conf/bpmportal.conf and change the property bpmportal.customrealm.managegroup to true.

Step 4 – Log in with default user and set user attributes and permissions

Start your AppServer and BP Server and log in with the admin user. If everything is configured properly, you will be able to log in and set your users' attributes and permissions for use with BP Server. Note that because the authentication realm is external to BP Server, there is no support for adding or deleting users from the portal.

Generating a Sealed Client-Principal File

A utility, genspacp, is provided to generate a sealed client-principal file for use with the SPA service. From a proenv window, type genspacp to see its usage.

genspacp 1.0

usage: genspacp -password <text> [-user <text> -domain <text> -role <text> -file <text>]

-password: Required. The domain password used to seal the client-principal.

-user: Optional. The user name set in the client-principal. The default is BPSServer.

-domain: Optional. The domain name set in the client-principal. The default is OESPA.

-role: Optional. The role set in the client-principal. The default is SPAClient.

-file: Optional. The file name the client-principal is saved to. The default is oespaclient.cp.

Running genspacp with –password value of foobar will generate the following output:

```
proenv>genspacp -password foobar
genspacp 1.0
Generated sealed Client Principal...
   User: BPSServer@OESPA
   Id: Rpj2gs5WT2G+EqivLYaKzA
   Role: SPAClient
   Encoded Password: oech1::363d20253337
   File: oespaclient.cp
   State: SSO from external authentication system
   Seal is valid
```

You would copy the value oech1::363d20253337 to the Password property of your spaservice.properties file in order to use the client-principal. You would copy the client-principal file created, in this example oespaclient.cp, to the BPSHOME/conf directory and update the umacl.conf file with the filename.

Implementing a SPA Service

An SPA Service is created by implementing the built-in interface Progress.Security.Realm.IHybridRealm. This interface is implemented as a singleton class and should not save any state related to a particular client. The interface is specified below.

```
INTERFACE Progress.Security.Realm.IOEHybridRealm:

/*------------------------------------------------------------------
 Purpose: Returns the specified user attribute for the user id provided
 Notes: If an attribute is not found, return UNKNOWN
------------------------------------------------------------------*/
      METHOD PUBLIC CHARACTER GetAttribute(INPUT theUserId AS INTEGER,
attrName AS CHARACTER).


/*------------------------------------------------------------------
 Purpose: Returns all attribute names associated with the user id
 Notes:
------------------------------------------------------------------*/
      METHOD PUBLIC CHARACTER EXTENT GetAttributeNames(INPUT theUserId AS
INTEGER).


/*------------------------------------------------------------------
 Purpose: Returns a list of all the users in the realm
 Notes:
------------------------------------------------------------------*/
      METHOD PUBLIC CHARACTER EXTENT GetUserNames(  ).


/*------------------------------------------------------------------
 Purpose: Returns a list of users that match a query string
 Notes:
------------------------------------------------------------------*/
      METHOD PUBLIC CHARACTER EXTENT GetUserNamesByQuery(INPUT queryString AS
CHARACTER).


/*------------------------------------------------------------------
 Purpose: Removes the specified user attribute for the user id provided
 Notes:
------------------------------------------------------------------*/
      METHOD PUBLIC LOGICAL RemoveAttribute(INPUT theUserId AS INTEGER, INPUT
attrName AS CHARACTER).


/*------------------------------------------------------------------
 Purpose: Sets the specified user attribute for the user id provided
```

```
 Notes:
------------------------------------------------------------------------*/
      METHOD PUBLIC LOGICAL SetAttribute(INPUT theUserId AS INTEGER, INPUT
attrName AS CHARACTER, INPUT attrValue AS CHARACTER).


/*------------------------------------------------------------------------
 Purpose: Validates a user password using the basic authentication method
 Notes: Password is sent over the wire in clear text
------------------------------------------------------------------------*/
      METHOD PUBLIC LOGICAL ValidatePassword(INPUT theUserId AS INTEGER,
INPUT password AS CHARACTER).

/*------------------------------------------------------------------------
 Purpose: Validates a user password using the digest authentication method
 Notes: Password value is a digest consisting of BASE64(SHA1(password + nonce
+ timestamp))
------------------------------------------------------------------------*/
    METHOD PUBLIC LOGICAL ValidatePassword(INPUT theUserId AS INTEGER, INPUT
password AS CHARACTER, INPUT nonce AS CHARACTER, INPUT timestamp AS
CHARACTER).



/*------------------------------------------------------------------------
 Purpose: Validates that the specified user name exists and returns its id
 Notes: Most of the other APIs require the user id this method returns
------------------------------------------------------------------------*/
      METHOD PUBLIC INTEGER ValidateUser(INPUT userName AS CHARACTER).

END INTERFACE.
```