PAS for OpenEdge WebSpeed with OERealm Security for Multi-tenant Database

(For OpenEdge Version 11.6.x)

Irfan Syed
Progress Software

Table of Contents

Introduction 3
Security for WebSpeed Applications in PAS for OpenEdge 4
WebSpeed User accounts via OERealm 4
Database Security Configurations to run OERealm 6
OERealm Configuration in Spring Security 7
OERealm Server Class 12
Appserver Startup and Event Procedures 14
Startup Procedure 14
Activate event procedure 15
FAQs 19

Introduction

The Progress Application Server for OpenEdge (PAS for OpenEdge) is a web server preconfigured to expose your ABL business applications to HTTP clients[1] in an internet/intranet environment. Instead of using adapters like the classic OpenEdge AppServer, PAS for OpenEdge implements each service-interface type as a transport. Transports translate HTTP messages into requests that can be executed in ABL business applications. Currently, there are four transports: APSV[2], REST, SOAP and WEB[3]. All the transports are packaged into a single PAS for OpenEdge web application (OEABL)[4] which facilitates a single ABL business application handling requests from different HTTP client types. (Compare to the adapter-based "classic" OpenEdge AppServer where you must run a separate instance for each service-interface type.)

Also included in the OEABL web application is an industry known and tested web security layer, the Spring Security framework, that effectively handles user authentication and URL authorization actions. You no longer need to build that security functionality into your business application yourself. ABL business applications not only enjoy the benefit of running in a secured web environment but they also benefit from their ABL logic being executed in a Multi-Session Agent with PAS for OpenEdge.

The OpenEdge application server technology that ran classic WebSpeed applications via WebSpeed Messengers depended largely on strong web security support from ABL code in the form of OpenEdge application-level security. PAS for OpenEdge opens the door for legacy WebSpeed applications to be simpler and more secure by taking advantage of the security features available in the OEABL web application. However, you do not need to sacrifice your ABL application's user account infrastructure.

To support user accounts implemented in ABL applications that can coexist with PAS for OpenEdge's strong web security layer, OpenEdge includes the *OERealm* ABL interface. OERealm has two components: one that runs in the PAS for OpenEdge OEABL application, and a developer-written OOABL class that runs in the PAS for OpenEdge multi-session Agent.

In this document, we will go through the steps that are required to configure OERealm security for WebSpeed Applications in PAS for OpenEdge for both single and multi-tenant OpenEdge databases.

---

[1] HTTP clients include OpenEdge ABL/Java/.NET, browsers, mobile devices, etc.

[2] APSV is a replacement for the classic OpenEdge AppServer's AIA adapter

[3] WEB is the transport name given to the next generation of classic WebSpeed support

[4] The OEABL (`oeabl.war`) web application contains all PAS for OpenEdge transports, and is the foundation for the built-in ROOT web application and all WAR files generated from Progress Developer Studio for OpenEdge projects.

Security for WebSpeed Applications in PAS for OpenEdge

The PAS for OpenEdge web application derived from `oeabl.war` uses the Spring Security framework for its web application security. The architectural strategy is to provide an industry-proven implementation that replaces the need for ABL developers to write, test, and maintain their own security functionality in their own code.

Among the features providing secure client access to WebSpeed applications, which is supplied by the Spring Security framework in PAS for OpenEdge, are:

- User authentication
- Role Based Access (RBA) authorization to URL resources
- Session login and logout
- Cross-Origin Resource Sharing (CORS) support
- Welcome (home) page support

Once Spring Security has authenticated and authorized a client, it passes a sealed ABL Client-Principal containing the user's session, roles, and optional account data to the ABL business application where it can be used in ABL databases, auditing, and its own application authorization functionality.

> **Note:** Beginning in OpenEdge release 11.6.0, the WEB transport was added to enable the Spring Security framework to supply secure client access to WebSpeed business applications. The Web transport's Spring Security configuration is an extension of the configuration used by the existing APSV, SOAP, and REST transports

As good as the Spring Security framework is, to fully secure a WebSpeed application against both internet and intranet attacks a WebSpeed developer should also become familiar with the underlying Apache Tomcat security architecture in PAS for OpenEdge, and with industry recommended best practices for securing web servers and web applications.

The focus of this paper is on securing the WEB transport for a WebSpeed application **Web** transport, it's supporting static files, and in using the Client-Principal within their ABL code.

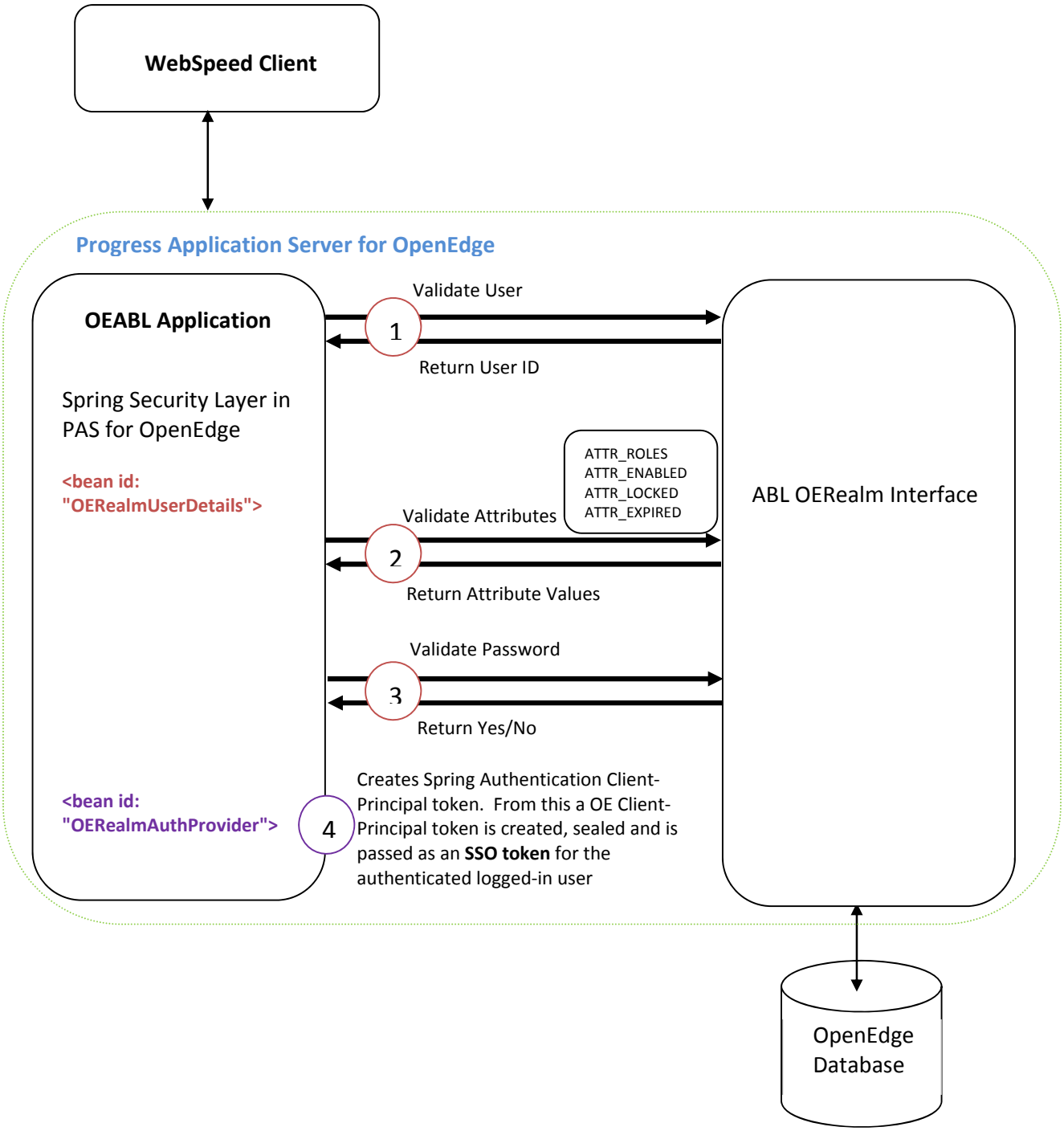
WebSpeed User Accounts via OERealm

WebSpeed and other OpenEdge applications have often been implemented with their own databases for user account storage and related functionality. To support this in the past, OpenEdge added the OERealm feature. Currently, in PAS for OpenEdge, OERealm can also interface with the Spring Security authentication process for OpenEdge business applications. With a WebSpeed business application as the effective source of user accounts through OERealm, the Spring Security framework can be used when testing, demonstrating or deploying your application.

OpenEdge supplies a configurable OERealm client in its `oeablSecurity-*-oerealm.xml` configuration files, which call an OERealm server ABL class interface. That OERealm client

can be configured to execute methods in an OOABL singleton class (aka OERealm server class) written by the WebSpeed developer. The OERealm server class implementation can choose to supply user account information to Spring Security from any source it is configured to use: OpenEdge database `_user` table, OpenEdge database WebSpeed application account table, Lightweight Directory Access Protocol (LDAP), Microsoft Active Directory (AD), etc. The only limitation is that the source of user accounts must be accessible via the ABL programming language.

The following diagram illustrates the functional behavior of the OERealm Security Model:



Database Security Configurations to run OERealm

To configure and run OERealm in the web application's user authentication process, you should observe the following best-practices:

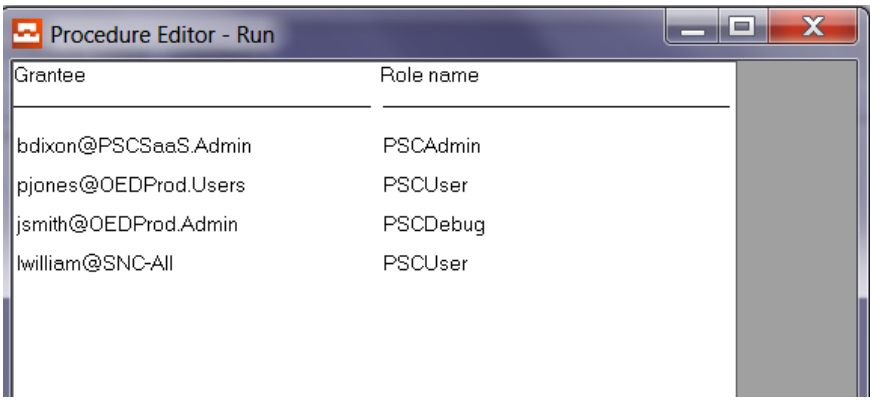
1. Every user account must have a unique numerical identifier as character based user accounts names would not be considered to describe a unique individual.

   The unique number assigned to the user is used to describe a single unique account across multiple OERealm client requests during a single web application client authentication. This number is used by realm APIs once a user-id has been validated.

2. Every database user should be assigned to one or more roles.

   The Spring Security framework provides Role-Based Access (RBA) for URL resources, including WebSpeed application URLs.

   Spring Security follows the convention of `ROLE_` as prefix for user roles. When creating new roles, **do not** use the `ROLE_` prefix. For example:



3. Configure user account validation attributes in Spring Security

   In the OERealm Spring Security, the OERealmUserDetails bean validates a few user attributes as shown in the OERealm Security Model diagram on the previous page.

   ATTR_ROLES – the OERealm class looks for the roles assigned to the user and returns them as a string to OERealmUserDetails.

   ATTR_ENABLED - the OERealm class checks the user account table to verify if the user is enabled.

   ATTR_LOCKED - the OERealm class checks the user account table to verify if the user is locked.

ATTR_EXPIRED - the OERealm class checks the user account table to verify if the user is expired.

Return values should be a single character and should equal to:

- `T`,`Y`, or `1` if the account is enabled/not-locked/not-expired
- `F`,`N`, or `0` if the account is disabled/locked/expired

Note: The OERealmUserDetails service requires the above attributes but if these values are not used in the OERealm Server Class, then return the default values as:

ATTR_LOCKED=false, ATTR_EXPIRED=false, ATTR_ENABLED=true

4. The OERealm service Interface has a role in the authentication security process and needs to secure itself against hackers and rogue users. While this is optional, it is highly recommended.

OERealm Configuration in Spring Security

After setting the security configuration to one of the OERealm XML files in an instance's `web.xml` file, you must modify the appropriate properties in the `oeablSecurity-xxx-oerealm.xml` file to configure OERealm authentication. The properties to modify are in the `OERealmUserDetails`, the `OERealmAuthProvider` and the `OEClientPrincipalFilter` beans.

OERealmUserDetails – Responsible for executing the OEUserRealm Class and validating the logged-in user. The following shows the default property settings:

```
<b:bean id="OERealmUserDetails"
      class="com.progress.appserv.services.security.OERealmUserDetailsImpl" >
      <b:property name="realmURL" value="AppServer://localhost:5162/oerealm" />
      <b:property name="realmClass" value="OpenEdge.Security.Realm.HybridRealm" />
      <b:property name="grantedAuthorities" value="ROLE_PSCUser" />
      <b:property name="rolePrefix" value="ROLE_" />
      <b:property name="roleAttrName" value="ATTR_ROLES" />
      <b:property name="enabledAttrName" value="ATTR_ENABLED" />
      <b:property name="lockedAttrName" value="ATTR_LOCKED" />
      <b:property name="expiredAttrName" value="ATTR_EXPIRED" />
      <b:property name="realmPwdAlg" value="0" />
      <!--
      <b:property name="realmTokenFile" value="" />
       -->
      <!-- For SSL connection to the oeRealm appserver provide the complete
           path of psccerts.jar as the value of 'certLocation' property
      -->
      <b:property name="certLocation" value="" />
      <!-- set appendRealmError = true in order to append the Realm
      class thrown error in the error details send to the REST Client -->
      <b:property name="appendRealmError" value="false" />
</b:bean>
```

Modify the following properties in `OERealmUserDetails`:

`realmURL` – The AppServer URL where the OERealm Class is deployed. By default, it points to a classic OpenEdge AppServer. To execute the OERealm Class that is deployed in the same PAS for OpenEdge instance, the URL can be changed to `internal://localhost/nxgas`.

`realmClass` – OERealm Service Interface OOABL class. The default points to the OpenEdge sample HybridRealm class, `OpenEdge.Security.Realm.HybridRealm`. It should be changed to specify the custom OERealm class that is written for and distributed with your deployment.

`AttrName properties`– the `roleAttrName`, `enabledAttrName`, `lockedAttrName`, and `expiredAttrName` properties designate the *`attrName`* CHARACTER parameter input to the OERealm server class's `GetAttribute()` method. The values of these properties provide abstract names for common user account fields, which allow the OERealm server class to map the abstract user account field name to the physical user account field name that it is programmed to use. While these OERealmUserDetails properties may never change, they exist to provide the option of changing them. This is useful when another OERealm client uses the same OERealm server class and requires that certain strings be used for the `GetAttribute(` *`attrName`* `)` parameter.

`realmTokenFile` – By default in OERealmUserDetails is configured to invoke the OERealm class as an anonymous user to make access easier during development. It is always recommended in a production environment to use authenticated user access to the OERealm class in order to block hackers gaining access to your application's user account data. In this case the authenticated user is the web application's OERealmUserDetails class and the `realmTokenFile` is a file containing a sealed Client-Principal.

In a multi-tenant OpenEdge database, the authenticated user needs super tenant privileged user to access the OERealm class and a validated logged-in user's (tenant's) identity.

Using a `realmTokenFile` provides the ability for the AppServer's OERealm class to limit authorized access to the clients executing the OERealmUserDetails class. The `realmTokenFile` points to a sealed Client-Principal file that can be generated using the OpenEdge `$DLC/bin/genspacp` utility.

The utility requires the user ID and role, domain name and access code, and output file name to create a binary file that contains the sealed Client Principal in the `.cp` format. The domain name and domain access code correspond to those that are in use in the OERealm service interface to validate clients.

The syntax for genspacp is:

```
genspacp -password domain-access-code
[-user db-user -domain db-user-domain -role db-user-role
-file filepath.cp]
```

For example:

```
genspacp -password pscsaas.admin -user mywebapp -role
OERealmClient
-domain OERealmClients -file wrk/superuser.cp
```

Note: We recommend that the Domain name used in the -domain option and the domain access code used in the -password option be configured in your OpenEdge database for secure storage and for Client-Principal validation.

OERealmAuthProvider – Uses the user account details from OERealmUserDetails and creates a Spring authentication token. The following shows the default property values:

```
<b:bean id="OERealmAuthProvider"
        class="com.progress.appserv.services.security.OERealmAuthProvider" >
        <b:property name="userDetailsService">
                    <b:ref bean="OERealmUserDetails"/>
        </b:property>
        <!--
        When createCPAuthn property is true:
           will create an OE ClientPrincipal with its additional attributes
           that the Spring token does not. This is qualified by the
           "sealClientPrincipal" property, which when true will use this bean's
           domain and registry configuration to "seal" the ClientPrincipal; and
           when false will leave the ClientPrincipal unsealed until the
           OEClientPrincipalFilter performs the seal operation.
        When createCPAuthn property is false
           a basic Spring token is generated and passed to the OEClientPrincpial
           filter, which will use its configuration to create a ClientPrincipal,
           copy the Spring token's information into it, and then seal it using
its
           domain and access-code configguration.  All other properties relating
           to ClientPrincpal generation and sealing are ignored
        -->
        <!--
            <b:property name="createCPAuthn" value="true" />
            <b:property name="multiTenant" value="false" />
            <b:property name="sealClientPrincipal" value="false" />
            <b:property name="key" value="oech1::23263c143737253a3337" />
            <b:property name="registryFile" value="" />
            <b:property name="userDomain" value="" />
            <b:property name="authz" value="false" />
            <b:property name="properties" >
                <b:map>
                     <b:entry key="prop-1" value="string1"/>
                     <b:entry key="prop-2" value="string2"/>
                </b:map>
            </b:property>
            <b:property name="expires" value="600" />
            -->
</b:bean>
```

The following table describes the relevant properties in `OERealmAuthProvider`:

Spring Property	Description
`createCPAuthn`	Creates a Spring ClientPrincipal Authentication token and an OEClientPrincipal if the value is `true`, which is the default. If set to `false`, the ClientPrincipal token is passed to the OEClientPrincipal filter. The information returned from the OERealm Server will not be available in the sealed Client-Principal
`sealClientPrincipal`	If `true`, then it seals the client-principal with the values assigned to the `key` or `registryFile` properties. If the value is set to `false`, then it creates the client-principal and expects OEClientPrincipal filter to seal it. By default, this value is set to `false`. Recommendation is to leave this value as `false` so that OEClientPrincipal filter acts as the single point for sealing client-principals.
`Authz`	Load the user account's roles as the Spring Security "authorities" that are used in authorization. This ensures that the ABL application works with the same role names that are assigned to the user in the User table, which simplifies coordination of the authorization process between ABL applications and Spring Security.
`userDomain`	The OpenEdge domain name to append to the input user account name if the input user account name does not already contain one For Multi-domain configurations the value should be empty
`multiTenant`	When true it allows the user to enter a fully qualified OpenEdge userid like `user-id@domain`. When true `key` attribute should be removed and the `registryFile` attribute specified

OEClientPrincipalFilter – The OEClientPrincipalFilter is a standard Spring Security Filter bean which converts the Spring Authentication Client-Principal token generated at the OEAuthProvider to an OEClientPrincipal token and seals it with either the `key` or `registryFile` values (if the client-principal has not already been sealed in OERealmAuthProvider).

As the properties required for creating a OEClientPrincipal are already defined in the OERealmAuthProvider bean, the OEClientPrincipal created in the OEClientPrincipalFilter

inherits all those attributes, hence we will perform the action of sealing the Client-Principal based on the properties defined.

```
<b:bean id="OEClientPrincipalFilter"
 class="com.progress.appserv.services.security.OEClientPrincipalFilter"
>
            <b:property name="enablecp" value="false" />
            <b:property name="registryFile" value="" />
            </b:property>
    </b:bean>
```

Spring Property	Description
enableCP	Will perform Client-Principal creation and sealing. If a Client-Principal is already available then it will perform seal
Key	Domain-access-code to seal the Client-Principal token for single Tenant (domain) configurations; removed for multiple Tenant (domain) configurations
registryFile	Registry file used to seal the Client-Principal for multiple Tenant (domain) configurations; removed for single Tenant (domain) configurations. The registry file is generated using the gendomreg utility

In-order to seal the generated Client-Principal, either a `key` or a `registryFile` property must be specified.

The `key` property is used only when Spring Security is configured to run on a single domain. The value of `key` is a domain-access-code.

When Spring Security is configured is run on multiple domains, the `registryFile` property is set to the name of a file that contains a list of appropriate domain-access-codes. To generate a valid registry file that has a list of domain-access-codes, use the `$DLC/bin/gendomreg` utility.

The gendomreg utility requires as input a `.csv` text file that has a list of domain names and their domain-access-codes separated by commas. Domain/access code pairs appear on separate lines in the file. For example:

```
OEDProd.Admin,oedprod.admin
OEDProd.Users,oedprod.users
SNC-All,snc-all
PSCSaaS.Admin,pscsaas.admin
```

The output registry file, in which the domain/access code information is encrypted, is specified as a `.bin` file. For example:

```
gendomreg  domreg.csv  domreg.bin
```

When the `registryFile` property is set, comment out the `key` section in `OERealmAuthProvider`. The default location for the registry file is in *`instance-location`*`/conf`, If you put the registry file in that location, only the file name is required as the value for the `registryFile` property. But if the registry file is copied to some other location, then the entire path must be specified.

Note: This section highlighted only the important properties in the `OERealmAuthProvider` and `OERealmUserDetails` section. For a detailed overview of all the properties, please refer to the following documentation:

https://documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/asadm/configure-the-spa-security-configuration-model.html#

OERealm Server Class

The OERealm Server Class is responsible for providing user account specific information to OERealm clients. To write a custom OERealm Server Class, you can create a OOABL class that implements from the `Progress.Security.Realm.IHybridRealm` interface.

Mandatory methods of the `IHybridRealm` interface that are used in a OERealm Server Class are:

- `ValidateUser( )`
- `ValidatePassword( )`
- `GetAttribute( )`

You can find more information on both the mandatory and optional methods in *OpenEdge Application Server: Administration*.

An abstract OERealm Server Class would look similar to the following.

```
/*****************************************************************/

CLASS OEUserRealm IMPLEMENTS IHybridRealm:


  METHOD PUBLIC INTEGER ValidateUser( INPUT userName AS CHARACTER ):

  // Returns the User-id of the UserName

  // Return "-1" if the User not found and throw the error in appserver
  // logs
```

```
        UNDO,THROW NEW Progress.Lang.AppError("User not found", 6).


  END METHOD.

  METHOD PUBLIC LOGICAL ValidatePassword( INPUT theUserId AS INTEGER,
INPUT
  password AS CHARACTER ):

  // Returns "true" once the password is validated for the given User-id

  // Returns "false" if the password is not validated for the User-id and
  // throw the error to the appserver logs

  UNDO,THROW NEW Progress.Lang.AppError("Invalid User Id", 6).

  END METHOD.

  METHOD PUBLIC CHARACTER GetAttribute( INPUT userNum AS INTEGER,
         INPUT attrName AS CHARACTER ):

  CASE attrName:

  WHEN OERealmDefs:REALM_ATTR_ROLES THEN // REALM_ATTR_ROLES set to "
                                         // ATTR_ROLES"
        DO:
          DEFINE VARIABLE roles AS CHARACTER NO-UNDO. // Returns ROLES
        END.

  WHEN OERealmDefs:REALM_ATTR_ENABLED THEN  // REALM_ATTR_ENABLED set to
                                            // "ATTR_ENABLED"
        DO:
         // Return any of the values from {T,Y,1}  if account is enabled
         // Return any of the values from {F,N,0} if account is disabled
        END.

  WHEN OERealmDefs:REALM_ATTR_LOCKED THEN    // REALM_ATTR_LOCKED set to
                                             //"ATTR_LOCKED"
        DO:
         // Return any of the values from {T,Y,1}  if account is locked
         // Return any of the values from {F,N,0} if account is not
locked
        END.

  WHEN OERealmDefs:REALM_ATTR_EXPIRED THEN   // REALM_ATTR_EXPIRED set to
                                             // "ATTR_EXPIRED"
        DO:
         // Return any of the values from {T,Y,1}  if account is expired
         // Return any of the values from {F,N,0} if account is not
expired
                END.
        END.

    END METHOD.
END CLASS.
/*****************************************************************/
```

If mandatory values like username or password do not match, then the OERealmUserDetails filter will fail with a 401 Error. If other values are not set in the user account table, then the OERealmUserDetails will set them to its own default values.

The OOABL Realm Class exceptions or errors are returned to the OERealmUserDetails Client and those errors are displayed in the session-manager log file of the PAS for OpenEdge instance.

If you want ABL errors to be returned to the client of the PAS for OpenEdge instance, you must enable the following property in the OERealmUserDetails filter in the Spring Security properties file:

```
<b:property name="appendRealmError" value="true" />
```

> **Caution:** The recommended best practice is only to enable `appendRealmError` during development for debugging purposes, and not in production deployments. In production, there is the potential for exposing information that may be useful to hackers.

AppServer Startup and Event Procedures

When the Spring Security OERealmUserDetails `userTokenFile` attribute is configured to send a sealed Client-Principal to ABL business logic, the recommended best practice is to validate the user identity before executing the business logic. To do that, we have to perform appropriate initialization in a Startup Procedure and validation in an Activate Event Procedure.

Startup Procedure

An MS-Agent in PAS for OpenEdge can be configured to load/execute a user context when starting a new session. With OERealm Authentication, it is easier to validate the Client-Principal objects in that session (including the sealed Client-Principal sent by the OERealmUserDetails object) if the MS-Agent sessions load the registered domains into the session's domain registry.

In OpenEdge ABL, the LOAD-DOMAINS( ) method of the SECURITY-POLICY system handle loads the registered domains of the connected database into the ABL session domain registry. (See *OpenEdge Development: ABL Reference* for more information.)

`security-policy:load-domains( )` can be utilized in `sessionStartupProc` as below

```
DEFINE INPUT PARAMETER startup-data AS CHARACTER NO-UNDO.

Define Var m_lOk as logical no-undo.

// Loading domain registry for the session
m_lOK = security-policy:load-domains(1).
```

```
// Validate if load-domains() executed successfully
if (not m_lOk) then
    do:
        message "Error loading domains" error-status:get-message(1).
        return error "Error in load-domains():" + error-status:get-
message(1).
end.
else do:
        message "domain registries loaded successfully".
    end.
```

After sessionStartupProc executes, all the domains registered to the database connected to PAS for OpenEdge instance will be loaded in the MS-Agent session if the database configuration is correct. If not, any future Client-Principal validation will fail and sessionStartupProc should return an error to the client.

Activate event procedure

When an activate procedure is configured, it gets executed for every stateless or state-free client request. In a way, it acts as a gateway controlling a client request's access to the ABL business logic running on the AppServer. For OERealm Security, you can perform the following validations in the activate procedure:

1. **Allow requests from only specific transports**

   Beginning with the OpenEdge 11.6 release, Progress.ApplicationServer.AdapterTypes (a new Progress.Lang.Enum class) has been added to give the transport used for the current request executing on PAS for OpenEdge. AdapterTypes can be used in activate procedures to restrict access to requests from specific transports.

   The transport type can be extracted from the AdapterType property in session:current-request-info. For example:

```
using Progress.ApplicationServer.AdapterTypes.

session:current-request-info:AdapterType

      /*
      Appserver    - APSV
      WebSpeed     - WEB
      REST Adapter - REST
      SOAP Adapter - SOAP
      */
```

2. **Allow requests to execute only specific classes/methods/procedures**

   Although AdapterType can restrict access based on transport type , the application server is still free to execute any requests allowed in a transport. You can place restrictions on execution of specified classes, methods, or procedures by extraction of information using `session:current-request-info:`*`procedurename`* in a test.

   The *`procedurename`* value is the same you configured in the OERealmUserDetails realmClass property.

3. **Validate the OERealmUserDetails user identity**

   To validate the sealed Client-Principal passed by Spring Security, one can use the VALIDATE-SEAL( ) method to see if the message authentication code (MAC) generated while sealing a client-principal object is valid. For example:

```
client-principal-handle: validate-seal(oech1::encodedDomainAccessCode)
```

   where `oech1` specifies the OpenEdge encryption algorithm.

   If the domain registries are already loaded when sessionStartupProc executes, then you can validate the Client-Principal without providing the Domain Access Code. For example:

```
client-principal-handle: validate-seal( )
```

   If the OERealm ABL class needs to perform database transactions, the validated client-principal can be used to set the authenticated user as a client of a database. (For multi-tenant databases, the client-principal is also used to set an authenticated user's tenancy.) For example:

```
set-db-client (SESSION:CURRENT-REQUEST-INFO:GETCLIENTPRINCIPAL, database)
```

   where *`database`* is an integer expression, logical name or alias representing a connected database.

   For OERealm Security mode with multi-tenant databases, it is required to have a super-tenant user to validate the logged-in user's identity by running OOABL Realm Class and a logged-in user (regular tenant) to execute the AppServer business logic. By using `set-db-client()` we will separately set two different users as database clients to perform different actions.

   In the case of a tenant disabled database, any authorized user can execute the OOABL OERealm Class.

Combining all the above three steps provides a well-protected security mechanism for accessing the AppServer ABL business logic in an activate procedure.

An activate procedure implementing the above-mentioned best practices would be similar to the following code snippet:

```
/*****************************************************************/

Using Progress.ApplicationServer.AdapterTypes.


define variable hCP      as handle  no-undo.
def    var      lok      as logical no-undo.
def    var      hReqProc as char    no-undo.

// Get the Client-Principal object
hCP = session:current-request-info:GetClientPrincipal().

// Get the Client Request Procedure
hReqProc = session:current-request-info:procedurename.

// Validate the seal of the Client-Principal passed by the Spring
Security
lok = hCP:validate-seal().

// If success then set it as database client
if (lok) then
    set-db-client(hCP).
else
    return error "CP Validation failed.  Keys mismatch".


// Check for the validation of procedures executed on APSV transport
if (session:current-request-info:AdapterType eq AdapterTypes:APSV ) then
do:
    /*
        Verifying that OERealm Class is executed only by a user who is
an Administrator.  In the case of tenant enabled database it is super-
tenant and in the case of regular database it would be a user who is an
Administrator.

        In a way, we are trying to perform a role based authorization

        */
    if hReqProc matches("OpenEdge.Security.Realm.OEUserRealm.*") then
            if hCP:ROLE = "PSCAdmin" then
        message "Running OEUserRealm Interface to validate the user
details".
    else
        return error "Unauthorized role executing Realm Class".
end.

/*****************************************************************/
```

By following the steps mentioned above one can easily secure their WebSpeed Application by configuring it to the OERealm AppSecurity Model in PAS for OpenEdge. The OERealm AppSecurity model is used to authenticate when the application's users are defined in their Database User tables. If the security model of existing WebSpeed applications does involve database user authentication then the nearest possible security models (like basic/form-local, LDAP, SAML or Pre-Auth filter for SSO) can be adapted for efficient security. Regardless of security the model, the underlying functionality that Spring Security model provides is an authenticated SSO token which can be used to validate the user's identity to execute the business logic a PAS for OpenEdge instance.

FAQs

1. Can multiple transports be configured to run OERealm AppSecurity at the same time?

Yes, one oeabl.war application can support multiple transports and all of them can use the same Appsecurity mode.

2. What happens when my client-principal token expires?

For a form-oerealm security configuration, as soon as the client-principal token expires the user will be redirected to login.html to login again.

For basic-oerealm authentication, client-principal expiration returns a 401 error.

3. How do you change the time it takes for the client-principal token to expire?

In the OERealmAuthProvider bean, the default expiration time for the client-principal is set to 600 seconds (10 minutes):

```
<b:property name="expires" value="600" />
```

Change the value of the `expires` property to increase or decrease the expiration time of the Spring Security generated client-principal token.

4. How do you return the errors found and exit from an activate procedure? Can we set our own error codes?

The following ABL statement exits from the activate procedure and return errors:

```
return error "Error Message".
```

It returns the specified error message to the client with a 500 HTTP response code. You cannot override or set your own HTTP response codes

5. How do you include domains and their corresponding domain-access-codes that are not available in the database in the MS-Agent Session registry?

Using the `register-domain( )` method of the `security-policy` handle, you can register new domains for the session. For example:

```
security-policy:register-domain(domain-name,
domain-access-code, domain-description,domain-type).
```

After registering domains, the session can be locked so no additional domains can be registered during the life of the session.

6. What happens when a realmTokenFile is not used in the OERealmUserDetails bean?

   The activate procedure will receive a request which does not have an identity. So it is up to you to decide if you should allow anonymous users to execute the OOABL Realm Interface.

   In case of Multi-tenancy, it would be the safest bet to configure the super-tenant privileged user to access the OOABL Realm Interface in order to validate other user's identity.

7. If the OERealm Interface is deployed on a different server (either PAS for OpenEdge or the classic OpenEdge AppServer), how can you protect the password of a user that is passed to the AppServer OERealm Class?

   By configuring the property `realmPwdAlg` in the `OERealmUserDetails` bean, you can secure the password that is passed to the AppServer.

   If set to "0" it sends clear-text passwords. If set to "3" it sends HTTP digest passwords.

8. Is there a detailed document on implementing Security for WebSpeed Applications in PAS for OpenEdge?

   https://community.progress.com/community_groups/openedge_deployment/m/documents/2882

9. Why is this document relevant only for OpenEdge releases 11.6.x?

   Starting with OpenEdge 11.7, there may be changes in the `appsecurity.xml` files to define properties for all the security modes.