Authentication Using Active Directory

Roundtable TSMS

Roundtable Development Team

Updated June 2022

Table of Contents

1 Overview

For some organizations, authenticating against Active Directory is a security requirement for all internal applications. While there are various means to achieve this integration with OpenEdge applications, this white paper highlights one approach to implementing Active Directory authentication for Roundtable TSMS.

2 Analysis

An application-independent user-defined authentication system that uses an ABL callback procedure can perform the required user authentication against Active Directory. The Data Administration tool can then be used to configure an authentication system that specifies the callback, thus ensuring that it will automatically execute to validate user credentials whenever the Roundtable application code asserts user identity.

By default, Roundtable uses the _user table to identify users and provide application security. Therefore, _user records must still exist even if the user is authenticated externally. To facilitate implementation, the callback procedure should also automatically create the _user records, as necessary.

3 Implementation

Our first step is to create a callback procedure. The sample callback procedure provided uses the .NET S*ystem.DirectoryServices.AccountManagement* package to facilitate authentication of user credentials against the Active Directory domain.

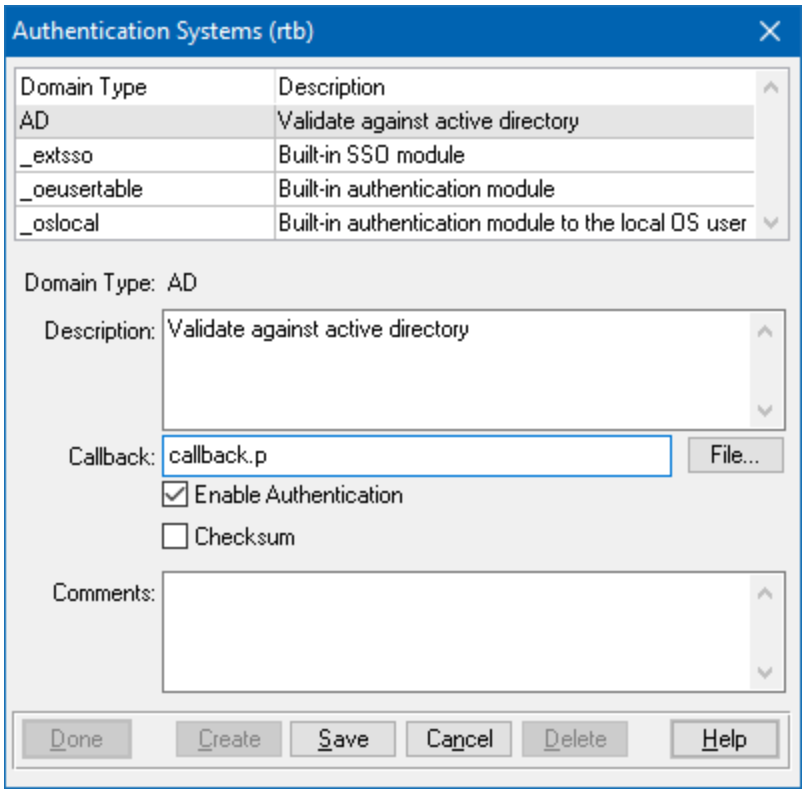
Next, using the OpenEdge Data Administration tool, we will then create a custom authentication system with its callback procedure referencing our callback procedure.

Finally, we will then create a security domain that uses the custom authentication system. Roundtable users will then use that domain when logging into Roundtable.

For more detailed information on ABL callback authentication procedures, please see the "*Creating an ABL Authentication Callback Procedure*" topic in the OpenEdge **Programming Interfaces > Application Security** section of the OpenEdge user guide.

3.1 Define Authentication System with Callback

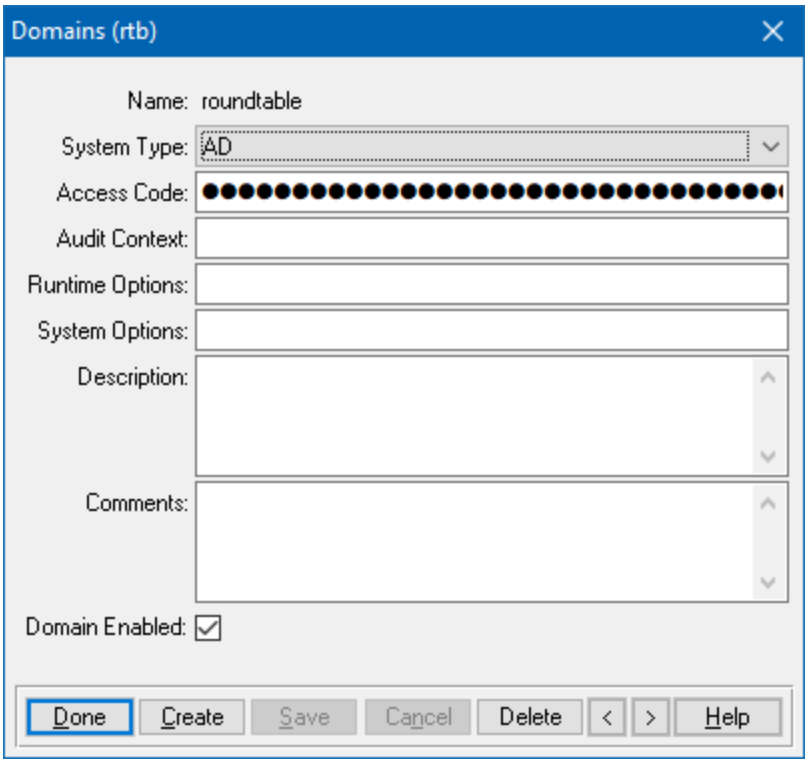
After connecting to the Roundtable database, open the OpenEdge Data Administration tool and choose **Admin > Security > Domain Maintenance > Authentication Systems** from the main menu to open the Authentication Systems dialog. Select **Create** to create a new authentication system.



Configure the new authentication system like the example above. As a best practice, the callback procedure should be placed in a directory in your PROPATH. Otherwise, an absolute path can be specified.

3.2 Create Security Domain

In the Data Administration tool, choose **Admin > Security > Domain Maintenance > Domains** to open the Domains dialog. Select **Create** to create a new security domain.



Configure the new domain system similar to the example above.

To work with the sample callback procedure, the Name of the domain should be the name of the Active Directory domain you wish to authenticate against. In this example, the name of the Active Directory domain is 'roundtable'.

The Access Code is an arbitrary value used when the client-principal object is sealed.

3.3 The Callback Procedure

```
/*
 callback.p
 Example callback routine that authenticates against an active directory
 Note: Assumes that domain provided is a valid active directory domain.
*/
ROUTINE-LEVEL ON ERROR UNDO, THROW.

USING Progress.Security.*.
USING System.DirectoryServices.AccountManagement.*.

LOG-MANAGER:LOGFILE-NAME = "callback.log".
/*LOG-MANAGER:CLEAR-LOG().*/

PROCEDURE AuthenticateUser:

  DEFINE INPUT  PARAMETER phCP            AS HANDLE              NO-UNDO.
  DEFINE INPUT  PARAMETER pcSystemOptions AS CHARACTER EXTENT    NO-UNDO.
  DEFINE OUTPUT PARAMETER piPAMStatus     AS INTEGER   INITIAL ? NO-UNDO.
  DEFINE OUTPUT PARAMETER pcErrorMsg      AS CHARACTER           NO-UNDO.

  DEFINE VARIABLE objContext AS PrincipalContext.
  DEFINE VARIABLE objUser    AS UserPrincipal.
```

```
  DEFINE BUFFER bUser FOR rtb._user.

  LOG-MANAGER:WRITE-MESSAGE('*** Executing AuthenticateUser ***').
  LOG-MANAGER:WRITE-MESSAGE('User is: ' + phCP:USER-ID).
  LOG-MANAGER:WRITE-MESSAGE('Domain is: ' + phCP:DOMAIN-NAME).
  LOG-MANAGER:WRITE-MESSAGE('Attemping to validate against active directory..').

  /*
   Domain name provided at login should be valid AD domain.
  */
  objContext = NEW PrincipalContext(ContextType:Domain,phCP:DOMAIN-NAME).

  IF objContext:ValidateCredentials(phCP:USER-ID,phCP:PRIMARY-PASSPHRASE) THEN DO:

    piPAMStatus = PAMStatus:Success.
    LOG-MANAGER:WRITE-MESSAGE('Success.').

    /*
     If user does not exist in _user table, create user account.
    */
    FIND bUser NO-LOCK
      WHERE bUser._userid = phCP:USER-ID NO-ERROR.

    IF NOT AVAILABLE(bUser) THEN DO TRANSACTION ON ERROR UNDO:

      LOG-MANAGER:WRITE-MESSAGE('Attemping to create _user record..').

      objUser = UserPrincipal:FindByIdentity(objContext,phCP:USER-ID).

      CREATE bUser.
      ASSIGN
        bUser._userid    = phCP:USER-ID
        bUser._user-name = objUser:NAME
        bUser._password  = ENCODE("").

      CATCH e AS PROGRESS.Lang.SysError:
        piPAMStatus = PAMStatus:Custom.
        pcErrorMsg = "Unable to create _user record. " + e:getMessage(1).
        LOG-MANAGER:WRITE-MESSAGE(pcErrorMsg).
      END CATCH.

      FINALLY:
        DELETE OBJECT e NO-ERROR.
      END FINALLY.

    END.

  END.
  ELSE DO:
    piPAMStatus = PAMStatus:Custom.
    pcErrorMsg = "You shall not pass!".
    LOG-MANAGER:WRITE-MESSAGE(pcErrorMsg).
  END.

  RETURN.

  FINALLY:
    DELETE OBJECT objContext NO-ERROR.
    DELETE OBJECT objUser    NO-ERROR.
  END FINALLY.

END. /* AuthenticateUser */

PROCEDURE AfterSetIdentity:

  DEFINE INPUT PARAMETER phCP         AS HANDLE           NO-UNDO.
  DEFINE INPUT PARAMETER pcSysOptions AS CHARACTER EXTENT NO-UNDO.

  LOG-MANAGER:WRITE-MESSAGE("*** Executing AfterSetIdentity ***").
```

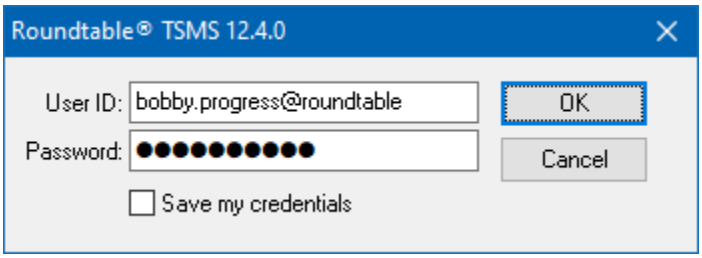
```
  LOG-MANAGER:WRITE-MESSAGE(SUBSTITUTE('QUALIFIED USER: &1', phCP:QUALIFIED-USER-ID)).
  LOG-MANAGER:WRITE-MESSAGE(SUBSTITUTE('LOGIN-STATE: &1', phCP:LOGIN-STATE)).
  LOG-MANAGER:WRITE-MESSAGE(SUBSTITUTE('LOGIN-STATE-DETAIL: &1', phCP:STATE-DETAIL)).
  LOG-MANAGER:WRITE-MESSAGE(SUBSTITUTE('SESSION ID: &1', phCP:SESSION-ID)).
  LOG-MANAGER:WRITE-MESSAGE(SUBSTITUTE('EXPIRATION: &1', phCP:LOGIN-EXPIRATION-TIMESTAMP)).
  LOG-MANAGER:WRITE-MESSAGE(SUBSTITUTE('DOMAIN TYPE: &1', phCP:DOMAIN-TYPE)).
  LOG-MANAGER:WRITE-MESSAGE(SUBSTITUTE('DOMAIN: &1', phCP:DOMAIN-NAME)).
  LOG-MANAGER:WRITE-MESSAGE(SUBSTITUTE('SEALED: &1', phCP:SEAL-TIMESTAMP)).

  RETURN.

  FINALLY:
    LOG-MANAGER:CLOSE-LOG().
  END FINALLY.

END.
```

To use the sample callback procedure provided, you must create an assemblies.xml file with the following content.

```
<?xml version="1.0" encoding="utf-8"?>
<references xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <assembly name="System.DirectoryServices.AccountManagement, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" />
</references>
```

3.4 Logging In

The final step is to begin using the new domain when logging into Roundtable.



With the domain specified, the callback procedure associated with the provided domain will execute, validate against Active Directory, and establish a session in Roundtable.