Progress®



Authenticate Users Defined in Microsoft Active Directory

OpenEdge®

Copyright

© 2021 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Defrag This, Deliver More Than Expected, DevReach (and design), Icenium, Inspec, Ipswitch, iMacros, Kendo UI, Kinvey, MessageWay, MOVEit, NativeChat, NativeScript, OpenEdge, Powered by Chef, Powered by Progress, Progress, Progress Software Developers Network, SequeLink, Sitefinity (and Design), Sitefinity, Sitefinity (and design), SpeedScript, Stylus Studio, Stylized Design (Arrow/3D Box logo), Styleized Design (C Chef logo), Stylized Design of Samurai, TeamPulse, Telerik, Telerik (and design), Test Studio, WebSpeed, WhatsConfigured, WhatsConnected, WhatsUp, and WS_FTP are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries.

Analytics360, AppServer, BusinessEdge, Chef Automate, Chef Compliance, Chef Desktop, Chef Habitat, Chef WorkStation, Corticon.js, Corticon Rules, Data Access, DataDirect Autonomous REST Connector, DataDirect Spy, DevCraft, Fiddler, Fiddler Everywhere, FiddlerCap, FiddlerCore, FiddlerScript, Hybrid Data Pipeline, iMail, JustAssembly, JustDecompile, JustMock, KendoReact, NativeScript Sidekick, OpenAccess, PASOE, Pro2, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Push Jobs, SafeSpaceVR, Sitefinity Cloud, Sitefinity CMS, Sitefinity Digital Experience Cloud, Sitefinity Feather, Sitefinity Insight, Sitefinity Thunder, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Supermarket, SupportLink, Unite UX, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

July 2021

Product version: Progress OpenEdge 12.4

Updated: 2021/06/30

Table of Contents

Part I: Get Started....11

Learn about Spring Security....11
Learn about the LDAP authentication process....12
Learn about authentication manager plug-ins....13
Learn about security tokens....15
Learn about OpenEdge security domains....15
Learn about CLIENT-PRINCIPAL objects....16

Part II: Configure....17

Connect to Microsoft Active Directory....17
Set up the Active Directory authentication manager plug-in....19
Set up the generic LDAP authentication manager plug-in....21
Add domains and domain access codes to domain registries....28
Set up URL access controls....29
Set up CLIENT-PRINCIPAL token generation....30
Secure the Tomcat Manager and OpenEdge Manager web applications....34

Preface

Purpose

This book provides security solutions for OpenEdge.

Audience

Anyone who is interested in OpenEdge security, including architects, developers, and IT administrators will find useful information in this manual to both plan and implement security for an application. Additionally, anyone who is interested in implementing an auditing solution will find details regarding planning for, implementing, and maintaining audit policies and data.

Organization

This guide consists of the following parts:

- Get Started introduces you to the basic components of OpenEdge that are involved with authenticating users who are defined in Microsoft Active Directory:
  - Learn about Spring Security on page 11 provides an overview of Spring Security and explains how it fits into the OpenEdge application architecture.
  - Learn about authentication manager plug-ins on page 13 introduces the Spring-provided components that handle the process of authenticating users defined in Active Directory.
  - Learn about security tokens on page 15 introduces the signed tokens that Spring produces when a user is successfully authenticated.
  - Learn about OpenEdge security domains on page 15 introduces the basic administrative component that you configure that defines the authentication system used in your OpenEdge environment.
  - Learn about CLIENT-PRINCIPAL objects on page 16 describes the characteristics of CLIENT-PRINCIPAL tokens that are derived from Microsoft Active Directory.
  - Learn about the LDAP authentication process on page 12 explains how the LDAP authentication process works in OpenEdge.

- Configure explains the process and steps to configure OpenEdge so that users who are defined in Microsoft Active directory can be authenticated for access to an ABL application:
  - Connect to Microsoft Active Directory on page 17explains how to validate your ability to connect to Active Directory directly.
  - Set up the Active Directory authentication manager plug-in on page 19 explains how to configure OpenEdge to use this authentication manager plug-in.
  - Set up the generic LDAP authentication manager plug-in on page 21 explains how to configure OpenEdge to use this authentication manager plug-in.
  - Add domains and domain access codes to domain registries on page 28 explains how to add your OpenEdge security domains and domain access codes in OpenEdge Database.
  - Set up URL access controls on page 29 explains how to set up the ABL application's URL authorization processes to grant or deny access to your deployment's APIs.
  - Set up CLIENT-PRINCIPAL token generation on page 30 explains how to load additional Active Directory account information into a CLIENT-PRINCIPAL token.

Documentation conventions

See Documentation Conventions for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.

Purpose

This book provides security solutions for OpenEdge.

Audience

Anyone who is interested in OpenEdge security, including architects, developers, and IT administrators will find useful information in this manual to both plan and implement security for an application. Additionally, anyone who is interested in implementing an auditing solution will find details regarding planning for, implementing, and maintaining audit policies and data.

Organization

This guide consists of the following parts:

- Get Started introduces you to the basic components of OpenEdge that are involved with authenticating users who are defined in Microsoft Active Directory:
  - Learn about Spring Security on page 11 provides an overview of Spring Security and explains how it fits into the OpenEdge application architecture.
  - Learn about authentication manager plug-ins on page 13 introduces the Spring-provided components that handle the process of authenticating users defined in Active Directory.
  - Learn about security tokens on page 15 introduces the signed tokens that Spring produces when a user is successfully authenticated.
  - Learn about OpenEdge security domains on page 15 introduces the basic administrative component that you configure that defines the authentication system used in your OpenEdge environment.
  - Learn about CLIENT-PRINCIPAL objects on page 16 describes the characteristics of CLIENT-PRINCIPAL tokens that are derived from Microsoft Active Directory.
  - Learn about the LDAP authentication process on page 12 explains how the LDAP authentication process works in OpenEdge.

- Configure explains the process and steps to configure OpenEdge so that users who are defined in Microsoft Active directory can be authenticated for access to an ABL application:
  - Connect to Microsoft Active Directory on page 17explains how to validate your ability to connect to Active Directory directly.
  - Set up the Active Directory authentication manager plug-in on page 19 explains how to configure OpenEdge to use this authentication manager plug-in.
  - Set up the generic LDAP authentication manager plug-in on page 21 explains how to configure OpenEdge to use this authentication manager plug-in.
  - Add domains and domain access codes to domain registries on page 28 explains how to add your OpenEdge security domains and domain access codes in OpenEdge Database.
  - Set up URL access controls on page 29 explains how to set up the ABL application's URL authorization processes to grant or deny access to your deployment's APIs.
  - Set up CLIENT-PRINCIPAL token generation on page 30 explains how to load additional Active Directory account information into a CLIENT-PRINCIPAL token.

Documentation conventions

See Documentation Conventions for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.

Get Started

For details, see the following topics:

- Learn about Spring Security
- Learn about the LDAP authentication process
- Learn about authentication manager plug-ins
- Learn about security tokens
- Learn about OpenEdge security domains
- Learn about CLIENT-PRINCIPAL objects

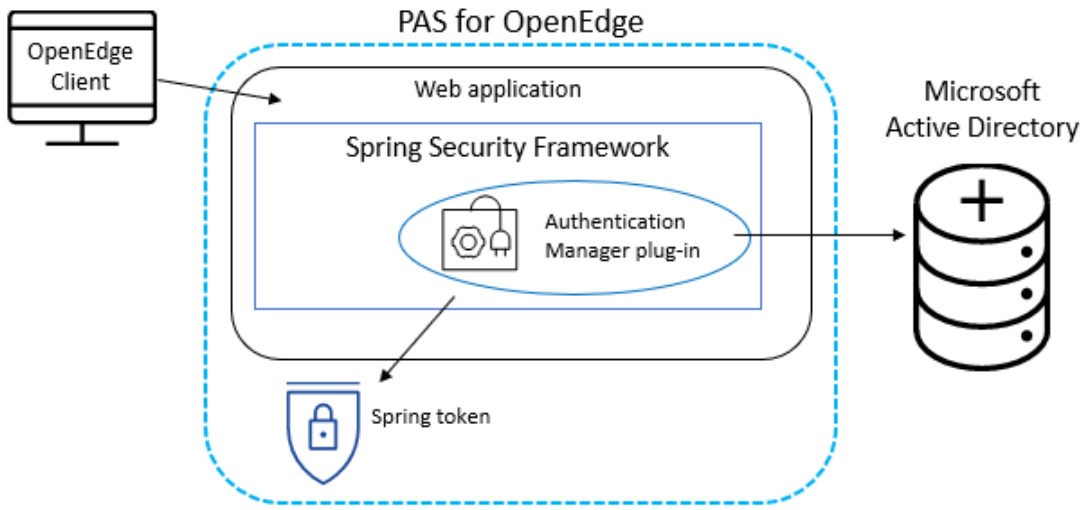
Learn about Spring Security

Web applications hosted on Progress Application Server (PAS) for OpenEdge automatically use the Spring Security framework to receive all incoming requests and to perform authentication and authorization operations for those web applications. Spring Security is a mature, commercial-grade, Java-based, and always-on access-control security framework that:

- Is automatically started when you start a PAS for OpenEdge instance
- Is always on to receive all incoming requests
- Applies industry-standard filters to ensure that each request is compliant
- Blocks a request if authentication fails

- Generates a security token if authentication succeeds
- Provides an easy-to-configure plug-in framework for multiple authentication provider types

When a client issues a request on a web application that is hosted on a PAS for OpenEdge instance, the request must go through the Spring Security framework. Spring Security invokes the configured authentication manager plug-in to authenticate the client's credentials against an identity store, such as Microsoft Active Directory. If the authentication operation is successful, then the Spring Security framework generates a Spring token, as shown in the following figure.

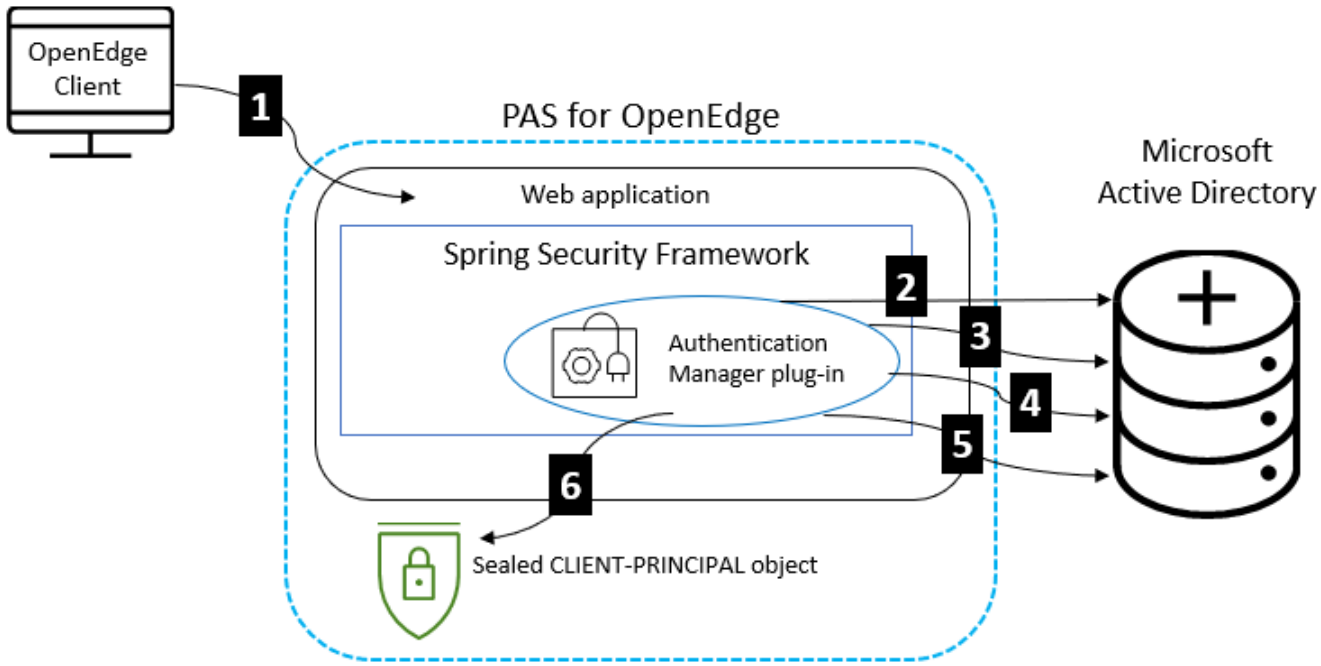


Spring Security is implemented in the *$CATALINA_HOME*/common/lib directory of the PAS for OpenEdge installation directory. For more information about Spring Security, see http://www.springsource.org.

Learn about the LDAP authentication process

The LDAP server authentication process in OpenEdge uses configuration properties specific to the enterprise. Understanding this process clarifies the purpose of the various configuration options, properties, and property values that you need to configure.

When a client request arrives in an ABL web application that is deployed on a PAS for OpenEdge instance, and the instance is configured with Active Directory, the authentication process used by either the Active Directory or generic LDAP authentication manager plug-in works as shown in the following figure.



1. The authentication manager plug-in obtains the LDAP user account identity needed for authentication. This may come from the client login information or by searching Active Directory using the client login information as the search key.
2. Connect to Active Directory, using an LDAP user account identity and password. Generate an exception if an error occurs.
3. Search for the Active Directory user account storage object to return account attributes that can be propagated into a `CLIENT-PRINCIPAL` object.
4. Search Active Directory for groups that the authenticated user account was granted access to by the Active Directory administrator. If no groups are found, then generate an exception.
5. Disconnect from Active Directory.
6. Propagate the Active Directory user account information to a sealed `CLIENT-PRINCIPAL` object.

The ABL application can then use the `CLIENT-PRINCIPAL` object for authorizing access to resources, such as PAS for OpenEdge services, OpenEdge database, or ABL application processes or data.

Learn about authentication manager plug-ins

A deployed ABL application has one or more ABL web application front-end subsystems, and one or more backend MS-Agent process sub-systems. They all operate as a single unit. When you set up Progress Application Server (PAS) for OpenEdge so that it can authenticate clients against the user account information stored in Microsoft Active Directory, your fundamental goals are:

- To have authentication and authorization decisions enforced at the client entry point into the ABL application, which is in the web application front end
- To use the Active Directory-stored user account information to authenticate a user and to authorize access by that user to the ABL web application

- To produce a `CLIENT-PRINCIPAL` object that can be passed to the ABL application so that that same identity can be used to manage the OpenEdge Database user, and therefore access to tables, fields, and so on

PAS for OpenEdge includes two Spring-implemented plug-ins that can satisfy the requirement to authenticate user identities that have accounts that are stored in Active Directory:

- Active Directory authentication manager—`ad`
- Generic LDAP authentication manager—`ldap`

An authentication manager plug-in performs the authentication process on behalf of an ABL web application that is hosted on PAS for OpenEdge, and returns a sealed token that contains the user's credentials and other information, as explained in Learn about security tokens on page 15.

When to choose the Active Directory authentication manager plug-in

The Active Directory authentication manager plug-in may be appropriate if the following conditions are met:

- Your PAS for OpenEdge instance authenticates client credentials (that is, username and password) using Windows Active Directory.
- Your ABL web application uses the Active Directory user account's `userPrincipalName` (UPN) attribute as its application user identity *and* OpenEdge domain. For example, the `userPrincipalName` `bobs@acme.com` consists of the `CLIENT-PRINCIPAL` user identity `bobs` in the OpenEdge domain `acme.com`.

When to choose the generic LDAP server authentication manager plug-in

The generic LDAP authentication manager plug-in is appropriate when you need more direct control of the authentication process interaction with Microsoft Active Directory, such as the following:

- User account class object schema
- User account storage location in Active Directory
- The mapping of a client's identity to information stored in the corresponding user account in Active Directory, such as an email address, full name, nickname, and so on
- The mapping of the Active Directory user account group object to `ROLE_names`
- LDAP group object storage location in Active Directory
- The ability to assign Active Directory user accounts to an OpenEdge domain

Most significant, if the ABL application is multi-tenant, then you must use the generic LDAP server authentication manager plug-in.

How to configure the authentication manager plug-in

You configure your chosen authentication manager plug-in using the SECPROP utility, which manages all Spring Security framework properties that are set in the `oeablSecurity.properties` file of your web application. You select your authentication manager plug-in by setting the `http.all.authmanager` property to one of the following:

- `ad`—Selects the Active Directory authentication manager plug-in
- `ldap`—Selects the generic LDAP authentication manager plug-in

Learn about security tokens

When the Spring Security framework successfully authenticates a client, Spring Security creates a security token that contains the user's credentials and other information about the user's roles and capabilities that is extracted from the user account information stored in Active Directory.

Security tokens:

- Are sealed, meaning that they are digitally signed so that they cannot be tampered with or transferred to another user
- Have an expiration date, which limits the opportunity for another to use that credential
- Can be passed to other systems and resources in OpenEdge on behalf of the client request, avoiding the need to re-authenticate the user

When PAS for OpenEdge receives a security token from Spring, OpenEdge generates a sealed `CLIENT-PRINCIPAL` object from that token so that it can be used in an ABL application. A `CLIENT-PRINCIPAL` object is a transformation of the Spring security token that includes ABL-specific details, which are primarily the OpenEdge domains and domain access keys.

For more information about `CLIENT-PRINCIPAL` objects, see Learn about CLIENT-PRINCIPAL objects on page 16.

Learn about OpenEdge security domains

In OpenEdge, the authentication system used to validate users is identified by an OpenEdge security domain—or, simply, domain. A domain is a grouping of user accounts and resources, such as servers, applications, or collection of applications, whose security systems are configured to create and trust a common security token for authentication and authorization. A domain also allows all authenticated users to share other common attributes, such as a common tenancy in a multi-tenant database and (through domain authorization settings) access to a common set of database and application resources.

In OpenEdge, a security domain is configured with:

- A name.
- The authentication system used to authenticate the identity of all users who are members of the domain. For example, Microsoft Active Directory.
- Common attributes associated with all authenticated users that are specified.
- A secret access code used to cryptographically seal a security token after a successful user authentication, and also used to later validate the user identity represented by the sealed security token.
- An indication of whether the domain is enabled for use at run time.

An OpenEdge security domain is specified within a domain registry. The domain registry contains a list of individual security domain names, and their associated domain access codes, as well as other details that are contextual to the OpenEdge product with which the domain registry is configured, from which an identity can be authenticated. Each OpenEdge product that seals a `CLIENT-PRINCIPAL` object, and every OpenEdge product that can validate a `CLIENT-PRINCIPAL` seal, has a domain registry.

A domain registry functions much like a filter. When an authenticated user presents a `CLIENT-PRINCIPAL` token, the domain and domain access code within that token are compared to the domains and access codes listed in the domain registry. Access is permitted only if the domain and access code are a match with the contents in the domain registry.

The configuration requirements for each domain registry, including where the domain registry is stored, differ from one OpenEdge product to another. For example:

OpenEdge product	Domain registry configuration and storage
PAS for OpenEdge	The domain registry is stored is an encrypted keystore file that contains a set of domain name and domain access code pairs.
OpenEdge Database	The domain registry is contained in two database tables.
OpenEdge Authentication Gateway	The domain registry consists of two files: • The `domains.JSON` file, which defines the configuration of each domain and how it authenticates its users and performs SSO operations • An encrypted keystore file that that containsa set of domain name and domain access code pairs

In a multi-tenant database environment, the domain also identifies the tenancy of all users whose identity is authenticated in the domain.

For more information about managing identities in OpenEdge domains used with PAS for OpenEdge, see "What is Identity Management?" in *Learn about Identity Management*.

Learn about CLIENT-PRINCIPAL objects

As mentioned in Learn about security tokens on page 15, a `CLIENT-PRINCIPAL` object is a transformation of the Spring-generated security token that contains additional details for use in an ABL application.

`CLIENT-PRINCIPAL` objects:

- Are generated by OpenEdge from the Spring-generated token
- Add a domain and domain access code
- Add ABL-specific details
- Are required for assigning a connection identity for an OpenEdge multi-tenant database

When OpenEdge successfully authenticates a client identity, OpenEdge then seals the `CLIENT-PRINCIPAL` object. After the `CLIENT-PRINCIPAL` object is sealed, you cannot change the user identity (or any other settings) in the object, but you can use the sealed object in single sign-on (SSO) operations to validate and assign the identity it represents to any session or database connection in an ABL application. Therefore, your application can use a `CLIENT-PRINCIPAL` object to both authenticate a user identity for one application resource and assign that identity to other ABL sessions and database connections in the application.

For more information, see "Client-principal objects" in *OpenEdge Programming Interfaces*.

II

Configure

For details, see the following topics:

- Connect to Microsoft Active Directory
- Set up the Active Directory authentication manager plug-in
- Set up the generic LDAP authentication manager plug-in
- Add domains and domain access codes to domain registries
- Set up URL access controls
- Set up CLIENT-PRINCIPAL token generation
- Secure the Tomcat Manager and OpenEdge Manager web applications

Connect to Microsoft Active Directory

The first step in configuring an authentication manager plug-in for Microsoft Active Directory is to validate your ability to connect to Active Directory directly. Always do this before attempting to authenticate a user through either PAS for OpenEdge or OpenEdge Authentication Gateway. As a best practice, Progress recommends that you make this initial connection using an LDAP browser product, which also allows you to view the Active Directory structure so that you can understand how user account, group information, and other details in Active Directory are structured. Understanding this structure is essential to being able to map the user account and group information to the specific properties that you need to configure in the authentication manager plug-in.

Obtain an LDAP browser

There are a number of freely available LDAP browsers that you can use to examine the structure and contents of Microsoft Active Directory. For example, Microsoft provides Active Directory Explorer. There are also open source LDAP browsers, such as JXplorer. Download and install the one that you prefer.

Obtain the Active Directory host details

You need the Active Directory server host name, IP address, port number, and required TLS connection details to connect to an Active Directory host used in your organization. You can obtain these details from the network administrator, or by completing the following steps on Windows:

1. Open a command prompt.
2. Use the SET command to view the definition of the `LOGONSERVER` environment variable. This variable contains the definition of the Active Directory host name.

   For example, the following command shows that the Active Directory host name in use is `NTMADACME`:

   ```
   C:\Users\jsmith>SET LOGONSERVER
   LOGONSERVER=\\NTMADACME
   ```

Test your Active Directory connection

To test your Active Directory connection:

1. From a command prompt, enter the `WHOAMI /UPN` command to obtain your Active Directory user ID,for example:

   ```
   C:\Users\jsmith>WHOAMI /UPN
   jsmith@acme.com
   ```

   Note: Your Active Directory ID is associated with your Windows login ID and is accessed using your Windows password.

2. Start your LDAP browser, and connect to Active Directory using the following login settings. (You might need to verify some of these connection details with your network administrator.)
   - The Active Directory host name.
   - The port number used by your organization. The default is 389.
   - The LDAP version number used for the connection. Progress recommends using `LDAP v3`.
   - The security level with which you want to connect. For example, choose `User + Password` to log in using your Active Directory user ID and password.
   - Your Active Directory user ID and your Windows password.

Obtain the Active Directory server's rootDN

From the LDAP browser, search for the Active Directory root distinguished name (DN), also called simply **rootDN**. This information is typically available in Microsoft Active Directory from the rootDSE. The rootDSE contains the `defaultNamingContext` attribute, which contains the rootDN and is the top-level branch in Active Directory from which you search for user account and group information.

For information about locating the rootDSE, see the Microsoft Active Directory information at the following location:

https://docs.microsoft.com/en-us/windows/win32/adsi/active-directory-service-interfaces-adsi

Set up the Active Directory authentication manager plug-in

If you have determined that the Active Directory authentication manager plug-in meets your ABL application's requirements, then configure the required and optional plug-in properties in the *instance-name*/webapps/*webapp-name*/WEB-INF/oeablSecurity.properties file for your ABL web application as explained in this topic.

Required properties

Property	Description
ad.ldap.url	The fully qualified URL of the LDAP connection to the Active Directory host using the following syntax: {ldap|ldaps}://*host-name*[:*port*] Use the ldaps URL scheme if your organization requires the Active Directory connection to use a secure TLS connection. (Use ldap only if the Active Directory host does not support TLS connections.) To facilitate failover high availability, you can specify multiple LDAP URLs as space-separated values in a single string.
ad.ldap.rootdn	The rootDN of the Active Directory service that contains the user accounts and group information that is used for authentication.
ad.user.domain	The userPrincipalName domain value that becomes automatically appended to the client's login user ID if the client does not provide one. For example, if the user ID supplied by the client is jsmith, then the string @acme.com could be appended as the userPrincipalName value. The default value is "", which corresponds to the blank domain. However, if you specify this property, but enter no corresponding value for it, then no userPrincipalName domain is appended to the input user ID, and the client is forced to specify a fully qualified domain, for example, jsmith@acme.com.

Optional properties

Property	Description
ad.AuthoritiesMapper.prefix	The prefix that is used to distinguish a Spring role name from a user account name when validating URL access to an application resource. By default, the prefix ROLE_ is used to remove ambiguity between a user ID name and a role name. (It is a Spring Security default convention, but its use is not a requirement.) Change the prefix when you need to have role names that are exactly as they are defined in Active Directory.
ad.AuthoritiesMapper.convertToUpperCase	The property that forces Active Directory group object names to be converted to uppercase when Spring Security creates a role name. The default value is true. Change this value to false when the role names that get placed in the CLIENT-PRINCIPAL object, and are used for the URL access controls, must be exactly as they are defined in Active Directory. Note that the uppercase conversion of the LDAP group attribute name is a default convention and is not a requirement.

Example

The following shows an example of the configuration of the Active Directory authentication manager plug-in:

```
http.all.authmanager=ad
ad.user.domain=acme.com
ad.ldap.url=ldap://vm-pasoeldap:389
ad.ldap.rootdn=dc=anytown,dc=acme,dc=com
```

In this example, note the property descriptions in the preceding configuration:

Property	Description
http.all.authmanager	Specifies ad, which corresponds to the Active Directory authentication manager plug-in.
ad.user.domain	Specifies the userPrincipalName domain, acme.com, that gets automatically appended to the client-supplied user ID, if one is not provided by the client.

Property	Description
`ad.ldap.url`	Specifies the fully qualified URL of the LDAP connection to the Active Directory host, including port number: `ldap://vm-pasoeldap:389`
`ad.ldap.rootdn`	Specifies the rootDN of the Active Directory service where user account information is defined: `dc=anytown,dc=acme,dc=com`.

The effect of this example configuration is to search the Active Directory service for the client-supplied user ID, starting at the specified rootDN, `dc=anytown,dc=acme,dc=com`. If the client does not provide a domain specification, then `acme.com` is appended.

Set up the generic LDAP authentication manager plug-in

If you determined that the generic LDAP authentication manager plug-in meets your ABL application's requirements, then configure the required and optional plug-in properties in the *`instance-name`*`/webapps/`*`webapp-name`*`/WEB-INF/oeablSecurity.properties` file for your ABL web application, as explained in this topic.

Required properties

Property	Description
`ldap.url`	The fully qualified URL of the LDAP connection to the Active Directory host using the following syntax: `{ldap|ldaps}://`*`host-name`*`[:`*`port`*`]` Use the `ldaps` URL scheme if your organization requires the Active Directory connection to use a secure TLS connection. (Use `ldap` only if the Active Directory host does not support TLS connections.) To facilitate failover high availability, you can specify multiple LDAP URLs as space-separated values in a single string.
`ldap.rootdn`	The rootDN of the Active Directory service that contains the user accounts and group information that is used for authentication.

Property	Description
`ldap.manager-dn`	A valid Active Directory account ID that is used to bind to Active Directory and search for the corresponding LDAP user account object DN of the user being authenticated. **Note:** The Active Directory account ID is a `userPrincipalName` object, for example, `safe-user@acme.com`.
`ldap.manager-password`	The password for the user that corresponds to the `ldap.manager-dn` property, described in the preceding row. For production environments, Progress strongly recommends that you specify this password as an encrypted value that you have obtained using the *`DLC`*`/bin/stspwdutil` utility. For details,"Strong Password Encryption Utility (stspwdutil)" in the *OpenEdge Authentication Gateway Guide*.

Optional properties

Property	Description
`ldap.usersearch.base`	The LDAP object DN where the search for LDAP user account objects is initiated. The default value is blank, which causes all user searches in Active Directory to be initiated from the RootDN object. You can specify a valid LDAP server object DN (for example, a subtree) where the search starts. Setting a value for this property is appropriate when Active Directory has more than one account with exactly the same name, but in different subtrees.

Property	Description
`ldap.usersearch.filter`	LDAP search specification for locating the LDAP user account object that corresponds to the client's login user ID. The default value for Active Directory is `( | (userPrincipalName={0}) (sAMAccountName={0}) (mail={0}) (cn={0}))`. Use this property to choose, at login time, which login ID type you want to use, allowing you to narrow the search for a given user account in Active Directory. For example, you could use this when want users to log in using only one ID type, such as their email name, or you want them to log in using some other kind of ID, such as their surname. The portion of the filter specification that designates the client's user ID must consist of the string `{0}`.
`ldap.usersearch.searchSubtree`	A boolean value that specifies whether to search the LDAP object hierarchy, starting at the user object search base DN, for an LDAP user account object that corresponds to the login user ID. The default value is `true`. If your search base in Active Directory is at a level where there are multiple subtrees of user accounts, you can set this property to `false` and use it together with `ldap.usersearch.base` to search for users within a specific subtree only.
`ldap.groupsearch.base`	The LDAP object DN where the search for LDAP group objects is initiated. A blank value initiates all searches from the rootDN object. Otherwise, this must be a valid LDAP server object DN. The default value is `""`. Setting a value for this property is appropriate when Active Directory has more than one group object with exactly the same name, but in different subtrees.

Property	Description
`ldap.groupsearch.filter`	The LDAP search specification to find the LDAP group objects that correspond to the LDAP user account's granted roles. The default value is `(&(objectclass=group)(member={0}))`. The portion of the filter specification that designates the client's user ID must consist of the string `{0}`. Use this property to refine the selection of groups that a given user account is a member of. An account is a member of a group.
`ldap.grouprole.attribute`	The LDAP group object attribute name from which to derive the role name used in Spring and in OpenEdge security. The value cannot be empty. See the Active Directory schema definition for alternatives to the usual `cn` attribute name. The default value is `cn`. Change this when you want to use a different attribute of the group as the role name for the user.
`ldap.authpopulator.searchSubtree`	A boolean value to specify whether to search the LDAP object hierarchy, starting at the `ldap.groupsearch.base`, when finding LDAP group objects. The default value is `true`.
`ldap.authpopulator.ignorePartialResultException`	A boolean value to specify whether to ignore Active Directory notifications if not all the results of an LDAP search are available. The default value is `true`. Change this when you detect that you are not receiving group objects in the user's list of roles, most likely owing to the the termination in Active Directory of further searching due to an excessive number of notifications having been returned.
`ldap.authpopulator.convertToUpperCase`	A boolean value to specify whether to follow LDAP server object referrals. The default value is `true`. Change this value to `false` when the role names that get placed in the `CLIENT-PRINCIPAL` object, and are used for the URL access controls, must be exactly as they are defined in Active Directory.

Property	Description
`ldap.contextSrc.referral`	Specifies whether to ignore LDAP referrals; that is, soft-links to LDAP object subtree storage in other Active Directory servers. The default value is `ignore`. Change this to `follow` when you want searches for an Active Directory user account to follow referrals. Note that following referrals may be slow, and can, in some directory services like Active Directory, return a `PartialResultsException`.

Property	Description
ldap.contextSrc.timeout	The number of milliseconds to wait for an Active Directory connection or read operations to complete. The purpose of this attribute is to prevent user logins from hanging, which results in hanging a user's login to an application. The value specified must be a valid positive integer. The default value is 5000. Change this when Active Directory account login operations may require more time than the default interval.
OEClientPrincipalFilter.loadAccntAttrList OEClientPrincipalFilter.domainRoleFilter OEClientPrincipalFilter.validateClientDomain	These properties apply to process of converting tokens that are generated by the Spring Security framework into CLIENT-PRINCIPAL tokens. These properties allow you to select attributes of the user account in Active Directory and add them as properties of the CLIENT-PRINCIPAL token. In a multi-tenant architecture, the OEClientPrincipalFilter.domainRoleFilter property allows you to derive a domain name from the attributes assigned to a user account in Active Directory. Using these properties allows you to set up pattern matching expressions as part of the direct login operations to determine, based on the groups that a user account as a member of, the group name that is used to extract the OpenEdge domain name. The OEClientPrincipalFilter.validateClientDomain property is used when a CLIENT-PRINCIPAL token is received as a SSO token, and you need to validate that the user is correctly a member of the corresponding domain by verifying the appropriate user account attributes in Active Directory . For more information about setting up domain role filters, see Set up CLIENT-PRINCIPAL token generation on page 30. Note that the OEClientPrincipalFilter property is enabled by default, which is required for generating CLIENT-PRINCIPAL objects.

Example

The following shows an example of the configuration of the generic LDAP authentication manager plug-in:

```
http.all.authmanager=ldap
ldap.url=ldap://vm-pasoeldap:10389
ldap.manager-dn=uid=admin,ou=system
ldap.manager-password=secret
```

```
ldap.root.dn=anytown,dc=acme,dc=com
ldap.grouprole.attribute=cn
ldap.groupsearch.filter=(uniqueMember={0})
ldap.usersearch.filter=(uid={0})
```

In this example, note the property descriptions in the preceding configuration:

Property	Description
`http.all.authmanager`	Specifies the generic LDAP authentication manager plug-in, `ldap`.
`ldap.url`	Specifies the fully qualified URL of the LDAP connection to the Active Directory host, including the port number: `ldap://vm-pasoeldap:10389`.
`ldap.manager-dn`	Specifies the Active Directory account ID that is used for searching for the LDAP user account object DN of the client user being authenticated. In this example, the user account ID is `admin`, which is in the organizational unit `system`.
`ldap.manager-password`	Specifies the password for the user account ID that corresponds to the `ldap.manager-dn` property.
`ldap.root.dn`	Specifies the rootDN of the Active Directory service where user account information is defined: `dc=anytown,dc=acme,dc=com`.
`ldap.grouprole.attribute`	Specifies the LDAP group object attribute, `cn`, that is used to derive the role name to be used in the Spring and `CLIENT-PRINCIPAL` tokens that are generated.
`ldap.groupsearch.filter`	Specifies the LDAP search string for locating the LDAP group objects that correspond to the LDAP user account's granted roles. In this example, `(uniqueMember={0})` indicates that
`ldap.usersearch.filter`	Specifies the LDAP search string for locating the LDAP user account object that corresponds to the client's login user ID. In this example, `(uid={0})` results in

The effect of this configuration is for this authentication manager plug-in to do the following when a client provides a user ID:

1. Bind to the Active Directory service using the `admin` account DN and password, and set the rootDN to operate in.
2. Starting at the rootDN, apply the LDAP user search filter to locate the user account being authenticated.

   If the user is not found, then generate an error, and unbind from the Active Directory service.
3. Obtain the user account object's DN, and unbind.
4. Bind to the Active Directory service using the user account DN that is being authenticated, including password.

   If a bind error occurs, then return an authentication error and unbind.

5. Load the user account object's attributes, and filter by the specified property list.
6. Starting at the rootDN, search the object tree to locate all group objects that have the user account's DN as a member.
7. From the group object attribute name, extract the root role name that will be appended to the configured role prefix (`ROLE_`).
8. If an OpenEdge security domain is configured, then extract the domain name for the user, if the user has not supplied one on login.
9. Unbind.
10. Generate a `CLIENT-PRINCIPAL` token from user account information and role names.

Add domains and domain access codes to domain registries

In a very basic deployment environment in which you have a web application hosted on a PAS for OpenEdge instance, and an OpenEdge Database instance backend, you need to create two security domain configurations. One is for the PAS for OpenEdge instance, and the other is for OpenEdge Database. Therefore, each security domain configuration must have an individual domain registry. The domain registry includes the names of the domains from which authenticated users may obtain access to resources, and the corresponding domain access codes.

Domain access codes

A domain access code is a secret value used by OpenEdge or your ABL application both to cryptographically seal a security token during user authentication and to validate a user's sealed security token. The domain access code is typically a long character string value known only to the domain configuration. When setting up domain registries, you need to make sure that for a given domain name that is entered into each domain registry, the domain access code is identical.

Set up the domain registry for PAS for OpenEdge

To set up the domain registry for a PAS for OpenEdge instance:

1. Update the default domain registry CSV file, `oeablSecurity.csv`, located by default in the PAS for OpenEdge instance folder of the OpenEdge working directory, to include your domain names and corresponding domain access codes.
2. Use the `gendomreg` utility to create an encrypted Java keystore file from the CSV file.
3. Copy the encrypted keystore file to the `conf` directory of each PAS for openEdge instance.

For more information, see "Enable ABL application authentication" in *Manage Progress Application Server (PAS) for OpenEdge*.

Set up the domain registry for OpenEdge Database

You typically configure a OpenEdge security domains for OpenEdge Database using database administration tools, including the:

- Database Administration Console in OpenEdge Management
- Data Administration utility in the OpenEdge program group on Windows
- Admin menu of the character-mode Data Dictionary

For more information, see "Defining and configuring security domains" in *Learn about Identity Management*.

After you have configured your authentication manager plug-in, you need to add your OpenEdge security domains and domain access codes in OpenEdge Database.

Set up URL access controls

The `CLIENT-PRINCIPAL` token that is generated by PAS for OpenEdge contains the Active Directory user account's granted roles in its `roles` attribute. The `CLIENT-PRINCIPAL` role attribute is used by the ABL web application's URL authorization processes to grant or deny access to your deployment's APIs. When configured effectively, URL authorization can significantly reduce the volume of ABL application code and strengthen the application's overall security. Using best-practice designs, each ABL web application contains its own configuration of the `CLIENT-PRINCIPAL` token's role that is allowed to execute a specific HTTP method on a specific transport's URI.

Because of the implementation of its enterprise-strength security, Active Directory is a trusted source of a user's role information. Enterprises define their own role names in their Active Directory services and individually control the granting and denying of the roles each user account has.

You can use two possible approaches for working with Active Directory role names:

- You can use the existing enterprise's role names, and adapt your deployment's security configuration to use those names.
- You can define your deployment's built-in role names, and collaborate with the Active Directory administrator to use them; that is, to grant your deployment's roles to the existing user accounts in Active Directory.

OpenEdge does not include tools for managing those roles, but the role information is stored in a `.CSV` file format that is supported by a variety of third-party tools that you can use to tailor the information. Because most OpenEdge-based products have their own installation and tailoring tools, Progress recommends that you choose the one best fit for your deployment.

The default role names shipped in your deployment's ABL web application's `.CSV` file, `WEB-INF/oeablSecurity.csv`, will be replaced by your enterprise's role names in, for example:

Active Directory role name	Default oeablSecurity.csv role name	Updated oeablSecurity.csv role name
`enterprise-user`	`ROLE_PSCUser`	`ROLE_ENTERPRISE-USER`
`oe-admin`	`ROLE_PSCAdmin`	`ROLE_OE-ADMIN`
`role-oeuser`	`ROLE_PSCUser`	`ROLE_ROLE-OEUSER`
`PSCUser`	`ROLE_PSCUser`	`ROLE_PSCUSER`
`PSCAdmin`	`ROLE_PSCAdmin`	`ROLE_PSCADMIN`

Note: The reason for setting roles names in the updated `oeablSecurity.csv` file to all uppercase is to ensure that duplicate role names are avoided. That way, any authentication manager plug-in will validate against the uppercase roles.

Set up CLIENT-PRINCIPAL token generation

When a user is successfully authenticated by the Spring authentication manager, OpenEdge creates a sealed `CLIENT-PRINCIPAL` token that is returned to the ABL application for its own use and for getting access to OpenEdge database connections. Note that there are a few differences between the way that the generic LDAP authentication manager and the Active Directory authentication manager work to produce the `CLIENT-PRINCIPAL` token.

CLIENT-PRINCIPAL token expiration

By default, `CLIENT-PRINCIPAL` tokens have no expiration interval. However, you can set the `LOGIN-EXPIRATION -TIMESTAMP` attribute to specify when a `CLIENT-PRINCIPAL` token expires. When a `CLIENT-PRINCIPAL` token expires, the token can no longer represent a user identity, and any started login session for the object ends. Note that OpenEdge evaluates the value of this attribute only at the times when OpenEdge either attempts to seal or validate the object.

About loading additional Active Directory account information into a CLIENT-PRINCIPAL token

The generic LDAP authentication manager loads all defined Active Directory user account attributes as part of its authentication process. Those attributes are not automatically added to the `CLIENT-PRINCIPAL` token that is generated by PAS for OpenEdge. However, there may be some instances when the LDAP user account attributes are relevant to your ABL application, and you can selectively control which attributes to add into the `CLIENT-PRINCIPAL` token's properties list.

You can specify these additional attributes by using the `OEClientPrincipalFilter.loadAccntAttrList` property in the configuration of the generic LDAP authentication manager. These attributes correspond to the definition of the `inetOrgPerson` LDAP object class in RFC 2798, available at https://tools.ietf.org/html/rfc2798. You can specify the following user attributes to this property:

Attribute	Description
`cn`	Common name for the object instance
`dn`	Distinguished name
`sn`	Surname
`uid`	User ID
`mail`	Electronic address
`employeeNumber`	Employee identification number
`title`	Employee job title
`ou`	Organization name
`o`	An organization within an enterprise
`givenName`	Employee's first name
`displayName`	Employee's preferred name (nickname)

Attribute	Description
homePhone	Home telephone number
homePostalAddress	Home postal address
initials	Employee's initials
mobile	Mobile phone number
postalAddress	Organization's postal address
postalCode	Organization's postal code
roomNumber	Office number at the organization's facility
street	Organization's street address
telephoneNumber	Office telephone number
token_type	Type of token

Note the following when you specify the OEClientPrincipalFilter.loadAccntAttrList property:

- The default value is an asterisk (*), which causes all attributes to be loaded into the CLIENT-PRINCIPAL token properties.
- You can specify mutliple attributes as a comma-separated list.
- You can specify a blank to load no additional account attributes.
- If a specified attribute is not present in the account object in Active Directory, then nothing is loaded into the CLIENT-PRINCIPAL token properties for that attribute.

How the CLIENT-PRINCIPAL token role attribute is used

The CLIENT-PRINCIPAL token's role attribute is used by the PAS for OpenEdge ABL application URL authorization process to control access to APIs.

Support for multiple OpenEdge domains or tenants

Active Directory, like any LDAP server, is not multi-tenant enabled. It is an identity store for a single enterprise. When your ABL application uses a single OpenEdge domain or tenant, you can configure CLIENT-PRINCIPAL properties to use that single domain name and supply its access code in the domain registry file. However, when your ABL application *does* support multiple OpenEdge domains or multiple tenants, you must perform additional configuration work to ensure that a client cannot spoof access to tenant data to which they are not authorized.

Active Directory user accounts do not have attribute data that identifies the specific tenant of which they are members. However, when you use the generic LDAP authentication manager, you can assert direct control over the specific OpenEdge domain or tenant to which an authenticated user account is assigned or validated against. You can do this by using LDAP group memberships as a way to allow the Active Directory administrator to control user membership in an OpenEdge domain or tenant. This uses the same principle whereby an LDAP user account is a member of an LDAP group membership and is authorized to access restricted data.

The OpenEdge domain or tenant is a collection of related users who are authorized to access restricted data. Therefore, by using a domain and LDAP groups, you can accomplish the same type of user grouping for authorization to tenant data. With LDAP group memberships, you can use an LDAP group to:

- Designate the OpenEdge domain or tenant to which a simple client-provided user ID (for example, `dan`) must be assigned before a `CLIENT-PRINCIPAL` token can be created and sealed.
- Validate that a fully qualified client-provided user ID (for example, `dan@acme.com`) is a member of the `acme.com` OpenEdge domain or tenant.

PAS for OpenEdge provides the ability to control the mapping of user IDs to domains and ensure correct tenant database access. You can do this by using the LDAP user account's granted group objects, which are mapped to Spring Security `ROLE_*` roles, to designate the domain in which the user is a member. Using LDAP group objects in Active Directory in this way maintains the secure central administration of enterprise user accounts and allows you to designate the domain name of which each Active Directory user account is a member.

The key to making this work is to create an LDAP group object in Active Directory for each OpenEdge domain. Then, each time the Active Directory administrator adds a user account, they must add that account to the LDAP group that represents the OpenEdge domain or tenant of which the account owner is a member. When that user account control process is in place, you can then configure PAS for OpenEdge to assign and validate the user's tenancy automatically.

To set up LDAP group objects in Active Directory for use and an OpenEdge domain:

1. Establish a naming format that allows OpenEdge and your application to easily distinguish between a role used for URL authorization and one that is used as the OpenEdge domain, for example:

   `OED_`*`openedge-domain-name`*

   In this example, `OED_` is a prefix that designates that the specified name is related to an OpenEdge domain. The string *`openedge-domain-name`* corresponds to the domain name that is configured in the PAS for OpenEdge instance's domain registry as a valid domain name that includes an access code. The prefix you choose does not need to be `OED_`. It can be anything that the enterprise decides to use. OpenEdge strips this prefix for you before the ABL application receives the `CLIENT-PRINCIPAL` token that is generated.

2. Work with the Active Directory administrator to create an LDAP group for each OpenEdge domain that is configured in your application's OpenEdge servers. The administrator must use the designated naming format for the group object's common name (`CN`) subfield of its `DN`. Then, the administrator must assign a specific OpenEdge domain group object membership for each LDAP user account as they would usually do.

3. Configure PAS for OpenEdge to recognize the designated naming format for the domain. You do this by using a standard, regular-expression, group-pattern-matching mechanism. Using the example in Step 1, you designate the regular expression group pattern:

   ```
   ROLE_OED_(.*)
   ```

   In the preceding search pattern:

   - `ROLE_` is the Spring Security prefix that is prefixed to LDAP group object `CN` field values to designate it for use in URL authorization.
   - `OED_` is the prefix chosen to distinguish the LDAP group that specifically represents an OpenEdge domain.
   - The remaining characters in the pattern represent the OpenEdge domain name that is used in the CLIENT-PRINCIPAL token domain property.

Specify a domain role filter in the authentication manager

The primary use case for the domain role filter is to support multi-tenancy in OpenEdge. If you deploy a multi-tenant ABL application or configure a multi-tenant database, note that Windows itself is not multi-tenant; however, role name mapping can be used to support multi-tenancy. When you configure the generic LDAP authentication manager, you can specify the properties listed and described in the following table to obtain OpenEdge domain names from Active Directory and to validate a user's membership in an OpenEdge domain:

Property	Description
`OEClientPrincipalFilter.domainRoleFilter`	This property is provided as a security measure to prevent a user from spoofing their membership in an OpenEdge domain or to provide a domain name to an unauthorized user. When you use this property, you specify a pattern using the following syntax: `ROLE_`*`domain-identifier`*`(.*)` • The string *`domain-identifier`* represents the unique domain name. • The pattern must include the substring `(.*)`, which designates where the domain name characters are located in the role names that are extracted from Active Directory. • A unique domain identifier string should be used in the role name to designate that the role name contains OpenEdge domain name information. • The default value for this property is `""`. When an OpenEdge domain name is missing from the client-provided user ID, this property causes PAS for OpenEdge to derive the user's OpenEdge domain membership from the LDAP user account's role names. PAS for OpenEdge also uses the specified pattern to validate that the client user ID's domain name corresponds to an LDAP user account's role.
`OEClientPrincipalFilter.validateClientDomain`	This propery validates that the client's user ID is mapped to a specific OpenEdge domain through role names that are loaded from Active Directory.

Secure the Tomcat Manager and OpenEdge Manager web applications

OpenEdge Manager is a Java web application that provides a REST API for remote administration of the OpenEdge web applications and multi-session agents. OpenEdge Manager duplicates the administration API supported by the JMX interface from Tomcat, but it uses JSON input/output payloads instead. The `oemanager.war` file is required to use REST APIs to manage and monitor ABL applications and the PAS for OpenEdge instances on which they are hosted.

However, if you use either Tomcat Manager or OpenEdge Manager for remote administration, Progress strongly recommends that you take the following actions to ensure the security of the PAS for OpenEdge instance and deployed web applications:

1. Limit access to the Tomcat Manager and OpenEdge Management endpoint URLs that enable management of the PAS for OpenEdge instance.
2. Change the default passwords for the Tomcat server.

Limit access to URLs used for management

You control access to the endpoint URLs of the Tomcat Manager or OpenEdge Manager web applications that are used for remote management of a PAS for OpenEdge instance by means of the Remote Address Valve class. You configure the Remote Address Valve on Tomcat Manager or OpenEdge Manager by specifying one or more Java RegEx (`java.util.regex`) expressions. When a client attempts to access a Tomcat Manager or OpenEdge Manager endpoint URL, the IP address of that client is evaluated against the RegEx expressions to determine whether the client request can continue or be refused.

To configure the Remote Address Valve for a Manager or OpenEdge Manager endpoint URL:

1. Open the following file in a text editor:

   ```
   CATALINA_BASE/webapps/[manager|oemanager]/META-INF/context.xml
   ```

2. Uncomment, if necessary, the `<Valve>` element that specifies the `RemoteAddrValve` class, and specify a Java RegEx expression for the `allow` property that limits access to only the permitted IPv4/IPv6 `localhost` address.

   For example:

   ```
   <Valve className="org.apache.catalina.valves.RemoteAddrValve"
          allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1"/>
   ```

Note the following:

- Remember to use Java RegEx expressions when you need to adjust the `allow` property values.
- By default, the `deny` property is disabled. Consequently, you can control access simply by means of the `allow` property, and the `denyStatus` HTTP response status code that is used when rejecting a denied request is `403`.
- When you configure the `RemoteAddrValve` class correctly, the `HTTP-403` status code is returned when an unauthorized client attempts to access the `/manager` or `/oemanager` endpoints.

Change the default passwords

Changing the default passwords in Tomcat involves two separate procedures:

- Updating the PAS for OpenEdge instance in the *`CATALINA_BASE`*`/conf/tomcat-users.xml` file

- Updating the administrator credentials in the OpenEdge Management Console

To change the default passwords:

1. Create a backup of the following files:
   - *CATALINA_BASE*/conf/server.xml
   - *CATALINA_BASE*/conf/tomcat-users.xml
2. Open *CATALINA_BASE*/conf/server.xml in a text editor.
3. Disable the default UserDatabase realm definition by specifying the off value. For example:

```
<!-- feature:begin:UserDatabase:off
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
            resourceName="UserDatabase" />
 feature:end:UserDatabase:off -->
```

4. Create and enable a new feature definition for UserDatabase that uses the CredentialHandler class. For example:

```
<!-- feature:begin:UserDatabase-pbkdf2:on -->
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
            resourceName="UserDatabase" >
          <CredentialHandler
className="org.apache.catalina.realm.SecretKeyCredentialHandler"
               algorithm="PBKDF2WithHmacSHA512"
               iterations="10000"
               saltLength="16"
               keyLength="256" />
        </Realm>
<!--     feature:end:UserDatabase-pbkdf2:on -->
```

   In the preceding example:

   - pbkdf2 is FIPS compliant and is also the strongest that Tomcat currently supports.
   - The more iterations that you specify slows the speed with which passwords are calculated.

5. Save your changes to *CATALINA_BASE*/conf/server.xml.
6. Change to the *DLC*/servers/pasoe/bin directory.
7. Run the digest utility to generate a new user password. For example:

```
digest -a PBKDF2WithHmacSHA512 -i 10000 -s 16 -k 256 -h
"org.apache.catalina.realm.SecretKeyCredentialHandler" password
```

   In the preceding example:

   - The parameters for generating the password need to exactly match those that are specified for the CredentialHelper class in server.xml.
   - The string *password* is a plain-text password.

8. Allow the program a moment to run. Then observe the output, which should be displayed as <*password*>:<*hash*>.
9. Open *CATALINA_BASE*/conf/tomcat-users.xml in an editor, and make the following changes:
   - Comment out any username definitions that are not to be used by your application or organization.
   - Update the tomcat user, or create a new username, providing the <*hash*> value from the previous step as the password.

- Make sure that your target user definition has been provided the roles of `"ROLE_PSCAdmin,ROLE_PSCOper,ROLE_PSCUser"`.

10. Start (or restart) your PAS for OpenEdge instance to put all of the changes to the application's security into effect.

After you set the new password for the PAS for OpenEdge instance, you must update the OpenEdge Management Console with new administrator credentials.

To update the administrator credentials in the OpenEdge Management Console:

1. Start the OpenEdge Management Console, if necessary.
2. Under the Resources section, navigate to the PAS for OpenEdge instance.
3. Select the **Configuration** tab from the menu on the left.
4. Enter the new password, in plain text, in both the **Tomcat manager password** and **OpenEdge manager password** fields.

   If you change the administrator username, then update the respective login fields as well.
5. Save your changes, then click **Test Connection** at the bottom of the page to confirm your changes.

   The updated passwords are encoded in the *DLC*`/properties/pasmgr.properties` file as `tomcatMgrPassword` and `webPassword`.

Best practice

As a sanity test, you can first try creating a hash of the default password `tomcat` and testing your OpenEdge Manager endpoint URL using the default credentials. This can ensure that the password you enter, using the prompt for credentials, performs as expected. Then hash a new password and update the `tomcat-users.xml` file, restart your PAS for OpenEdge instance, and confirm that the old password is rejected and that the new password is accepted.

Note: PAS for OpenEdge includes a lockout mechanism for accounts for which an incorrect password is entered too many times. By default, the lockout occurs after 5 incorrect passwords are entered. You can set the lockout value in the `psc.as.realm.failureCount` property in the *CATALINA_BASE*`/conf/catalina.properties` file. Because so many internal services attempt to use the default `tomcat` account, changing this password in the PAS for OpenEdge instance, but not in the calling programs, can cause the lockout mechanism to be triggered quickly; consequently, the account is locked out for 300 seconds. As a workaround, while you work on updating passwords elsewhere, you can temporarily change the `failureCount` property to a higher number and restart the PAS for OpenEdge instance. Then after you have changed all of the passwords for your default administrator account, restore this value to the default and restart the PAS for OpenEdge instance.
