Oracle® Fusion Middleware

Securing Resources Using Roles and Policies for Oracle WebLogic Server



26 (26.1.0.0.0)
G56949-01
November 2026



Oracle Fusion Middleware Securing Resources Using Roles and Policies for Oracle WebLogic Server, 26 (26.1.0.0.0)

G56949-01

Copyright © 2007, 2026, Oracle and/or its affiliates.

Primary Author: Oracle Corporation

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Contents

Preface

Audience i
Documentation Accessibility i
Diversity and Inclusion i
Related Documentation ii
Conventions ii

1 Introduction

2 Understanding WebLogic Resource Security

Overview of Securing WebLogic Resources 1
- Using Policies to Protect Multiple Resources 3
  - Protecting Resources by Type 3
  - Protecting a Hierarchy of Resources 3

Designing Roles and Policies for WebLogic Resources: Main Steps 4
- Best Practices: Conditionalize Policies or Conditionalize Roles 6
- Best Practices: Configure Entitlements Caching When Using WebLogic Providers 6

3 Resource Types You Can Secure with Policies

Administrative Resources 1
Application Resources 3
COM Resources 4
EJB Resources 4
Enterprise Information Systems (EIS) Resources 5
Java DataBase Connectivity (JDBC) Resources 5
- JDBC Operations 5

Java Messaging Service (JMS) Resources 6
- JMS Operations 6

Java Naming and Directory Interface (JNDI) Resources 7
- Supported Token Types 7

JMX Resources 7

Copyright © 2007, 2026, Oracle and/or its affiliates.

Maintaining a Consistent Security Scheme 9
NoSQL Resources 10
Server Resources 10
Permissions for the weblogic.Server Command and the Node Manager 11
Permissions for Using the weblogic.Server Command 11
Permissions for Using the Node Manager 12
URL Resources 12
Web Service Resources 12
Work Context Resources 13
Coherence Resources 14

4 Options for Securing Web Application and EJB Resources

Deployment Descriptors Not Required 2
Comparison of Security Models for Web Applications and EJBs 2
Discussion of Each Model 3
Metadata Annotations 3
Deployment Descriptor Only Model 4
Custom Roles Model 5
Custom Roles and Policies Model 6
Advanced Model 6
Understanding the Advanced Security Model 7
Disabling a Globally Attached OWSM Policy 7
Understanding the When Deploying Web Applications or EJBs Setting 8
How the Check Roles and Policies and When Deploying Web Applications or EJBs Settings Interact 9
Understanding the Combined Role Mapping Enabled Setting 10
Usage Examples 11
Securing Web Applications and EJBs 12

5 Security Policies

Security Policy Storage and Prerequisites for Use 1
Default Root Level Security Policies 2
Security Policy Conditions 3
Basic Policy Conditions 3
Example of Using the @Policy and @Policies JWS Annotations 4
Context Element Policy Conditions 6
Attaching OWSM Security Policies to JAX-WS Web Services 6
Using to Manage Security Policies 7

Copyright © 2007, 2026, Oracle and/or its affiliates.

6 Users, Groups, And Security Roles

Overview of Users and Groups 1
Default Users 1
Default Groups 2
Run Time Groups 3
Revised configWss_Service.py 3
Overview of Security Roles 4
Types of Security Roles: Global Roles and Scoped Roles 4
Default Global Roles 5
Security Role Conditions 7
Basic Role Conditions 8
Date and Time Role Conditions 8
Context Element Role Conditions 9
Using to Manage Users, Groups, and Roles 10

7 Using XACML Documents to Secure WebLogic Resources

Prerequisites 1
Adding a XACML Role or Policy to a Realm: Main Steps 2
Caution: Indeterminate Results Can Lock Out All Users 2
Determine Which Resource to Secure 2
Get the ID of the Resource to Secure 3
Create XACML Documents 4
Example: Defining Role Assignments 4
Example: Defining Authorization Policies 6
Configuring Transport-Level Security Through Policy: Main Steps 6
Verify That Your Roles and Policies Are in the Realm 7
Creating Roles and Policies for Custom MBeans 8
Determine the Resource IDs for a Custom MBean 8
Exporting Roles and Policies to XACML Documents 9

A Revised configWss.py

Comparison of WebLogic Server and XACML Security Models A-2
Comparison of Terminology A-2
Description of Data Types A-3
Action Identifiers A-3
Examples A-5
Updating the JWS File with @Policy and @Policies Annotations A-6
Examples A-6
Policy and PolicySet Identifiers A-7

Copyright © 2007, 2026, Oracle and/or its affiliates.

Examples A-7
Resource Identifiers A-7
Example of Adding Security to a JAX-WS Web Service A-8
Subject Identifiers A-16
Examples A-16
WebLogic Server Functions for XACML A-16
Configuring STS URI for SAML: Standalone Client A-17
Examples A-17
Secure Conversation Policies A-17
Example A-18
Time/Date Conversions A-19
Arithmetic Conversions and Functions A-21
Object Type Conversions A-24
Object Comparisons A-26
String Comparisons and Manipulations A-28
Rule and Policy-Combining Algorithm A-29

Copyright © 2007, 2026, Oracle and/or its affiliates.

Preface

This documentation describes how to use security roles and policies in Oracle WebLogic Server 26 to determine who can access resources in a domain.

Audience

This document contains information that is useful for security architects and security administrators who are designing a security strategy for resources within a WebLogic Server domain. It includes information about resource types, options for securing Web applications and EJBs, different types of security roles and policies, and the components of a role and policy.

It is assumed that the reader is familiar with Jakarta Security and the other features of the WebLogic Security Service.

The information in this document is relevant during the design and development phases of a software project. This document does not address production phase administration topics. For links to WebLogic Server documentation and resources related to these topics, see Related Documentation .

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Related Documentation

Use the reference books as and when it is required for better understanding.

Other WebLogic Server documents that may be of interest to security administrators wanting to secure WebLogic resources are:

- *Understanding Security for Oracle WebLogic Server*—Summarizes the features of the WebLogic Security Service, including an overview of its architecture and capabilities. It is the starting point for understanding WebLogic security.
- *Administering Security for Oracle WebLogic Server*—Describes how to ensure that security is comprehensively configured for a WebLogic Server installation, including information about security providers, identity and trust and SSL.
- Security Policies and Roles in *Oracle WebLogic Remote Console Online Help* Provides instructions for using the to complete the tasks described in this document.

These documents provide additional information about specific resource types:

- Securing Web Applications, Securing Enterprise JavaBeans (EJBs) and Using Java Security to Protect WebLogic Resources in *Developing Applications with the WebLogic Security Service*
- Configuring Access Control in *Developing JCOM Applications for Oracle WebLogic Server* (COM resources)
- Security in *Developing Resource Adapters for Oracle WebLogic Server* (EIS resources)
- *Securing WebLogic Web Services for Oracle WebLogic Server*

Tutorials and Samples

Additional security documents are listed in Code Examples and Sample Applications in *Understanding Oracle WebLogic Server*.

New and Changed WebLogic Server Features

For a comprehensive listing of the new WebLogic Server features introduced in this release, see *What's New in Oracle WebLogic Server*.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Copyright © 2007, 2026, Oracle and/or its affiliates.

1

Introduction

The WebLogic Security Service combines several layers of security features to prevent unauthorized access to your WebLogic Server domains.
This document describes using roles and policies to determine who can access resources in a domain. The roles and policies feature fulfills the same function as the familiar Access Control List (ACL), but offers an improvement over ACLs: an ACL is static while roles and policies specify conditions under which users can access resources, and these conditions are evaluated at run time.

Copyright © 2007, 2026, Oracle and/or its affiliates.

2

Understanding WebLogic Resource Security

Learn the terms and concepts related to securing WebLogic Server resources and understand the main steps you can follow to create roles and policies to secure them.
This chapter includes the following sections:

- Overview of Securing WebLogic Resources
- Designing Roles and Policies for WebLogic Resources: Main Steps

Overview of Securing WebLogic Resources

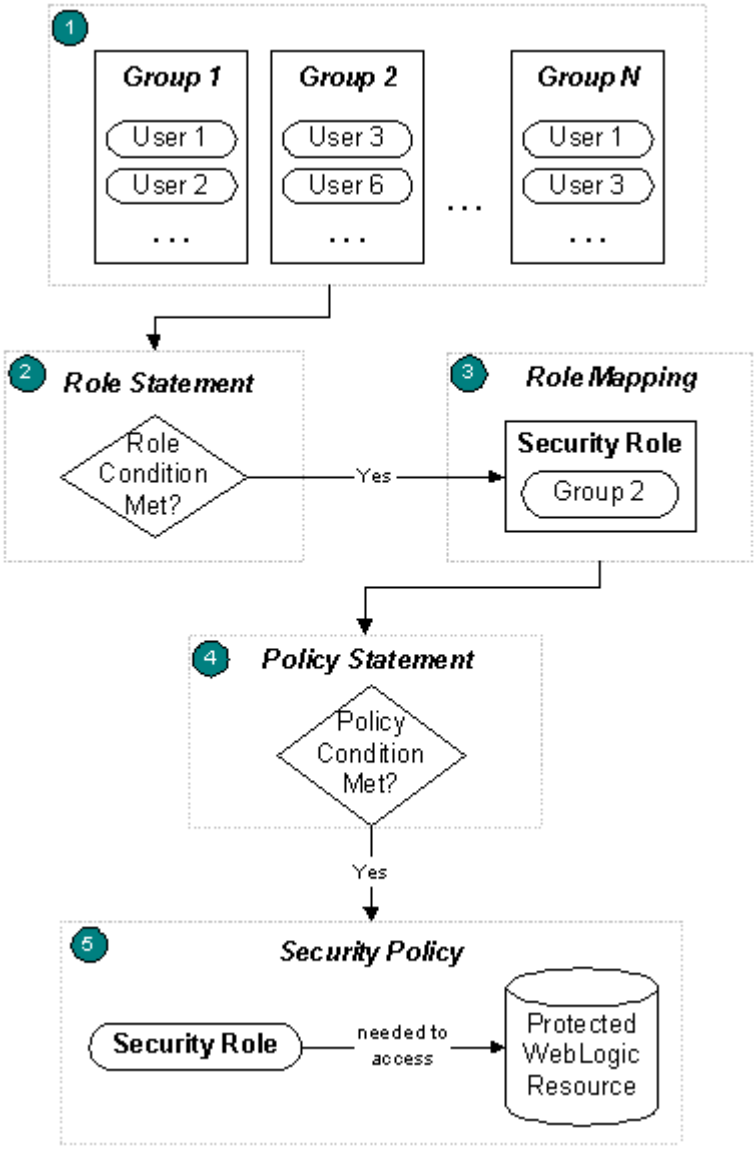
To secure a resource in a WebLogic Server domain, you create a policy and an optional role. A **resource** is an entity (such as a Web Service or a server instance) or an action (such as a method in a Web Service or the act of shutting down a server instance). A **policy** specifies which users, groups, or roles can access the resource under a set of conditions. A security **role**, like a security group, grants an identity to a user. Unlike a group, however, membership in a role can be based on a set of conditions that are evaluated at run time.

> **Note**
>
> Jakarta Security requires that group principal names are mapped to roles of the same name by default. In WebLogic Server, if the `security-role-assignment` element in the `weblogic.xml` deployment descriptor does not declare a mapping between a security role and one or more principals in the WebLogic Server security realm, then the role name is used as the default principal.

Figure 2-1 describes how you create roles and policies and how the Security Service uses them to determine whether a client can access a resource. A brief explanation follows the figure.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 2-1 How a Policy Grants Access to a Resource



1. Before creating security policies and roles, Administrators statically assign users to groups, which can represent organizational boundaries. The same user can be a member of multiple groups. Figure 2-1 shows three groups with two users each. User 1 and User 3 are members of multiple groups.

   Oracle recommends assigning users to groups because doing so increases efficiency for administrators who work with many users.

2. Administrators create a security role based on their organization's established business procedures. The security role consists of one or more conditions, which specify the circumstances under which a particular user, group, or other role should be granted the security role.

3. At run time, the Security Service compares the groups against the role condition(s) to determine whether users in the group should be dynamically granted a security role. This process of comparing groups to roles is called role mapping. In Figure 2-1, Group 2 is the only group that is granted a security role.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Individual users can also be granted a security role, but this is a less typical practice.

4. Administrators create a security policy based on their organization's established business procedures. The security policy consists of one or more policy conditions which specify the circumstances under which a particular security role should be granted access to a WebLogic resource.
5. At run time, the WebLogic Security Service uses the security policy to determine whether access to the protected WebLogic resource should be granted. Only users who are members of the group that is granted the security role can access the WebLogic resource. In Figure 2-1, User 3 and User 6 can access the protected WebLogic resource because they are members of Group 2, and Group 2 is granted the necessary security role.

Using Policies to Protect Multiple Resources

WebLogic Server provides two techniques for using a single policy to protect a collection of resources, described in the following sections:

- Protecting Resources by Type
- Protecting a Hierarchy of Resources

Protecting Resources by Type

You can create a policy that protects all resources of a specific type. Such policies are called **root-level policies**. For example, you can create a root-level policy for the Web Service type. All Web Services that you deploy in the domain for which you have defined this root-level policy will be protected by the root-level policy.

If you define a policy for a specific Web Service, then the Web Service will be protected by its own policy and will ignore the root-level policy.

Protecting a Hierarchy of Resources

All of the resources within a Jakarta EE application or module that you deploy exist within a hierarchy, and policies on resources higher in the hierarchy act as default policies for resources lower in the same hierarchy. Policies lower in a hierarchy always override policies higher in the hierarchy.

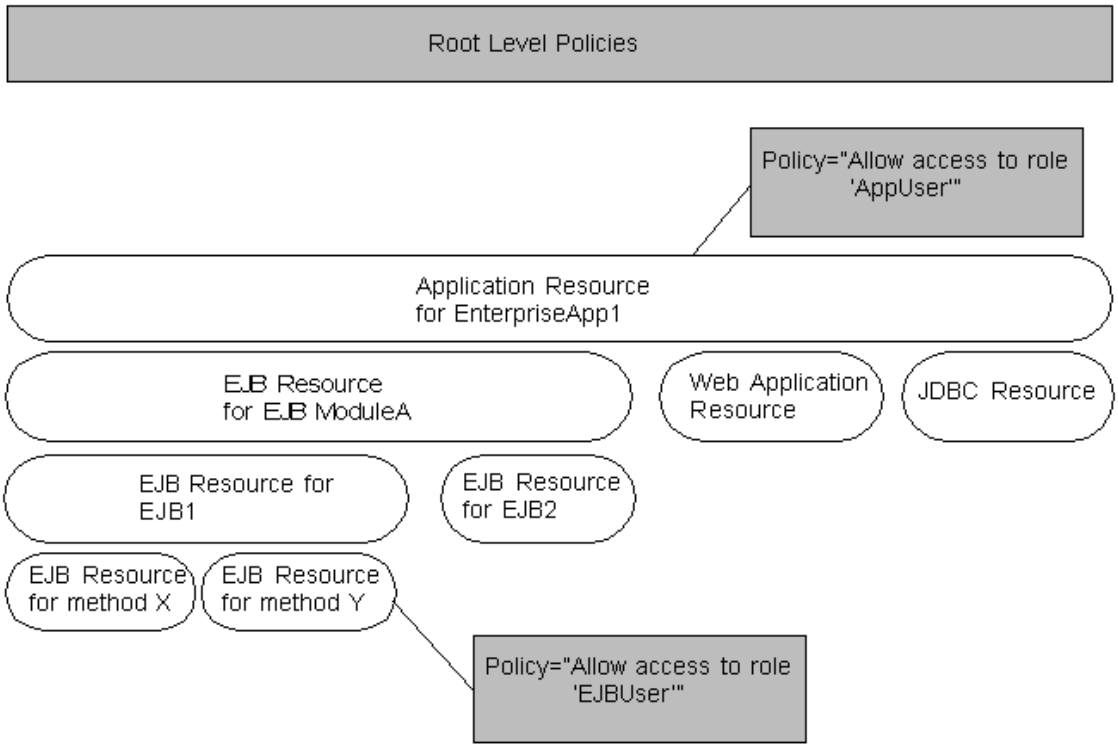
For example, EnterpriseApp1contains EJB ModuleA along with a Web application and a JDBC module (see Figure 2-2). You create a policy for EnterpriseApp1 and for method Y within EJB ModuleA. When an EJB client attempts to invoke method Y, the WebLogic Security Service enforces the specific policy and ignores the policy for the enterprise application.

When a client requests access to EJB method X (which is not protected by its own policy), the WebLogic Security Service asks:

- Is there a policy for this EJB method? No, therefore go to the next higher level in the hierarchy.
- Is there a policy for the EJB that contains this method? No, therefore go to the next higher level in the hierarchy.
- Is there a policy for the EJB module that contains the method's parent EJB? No, therefore go to the next higher level in the hierarchy.
- Is there a policy for the enterprise application that contains this EJB method? Yes, use it. (If there were no such policy, the Security Service would have used the default root-level policy for EJBs.)

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 2-2 Hierarchy of Resources and Policies



Designing Roles and Policies for WebLogic Resources: Main Steps

To secure WebLogic resources, define roles and policies to a list of resources in the domain. For this purpose, determine the kind of policies you want to create and analyze the user groups and roles associated with the group.

To design a set of roles and policies that can secure the resources in your domain:

1. List all of the resources that will be in your domain and determine which ones should be accessed only by specific users or groups.

   To see a list of all the types of resources that could be in any given domain, see Resource Types You Can Secure with Policies.

   For planning purposes, organize the resources into the following categories:

   - Server resources, administrative resources, JMX resources, and Jakarta NoSQL resources. Server resources determine who can start and stop server instances. Administrative resources determine who can complete such tasks as unlocking users who have been locked out of their accounts, uploading files (used during deployment), and viewing the domain and server logs. JMX resources determine who can change the configuration of servers, clusters, machines, and other components that are defined in the domain's configuration document (config.xml). NoSQL resources determine who can access globally configured NoSQL resources from applications deployed to WebLogic Server.

     For these tasks, WebLogic Server already provides a detailed, layered security scheme that grants different types of access to eight security roles (Admin, Deployer,

Copyright © 2007, 2026, Oracle and/or its affiliates.

Operator, Monitor, Anonymous, AppTester, CrossDomainConnector, AdminChannelUser). For most environments, this security scheme is adequate and only requires you to assign users to the eight default security roles appropriately (see step 3).

While it is possible to modify some parts of this layered security scheme, such modifications are usually not needed and require careful planning to maintain consistency between the different layers. See Administrative Resources, Server Resources, JMX Resources, and NoSQL Resources.

- Web application resources and EJB resources, which determine who can access the Web applications and EJBs that you deploy in your domain.

  The Jakarta EE platform already provides a standard model for securing Web applications and EJBs. In this standard model, developers define role mappings and policies in the Web application or EJB deployment descriptors.

  You can use the standard model or you can use the to define policies and roles, which offers unified and dynamic security management. See Options for Securing Web Application and EJB Resources.

- All other resources, which determine who can access the business logic and business content in the enterprise applications and other modules that you deploy or otherwise configure for the domain.

  By default, most of these resources are not protected by policies; you must define policies to determine who can access them. Jakarta NoSQL resources have a default policy that allows access to everyone. To restrict access to NoSQL resources, define policies that specify who can access them.

2. For each type of resource that you want to secure, determine if you need to create root-level policies, scoped policies, or a combination of both.

   A **root-level** policy applies to all instances of a resource type. For example, if you define a root-level policy for the Web Services resource type, then the policy will apply to all Web Services in your domain.

   A **scoped** policy applies to a specific resource instance and **overrides** a root-level policy.

   See Security Policies.

3. Analyze your users and the resources that you want them to access. Organize users into security groups and roles as follows:

   - Add any user that you want to start and stop servers or to engage in other administrative tasks to one of the eight default global roles. The WebLogic Server security scheme allows only the eight global roles to perform many of these tasks.
   - For other users (that you do **not** want to access administrative or server resources but you do want to access other resources for which you have defined policies), create additional security groups and roles. Because role membership can be granted at run time, you can place users or groups in roles based on business rules or the context of the request.

     You can create **global** roles, which can be used in any policy, or **scoped** roles, which can be used only in a policy for a specific resource instance.

   See Users, Groups, And Security Roles.

4. Use the to create users, groups, roles, and policies:

   a. To create the users and groups, see Users and Groups in *Oracle WebLogic Remote Console Online Help*.

Copyright © 2007, 2026, Oracle and/or its affiliates.

b. To create security roles and policies, see Security Policies and Roles in *Oracle WebLogic Remote Console Online Help*.

Best Practices: Conditionalize Policies or Conditionalize Roles

Because both roles and policies can evaluate a set of conditions at run time, you should consider which parts of your security data should be static and which should be dynamic. For example, you might want some policies to always allow one specific role to access a resource, and then you use conditions in the role's definition to move users in and out of the roles as needed. In other cases, you might want a static role definition and create a policy that allows access to different roles at different times of the day.

As a general guideline, if you base the authorization decision on the resource instead of the entities (roles) who can access the resource, you would add conditions to the security policy. If you base authorization on who can access the resource, then you would add conditions to the security role.

For an example of authorization based on who can access the resource, consider a manager who is going on vacation. You can temporarily place a user in a `Manager` security role. Dynamically granting this security role means that you do not need to change or redeploy your application to allow for such a temporary arrangement. You simply specify the hours between which the temporary manager should have special privileges. Further, you do not need to remember to revoke these special privileges when the actual manager returns as you would if you temporarily added the user to a management group.

Best Practices: Configure Entitlements Caching When Using WebLogic Providers

The WebLogic Authorization provider, known as the `DefaultAuthorizer`, and the WebLogic Role Mapping provider, referred to as the `DefaultRoleMapper`, improve performance by caching the roles, predicates, and resource data that they look up. If you modify your realm to use these WebLogic providers, you can configure the maximum number of items that they store in the caches.

In WebLogic Server 14.1.1.0.0, the WebLogic Authorization provider (`DefaultAuthorizer`) and the Role Mapping provider (`DefaultRoleMapper`) are deprecated and will be removed in a future release.

> **Note**
>
> By default, security realms in newly created domains include the XACML Authorization and Role Mapping providers. The XACML providers use their own cache, but this cache is not configurable. The WebLogic Authorization provider (`DefaultAuthorizer`) and the Role Mapping provider (`DefaultRoleMapper`) are the only WebLogic-provided security providers with configurable caches of entitlement data.

By default, the Weblogic Authorization and Role Mapping providers store the following number of items in each cache:

- 2000 items in the roles cache

  This cache contains the name of each role that has been looked up and the policy that protects it.

Copyright © 2007, 2026, Oracle and/or its affiliates.

- 200 items in the predicates cache

  This cache contains each predicate that the WebLogic entitlements engine has looked up.

- 5000 items in the resources cache

  This cache contains the name of each resource that has been looked up and the policy that protects it.

If a cache exceeds its maximum size, the WebLogic entitlements engine removes the least recently used (LRU) item from the cache.

If the applications on a WebLogic Server instance use more than 2000 roles or 5000 resources, consider increasing the cache sizes. (The WebLogic providers include less than 50 predicates, so there is no need to increase the size of this cache.)

To change the maximum number of items that a cache contains, pass one of the following system properties in the `java` startup command for a WebLogic Server instance:

- `-Dweblogic.entitlement.engine.cache.max_role_count=max-roles`

  where `max-roles` is the maximum number of roles that you want to cache.

- `-Dweblogic.entitlement.engine.cache.max_predicate_count=`*`max-predicates`*

  where *`max-predicates`* is the maximum number of predicates that you want to cache.

- `-Dweblogic.entitlement.engine.cache.max_resource_count=`*`max-resources`*

  where *`max_resource_count`* is the maximum number of resources that you want to cache.

By default, the WebLogic providers add items to the cache as they use them. With this configuration, the initial lookup of entitlement data takes longer than subsequent lookups. You can, however, decrease the amount of time needed for an initial lookup by configuring a WebLogic Server instance to load the caches during its startup cycle. To do so, pass the following system property to the server's `java` startup command:

```
-Dweblogic.entitlement.engine.cache.preload=true
```

For example:

```
java -Dweblogic.entitlement.engine.cache.max_role_count=6001 -
Dweblogic.entitlement.engine.cache.max_resource_count=3001 -
Dweblogic.entitlement.engine.cache.preload=true weblogic.Server
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

3

Resource Types You Can Secure with Policies

Learn about the types of resources that you can secure using policies in WebLogic Server. This chapter includes the following sections:

- Administrative Resources
- Application Resources
- COM Resources
- EJB Resources
- Enterprise Information Systems (EIS) Resources
- Java DataBase Connectivity (JDBC) Resources
- Java Messaging Service (JMS) Resources
- Java Naming and Directory Interface (JNDI) Resources
- JMX Resources
- NoSQL Resources
- Server Resources
- URL Resources
- Web Service Resources
- Work Context Resources
- Coherence Resources

Administrative Resources

Policies for administrative resources are different from the policies defined for regular users. Administrative resources have special privileges such as uploading files during deployment, viewing the domain and server logs, and unlocking users, and so on.

Policies for administrative resources determine who can complete such tasks as uploading files (used during deployment), viewing the domain and server logs, and unlocking users who have been locked out of their accounts.

For the most security-sensitive of these tasks, users must first be authorized by additional policies on a JMX resource (see Figure 3-1). For information about JMX resources and how to design roles and policies for activities that are protected by multiple resources, see JMX Resources.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 3-1 Some Policies Overlap

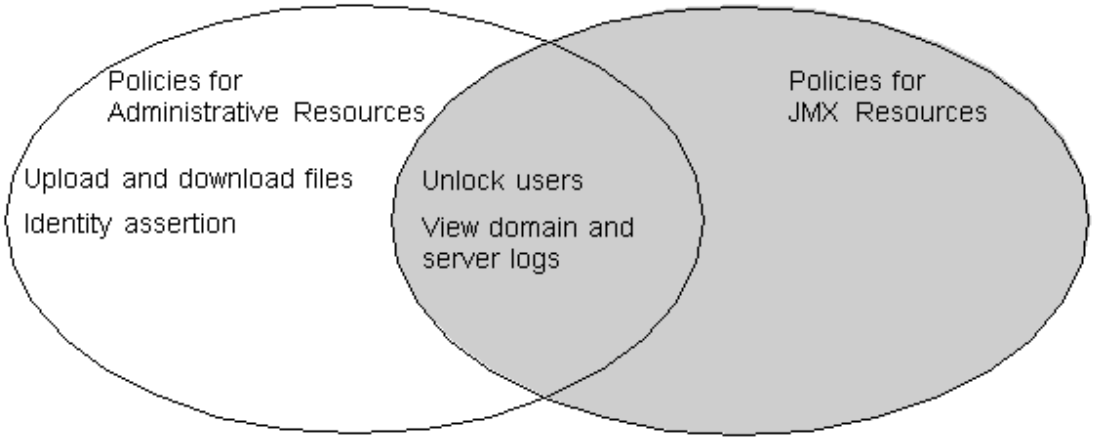


Table 3-1 describes the administrative activities that administrative resources protect and which of these activities are also protected by additional JMX resources. For activities that are protected by multiple resources, the default policy in the JMX resource duplicates the protections in the Administrative resource.

Table 3-1 Activities And Default Policies For Administrative Resources

Administrative Activities	Default Policy Allows These Roles	Also Protected By a JMX Resource?
Upload files for deployment.	`Admin`, `Deployer`	No
Control access to these methods in the file download servlet: • ALL methods • `wl_component_request` • `wl_ear_resource_request` • `ear_request` • `wl_xml_entity_request` • `wl_jsp_refresh_request` • `file` • `wl_init_replica_request` • `wl_file_realm_request` • `wl_managed_server_independence_request` **Note:** The file download servlet is used internally by WebLogic Server. Oracle recommends that you do not modify the default policies for any of its methods. They are listed here only for completeness.	`Admin`, `Operator`	No

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 3-1 (Cont.) Activities And Default Policies For Administrative Resources

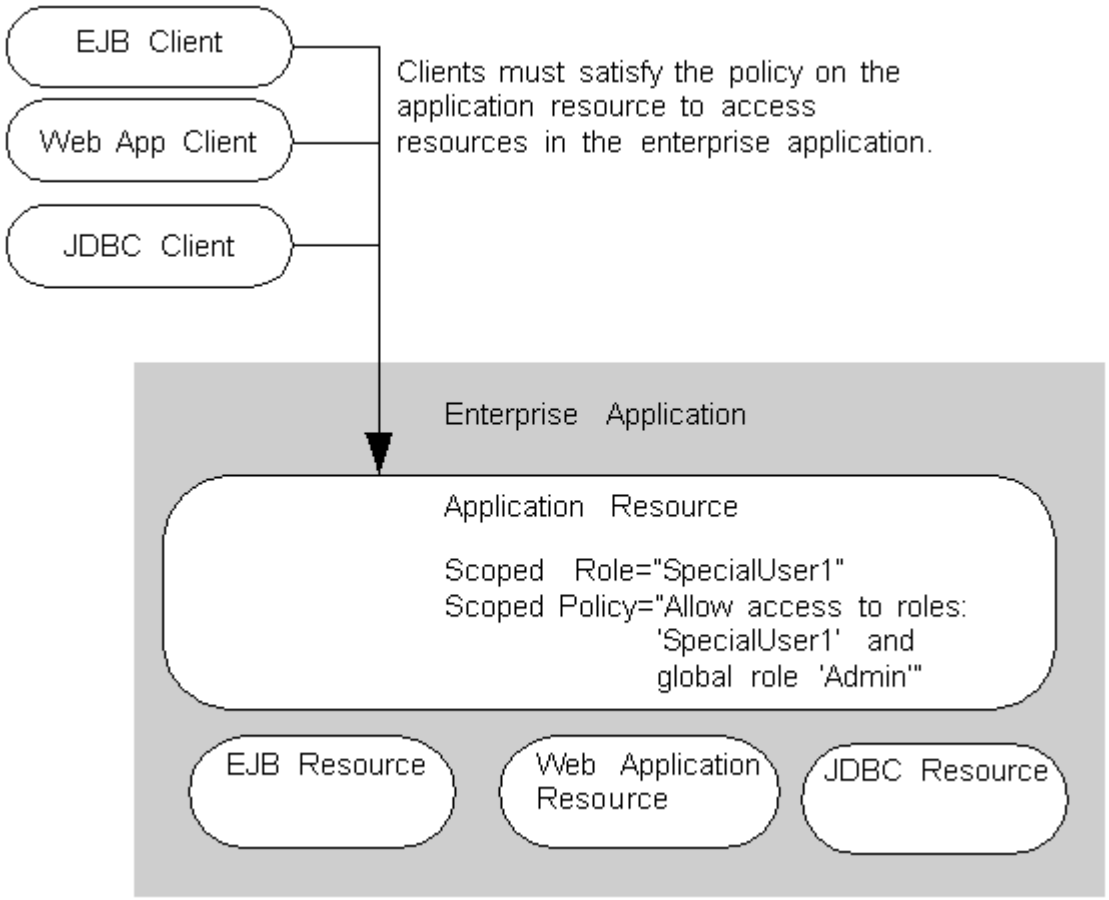
Administrative Activities	Default Policy Allows These Roles	Also Protected By a JMX Resource?
Enable applications to use identity assertion. The default policy for this activity specifies that an application must supply credentials for a user who is in the `Admin` role before it can successfully invoke the `Authentication.assertIdentity()` API. See `weblogic.security.services.Authentication` in the *Java API Reference for Oracle WebLogic Server*.	`Admin`	No
View domain and server logs through the .	`Admin`, `Deployer`, `Operator`, `Monitor`	Yes
Unlock users who have been locked out of their accounts.	`Admin`	Yes

Application Resources

An application resource is an enterprise application, Web application, or other Jakarta EE module that you deploy as a stand-alone application (for example, you can deploy Web Services and JDBC modules as stand-alone applications). You secure an application resource when you want to protect all resources that constitute the application. For example, securing an enterprise application protects access to all WebLogic resources within that application (see Figure 3-2).

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 3-2 Application Resource Protects All Resources



See Protecting a Hierarchy of Resources.

COM Resources

A COM resource represents a package that contains one or more jCOM classes. jCOM is a software bridge that allows bidirectional access between Jakarta EE objects deployed in WebLogic Server and Microsoft ActiveX components available within the Microsoft Office family of products, Visual Basic and C++ objects, and other Component Object Model/ Distributed Component Object Model (COM/DCOM) environments.

A policy on a COM resource protects access to all jCOM objects in a package.

For related information, see Configuring Access Control in *Developing JCOM Applications for Oracle WebLogic Server*.

EJB Resources

An EJB (Enterprise JavaBean) resource is an EJB deployment module (JAR), individual EJB, or individual method in an EJB. EJB resources exist within a hierarchy of resources, and at the top of the hierarchy is an application resource. See Protecting a Hierarchy of Resources.

Because the Jakarta EE platform standardizes EJB security in deployment descriptors, WebLogic Server integrates this standard mechanism with its Security Service to give you a

Copyright © 2007, 2026, Oracle and/or its affiliates.

choice of techniques for securing EJB resources. See Options for Securing Web Application and EJB Resources.

Enterprise Information Systems (EIS) Resources

An EIS resource is a system-level software driver used by an application server, such as WebLogic Server, to connect to an Enterprise Information System. Oracle supports resource adapters developed by EIS vendors and third-party application developers. Resource adapters can be deployed in any application server supporting the applicable Jakarta EE Platform Specification. Resource Adapters contain the Java code, and if necessary, the native components required to interact with the EIS.

To secure access to an EIS, create security policies and security roles for all resource adapters as a group, or for individual adapters. These resources exist within a hierarchy of resources, and at the top of the hierarchy is an application resource. See Protecting a Hierarchy of Resources.

For related information, see Security in *Developing Resource Adapters for Oracle WebLogic Server*.

Java DataBase Connectivity (JDBC) Resources

A Java DataBase Connectivity (JDBC) resource is a JDBC system resource, JDBC module that is part of an application, JDBC data source, or a specific method within a data source. If you deploy a JDBC module as a stand-alone application, the application is represented by an application resource (see Application Resources).

JDBC resources exist within a hierarchy of resources, and at the top of the hierarchy is an application resource. See Protecting a Hierarchy of Resources.

JDBC Operations

When you secure an individual data source, you can choose whether to protect `JDBC operations` using one or more of the following administrator methods:

- `admin`—The following methods on the `JDBCDataSourceRuntimeMBean` are invoked as `admin` operations: `clearStatementCache`, `suspend`, `forceSuspend`, `resume`, `shutdown`, `forceShutdown`, `start`, `getProperties`, and `poolExists`.
- `reserve`—Applications reserve a connection in the data source by looking up the data source and then calling `getConnection`.

> **Note**
>
> Giving a user the `reserve` permission enables them to execute vendor-specific operations. Depending on the database vendor, some of these operations may have database security implications.

- `shrink`—Shrinks the number of connections in the data source to the maximum of the currently reserved connections or to the initial size.
- `reset`—Resets the data source connections by shutting down and re-establishing all physical database connections. This also clears the statement cache for each connection. You can only reset data source connections that are running normally.

Copyright © 2007, 2026, Oracle and/or its affiliates.

- `All`—An individual data source is protected by the union of the `Admin`, `reserve`, `shrink`, and `reset` administrator methods.

> **Note**
>
> If a security policy controls access to connections in a multi data source, access checks are performed at both levels of the JDBC resource hierarchy (once at the multi data source level, and again at the individual data source level). As with all types of WebLogic resources, this double-checking ensures that the most specific security policy controls access.

Table 3-2 lists the JDBC administrator methods and the roles that the default policy allows for each operation.

Table 3-2 Default Policies for JDBC Resources

JDBC Administrator Methods	Default Policy Allows These Roles
`admin`	`Admin`, `Deployer`
`reset`	`Admin`, `Deployer`
`reserve`	`Everyone`
`shrink`	`Admin`, `Deployer`

Java Messaging Service (JMS) Resources

A Java Messaging Service (JMS) resource is a JMS system module, a JMS module that is part of an application (deprecated), a specific WebLogic JMS destination or destination template defined within a module, or a specific operation within a WebLogic JMS destination. You can create security policies and roles for WebLogic JMS destinations (JMS queues and JMS topics) as a group by securing the JMS module where they are defined, or for specific WebLogic JMS destinations (JMS queues or JMS topics) defined within a JMS module. The WebLogic destinations defined in a JMS module run on WebLogic JMS Servers.

These resources exist within a hierarchy of resources, and at the top of the hierarchy is an application resource. See Protecting a Hierarchy of Resources.

JMS Operations

When you secure a specific destination on a JMS server, you can protect operations on the destination. By default, destinations are not protected. This means that any valid user for a WebLogic server instance can send, receive, and browse messages on a destination. Only users defined by the policy condition can access control of the destination. Valid protection operations are:

- `send`—Required to send a message to a queue or a topic. This includes calls to the `MessageProducer.send()`, `QueueSender.send()`, and `TopicPublisher.publish()` methods, as well as the Messaging Bridge.
- `receive`—Required to create a consumer on a queue or a topic. This includes calls to the `Session.createConsumer()`, `Session.createDurableSubscriber()`, `QueueSession.createReceiver()`, `TopicSession.createSubscriber()`, `TopicSession.createDurableSubscriber()`, `Connection.createConnectionConsumer()`, `Connection.createDurableConnectionConsumer()`,

Copyright © 2007, 2026, Oracle and/or its affiliates.

`QueueConnection.createConnectionConsumer()`, `TopicConnection.createConnectionConsumer()`, and `TopicConnection.createDurableConnectionConsumer()` methods, as well as the Messaging Bridge and message-driven beans.

- `browse`—Required to view the messages on a queue using the `QueueBrowser` interface.
- `ALL`—Required to `send`, `receive`, and `browse` methods on a destination.

Java Naming and Directory Interface (JNDI) Resources

A Java Naming and Directory Interface (JNDI) resource is a node in a server's JNDI tree. A policy on a JNDI resource determines who can access WebLogic Server entities and actions through JNDI. You can create a policy on the root node of the JNDI tree or on individual nodes.

If your domain is configured to run in secured production mode, then the authorization policies for JNDI and MBean access are more restrictive to enforce a highly secure environment for your production domain. In this mode, remote anonymous JNDI access is not allowed for modify and list operations. The `SecurityConfigurationMBean.RemoteAnonymousJNDIEnabled` attribute controls whether remote anonymous JNDI access is permitted for list and modify operations. In secured production mode, its default value is `false`, so that access is not permitted. See SecurityConfigurationMBean in *MBean Reference for Oracle WebLogic Server* for more information about this MBean attribute. When in secured production mode, only the standard roles (`Admin`, `Deployer`, `Operator`, and `Monitor`) can look up MBeans using the JNDI `lookup` methods.

Supported Token Types

Web Service Secure Conversation Language (WS-SecureConversation) and SAML tokens are supported. The tokens have the following namespace and URI:

- For WS-SecureConversation 1.3:

  ```
  http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512
  http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
  ```

- For WS-SecureConversation 1.2:

  ```
  http://schemas.xmlsoap.org/ws/2005/02/sc
  http://schemas.xmlsoap.org/ws/2005/02/sc/sct
  ```

- For SAML 2.0:

  ```
  urn:oasis:names:tc:SAML:2.0:assertion
  ```

  Supported confirmation methods are sender-vouches, holder-of-key and bearer. Symmetric holder-of-key is not supported.

JMX Resources

A JMX resource is an MBean attribute or MBean operation. A policy on a JMX resource controls who can read or write MBean attributes or invoke operations.

WebLogic Server uses managed beans (MBeans) in the implementation of its management system. Almost all administrative activities require you to invoke an MBean operation or modify an MBean attribute using a Java Management Extensions (JMX) client. For example, the WebLogic Scripting Tool is a JMX client. See Understanding WebLogic Server MBeans in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Oracle provides a default set of JMX resources to protect WebLogic Server MBeans. See Default Security Policies for MBeans in the *MBean Reference for Oracle WebLogic Server*. For MBean attributes and operations that represent particularly sensitive data or actions, WebLogic Server uses additional types of resources to secure access. For example, the `ServerLifeCycleRuntimeMBean`'s `shutdown()` operation is protected by a JMX resource and a Server resource.

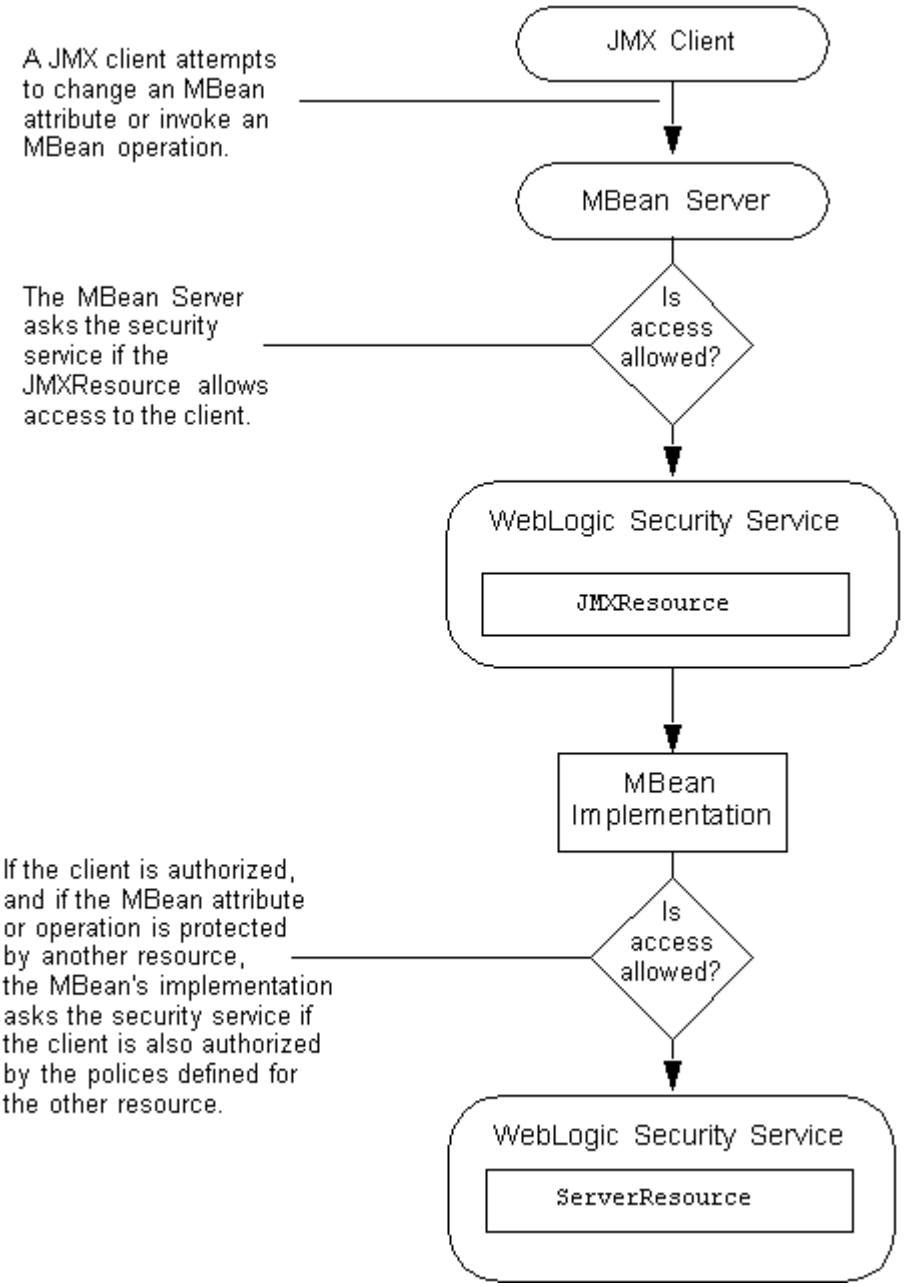
If your domain is configured to run in secured production mode, then more restrictive policies are enabled by default for JMX authorization. If you enable restrictive JMX policies, then the default policies allow MBean access `only` to the standard WebLogic Server roles (`Admin`, `Deployer`, `Operator`, or `Monitor`). You can enable or disable restrictive JMX policies using as follows:

- Use Fusion Middleware Control. From the WebLogic Domain menu, select **Security > Administration**. Expand **Advanced**, then expand **Secure Mode Settings**. Under Secure Mode Settings, select the **Restrictive JMX Policies** check box. See Configure domain security in *Administering Oracle WebLogic Server with Fusion Middleware Control*.

When a JMX client attempts to invoke an operation or change an attribute that is secured by a JMX resource and some other resource type, the client must satisfy the policies defined in both resources (see Figure 3-3).

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 3-3 MBean Server Checks with Both Resources



Maintaining a Consistent Security Scheme

The default configuration of groups, global roles, and security policies on all resources that are used to protect an entity or action create a consistent security scheme. You can, however, make modifications to that limit access in ways that you do not intend. Make sure that any modifications you make to the default security settings do not prevent a user from being authorized by both the JMX resource and other resource type. When you create or modify a security policy, consider taking the following action:

- Always include the `Admin` and `Operator` global roles in policies for Server resources.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Failure to use the `Operator` global role or a security role nested within this default global role may result in inconsistent behavior by the WebLogic Security Service.

- For a security policy on a deployable resource (such as an Web application or EJB module, Connector module, or startup/shutdown class), use the `Deployer` global role.

NoSQL Resources

WebLogic Server protects global NoSQL system resources with `weblogic.security.service.NoSQLResource`. The resource type is `<nosql>`, its keys are `resource` and `action`, and its runtime action is `access`. The resource identity uses the global management-resource name.

The shipped global `access` policy grants the `everyone` group, represented by `Grp(everyone)`. The repository alias and descriptor JNDI name are not policy identities. Review and configure the effective policy for each resource, including any resource-specific restrictions. Targeting and application/JNDI mappings do not substitute for authorization.

WebLogic Server evaluates the current authenticated subject before lazy provider or credential initialization and on guarded manager operations. These checks also apply to retained streams, iterators, cursors, prepared operations, and applicable metadata access. A missing eligible subject, authorization denial, or unavailable authorization service fails closed with `SecurityException`.

The subject that deployed the application or originally obtained a handle does not permanently authorize subsequent calls. Passing a handle or derived capability to another deployment does not transfer ownership.

Effective policy revocation affects the next guarded operation through an existing handle or retained capability without application redeployment. An already admitted operation can drain; revocation does not promise to interrupt it. Terminal resource invalidation has a separate recovery contract.

For the hierarchy, resource-specific policy workflow, verification, and revocation behavior, see Secure a NoSQL Resource in *Administering NoSQL Resources for Oracle WebLogic Server*.

Server Resources

Policies for a server resource determine who can control the state of a WebLogic Server server instance.

When users start server instances by directly invoking the `weblogic.Server` class in a Java command, the policy on the Server resource is the only security check that occurs. All other tasks that change the state of a WebLogic Server instance require the use of the , WebLogic Scripting Tool, Node Manager, or some other JMX client, and therefore require users to be authorized first by an additional JMX resource. See JMX Resources.

You can create security policies that apply to all WebLogic Server instances in a domain or to individual servers. If you define a policy for an individual server, you can protect all of its life cycle operations or define individual policies for each of the following operations:

- `boot`—A user who tries to start a WebLogic Server instance, either an Administration Server or Managed Server, must have permission to do so. This action is typically initiated through a call to the `java weblogic.Server` command on the command line, by a configured start script (which in turn calls the `java weblogic.Server` command), or through the Node Manager capabilities that allow for remote start of WebLogic Server

Copyright © 2007, 2026, Oracle and/or its affiliates.

- `shutdown`—A user who tries to shut down a running WebLogic Server instance, either an Administration Server or Managed Server, must have permission to do so. This action is typically initiated through the or the `WLST SHUTDOWN` or `FORCESHUTDOWN` commands.
- `suspend`—A user who tries to prohibit additional logins (logins other than for privileged administrative actions) to a running WebLogic Server instance, either an Administration Server or Managed Server, must have permission to do so. This action is typically initiated through the .
- `resume`—A user who tries to re-enable non-privileged logins to a running WebLogic Server instance, either an Administration Server or Managed Server, must have permission to do so. This action is typically initiated through the .

All server resources inherit a default security policy that gives permission to the `Admin` and `Operator` global security roles.

> **Note**
>
> If you enable the domain-wide administration port, then only the `Admin` role (and not `Operator`) can control the state of a WebLogic Server server instance. See Configure the Domain-Wide Administration Port in *Oracle WebLogic Remote Console Online Help*.

> **Note**
>
> Do not remove roles from the default security policies. Eliminating some of the existing security roles might negatively affect the functioning of WebLogic Server. However, if you like, you can make the default security policies more inclusive (for example, by adding new security roles). See Maintaining a Consistent Security Scheme.

Permissions for the weblogic.Server Command and the Node Manager

WebLogic Server provides two ways to start and shut down WebLogic Server instances (servers): the `weblogic.Server` command and the Node Manager. Because the underlying components for the `weblogic.Server` command and the Node Manager are different, the two commands use different authorization methods.

Permissions for Using the weblogic.Server Command

The `weblogic.Server` command, which you can use to start both Administration and Managed Servers, calls methods that are protected by a security policy on the Server resource. To use this command, you must satisfy the requirements of the security policy on the Server resource.

Some `weblogic.Server` arguments set attributes for MBeans. However, because these arguments modify an MBean before the server is in the `RUNNING` state, the security policy on the Server resource, not the protection on the MBean, is the authorizer. For example, a user in the `Operator` global role can use the `-Dweblogic.ListenPort` argument to change a server's default listen port, but once the WebLogic Server instance is running, this user cannot change the listen port value.

See weblogic.Server Command-Line Reference in the *Command Reference for Oracle WebLogic Server*.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Permissions for Using the Node Manager

The Node Manager uses both MBeans and the security policy on the Server resource to start a remote server.

If you configure a Node Manager on the host machine of a remote WebLogic Server instance, by default a user in the `Admin` or `Operator` global role can use the Node Manager to start the remote server.

See Node Manager Overview in *Administering Node Manager for Oracle WebLogic Server*.

Shutting down a WebLogic Server instance involves both MBeans and the security policy on the Server resource. When a user issues a shutdown command, the server first determines whether that user is granted the `Admin` or `Operator` global role (per the MBean security layer). Then, after the MBean operations run, the server determines whether the security policy on the Server resource authorizes the user to shut down the server.

See Starting and Stopping Servers: Quick Reference in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

URL Resources

A URL resource is a specific URL or URL pattern in a Web application. You can create a policy for a URL resource that protects all HTTP methods for a specified URL or URL pattern, or that protects only specific HTTP methods. These resources exist within a hierarchy of resources, and at the top of the hierarchy is an application resource. See Protecting a Hierarchy of Resources.

Because the Jakarta EE platform standardizes Web application security in deployment descriptors, WebLogic Server integrates this standard mechanism with its Security Service to give you a choice of techniques for securing Web application resources. See Options for Securing Web Application and EJB Resources.

Web Service Resources

A Web Service resource is a Web Service module (WAR or JAR) or an operation within a Web Service module. Web Services are protected by the following hierarchy of resources:

- The application resource for the parent application.
- The Web Service resource for the Web Service module (WAR or JAR).
- Individual Web Service resources for each Web Service operation.

  If you implement the Web Service with standard Java objects, any of the above resources protect the Java objects.

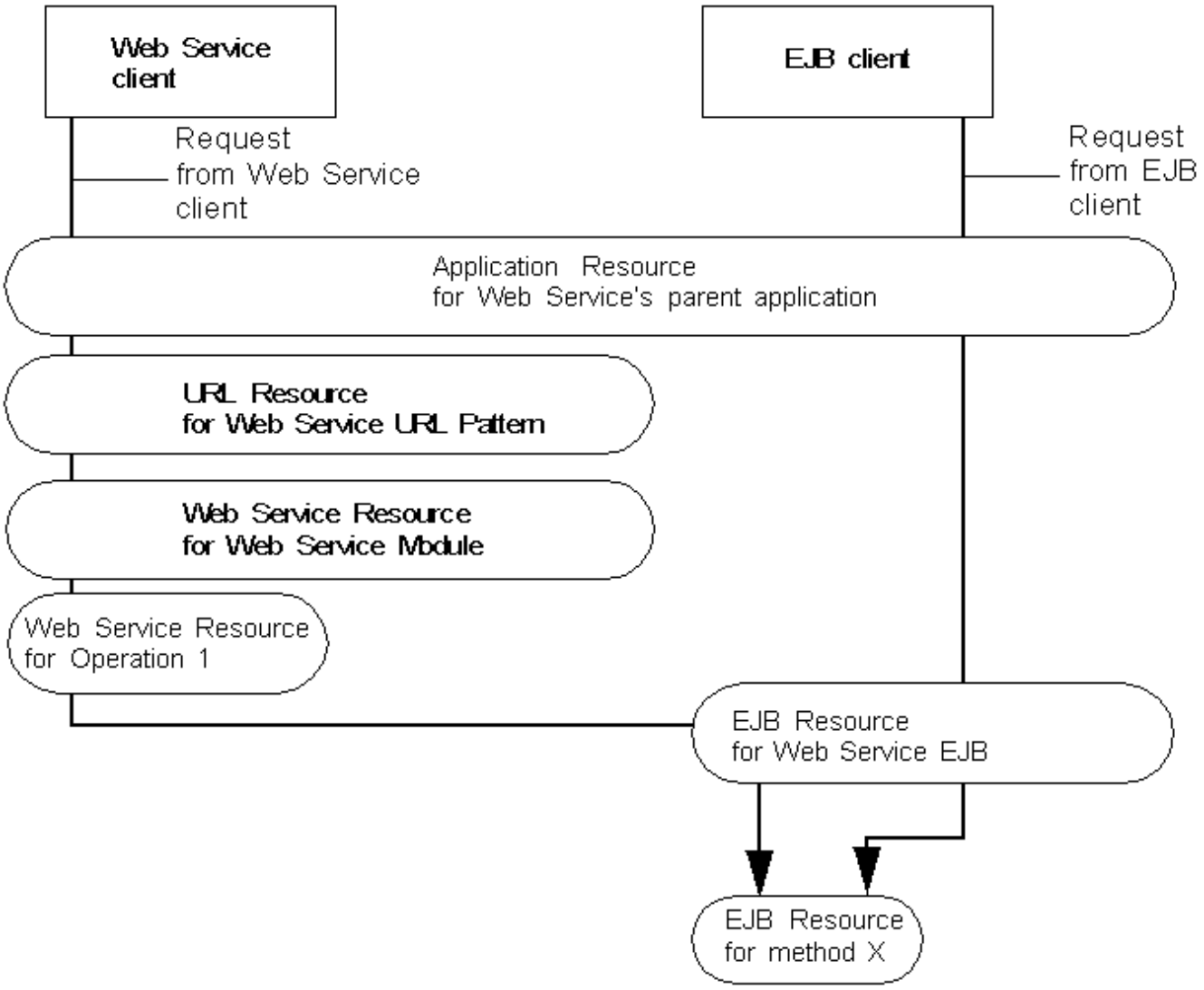
  If you implement the Web Service with an EJB any of the above **or** any of the following resources protect the EJB implementation:

  - The EJB resource for the EJB.
  - Individual EJB resources for each EJB method.

If you use an EJB to implement your Web Service, Oracle recommends that you create a policy at the application level. Policies on the Web Service module and individual Web Service operations apply only to Web Service clients. EJB clients can use RMI or JNDI to bypass the Web Service module and directly invoke EJB operations (see Figure 3-4).

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 3-4 Hierarchy of Resources for Web Service with EJB Implementation



For information on using Java annotations to secure Web Services, see Configuring Message-Level Security in *Securing WebLogic Web Services for Oracle WebLogic Server*.

Work Context Resources

Work Contexts enable Jakarta EE developers to define and pass properties without including them in a remote call. A Work Context resource represents the operations that create, delete, read, or modify a property. You can use one Work Context resource for all operations of a given property, or you can create individual resources for each operation.

See Best Practices for Application Design in *Developing RMI Applications for Oracle WebLogic Server*.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Coherence Resources

Coherence resources provide distributed, in-memory caching and data grid processing for applications. Roles and policies can be applied to two types of Coherence resources:

- Caches – A cluster contains any number of caches that are shared by all cluster members. The caches are used by applications to store and retrieve data.
- Services – A cluster contains any number of services that are shared by all cluster members. The services include connectivity services, cache services, and processing services. Each cluster member can provide and consume such services.

The default authorization policy allows everybody access to all Coherence resources. To define policies and roles on caches and services, the names of the caches and services must be known. In some cases, the cache configuration file in a Coherence Grid ARchive (GAR) module can be inspected to discover cache and service names. However, there are some configurations that allow applications to use different names to refer to the same cache. Always consult an application's developers or architects to be certain of the cache and service names used by an application.

Copyright © 2007, 2026, Oracle and/or its affiliates.

4

Options for Securing Web Application and EJB Resources

WebLogic Server offers several options for securing your Web application and EJB resources, including the Jakarta EE standard model.
In the Jakarta EE standard model, you can secure Web applications and EJBs in either of the following ways:

- Declaratively, using deployment descriptors or metadata annotations.

  For EJB 3.*x*, EJB security metadata annotations can be specified directly in the EJB bean class to specify the roles that are allowed to invoke all, or a subset, of the EJB's methods.

- Programmatically, as described in the following topics in *Developing Applications with the WebLogic Security Service*:

  – Using Programmatic Security With Web Applications

    To implement programmatic security in web applications, WebLogic Server supports the use of the Jakarta EE Security API `getCallerPrincipal`, `getPrincipalsByType`, `isCallerInRole`, `hasAccessToWebResource`, and `authenticate` methods of the `SecurityContext` interface, and the `getUserPrincipal` and `isUserInRole` methods of the `HttpServletRequest` interface.

  – Using Programmatic Security With EJBs

    To implement programmatic security in EJBs, WebLogic Server supports the use of the Jakarta EE Security API (JSR 375) `getCallerPrincipal`, `getPrincipalsByType`, and `isCallerInRole` methods of the `SecurityContext` interface, and the `isCallerInRole` and `getCallerPrincipal` methods of the `EJBContext` interface. WebLogic Server also supports the use of the `security-role-ref` element in deployment descriptors.

Because this Jakarta EE standard may be too inflexible for some environments, WebLogic Server also offers a choice of other, more flexible models in addition to supporting the Jakarta EE standard.

> **Note**
>
> If you are implementing security using Jakarta Authorization, you must use the Jakarta EE standard model. Other WebLogic Server models are not available and the security functions for Web applications and EJBs in are disabled. See Using the Java Authorization Contract for Containers in *Developing Applications with the WebLogic Security Service*.

This chapter includes the following sections:

- Deployment Descriptors Not Required
- Comparison of Security Models for Web Applications and EJBs
- Understanding the Advanced Security Model
- Securing Web Applications and EJBs

Copyright © 2007, 2026, Oracle and/or its affiliates.

> **Note**
>
> The instructions for EJB resources provided in this section also apply to Message-Driven Beans (MDBs).

Deployment Descriptors Not Required

As described in What Was New and Changed in EJB 3.0 in *Developing Jakarta Enterprise Beans for Oracle WebLogic Server*, you are no longer required to create the deployment descriptor files (such as `ejb-jar.xml`). You can now use metadata annotations in the bean file itself to configure metadata. Annotations simplify the EJB development process by allowing a developer to specify within the Java class itself how the bean behaves in the container, requests for dependency injection, and so on. Annotations are an alternative to deployment descriptors that were required by older versions (2.*x* and earlier) of EJB.

You can still use XML deployment descriptors in addition to, or instead of, the metadata annotations if you so choose.

> **Note**
>
> Deployment descriptor elements always override their annotation counterparts. In the case of conflicts, the deployment descriptor value overrides the annotation value.

Comparison of Security Models for Web Applications and EJBs

For EJBs, the application developer can make the deployer's task easier by using the security-related metadata annotations to specify the security roles that are allowed to invoke the EJB methods. If you use annotations or deployment descriptors, you choose a security model when you deploy each Web application or EJB and your choice is immutable for the lifetime of the deployment. If you want to use a different model, you must delete and redeploy the Web application or EJB.

Each deployment descriptor-based security model defines two types of behaviors for securing Web applications and EJBs: where roles and policies are defined and which URL patterns and EJB methods trigger the Security Service to perform security checks.

Table 4-1 compares the security model behaviors.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 4-1 Security Model Behaviors

This Model . . .	Uses Roles and Policies From . . .	And Performs Security Checks . . .
Metadata Annotations Categorized as Deployment Descriptor Only (Jakarta EE standard) in	Metadata annotations in the EJB bean file itself specify the roles that are allowed to invoke all, or a subset, of the EJB's methods.	Only when clients request EJB methods that are protected by a role specified in an annotation. You can use metadata annotations in conjunction with deployment descriptor- and -based mechanisms. In case of conflict, deployment descriptor elements always override their annotation counterparts.
Deployment Descriptor Only (Jakarta EE standard)	The `web.xml`, `weblogic.xml` and `ejb-jar.xml`, `weblogic-ejb-jar.xml` deployment descriptors. If roles have been defined for the application that contains the Web application or EJB, all roles are combined using a logical OR operation.	Only when clients request URLs or EJB methods that are protected by a policy in the deployment descriptor. Deployment descriptor elements always override their annotation counterparts, if any.
Custom Roles	This model uses role mappings from a role mapping provider that you configure for the security realm. You can use to configure the provider. Any role mappings in the deployment descriptors or annotations are ignored. The model uses the policies that are defined in the `web.xml` and `ejb-jar.xml` deployment descriptors.	Only when clients request URLs or EJB methods that are protected by a policy in the deployment descriptor.
Custom Roles and Policies	A role mapping provider and an authorization provider that you configure for the security realm. You can use to configure the providers. Any role mappings or policies in the deployment descriptors or annotations are ignored.	For all URLs and EJB methods.
Advanced	Configurable. You can configure this model to use only security data from annotations or deployment descriptors, use only the data from security providers, or import security data from deployment descriptors into the security provider databases to provide a baseline for further modifications.	Configurable.

Discussion of Each Model

The following sections describe each model and suggest scenarios under which each is appropriate.

Metadata Annotations

As described in Overview of Metadata Annotations and EJB Bean Files in *Developing Jakarta Enterprise Beans for Oracle WebLogic Server*, the EJB programming model uses the metadata annotations feature in which you create an annotated EJB bean file, compile the class with the standard Java compiler, and the resulting class can then be packaged into a target module for

Copyright © 2007, 2026, Oracle and/or its affiliates.

deployment. At runtime, WebLogic Server parses the annotations and applies the required behavioral aspects to the bean file.

The following security-related annotations are available:

- jakarta.annotation.security.DeclareRoles — Explicitly lists the security roles that will be used to secure the EJB.
- jakarta.annotation.security.RolesAllowed — Specifies the security roles that are allowed to invoke all the methods of the EJB (when specified at the class-level) or a particular method (when specified at the method-level.)
- jakarta.annotation.security.DenyAll — Specifies that the annotated method cannot be invoked by any role.
- jakarta.annotation.security.PermitAll — Specifies that the annotated method can be invoked by all roles.
- jakarta.annotation.security.RunAs — Specifies the role which runs the EJB. By default, the EJB runs as the user who actually invokes it.

At deployment time, the deployer must then create these security roles if they do not already exist and map users to these roles using the to update your security realm. For details, see Security Policies and Roles in *Oracle WebLogic Remote Console Online Help*.

This model gives the application developer more control without having to implement programmatic authorization in EJBs.

Table 4-2 Metadata Annotations: Typical Scenario

Company A, Developer	Company A, Admin/Deployer
In Company A, a user in the role of developer performs the following tasks: • Adds one or more security-related metadata annotations. • Uses the WebLogic compile tool weblogic.appc (or its Ant equivalent wlappc) to compile the bean file. • Turns application over to the Admin/Deployer. Provides instructions as to which roles are required.	In Company A, a user in the role of administrator or deployer performs the following tasks: • Uses the to ensure that the roles exist and are properly mapped to groups and users. • To have only the annotations considered, deploys the application as Deployment Descriptor Only (Jakarta EE standard) in .

You can use metadata annotations in conjunction with deployment descriptor- and mechanisms. If you do so, note the following:

- In case of conflict, deployment descriptor elements always override their annotation counterparts. For example, if the annotation specifies Deny All for a method and the deployment descriptor specifies that the Developer role has access to that method, the Developer role does have access to the method.
- If you specify Custom Roles in the , any role mappings in the deployment descriptors or annotations are ignored.

See Securing Access to the EJB in *Developing Jakarta Enterprise Beans for Oracle WebLogic Server* for an example of a simple stateless session EJB that uses all of the security-related annotations.

Deployment Descriptor Only Model

This is part of the standard Jakarta EE model and is a technique for adding declarative security to Web applications and EJBs. It uses only roles and policies defined by a developer in the Jakarta EE deployment descriptor (DD) and the WebLogic Server DD. It requires the security

Copyright © 2007, 2026, Oracle and/or its affiliates.

administrator to verify that the security principals (groups or users) in the deployment descriptors exist and are mapped properly in the security realm.

> **Note**
>
> This model also affects application-scoped roles that apply to an EAR: with this model, the Security Service uses only the application-scoped roles defined in the WebLogic Server DD.

If a developer changes roles or policies in a deployment descriptor, WebLogic Server recognizes the change as soon as you redeploy the Web application, EJB, or EAR.

With this model, EJBs and URL patterns are not protected by roles and policies of a broader scope (such as a policy scoped to an entire Web application). If an EJB or URL pattern is not protected by a role or policy in the DD, then it is unprotected: anyone can access it. For example, if you create an application-scoped policy for an EAR and the EAR contains an EJB, the EJB will not be protected by the EAR's application-scoped policy.

This model is appropriate if developers and security administrators can closely coordinate their work, both upon initial deployment of the Web application or EJB and upon subsequent redeployments. Table 4-3 shows a typical scenario:

Table 4-3 Deployment Descriptors Only: Typical Scenario

Company A, Developer	Company A, Admin/Deployer
In Company A, a user in the role of developer performs the following tasks: • Maps EJBs and/or Web URLs to roles in the Jakarta EE DD. • Maps roles to principals in the WebLogic Server DD. • Turns application over to the Admin/Deployer.	In Company A, a user in the role of administrator or deployer performs the following tasks: • Uses the to ensure that groups exist and are properly mapped to users.

Custom Roles Model

This security model uses policies defined in the Jakarta EE DD and ignores any principal mappings in the WebLogic Server DD. The security administrator completes the role mappings using the .

The model enables team members to focus on their areas of expertise. Web application and EJB developers need only to declare which URL patterns or EJB methods should be secured. Then the security administrator creates role mappings that fit within the existing hierarchy of roles and principals for a given realm.

If a developer changes policies in a deployment descriptor, WebLogic Server recognizes the change as soon as you redeploy the Web application or EJB. If an administrator changes role mappings, the changes take effect immediately without requiring a redeployment.

This model is appropriate if developers and administrators cannot closely coordinate their work or if role mappings change frequently. Table 4-4 shows a typical scenario:

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 4-4 Customize Roles Only: Typical Scenario

Company A, ISV Developer; or Company B, Developer	Company B, Admin/Deployer
An ISV developer from Company A, or a developer from Company B, does the following tasks: • Maps EJBs/URLs to roles in Jakarta EE deployment descriptor. • Turns application over to Admin/Deployer	An administrator or a deployer from Company B does the following task: • Uses the to define security roles.

Custom Roles and Policies Model

This security model offers unified and dynamic security management. It uses roles and policies that a security administrator has created using the and ignores any roles and policies defined in deployment descriptors.

Instead of requiring developers to modify multiple deployment descriptors when organizational security requirements change, administrators can modify all security configurations from a centralized, graphical user interface. Users, groups, security roles, and security policies can all be defined using the . As a result, the process of making changes based on updated security requirements becomes more efficient.

This model is appropriate if you require only that entire Web applications or EJBs be secured, but is less appropriate if you require fine-grained control of a large number of specific URL patterns or EJB methods. Such fine-grained control requires a developer to provide to administrators detailed information about the URL patterns or EJB methods that must be secured. If you require such fine-grained control, consider using the Custom Roles model (see Custom Roles Model).

The model also introduces a slight performance degradation because it checks permissions regardless of which URL a client requests or EJB method a client attempts to invoke.

Table 4-5 shows a typical scenario:

Table 4-5 Customize roles and Policies: Typical Scenario

Company A, Developer	Company A, Admin/Deployer
• Provides no mappings in the Jakarta EE or WebLogic Server DD. • Turns application over to Admin/Deployer	• Uses the to define roles and policies for EJBs and Web applications.

Advanced Model

WebLogic Server provides this model primarily for backwards compatibility with releases prior to 9.0.

You can configure the following behaviors for this model (see Understanding the Combined Role Mapping Enabled Setting):

- Perform security checks for all URLs and EJB methods or only those that are protected in the deployment descriptors.
- (Not applicable if you configure this model to perform security checks only for URLs and EJB methods that are secured in deployment descriptors.) Use only roles and policies defined in the deployment descriptors, or use only roles and policies defined in the realm's

Copyright © 2007, 2026, Oracle and/or its affiliates.

security providers, or import security data from deployment descriptors into the realm's authorization provider or role mapping provider databases.

- (Not applicable if you configure this model to use only roles and policies defined in the realm's security providers.) Combine roles in parent applications with roles in the Web application or EJB, or override roles in parent applications.

If you change the configuration of this model, the change applies to all Web applications and EJBs that use this model. For example, you configure the Advanced model to perform security checks for all URLs and methods, and then you deploy several EJBs and configure them to use the Advanced model. The EJB container will request a security check any time a client tries to invoke any method in any of the several EJBs. If you then modify the Advance model to perform security checks only for the EJB methods that are protected in deployment descriptors, then the EJB container immediately begins to request security checks only for protected methods for the several EJBs.

Understanding the Advanced Security Model

> **Note**
>
> This section applies only for those Web applications and EJBs that use the Advanced security model.
>
> Oracle recommends using an alternative security model instead of the advanced security model. See Comparison of Security Models for Web Applications and EJBs.

Three settings configure the Advanced model: **Check Roles and Policies**, **When Deploying Web Applications or EJBs**, and **Combined Role Mapping Enabled**. Failure to understand these settings could result in incorrect or lost security data.

If you change the configuration of this model, the change applies to all Web applications and EJBs that use this model.

The following sections describe the settings for the Advanced security model:

- #unique_65
- Understanding the When Deploying Web Applications or EJBs Setting
- How the Check Roles and Policies and When Deploying Web Applications or EJBs Settings Interact
- Understanding the Combined Role Mapping Enabled Setting

Disabling a Globally Attached OWSM Policy

No behavior policies provide the ability to effectively disable an OWSM policy attached globally in a policy set. lists the no behavior policies that are supported by Java EE web services and clients to disable a globally attached OWSM security policy.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 4-6 No Behavior Policies Supported by Java EE Web Services and Clients

No Behavior Policy	Description
OWSM no behavior policies	OWSM no behavior policies that are valid for Java EE web services and clients include: • `no_authentication_client_policy` and `no_authentication_service_policy` • `no_authorization_service_policy` • `no_messageprotection_client_policy` and `no_messageprotection_service_policy` See No Behavior Policies in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. Attach an OWSM no behavior policy directly to the Java EE web service or client using the procedures defined in the following sections in *Securing Web Services and Managing Policies with Oracle Web Services Manager*: • Attaching Policies to Java EE Web Services and Clients at Design Time • Attaching Policies Directly Using Fusion Middleware Control • Attaching Policies Directly to Java EE Web Services and Clients Using WLST

WebLogic no behavior policy	The WebLogic no behavior policy, `Wssp1.5-No-Op.xml`, is defined as follows:
	`<?xml version="1.0"?>`
	`<wsp15:Policy xmlns:wsp15="http://www.w3.org/ns/ws-policy">`
	`<wsp15:All>`
	`<wsp15:Policy/>`
	`</wsp15:All>`
	`</wsp15:Policy>`
	Attach the `Wssp1.5-No-Op.xml` no behavior policy to the Java EE web service or client using the procedures defined in the following sections.
	Web service:
	• Configuring Simple Message-Level Security
	•
	Web service client:
	• Using a Client-Side Security Policy File

See the following sections in *Securing Web Services and Managing Policies with Oracle Web Services Manager*:

- Attaching Policies Globally Using Fusion Middleware Control
- Attaching Policies Globally Using WLST

Understanding the When Deploying Web Applications or EJBs Setting

The **When Deploying Web Applications or EJBs** setting determines whether the Security Service ignores role and policy data in deployment descriptors and annotations or imports the data into role mapping and authorization provider databases **each time** you deploy a Web application or EJB.

> **Note**
>
> This setting is valid only if you have set **Check Roles and Policies** to **All Web applications and EJBs** (`RealmMBean.FullyDelegateAuthorization=true`).

Copyright © 2007, 2026, Oracle and/or its affiliates.

Using WLST, set the value of **When Deploying Web Applications or EJBs** as follows:

- To secure Web application and EJB resources using *only* , set both `RealmMBean.DeployPolicyIgnored` and `RealmMBean.DeployRoleIgnored` to `true`.

  At this point, you can begin to use to secure the resources. See Create a Scoped Role and Create a Policy for Resource Instances in *Oracle WebLogic Remote Console Online Help*.
- To import security data from the deployment descriptors and annotations, set both `RealmMBean.DeployPolicyIgnored` and `RealmMBean.DeployRoleIgnored` to `false`.

> **Note**
>
> Importing security data introduces risks to the integrity of your security data. Each time you import security data, the Security Service attempts to remove all associated security data from the provider databases and re-imports data from the deployment descriptors and annotations. If you modified the imported security data, then your modifications could become invalid or could be removed.

How the Check Roles and Policies and When Deploying Web Applications or EJBs Settings Interact

Table 4-7 shows how to achieve the behavior you want from the WebLogic Security Service using different combinations of the Check Roles and Policies and When Deploying Web Applications and EJBs settings.

Table 4-7 Interaction Between the Check Roles and Policies Setting and the When Deploying Web Applications or EJBs Setting

If you want to control security for . . .	and set security for Web application and EJB resources . . .	then set Check Roles and Policies to . . .	and set When Deploying Web Applications or EJBs to . . .
All Web application and EJB resources	using *only*	All Web applications and EJBs	Ignore Roles and Policies from DD
All Web application and EJB resources	by *copying* or *reinitializing* security data from the deployment descriptors and annotations into the configured Authorization and Role Mapping providers' databases when the Web application or EJB resource is deployed, then use one of the other techniques to modify security roles and security policies **Note:** Security data is copied/reinitialized *each time* the Web application or EJB resource is deployed.	All Web applications and EJBs	Initialize Roles and Policies from DD

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 4-7 (Cont.) Interaction Between the Check Roles and Policies Setting and the When Deploying Web Applications or EJBs Setting

If you want to control security for . . .	and set security for Web application and EJB resources . . .	then set Check Roles and Policies to . . .	and set When Deploying Web Applications or EJBs to . . .
Only on Web applications and EJB methods that are specified in the deployment descriptors and annotations (default configuration)	using *only* the deployment descriptors and annotations	Web applications and EJBs Protected in DD	--

Understanding the Combined Role Mapping Enabled Setting

The Combined Role Mapping Enabled setting determines how the role mappings in the Enterprise Application, Web application, and EJB containers interact.

WebLogic Server provides this setting for backwards compatibility with 8.x versions. For all applications initially deployed in version 9.x, the default value for this setting is true (enabled). For all applications previously deployed in version 8.1 and upgraded to version 9.x, the default value is false (disabled). If either of the following is true, consider changing the default value for Combined Role Mapping Enabled:

- You selected the Advanced security model for an 8.x application upgrade and want to use the combine role mapping behavior available in version 9.x.
- You selected the Advanced security model for a 9.x application and want to use the role mapping behavior in version 8.x.

compares how this setting affects security for Web applications and EJBs:

Table 4-8 How Combined Role Mapping Affects Security for Web Applications and EJBs

When Combined Role Mapping is Disabled...	When Combined Role Mapping is Enabled...
Role mappings for each container are exclusive to other containers unless defined by the `<externally-defined>` descriptor element.	Application role mappings are combined with EJB and Web application mappings so that all principal mappings are included. The Security Service combines the role mappings with a logical OR operator.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 4-8 (Cont.) How Combined Role Mapping Affects Security for Web Applications and EJBs

When Combined Role Mapping is Disabled...	When Combined Role Mapping is Enabled...

If one or more policies in the web.xml file specifies a role for which no role mapping exists in the weblogic.xml file, the Web application container assumes that the undefined role is the name of a principal. It therefore maps the assumed principal to the role name. For example, if the web.xml file contains the following stanza in one of its policies:
 	
`<auth-constraint>`
`  <role-name>PrivilegedUser</role-name>`
`</auth-constraint>`
 	
but the weblogic.xml file has no role mapping for PrivilegedUser, then the Web application container creates an in-memory mapping that is equivalent to the following stanza:
 	
`<security-role-assignment>`
`<role-name>PrivilegedUser</role-name>`
`<principal-name>PrivilegedUser </principal-name>`
`</security-role-assignment>`
	If one or more policies in the web.xml file specifies a role for which no mapping exists in the weblogic.xml file, the Web application container creates an empty map for the undefined role (that is, the role is explicitly defined as containing no principal). Therefore, no one can access URL patterns that are secured by such policies.

Role mappings for EJB methods must be defined in the weblogic-ejb-jar.xml file. Role mappings defined in the other containers are not used unless defined by the `<externally-defined>` descriptor element.	If one or more policies in the ejb-jar.xml file specifies a role for which no mapping exists in the weblogic-ejb-jar.xml file, the EJB container creates an empty map for the undefined role (that is, the role is explicitly defined as containing no principal). Therefore, no one can access methods that are secured by such policies.

Usage Examples

The following examples show the differences in role mapping behaviors depending on whether Combined Role Mapping is enabled or disabled.

Example for EAR, WAR and EJB

MyAppEar contains MyAppWAR which contains MyEJB. Role to Principal mappings (p1 and p2) are as follows:

- EAR descriptor, myRole = p1
- WAR descriptor, myRole = p2
- EJB-JAR descriptor, myRole = empty

When Combined Role Mapping is enabled, the role mappings would be:

- For the Ear container, myRole maps to p1.
- For the WAR container, myRole maps to p1 or p2.
- For the EJB container, myRole maps to p1.

When Combined Role Mapping is disabled, the role mappings would be

Copyright © 2007, 2026, Oracle and/or its affiliates.

- For the Ear container, myRole maps to p1.
- For the WAR container, myRole maps to p2.
- For the EJB container: Must be externally-defined or the deployment fails.

Example for EAR and WAR

MyAppEar contains MyAppWAR. Role to Principal mappings are as follows:

- In MyAppEAR descriptor, myRole = p1
- In MyAppWAR descriptor, myRole = (none defined)

When Combined Role Mapping is enabled, the role mappings would be:

- For the Ear container, myRole maps to p1.
- For the WAR container, myRole maps to p1.

  The mapping is the same because of the combined role behavior.

When Combined Role Mapping is disabled, the role mappings would be:

- For the Ear container, myRole maps to p1.
- For the WAR container, myRole maps to MyRole.

Securing Web Applications and EJBs

For metadata annotations, when you code an EJB you add the security-related annotations to specify the roles that are allowed to invoke all, or a subset, of the methods. At deployment time, the deployer must then create these security roles if they do not already exist and map users to these roles using to update your security realm. For details, see Security Policies and Roles in *Oracle WebLogic Remote Console Online Help*.

For deployment descriptor- and -based security, you choose a security model when you deploy each Web application or EJB, and your choice is immutable for the lifetime of the deployment. If you want to use a different model, you must delete and redeploy the Web application or EJB.

For information on using to deploy applications, choose a security model, modify roles and policies, and complete other related tasks, see Deploying Applications in *Oracle WebLogic Remote Console Online Help*.

If you plan to use deployment descriptors to secure Web applications or EJBs, see Using Declarative Security With Web Applications and Using Declarative Security With EJBs in *Developing Applications with the WebLogic Security Service*.

Copyright © 2007, 2026, Oracle and/or its affiliates.

5

Security Policies

Understand the features and functions of security policies, which specify who can access a WebLogic Server resource. You can create simple policies, such as "allow access if user is in Admin role", or more complex policies, such as "between the hours of 8 and 5, allow access if user is in Admin role".
This chapter includes the following sections:

- Security Policy Storage and Prerequisites for Use
- Default Root Level Security Policies
- Security Policy Conditions
- #unique_76
- Using to Manage Security Policies

For information on using security policies to protect multiple resources, see Using Policies to Protect Multiple Resources.

Security Policy Storage and Prerequisites for Use

Security policies for all resources other than Web Application resources and EJB resources are always stored in the security provider database of the Authorization provider that is configured in the security realm. The security realm that WebLogic Server provides stores policies in the embedded LDAP server.

For Web Application resources and EJB resources, the location of policies depends on the following:

- If you implement security using Jakarta Authorization, deployment-descriptor security constraints are translated into permissions in the configured authorization policy provider. The deployment descriptors declare the constraints; they are not the provider's policy store.
- If you use the DDOnly model to secure a Web application or EJB, the policies are stored in the deployment descriptors.
- If you use a security model that ignores the policies in the descriptors, then the Authorization provider determines where the policies are stored. The security realm that WebLogic Server provides stores policies in the embedded LDAP server.
- If you use the Advanced security model, the location of policies depends on how you configure the model.

See Options for Securing Web Application and EJB Resources.

Each user or group that you add to a security policy must be defined in the security provider database of the Authentication provider that is configured in the active security realm. Each role that you add must be defined in the security provider database of the Role Mapping provider that is configured in the active security realm. The security realm that WebLogic Server provides is configured to use the WebLogic Authentication and WebLogic XACML Role Mapping providers, which store users, groups, and roles in the embedded LDAP server.

See WebLogic Security Providers in *Understanding Security for Oracle WebLogic Server*.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Default Root Level Security Policies

A **root level policy** is inherited by all instances of a specific resource type. Table 5-1 describes the default root level policies that are defined in the security realm that WebLogic Server installs. For information about the roles and groups that are named in these policies, see Users, Groups, And Security Roles.

> **Note**
>
> You can access root level policies in the . See Security Policies in *Oracle WebLogic Remote Console Online Help*.

Table 5-1 Default Security Policies for WebLogic Resources

WebLogic Resource	Security Policy
Administrative resources	Default global role: `Admin`
Application resources	None
EIS (Resource Adapter) resources	Default group: `Everyone`
EJB resources	Default group: `Everyone`
COM resources	None
JDBC resources	Default group: `Everyone`
JNDI resources	Default group: `Everyone`
JMS resources	Default group: `Everyone`
Server resources	Default global roles: • `Admin` • `Operator`
Work Context	Default group: `Everyone`
URL resources	Default group: `Everyone`
Web Services resources	Default group: `Everyone`
Coherence resources	Default group: `Everyone`

> **Note**
>
> Do not modify the default root level policies for Administrative and Server resources to make them more restrictive. Eliminating some of the existing security roles might negatively impact the functioning of WebLogic Server. However, if you like, you can make the default security policies more inclusive (for example, by adding new security roles). See Maintaining a Consistent Security Scheme.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Security Policy Conditions

To determine who can access a resource, a policy contains one or more conditions. The most basic policy simply contains the name of a security role or a principal. For example, a basic policy might simply name the Admin global role. At run time, the WebLogic Security Service interprets this policy as allow access if user is in Admin role. You can create more complex conditions and combine them using the logical operators AND and OR (which is an inclusive OR). You can also negate any condition, which would prohibit access under the specified condition.

The WebLogic Server Authorization providers display three kinds of built-in policy conditions in the :

> **Note**
>
> These sections describe the conditions that are available in realms that use the WebLogic Authorization provider or the WebLogic XACML Authorization provider. If your security realm uses a third-party Authorization provider, refer to the third-party documentation for information on its capabilities.

- Basic Policy Conditions
- #unique_79
- Context Element Policy Conditions

Basic Policy Conditions

The basic policy conditions that are available in this release of WebLogic Server are:

- `User`—Allows a specific user to access the resource. For example, you might create a condition indicating that only the user `John` can access the `Deposit` EJB.

  > **Note**
  >
  > Instead of the User condition, Oracle recommends adding the user to a Group and using the Group condition in the security policy. If you do use the User condition, ensure that you update any existing policies when the user is deleted from the authentication store.

- `Group`—Allows all users or groups in the specified group to access the resource unless a User or Role condition contradicts the Group condition.
- `Role`—Allows all users or groups in the specified role to access the resource unless a User or Group condition contradicts the Role condition. For example, if you create a Role condition that specifies ‘Admin’ and a User condition that negates ‘Joe’, then user ‘Joe’ will be denied access even if he is in the Admin role.
- `Server is in Development Mode`—Allows access if the server that hosts the resource is running in development mode. See Creating a WebLogic Domain in *Creating WebLogic Domains Using the Configuration Wizard*.
- `Allow access to everyone`—Allows access for all users, groups, and roles.

Copyright © 2007, 2026, Oracle and/or its affiliates.

- `Deny access to everyone`—Prohibits access for all users, groups, and roles.
- `Element requires signature by`—(Used only when securing Web Services resources) Creates a condition for a security policy based on who has digitally signed an element in the SOAP request message that invokes a Web Service operation. For example, you might create a condition that says the `getBalance` operation can only be invoked if the `AccountNumber` element in the incoming SOAP request has been digitally signed by a user who is named `joe`.

  To create an `Element requires signature by` condition, provide the following information:

  – Specify whether a group or a user is required to sign the SOAP element.

    For example, enter `user` to specify that a user must sign the element.

  – The name of the user or group that must sign the element.

  – The name of the SOAP message element that must be digitally signed. Use the following format:

    `{`*`Namespace`*`}`*`LocalPart`*

    where *`LocalPart`* refers to the name of the element in the SOAP message that must be digitally signed and *`Namespace`* refers to its namespace. Use the WSDL of the Web Service to get these values.

    For example:

    `{http://schemas.xmlsoap.org/soap/envelope/}AccountNumber`

> **Note**
>
> You can specify only those elements that have already been configured to be digitally signed in the WS-Policy of the Web Service. For details, see Configuring Message-Level Security in *Securing WebLogic Web Services for Oracle WebLogic Server*.

Example of Using the `@Policy` and `@Policies` JWS Annotations

The following example shows how to use the `@Policy` and `@Policies` JWS annotations, with the relevant sections shown in bold:

Example 5-1 Using @Policy and @Policies Annotations

```
package wssp12.wss10;

import weblogic.jws.WLHttpTransport;
import weblogic.jws.Policy;
import weblogic.jws.Policies;

import jakarta.jws.WebService;
import jakarta.jws.WebMethod;
import jakarta.jws.Oneway;

/**
 * This web service demonstrates how to use WS-SecurityPolicy 1.2
 * to enable message-level security specified in WS-Security 1.0.
 *
 * The service authenticates the client with a username token.
 * Both the request and response messages are signed and encrypted with X509
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
 certificates.
 *
*/
@WebService(name="Simple", targetNamespace="http://example.org")
@WLHttpTransport(contextPath="/wssp12/wss10",
 serviceUri="UsernameTokenPlainX509SignAndEncrypt")
@Policy(uri="policy:Wssp1.2-2007-Wss1.0-UsernameToken-Plain-X509-Basic256.xml")
public class UsernameTokenPlainX509SignAndEncrypt {

  @WebMethod
  @Policies({
      @Policy(uri="policy:Wssp1.2-2007-SignBody.xml"),
      @Policy(uri="policy:Wssp1.2-2007-EncryptBody.xml")})
  public String echo(String s) {

    return s;
  }

  @WebMethod
  @Policies({
      @Policy(uri="policy:Wssp1.2-2007-SignBody.xml"),
      @Policy(uri="policy:Wssp1.2-2007-Sign-Wsa-Headers.xml")})
  public String echoWithWsa(String s) {
    return s;
  }

  @WebMethod
  @Policy(uri="policy:Wssp1.2-2007-SignBody.xml",
 direction=Policy.Direction.inbound)
  @Oneway
  public void echoOneway(String s) {
    System.out.println("s = " + s);
  }

  @WebMethod
  @Policies({
      @Policy(uri="policy:Wssp1.2-2007-Wss1.0-X509-Basic256.xml",
direction=Policy.Direction.inbound),
      @Policy(uri="policy:Wssp1.2-2007-SignBody.xml",
direction=Policy.Direction.inbound)
  })
  @Oneway
  public void echoOnewayX509(String s) {
    System.out.println("X509SignEncrypt.echoOneway: " + s);
  }
}
```

The following section of the example is the binding policy for the web service, specifying the policy:

```
@WebService(name="Simple", targetNamespace="http://example.org")
@WLHttpTransport(contextPath="/wssp12/wss10",
 serviceUri="UsernameTokenPlainX509SignAndEncrypt")
@Policy(uri="policy:Wssp1.2-2007-Wss1.0-UsernameToken-Plain-X509-Basic256.xml")
```

In the example, security policy files are attached to the web service at the method level. The specified policy files are those predefined with WebLogic Server, which means that the developers do not need to create their own files or package them in the corresponding archive.

The `Wssp1.2-2007-SignBody.xml` policy file specifies that the body and WebLogic system headers of both the request and response SOAP message be digitally signed. The

Copyright © 2007, 2026, Oracle and/or its affiliates.

`Wssp1.2-2007-EncryptBody.xml` policy file specifies that the body of both the request and response SOAP messages be encrypted.

Context Element Policy Conditions

You can use the context element conditions to create security policies based on the value of HTTP Servlet Request attributes, HTTP Session attributes, and EJB method parameters. WebLogic Server retrieves this information from the ContextHandler object and allows you to defined policy conditions based on the values. When using any of these conditions, it is your responsibility to ensure that the attribute or parameter/value pairs apply to the context in which you are using them. See ContextHandlers and WebLogic Resources in *Developing Security Providers for Oracle WebLogic Server*.

The context element role conditions available in this release of WebLogic Server are:

- `Context element defined`—Allows access based on the existence of a specified attribute or parameter.
- `Context element's value equals a numeric constant`—Allows access based on a specified attribute or parameter's number value (or string representing a double number) being equal to a specified `double` number.
- `Context element's value is greater than a numeric constant`—Allows access based on a specified attribute or parameter's number value (or string representing a double number) being greater than a specified `double` number.
- `Context element's value is less than a numeric constant`—Allows access based on a specified attribute or parameter's number value (or string representing a double number) being less than a specified `double` number.
- `Context element's value equals a string constant`—Allows access based on a specified attribute or parameter's string value being equal to a specified string.

Attaching OWSM Security Policies to JAX-WS Web Services

The OWSM WS-Security policy attachment model is similar to that of the WebLogic web service policies. You can attach OWSM policies to WebLogic JAX-WS web services using one of the following methods:

- Policy annotations at design time, as described in Attaching Policies to Java EE Web Services and Clients Using Annotations in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Fusion Middleware Control or WLST at runtime, as described in the following sections in *Securing Web Services and Managing Policies with Oracle Web Services Manager*:
  - Attaching Policies Directly Using Fusion Middleware Control
  - Attaching Policies Globally Using Fusion Middleware Control
  - Attaching Policies Directly to Java EE Web Services Using WLST
  - Attaching Policies Globally Using WLST

You can attach only one type of security policy to a web service, either WebLogic web service security policies or OWSM policies. You cannot attach both WebLogic web service policies and OWSM policies to the same web service, through either the annotation mechanism, Fusion Middleware Control, or a combination of the tools.

You can attach an OWSM security policy only to a JAX-WS web service.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Using to Manage Security Policies

> **Note**
>
> This section describes the features and functions that are available in security realms that are using the WebLogic Authorization provider or the WebLogic XACML Authorization provider. If your security realm uses a third-party Authorization provider, refer to the third-party documentation for information on how to add policies to the provider database.

You can use the to access WebLogic resources for creating and modifying security policies. See Security Policies and Roles in *Oracle WebLogic Remote Console Online Help*.

Copyright © 2007, 2026, Oracle and/or its affiliates.

6

Users, Groups, And Security Roles

Understand the features and functions of users, groups, and security roles within security realms in WebLogic Server.
This chapter includes the following sections:

- Overview of Users and Groups
- Default Users
- Default Groups
- Overview of Security Roles
- Types of Security Roles: Global Roles and Scoped Roles
- Default Global Roles
- Security Role Conditions
- Using to Manage Users, Groups, and Roles

Overview of Users and Groups

A user is an entity that can be authenticated. A user can be a person or a software entity, such as a Java client. Each user is given a unique identity within a security realm. For efficient security management, Oracle recommends adding users to groups. A group is a collection of users who usually have something in common, such as working in the same department in a company.

Default Users

When you create a WebLogic Server domain, three user accounts are created in the security realm by default: the Administration user account that you define during domain creation, `OracleSystemUser` and `LCMUser`.

- The default WebLogic Administrator user account for the domain is used to boot and connect to the domain's Administration Server. You specify the user name and password for this account when you create the domain. You should make note of these credentials because you will need them to start and access the Administration Server, and to log into the .

  Note:

  – For additional security, avoid choosing an obvious name such as `system`, `admin`, `administrator`, or `weblogic` for your system administrator user accounts. If you enable secured production mode, then WebLogic Server logs warnings if users in the administrator group have these obvious user names.

  – You should create at least one additional user and assign them the Admin security role. Be sure to use unique names that cannot be easily guessed. Having at least two system administrator user accounts helps to ensure that one user maintains account access in case another user becomes locked out by a dictionary/brute force attack.

Copyright © 2007, 2026, Oracle and/or its affiliates.

- `OracleSystemUser` facilitates the installation of Fusion Middleware components that run on WebLogic Server, such as Oracle SOA Suite, and is the default user for Fusion Middleware components such as Oracle Web Services Manager.

  Note:

  - `OracleSystemUser` is assigned the same password that you specify for the default Administration user when you create the domain. If necessary, you can change this password in the .
  - `OracleSystemUser` is assigned to the `OracleSystemGroup` and is granted the `OracleSystemRole` by default. This user is not a member of any of the default WebLogic Server groups or global roles. This account does not receive the default WebLogic administrative roles, but it retains `OracleSystemRole`. Protect its credentials and change the shared initial password according to your domain's credential-management policy.
  - Although you can remove `OracleSystemUser` with no issues if you have a standalone WebLogic Server installation, if you do so you will encounter problems if you attempt to install Fusion Middleware components on top of WebLogic Server at a later date, or use Oracle Web Services Manager.

- `LCMUser` is an internal WebLogic account that is required for software features such as Lifecycle Manager, RESTful Management Services, and Fusion Middleware Control.

  Note:

  - The password for `LCMUser` is randomly generated, therefore you cannot log into the as this user. Oracle recommends that you do not change this password.
  - Do not remove the `LCMUser` account. If you remove `LCMUser` from the WebLogic security realm, then you are disabling the maintenance capability of the Lifecycle Manager tool and you may encounter errors if you later attempt to upgrade your installation.
  - If you replace the embedded LDAP with an external provider and you plan to use the WebLogic Server Lifecycle Manager functionality, then you need to recreate this user in the new provider. See Configuring Authentication Providers in *Administering Security for Oracle WebLogic Server*.

Default Groups

Table 6-1 lists the groups that WebLogic Server defines in the security realm that it installs. By default, if you add a user to one of these groups, you also place the user in one of the default global security roles (see Default Global Roles).

Table 6-1 Default Groups

Group Name	Membership
`Administrators`	By default, this group contains the user information entered as part of the installation process (that is, the Configuration Wizard). Any user assigned to the `Administrators` group is granted the `Admin` security role by default. See #unique_91.
`Deployers`	By default, this group is empty. Any user assigned to the `Deployers` group is granted the `Deployer` security role by default.
`Operators`	By default, this group is empty. Any user assigned to the `Operators` group is granted the `Operator` security role by default.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 6-1 (Cont.) Default Groups

Group Name	Membership
`Monitors`	By default, this group is empty. Any user assigned to the `Monitors` group is granted the `Monitor` security role by default.
`AppTesters`	By default, this group is empty. Any user assigned to the `AppTesters` group is granted the `AppTester` security role by default.
`CrossDomainConnectors`	By default, this group is empty. Any user assigned to the `CrossDomainConnectors` group is granted the `CrossDomainConnector` security role by default.
`AdminChannelUsers`	By default, this group is empty. Any user assigned to the `AdminChannelUsers` group is granted the `AdminChannelUser` security role by default.
`OracleSystemGroup`	By default, this group contains the user `OracleSystemUser` and is granted the `OracleSystemRole` role by default.

Run Time Groups

At run time, WebLogic Server places all users in the following groups:

- `users`. This group contains all users who have been authenticated.
- `everyone`. This group contains all anonymous users and, because it contains the `users` group, all users who have been authenticated.

Unlike the groups in Table 6-1 (or other groups that you create), you cannot add or remove users directly to these groups; WebLogic Server assigns users to them dynamically. These groups do not appear under Groups in the , and they are not exported with the authentication database.

Revised configWss_Service.py

The `configWss_Service.py` script is similar to `configWss.py`, but it is used only when the source and destination web service are hosted in two servers.

The `configWss_Service.py` file is the same as that in *EXAMPLES_HOME*`\wl_server\examples\src\examples\webservices\wsrm_security\configWss_Service.py`, with the changes shown in bold. The `build.xml` file provides the command input.

> **Note**
>
> Long lines in this script have been formatted for readability.

Example 6-1 configWss_Service.py

```
:
 #Create credential provider for SCT
cpName='default_sct_cp'
wtm=defaultWss.lookupWebserviceCredentialProvider(cpName)
if wtm == None:
  print 'creating new webservice credential provider : ' + cpName
  wtm = defaultWss.createWebserviceCredentialProvider(cpName)
 wtm.setClassName('weblogic.wsee.security.wssc.
      v13.sct.ServerSCCredentialProvider')
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
  wtm.setTokenType('sct')
  cpm = wtm.createConfigurationProperty('TokenLifeTime')
  cpm.setValue('43200000')
else:
  print 'found exsiting bean for: ' + cpName

#Create credential provider for DK
cpName='default_dk_cp'
wtm=defaultWss.lookupWebserviceCredentialProvider(cpName)
if wtm == None:
        wtm = defaultWss.createWebserviceCredentialProvider(cpName)
      wtm.setClassName('weblogic.wsee.security.wssc.v13.dk.
           DKCredentialProvider')
        wtm.setTokenType('dk')
        cpm = wtm.createConfigurationProperty('Label')
        cpm.setValue('WS-SecureConversationWS-SecureConversation')
        cpm = wtm.createConfigurationProperty('Length')
        cpm.setValue('16')
else:
  print 'found existing bean for: DK ' + cpName
:
```

Overview of Security Roles

A security role is an identity granted to users or groups based on specific conditions. Multiple users or groups can be granted the same security role and a user or group can be in more than one security role. Security roles are used by policies to determine who can access a WebLogic resource. (See Security Policies.)

Like a security group, a role grants an identity to a user. Security roles differ from groups as follows:

- Security roles can be scoped to specific WebLogic resources within a single application in a WebLogic Server domain. Groups, on the other hand, are always scoped to an entire WebLogic Server domain. See Types of Security Roles: Global Roles and Scoped Roles.
- Security roles are computed and granted to users or groups dynamically, based on conditions such as user name, group membership, or the time of day. Groups are static.

  The process of computing and granting roles is referred to as role mapping and occurs just before the WebLogic Security Service renders an access decision for a protected WebLogic resource. An access decision is the component of an Authorization provider that determines whether a subject has permission to perform a given operation on a WebLogic resource. (See Access Decisions in *Developing Security Providers for Oracle WebLogic Server*.)

Types of Security Roles: Global Roles and Scoped Roles

There are two types of security roles in WebLogic Server:

- A **global** security role can be used in any security policy. Oracle provides several default global roles that you can use out of the box to secure your WebLogic resources; these are described in Default Global Roles.

Copyright © 2007, 2026, Oracle and/or its affiliates.

> **Note**
>
> If you are implementing security using Jakarta Authorization 3.0, global security roles cannot be used.

- A **scoped** role can be used only in policies that are defined for a specific instance of a WebLogic resource (such as a method on an EJB or a branch of a JNDI tree). You might never need to use scoped roles. They are provided for their flexibility and are an extra feature for advanced customers.

Default Global Roles

Table 6-2 lists the global roles that WebLogic Server defines in the security realm that it installs. The table also summarizes the access that the default security policies grant to each role and indicates which groups are in each role by default. Table 6-3 summarizes the permissions in the for the Admin, Deployer, Operator, and Monitor global roles.

> **Note**
>
> Do not delete these roles. They are used in the default security policies that protect most types of WebLogic resources. In addition, they are used by the MBean security layer. If you delete the Admin role, no one will be able to modify the configuration of a running domain. See Maintaining a Consistent Security Scheme.

Table 6-2 Default Global Roles, Privileges, and Default Group Assignments

Global Role	Default Policies Grant Access To . . .	Default Conditions Include This Group . . .
`Admin`	• View the server configuration, including the encrypted value of some encrypted attributes. • Modify the entire server configuration. • Deploy Enterprise Applications and Web application, EJB, Jakarta EE Connector, and Web Service modules. • Start, resume, and stop servers.	`Administrators`
`AdminChannelUser`	Access the administrative channel, `AdminChannel`.	`AdminChannelUsers`, `Administrators`, `Deployers`, `Operators`, `Monitors`, and `AppTesters`
`Anonymous`	All users (the group `everyone`) are granted this global role. **Note:** This global role is provided as a convenience, and can be specified in the `weblogic.xml` and `weblogic-ejb-jar.xml` deployment descriptors. See weblogic.xml Deployment Descriptor Elements in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server* and weblogic-ejb-jar.xml Deployment Descriptor Reference in *Developing Jakarta Enterprise Beans Using Deployment Descriptors*.	`everyone`

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 6-2 (Cont.) Default Global Roles, Privileges, and Default Group Assignments

Global Role	Default Policies Grant Access To . . .	Default Conditions Include This Group . . .
`Deployer`	• View the server configuration, including some encrypted attributes related to deployment activities. • Change startup and shutdown classes, Web applications, JDBC data pool connections, EJB, Jakarta EE Connector, Web Service, and WebLogic Tuxedo Connector components. If applicable, edit deployment descriptors. • Access deployment operations in the Jakarta EE Deployment Implementation. See Understanding WebLogic Server Deployment in *Deploying Applications to Oracle WebLogic Server*.	`Deployers`
`Operator`	• View the server configuration, *except* for encrypted attributes. • Start, resume, and stop servers. • Start and stop applications using MBean APIs and WLST. You cannot start and stop applications in the . See Table 6-3.	`Operators`
`Monitor`	View the server configuration, *except* for encrypted attributes. This security role effectively provides read-only access to the , WLST, and MBean APIs.	`Monitors`
`AppTester`	Access applications for testing purposes that are running in Administration mode. See Administration Mode for Isolating Production Applications in *Deploying Applications to Oracle WebLogic Server*.	`AppTesters`
`CrossDomainConnector`	Make inter-domain calls from foreign domains. See Enabling Trust Between WebLogic Server Domains in *Administering Security for Oracle WebLogic Server*.	`CrossDomainConnectors`
`OracleSystemRole`	Assert identity on behalf of users whose WS-Security tokens have been authenticated. **Note:** This global role is provided for use by Oracle Web Services Manager.	`OracleSystemGroup`

Table 6-3 summarizes the privileges provided in the for the Admin, Deployer, Operator, and Monitor global roles. Note that the information provided in the table is based on the default out-of-the-box security policy for WebLogic Resources and MBeans. You can manage the default security policies in the , as described in Security Policies and Roles in *Oracle WebLogic Remote Console Online Help*.

Table 6-3 WebLogic Server Privileges for Default Global Roles

Privileges	Administrator	Deployer	Operator	Monitor
Edit session operations: start or release session, activate or undo changes	Yes	Yes	No	No
Server, Cluster, Template, or Machine				
Lifecyle operations: create, delete	Yes	No	No	No

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 6-3 (Cont.) WebLogic Server Privileges for Default Global Roles

Privileges	Administrator	Deployer	Operator	Monitor
Modify configuration	Yes	No	No	No
Control operations: start, stop, resume	Yes	No	Yes	No
View configuration	Yes	Yes	Yes	Yes
Application Deployments				
Lifecycle operations: deploy, undeploy, redeploy	Yes	Yes	No	No
Modify configuration	Yes	Yes	No	No
Control operations: start, stop	Yes	Yes	No	No
JDBC and JMS resources				
Lifecyle operations: create, delete	Yes	Yes	No	No
Modify configuration	Yes	Yes	No	No
Control operations: start, stop	Yes	No	No	No
View configuration	Yes	Yes	Yes	Yes
Startup and Shutdown Classes, Coherence Clusters				
Lifecyle operations: create, delete	Yes	No	No	No
Modify configuration	Yes	Yes	No	No
View configuration	Yes	Yes	Yes	Yes

Security Role Conditions

To determine who is in a security role at run time, a role contains one or more conditions. For example, a basic role might simply be the Administrator group. At run time, the WebLogic Security Service interprets this policy as place the Administrator group in this role. You can create more complex conditions and combine them using the logical operators AND and OR (which is an inclusive OR). You can also negate any condition, which would make sure that a user is not in the role. A user or group is granted the role when the complete condition expression, including its AND, OR, and negation operators, evaluates to `true`. More restrictive expressions should come later in a role statement.

In the , the WebLogic Server Role Mapping providers display the three built-in policy conditions described in the following sections:

- Basic Role Conditions
- Date and Time Role Conditions
- Context Element Role Conditions

Copyright © 2007, 2026, Oracle and/or its affiliates.

> **Note**
>
> The following sections describe the conditions that are available in realms that use the WebLogic Role Mapping provider or the WebLogic XACML Role Mapping provider. If your security realm uses a third-party Role Mapping provider, refer to the third-party documentation for information about its capabilities.

Basic Role Conditions

The basic role conditions available in this release of WebLogic Server are:

- `User`—Adds the specified user to the role. For example, you might create a condition indicating that only the user `John` can be granted the `BankTeller` security role.

  > **Note**
  >
  > Instead of the User condition, Oracle recommends adding the user to a Group and using the Group condition in the security role. If you do use the User condition, ensure that you update any existing roles when the user is deleted from the authentication store.

- `Group`—Adds the specified group to the role. For example, you might create a condition indicating that only users in the group `FullTimeBankEmployees` can be granted the `BankTeller` security role.

  As a minimum requirement, Oracle recommends this role condition for more efficient security management.

- `Server is in development mode`—Adds principals to the role only when the server is running in development mode. See Creating a WebLogic Domain in *Creating WebLogic Domains Using the Configuration Wizard*.
- `Allow access to everyone`—Adds all users and groups to the role.
- `Deny access to everyone`—Prevents any user or group from being in the role.

Date and Time Role Conditions

When you use any of the date and time role conditions, the security role is granted to *all users* for the date or time you specify, unless you further restrict the users by adding one of the other role conditions. The date and time role conditions available in this release of WebLogic Server are:

- `Access occurs between specified hours`—Adds principals to the role only during the specified time period. For example, you might create a condition indicating that the `BankTeller` security role can only be granted to users when the bank is open.
- `Access occurs after`—Adds principals to the role only if the current time is after a specified time. For example, you might create a condition indicating that the `BankTeller` security role can only be granted to users after the bank opens or after a certain date and time.
- `Access occurs before`—Adds principals to the role only if the current time is before a specified time. For example, you might create a condition indicating that the `BankTeller`

Copyright © 2007, 2026, Oracle and/or its affiliates.

security role can only be granted to users before the bank closes or before a certain date and time.

- `Access occurs on specified days of the week`—Adds principals to the role only on specified days. For example, you might create a condition indicating that the `BankTeller` security role can only be granted to users on week days.
- `Access occurs on the specified day of the month`—Adds principals to the role only on an ordinal day of the month. For example, you might create a condition indicating that the `BankTeller` security role can only be granted to users on the first day of each month.
- `Access occurs after the specified day of the month`—Creates a condition for a security role based on a time after an ordinal day in the month. For example, you might create a condition indicating that the `BankTeller` security role can only be granted to users after the 15th day of the month.
- `Access occurs before the specified day of the month`—Adds principals to the role only if the current day is before an ordinal day in the month. For example, you might create a condition indicating that the `BankTeller` security role can only be granted to users before the 15th day of the month.

Context Element Role Conditions

You can use the context element conditions to create security roles based on the value of HTTP Servlet Request attributes, HTTP Session attributes, and EJB method parameters. WebLogic Server retrieves this information from the ContextHandler object and allows you to defined role conditions based on the values. When using any of these conditions, it is your responsibility to ensure that the attribute or parameter/value pairs apply to the context in which you are using them. See ContextHandlers and WebLogic Resources in *Developing Security Providers for Oracle WebLogic Server*.

The context element role conditions available in this release of WebLogic Server are:

- `Context element defined`—Adds principals to the role based on the existence of a specified attribute or parameter.
- `Context element's value equals a numeric constant`—Adds principals to the role based on a specified attribute or parameter's number value (or string representing a double number) being equal to a specified `double` number.
- `Context element's value is greater than a numeric constant`—Adds principals to the role based on a specified attribute or parameter's number value (or string representing a double number) being greater than a specified `double` number.
- `Context element's value is less than a numeric constant`—Adds principals to the role based on a specified attribute or parameter's number value (or string representing a double number) being less than a specified `double` number
- `Context element's value equals a string constant`—Adds principals to the role based on a specified attribute or parameter's string value being equal to a specified string.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Using to Manage Users, Groups, and Roles

> **Note**
>
> This section describes the features that are available in realms that use the WebLogic Authentication provider and the WebLogic Role Mapping provider or the WebLogic XACML Role Mapping provider. If your security realm uses a third-party Authentication or Role Mapping provider, refer to the third-party documentation for information on its capabilities.

For information on adding users and groups to a security realm, see Users and Groups in *Oracle WebLogic Remote Console Online Help*.

For information on creating security roles, see Security Policies and Roles in *Oracle WebLogic Remote Console Online Help*.

Copyright © 2007, 2026, Oracle and/or its affiliates.

7

Using XACML Documents to Secure WebLogic Resources

Learn how to use the eXtensible Access Control Markup Language (XACML), an XML language for expressing authorization policies and role assignments, to secure WebLogic resources. You can create roles and policies in an XACML document and then use the WebLogic Scripting Tool (WLST) to add them to your security realm. This is useful if you need to create security roles or policies that are more complex than can be created with the , or if you are required to use a standard language. You can also export your realm's roles and policies to a XACML document and then import the document in other WebLogic Server realms.
This chapter includes the following sections:

- Prerequisites
- Adding a XACML Role or Policy to a Realm: Main Steps
- Creating Roles and Policies for Custom MBeans
- Exporting Roles and Policies to XACML Documents

> **Note**
>
> Always create a backup of a domain before you load XACML documents into a security realm. If you make a typographical or other type of error in an attribute description, you can cause the XACML provider to evaluate your realm's roles and policies as indeterminate, which locks all users (including the administrative user) out of the domain. See Caution: Indeterminate Results Can Lock Out All Users.

The WebLogic Server XACML Authorization Provider and the WebLogic Server XACML Role Mapping Provider implement the *XACML 2.0 Core Specification*, available at `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf`.

Prerequisites

Note the following prerequisites for using XACML documents to secure WebLogic resources:

- To add XACML authorization policies to a security realm, the realm must use either the WebLogic Server XACML Authorization Provider or a third party authorization provider that implements the `weblogic.management.security.authorization.PolicyStoreMBean` interface.
- To add XACML role assignments to a security realm, the realm must use either the WebLogic Server XACML Role Mapping Provider or a third party authorization provider that implements the `weblogic.management.security.authorization.PolicyStoreMBean` interface.

Copyright © 2007, 2026, Oracle and/or its affiliates.

- To secure a resource with a XACML authorization or role policy, you need the resource identifier (ID). Because WebLogic Server creates an immutable, unique ID when you deploy or create a resource, you must deploy the resource before creating a policy for it.
- To secure an EJB or Web application, you must deploy using the Custom Roles or Custom Roles and Policies security model. You cannot use a XACML document to create roles and policies for an EJB or Web application that you have deployed using the Deployment Descriptor Only security model. See .

Adding a XACML Role or Policy to a Realm: Main Steps

You can create a XACML document that describes roles and policies and then use the WebLogic Scripting Tool to add the policy or role to your security realm.

The main steps for this process are as follows:

1. Determine Which Resource to Secure
2. Get the ID of the Resource to Secure
3. Create XACML Documents
4. #unique_106
5. Verify That Your Roles and Policies Are in the Realm

Caution: Indeterminate Results Can Lock Out All Users

The XACML specification requires that if the decision engine is unable to process a decision point, the engine returns a result of indeterminate. Depending on the combining algorithms that you use for a decision point and its associated decision points, an indeterminate result can propagate to the top of the decision and cause the provider to deny access to all requests.

For example, the following attribute specifies `MustBePresent='true'` and contains a spelling mistake (`ancester` instead of `ancestor`). It will evaluate as indeterminate and will cause the security provider to deny access:

```
<ResourceAttributeDesignator
    AttributeId="urn:oasis:names:tc:xacml:2.0:resource:resource-ancester"
    DataType="http://www.w3.org/2001/XMLSchema#string"
    MustBePresent="true"/>
```

Determine Which Resource to Secure

WebLogic Server organizes its resources into a hierarchy. If you use the or a Jakarta EE deployment descriptor (instead of a XACML document) to secure WebLogic resources, policies that you create on resources that are higher in the hierarchy act as default policies for resources lower in the same hierarchy. Policies lower in a hierarchy always override policies higher in the hierarchy.

Your XACML document can encode this hierarchical protection scheme, though XACML's hierarchical model differs slightly from WebLogic Server. See Comparison of WebLogic Server and XACML Security Models.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Get the ID of the Resource to Secure

WebLogic Server creates an immutable, unique identifier (ID) when you deploy or create a resource, and your XACML document must include a resource identifier that specifies the WebLogic Server ID.

To find the ID that WebLogic Server has assigned to a resource:

1. Enable security auditing for your realm by configuring an Auditing provider.

   See Configure an Auditing Provider in *Oracle WebLogic Remote Console Online Help*. Take note of the location in which the Auditing provider saves its log files (by default, in the server's `logs` directory).

2. Deploy or create the resource.

3. Send a request from an external client to the resource.

   For example, use a Web Service client to invoke a Web Service method that you want to secure.This will trigger an event that causes the Auditing provider to generate a message for the resource.

   > **Note**
   >
   > The Web Service client needs to authenticate itself before it can be granted access to the Weblogic resource that is secured by the roles and policies specified in the XACML document.

4. Open the log file for the Auditing provider and find the entry for the event that you triggered.

   For example, if you configure the WebLogic Server Default Auditor to generate messages for severity level `INFORMATION` and higher, when you invoke the `sayHello` method from a Web Service named `SimpleSoapPort`, the audit log contains the following entries, one from the Role Mapping provider and the other from the Authorization provider:

   ```
   #### Audit Record Begin <Mar 30, 2006 9:24:12 AM>
   <Severity =INFORMATION>
   <<<Event Type = RoleManager Audit Event ><Subject: 0>
   <<webservices>><type=<webservices>,
   application=webservicesJwsSimpleEar, contextPath=/jws_basic_simple,
   webService=SimpleSoapPort, method=sayHello,
   signature={java.lang.String}><>>> Audit Record End ####
   #### Audit Record Begin <Mar 30, 2006 9:24:12 AM>
   <Severity =SUCCESS>
   <<<Event Type = Authorization Audit Event V2 ><Subject: 0>
   <ONCE><<webservices>><type=<webservices>,
   application=webservicesJwsSimpleEar, contextPath=/jws_basic_simple,
   webService=SimpleSoapPort, method=sayHello,
   signature={java.lang.String}>>> Audit Record End ####
   ```

   The resource ID for the `sayHello` method is:

   ```
   type=<webservices>,
   application=webservicesJwsSimpleEar, contextPath=/jws_basic_simple,
   webService=SimpleSoapPort, method=sayHello,
   signature={java.lang.String}
   ```


Copyright © 2007, 2026, Oracle and/or its affiliates.

5. Edit the resource ID from the auditing record to specify the resource that you want to protect.

   The IDs in the audit log are for resources that are at the bottom of the WebLogic Server resource hierarchy. Typically, instead of creating policies for a specific operation (such as a Web Service or EJB method or an HTTP method on a specific URL), you design policies for resources higher in the hierarchy, such as for a URL pattern or an entire Web Service.

   You can derive the following resource IDs from the resource ID from the previous step:

   - The ID for the Web Service that contains the sayHello method is:

     ```
     type=<webservices>,
     application=webservicesJwsSimpleEar, contextPath=/jws_basic_simple,
     webService=SimpleSoapPort
     ```

   - The ID for the application that contains the Web Service is:

     ```
     type=<application>,
     application=webservicesJwsSimpleEar
     ```

     Note that resource ID for an application specifies `type=<application>`.

   - The ID for the Web Service type, which you would use to create a root-level policy for all Web Services is:

     ```
     type=<webservices>
     ```

   For information about root-level policies and the hierarchy of resources, see Using Policies to Protect Multiple Resources.

Create XACML Documents

If you want to create role assignments and authorization policies, create two XACML documents: one that describes your roles and another that describes your policies. You load one of the documents into the Role Mapping provider's store and the other into the Authorization provider's store.

For information about using XACML to describe WebLogic Server resources, see #unique_109.

Example: Defining Role Assignments

The syntax for describing role assignments in a XACML document is specified in the OASIS RBAC Profile specification, available at `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf`. (WebLogic Server supports only a subset of this specification.)

The syntax requires the following elements:

- A `Policy` parent element.
- Under `Policy`, a `Target` element.
  - Under `Target`, at least one `Resource` element that contains the following `ResourceMatch` elements:

    (Optional) One `ResourceMatch` element to identify the name of the role. If you do not include this `ResourceMatch` element, then the role policy applies to all roles in the realm. The `MatchId` attribute may specify function identifiers and, thus, wildcard role names. The `DataType` attribute must specify the `string` type.

Copyright © 2007, 2026, Oracle and/or its affiliates.

(Optional) Another `ResourceMatch` element to identify the WebLogic resource to which the role applies. If you do not include this `ResourceMatch` element, the role applies to all WebLogic resources.

- Under `Target`, an `Action` element that indicates that the policy applies to a role instead of some other type of resource.

- Under `Policy`, one or more `Rule` elements that define which users, groups, or roles are in the role.

The XACML document in Example 7-1 specifies that a role named `MyRole` role can be used with the `SimpleSoapPort` Web Service. It also specifies that the `webServiceGroup` group is in the role.

> **Note**
>
> When specifying values in the `<AttributeValue>` element, you must not include extraneous space characters. In particular:
>
> - Do not precede the first value with more than a single space character.
> - When specifying multiple values, do not separate each value by more than a single space character.
> - Make sure there are no space characters that follow the last value.
>
> For example:
>
> ```
> <AttributeValue DataType="http://www.w3.org/2001/
> XMLSchema#string">value1, value2, value3</AttributeValue>
> ```

Example 7-1 XACML Policy for a Role

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicyId="urn:mycompany:myapplication:myRolePolicy"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
   <Description>Grp(group1)</Description>
   <Target>
    <Resources>
     <Resource>
      <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">type=&lt;webservices&gt;,
application=webservicesJwsSimpleEar, contextPath=/jws_basic_simple, webService=SimpleSoapPort</
AttributeValue>
        <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:resource:resource-
ancestor-or-self" DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
      </ResourceMatch>
      <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">MyRole</AttributeValue>
        <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
      </ResourceMatch>
     </Resource>
    </Resources>
    <Actions>
     <Action>
       <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
         <AttributeValue DataType="http://www.w3.org/2001/
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
XMLSchema#anyURI">urn:oasis:names:tc:xacml:2.0:actions:enableRole</AttributeValue>
         <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#anyURI" MustBePresent="true"/>
       </ActionMatch>
     </Action>
    </Actions>
   </Target>
   <Rule RuleId="primary-rule" Effect="Permit">
    <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">webServiceGroup</
AttributeValue>
        <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:subject:group"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </Apply>
    </Condition>
   </Rule>
   <Rule RuleId="deny-rule" Effect="Deny"/>
</Policy>
```

Example: Defining Authorization Policies

The XACML document in Example 7-2 specifies that only the `MyRole` role can access the `SimpleSoapPort` Web Service.

Example 7-2 XACML Policy for a Web Service

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicyId="urn:mycompany:myapplication:myPolicy" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-
combining-algorithm:first-applicable">
   <Description>Rol(MyRole)</Description>
   <Target>
    <Resources>
     <Resource>
      <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">type=&lt;webservices&gt;,
application=webservicesJwsSimpleEar, contextPath=/jws_basic_simple, webService=SimpleSoapPort</
AttributeValue>
        <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:resource:resource-
ancestor-or-self" DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
      </ResourceMatch>
     </Resource>
    </Resources>
   </Target>
   <Rule RuleId="primary-rule" Effect="Permit">
    <Condition>
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">MyRole</AttributeValue>
       <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
     </Apply>
    </Condition>
   </Rule>
   <Rule RuleId="deny-rule" Effect="Deny"/>
</Policy>
```

Configuring Transport-Level Security Through Policy: Main Steps

To configure transport-level web services security via one or more policy files:

Copyright © 2007, 2026, Oracle and/or its affiliates.

1. As outlined in Prerequisite: Configure SSL, configure SSL for the core WebLogic Server security subsystem.

   You can configure one-way SSL where WebLogic Server is required to present a certificate to the client application, or two-way SSL where both the client applications and WebLogic Server present certificates to each other.

   To configure two-way or one-way SSL for the core WebLogic Server security subsystem, see Configuring SSL in *Administering Security for Oracle WebLogic Server*.

2. Use `@Policy` or `@Policies` JWS annotations in your JWS file, or associate policy files only at runtime using , or specify some policy files using the annotations and then associate additional ones at runtime.

   See #unique_114/unique_114_Connect_42_CHDEBHIJ for a description of the available transport-level policies.

   > **Note**
   >
   > If you specify a transport-level security policy for your web service, it must be at the class level.
   >
   > In addition, the transport-level security policy must apply to both the inbound and outbound directions. That is, you cannot have HTTPS for inbound and HTTP for outbound.

   The following example attaches the policy at the class level:

   ```
   @Policy(uri="policy:Wssp1.2-2007-Saml2.0-Bearer-Https.xml")
   public class EchoService {
      ....
   ```

3. If you added `@Policy` or `@Policies` JWS annotations in your JWS file, compile and redeploy your web service as part of the normal iterative development process.

4. When you run the client application that invokes the web service, specify certain properties to indicate the SSL implementation that your application should use. In particular:

   - To specify the Sun SSL implementation, use the following properties:

     ```
     -Djavax.net.ssl.trustStore=trustStore
     ```

     where *trustStore* specifies the name of the client-side truststore that contains the list of trusted certificates (one of which should be the server's certificate).

     Keep certificate-path and hostname verification enabled. The server certificate must chain to a certificate trusted by the client, and its subject alternative name must match the hostname in the HTTPS endpoint. Correct the certificate, truststore, or endpoint hostname instead of disabling either check.

     See Configuring Two-Way SSL for a Client Application for additional details about two-way SSL.

Verify That Your Roles and Policies Are in the Realm

The does not display roles and policies that you add from a XACML document.

Instead, to verify that your roles and policies were added to the realm, see Exporting Roles and Policies to XACML Documents.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Creating Roles and Policies for Custom MBeans

A Managed Bean (MBean) is a standard for exposing management data and operations for a resource. Your application developers can greatly reduce the cost of operating and maintaining your applications by creating MBeans (custom MBeans) to monitor and manage your applications. See *Developing Manageable Applications Using JMX for Oracle WebLogic Server*.

If you register custom MBeans in a WebLogic Server MBean server, you can create a XACML document that defines who can access your MBeans. In addition to the prerequisites described in Prerequisites, note that your MBean's object name must include a `Type=value`key property.

The main steps for creating roles and policies for custom MBeans are:

1. Determine the Resource IDs for a Custom MBean
2. Create XACML Documents
3. #unique_106
4. Verify That Your Roles and Policies Are in the Realm

Determine the Resource IDs for a Custom MBean

When you register a custom MBean with a WebLogic MBean server, the WebLogic security service creates two resource IDs for each attribute in the MBean: one for the attribute's getter method and another for the setter. It creates one resource ID for each MBean operation.

The IDs use the following pattern:

```
type=<jmx>, operation=type-of-access, application=, mbeanType=type-name,
target=attribute-or-operation
```

In the preceding pattern:

- *`type-of-access`* specifies the type of access that the policy secures. Use one of the following values:
  - `get`

    Indicates that the policy controls who can read one or more MBean attributes.
  - `set`

    Indicates that the policy controls who can write one or more MBean attributes.
  - `invoke`

    Indicates that the policy controls who can invoke one or more MBean operations.
  - `create`

    Indicates that the policy controls who can use the MBean-server's `create` method to create an instance of an MBean.
  - `unregister`

    Indicates that the policy controls who can use the MBean-server's `unregister` method to unregister an instance of an MBean.
- *`type-name`* is the value of the MBean object name's `Type` key property.
- *`attribute-or-operation`* is the name of an MBean attribute or operation.

Copyright © 2007, 2026, Oracle and/or its affiliates.

For example, if you create an MBean that contains a single attribute named `NewUserCount` and an operation named `clearNewUserCount`, and if you register the MBean under the object name `medrec:Name=AdminReportMBean,Type=CustomMBeanType`, then the security service creates the following resource IDs:

```
type=<jmx>, operation=get, application=, mbeanType=CustomMBeanType,
target=NewUserCount

type=<jmx>, operation=set, application=, mbeanType=CustomMBeanType,
target=NewUserCount

type=<jmx>, operation=invoke, application=, mbeanType=CustomMBeanType,
target=clearNewUserCount
```

Exporting Roles and Policies to XACML Documents

To see a XACML representation of all roles and policies that are in your security realm, you can export the data from the Authorization and Role Mapping providers.

> **Note**
>
> Do not attempt round-trip editing of roles and policies. That is, do not export roles and policies, modify the XACML documents, and then import the modified documents. Editing exported files might result in an unusable WebLogic Server configuration and is not supported.

Copyright © 2007, 2026, Oracle and/or its affiliates.

A
Revised configWss.py

The ReliableEchoServiceImpl web service does not explicitly invoke any WebLogic Server API to handle the requirements imposed by any associated policy files, nor does this web service have to understand which, if any, security providers, tokens, or other such mechanisms are involved.

The script file `configWss.py` uses WLST to create and configure the default web service security configuration, `default_wss`, for the active security realm. (The default web service security configuration is used by *all* web services in the domain unless they have been explicitly programmed to use a different configuration.) Further, this script makes sure that x509 tokens are supported, creates the needed security providers, and so forth.

The `configWss.py` file is the same as that in *EXAMPLES_HOME*`\wl_server\examples\src\examples\webservices\wsrm_security\configWss.py`, with the changes shown in bold. The `build.xml` file provides the command input.

> **Note**
>
> Long lines in this script have been formatted for readability.

Example A-1 configWss.py

```
:
#Create credential provider for SCT
cpName='default_sct_cp'
wtm=defaultWss.lookupWebserviceCredentialProvider(cpName)
if wtm == None:
  print 'creating new webservice credential provider : ' + cpName
  wtm = defaultWss.createWebserviceCredentialProvider(cpName)
 wtm.setClassName('weblogic.wsee.security.wssc.v13.sct.
    ServerSCCredentialProvider')
  wtm.setTokenType('sct')
  cpm = wtm.createConfigurationProperty('TokenLifeTime')
  cpm.setValue('43200000')
else:
  print 'found exsiting bean for: ' + cpName

#Create credential provider for DK
cpName='default_dk_cp'
wtm=defaultWss.lookupWebserviceCredentialProvider(cpName)
if wtm == None:
        wtm = defaultWss.createWebserviceCredentialProvider(cpName)
      wtm.setClassName('weblogic.wsee.security.wssc.v13.
           dk.DKCredentialProvider')
        wtm.setTokenType('dk')
        cpm = wtm.createConfigurationProperty('Label')
        cpm.setValue('WS-SecureConversationWS-SecureConversation')
        cpm = wtm.createConfigurationProperty('Length')
        cpm.setValue('16')
else:
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
  print 'found exsiting bean for: DK ' + cpName
:
```

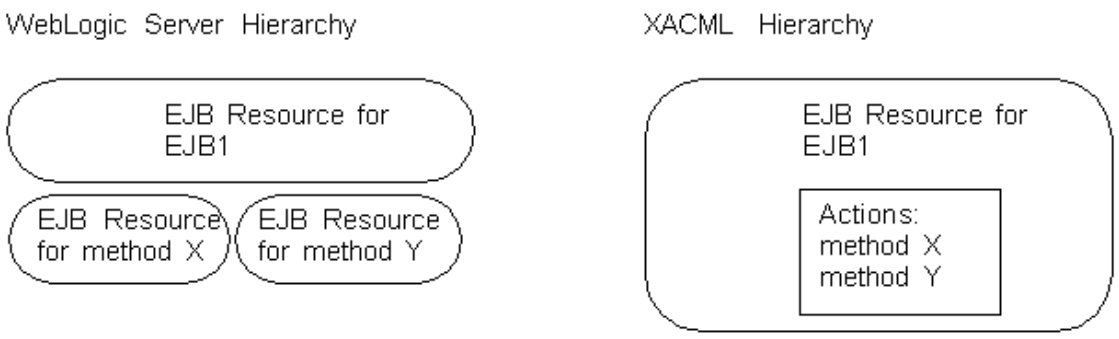
Comparison of WebLogic Server and XACML Security Models

The WebLogic Server model for representing resources and policies follows the model of Jakarta EE deployment descriptors. This Jakarta EE model creates a hierarchy of resources in which roles and authorization policies at the top of the hierarchy protect resources that are lower in the hierarchy. (See Protecting a Hierarchy of Resources.) Policies lower in a hierarchy always override policies higher in the hierarchy. The higher levels of the resource hierarchy contain enterprise applications, Web applications, and EJBs. The lowest levels of the resource hierarchy contain EJB methods, HTTP methods on specific URL patterns, and MBean getters and setters.

The XACML model also recognizes a hierarchy of resources. Unlike the native WebLogic Server model, your XACML policies must specify how to interpret cases in which a resource is protected by its own policy and by a policy on the resource's parent or ancestor.

In addition, a XACML document typically distinguishes between a resource and the actions of a resource. For example, a XACML document defines a resource such as an EJB, and then defines an action within the EJB resource to represent a method in the EJB. The native WebLogic Server model considers an EJB and each EJB method to be resources. See Figure A-1.

Figure A-1 WebLogic Resource Hierarchy Includes Methods



While it is possible to describe an action such as an EJB method as a XACML resource, a more natural expression in XACML would define an EJB as a resource and an EJB method as an action within the resource.

Comparison of Terminology

The WebLogic Server terminology for describing resources and policies follows the model of Jakarta EE deployment descriptors. This Jakarta EE model uses the following terms to describe key concepts:

- **Role**, which contains one or more conditions that describe which users or groups are in the role at any given time. A role **expression** is a collection of conditions and the name of a user or group whom the conditions add to the role. The collection of expressions is the role **statement**.
- **Policy**, which contains one or more conditions that describe who can access a resource at any given time. A policy expression is a collection of conditions and the name of a user,

Copyright © 2007, 2026, Oracle and/or its affiliates.

group, or role whom the conditions allow access. The collection of expressions is the policy statement.

In XACML, a set of **rules** comprise a policy, and policies can be used to determine who is in a role or who can access a resource. In general, a XACML policy is equivalent to a role statement or policy statement in WebLogic Server.

Description of Data Types

Oracle implements support for all of the data types that are required by the XACML core specification. It supports additional, standard XML data types and provides a group of custom data types. This document uses the `bea:` prefix to indicate that a data type is a custom Oracle type.

For a description of all data types that the WebLogic XACML providers recognize, see `com.bea.common.security.xacml.Type` in *Java API Reference for Oracle WebLogic Server*.

Action Identifiers

XACML uses an `Action` element to identify an operation in a resource or a hierarchy of resources.

WebLogic Server supports all of the XACML `Action` identifiers, as described in the XACML 2.0 Core Specification, and adds support for an additional one that can appear anywhere that a standard XACML environment identifier can appear.

To identify operations in WebLogic Server resources (for example, to identify a specific EJB method), use action identifiers as described in Table A-1.

> **Note**
>
> While it is possible to use a resource identifier to describe an operation such as an EJB method, a more natural expression in XACML would use an action identifier. See Comparison of WebLogic Server and XACML Security Models.

Table A-1 Action Identifiers

To Identify...	Use This Identifier...
An operation	• Attribute ID: `urn:oasis:names:tc:xacml:1.0:action:action-id` • Data Type: `string` • Value: Depends on the type of resource that contains the operation. See Table A-2.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table A-1 (Cont.) Action Identifiers

To Identify...	Use This Identifier...
When the provider performs a security check	• Attribute ID: `urn:bea:xacml:2.0:action:direction` • Data Type: `string` • Values: `ONCE`, `PRIOR`, or `POST` The WebLogic Security SPI contains an optional feature that enables containers to specify when a provider performs a security check on a request: • `ONCE` requests an authorization check once with no preference of whether it is done before or after an operation. • `PRIOR` requests an authorization check prior to processing the request. • `POST` requests an authorization check after the request has been processed but before the results have been returned. You can use this `direction` identifier to match requests that have been checked `ONCE`, `PRIOR`, or `POST`. See `weblogic.security.spi.Direction` in the *Java API Reference for Oracle WebLogic Server*, which is the object type that is used to pass `ONCE`, `PRIOR`, or `POST` to the security provider. **Note:** Using a Direction object in a decision is optional for Authorization providers. The WebLogic Server XACML Authorization provider supports only the `ONCE` value.

Table A-2 describes the value that you specify for the `action-id` identifier.

Table A-2 Value for the action-id Identifier

If the operation is in this resource type...	Specify...
Admin	The name of an administrative activity that is protected by an Admin resource. For example, `UserLockout`. For a list of valid values, see the action parameter for the `weblogic.security.service.AdminResource` constructor in the *Java API Reference for Oracle WebLogic Server*.
Application	The name of the application as displayed in the .
Control	The name of a method in a Java control. Java controls are reusable components that you can create and use anywhere within a WebLogic Platform application.
EJB	The name of an EJB method. For example, `mymethod`. If the method is overloaded, all methods with the specified method name will be matched.
JDBC	The name of an administrative activity that is protected by a JDBC resource. For a list of valid values, see the `action` parameter for the `weblogic.security.service.JDBCResource` constructor in the *Java API Reference for Oracle WebLogic Server*.
JMS	The name of an administrative activity that is protected by a JMS resource. For a list of valid values, see the `action` parameter for the `weblogic.security.service.JMSResource` constructor in the *Java API Reference for Oracle WebLogic Server*.
JMX	The name of an operation in a WebLogic Server MBean. For example, `shutdown`.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table A-2 (Cont.) Value for the action-id Identifier

If the operation is in this resource type...	Specify...
JNDI	The name of an administrative activity that is protected by a JNDI resource. For a list of valid values, see the `action` parameter for the `weblogic.security.service.JNDIResource` constructor in the *Java API Reference for Oracle WebLogic Server*.
Server	The name of a server life cycle activity that is protected by a Server resource. For example, `boot`. For a list of valid values, see Server Resources.
URL	The name of an HTTP method. For example, `POST`.
Web Service	The name of a Web Service method. For example, `mymethod`.
Work Context	The name of an administrative activity that is protected by a Work Context resource. For a list of valid values, see the `action` parameter for the `weblogic.security.service.WorkContextResource` constructor in the *Java API Reference for Oracle WebLogic Server*.
All others	The following string: `access`

Examples

The following example uses an `Action` element to specify that the target is `mymethod` within the SimpleSoap Web Service.

> **Note**
>
> When specifying values in the `<AttributeValue>` element, you must not include extraneous space characters. In particular:
>
> - Do not precede the first value with more than a single space character.
> - When specifying multiple values, do not separate each value by more than a single space character.
> - Make sure there are no space characters that follow the last value.
>
> For example:

```
<AttributeValue datatype>value1, value2, value3</AttributeValue>
```

```
<Target>
  <Resources>
   <Resource>
   <ResourceMatch
    MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">type=&lt;webservices&gt;,
application=webservicesJwsSimpleEar,contextPath=/jws_basic_simple,
webService=SimpleSoapPort</AttributeValue>
    <ResourceAttributeDesignator
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
     AttributeId="urn:oasis:names:tc:xacml:2.0:resource:resource-ancestor"
     DataType="http://www.w3.org/2001/XMLSchema#string"
     MustBePresent="true"/>
   </ResourceMatch>
  </Resource>
  </Resources>
  <ActionMatch
    MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">mymethod</
AttributeValue>
     <ActionAttributeDesignator
      AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
      DataType="http://www.w3.org/2001/XMLSchema#string"
      MustBePresent="true"/>
  </ActionMatch>
</Target>
```

Updating the JWS File with @Policy and @Policies Annotations

Use the `@Policy` and `@Policies` annotations in your JWS file to specify that the web service has one or more policy files attached to it. You can use these annotations at either the class or method level.

> **Note**
>
> If you specify a transport-level security policy for your web service, it must be at the class level.
>
> In addition, the transport-level security policy must apply to both the inbound and outbound directions. That is, you cannot have HTTPS for inbound and HTTP for outbound.

See Loading a Policy From the CLASSPATH for an additional policy option.

The `@Policies` annotation simply groups two or more `@Policy` annotations together. Use the `@Policies` annotation if you want to attach two or more policy files to the class or method. If you want to attach just one policy file, you can use `@Policy` on its own.

The `@Policy` annotation specifies a single policy file, where it is located, whether the policy applies to the request or response SOAP message (or both), and whether to attach the policy file to the public WSDL of the service.

Examples

The following example uses an `Environment` element to match value of a WebLogic Server listen port. Such an element could create a policy that requires a request to come through listen port 9001:

```
<Environment>
   <EnvironmentMatch
      MatchId="urn:oasis:names:tc:xacml:1.0:function:double-equal">
      <EnvironmentAttributeDesignator
           AttributeId="urn:bea:xacml:2.0:environment:context:com.bea.cont
           extelement.channel.Port"
            DataType="http://www.w3.org/2001/XMLSchema#double"/>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#double">9001</
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
AttributeValue>
   </EnvironmentMatch>
</Environment>
```

Policy and PolicySet Identifiers

A XACML `Policy` contains rules and is identified by its `PolicyId`. A `PolicySet` contains policies or policy sets and is identified by its `PolicySetId`. Both identifiers must be valid URI values. Within a `PolicySet`, use `PolicyIdReference` to include a `Policy` by ID and `PolicySetIdReference` to include a `PolicySet` by ID. WebLogic Server uses the corresponding ID as the key in the Authorization or Role Mapping provider's policy store.

WebLogic Server reserves URI values beginning with `urn:bea:` for its internal use. While you cannot create your own policies with URIs that begin with `urn:bea:`, you can use these values to include Oracle's policies in your policy sets.

Examples

The following example is a valid identifier for a `Policy` element:

```
<Policy
    PolicyId="urn:mycompany:myapplication:policyid:1"
...>
```

The following example is a valid reference to the `Policy` element above:

```
<PolicyIdReference>
    urn:mycompany:myapplication:policyid:1
</PolicyIdReference>
```

Resource Identifiers

XACML uses a `Resource` element to represent data, a service, or a system component.

WebLogic Server supports all of the XACML `Resource` identifiers, as described in the XACML 2.0 Core Specification.

To identify a WebLogic Server resource, use resource identifiers as described in Table A-3. For information about WebLogic Server resources, see Resource Types You Can Secure with Policies.

Table A-3 WebLogic Server Resource Identifiers

To identify a...	Use the following identifier...
Resource	• Attribute ID: `urn:oasis:names:tc:xacml:1.0:resource:resource-id` • Data Type: `string` • Value: A collection of name and value pairs that specify the WebLogic Server resource type and the location of the resource in the WebLogic Server resource hierarchy. WebLogic Server generates these identifiers. Note that a policy that uses this identifier will not protect resources that are below the specified resource. For example, if you use this identifier for a policy on a Web Service module, the policy will not protect methods within the Web Service. See Get the ID of the Resource to Secure .

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table A-3 (Cont.) WebLogic Server Resource Identifiers

To identify a...	Use the following identifier...
Resource and its ancestors	• Attribute ID: `urn:oasis:names:tc:xacml:2.0:resource:resource-ancestor-or-self` • Data Type: `string` • Values: A collection of name and value pairs that specify a WebLogic Server resource type. WebLogic Server generates these identifiers. See Get the ID of the Resource to Secure .
Parent of a resource	• Attribute ID: `urn:oasis:names:tc:xacml:1.0:resource:resource-parent` • Data Type: `string` • Values: A collection of name and value pairs that specify a WebLogic Server resource type. WebLogic Server generates these identifiers. See Get the ID of the Resource to Secure .
Ancestor of a resource	• Attribute ID: `urn:oasis:names:tc:xacml:2.0:resource:resource-ancestor` • Data Type: `string` • Values: A collection of name and value pairs that specify a WebLogic Server resource type. WebLogic Server generates these identifiers. See Get the ID of the Resource to Secure .

Example of Adding Security to a JAX-WS Web Service

This section provides a simple example of adding security to a JAX-WS web service. The example attaches four policies:

- `Wssp1.2-2007-SignBody.xml`
- `Wssp1.2-2007-EncryptBody.xml`
- `Wssp1.2-2007-Wss1.1-UsernameToken-Plain-EncryptedKey-Basic128.xml`
- `Wssp1.2-Wss1.0-UsernameToken-Plain-X509-Basic256.xml`

The examples include extensive inline comments in the code.

shows the web service code.

> **Note**
>
> This web service implements *attachToWsdl=false*, and therefore the web service client needs to load a client-side version of the policy, as shown in .

Example A-2 Web Service SignEncrypt.java

```
package signencrypt;

import java.io.File;

import weblogic.jws.Policies;
import weblogic.jws.Policy;
import weblogic.jws.security.WssConfiguration;
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
import jakarta.activation.DataHandler;
import jakarta.activation.FileDataSource;
import jakarta.jws.WebMethod;
import jakarta.jws.WebService;
import jakarta.xml.ws.BindingType;
import jakarta.xml.ws.soap.MTOM;

import com.sun.xml.ws.developer.SchemaValidation;

/**
 *
 * Webservice which accepts a SOAP Message which is Signed And
 *         Encrypted Uses the WS-Policy 1.2
 */


@WebService(name = "SignEncrypt", portName = "SignEncryptPort", serviceName =
"SignEncrypt", targetNamespace = "http://signencrypt")
@BindingType(value = "http://schemas.xmlsoap.org/wsdl/soap/http")
// Domain Level WebserviceSecurity Configuration
@WssConfiguration(value = "Basic-UNT")
@MTOM()
//@SchemaValidation

public class SignEncrypt {

  @Policies( {
  @Policy(uri = "policy:Wssp1.2-2007-SignBody.xml", attachToWsdl=false),
      @Policy(uri = "policy:Wssp1.2-2007-EncryptBody.xml", attachToWsdl=false),
    /*
     * WSS 1.1 X509 with symmetric binding and authentication with plain-text
     * Username Token which is encrypted and signed using the Symmetric key
     */
    /* Use Basic-UNT WssConfiguration */
      @Policy(uri = "policy:Wssp1.2-2007-Wssp1.2-2007-Wss1.1-UsernameToken-Plain-
EncryptedKey-Basic128.xml",
attachToWsdl=false)
  /*
   * The client side public certificate and private key is not required in
   * this scenario. Username token with plain text password is sent in the
   * request for authentication, and signed and encrypted with the symmetric
   * key. The symmetric key is encrypted by the server's public key. The client
   * also signs and encrypts the request header elements and body with the
   * symmetric key. The server signs and encrypts the response body with the
   * symmetric key. Both request and response messages include the signed time
   * stamps. The encryption method is Basic128.
   */
  /* Use untx509webservicesecurity WssConfiguration */

  /*
   * @Policy(uri =
   * "policy:Wssp1.2-Wss1.0-UsernameToken-Plain-X509-Basic256.xml")
   */})
  @WebMethod()
  public String echoString(String input) {
    String result = "[SignEncrypt.echoString]: " + input;
    System.out.println(result);
    return result;
  }

   @WebMethod()
  public String echoStringWithoutSecurity(String input) {
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
    String result = "[SignEncrypt.echoString]: " + input;
    System.out.println(result);
    return result;
  }

  @WebMethod()
  public byte[] echoStringAsByteArray(String data) {
    System.out.println("echoByteArray data: " + data);
    byte[] output = data.getBytes();
    System.out.println("Output Length : " + output.length + " Output: " +
output.toString());
    return data.getBytes();
  }

   @Policies( {
  @Policy(uri = "policy:Wssp1.2-2007-SignBody.xml", attachToWsdl=false),
      @Policy(uri = "policy:Wssp1.2-2007-EncryptBody.xml", attachToWsdl=false),
    /*
     * WSS 1.1 X509 with symmetric binding and authentication with plain-text
     * Username Token which is encrypted and signed using the Symmetric key
     */
    /* Use Basic-UNT WssConfiguration */
      @Policy(uri = "policy:Wssp1.2-2007-Wssp1.2-2007-Wss1.1-UsernameToken-Plain-
EncryptedKey-Basic128.xml",
attachToWsdl=false)
  /*
   * The client side public certificate and private key is not required in
   * this scenario. Username token with plain text password is sent in the
   * request for authentication, and signed and encrypted with the symmetric
   * key. The symmetric key is encrypted by the server's public key. The client
   * also signs and encrypts the request header elements and body with the
   * symmetric key. The server signs and encrypts the response body with the
   * symmetric key. Both request and response messages include the signed time
   * stamps. The encryption method is Basic128.
   */
  /* Use untx509webservicesecurity WssConfiguration */

  /*
  * @Policy(uri =
  * "policy:Wssp1.2-Wss1.0-UsernameToken-Plain-X509-Basic256.xml")
  */})
 @WebMethod()
 public byte[] echoByteArrayWithSecurity(byte[] inputData) {
   System.out.println("echoByteArrayWithSecurity data: " + inputData.length + " bytes");
   return inputData;
 }

 @WebMethod()
 public byte[] echoByteArray(byte[] inputData) {
   System.out.println("echoByteArray data: " + inputData);
   return inputData;
 }

 @WebMethod()
 public DataHandler getDataHandler(String fileName) {

   DataHandler handler = null;
   try {
   File file = new File(fileName);
   System.out.println("file: " + file.getCanonicalPath() + ", " +  file.getPath());

   FileDataSource fileDataSource = new FileDataSource(file);
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
   handler = new DataHandler(fileDataSource);

   } catch(Exception e) {
     System.out.println("Error Creating Data Handelr:  " + e.getMessage());
   }

   return handler;

  }


  @WebMethod()
 @Policies( {
   @Policy(uri = "policy:Wssp1.2-2007-SignBody.xml", attachToWsdl=false),
   @Policy(uri = "policy:Wssp1.2-2007-EncryptBody.xml", attachToWsdl=false),
   /*
    * WSS 1.1 X509 with symmetric binding and authentication with plain-text
    * Username Token which is encrypted and signed using the Symmetric key
    */
   /* Use Basic-UNT WssConfiguration */
       @Policy(uri = "policy:Wssp1.2-2007-Wssp1.2-2007-Wss1.1-UsernameToken-Plain-
EncryptedKey-Basic128.xml", attachToWsdl=false)
  /*
   * The client side public certificate and private key is not required in
   * this scenario. Username token with plain text password is sent in the
   * request for authentication, and signed and encrypted with the symmetric
   * key. The symmetric key is encrypted by the server's public key. The client
   * also signs and encrypts the request header elements and body with the
   * symmetric key. The server signs and encrypts the response body with the
   * symmetric key. Both request and response messages include the signed time
   * stamps. The encryption method is Basic128.
   */
 /* Use untx509webservicesecurity WssConfiguration */
  /*
  * @Policy(uri =
  * "policy:Wssp1.2-Wss1.0-UsernameToken-Plain-X509-Basic256.xml")
  */})
 public DataHandler getDataHandlerWithSecurity(String fileName) {

   DataHandler handler = null;
   try {
   File file = new File(fileName);
   System.out.println("file: " + file.getCanonicalPath() + ", " +  file.getPath());

   FileDataSource fileDataSource = new FileDataSource(file);
   handler = new DataHandler(fileDataSource);

   } catch(Exception e) {
     System.out.println("Error Creating Data Handelr:  " + e.getMessage());
   }

   return handler;

 }

}
```

As noted, the web service implements *attachToWsdl=false*, and therefore the web service client needs to load a client-side version of the policy. shows an example of using the *weblogic.jws.jaxws.ClientPolicyFeature* class to load client-side policies.

The example includes extensive inline comments.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Example A-3 SOAClient.java

```
package signencrypt.client;
import weblogic.jws.jaxws.ClientPolicyFeature;
import weblogic.jws.jaxws.policy.InputStreamPolicySource;
import weblogic.security.SSL.TrustManager;
import weblogic.wsee.policy.runtime.BuiltinPolicyFinder;
import weblogic.wsee.security.bst.ClientBSTCredentialProvider;
import weblogic.wsee.security.bst.StubPropertyBSTCredProv;
import weblogic.wsee.security.unt.ClientUNTCredentialProvider;
import weblogic.wsee.security.util.CertUtils;
import weblogic.xml.crypto.wss.WSSecurityContext;
import weblogic.xml.crypto.wss.provider.CredentialProvider;

import soa.client.Bpelprocess1ClientEp;
import soa.client.BPELProcess1;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.URL;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import jakarta.activation.DataHandler;
import jakarta.activation.FileDataSource;
import javax.xml.namespace.QName;
import jakarta.xml.ws.BindingProvider;
import jakarta.xml.ws.WebServiceFeature;
import jakarta.xml.ws.soap.MTOMFeature;

public class SOAClient {

  private final static boolean debug = true;

  private final static String endpointURL =
    "http://....com:8001/soa-infra/services/default/soa/bpelprocess1_client_ep";
  private final static String certsDir = "C:/webservices/server/keystores";

  private final static String serverKeyStoreName = "default-keystore.jks";
  private final static String serverKeyStorePass = "...";
  private final static String serverCertAlias = "alice";
  private final static String serverKeyPass = "...";

  private final static String username = "weblogic";
  private final static String password = "...";

  private final static String fileName =
    "C:/webservices/farallon/owsm-interop/mtom.JPG";

  private final static String outputFileName =
    "C:/webservices/farallon/owsm-interop/output.jpg";

  private final static String[] clientPolicyFileNames =
    {
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
      "./policy/Wssp1.2-2007-Wssp1.2-2007-Wss1.1-UsernameToken-Plain-EncryptedKey-
Basic128.xml",
      "./policy/Wssp1.2-2007-SignBody.xml",
      "./policy/Wssp1.2-2007-EncryptBody.xml" };

  private BPELProcess1 port = null;

  /**
   * Create the Stub/Port and set the Stub/Port with Client Side Security Policy
   * Feature and MTOM Feature.
   * @throws Exception
   */

  private void createStubWithClientPolicy() throws Exception {

    URL url = new URL(endpointURL + "?WSDL");

    QName serviceName =
      new QName("http://xmlns.oracle.com/SOASecurity/soa/BPELProcess1",
        "bpelprocess1_client_ep");

    Bpelprocess1ClientEp service = new Bpelprocess1ClientEp(url, serviceName);

    QName operationName =
      new QName("http://xmlns.oracle.com/SOASecurity/soa/BPELProcess1", "process");

    ClientPolicyFeature policyFeature = new ClientPolicyFeature();

    // Set the Client Side Policy on the operation with QName <operationName>

policyFeature.setEffectivePolicyForOperation(operationName, new
InputStreamPolicySource(getPolicyInputStreamArray(clientPolicyFileNames)
));
    MTOMFeature mtomFeature = new MTOMFeature();

    WebServiceFeature[] features = { policyFeature, mtomFeature };
    // WebServiceFeature[] features = { mtomFeature };
    //WebServiceFeature[] features = {policyFeature};

    port = service.getBPELProcess1Pt(features);
  }

  /**
  * Setup the Client Port/Stub used to invoke the webservice with Security
  *
  * @throws Exception
  */
 private void setUp() throws Exception {
   createStubWithClientPolicy();
   /**
    * Get the Server Public Certificate to Encrypt the Symmetric Key or the
    * SOAP Message
    */
   /**
    * Get the Server Public Certificate to Verify the Signature of the
    * Symmetric Key or the SOAP Message
    */
    X509Certificate serverCert =
     (X509Certificate) CertUtils.getCertificate(
       certsDir + "/" + serverKeyStoreName, serverKeyStorePass,
       serverCertAlias, "JKS").get(0);
   List<CredentialProvider> credProviders =
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
    new ArrayList<CredentialProvider>();
  /*
   * Set up UserNameToken
   */
  credProviders.add(new ClientUNTCredentialProvider(username.getBytes(),
    password.getBytes()));
  Map<String, Object> rc = ((BindingProvider) port).getRequestContext();
   /*
   * For Wssp1.2-2007-Wssp1.2-2007-Wss1.1-UsernameToken-Plain-EncryptedKey-Basic128.xml
   * there is no need to specify the client side public certificate and
   * private key as this is a symmetric key use case. serverCert is used to
   * encrypt the Symmetric Key/Keys
   */
  rc.put(StubPropertyBSTCredProv.SERVER_ENCRYPT_CERT, serverCert);
  rc.put(WSSecurityContext.CREDENTIAL_PROVIDER_LIST, credProviders);
  rc.put(WSSecurityContext.TRUST_MANAGER, new TrustManager() {
    public boolean certificateCallback(X509Certificate[] chain,
      int validateErr) {
      // need to validate if the server cert can be trusted
      System.out.println("Validating Server Certificate");
      return true;
    }
  });

 }
/**
 * Returns an array of InputStreams of the policy files
 *
 * @param policyNames
 * @return array of InputStreams of Policy's
 * @throws FileNotFoundException
 */
private InputStream[] getPolicyInputStreamArray(String[] policyNames)
  throws FileNotFoundException {
  InputStream[] inpStreams = new InputStream[policyNames.length];
  for (int k = 0; k < policyNames.length; k++) {
    System.out.println("policy name: " + policyNames[k]);
    inpStreams[k] = getPolicyInputStream(policyNames[k]);
  }
  return inpStreams;
}
 /**
 * Returns an InputStream of the policy file
 *
 * @param myPolicyName
 * @return
 * @throws FileNotFoundException
 */
private InputStream getPolicyInputStream(String myPolicyName)
  throws FileNotFoundException {
  return new FileInputStream(myPolicyName);
}
/**
 * Invoke the webservice at endpointURL
 *
(http://....:9003/soa-infra/services/default/soa/bpelprocess1_client_ep)
 *
 * @throws Exception
 */
private void invokeProcess() throws Exception {
  InputStream inputstream = null;
  OutputStream outputstream = null;
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
    try {

      File file = new File(fileName);
      File outputFile = new File(outputFileName);

      inputstream = new BufferedInputStream(new FileInputStream(file));
      int bytesAvailable = -1;
      int counter = 0;
      int bytesRead = 0;
      int fileSize = (int) file.length();

      byte[] fileInBytes = new byte[fileSize];

      bytesRead = inputstream.read(fileInBytes);
      System.out.println("bytesRead: " + bytesRead + ", fileSize: " + fileSize + "
fileInBytes: " + fileInBytes.length);

      byte[] result = port.process(fileInBytes);
      /*byte[] input = "Hello".getBytes();
      System.out.println("input length : "+ input.length);

      byte[] result = port.process(input);*/
      if (!outputFile.exists()) {
        outputFile.createNewFile();
      }

      outputstream = new BufferedOutputStream(new FileOutputStream(outputFile));

      if (result != null) {
        System.out.println("Result Length: " + result.length);
      } else {
        System.out.println("result is null");
      }
      outputstream.write(result);

      // System.out.println(result);
    } catch (Exception e) {
      System.out.println("Error Creating Data Handler:  " + e.getMessage());
    } finally {

      if (inputstream != null) {
        inputstream.close();
      }

     if (outputstream != null) {
        outputstream.close();
      }
    }
  }
  public static void main(String[] args) {
    try {
      SOAClient client = new SOAClient();
      client.setUp();
      //client.createStubWithClientPolicy();
      client.invokeProcess();
    } catch (Exception e) {
      System.out.println("Error calling SOA Webservice: " + e.getMessage());
      if (debug) {
        e.printStackTrace();
      }
    }
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
 }
}
```

Subject Identifiers

XACML uses a `Subject` element to represent an actor whose attributes may be referenced by a predicate.

WebLogic Server supports all of the XACML Subject identifiers, as described in the XACML 2.0 Core Specification.

To identify a WebLogic Server user, group, or role as defined in a WebLogic Server realm, use subject identifiers as described Table A-4.

Table A-4 WebLogic Subject Identifiers

To identify a...	Use the following identifier...
User principal	• Attribute ID: `urn:oasis:names:tc:xacml:1.0:subject:subject-id` • Data Type: `string` • Value: Name of a WebLogic Server user principal.
Group principal	• Attribute ID: `urn:oasis:names:tc:xacml:2.0:subject:group` • Data Type: `string` • Values: Name of a WebLogic Server group principal.
Role	• Attribute ID: `urn:oasis:names:tc:xacml:2.0:subject:role` • Data Type: `string` • Values: Names of one or more roles as resolved by a XACML Role Mapping provider. **Note:** WebLogic Server supports only a subset of the RBAC Profile specification, which is the specification that defines this attribute.
Subject who has signed a SOAP document	• Attribute ID: `urn:bea:xacml:2.0:subject:context:com.bea.contextelement.webservice.Integrity{{`*`namespace`*`}`*`element`*`}` where *`namespace`* and *`element`* are the namespace and element that was signed. For example: `urn:bea:xacml:2.0:subject:context:com.bea.contextelement.webservice.Integrity{{ http://schemas.xmlsoap.org/soap/envelope/}Body}` • Data Type: `string` • Values: The Java subject who has signed the element (either user ID or group ID).

Examples

For an example of a XACML document that uses identifiers from Table A-4 to define a security role that can be used to protect access to a Web Service, see Example 7-2.

WebLogic Server Functions for XACML

The following sections describe the functions that the WebLogic Server XACML providers support in addition to the functions described in the XACML Core Specification:

- #unique_132

Copyright © 2007, 2026, Oracle and/or its affiliates.

- #unique_133
- Time/Date Conversions
- Arithmetic Conversions and Functions
- Object Type Conversions
- Object Comparisons
- String Comparisons and Manipulations

Configuring STS URI for SAML: Standalone Client

When the STS is used for retrieving the SAML token, the STS is not co-located with the service and there is no default STS URI. You must configure the STS URI in this case.

Save the following complete class as `SamlEndpoint.java`. Before invoking an operation, call `SamlEndpoint.configure((BindingProvider) port, stsEndpoint)`, where `port` is the generated JAX-WS port and `stsEndpoint` is the absolute HTTPS URI of the SAML STS. Compile with the WebLogic client libraries. Configure credentials and HTTPS trust separately. The method changes the request context; it does not request a token or invoke the service.

```
import java.net.URI;
import java.util.Map;
import jakarta.xml.ws.BindingProvider;
import weblogic.wsee.jaxrpc.WLStub;

public final class SamlEndpoint {
    private SamlEndpoint() {}

    // Call before invoking an operation on the generated JAX-WS port.
    public static void configure(BindingProvider port, URI stsEndpoint) {
        if (!"https".equalsIgnoreCase(stsEndpoint.getScheme())
                || stsEndpoint.getHost() == null) {
            throw new IllegalArgumentException("Provide an absolute HTTPS STS URI");
        }
        Map<String, Object> context = port.getRequestContext();
        context.put(WLStub.WST_STS_ENDPOINT_ON_SAML, stsEndpoint.toASCIIString());
    }
}
```

Examples

The following example is a `Condition` that uses `urn:bea:xacml:2.0:function:character-equal` to compare two `bea:characters`:

```
<Condition>
   <Apply FunctionId="urn:bea:xacml:2.0:function:character-equal">
     <AttributeValue DataType="urn:bea:xacml:2.0:data-type:character">Q</AttributeValue>
     <AttributeValue DataType="urn:bea:xacml:2.0:data-type:character">Q</AttributeValue>
   </Apply>
</Condition>
```

Secure Conversation Policies

The following predefined policies use the SHA-1 algorithm. Change them as described in this section to instead use SHA-256.

- `Wssp1.2-2007-Wssc1.3-Bootstrap-Https.xml`

Copyright © 2007, 2026, Oracle and/or its affiliates.

- Wssp1.2-2007-Wssc1.3-Bootstrap-Https-BasicAuth.xml
- Wssp1.2-2007-Wssc1.3-Bootstrap-Https-ClientCertReq.xml
- Wssp1.2-2007-Wssc1.3-Bootstrap-Https-UNT.xml
- Wssp1.2-2007-Wssc1.3-Bootstrap-Wss1.0.xml
- Wssp1.2-2007-Wssc1.3-Bootstrap-Wss1.1.xml
- Wssp1.2-2007-Wssc1.4-Bootstrap-Wss1.0-UsernameToken-Plain-X509-Basic256.xml
- Wssp1.2-2007-Wssc1.4-Bootstrap-Wss1.1-Saml2.0-Bearer.xml
- Wssp1.2-2007-Wssc1.4-Bootstrap-Wss1.1-UsernameToken-Plain-EncryptedKey.xml
- Wssp1.2-Wssc200502-Bootstrap-Wss1.1.xml

Example

The following policy uses the `instance-method` function to invoke the `HttpServletRequest.getAuthType()` method on requests that match a specific URL pattern. See `jakarta.servlet.http.HttpServletRequest` in the Jakarta Servlet API Specification . The WebLogic Server `ContextHandler` makes this `HttpServletRequest` object available to the Authorization and Role Mapping providers for all requests that come through the servlet container. Any policy for a URL resource can invoke this or other `HttpServletRequest` methods.

Example A-4 Policy That Invokes HttpServletRequest.getAuthType()

```
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
   PolicyId="urn:sample:xacml:2.0:function:instance-method"
   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:
     first-applicable">
  <Description>function:instance-method</Description>
  <Target>
    <Resources>
      <Resource>
         <ResourceMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">type=&lt;url&gt;, application=MedRecEAR, contextPath=,uri=/docs/*</
AttributeValue>
             <ResourceAttributeDesignator
               AttributeId="urn:oasis:names:tc:xacml:2.0:resource:
                  resource-ancestor-or-self"
               DataType="http://www.w3.org/2001/XMLSchema#string"
                MustBePresent="true"/>
         </ResourceMatch>
      </Resource>
    </Resources>
  </Target>
  <!-- Declaring the instance-method function as a variable because this policy
       invokes it multiple times.
  -->
  <VariableDefinition VariableId="authType">
    <Apply FunctionId="urn:bea:xacml:2.0:function:instance-method">
      <!-- Passing the HttpServletRequest object to the function, which the
           BEA ContextHandler makes available to the security framework.
      -->
      <Apply FunctionId="urn:bea:xacml:2.0:function:object-one-and-only">
        <EnvironmentAttributeDesignator
           DataType="urn:bea:xacml:2.0:data-type:object"
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
            AttributeId="urn:bea:xacml:2.0:environment:context:com.bea.
               contextelement.servlet.HttpServletRequest" />
      </Apply>
      <!--  Passing "getAuthType()" as the name of the HttpServletRequest
             method to invoke
      -->
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">getAuthType</
AttributeValue>
      <!-- Because the getAuthType() method signature contains no parameters,
           pass an empty bag of Class.
      -->
      <Apply FunctionId="urn:bea:xacml:2.0:function:class-bag" />
    </Apply>
  </VariableDefinition>
  <!-- Creating a rule that allows access to the resource only if
       the getAuthType() returns a non-null value and if the non-null
       value is "CLIENT_CERT"
  -->
  <Rule RuleId="primary-rule" Effect="Permit">
    <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
          <Apply FunctionId="urn:bea:xacml:2.0:function:object-is-null">
            <VariableReference VariableId="authType" />
          </Apply>
        </Apply>
        <Apply
           FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
           <!-- Because the instance-method function returns a bea:Object,
                this policy wraps the function in an object-to-string function,
                which enables a comparison of the function output with another
                string.
           -->
           <Apply FunctionId="urn:bea:xacml:2.0:function:object-to-string">
              <VariableReference VariableId="authType" />
           </Apply>
           <!--  Declaring a String object to compare to the
                 HttpServletRequest.getAuthType() return value.
           -->
           <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">CLIENT_CERT</AttributeValue>
         </Apply>
       </Apply>
    </Condition>
  </Rule>
  <Rule RuleId="deny-rule" Effect="Deny" />
</Policy>
```

Time/Date Conversions

Table A-5 lists the functions that Oracle provides to convert XACML times and dates to different data types.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table A-5 WebLogic Server Time/Date Conversions

Function	Description
`dateTime-dayOfMonth`	• URI: `urn:bea:xacml:2.0:function:dateTime-dayOfMonth` • Input Type: `dateTime` • Return Type: `integer` • Description: This function takes a single argument of type `dateTime` and returns an integer value that represents the day of month implied by the `dateTime` input. The first day of the month is represented with a value of 1.
`dateTime-dayOfMonthMaximum`	• URI: `urn:bea:xacml:2.0:function:dateTime-dayOfMonthMaximum` • Input Type: `dateTime` • Return Type: `integer` • Description: This function takes a single argument of type `dateTime` and returns an integer value that represents the value of the last day of the month. For example, if the `dateTime` describes a time in the month of December, the function returns 31 (because December has a maximum of 31 days). This function adjusts the value returned for February on leap years.
`dateTime-dayOfWeek`	• URI: `urn:bea:xacml:2.0:function:dateTime-dayOfWeek` • Input Type: `dateTime` • Return Type: `integer` • Description: This function takes a single argument of type `dateTime` and returns an integer value that indicates the day of the week implied by the `dateTime` input. Sunday is the first day of the week and has a value of 1.
`dateTime-secondsOfDay`	• URI: `urn:bea:xacml:2.0:function:dateTime-secondsOfDay` • Input Type: `dateTime` • Return Type: `integer` • Description: This function takes a single argument of type `dateTime` and returns an integer value that indicates the number of whole seconds that have elapsed in the day that is indicated by the `dateTime` input.
`dayTimeDuration-timeZoneOffset`	• URI: `urn:bea:xacml:2.0:function:dayTimeDuration-timeZoneOffset` • Input Type: `null` • Return Type: `dayTimeDuration` • Description: This function takes no arguments and returns a `dayTimeDuration` value that indicates the time-zone offset between the local time and GMT time.
`string-to-dateTime`	• URI: `urn:bea:xacml:2.0:function:string-to-dateTime` • Input Type: `string` • Return Type: `dateTime` • Description: This function takes a single argument of type `string` and returns the argument as a `dateTime`.
`string-to-date`	• URI: `urn:bea:xacml:2.0:function:string-to-date` • Input Type: `string` • Return Type: `date` • Description: This function takes a single argument of type `string` and returns the argument as a `date`.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table A-5 (Cont.) WebLogic Server Time/Date Conversions

Function	Description
`object-to-dateTime`	• URI: `urn:bea:xacml:2.0:function:object-to-dateTime` • Input Type: `bea:Object` • Return Type: `dateTime` • Description: This function takes a single argument of type `bea:Object` and returns the value as a `dateTime` object. If the `bea:Object` is an instance of `java.util.Calendar`, then its value is returned directly. If the `bea:Object` is an instance of `java.sql.Timestamp` or `java.util.Date`, this function converts the `bea:Object` to `java.util.Calendar` and returns the `Calendar` value. Otherwise, this function converts the `bea:Object` to `java.lang.String` and parses the `String` into a `dateTime` value.
`object-to-date`	• URI: `urn:bea:xacml:2.0:function:object-to-date` • Input Type: `bea:Object` • Return Type: `date` • Description: This function takes a single argument of type `bea:Object` and returns the value as a `date` object. If the `bea:Object` is an instance of `java.util.Calendar`, then its value is returned directly. If the `bea:Object` is an instance of `java.util.Date`, this function converts the `bea:Object` to `java.util.Calendar` and returns the `Calendar` value. Otherwise, this function converts the `bea:Object` to `java.lang.String` and parses the `String` into a `date` value.

Arithmetic Conversions and Functions

Table A-6 lists the functions that Oracle provides to convert arithmetic values to different Input Types and to extend the basic set of arithmetic functions specified by XACML.

Table A-6 WebLogic Server Arithmetic Conversions and Functions

Function	Description
`float-to-double`	• URI: `urn:bea:xacml:2.0:function:float-to-double` • Input Type: `float` • Return Type: `double` • Description: This function takes a single argument of type `float` and returns the argument as a `double`.
`long-to-double`	• URI: `urn:bea:xacml:2.0:function:long-to-double` • Input Type: `long` • Return Type: `double` • Description: This function takes a single argument of type `long` and returns the argument as a `double`.
`long-to-float`	• URI: `urn:bea:xacml:2.0:function:long-to-float` • Input Type: `long` • Return Type: `float` • Description: This function takes a single argument of type `long` and returns the argument as a `float`.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table A-6 (Cont.) WebLogic Server Arithmetic Conversions and Functions

Function	Description
`integer-to-float`	• URI: `urn:bea:xacml:2.0:function:integer-to-float` • Input Type: `integer` • Return Type: `float` • Description: This function takes a single argument of type `integer` and returns the argument as a `float`.
`integer-to-long`	• URI: `urn:bea:xacml:2.0:function:integer-to-long` • Input Type: `integer` • Return Type: `long` • Description: This function takes a single argument of type `integer` and returns the argument as a `long`.
`string-to-double`	• URI: `urn:bea:xacml:2.0:function:string-to-double` • Input Type: `string` • Return Type: `double` • Description: This function takes a single argument of type `string` and returns the argument as a `double`.
`string-to-long`	• URI: `urn:bea:xacml:2.0:function:string-to-long` • Input Type: `string` • Return Type: `long` • Description: This function takes a single argument of type `string` and returns the argument as a `long`.
`string-to-integer`	• URI: `urn:bea:xacml:2.0:function:string-to-integer` • Input Type: `string` • Return Type: `integer` • Description: This function takes a single argument of type `string` and returns the argument as a `integer`.
`string-to-float`	• URI: `urn:bea:xacml:2.0:function:string-to-float` • Input Type: `string` • Return Type: `float` • Description: This function takes a single argument of type `string` and returns the argument as a `float`.
`to-degrees`	• URI: `urn:bea:xacml:2.0:function:to-degrees` • Input Type: `double` • Return Type: `double` • Description: This function takes a single argument of type `double`, converts the value from radians to degrees, and returns the degrees value as a `double`.
`to-radians`	• URI: `urn:bea:xacml:2.0:function:to-radians` • Input Type: `double` • Return Type: `double` • Description: This function takes a single argument of type `double`, converts the value from degrees to radians, and returns the radians value as a `double`.
`acos`	• URI: `urn:bea:xacml:2.0:function:acos` • Input Type: `double` • Return Type: `double` • Description: This function takes a single argument of type `double` and returns the arc cosine of the argument as a `double`.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table A-6 (Cont.) WebLogic Server Arithmetic Conversions and Functions

Function	Description
`asin`	• URI: `urn:bea:xacml:2.0:function:asin` • Input Type: `double` • Return Type: `double` • Description: This function takes a single argument of type `double` and returns the arc sine of the argument as a `double`.
`atan`	• URI: `urn:bea:xacml:2.0:function:atan` • Input Type: `double` • Return Type: `double` • Description: This function takes a single argument of type `double` and returns the arc tangent of the argument as a `double`.
`atan2`	• URI: `urn:bea:xacml:2.0:function:atan2` • Input Type: `double` • Return Type: `double` • Description: This function takes two arguments of type double that represent the x and y coordinates of a point. It returns a double value which is the theta component of the point in polar coordinates for the r value that corresponds to the x coordinate.
`ceil`	• URI: `urn:bea:xacml:2.0:function:ceil` • Input Type: `double` • Return Type: `double` • Description: This function takes a single argument of type `double` and returns a `double` value that is the smallest mathematical integer that is greater than or equal to the argument value.
`cos`	• URI: `urn:bea:xacml:2.0:function:cos` • Input Type: `double` • Return Type: `double` • Description: This function takes a single argument of type `double` and returns the cosine of the argument as a `double`.
`exp`	• URI: `urn:bea:xacml:2.0:function:exp` • Input Type: `double` • Return Type: `double` • Description: This function takes a single argument of type `double` and returns a `double` value that is Euler's number, `e`, raised to the power of the argument value.
`ieee-remainder`	• URI: `urn:bea:xacml:2.0:function:ieee-remainder` • Input Type: `double` • Return Type: `double` • Description: This function takes two arguments of type `double` and returns a `double` value that is the remainder operation result of the two arguments as described in the IEEE 754 standard. See `https://ieeexplore.ieee.org/document/5976968`.
`log`	• URI: `urn:bea:xacml:2.0:function:log` • Input Type: `double` • Return Type: `double` • Description: This function takes a single argument of type `double` and the natural logarithm as a `double`.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table A-6 (Cont.) WebLogic Server Arithmetic Conversions and Functions

Function	Description
`maximum`	• URI: `urn:bea:xacml:2.0:function:maximum` • Input Type: `double` • Return Type: `double` • Description: This function takes two arguments of type `double` and returns the larger of the two values as a `double`.
`minimum`	• URI: `urn:bea:xacml:2.0:function:minimum` • Input Type: `double` • Return Type: `double` • Description: This function takes two arguments of type `double` and returns the smaller of the two values as a `double`.
`pow`	• URI: `urn:bea:xacml:2.0:function:pow` • Input Type: `double` • Return Type: `double` • Description: This function takes two arguments of type `double` and returns a `double` value that is the result of taking the first argument value to the power of the second argument value.
`random-number`	• URI: `urn:bea:xacml:2.0:function:random-number` • Input Type: `double` • Return Type: `double` • Description: This function takes two arguments of type `double` and returns a `double` value that is a random number greater than or equal to the first argument and less than the second argument.
`rint`	• URI: `urn:bea:xacml:2.0:function:rint` • Input Type: `double` • Return Type: `double` • Description: This function takes a single argument of type `double` and returns a `double` value that is the mathematical integer that is closest to the argument value. If the higher and lower integer values are equally close, then the even value is returned.
`sqrt`	• URI: `urn:bea:xacml:2.0:function:sqrt` • Input Type: `double` • Return Type: `double` • Description: This function takes a single argument of type `double` and returns the square root as a `double`.
`tan`	• URI: `urn:bea:xacml:2.0:function:tan` • Input Type: `double` • Return Type: `double` • Description: This function takes a single argument of type `double` and returns the tangent as a `double`.

Object Type Conversions

WebLogic Server provides a collection of functions for converting XACML data into Java objects. The URI for each function in this collection is as follows:

`urn:bea:xacml:2.0:function:`*`type`*`-to-object`

Copyright © 2007, 2026, Oracle and/or its affiliates.

where *type* is the name of a XACML data type. Table A-7 lists all data types and the Java object that the corresponding function returns.

For example, this function returns test as a `java.lang.String` object:

```
<Apply FunctionId="urn:bea:xacml:2.0:function:string-to-object">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">test</
AttributeValue>
</Apply>
```

Table A-7 Data to Java Object Conversion

When type equals...	The urn:bea:xacml:2.0:function:type-to-object function returns...
character	`java.lang.Character`
string	`java.lang.String`
boolean	`java.lang.Boolean`
integer	`java.lang.Integer`
double	`java.lang.Double`
float	`java.lang.Float`
long	`java.lang.Long`
decimal	`java.lang.Double`
base64Binary	`java.lang.Byte[]`
hexBinary	`java.lang.Byte[]`
date	`java.util.Calendar`
time	`java.util.Calendar`
dateTime	`java.util.Calendar`
dayTimeDuration	`java.lang.Long`
yearMonthDuration	`java.lang.Integer`
rfc822Name	`java.lang.String`
x500Name	`java.lang.String`
anyURI	`java.net.URI`
ipAddress	`java.lang.String`
dnsAddress	`java.lang.String`

Table A-8 lists the functions that Oracle provides to convert strings or Java objects to different data or object types. To pass objects that the container makes available to the current context, use the `urn:bea:xacml:2.0:environment:context:`*key* environment identifier to specify the `bea:Object`. See #unique_143.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table A-8 WebLogic Server Object Conversions

Function	Description
`string-to-class`	• URI: `urn:bea:xacml:2.0:function:string-to-class` • Input Type: `string` • Return Type: `bea:Class` • Description: This function takes a single argument of type `string`, invokes `java.lang.Class.forName()` on the argument value, and returns the result as a `bea:Class`.
`object-to-string`	• URI: `urn:bea:xacml:2.0:function:object-to-string` • Input Type: `bea:Object` • Return Type: `string` • Description: This function takes a single argument of type `bea:Object`, invokes `java.lang.Object.toString()` on the argument value, and returns the result as a `string`.
`object-to-double`	• URI: `urn:bea:xacml:2.0:function:object-to-double` • Input Type: `bea:Object` • Return Type: `double` • Description: This function takes a single argument of type `bea:Object` and returns the value as a `double`. If the `bea:Object` is an instance of `double`, then its value is used directly. If the `bea:Object` is an instance of `java.lang.Number`, then this function invokes `Number.doubleValue()` on the value. Otherwise, this function converts the `bea:Object` to a `java.lang.String` and parses the `String` into a `double`.
`object-to-integer`	• URI: `urn:bea:xacml:2.0:function:object-to-integer` • Input Type: `bea:Object` • Return Type: `integer` • Description: This function takes a single argument of type `bea:Object` and returns the value as `integer`. If the `bea:Object` is an instance of `java.lang.Integer`, then its value is used directly. If the `bea:Object` is an instance of `java.lang.Number`, then this function invokes `Number.intValue()` on the value. Otherwise, this function converts the `bea:Object` to `java.lang.String` and parses the `String` into an `integer`.

Object Comparisons

Table A-9 lists the functions that Oracle provides to compare Java objects.

Table A-9 WebLogic Server Object Comparisons

Function	Description
`object-is-null`	• URI: `urn:bea:xacml:2.0:function:object-is-null` • Input Type: `bea:Object` • Return Type: `boolean` • Description: This function takes a single argument of type `bea:Object` and returns a boolean value indicating whether the object reference is equal to the Java keyword null. If there is no object that corresponds to the given object reference, this function returns true.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table A-9 (Cont.) WebLogic Server Object Comparisons

Function	Description
`object-equal`	• URI: `urn:bea:xacml:2.0:function:object-equal` • Input Type: `bea:Object`, `bea:Object` • Return Type: `boolean` • Description: This function takes two arguments of type `bea:Object`, invokes `java.lang.Object.equals()`, and returns a `boolean` value indicating whether the two `Objects` are equal. For information about the `java.lang.Object.equals()` method, see Object in Java SE and JDK API Specification .
`object-greater-than`	• URI: `urn:bea:xacml:2.0:function:object-greater-than` • Input Type: `bea:Object`, `bea:Object` • Return Type: `boolean` • Description: This function takes two arguments of type `bea:Object` and returns a `boolean` value indicating whether the first `bea:Object` is greater than the second `bea:Object`. The two `bea:Objects` must implement `java.lang.Comparable`, else the evaluation is indeterminate.
`object-greater-than-or-equal`	• URI: `urn:bea:xacml:2.0:function:object-greater-than-or-equal` • Input Type: `bea:Object`, `bea:Object` • Return Type: `boolean` • Description: This function takes two arguments of type `bea:Object` and returns a `boolean` value indicating whether the first `bea:Object` is greater than or equal to the second bea:Object. The two `bea:Objects` must implement `java.lang.Comparable`, else the evaluation is indeterminate.
`object-less-than`	• URI: `urn:bea:xacml:2.0:function:object-less-than` • Input Type: `bea:Object`, `bea:Object` • Return Type: `boolean` • Description: This function takes two arguments of type `bea:Object` and returns a `boolean` value indicating whether the first `bea:Object` is less than the second `bea:Object`. The two `bea:Objects` must implement `java.lang.Comparable`, else the evaluation is indeterminate.
`object-less-than-or-equal`	• URI: `urn:bea:xacml:2.0:function:object-less-than-or-equal` • Input Type: `bea:Object`, `bea:Object` • Return Type: `boolean` • Description: This function takes two arguments of type `bea:Object` and returns a `boolean` value indicating whether the first `bea:Object` is less than or equal to the second `bea:Object`. The two `bea:Objects` must implement `java.lang.Comparable`, else the evaluation is indeterminate.
`object-collection-contains`	• URI: `urn:bea:xacml:2.0:function:object-collection-contains` • Input Type: `bea:Object`, `bea:Object` • Return Type: `boolean` • Description: This function takes two arguments of type `bea:Object` and returns a `boolean` that indicates whether the first `bea:Object` contains the second `bea:Object`, as determined by `Collection.contains()`. The first `bea:Object` must implement `java.util.Collection`, else the evaluation is indeterminate. For information about the `Collection.contains()` method, see Interface Collection <E> in Java SE and JDK API Specification Java SE and JDK API Specification .

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table A-9 (Cont.) WebLogic Server Object Comparisons

Function	Description
`object-collection-contains-all`	• URI: `urn:bea:xacml:2.0:function:object-collection-contains-all` • Input Type: `bea:Object`, `bea:Object` • Return Type: `boolean` • Description: This function takes two arguments of type `bea:Object` and returns a `boolean` that indicates whether the first `bea:Object` contains all of the second `bea:Object` as determined by `Collection.containsAll()`. Both `bea:Objects` must implement `java.util.Collection`, else the evaluation is indeterminate. For information about the `Collection.containsAll()` method, see Interface Collection <E> in Java SE and JDK API Specification .

String Comparisons and Manipulations

Table A-10 lists the functions that Oracle provides to compare Java objects.

Table A-10 WebLogic Server String Comparisons and Manipulations

Function	Description
`string-char-at`	• URI: `urn:bea:xacml:2.0:function:string-char-at` • Input Type: `string`, `integer` • Return Type: `bea:Character` • Description: This function takes two arguments of type `string` and `integer`, searches in the `string` for the character that is in the position indicated by the `Integer`, and returns the character as a `bea:Character`.
`string-compare-to-ignore-case`	• URI: `urn:bea:xacml:2.0:function:string-compare-to-ignore-case` • Input Type: `string`, `string` • Return Type: `integer` This function takes two arguments of type `string` and returns an `integer` that indicates how the two string arguments compare: • `0` if the strings are identical • Less than `0` if the first string lexically precedes the second string • Greater than `0` if the first string lexically follows the second string Comparisons are preformed without considering case.
`string-contains`	• URI: `urn:bea:xacml:2.0:function:string-contains` • Input Type: `string`, `string` • Return Type: `boolean` • Description: This function takes two arguments of type `string` and returns a `boolean` value that indicates whether the first `string` contains the value of the second `string` as a substring.
`string-starts-with`	• URI: `urn:bea:xacml:2.0:function:string-starts-with` • Input Type: `string`, `string` • Return Type: `boolean` • Description: This function takes two arguments of type `string` and returns a `boolean` value that indicates whether the first `string` value starts with the value of the second `string`.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table A-10 (Cont.) WebLogic Server String Comparisons and Manipulations

Function	Description
`string-ends-with`	• URI: `urn:bea:xacml:2.0:function:string-ends-with` • Input Type: `string`, `string` • Return Type: `boolean` • Description: This function takes two arguments of type `string` and returns a `boolean` that indicates whether the first `string` value ends with the value of the second `string`.
`string-length`	• URI: `urn:bea:xacml:2.0:function:string-length` • Input Type: `string` • Return Type: `integer` • Description: This function takes a single argument of type `string` and returns an `integer` that indicates the length of the string value.
`string-replace`	• URI: `urn:bea:xacml:2.0:function:string-replace` • Input Type: `string`, `bea:Character`, `bea:Character` • Return Type: `string` • Description: This function takes three arguments of type `string`, `bea:Character`, and `bea:Character`, replaces in the `string` all instances of the first `bea:Character` value with the value of the second `bea:Character`, and returns the result as a `string`.
`string-substring`	• URI: `urn:bea:xacml:2.0:function:string-substring` • Input Type: `string`, `integer`, `integer` • Return Type: `string` • Description: This function takes three arguments of type `string`, `integer`, and `integer`, and returns a `string` that is the substring of the `string` argument from and including the index of the first `integer` argument to but excluding the index of the second `integer` argument.
`string-normalize-to-upper-case`	• URI: `urn:bea:xacml:2.0:function:string-normalize-to-upper-case` • Input Type: `string` • Return Type: `string` • Description: This function takes a single argument of type `string`, normalizes it to upper case, and returns the result as a `string`.

Rule and Policy-Combining Algorithm

If multiple `PolicySets` apply to a decision, their results are combined using the following algorithm:

`urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides`

Copyright © 2007, 2026, Oracle and/or its affiliates.