BEA WebLogic Server®

Securing WebLogic Resources Using Roles and Policies

Version 10.0
Revised: March 30, 2007

Contents

4. Options for Securing Web Application and EJB Resources

5. Security Policies

6. Users, Groups, And Security Roles

7. Using XACML Documents to Secure WebLogic Resources

CHAPTER 1

Introduction and Roadmap

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 runtime.

The following sections describe the content and organization of this document:

Document Scope and 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 Java EE 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 Information" on page 1-2.

Guide to This Document

The document is organized as follows:

- This chapter, Introduction and Roadmap, introduces the organization of this guide.
- Chapter 2, "Understanding WebLogic Resource Security," introduces terms and concepts, provides a workflow summary, and outlines the main steps for securing WebLogic resources.
- Chapter 3, "Resource Types You Can Secure with Policies," describes the different types of WebLogic resources that can be secured using the WebLogic Server Administration Console.
- Chapter 4, "Options for Securing Web Application and EJB Resources," describes options for securing EJB and Web application resources using deployment descriptors and/or the WebLogic Server Administration Console.
- Chapter 6, "Users, Groups, And Security Roles," describes users and groups who access WebLogic resources, including WebLogic Server default groups. Also describes scoped security roles and global security roles, including WebLogic Server default global roles. A final section describes the components of a security role.
- Chapter 5, "Security Policies," describes security policies, including WebLogic Server default security policies. Also describes the components of a security policy.

Related Information

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

- *Understanding WebLogic Security*—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.

- *Securing 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.
- Secure WebLogic Resources in *Administration Console Online Help*—Provides step-by-step instructions for using the WebLogic Server Administration Console to complete the tasks that this document describes.

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 *Programming WebLogic Security.*
- "Configuring Access Control" in *Programming WebLogic jCOM* (COM resources).
- "Security" in *Programming WebLogic Resource Adapters* (EIS resources).
- *WebLogic Web Services: Security* (Web Services resources).

Tutorials and Samples

Additional security documents are listed on the Sample Application Examples and Tutorials page.

New and Changed Features

For information about the new and changed features in WeLogic Server 10.0, see What's New in WebLogic Server 10.0.

CHAPTER 2

Understanding WebLogic Resource Security

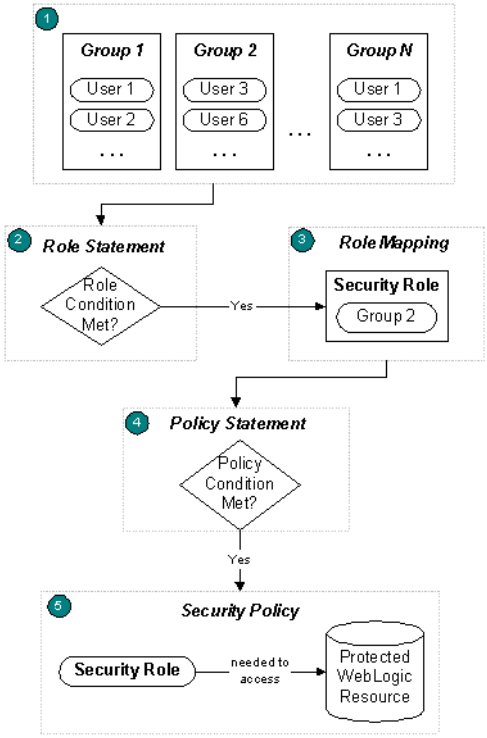
This chapter introduces terms and concepts, provides a workflow summary, and outlines the main steps for securing WebLogic resources:

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 runtime.

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.

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.

   BEA 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 runtime, 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.

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 runtime, 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:

- "Protecting Policies by Type" on page 2-7
- "Protecting a Hierarchy of Resources" on page 2-7

Protecting Policies 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

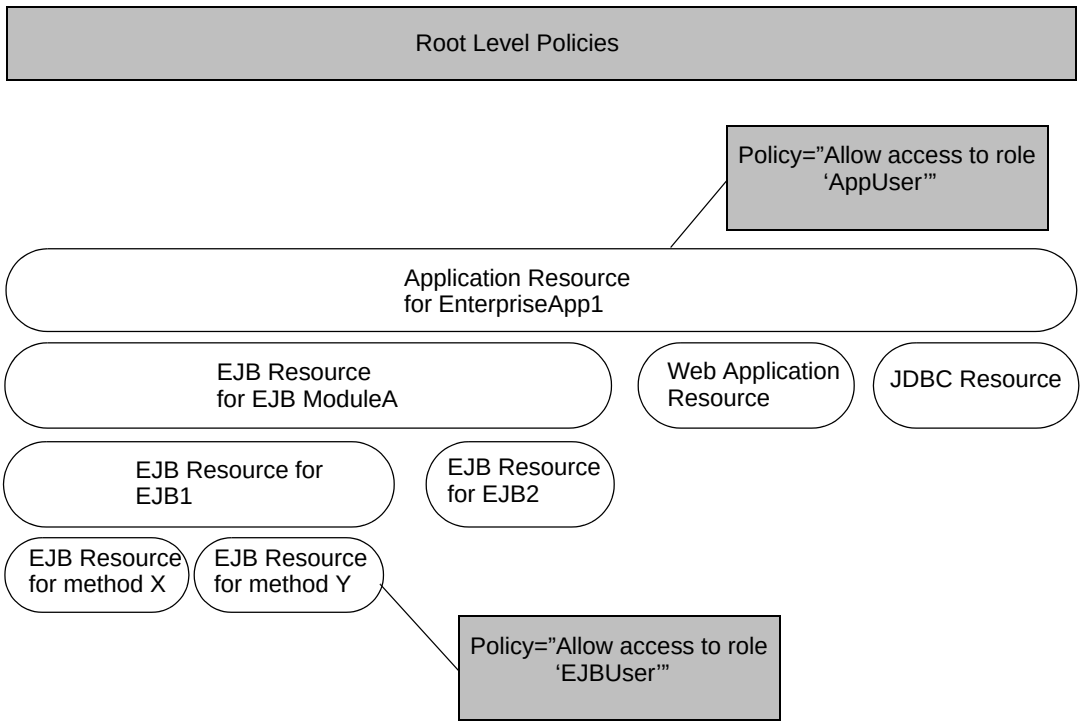
All of the resources within a Java EE application or module that you deploy exist within a hierarchy, and policies on resources higher in the hierarchy act as default policies for resources lower in the same hierarchy. Policies lower in a hierarchy always override policies higher in the hierarchy.

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:

1. Is there a policy for this EJB method? No, therefore go to the next higher level in the hierarchy.
2. Is there a policy for the EJB that contains this method? No, therefore go to the next higher level in the hierarchy.
3. 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.
4. Is there a policy for the enterprise application that contains this URL pattern? Yes, use it. (If there were no such policy, the Security Service would have used the default root-level policy for EJBs.)

Figure 2-2 Hierarchy of Resources and Policies



You can see a visual representation of resource and policy hierarchies in the Administration Console on the security realm's Roles and Policies: Policies page. For information about accessing this page, see Create Policies for Resource Instances in the *Administration Console Help*.

Designing Roles and Policies for WebLogic Resources: Main Steps

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" on page 3-1.

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

   – Server resources, administrative resources, and JMX 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`).

     For these tasks, WebLogic Server already provides a detailed, layered security scheme that grants different types of access to four security roles (Admin, Deployer, Operator, Monitor). For most environments, this security scheme is adequate and only requires you to assign users to the four 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" on page 3-2, "Server Resources" on page 3-11 and "JMX Resources" on page 3-8.

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

     The Java 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 Administration Console to define polices and roles, which offers unified and dynamic security management. See "Options for Securing Web Application and EJB Resources" on page 4-1.

– 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, these resources are not protected by policies; you must define policies to determine 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" on page 5-1.

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 four default global roles. The WebLogic Server security scheme allows only the four 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 runtime, 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" on page 6-1.

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

   a. To create the users and groups, see Manage Users and Groups in *Administration Console Online Help*.

   b. To create security roles, see Manage Security Roles in *Administration Console Online Help*.

c. To create security policies, see Manage Security Policies in *Administration Console Online Help*.

Best Practices: Conditionalize Policies or Conditionalize Roles

Because both roles and policies can evaluate a set of conditions at runtime, 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 (`DefaultAuthorizer`) and the WebLogic Role Mapping provider (`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.

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. WebLogic Server also includes the WebLogic Authorization provider (`DefaultAuthorizer`) and the Role Mapping provider (`DefaultRoleMapper`), which use a proprietary policy language. The

`DefaultAuthorizer` and `DefaultRoleMapper` providers are the only BEA provider 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.

- 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
```

CHAPTER 3

Resource Types You Can Secure with Policies

The following sections describe the types of resources that you can secure using policies:

Administrative Resources

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" on page 3-8.

Figure 3-1 Some Policies Overlap

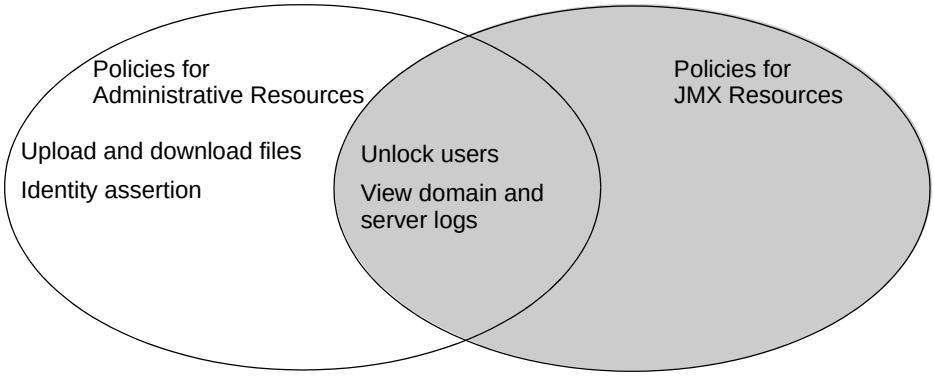


Table 3-1 describes the administrative activities that administrative resources protect and which of these activities are also protected by additional JMX resources. For activities that are protected

by multiple resources, the default policy in the JMX resource duplicates the protections in the Administrative resource.

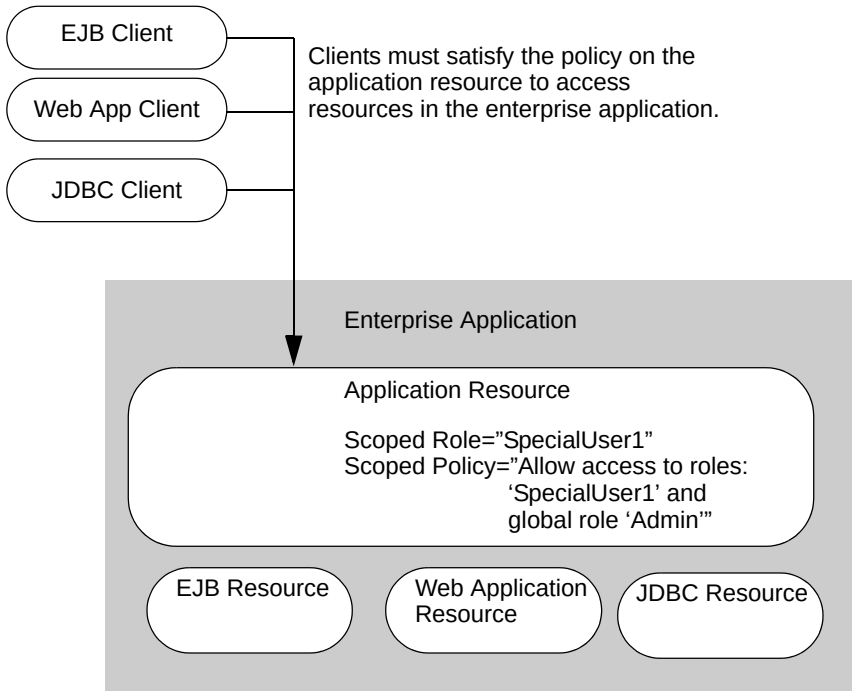
Table 3-1 Activities And Default Policies For Administrative Resources

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. BEA recommends that you do not modify the default policies for any of its methods. They are listed here only for completeness.	`Admin, Operator`	No
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 *WebLogic Server API Reference*.	`Admin`	No
View domain and server logs through the Administration Console.	`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 Java 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).

Figure 3-2 Application Resource Protects All Resources



See "Protecting a Hierarchy of Resources" on page 2-7.

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 Java/Java 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 the "Configuring Access Control" section of *Programming WebLogic jCOM*.

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" on page 2-7.

Because the Java EE platform standardizes EJB security in deployment descriptors, WebLogic Server integrates this standard mechanism with its Security Service to give you a choice of techniques for securing EJB resources. For more information, see Chapter 4, "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. BEA supports resource adapters developed by EIS vendors and third-party application developers. Resource adapters. can be deployed in any application server supporting the applicable Sun Microsystems Java 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" on page 2-7.

For related information, see the "Security" section of *Programming WebLogic Resource Adapters.*

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" on page 3-4).

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" on page 2-7.

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.
- `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.

Note: If you use an Oracle database, you can also control access to JDBC resources using an Oracle Virtual Private Database (VPD). For more information, see "Programming with Oracle Private Virtual Databases" in *Using Third-Party Drivers with WebLogic Server*.

Java Messaging Service (JMS) Resources

A Java Messaging Service (JMS) resource is a JMS system resource, JMS module that is part of an application, JMS destination, or an operation within a destination. You can create security policies and roles for all destinations (JMS queues and JMS topics) as a group, or an individual destination (JMS queue or JMS topic) on a JMS server.

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" on page 2-7.

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()`, `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.

JNDI Operations

For each JNDI node, you can create a policy for all operations or for one of the following operations:

- `modify`—Whenever an application modifies the JNDI tree in any way (that is, adding, removing, changing) the current user must have permission to make the modification. This includes the `bind()`, `rebind()`, `createSubContext()`, `destroySubContext()`, and `unbind()` methods.
- `lookup`—Whenever an application looks up an object in the JNDI tree, the current user must have permission to perform the lookup. This includes the `lookup()` and `lookupLink()` methods.
- `list`—Whenever an application lists the contents of a context in JNDI, the current user must have permission to perform the listing operation. This includes the `list()` and `listBindings()` methods.

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.

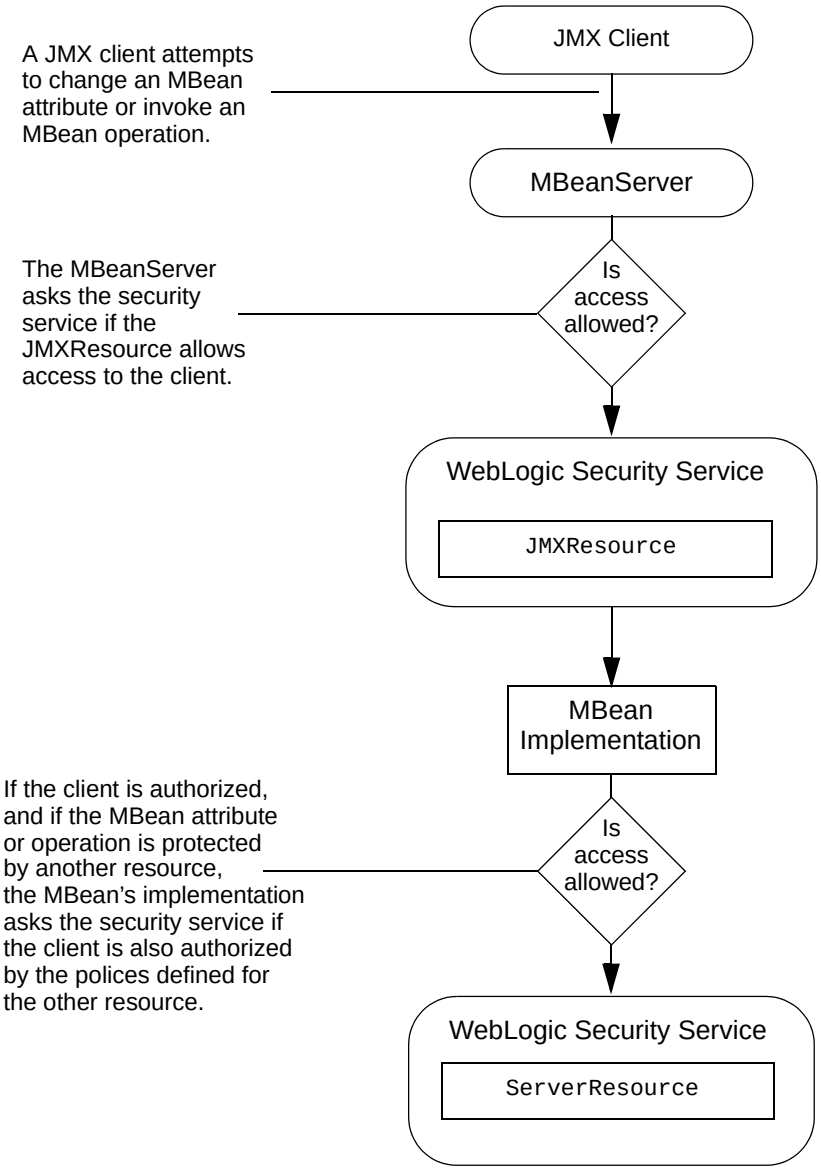
WebLogic Server uses managed beans (MBeans) in the implementation of its management system. Almost all administrative activities require you to invoke an MBean operation or modify an MBean attribute using a Java Management Extensions (JMX) client. For example, the Administration Console is a JMX client. If you use it to change the value of a server's listen port, the Administration Console changes the value of an MBean attribute. The WebLogic Scripting Tool is also a JMX client. For more information, see Understanding WebLogic Server MBeans in *Developing Custom Management Utilities with JMX*.

BEA provides a default set of JMX resources to protect WebLogic Server MBeans. (See Default Security Policies for MBeans in the *WebLogic Server MBean Reference*.) 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.

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).

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.

  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.

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 Administration Console, 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" on page 3-8.

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

- `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 WebLogic Server Administration Console 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 Administration Console.
- `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 Administration Console.

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 *Administration Console Online Help*.

Caution: 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" on page 3-11.

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.

For more information about `weblogic.Server`, see "weblogic.Server Command-Line Reference" in the *WebLogic Server Command Reference*.

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.

For more information, see "Using Node Manager to Control Servers" in *Managing Server Startup and Shutdown*.

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.

For more information about shutting down a WebLogic Server instance, see "Starting and Stopping Servers: Quick Reference" in *Configuring and Managing 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" on page 2-7.

Because the Java 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. For more information, see Chapter 4, "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.

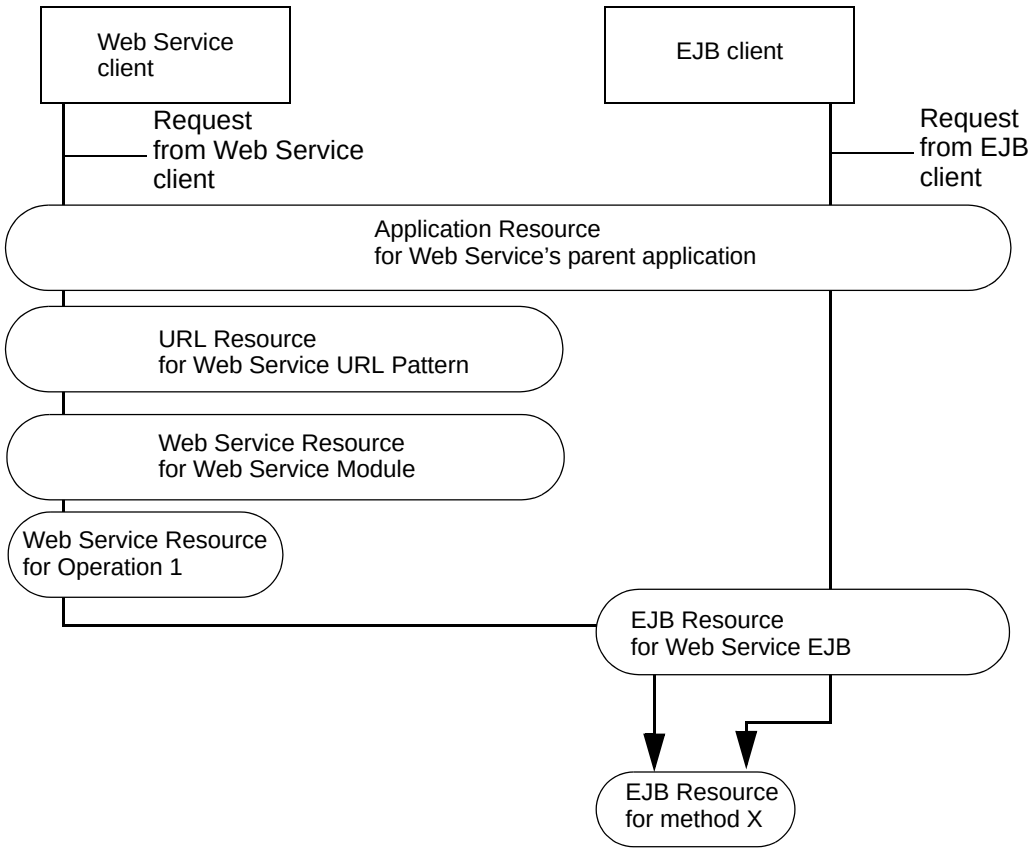
  If you implement the Web Service with an EJB any of the above **or** any of the following resources protect the EJB implementation:

  – The EJB resource for the EJB.

  – Individual EJB resources for each EJB method.

If you use an EJB to implement your Web Service, BEA 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).

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 *WebLogic Web Services: Security*.

Work Context Resources

Work Contexts enable Java 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.

For more information, see Best Practices for Application Design in *Programming WebLogic RMI*.

CHAPTER 4

Options for Securing Web Application and EJB Resources

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

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

Note: If you are implementing security using JACC (Java Authorization Contract for Containers as defined in JSR 115), you must use the Java EE standard model. Other WebLogic Server models are not available and the security functions for Web applications and EJBs in the Administration Console are disabled. See Using the Java Authorization Contract for Containers in *Programming WebLogic 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.

The following sections describe options for securing Web application and Enterprise Java Bean (EJB) resources:

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

Comparison of Security Models for Web Applications and EJBs

Each 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 models.

Table 4-1 Security Model Behaviors

This Model...	Uses Roles and Policies From...	And Performs Security Checks...
Deployment Descriptor Only (Java 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.
Custom Roles	This model uses role mappings from a role mapping provider that you configure for the security realm. You can use the Administration Console to configure the provider. Any role mappings in the deployment descriptors 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.

Table 4-1 Security Model Behaviors

This Model...	Uses Roles and Policies From...	And Performs Security Checks...
Custom Roles and Policies	A role mapping provider and an authorization provider that you configure for the security realm. You can use the Administration Console to configure the providers. Any role mappings or policies in the deployment descriptors are ignored.	For all URLs and EJB methods.
Advanced	Configurable. You can configure this model to use only security data from 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.

Deployment Descriptor Only Model

This is the standard Java EE model and is therefore a widely known technique for adding declarative security to Web applications and EJBs. It uses only roles and policies defined by a developer in the Java EE deployment descriptor (DD) and the WebLogic Server DD. It requires the security 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 4-2 shows a typical scenario:

Table 4-2 Deployment Descriptors Only: Typical Scenario

Company A, Developer	Company A, Admin/Deployer
• Maps EJBs and/or Web URLs to roles in the Java EE DD. • Maps roles to principals in the WebLogic Server DD. • Turns application over to the Admin/Deployer.	• Uses the WebLogic Server Administration Console to ensure that groups exist and are properly mapped to users.

Custom Roles Model

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

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 4-3 shows a typical scenario:

Table 4-3 Customize Roles Only: Typical Scenario

Company A, ISV Developer or Company B, Developer	Company B, Admin/Deployer
• Maps EJBs/URLs to roles in Java EE deployment descriptor. • Turns application over to Admin/Deployer	• Uses the WebLogic Server Administration Console 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 Administration Console 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 Administration Console. 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" on page 4-4).

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 4-4 shows a typical scenario:

Table 4-4 Customize roles and Policies: Typical Scenario

Company A, Developer	Company A, Admin/Deployer
• Provides no mappings in the Java EE or WebLogic Server DD. • Turns application over to Admin/Deployer	• Uses the WebLogic Administration Console 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" on page 4-10):

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

  BEA provides the ability to import security data for the following tasks:

  – As a step toward migrating to full security administration using the Administration Console. The import feature assumes that you want to use the WebLogic Server Administration Console exclusively to secure Web applications and EJBs, but you want to use the security data in deployment descriptors as a baseline.

  – To reinitialize security configurations for Web application and EJB resources to their original state, as specified in the deployment descriptors.

  Once the data is imported, you can use the Administration Console to modify the security data.

  WARNING: Importing security data introduces risks to the integrity of your security data. Each time you import the data, the Security Service attempts to remove all associated data from the provider databases and re-imports data from the deployment descriptors. If you modified the imported security data, then your

modifications could become invalid or could be removed. If you import security data, follow the recommendations in Manage Security for Web Applications and EJBs in the *Administration Console Help*.

- (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.

Three settings in the Administration Console 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:

- "Understanding the Check Roles and Policies Setting" on page 4-8
- "Understanding the When Deploying Web Applications or EJBs Setting" on page 4-8
- "How the Check Roles and Policies and When Deploying Web Applications or EJBs Settings Interact" on page 4-9
- "Understanding the Combined Role Mapping Enabled Setting" on page 4-10

Understanding the Check Roles and Policies Setting

The **Check Roles and Policies** setting determines whether the Security Service performs security checks for all URLs and EJB methods or only those that are protected in the deployment descriptors.

Set the value of **Check Roles and Policies** as follows:

- To perform security checks only on Web application and EJB resources that have security specified in their associated deployment descriptors (DDs), select **Web applications and EJBs Protected in DD**.

  Note: This selection is analogous to the Deployment Descriptor Only security model: the Security Service uses only roles and policies defined in a Web application or EJB's deployment descriptors.

- To perform security checks on all Web application and EJB resources, regardless of whether there are any security settings in the deployment descriptors for these WebLogic resources, select **All Web applications and EJBs**.

  Note: With this selection, you can also configure the **When Deploying Web Applications or EJBs** setting.

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 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**.

Set the value of **When Deploying Web Applications or EJBs** as follows:

- To secure Web application and EJB resources using *only* the WebLogic Server Administration Console, select **Ignore Roles and Policies From DD** (Deployment Descriptors). At this point you can begin to use the Administration Console to secure the resources. See Create Scoped Security Roles and Create Policies for Resource Instances in *Administration Console Online Help.*

- To import security data from the deployment descriptors, select **Initialize Roles and Policies from DD**.

WARNING: 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. If you modified the imported security data, then your modifications could become invalid or could be removed. If you import security data, follow the recommended procedures in Manage Security for Web Applications and EJBs in the *Administration Console Help*.

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

Table 4-5 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 4-5 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* the Administration Console	All Web applications and EJBs	Ignore Roles and Policies from DD

Table 4-5 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	by *copying* or *reinitializing* security data from the deployment descriptors 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
Only on Web applications and EJB methods that are specified in the deployment descriptors (default configuration)	using *only* the deployment descriptors	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.

Table 4-6 compares how this setting affects security for Web applications and EJBs:

Table 4-6 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.

Table 4-6 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

- 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.

  The mapping is the same because if there is no mapping defined for the Web application, WebLogic Server copies the EAR mapping to the WAR mapping.

Securing Web Applications and EJBs

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 the Administration Console to deploy applications, choose a security model, modify roles and polices, and complete other related tasks, see Manage Security for Web Applications and EJBs in the *Administration Console 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 *Programming WebLogic Security*.

CHAPTER 5

Security Policies

A security policy specifies 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."

The following sections describe the features and functions of security policies:

For information on using security policies to protect multiple resources, see "Using Policies to Protect Multiple Resources" on page 2-7.

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 default (active) 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 JACC (Java Authorization Contract for Containers as defined in JSR 115), the policies are stored in the Web application or EJB deployment descriptors.
- 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" on page 4-1.

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.

For more information about the WebLogic Authentication, Authorization, and Role Mapping providers, see "The WebLogic Security Providers" in *Introduction to WebLogic Security*.

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" on page 6-1.

Note: You can access root level policies in the Administration Console. See Create root-level policies in *Administration 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

Table 5-1 Default Security Policies for WebLogic Resources (Continued)

WebLogic Resource	Security Policy
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`

Caution: 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" on page 3-11.

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 runtime, 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 Administration Console:

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" on page 5-4
- "Date and Time Policy Conditions" on page 5-5
- "Context Element Policy Conditions" on page 5-6

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.
- `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 Differences Between Domain Startup Modes in *Creating WebLogic Domains Using the Configuration Wizard*.
- `Allow access to everyone`—Allows access for all users, groups, and roles.
- `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 *WebLogic Web Services: Security*.

Date and Time Policy Conditions

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

- `Access occurs between specified hours`—Allows access during a specified time period. For example, you might create a condition granting access to users only during business hours.
- `Access occurs after`—Allows access after a specified time. For example, you might create a condition that grants access to users after the business opens or after a certain date and time.
- `Access occurs before`—Allows access before a specified time. For example, you might create a condition that grants access to users before the business closes or before a certain date and time.
- `Access occurs on specified days of the week`—Allows access on specified days. For example, you might create a condition that grants access to users on week days.

- `Access occurs on the specified day of the month`—Allows access on an ordinal day of the month. For example, you might create a condition that grants access to users only the first day of each month.
- `Access occurs after the specified day of the month`—Allows access after an ordinal day in the month. For example, you might create a condition indicating that grants access to users after the 15th day of the month.
- `Access occurs before the specified day of the month`—Allows access before an ordinal day in the month. For example, you might create a condition that grants access to users before the 15th day of the month.

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. For more information, see ContextHandlers and WebLogic Resources in *Developing Security Providers for 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.

Protected Public Interfaces

The WebLogic Server Administration Console, the WebLogic Scripting Tool (WLST), and MBean APIs are secured using the default security policies, which are based on the default global roles and default groups described in Table 6-2. Therefore, to use the Administration Console, a user must belong to one of these default groups or be granted one of these global roles. Additionally, administrative operations that require interaction with MBeans are secured using the MBean protections described in "Maintaining a Consistent Security Scheme" on page 3-11. Therefore, interaction with the following protected public interfaces typically must satisfy both security schemes.

- *The WebLogic Server Administration Console*—The WebLogic Security Service verifies whether a particular user can access the Administration Console when the user attempts to log in. If a user attempts to invoke an operation for which they do not have access, they see an Access Denied error.

  For information about using this public interface, see the The WebLogic Server Administration Console in *Administration Console Online Help*.

- *The* `WebLogic Scripting Tool`—The WebLogic Scripting Tool (WLST) is a command-line scripting interface that system administrators and operators can use to monitor and manage WebLogic Server instances and domains. The WebLogic Security Service verifies whether a particular user has permission to execute a WLST command when the user attempts to invoke the command. If a user attempts to invoke an operation for which the user does not have access, WebLogic Server throws a `weblogic.management.NoAccessRuntimeException`, which developers can catch explicitly in their programs. The server sends this exception to its log file, but you can also configure the server to send exceptions to standard out.

  For information about using this public interface, see *WebLogic Scripting Tool*.

  Note: WLST is a convenience utility that abstracts the interaction with the MBean APIs (described next). Therefore, for any administrative task you can perform using WLST, you can also perform using the MBean APIs.

- *MBean APIs*—The WebLogic Security Service verifies whether a particular user has permission to access the API when the user attempts to perform an operation on the MBean. If a user attempts to invoke an operation for which the user does not have access, WebLogic Server throws a `weblogic.management.NoAccessRuntimeException`, which developers can catch explicitly in their programs. The server sends this exception to its log file, but you can also configure the server to send exceptions to standard out.

For information about using these APIs, see Understanding WebLogic Server MBeans in *Developing Custom Management Utilities with JMX*.

Using the Administration Console 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 polices to the provider database.

You can use the WebLogic Administration Console to access WebLogic resources for creating and modifying security policies. For more information, see Manage Security Policies in *Administration Console Online Help*.

CHAPTER 6

Users, Groups, And Security Roles

The following sections describe the features and functions of users, groups, and security 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, BEA 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 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" on page 6-4).

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), and the `system` user if the WebLogic Server instance is running Compatibility security. Any user assigned to the `Administrators` group is granted the `Admin` security role by default. See "Best Practices: Add a User To the Administrators Group" on page 6-3.
`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.
`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.

Runtime Groups

At runtime, 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 in the Administration Console's Groups tab and they are not exported with the authentication database.

Best Practices: Add a User To the Administrators Group

BEA recommends that you add at least one user to the `Administrators` group *in addition to* the user you defined at installation (using the Configuration wizard). Having at least two administrators at all times helps protect against a single admin user being locked out from a potential security breach. Also, avoid using predictable user names like "system", "admin", or "Administrator".

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" on page 5-1.)

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" on page 6-3.
- 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 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. BEA 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" on page 6-4.

  Note: If you are implementing security using JACC (Java authorization Contract for Containers as defined in JSR 115) 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.

Caution: 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" on page 3-11.

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, Java EE Connector, and Web Service modules. • Start, resume, and stop servers.	`Administrators`
`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 schema and ejb-jar deployment descriptor reference.	`everyone`

Table 6-2 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, Java EE Connector, Web Service, and WebLogic Tuxedo Connector components. If applicable, edit deployment descriptors. • Access deployment operations in the Java EE Deployment Implementation (JSR-88). See Understanding WebLogic Server Deployment in *Deploying Applications to WebLogic Server*.	`Deployers`
`Operator`	• View the server configuration, *except* for encrypted attributes. • Start, resume, and stop servers.	`Operators`
`Monitor`	View the server configuration, *except* for encrypted attributes. This security role effectively provides read-only access to the WebLogic Server Administration Console, `WLST`, and MBean APIs.	`Monitors`
`AppTester`	Access applications for testing purposes that are running in Administration mode. For more information, see Administration Mode for Isolating Production Applications in *Understanding WebLogic Server Deployment*.	`AppTesters`
`CrossDomainConnector`	Make inter-domain calls from foreign domains. For more information, see Enabling Trust Between WebLogic Server Domains in *Securing WebLogic Server*.	`CrossDomainConnectors`

Security Role Conditions

To determine who is in a security role at runtime, a role contains one or more conditions. For example, a basic role might simply name the “Administrator” group. At runtime, 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. The entire collection of conditions must be true for a user or group to be granted the security role. More restrictive expressions should come later in a role statement.

The WebLogic Server Role Mapping providers display three kinds of built-in policy conditions in the Administration Console:

Note: These 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 on its capabilities.

- "Basic Role Conditions" on page 6-6
- "Date and Time Role Conditions" on page 6-6
- "Context Element Role Conditions" on page 6-7

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.
- `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, BEA 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 Differences Between Domain Startup Modes 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` 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. For more information, see ContextHandlers and WebLogic Resources in *Developing Security Providers for 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.

Using the Administration Console 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 Manage Users and Groups in *Administration Console Online Help*.

For information on creating security roles, see Manage Security Roles in *Administration Console Online Help*.

CHAPTER 7

Using XACML Documents to Secure WebLogic Resources

The eXtensible Access Control Markup Language (XACML) is an XML language for expressing authorization policies and role assignments. The WebLogic Server XACML Authorization Provider and the WebLogic Server XACML Role Mapping Provider implement the XACML 2.0 Core Specification (see XACML 2.0 Core Specification).

If you need to create security roles or policies that are more complex than can be created with the WebLogic Server Administration Console, or if you are required to express your security roles and policies in a standard language, you can create roles and policies in a XACML document and then use WebLogic Scripting Tool to add them to your security realm. You can also export your realm's roles and policies to a XACML document and then import the document in other WebLogic Server realms. For example, you can export from a realm in a development environment and then import into a production realm.

The following sections describe how to use XACML documents to secure WebLogic resources:

Caution: 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 polices as indeterminate, which locks all users (including the administrative user) out

of the domain. See “Caution: Indeterminate Results Can Lock Out All Users” on page 7-3.

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.
- 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 polices for an EJB or Web application that you have deployed using the Deployment Descriptor Only security model. See “Comparison of Security Models for Web Applications and EJBs” on page 4-2.

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” on page 7-3.
2. “Get the ID of the Resource to Secure” on page 7-3.
3. “Create XACML Documents” on page 7-5.
4. “Use WebLogic Scripting Tool to Add the Role or Policy to the Realm” on page 7-9.
5. “Verify That Your Roles and Policies Are in the Realm” on page 7-10.

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 (`ancest`**`e`**`r` 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-ancest
    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 Administration Console or a Java 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" on page A-2.

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 in Configure Auditing Providers the *Administration 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.

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}
   ```

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" on page 2-7.

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 "Reference for XACML on WebLogic Server" on page A-1.

Example: Defining Role Assignments

The syntax for describing role assignments in a XACML document is specified in the RBAC Profile specification. (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.

(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 Listing 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.

Listing 7-1 XACML Policy for a Role

```
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
   PolicyId="urn:sample:xacml:2.0:myRolePolicy"
   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:
   first-applicable">

<Description>My Role Policy</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"
            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/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 Listing 7-2 specifies that only the `MyRole` role can access the `SimpleSoapPort` Web Service.

Listing 7-2 XACML Policy for a Web Service

```
<Policy PolicyId="urn:sample:xacml:2.0:myID"
   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:
   first-applicable">

   <Description>My XACML Authorization Policy</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"
         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">
           don
        </AttributeValue>
        <SubjectAttributeDesignator
           AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
           DataType="http://www.w3.org/2001/XMLSchema#string"
           SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:
            access-subject"/>
        </Apply>
     </Condition>
   </Rule>
   <Rule RuleId="deny-rule" Effect="Deny"/>
</Policy>
```

Use WebLogic Scripting Tool to Add the Role or Policy to the Realm

The WebLogic Scripting Tool (WLST) is a command-line scripting interface that you can use to load your XACML document into a WebLogic security realm.

You can use WLST in interactive mode or script mode. You cannot use WLST in offline mode; the Authentication provider and Role Mapping provider can update their policy stores only when the Administration Server is running.

For information about using script mode, see Using the WebLogic Scripting Tool in *WebLogic Scripting Tool*.

The following steps describe using the WLST interactive mode:

1. Start the WebLogic Server instance that contains the realm you want to configure.
2. Open a command prompt and set up the environment for running WLST.

   One way to set up the environment is as follows:

   a. Change to the root directory of the domain.

   b. Invoke the `setWLSenv` script (the Domain Configuration Wizard creates this script for you when you create the domain).
3. Change to the directory that contains your XACML document.
4. To start WLST and connect to a WebLogic Server instance that is listening at `localhost:7001`, enter the following commands:

   a. `java weblogic.WLST`

      This command returns a WLST prompt.

   b. `connect('username','password','localhost:7001')`

      where *`username`* and *`password`* are credentials for an administrative user.
5. To load a XACML document into a Java `String` object, enter the following commands:

   a. `xacmlFile = open('myfile','r')`

      where *`myfile`* is the name of your XACML document.

   b. `xacmlDoc = xacmlFile.read()`

c. (Optional) To verify that you have loaded your document into a `String`, enter:
`print(xacmlDoc)`

The command prints the value of the `xacmlDoc` variable to standard out.

6. To load role assignments into the WebLogic XACML Role Mapper, enter the following commands:

a. `cd ('SecurityConfiguration/`*`mydomain`*`/Realms/`*`myrealm`*`/RoleMappers/XACMLRoleMapper')`

where
`mydomain` is the name of your WebLogic Server domain
`myrealm` is the name of your domain's security realm

b. `cmo.addPolicy(xacmlDoc)` or `cmo.addPolicySet(xacmlDoc)`, depending on whether your XACML document contains a `Policy` or `PolicySet`.

7. To load authorization policies into the WebLogic XACML Authorizer, repeat step 5 to load your XACML policies document, Then enter the following commands:

a. `cd ('SecurityConfiguration/`*`mydomain`*`/Realms/`*`myrealm`*`/Authorizers/XACMLAuthorizer')`

where
`mydomain` is the name of your WebLogic Server domain
`myrealm` is the name of your domain's security realm

b. `cmo.addPolicy(xacmlDoc)` or `cmo.addPolicySet(xacmlDoc)`, depending on whether your XACML document contains a `Policy` or `PolicySet`.

To see a full list of operations that you can use to add, modify, or delete policies, see `XACMLAuthorizerMBean` in *WebLogic Server MBean Reference*.

Verify That Your Roles and Policies Are in the Realm

The WebLogic Server Administration Console 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" on page 7-12.

Creating Roles and Polices 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 with JMX*.

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" on page 7-2, 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" on page 7-11
2. "Create XACML Documents" on page 7-5.
3. "Use WebLogic Scripting Tool to Add the Role or Policy to the Realm" on page 7-9.
4. "Verify That Your Roles and Policies Are in the Realm" on page 7-10.

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
```

where:

- *`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.

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=clearNewUserCount, application=,
mbeanType=CustomMBeanType,target=
```

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.

Caution: 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.

For information on how to export security data, see Export Data from Security Providers in the *Administration Console Help*.

APPENDIX A

Reference for XACML on WebLogic Server

The eXtensible Access Control Markup Language (XACML) is an XML language for expressing authorization policies and role assignments. XACML offers extension points so that vendors such as BEA can express vendor-specific resources, data types, and functions in XACML.

The WebLogic Server XACML Authorization Provider and XACML Role Mapping Provider implement and extend the XACML 2.0 Core Specification (see XACML 2.0 Core Specification). These providers partially implement the Core and Hierarchical Role Based Access Control (RBAC) Profile of XACML 2.0 (see RBAC specification).

The following sections describe the extensions that you can use when writing XACML 2.0 documents to protect resources on WebLogic Server and the restrictions that WebLogic Server places on XACML:

This document describes only the WebLogic Server extensions and restrictions for XACML. For a complete reference of the XACML 2.0 language, see XACML 2.0 Core Specification and the RBAC specification.

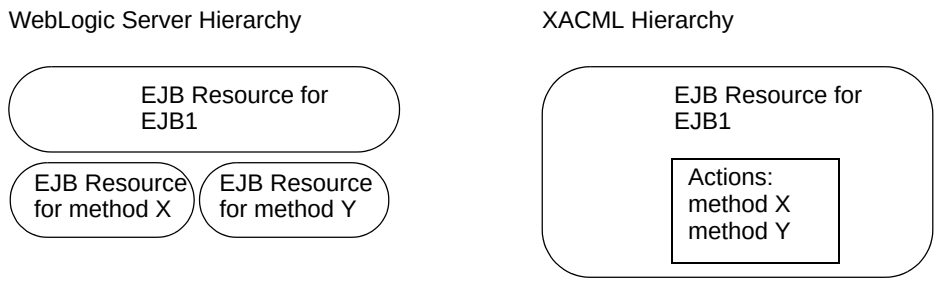
Comparison of WebLogic Server and XACML Security Models

The WebLogic Server model for representing resources and policies follows the model of Java EE deployment descriptors. This Java 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" on page 2-7.) 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 7-1.

Figure 7-1 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 Java EE deployment descriptors. This Java 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, 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

BEA 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 BEA type.

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

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 (see 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" on page A-2.

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.
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`. For more information, see `weblogic.security.spi.Direction` in the *WebLogic Server API Reference*, 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 *WebLogic Server API Reference*.
Application	The name of the application as displayed in the Administration Console.
Control	The name of a method in a Java control. Java controls are reusable components that you can use anywhere within a WebLogic Platform application. You can use built-in controls provided with WebLogic Workshop, or you can create your own.
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 *WebLogic Server API Reference*.
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 *WebLogic Server API Reference*.
JMX	The name of an operation in a WebLogic Server MBean. For example, `shutdown`.

Table A-2 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 *WebLogic Server API Reference*.
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" on page 3-11.
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 *WebLogic Server API Reference*.
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:

```
<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"
     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>
```

Environment Identifiers

XACML uses an optional `Environment` element to describe conditions in the operating environment that must be met before providing access to a target. For example, an `Environment` element can specify a time and date range within which access is allowed.

WebLogic Server supports all of the XACML `Environment` identifiers (see XACML 2.0 Core Specification) and adds support for an additional one that can appear anywhere that a standard XACML environment identifier can appear. See Table A-3.

Table A-3 WebLogic Server Environment Identifier

To Identify...	Use This Identifier...	
A value that the container passes to the provider	Attribute ID:	`urn:bea:xacml:2.0:environment:context:`*`key`* where *`key`* specifies a `ContextHandler` element name as defined in ContextHandlers and WebLogic Resources in *Developing Security Providers for WebLogic Server*. A `ContextHandler` is a WebLogic class that obtains additional context and container-specific information from the resource container and represents the information as a list of name/value pairs.
	Data Type:	`bea:Object`, `string`, or `double`
	Value:	The value of the `ContextHandler` element that you want to match.

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

Policy and PolicySet Identifiers

XACML uses a `Policy` element to contain one or more rules and a `PolicySet` element to contain one or more policies. Each element must include the `PolicySetId` attribute to provide a unique identification. The XACML specification requires `PolicySetId` identifiers to be legal URI values.

XACML documents use the `PolicySetId` to include a specific `Policy` or `PolicySet` element within another `PolicySet` element. WebLogic Server uses the `PolicySetId` as the key in the Authorization provider 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 BEA'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 (see XACML 2.0 Core Specification).

To identify a WebLogic Server resource, use resource identifiers as described in Table A-4. For information about WebLogic Server resources, see "Resource Types You Can Secure with Policies" on page 3-1.

Table A-4 WebLogic Server Resource Identifiers

To Identify a...	Use This 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" on page 7-3.
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" on page 7-3.

Table A-4 WebLogic Server Resource Identifiers

To Identify a...	Use This Identifier...	
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" on page 7-3.
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" on page 7-3.

Examples

The following example `Resource` element matches a Web Service named SimpleSoapPort and all methods within that Web Service:

```
<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"
     DataType="http://www.w3.org/2001/XMLSchema#string"
     MustBePresent="true"/>
   </ResourceMatch>
</Resource>
```

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 (see 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-5.

Table A-5 WebLogic Subject Identifiers

To Identify a...	Use This 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.

Table A-5 WebLogic Subject Identifiers

To Identify a...	Use This Identifier...	

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-5 to define a security role that can be used to protect access to a Web Service, see Listing 7-1.

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:

Custom Data Type Variants

The following function identifiers specify functions that are direct ports of standard XACML functions and operate on XML and WebLogic Server data types `long`, `float`, `decimal` and `bea:Character`. For a description of these data types, see `com.bea.common.security.xacml.Type` in *WebLogic Server API Reference*.

In this list, *`type`* refers to the names of the data types (`long`, `float`, `decimal` or `character`):

`urn:bea:xacml:2.0:function:`*`type`*`-equal`

`urn:bea:xacml:2.0:function:`*`type`*`-greater-than`

`urn:bea:xacml:2.0:function:`*`type`*`-greater-than-or-equal`

`urn:bea:xacml:2.0:function:`*`type`*`-less-than`

`urn:bea:xacml:2.0:function:`*`type`*`-less-than-or-equal`

`urn:bea:xacml:2.0:function:`*`type`*`-one-and-only`

`urn:bea:xacml:2.0:function:`*`type`*`-bag-size`

`urn:bea:xacml:2.0:function:`*`type`*`-is-in`

`urn:bea:xacml:2.0:function:`*`type`*`-bag`

`urn:bea:xacml:2.0:function:`*`type`*`-intersection`

`urn:bea:xacml:2.0:function:`*`type`*`-union`

`urn:bea:xacml:2.0:function:`*`type`*`-at-least-one-member-of`

`urn:bea:xacml:2.0:function:`*`type`*`-subset`

`urn:bea:xacml:2.0:function:`*`type`*`-set-equals`

For information on functions that compare `bea:Objects`, see "Object Comparisons" on page A-34.

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>
```

Miscellaneous Functions

Table A-6 lists the miscellaneous functions that WebLogic Server provides in addition to the standard XACML functions.

Table A-6 Miscellaneous WebLogic Server XACML Functions

Function	Description	
in-development-mode	URI:	`urn:bea:xacml:2.0:function:in-development-mode`
	Input Type:	null
	Return Type:	`boolean`
	Description:	This function takes no arguments and returns `true` if the WebLogic Server instance that hosts the realm is in development mode. See Difference Between Domain Startup Modes in *Creating WebLogic Domains Using the Configuration Wizard*.

Table A-6 Miscellaneous WebLogic Server XACML Functions

Function	Description	
instance-method	URI:	`urn:bea:xacml:2.0:function:instance-method`
	Input Type:	`bea:Object`, `string`, Bag of `bea:Class`, [`bea:Object ...`]
	Return Type:	`bea:Object`
	Description:	This function invokes a method on a `bea:Object` that the container makes available in the current context. The function takes the following arguments: • `bea:Object`. A Java object whose method will be invoked. Use the `urn:bea:xacml:2.0:environment:context:`*`key`* environment identifier to specify the `bea:Object`. See "Environment Identifiers" on page A-7. • `string`. The method name. • Bag of `bea:Class`. The Java classes that are declared in the method's signature. Use BEA's `string-to-class` function to create the required classes. See "Object Type Conversions" on page A-31. • Zero or more `bea:Object`s. Each object contains a parameter value to pass to the method. Use BEA's object conversion functions to create the required objects. See "Object Type Conversions" on page A-31. The function returns the return value of the invoked method as a `bea:Object`. Methods that return void may not be invoked.

Table A-6 Miscellaneous WebLogic Server XACML Functions

Function	Description	
instance-method-match	URI:	`urn:bea:xacml:2.0:function:instance-method-match`
	Input Type:	`bea:Object`, `string`, [`bea:Object ...`]
	Return Type:	`bea:Object`
	Description:	This function invokes a method on a `bea:Object` that the container makes available in the current context. The function takes the following arguments: • `bea:Object`. A Java object whose method will be invoked. Use the `urn:bea:xacml:2.0:environment:context:`*key* environment identifier to specify the `bea:Object`. See "Environment Identifiers" on page A-7. • `string`. The method name. • Zero or more `bea:Object`s. Each object contains a parameter value to pass to the method. Use BEA's object conversion functions to create the required objects. See "Object Type Conversions" on page A-31. This function uses the method name and the class types of the parameter `bea:Object`s to find the appropriate method in the target `bea:Object`. If the target `bea:Object` does not exactly one method that matches the parameters, then the function result is indeterminate. The function returns the return value of the invoked method as an `bea:Object`. Methods that return void may not be invoked.

Table A-6 Miscellaneous WebLogic Server XACML Functions

Function	Description	
instance-method-v2	URI:	`urn:bea:xacml:2.0:function:instance-method-v2`
	Input Type:	`bea:Class`, `bea:Object`, `string`, Bag of `bea:Class`, [`bea:Object ...`]
	Return Type:	`bea:Object`
	Description:	This function invokes a method on a `bea:Object` that the container makes available in the current context. The function takes the following arguments: • `bea:Class`. The class type of the `bea:Object`. Use BEA's `string-to-class` function to create the class. See "Object Type Conversions" on page A-31. • `bea:Object`. A Java object whose method will be invoked. Use the `urn:bea:xacml:2.0:environment:context:`*`key`* environment identifier to specify the `bea:Object`. See "Environment Identifiers" on page A-7. • `string`. The method name. • Bag of `bea:Class`. The Java classes that are declared in the method's signature. Use BEA's `string-to-class` function to create the required classes. See "Object Type Conversions" on page A-31. • Zero or more `bea:Object`s. Each object contains a parameter value to pass to the method. Use BEA's object conversion functions to create the required objects. See "Object Type Conversions" on page A-31. The function returns the return value of the invoked method as an `bea:Object`. Methods that return void may not be invoked.

Table A-6 Miscellaneous WebLogic Server XACML Functions

Function	Description	
instance-method-match-v2	URI:	`urn:bea:xacml:2.0:function:instance-method-match-v2`
	Input Type:	`bea:Class`, `bea:Object`, `string`, [`bea:Object ...`]
	Return Type:	`bea:Object`
	Description:	The function takes the following arguments: • `bea:Class`. The class type of the `bea:Object`. Use BEA's `string-to-class` function to create the class. See "Object Type Conversions" on page A-31. • `bea:Object`. A Java object whose method will be invoked. Use the `urn:bea:xacml:2.0:environment:context:`*`key`* environment identifier to specify the `bea:Object`. See "Environment Identifiers" on page A-7. • `string`. The method name. • Zero or more `bea:Object`s. Each object contains a parameter value to pass to the method. Use BEA's object conversion functions to create the required objects. See "Object Type Conversions" on page A-31. This function uses the class type of the `bea:Object`, method name, and the class types of the parameter `bea:Object`s to find the appropriate method in the target `bea:Object`. If the target `bea:Object` does not contain exactly one method that matches the parameters, then the function result is indeterminate. The function returns the return value of the invoked method as an `bea:Object`. Methods that return void may not be invoked.

Table A-6 Miscellaneous WebLogic Server XACML Functions

Function	Description	
instance-method-match-v3	URI:	`urn:bea:xacml:2.0:function:instance-method-match-v3`
	Input Type:	`string`, `bea:Object`, `string`, Bag of `string`,[`bea:Object ...`]
	Return Type:	`bea:Object`
	Description:	This function invokes uses the Java reflection API to invoke a method on a specified `bea:Object`. The function takes the following arguments: • `String`. The name of the `bea:Object`'s class type. • `bea:Object`. A Java object whose method will be invoked. Use the `urn:bea:xacml:2.0:environment:context:`*`key`* environment identifier to specify the `bea:Object`. See "Environment Identifiers" on page A-7. • `String`. The method name. • Bag of `string`. The class names of the parameters in the method signature. • Zero or more `bea:Object`s. Each object contains a parameter value to pass to the method. Use BEA's object conversion functions to create the required objects. See "Object Type Conversions" on page A-31. This function uses the class name of the `bea:Object`, method name, and the class types of the parameter `bea:Object`s to find the appropriate method in the target `bea:Object`. If the target `bea:Object` does not contain exactly one method that matches the parameters, then the function result is indeterminate. The function returns the return value of the invoked method as an `bea:Object`. Methods that return void may not be invoked.

Example

The following policy uses the `instance-method` function to invoke the `HttpServletRequest.getAuthType()` method on requests that match a specific URL pattern (see `javax.servlet.http.HttpServletRequest.getAuthType()` in *Java EE 5.0 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.

Listing A-1 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"
           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 comparison a 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-7 lists the functions that BEA provides to convert XACML times and dates to different data types.

Table A-7 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`.

Table A-7 WebLogic Server Time/Date Conversions

Function	Description	
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`.

Table A-7 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.security.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-8 lists the functions that BEA provides to convert arithmetic values to different Input Types and to extend the basic set of arithmetic functions specified by XACML.

Table A-8 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`.
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`.

Table A-8 WebLogic Server Arithmetic Conversions and Functions

Function	Description	
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:integer-to-long`
	Input Type:	`string`
	Return Type:	`float`
	Description:	This function takes a single argument of type `string` and returns the argument as a `float`.

Table A-8 WebLogic Server Arithmetic Conversions and Functions

Function	Description	
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`.
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`.

Table A-8 WebLogic Server Arithmetic Conversions and Functions

Function	Description	
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.

Table A-8 WebLogic Server Arithmetic Conversions and Functions

Function	Description	
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 http://grouper.ieee.org/groups/754/.
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`.
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`.

Table A-8 WebLogic Server Arithmetic Conversions and Functions

Function	Description	
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`.

Table A-8 WebLogic Server Arithmetic Conversions and Functions

Function	Description	
tan	URI:	`uurn: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`

where *`type`* is the name of a XACML data type. Table A-9 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">test</Apply>
```

Table A-9 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`

Table A-9 Data to Java Object Conversion

When *type* equals...	The `urn:bea:xacml:2.0:function:`*type*`-to-object` function returns...
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-10 lists the functions that BEA 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 "Environment Identifiers" on page A-7.

Table A-10 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 convert the `bea:Object` to a `java.lang.String` and parses the `String` into a `double`.

Table A-10 WebLogic Server Object Conversions

Function	Description	
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-11 lists the functions that BEA provides to compare Java objects.

Table A-11 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`.
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.

Table A-11 WebLogic Server Object Comparisons

Function	Description	
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:Object`s 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:Object`s 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:Object`s must implement `java.lang.Comparable`, else the evaluation is indeterminate.

Table A-11 WebLogic Server Object Comparisons

Function	Description	
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:Object`s 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.
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:Object`s must implement `java.util.Collection`, else the evaluation is indeterminate.

String Comparisons and Manipulations

Table A-12 lists the functions that BEA provides to compare Java objects.

Table A-12 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 `Interger`, 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`
	Description:	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.

Table A-12 WebLogic Server String Comparisons and Manipulations

Function	Description	
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`.
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`.

Table A-12 WebLogic Server String Comparisons and Manipulations

Function	Description	
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`