BEA WebLogic Server®

Securing WebLogic Server

Version 9.0
Revised: July 22, 2005

Contents

Introduction and Roadmap

The following sections describe the contents and organization of this guide—*Securing WebLogic Server*.

Document Scope

This document explains how to configure security for WebLogic Server®, including information about security providers, identity and trust, SSL, and Compatibility security. See Related Information for a description of other WebLogic security documentation.

Document Audience

This document is intended for the following audiences:

- Application Architects—Architects who, in addition to setting security goals and designing the overall security architecture for their organizations, evaluate WebLogic Server security features and determine how to best implement them. Application Architects have in-depth knowledge of Java programming, Java security, and network security, as well as knowledge of security systems and leading-edge, security technologies and tools.

- Security Developers—Developers who focus on defining the system architecture and infrastructure for security products that integrate into WebLogic Server and on developing custom security providers for use with WebLogic Server. They work with Application Architects to ensure that the security architecture is implemented according to design and that no security holes are introduced, and work with Server Administrators to ensure that security is properly configured. Security Developers have a solid understanding of security concepts, including authentication, authorization, auditing (AAA), in-depth knowledge of Java (including Java Management eXtensions (JMX), and working knowledge of WebLogic Server and security provider functionality.
- Application Developers—Developers who are Java programmers that focus on developing client applications, adding security to Web applications and Enterprise JavaBeans (EJBs), and working with other engineering, quality assurance (QA), and database teams to implement security features. Application Developers have in-depth/working knowledge of Java (including J2EE components such as servlets/JSPs and JSEE) and Java security.
- Server Administrators—Administrators work closely with Application Architects to design a security scheme for the server and the applications running on the server, to identify potential security risks, and to propose configurations that prevent security problems. Related responsibilities may include maintaining critical production systems, configuring and managing security realms, implementing authentication and authorization schemes for server and application resources, upgrading security features, and maintaining security provider databases. Server Administrators have in-depth knowledge of the Java security architecture, including Web services, Web application and EJB security, Public Key security, SSL, and Security Assertion Markup Language (SAML).
- Application Administrators—Administrators who work with Server Administrators to implement and maintain security configurations and authentication and authorization schemes, and to set up and maintain access to deployed application resources in defined security realms. Application Administrators have general knowledge of security concepts and the Java Security architecture. They understand Java, XML, deployment descriptors, and can identify security events in server and audit logs.

Guide to this Document

This document is organized as follows:

- Chapter 2, "Overview of Security Management," explains the differences between security in previous releases of WebLogic Server and this release of WebLogic Server; describes the default security configuration in WebLogic Server; lists the configuration steps for security, and describes Compatibility security.

- Chapter 3, "Customizing the Default Security Configuration," explains when to customize the default security configuration, the configuration requirements for a new security realm, and how to set a security realm as the default security realm.
- Chapter 4, "Configuring WebLogic Security Providers," describes the available configuration options for the security providers supplied by WebLogic Server and how to configure a custom security provider.
- Chapter 5, "Configuring Authentication Providers," describes the Authentication providers supplied by WebLogic Server, including information about how to configure them.
- Chapter 6, "Configuring Single Sign-On with Microsoft Clients," describes how to configure authentication between a WebLogic Server domain and .NET Web Service clients or browser clients (for example, Internet Explorer) in a Microsoft domain, using Windows authentication based on the Simple and Protected Negotiate (SPNEGO) mechanism.
- Chapter 7, "Configuring Single Sign-On with Web Browsers and HTTP Clients," describes how to configure authentication between a WebLogic Server domain and Web browsers or other HTTP clients, using authentication based on the Security Assertion Markup Language (SAML).
- Chapter 8, "Migrating Security Data," provides information about exporting and importing security data between security realms and security providers.
- Chapter 9, "Managing the Embedded LDAP Server," describes the management tasks associated with the embedded LDAP server used by the WebLogic security providers.
- Chapter 10, "Configuring Identity and Trust," describes how to configure identity and trust for WebLogic Server.
- Chapter 11, "Configuring SSL," describes how to configure SSL for WebLogic Server.
- Chapter 12, "Configuring Security for a WebLogic Domain," describes how to set security configuration options for a WebLogic domain.
- Chapter 13, "Using Compatibility Security,"describes how to use Compatibility security, a security configuration mode designed for backwards compatibility with security realms developed under WebLogic Server 6.x.

- Appendix A, “Security Configuration MBeans”, describes which WebLogic Security MBeans and MBean attributes are dynamic (can be changed without restarting the server) or non-dynamic (changes require a server restart).

Related Information

The following BEA WebLogic Server documents contain information that is relevant to the WebLogic Security Service:

- *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.
- *Developing Security Providers for WebLogic Server*—Provides security vendors and application developers with the information needed to develop custom security providers that can be used with WebLogic Server.
- *Securing a Production Environment*—Highlights essential security measures for you to consider before you deploy WebLogic Server in a production environment.
- *Securing WebLogic Resources*—Introduces the various types of WebLogic resources, and provides information about how to secure these resources using WebLogic Server. This document focuses primarily on securing URL (Web) and Enterprise JavaBean (EJB) resources.
- *Programming WebLogic Security*—Describes how to develop secure Web applications.
- Administration Console Online Help—Many security configuration tasks can be performed using the WebLogic Administration Console. The console’s online help describes configuration procedures and provides a reference for configurable attributes.
- *WebLogic Server Upgrade Guide*—Provides procedures and other information you need to upgrade from earlier versions of WebLogic Server to this release. It also provides information about moving applications from an earlier version of WebLogic Server to this release. For specific information on upgrading WebLogic Server security, see *Security* in the *WebLogic Server Upgrade Guide*.
- *Javadocs for WebLogic Classes*—Provides reference documentation for the WebLogic security packages that are provided with and supported by this release of WebLogic Server.

Security Samples and Tutorials

In addition to the documents listed in Related Information, BEA Systems provides a variety of code samples for developers.

Security Examples in the WebLogic Server Distribution

WebLogic Server optionally installs API code examples in *`WL_HOME`*`\samples\server\examples\src\examples\security`, where *`WL_HOME`* is the top-level directory of your WebLogic Server installation. You can start the examples server, and obtain information about the samples and how to run them from the WebLogic Server Start menu.

The following examples illustrate WebLogic security features:

- Java Authentication and Authorization Service
- Outbound and Two-way SSL

Additional Examples Available for Download

Additional API examples are available for download at http://dev2dev.bea.com. These examples are distributed as `.zip` files that you can unzip into an existing WebLogic Server samples directory structure.

You build and run the downloadable examples in the same manner as you would an installed WebLogic Server example. See the download pages of individual examples for more information.

New and Changed Security Features in This Release

WebLogic Server 9.0 introduces several important changes to WebLogic Server security:

New Security Providers

WebLogic Server includes the following new security providers:

Authentication Providers

- **Database Base Management System (DBMS) providers**—Access user, password, group, and group membership information in databases for authentication purposes and can be used to upgrade from the RDBMS security realm. These providers include SQL Authentication, Read-only SQL Authentication, and Custom DBMS Authentication.

- **Windows NT**—Enables the use of Windows NT users and groups for authentication purposes.

Identity Assertion Providers

- **New LDAP X509 Identity Assertion**—Receives an X509 certificate; looks up the LDAP object for the associated user; ensures that the certificate in the LDAP object matches the presented certificate; and retrieves user name from the LDAP object for the purpose of authentication.
- **Negotiate Identity Assertion**—Decodes Simple and Protected Negotiate (SPNEGO) tokens to provide SSO with the Windows desktop by obtaining Kerberos tokens; validates the Kerberos tokens; and maps Kerberos tokens to WebLogic users.

SAML Providers

The SAML providers use OpenSAML 1.0 to support SAML assertion generation and consumption.

- **SAML Identity Assertion**—Consumes and validates SAML assertion tokens and determines whether the assertion is to be trusted (using either the proof material available in the SOAP message, the client certificate, or some other configuration indicator).
- **SAML Credential Mapping**—Generates SAML assertion for users. The SAML assertion contains the name of the requesting user as well as any groups to which the user belongs.

Certificate Lookup and Validation Providers

- **WebLogic CertPath**—Completes certificate paths and validates certificates using the trusted CA configured for a particular server instance. It also checks signatures in the chain; ensures that the chain has not expired; and checks that one certificate in the chain is issued by one of the trusted CAs configured for the server. If any checks fail, the chain is not valid.
- **Certificate Registry**—Supports the Certificate lookup and validation framework. The registry allows the system administrator to explicitly configure a list of trusted CA certificates that are allowed access to the server. The Certificate Registry provides an inexpensive mechanism for performing revocation checking. An administrator revokes a certificate by removing it from the certificate registry. The registry is stored in the embedded LDAP server.

CHAPTER 2

Overview of Security Management

The following sections provide an overview of the security system for WebLogic Server. For a broader overview, see *Understanding WebLogic Security*.

Note: Throughout this document, the term *6.x* refers to WebLogic Server 6.0 and 6.1 and their associated Service Packs.

Security Realms in WebLogic Server

The security service in WebLogic Server simplifies the configuration and management of security while offering robust capabilities for securing your WebLogic Server deployment. Security realms act as a scoping mechanism. Each security realm consists of a set of configured security providers, users, groups, security roles, and security policies. You can configure multiple security realms in a domain; however, only one can be the default (active) security realm. WebLogic Server provides two default security realms:

- `myrealm`—Has the WebLogic Adjudication, Authentication, Identity Assertion, Authorization, Role Mapping, and Credential Mapping providers configured by default.
- `CompatibilityRealm`—Provides backward compatibility for 6.x security configurations. You can access an existing 6.x security configuration through the `CompatibilityRealm`.

Custom security realms written using the security application programming interfaces (APIs) are only supported in Compatibility security. You can customize authentication and authorization functions by configuring a new security realm to provide the security services you want and then set the new security realm as the default security realm.

For information about the default security configuration in WebLogic Server, see "The Default Security Configuration in WebLogic Server" on page 2-6.

For information about configuring a security realm and setting it as the default security realm, see Chapter 3, "Customizing the Default Security Configuration."

For information about Compatibility security, see Chapter 13, "Using Compatibility Security."

Security Providers

Security providers are modular components that handle specific aspects of security, such as authentication and authorization. Although applications can leverage the services offered via the default WebLogic security providers, the WebLogic Security Service's flexible infrastructure also allows security vendors to write their own custom security providers for use with WebLogic Server. WebLogic security providers and custom security providers can be mixed and matched to create unique security solutions, allowing organizations to take advantage of new technology advances in some areas while retaining proven methods in others. The WebLogic Administration Console allows you to administer and manage all your security providers through one unified management interface.

The WebLogic Security Service supports the following types of security providers:

- **Authentication**—Authentication is the process whereby the identity of users or system processes are proved or verified. Authentication also involves remembering, transporting, and making identity information available to various components of a system when that information is needed. Authentication providers supported by the WebLogic Security Service supply the following types of authentication:
  - Username and password authentication
  - Certificate-based authentication directly with WebLogic Server
  - HTTP certificate-based authentication proxied through an external Web server

- **Identity Assertion**—An Authentication provider that performs perimeter authentication—a special type of authentication using tokens—is called an Identity Assertion provider. Identity assertion involves establishing a client's identity through the use of client-supplied tokens that may exist outside of the request. Thus, the function of an Identity Assertion provider is to validate and map a token to a username. Once this mapping is complete, an Authentication provider's LoginModule can be used to convert the username to a principal.
- **Authorization**—Authorization is the process whereby the interactions between users and WebLogic resources are limited to ensure integrity, confidentiality, and availability. In other words, once a user's identity has been established by an authentication provider, authorization is responsible for determining whether access to WebLogic resources should be permitted for that user. An Authorization provider supplies these services.
- **Role Mapping**—You can assign one or more roles to multiple users and then specify access rights for users who hold particular roles. A Role Mapping provider obtains a computed set of roles granted to a requestor for a given resource. Role Mapping providers supply Authorization providers with this information so that the Authorization provider can answer the "is access allowed?" question for WebLogic resources that use role-based security (for example, Web applications and Enterprise JavaBeans (EJBs).
- **Adjudication**—When multiple Authorization providers are configured in a security realm, each may return a different answer to the "is access allowed" question for a given resource. Determining what to do if multiple Authorization providers do not agree is the primary function of an Adjudication provider. Adjudication providers resolve authorization conflicts by weighing each Authorization provider's answer and returning a final access decision.
- **Credential Mapping**—A credential map is a mapping of credentials used by WebLogic Server to credentials used in a legacy or remote system, which tell WebLogic Server how to connect to a given resource in that system. In other words, credential maps allow WebLogic Server to log into a remote system on behalf of a subject that has already been authenticated. Credential Mapping providers map credentials in this way.
- **Keystore**—A keystore is a mechanism for creating and managing password-protected stores of private keys and certificates for trusted certificate authorities. The keystore is available to applications that may need it for authentication or signing purposes. In the WebLogic Server security architecture, the WebLogic Keystore provider is used to access keystores.

  Note: The WebLogic Server Keystore provider is deprecated in this release of WebLogic Server and is only supported for backward compatibility. Use keystores instead. For more information about configuring keystores, see "Configuring Keystores For Production" on page 10-14.

- **Certificate Lookup and Validation (CLV)**—X.509 certificates need to be located and validated for purposes of identity and trust. CLV providers receive certificates, certificate chains, or certificate references, complete the certificate path (if necessary), and validate all the certificates in the path. There are two types of CLV providers:
  - A CertPath Builder looks up and optionally completes the certificate path and validates the certificates.
  - A CertPath Validator looks up and optionally completes the certificate path, validates the certificates, and performs extra validation (for example, revocation checking).
- **Certificate Registry**—A certificate registry is a mechanism for adding certificate revocation checking to a security realm. The registry stores a list of valid certificates. Only registered certificates are valid. A certificate is revoked by removing it from the certificate registry. The registry is stored in the embedded LDAP server. The Certificate Registry is both a CertPath Builder and a CertPath Validator.
- **Auditing**—Auditing is the process whereby information about security requests and the outcome of those security requests is collected, stored, and distributed for the purpose of non-repudiation. In other words, auditing provides an electronic trail of computer activity. An Auditing provider supplies these services.

For information about the functionality provided by the WebLogic security providers, see Chapter 4, "Configuring WebLogic Security Providers" and Chapter 5, "Configuring Authentication Providers."

For information about the default security configuration, see "The Default Security Configuration in WebLogic Server" on page 2-6.

For information about writing custom security providers, see *Developing Security Providers for WebLogic Server*.

Security Policies and WebLogic Resources

WebLogic Server uses security policies (which replace the ACLs and permissions used in WebLogic Server 6.x) to protect WebLogic resources. Security policies answer the question "who has access" to a WebLogic resource. A security policy is created when you define an association between a WebLogic resource and a user, group, or security role. You can also optionally associate a time constraint with a security policy. A WebLogic resource has no protection until you assign it a security policy.

Creating security policies is a multi-step process with many options. To fully understand this process, read *Securing WebLogic Resources*. That document should be used in conjunction with

Securing WebLogic Security to ensure security is completely configured for a WebLogic Server deployment.

WebLogic Resources

A WebLogic resource is a structured object used to represent an underlying WebLogic Server entity, which can be protected from unauthorized access. WebLogic Server defines the following resources:

- Administrative resources such as the WebLogic Administration Console and the WebLogic Scripting Tool.
- Application resources that represent Enterprise applications. This type of resource includes individual EAR (Enterprise Application aRchive) files and individual components, such as EJB JAR files contained within the EAR.
- Component Object Model (COM) resources that are designed as program component objects according to Microsoft's framework. This type of resource includes COM components accessed through BEA's bidirectional COM-Java (jCOM) bridging tool.
- Enterprise Information System (EIS) resources that are designed as resource adapters, which allow the integration of Java applications with existing enterprise information systems. These resource adapters are also known as connectors.
- Enterprise JavaBean (EJB) resources including EJB JAR files, individual EJBs within an EJB JAR, and individual methods on an EJB.
- Java DataBase Connectivity (JDBC) resources including groups of connection pools, individual connection pools, and multipools.
- Java Naming and Directory Interface (JNDI) resources.
- Java Messaging Service (JMS) resources.
- Server resources related to WebLogic Server instances, or servers. This type of resource includes operations that start, shut down, lock, or unlock servers.
- URL resources related to Web applications. This type of resource can be a Web Application aRchive (WAR) file or individual components of a Web application (such as servlets and JSPs).

  Note: Web resources are deprecated in this release of WebLogic Server. Use the URL resource instead.

- Web Services resources related to services that can be shared by and used as components of distributed, Web-based applications. This type of resource can be an entire Web service or individual components of a Web service (such as a stateless session EJB, particular methods in that EJB, the Web application that contains the `web-services.xml` file, and so on).
- Remote resources.

Deployment Descriptors and the WebLogic Administration Console

The WebLogic Security Service can use information defined in deployment descriptors to grant security roles and define security policies for Web applications and EJBs. When WebLogic Server is booted for the first time, security role and security policy information stored in `web.xml`, `weblogic.xml`, `ejb-jar.xml`, or `weblogic-ejb-jar.xml` deployment descriptors is loaded into the Authorization and Role Mapping providers configured in the default security realm. Changes to the information can then be made through the WebLogic Administration Console.

To use information in deployment descriptors, at least one Authorization and Role Mapping provider in the security realm must implement the `DeployableAuthorizationProvider` and `DeployableRoleProvider` Security Service Provider Interface (SSPI). This SSPI allows the providers to store (rather than retrieve) information from deployment descriptors. By default, the WebLogic Authorization and Role Mapping providers implement this SSPI.

If you change security role and security policy in deployment descriptors through the WebLogic Administration Console and want to continue to modify this information through the WebLogic Administration Console, you can set configuration options on the security realm to ensure changes made through the WebLogic Administration Console are not overwritten by old information in the deployment descriptors when WebLogic Server is rebooted.

For more information, see *Securing WebLogic Resources*.

The Default Security Configuration in WebLogic Server

To simplify the configuration and management of security, WebLogic Server provides a default security configuration. In the default security configuration, `myrealm` is set as the default security realm and the WebLogic Adjudication, Authentication, Identity Assertion, Authorization, Credential Mapping, Role Mapping, and CertPath providers are defined as the security providers. WebLogic Server's embedded LDAP server is used as the data store for these default security providers. To use the default security configuration, you need to define users, groups, and security roles for the security realm, and create security policies to protect the WebLogic resources in the domain.

For a description of the functionality provided by the WebLogic Security providers, see the *Understanding WebLogic Security*. If the WebLogic security providers do not fully meet your security requirements, you can supplement or replace them. For more information, see *Developing Security Providers for WebLogic Server*.

If the default security configuration does not meet your requirements, you can create a new security realm with any combination of WebLogic and custom security providers and then set the new security realm as the default security realm. For more information, see Chapter 3, "Customizing the Default Security Configuration."

Configuring WebLogic Security: Main Steps

Because WebLogic Server's security features are closely related, it is difficult to determine where to start when configuring security. In fact, configuring security for your WebLogic Server deployment may be an iterative process. Although more than one sequence of steps may work, BEA Systems recommends the following procedure:

1. Determine whether or not to use the default security configuration by reading "Why Customize the Default Security Configuration?" on page 3-1
   - If you are using the default security configuration, begin at step 3.
   - If you are not using the default security configuration, begin at step 2.
2. Configure additional security providers (for example, configure an LDAP Authentication provider instead of using the WebLogic Authentication provider) or configure custom security providers in the default security realm. This step is optional. By default, WebLogic Server configures the WebLogic security providers in the default security realm (`myrealm`). For information about the circumstances that require you to customize the default security configuration, see "Why Customize the Default Security Configuration?" on page 3-1. For information about creating custom security providers, see *Developing Security Providers for WebLogic Server*.

   Note: You can also create a new security realm, configure security providers (either WebLogic or custom) in the security realm and set the new security realm as the default security realm. See Chapter 3, "Customizing the Default Security Configuration."

3. Optionally, configure the embedded LDAP server. WebLogic Server's embedded LDAP server is configured with default options. However, you may want to change those options to optimize the use of the embedded LDAP server in your environment. For more information, see Chapter 9, "Managing the Embedded LDAP Server."
4. Ensure that user accounts are properly secured. WebLogic Server provides a set of configuration options for protecting user accounts. By default, they are set for maximum security. However, during the development and deployment of WebLogic Server, you may need to weaken the restrictions on user accounts. Before moving to production, check that the options on user accounts are set for maximum protection. If you are creating a new security realm, you need to set the user lockout options. For more information, see "How Passwords are Protected in WebLogic Server" on page 12-5 and "Protecting User Accounts" on page 12-5.
5. Protect WebLogic resources with security policies. Creating security policies is a multi-step process with many options. To fully understand this process, read *Securing WebLogic Resources*. This document should be used in conjunction with *Securing WebLogic Resources* to ensure security is completely configured for a WebLogic Server deployment.
6. Configure identity and trust for WebLogic Server. (This step is optional but encouraged.) For more information, see Chapter 10, "Configuring Identity and Trust."
7. Enable SSL for WebLogic Server. (This step is optional but recommended.) For more information, see Chapter 11, "Configuring SSL."
8. When you have moved to production, review and implement the additional security options described in *Securing a Production Environment*.

In addition, you can:

- Configure a connection filter. See "Using Connection Filters" on page 12-3.
- Enable interoperability between WebLogic domains. See "Enabling Trust Between WebLogic Server Domains" on page 12-1.

Note: In many cases, this document describes how to configure WebLogic security using the WebLogic Administration Console. Generally, any configuration task you can accomplish using the console you can also accomplish using the WebLogic Scripting Tool or the Java Management Extensions (JMX) APIs.

What Is Compatibility Security?

Compatibility security refers to the capability to run security configurations developed under WebLogic Server 6.x in this release of WebLogic Server. In Compatibility security, you manage

6.x security realms, users, groups, and ACLs, protect user accounts, and configure the Realm Adapter Auditing provider and optionally the Identity Assertion provider in the Realm Adapter Authentication provider.

The only security realm available in Compatibility security is the `CompatibilityRealm`. The Realm Adapter providers (Auditing, Adjudication, Authorization, and Authentication) in the Compatibility realm allow backward compatibility with the authentication, authorization, and auditing services in 6.x security realms. For more information, see Chapter 13, "Using Compatibility Security."

Note: Compatibility security is deprecated in this release of WebLogic Server and will not be supported in future major releases. BEA strongly recommends upgrading your WebLogic Server deployment to the security features in this release of WebLogic Server. You should only use Compatibility security pending such an upgrade.

Management Tasks Available in Compatibility Security

Because Compatibility security allows you to access only authentication, authorization, and custom auditing implementations supported in WebLogic Server 6.x, not all 6.x security tasks are allowed in Compatibility security. Use Compatibility security to:

1. Configure the Realm Adapter Auditing provider. For more information, see "Configuring a Realm Adapter Auditing Provider" on page 13-5.

2. Configure the Identity Assertion provider in the Realm Adapter Authentication provider so that implementations of the `weblogic.security.acl.CertAuthenticator` class can be used. For more information, see "Configuring the Identity Assertion Provider in the Realm Adapter Authentication Provider" on page 13-4.

   Note: The Realm Adapter Adjudication and Authorization providers are configured by default in the `CompatibilityRealm` using information in an 6.x existing `config.xml` file. These providers can only be used in the `CompatibilityRealm`. The Realm Adapter Authentication provider is also automatically configured in the `CompatibilityRealm`. However, this provider can also be configured in other realms to provide access to users and groups stored in 6.x security realms. For more information, see "Configuring RDBMS Authentication Providers" on page 5-12.

3. Change the password of the `system` user to protect your WebLogic Server deployment.

4. Manage the security realm in the `CompatibilityRealm`.

5. Define additional users for the security realm in the `CompatibilityRealm`. Organize users further by implementing groups in the security realm.

6. Manage ACLs and permissions for the resources in your WebLogic Server deployment.
7. Create security roles and security policies for WebLogic resources you add to the `CompatibilityRealm`. For more information, see *Securing WebLogic Resources*.

You can still configure identity and trust, use SSL, configure connection filters, and enable interoperability between domains; however, you use the security features available in this release of WebLogic Server to perform these tasks. For more information, see:

- Chapter 10, "Configuring Identity and Trust"
- Chapter 11, "Configuring SSL"
- Chapter 12, "Configuring Security for a WebLogic Domain"

CHAPTER 3

Customizing the Default Security Configuration

The following sections provide information about customizing the default security configuration and creating a new security realm:

- "Why Customize the Default Security Configuration?" on page 3-1
- "Configuration Decisions When Creating a New Security Realm" on page 3-3
- "Creating a New Security Realm: Main Steps" on page 3-4

For information about configuring security providers, see Chapter 4, "Configuring WebLogic Security Providers," and Chapter 5, "Configuring Authentication Providers."

For information about migrating security data to a new security realm, see Chapter 8, "Migrating Security Data."

Why Customize the Default Security Configuration?

To simplify the configuration and management of security, WebLogic Server provides a default security configuration. In the default security configuration, `myrealm` is set as the default (active) security realm, and the WebLogic Adjudication, Authentication, Identity Assertion, Authorization, Credential Mapping, Role Mapping, and CertPath providers are defined as the security providers.

Customize the default security configuration if you want to:

- Replace one of the WebLogic security providers with a different security provider.

- Configure additional security providers in the default security realm. (For example, if you want to use two Authentication providers, one that uses the embedded LDAP server and one that uses a Windows NT store of users and groups.)
- Use an Authentication provider that accesses an LDAP server other than WebLogic Server's embedded LDAP server.
- Use an existing store of users and groups (for example, a DBMS database) instead of defining users and groups in the WebLogic Authentication provider.
- Add an Auditing provider to the default security realm.
- Use an Identity Assertion provider that handles SAML assertions or Kerberos tokens.
- Use the Certificate Registry to add certificate revocation to the security realm.
- Change the default configuration settings of the security providers.

For information about configuring different types of security providers in a security realm, see see Chapter 4, "Configuring WebLogic Security Providers," and Chapter 5, "Configuring Authentication Providers."

The easiest way to customize the default security configuration is to add the security providers you want to the default security realm (`myrealm`). However, BEA recommends the following procedure to customize the default security configuration:

1. Create an entirely new security realm.
2. Configure security providers in the new realm.
3. Migrate any security data, such as users and groups, from the existing default security realm to the new realm.
4. Set the new security realm as the default security realm.

The remainder of this section explains describes the configuration decisions that need to be made when creating a new security realm and the main steps used to create a new security realm. Configuring a security realm is only one step in creating a new security configuration; you also need to configure security providers in that realm before in order for the security realm to be valid. For information about configuring different types of security providers in a security realm, see Chapter 4, "Configuring WebLogic Security Providers," and Chapter 5, "Configuring Authentication Providers."

Configuration Decisions When Creating a New Security Realm

Before creating a new security realm, you need to make decisions about how the WebLogic Security service will use security information defined in deployment descriptors (DDs), the method for securing URLs and EJBs, and how credential maps will be managed.

When creating a new security realm, consider the following:

- Whether or not to set security roles and security policies for Web Applications and EJBs through deployment descriptors or through the WebLogic Administration Console.

  The Check Roles and Security Policies option determines how the WebLogic Security Service uses the security information defined in DDs. The option can be set as follows:

  – Web Applications and EJBs Protected in DD—specifies that the WebLogic Security Service only performs security checks on URL and EJB resources that have security specified in their associated deployment descriptors (DDs). This option is the default Check Roles and Policies setting.

  – All Web Applications and EJBs—specifies that the WebLogic Security Service performs security checks on all URL (Web) and EJB resources, regardless of whether there are any security settings in the deployment descriptors (DDs) for these WebLogic resources. If you change the setting of the Check Roles and Policies drop-down menu to All Web Applications and EJBs, specify Future Redeploys as specified in Step 2.

- How URL and EJB resources are to be secured. The Future Redeploy option determines how these WebLogic resources are to be secured. The option can be set as follows:

  – To secure URL and EJB resources using only the WebLogic Administration Console, select the `Ignore Roles and Policies From DD` (Deployment Descriptors) option.

  – To secure URL and EJB resources using only the deployment descriptors (that is, the `ejb-jar.xml`, `weblogic-ejb-jar.xml`, `web.xml`, and `weblogic.xml` files), select `Initialize roles and policies from DD` option.

- How to create and manage credential maps. You can load credential maps from a resource adapter's deployment descriptor file (`weblogic-ra.xml`) into the embedded LDAP server and then use the WebLogic Administration Console to create new credential maps, or directly modify credential maps defined in the deployment descriptor.

  Once information from a `weblogic-ra.xml` deployment descriptor file is loaded into the embedded LDAP server, the original resource adapter remains unchanged. Therefore, if you redeploy the original resource adapter (which will happen if you redeploy it through the WebLogic Administration Console, modify it on disk, or restart WebLogic Server), the data will once again be imported from the `weblogic-ra.xml` deployment descriptor file and new credential mapping information may be lost.

- Whether or not to use the Web resource.

  The Web resource is deprecated in this release of WebLogic Server. If you are configuring a custom Authorization provider that uses the Web resource (instead of the URL resource) in the new security realm, enable Use Deprecated Web Resource on the new security realm. This option changes the runtime behavior of the Servlet container to use a Web resource rather than a URL resource when performing authorization.

Note: When creating a new security realm, at least one of the configured Authentication providers must return asserted LoginModules. Otherwise, `run-as` tags defined in deployment descriptors will not work.

For more information, see "Configure new security realms" in the Administration Console online help.

Creating a New Security Realm: Main Steps

To create a new security realm:

1. Define a name and set the configuration options for the security realm. See "Configuration Decisions When Creating a New Security Realm" on page 3-3 and "Configure new security realms" in the Administration Console online help.

2. Configure the required security providers for the security realm. A valid security realm requires an Authentication provider, an Authorization provider, an Adjudication provider, a Credential Mapping provider, and a Role Mapping provider. Otherwise, you will not be able to set the new security realm as the default security realm. See Chapter 4, "Configuring WebLogic Security Providers," and Chapter 5, "Configuring Authentication Providers."

3. Optionally, define Identity Assertion, Auditing, and Cert Registry providers. See Chapter 4, "Configuring WebLogic Security Providers," and Chapter 5, "Configuring Authentication Providers."

4. If you configured the WebLogic Authentication, Authorization, Credential Mapping or Role Mapping provider or the Certificate Registry in the new security realm, verify that the settings of the embedded LDAP server are appropriate. See "Managing the Embedded LDAP Server" on page 9-1.

5. Optionally, configure caches to improve the performance of the WebLogic or LDAP Authentication providers in the security realm. See "Improving the Performance of WebLogic and LDAP Authentication Providers" on page 5-9.

6. Protect WebLogic resources in the new security realm with security policies. Creating security policies is a multi-step process with many options. To fully understand this process, read *Securing WebLogic Resources*. This document should be used in conjunction with *Securing WebLogic Server* to ensure security is completely configured for a WebLogic Server deployment.
7. Protect user accounts in the new security realm from dictionary attacks by setting lockout attributes. See "Protecting User Accounts" on page 12-5 and "Protect user accounts" in the Administration Console online help.
8. Set the new realm as the default security realm for the WebLogic domain. See "Change the default security realm" in the Administration Console online help.

Note that you can also use the WebLogic Scripting Tool or Java Management Extensions (JMX) APIs to create a new security configuration. For information more information, see *WebLogic Scripting Tool*.

Configuring WebLogic Security Providers

The following sections describe how to configure the security providers supplied by WebLogic Server.

WebLogic Server includes such a large number of Authentication providers and Identity Assertion providers that they are better handled in a separate section. For information about configuring Authentication providers and Identity Assertion providers, see "Configuring Authentication Providers" on page 5-1.

When Do I Need to Configure a Security Provider?

By default, most WebLogic security providers are generally configured to run out of the box. However, the following circumstances require you to supply configuration information:

- Before using the WebLogic Identity Assertion provider, define the active token type. See "Configuring Identity Assertion Providers" on page 5-17.
- To map tokens to a user in a security realm, configure the user name mapper in the WebLogic Identity Assertion provider. See "Configuring a WebLogic Credential Mapping Provider" on page 4-15.
- To use auditing in the default (active) security realm, configure either the WebLogic Auditing provider or a custom Auditing provider. See "Configuring the WebLogic Auditing Provider" on page 4-4.
- To use HTTP and Kerberos-based authentication in conjunction with WebLogic Server. See Chapter 6, "Configuring Single Sign-On with Microsoft Clients."
- To use identity assertion based on SAML assertions. See Chapter 7, "Configuring Single Sign-On with Web Browsers and HTTP Clients."
- To use certificate revocation. See "Certificate Registry" on page 4-25.
- To use an LDAP server other than the embedded LDAP server, configure one of the LDAP Authentication providers. An LDAP authentication provider can be used instead of or in addition to the WebLogic Authentication provider. See "Configuring LDAP Authentication Providers" on page 5-5.
- To access user, password, group, and group membership information stored in databases for authentication purposes. See "Configuring RDBMS Authentication Providers" on page 5-12 The RDBMS Authentication providers can be used to upgrade from the RDBMS security realm.
- To use Windows NT users and groups for authentication purposes. See "Configuring a Windows NT Authentication Provider" on page 5-15. The Windows NT Authentication provider is the upgrade path for the Window NT security realm.
- When creating a new security realm, configure security providers for that new realm. See "Creating a New Security Realm: Main Steps" on page 3-4.
- When adding a custom security provider to a security realm or replacing one of the WebLogic security providers with a custom security provider, configure options for the custom security provider. When writing a custom security provider, you can implement options that are configurable through the WebLogic Administration Console. However, those options are implementation-specific and are not addressed in this manual. See *Extending the Administration Console*.

You can use either the WebLogic-supplied security providers or a custom security provider in a security realm. For information about configuring a custom security provider, see "Configure a custom security provider" in the Administration Console online help.

The remainder of this section contains conceptual information and configuration options for each security provider, other than the Authorization security providers, for which see Chapter 5, "Configuring Authentication Providers."

Configuring the WebLogic Authorization Provider

Authorization is the process whereby the interactions between users and resources are limited to ensure integrity, confidentiality, and availability. In other words, authorization is responsible for controlling access to resources based on user identity or other information. By default, the WebLogic Authorization provider is configured. You should only need to configure a WebLogic Authorization provider when creating a new security realm.

See Configure Authorization providers in the Administration Console online help.

Configuring the WebLogic Adjudication Provider

When multiple Authorization providers are configured in a security realm, each may return a different answer to the "is access allowed" question for a given resource. This answer may be `PERMIT`, `DENY`, or `ABSTAIN`. Determining what to do if multiple Authorization providers do not agree on the answer is the primary function of the Adjudication provider. Adjudication providers resolve authorization conflicts by weighting each Authorization provider's answer and returning a final decision.

Each security realm requires an Adjudication provider, and can have no more than one active Adjudication provider. By default, a WebLogic security realm is configured with the WebLogic Adjudication provider. You can use either the WebLogic Adjudication provider or a custom Adjudication provider in a security realm. Note that in the Administration Console, the WebLogic Adjudication provider is referred to as the Default Adjudicator.

By default, most of the configuration options for the WebLogic Adjudication provider are defined. However, you can set the Require Unanimous Permit option to determine how the WebLogic Adjudication provider handles a combination of `PERMIT` and `ABSTAIN` votes from the configured Authorization providers.

- If the option is enabled (the default), all Authorization providers must vote `PERMIT` in order for the Adjudication provider to vote `true`.
- If the option is disabled, `ABSTAIN` votes are counted as `PERMIT` votes.

Configuring a WebLogic Role Mapping Provider

Role Mapping providers compute the set of roles granted to a subject for a given resource. Role Mapping providers supply Authorization providers with this role information so that the Authorization provider can answer the "is access allowed?" question for WebLogic resources. By default, a WebLogic security realm is configured with the WebLogic Role Mapping provider. You can use either the WebLogic Role Mapping provider or a custom Role Mapping provider in a security realm. Note that in the Administration Console, the WebLogic Role Mapping provider is referred to as the Default Role Mapper.

By default, most configuration options for the WebLogic Role Mapping provider are already defined. However, you can set Role Mapping Deployment Enabled, which specifies whether or not this Role Mapping provider imports information from deployment descriptors for Web applications and EJBs into the security realm. This setting is enabled by default.

In order to support Role Mapping Deployment Enabled, a Role Mapping provider must implement the `DeployableRoleProvider` SSPI. Roles are stored by the WebLogic Role Mapping provider in the embedded LDAP server.

For more information, see:

- Users, Groups, And Security Roles in *Securing WebLogic Resources*
- Role Mapping Providers in *Developing Security Providers for WebLogic Server*
- "Configure Role Mapping providers" in the Administration Console online help

Configuring the WebLogic Auditing Provider

Auditing is the process whereby information about operating requests and the outcome of those requests are collected, stored, and distributed for the purposes of non-repudiation. In other words, Auditing providers produce an electronic trail of computer activity.

Configuring an Auditing provider is optional. The default security realm (`myrealm`) does not have an Auditing provider configured. WebLogic Server includes a provider named the WebLogic Auditing provider (referred to as `DefaultAuditor` in the Administration Console). You can also develop custom Auditing providers, as described in *Auditing Providers* in *Developing Security Providers for WebLogic Server*.

The WebLogic Auditing provider can log the events described in Table 4-1. In addition, if you enable configuration auditing (as described in "Configuration Auditing" on page 4-9), the WebLogic Auditing provider can log the events described in Table 4-5.

Table 4-1 WebLogic Auditing Provider Events

Audit Event	Indicates...
AUTHENTICATE	Simple authentication (username and password) occurred.
ASSERTIDENTITY	Perimeter authentication (based on tokens) occurred.
USERLOCKED	A user account is locked because of invalid login attempts.
USERUNLOCKED	The lock on a user account is cleared.
USERLOCKOUTEXPIRED	The lock on a user account expired.
ISAUTHORIZED	An authorization attempt occurred.
ROLEEVENT	A `getRoles` event occurred.
ROLEDEPLOY	A `deployRole` event occurred.
ROLEUNDEPLOY	An `undeployRole` event occurred.
POLICYDEPLOY	A `deployPolicy` event occurred.
POLICYUNDEPLOY	An `undeployPolicy` event occurred.

By default, most configuration options for the WebLogic Auditing provider are already defined. However, when configuring the WebLogic Auditing provider, you need to define the following, which can be configured in the Administration Console on the Configuration: Provider Specific page for the provider, by using WebLogic Scripting tool or by using Java Management Extensions (JMX) APIs:

- Severity—Specifies the severity level appropriate for your WebLogic Server deployment. The WebLogic Auditing provider audits security events of the specified severity, as well as all events with a higher numeric severity rank. For example, if you set the severity level to ERROR, the WebLogic Auditing provider audits security events of severity level ERROR, SUCCESS, and FAILURE. Audit events include both the severity name and numeric rank; therefore, a custom Auditing provider can filter events by either the name or the numeric rank. Auditing can be initiated when the following levels of security events occur:

Table 4-2 Audit Severity Levels

Event Severity	Rank
INFORMATION	1
WARNING	2
ERROR	3
SUCCESS	4
FAILURE	5

- Rotation Minutes—Specifies how many minutes to wait before creating a new `DefaultAuditRecorder.log` file. At the specified time, the audit file is closed and a new one is created. A backup file named `DefaultAuditRecorder.`*`YYYYMMDDHHMM`*`.log` (for example,`DefaultAuditRecorder.200405130110.log`) is created in the same directory.

All auditing information recorded by the WebLogic Auditing provider is saved in *`WL_HOME`*`\`*`yourdomain`*`\`*`yourserver`*`\logs\DefaultAuditRecorder.log` by default. Although an Auditing provider is configured per security realm, each server writes auditing data to its own log file in the server directory. You can specify a new directory location for the `DefaultAuditRecorder.log` file on the command line with the following Java startup option:

```
-Dweblogic.security.audit.auditLogDir=c:\foo
```

The new file location will be `c:\foo\`*`yourserver`*`\logs\DefaultAuditRecorder.log`.

For more information, see “Security” in the *Weblogic Server Command Reference*.

Warning: Using an Auditing provider affects the performance of WebLogic Server even if only a few events are logged.

For more information, see “Configure Auditing providers” in the Administration Console online help.

Auditing ContextHandler Elements

An Audit Event includes a `ContextHandler` that can hold a variety of information or objects. The WebLogic Auditing provider’s Active ContextHandler Entries attribute determines which `ContextElement` entries in the `ContextHandler` are recorded by the Auditing provider. By default, none of the `ContextElements` are audited. Objects in the `ContextHandler` are in most cases logged using the `toString` method.

Table 4-3 lists the available `ContextHandler` entries.

Table 4-3 Context Handler Entries for Auditing

Context Element Name	Description and Type
com.bea.contextelement. servlet.HttpServletRequest	A servlet access request or SOAP message via HTTP `javax.http.servlet.HttpServletRequest`
com.bea.contextelement. servlet.HttpServletResponse	A servlet access response or SOAP message via HTTP `javax.http.servlet.HttpServletResponse`
com.bea.contextelement. wli.Message	A WebLogic Integration message. The message is streamed to the audit log. `java.io.InputStream`
com.bea.contextelement. channel.Port	The internal listen port of the network channel accepting or processing the request `java.lang.Integer`
com.bea.contextelement. channel.PublicPort	The external listen port of the network channel accepting or processing the request `java.lang.Integer`
com.bea.contextelement. channel.RemotePort	The port of the remote end of the TCP/IP connection of the network channel accepting or processing the request `java.lang.Integer`
com.bea.contextelement. channel.Protocol	The protocol used to make the request of the network channel accepting or processing the request `java.lang.String`
com.bea.contextelement. channel.Address	The internal listen address of the network channel accepting or processing the request `java.lang.String`
com.bea.contextelement. channel.PublicAddress	The external listen address of the network channel accepting or processing the request `java.lang.String`

Context Element Name	Description and Type
com.bea.contextelement. channel.RemoteAddress	The remote address of the TCP/IP connection of the network channel accepting or processing the request `java.lang.String`
com.bea.contextelement. channel.ChannelName	The name of the network channel accepting or processing the request `java.lang.String`
com.bea.contextelement. channel.Secure	Is the network channel accepting or processing the request using SSL? `java.lang.Boolean`
com.bea.contextelement. ejb20.Parameter[1-N]	Object based on parameter
com.bea.contextelement. wsee.SOAPMessage	`javax.xml.rpc.handler.MessageContext`
com.bea.contextelement. entitlement.EAuxiliaryID	Used by WebLogic Server internal process. `weblogic.entitlement.expression.EAuxiliary`
com.bea.contextelement. security.ChainPrevalidatedBySSL	The SSL framework has validated the certificate chain, meaning that the certificates in the chain have signed each other properly; the chain terminates in a certificate that is one of the server's trusted CAs; the chain honors the basic constraints rules; and the certificates in the chain have not expired. `java.lang.Boolean`
com.bea.contextelement. xml.SecurityToken	Not used in this release of WebLogic Server. `weblogic.xml.crypto.wss.provider.SecurityToken`
com.bea.contextelement. xml.SecurityTokenAssertion	Not used in this release of WebLogic Server. `java.util.Map`
com.bea.contextelement. webservice.Integrity{id:XXXXX}	`javax.security.auth.Subject`

Context Element Name	Description and Type
com.bea.contextelement. saml.SSLClientCertificateChain	The SSL client certificate chain obtained from the SSL connection over which a sender-vouches SAML assertion was received. `java.security.cert.X509Certificate[]`
com.bea.contextelement. saml.MessageSignerCertificate	The certificate used to sign a Web Services message. `java.security.cert.X509Certificate`
com.bea.contextelement. saml.subject.ConfirmationMethod	The type of SAML assertion: bearer, artifact, sender-vouches, or holder-of-key. `java.lang.String`
com.bea.contextelement. saml.subject.dom.KeyInfo	The `<ds:KeyInfo>` element to be used for subject confirmation with holder-of-key SAML assertions. `org.w3c.dom.Element`

Configuration Auditing

You can configure the Administration Server to emit log messages and generate Audit Events when a user changes the configuration or invokes management operations on any resource within a domain. For example, if a user disables SSL on a Managed Server in a domain, the Administration Server emits log messages. If you have enabled the WebLogic Auditing provider, it writes the Audit Events to an additional security log. These messages and Audit Events provide an audit trail of changes within a domain's configuration (configuration auditing).

The Administration Server writes configuration auditing messages to its local log file. Because all configuration auditing messages are of the `Info` severity, they are not written to the domain-wide message log by default.

Enabling Configuration Auditing

You can do the following to enable configuration auditing:

- Use the Administration Console. On the Configuration: General page for your domain, set the Configuration Audit Type. See Enabling Configuration Auditing in the *Administration Console Online Help*.
- When you start the Administration Server, include one of the following Java options in the `weblogic.Server` command:

- `-Dweblogic.domain.ConfigurationAuditType="audit"`

  Causes the domain to emit Audit Events only.

- `-Dweblogic.domain.ConfigurationAuditType="log"`

  Causes the domain to write configuration auditing messages to the Administration Server log file only.

- `-Dweblogic.domain.ConfigurationAuditType="logaudit"`

  Causes the domain to emit Audit Events and write configuration auditing messages to the Administration Server log file.

See weblogic.Server Command-Line Reference.

- Use the WebLogic Scripting Tool to change the value of the `ConfigurationAuditType` attribute of the `DomainMBean`. See *WebLogic Scripting Tool*.

Configuration Auditing Messages

Configuration auditing messages are of the following severities:

Table 4-4 Configuration Auditing Message Severities

Severity	Description
SUCCESS	A successful configuration change occurred.
FAILURE	An attempt to modify the configuration failed due to insufficient user credentials.
ERROR	An attempt to modify the configuration failed due to an internal error.

Configuration auditing messages are identified by message IDs that fall within the range of `159900-159910`.

The messages use MBean object names to identify resources. Object names for WebLogic Server MBeans reflect the location of the MBean within the hierarchical data model. To reflect the location, object names contain name/value pairs from the parent MBean. For example, the object name for a server's `LogMBean` is: `mydomain:Name=myserverlog,Type=Log,Server=myserver`. See WebLogic Server MBean Trees in *Developing Custom Management Utilities with JMX*.

Table 4-5 summarizes the messages.

Table 4-5 Summary of Configuration Auditing Messages

When This Event Occurs...	WebLogic Server Generates a Message With This ID...	And This Message Text...
Authorized user creates a resource.	`159900`	`USER` *`username`* `CREATED` *`MBean-name`* where *`username`* identifies the WebLogic Server user who logged in and created a resource.
Unauthorized user attempts to create a resource.	`159901`	`USER` *`username`* `CREATED` *`MBean-name`* `FAILED weblogic.management. NoAccessRuntimeException:` *`exception-text stack-trace`* where *`username`* identifies the unauthorized WebLogic Server user.
Authorized user deletes a resource.	`159902`	`USER` *`username`* `REMOVED` *`MBean-name`* where *`username`* identifies the WebLogic Server user who logged in and deleted a resource.
Unauthorized user attempts to delete a resource.	`159903`	`USER` *`username`* `REMOVE` *`MBean-name`* `FAILED weblogic.management. NoAccessRuntimeException:` *`exception-text stack-trace`* where *`username`* identifies the unauthorized WebLogic Server user.

Authorized user changes a resource's configuration.	`159904`	`USER` *`username`* `MODIFIED` *`MBean-name`* `ATTRIBUTE` *`attribute-name`*
		`FROM` *`old-value`* `TO` *`new-value`*
		where *`username`* identifies the WebLogic Server user who logged in and changed the resource's configuration.

Table 4-5 Summary of Configuration Auditing Messages (Continued)

When This Event Occurs...	WebLogic Server Generates a Message With This ID...	And This Message Text...

Unauthorized user attempts to change a resource's configuration.	`159905`	`USER` *`username`* `MODIFY` *`MBean-name`* `ATTRIBUTE` *`attribute-name`*
		`FROM` *`old-value`* `TO` *`new-value`*
		`FAILED weblogic.management. NoAccessRuntimeException:` *`exception-text stack-trace`*
		where *`username`* identifies the unauthorized WebLogic Server user.

Authorized user invokes an operation on a resource. For example, a user deploys an application or starts a server instance.	`159907`	`USER` *`username`* `INVOKED ON` *`MBean-name`* `METHOD` *`operation-name`* `PARAMS` *`specified-parameters`* where *`username`* identifies the WebLogic Server user who logged in and invoked a resource operation.

Unauthorized user attempts to invoke an operation on a resource.	`159908`	`USER` *`username`* `INVOKED ON` *`MBean-name`* `METHOD` *`operation-name`*
		`PARAMS` *`specified-parameters`*
		`FAILED weblogic.management. NoAccessRuntimeException:` *`exception-text stack-trace`*
		where *`username`* identifies the unauthorized WebLogic Server user.

Authorized user enables configuration auditing.	`159909`	`USER` *`username`*`, Configuration Auditing is enabled` where *`username`* identifies the WebLogic Server user who enabled configuration auditing.
Authorized user disables configuration auditing.	`159910`	`USER` *`username`*`, Configuration Auditing is disabled` where *`username`* identifies the WebLogic Server user who disabled configuration auditing.

Note: Each time an authorized user adds, modifies, or deletes a resource, the Management subsystem also generates an `Info` message with the ID `140009` regardless of whether configuration auditing is enabled. For example:

```
<Sep 15, 2005 11:54:47 AM EDT> <Info> <Management> <140009>
<Configuration changes for domain saved to the repository.>
```

While the message informs you that the domain's configuration has changed, it does not provide the detailed information that configuration auditing messages provide. Nor does the Management subsystem generate this message when you invoke operations on resources.

Table 4-6 lists additional message attributes for configuration auditing messages. All configuration auditing messages specify the same values for these attributes.

Table 4-6 Common Message Attributes and Values

Message Attribute	Attribute Value
Severity	`Info`
Subsystem	`Configuration Audit`
User ID	`kernel identity` This value is always `kernel identity`, regardless of which user modified the resource or invoked the resource operation.
Server Name	*`AdminServerName`* Because the Administration Server maintains the configuration data for all resources in a domain, this value is always the name of the Administration Server.
Machine Name	*`AdminServerHostName`* Because the Administration Server maintains the configuration data for all resources in a domain, this value is always the name of the Administration Server's host machine.
Thread ID	*`execute-thread`* The value depends on the number of execute threads that are currently running on the Administration Server.
Timestamp	*`timeStamp`* at which the message is generated.

Audit Events and Auditing Providers

An Audit Event is an object that Auditing providers can read and process in specific ways. An Auditing provider is a pluggable component that the security realm uses to collect, store, and distribute information about operating requests and the outcome of those requests for the purposes of non-repudiation.

If you enable a domain to emit Audit Events, the domain emits the events described in Table 4-7. All Auditing providers that are configured for the domain can handle these events.

All of the events are of severity level `SUCCESS` and describe the security principal who initiated the action, whether permission was granted, and the object (MBean or MBean attribute) of the requested action.

Table 4-7 Summary of Audit Events for Configuration Auditing

When This Event Occurs...	WebLogic Server Generates This Audit Event Object...
A request to create a new configuration artifact has been allowed or prevented.	`weblogic.security.spi.AuditCreateConfigurationEvent` See Javadoc.
A request to delete an existing configuration artifact has been allowed or prevented.	`weblogic.security.spi.AuditDeleteConfigurationEvent` See Javadoc.
A request to modify an existing configuration artifact has been allowed or prevented.	`weblogic.security.spi.AuditInvokeConfigurationEvent` See Javadoc.
A invoke an operation on an existing configuration artifact has been allowed or prevented.	`weblogic.security.spi.AuditSetAttributeConfigurationEvent` See Javadoc.

If you enable the default WebLogic Server Auditing provider, it writes all Audit Events as log messages in its own log file.

Other Auditing providers that you create or purchase can filter these events and write them to output repositories such as an LDAP server, database, or a simple file. In addition, other types of security providers can request audit services from an Auditing provider. See Auditing Providers in *Developing Security Providers for WebLogic Server.*

Configuring a WebLogic Credential Mapping Provider

Credential mapping is the process whereby the authentication and authorization mechanisms of a remote system (for example, a legacy system or application) are used to obtain an appropriate set of credentials to authenticate users to a target WebLogic resource. The WebLogic Credential Mapping provider maps WebLogic Server subjects to the username/password pairs to be used when accessing such resources.

By default, most configuration options for the WebLogic Credential Mapping provider are defined. However, you have the option of setting Credential Mapping Deployment Enabled, which specifies whether or not this Credential Mapping provider imports credential maps from a resource adapter's deployment descriptor (`weblogic-ra.xml` file) into the security realm. This setting is enabled by default.

In order to support Credential Mapping Deployment Enabled, a Credential Mapping provider must implement the `DeployableCredentialProvider` SSPI. The credential mapping information is stored in the embedded LDAP server.

For more information:

- See Credential Mapping Providers in *Developing Security Providers for WebLogic Server*
- See Configure Credential Mapping providers in the Administration Console online help.
- For information about using credential maps, see *Programming WebLogic Resource Adapters*.
- You can also use the WebLogic Scripting Tool or Java Management Extensions (JMX) APIs to create a new security configuration.
- For information about other credential mapping providers, see "Configuring a PKI Credential Mapping Provider" on page 4-16 and "Configuring a SAML Credential Mapping Provider" on page 4-19.

Creating Credential Mappings

You can create credential mappings using the Administration Console. To create a credential mapping for the WebLogic Credential Mapping provider:

1. In the left pane of the console, select Security Realm.
2. Expand Credential Mappings > Default and click New.

3. Enter information about the remote resource to be accessed using this credential mapping. This information is used to identify the remote resource and can include one or more of the following:

   a. Protocol—The protocol to use to reach the remote resource.

   b. Remote Host—The host name of the remote resource.

   c. Remote Port—The port number of the remote resource.

   d. Path—If the remote resource is identified by a path, rather than a hostname and port.

   e. Method—The method on the remote resource this credential is used with.

   Click Next.

4. Enter the name of the local user that you are mapping from. This is the WebLogic username that will be the initiator when you want to access the remote resource using this credential mapping.

5. Enter the name of the remote user that you are mapping to. This is the username that is authorized to access the resource using this credential mapping.

6. Enter the remote password required by the remote resource for the remote user you specified in the previous step.

7. Click Finish.

Configuring a PKI Credential Mapping Provider

The PKI (Public Key Infrastructure) Credential Mapping provider included in WebLogic Server maps (a) a WebLogic Server subject (the initiator) and target resource (and an optional credential action) to (b) a key pair or public certificate that can be used by applications when accessing the targeted resource. The PKI Credential Mapping provider uses the subject and resource name to retrieve the corresponding credential from the keystore.

To use the PKI Credential Mapping provider, you need to:

1. Configure keystores with appropriate keys and distribute the keystores on all machines in a WebLogic Server cluster. Setting up keystores is not a WebLogic Server function. For information about setting up keystores, see the help for the Java keytool utility at http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/keytool.html . See also “Configuring Identity and Trust” on page 10-1 for information about keystores and keys in WebLogic Server.

2. Configure a PKI Credential Mapping provider. A PKI Credential Mapping provider is not already configured in the default security realm (`myrealm`). See “PKI Credential Mapper Attributes” on page 4-17 and “Configure Credential Mapping providers” in the Administration Console online help.
3. Create credential mappings. See “Creating PKI Credential Mappings” on page 4-17.

PKI Credential Mapper Attributes

To configure the PKI Credential Mapping provider, set values for these attributes:

- Keystore Provider—A provider for the Java Security API. If no value is provided for this attribute, the default provider class is used.
- Keystore Type—Two types are supported, JKS (the default) and PKCS12.
- Keystore Pass Phrase—The password used to access the keystore
- Keystore File Name—The location of the keystore relative to the directory where the server was started.

In addition, two optional attributes determine how the PKI Credential Mapping provider locates credential mappings in cases where the exact resource or subject may not be available:

- Use Resource Hierarchy—A credential is located by traversing up the resource hierarchy for each type of resource. The search for all possible PKI credentials will start from the specific resource and will walk up the resource hierarchy to find all possible matches. This is enabled by default.
- Use Initiator Group Names—When a Subject is passed to the PKI Credential Mapper provider, a credential is located by examining the groups of which the initiator is a member. This is enabled by default.

For more information, see “Configure Credential Mapping providers” in the Administration Console online help.

Creating PKI Credential Mappings

You can create credential mappings using the Administration Console. To create a credential mapping for the PKI Credential Mapping provider:

1. In the left pane of the console, select Security Realm.
2. Expand Credential Mappings > PKI and click New.

3. Enter information about the remote resource to be accessed using this credential mapping. This information is used to identify the remote resource and can include one or more of the following:
   - Protocol—The protocol to use to reach the remote resource.
   - Remote Host—The host name of the remote resource.
   - Remote Port—The port number of the remote resource.
   - Path—A path that identifies the remote resource.
   - Method—The method on the remote resource this credential is used with.

   If all of these are left unspecified, then the credential mapping applies to all remote resources.

   Click Next.
4. Select Key Pair or Certificate to indicate the type of credential you are mapping to.
5. Enter the name of the principal that you are mapping from. This is the WebLogic username that will be the initiator when you want to access the remote resource using this credential mapping.
6. Indicate whether the principal that you are mapping from is a user or a group.
7. Optionally, specify a credential action. See "Credential Actions" on page 4-18.
8. Enter the alias used in the keystore to identify the credential.
9. If this is a Key Pair credential, enter the password used to retrieve the credential from the keystore.
10. Click Finish.

Credential Actions

You can optionally label a credential mapping with a Credential Action. The Credential Action is an arbitrary string that you can use to distinguish credential mappings used in different circumstances. For example, you might have one credential mapping used for decrypting a message from a remote resource and another credential mapping used to sign messages to be sent to the same resource. The subject initiator and the target resource are not sufficient to distinguish these two credential mappings. You can use the Credential Action to label one of these credential mappings something like `decrypt` and the other one `sign`. Then, the application calling the PKI

Credential Mapping provider can provide the desired Credential Action value in the `ContextHandler` that is passed to the provider.

Configuring a SAML Credential Mapping Provider

The SAML Credential Mapping provider acts as a producer of SAML security assertions, allowing WebLogic Server to act as a source site for using SAML for single sign-on. The SAML Credential Mapping provider generates valid SAML 1.1 assertions for authenticated subjects based on the configuration of the target site or resource. The SAML Credential Mapping provider can be configured as a SAML Intersite Transfer Service. You can configure the provider to add user information and group membership to the AttributeStatement contained in the assertion, which then can be used as part of the asserted identity. Note that you can define default values in the SAML Credential Mapping provider for generating attributes in assertions and override those values for particular classes of assertions in the assertion configuration, as described in "Produced Assertion Configuration" on page 4-22.

For general information about WebLogic Server's support for SAML, see Securlity Assertion Markup Language (SAML) and Single Sign-On with the WebLogic Security Framework in *Understanding WebLogic Security*. For information about how to use the SAML Credential Mapping provider in a SAML single sign-on configuration, see "Configuring Single Sign-On with Web Browsers and HTTP Clients" on page 7-1.

SAML Authority Configuration

These attributes configure the SAML Credential Mapping provider as a SAML authority.

Table 4-8 Source Site Configuration Attributes

Attribute	Description
Issuer URI	The value of the Issuer attribute for SAML assertions generated by the SAML Credential Mapping provider.
Name Qualifier	The Name Qualifier value used by the Name Mapper. This value is the security or administrative domain that is appended to the name of the subject.

Attribute	Description
Default Time To Live	The assertion lifetime, in seconds. Bearer and artifact assertions must have a non-zero (but hopefully short) time-to-live value. Sender-vouches and holder-of-key assertions may specify 0 to indicate that the `NotBefore` and `NotOnOrAfter` conditions should be omitted from the assertion.
Default Time To Live Delta	If non-zero, allow for clock skew between source site and destination site when evaluating assertion lifetimes. A value of -N allows for accepting assertions up to N seconds before their `NotBefore` time; a value of +N allows for accepting assertions up to N seconds after their `NotOnOrAfter` time; and a value of N allows for accepting assertions up to N seconds before or after their validity period.

Source Site Configuration

These attributes configure the SAML Credential Mapping provider as a SAML source site:

Table 4-9 Source Site Configuration Attributes

Attribute	Description
Source Site URL	The SAML source site URL, which participates in SAML SSO. A SAML `SourceID` is automatically generated from this URL.
Intersite Transfer URIs	The URIs that provide the Intersite Transfer Service for Artifact and POST profiles. The default values are: `/samlits_ba/its/post` `/samlits_ba/its/artifact` `/samlits_cc/its/post` `/samlits_cc/its/artifact`

POST Profile Configuration

These attributes configure the SAML Credential Mapping provider for the POST SAML profile:

Table 4-10 POST Profile Configuration Attributes

Attribute	Description
Post Enabled	Enables the use of Artifact profile. If this is true, then the ITS servlet will be deployed and will respond to incoming SAML POST profile requests.
Default Post Form	The path name of the form to use in POST profile assertions. WebLogic Server includes a default form for this purpose, which you may want to replace with a form that uses your own branding or other information. Additionally, you can override this default for `bearer` type assertions, using the PostForm assertion configuration attribute.

Artifact Profile Configuration

These attributes configure the SAML Credential Mapping provider for the Artifact SAML profile:

Table 4-11 Artifact Profile Configuration Attributes

Attribute	Description
Artifact Enabled	Enables the use of Artifact profile. If this is true, then the ITS servlet will be deployed and will respond to incoming SAML Artifact profile requests.
Assertion Retrieval URIs	One or more URIs to which the SAML Credential Mapping provider should listen for incoming assertion retrieval requests. The default is `/samlars/ars`.
Assertion Store Class Name	The class that acts as the persistent store for Artifact profile assertions. An Assertion Store is required for Artifact profile, so that when assertion retrieval requests come in, the corresponding assertion can be located. You can use a custom class that implements `weblogic.security.providers.saml.SAMLAssertionStore`. If you do not designate a custom class, the default class will be used.

Attribute	Description
Assertion Store Properties	These properties, if provided, will be passed to the `initStore()` method of your custom Assertion Store class.

Produced Assertion Configuration

The following settings are used to configure individual assertions produced by the SAML Credential Mapping provider. You can configure the SAML Credential Mapping provider with any number of assertions, allowing the Credential Mapper to generate different assertions for different purposes. You configure assertions with Java-style properties, in key=value format. The key takes the form `assertionName.propertyName`.

Table 4-12 Produced Assertion Configuration Properties

Property	Description
Assertions	The names assigned to these assertion configurations.
AssertionType	One of: `bearer`, `artifact`, `sender-vouches`, `holder-of-key`.
Target	For `bearer` and `artifact` assertions, the destination site URL for which authentication is desired. The Identity Asserter uses the target URL, in conjunction with the assertion type and issuer URL, to identify the assertion configuration against which the assertion should be validated. This attribute is ignored for `sender-vouches` and `holder-of-key` assertions.
ConsumerURL	For bearer and artifact assertions only, the URL to which the assertion or artifact should be posted or redirected.
PostForm	The POST form template to use with POST profile. This value overrides the default form setting in the SAML source site configuration. Ignored for assertion types other than `bearer`.

Property	Description
TimeToLive	The assertion lifetime, in seconds. Bearer and artifact assertions must have a non-zero (but hopefully short) TTL value. Sender-vouches and holder-of-key assertions may specify 0 to indicate that the `NotBefore` and `NotOnOrAfter` conditions should be omitted from the assertion.
TTLDelta	If non-zero, allow for clock skew between source site and destination site when evaluating assertion lifetimes. A TTLDelta value of -N allows for accepting assertions up to N seconds before their `NotBefore` time; a value of +N allows for accepting assertions up to N seconds after their `NotOnOrAfter` time; and a value of N allows for accepting assertions up to N seconds before or after their validity period.
AudienceURI	An audience URI to include in the assertion. Optional for all assertion types.
Signed	If `true`, assertions will be signed. If `false`, signature elements will be ignored. Signing is optional for all assertion types, but the SAML Token Profile specifies that holder-of-key assertions used for WSS should be signed.
SigIncludeCerts	If `true`, include signing certificates in the signature element.
NameMapperClass	If this attribute is present, then the Credential Mapping provider will use the specified `SAMLCredentialNameMapper` class instead of the default Name Mapper.
IncludeGroups	If `true`, include in the assertion an Attribute Statement containing WebLogic Server groups.

Example of Produced Assertion Configuration

Listing 4-1 is an example of how you might configure produced assertions. It includes both a POST profile assertion, named `mypost`, and an Artifact profile assertion, named `myart`.

Listing 4-1 Produced Assertion Configuration

```
Assertions=mypost,myart
mypost.AssertionType=bearer
mypost.Target=http://www.destination_site.com:7001/targets/PostTarget.j
sp
mypost.ConsumerURL=https://www.destination_site.com:7002/samlacs/acs
mypost.IncludeGroups=true
myart.AssertionType=artifact
myart.Target=http://www.destination_site.com:7001/targets/Art*
myart.ConsumerURL=https://www.destination_site.com:7002/samlacs/acs
```

Configuring the Credential Lookup and Validation Framework

WebLogic Server may receive digital certificates as part of Web Services requests, two-way SSL, or other secure interactions. To validate these certificates, WebLogic Server includes a Certificate Lookup and Validation (CLV) framework, whose function is to look up and validate X.509 certificate chains. The key elements of the CLV framework are the CertPathBuilder and the CertPathValidators. The CLV framework requires one and only active CertPathBuilder which, given a reference to a certificate chain, finds the chain and validates it, and zero or more CertPathValidators which, given a certificate chain, validates it.

When WebLogic Server receives a certificate, the CLV framework uses the security provider configured as the CertPathBuilder to look up and validate the certificate chain. If the certificate chain is found and valid, the framework then calls each configured CertPathValidator, in the order the administrator configured them, to perform extra validation on the chain. The chain is only valid if the builder and all the validators successfully validate it.

A chain is valid only if all of the following are true:

- The certificates in the chain have signed each other properly.
- The chain terminates in a certificate that is one of the server's trusted CAs.
- The chain honors the basic constraints rules (for example, that no certificate in the chain has been issued by a certificate that is not allowed to issue certificates).
- The certificates in the chain have not expired.

WebLogic Server includes two CLV security providers: the WebLogic CertPath provider (which acts as both a CertPathBuilder and a CertPathValidator), described in "CertPath Provider" on page 4-25, and the Certificate Registry, described in "Certificate Registry" on page 4-25. Use just the WebLogic CertPath provider if you want to use trusted CA-based validation of the full certificate chain. Use just the Certificate Registry if you want only to validate that certificates are registered. Use both, designating the Certificate Registry as the current builder, if you want to use both types of validation.

For more information about certificate lookup and validation, see "Configuring Identity and Trust" on page 10-1.

CertPath Provider

The default security realm in WebLogic Server is configured with the WebLogic CertPath provider. The CertPath provider serves two functions. It is a CertPathBuilder and a CertPathValidator. The CertPath provider receives an end certificate or a certificate chain. It uses the server's list of trusted CAs to complete the certificate chain, if necessary. After building the chain, the CertPath provider validates the chain, checking the signatures in the chain, ensuring that the chain has not expired, checking the chain's basic constraints, and verifying that the chain terminates in a certificate issued by one of the server's trusted CAs.

The WebLogic CertPath provider requires no configuration, other than its Current Builder attribute, which indicates whether the CertPath provider should be used as the active certificate chain builder.

Certificate Registry

The Certificate Registry is a security provider that allows you to explicitly register the list of trusted certificates that are allowed to access WebLogic Server. If you configure a Certificate Registry as part of your security realms, then only certificates that are registered in the Certificate Registry will be considered valid. The Certificate Registry provides an inexpensive mechanism for performing revocation checking. By removing a certificate from the Certificate Registry, you can invalidate a certificate immediately. The registry is stored in the embedded LDAP server.

The Certificate Registry is both a CertPath Builder and a CertPath Validator. In either case, the Certificate Registry ensures that the chain's end certificate is stored in the registry, but does no other validation. If you use the Certificate Registry as your security realm's CertPath Builder and you also want to use the WebLogic CertPath provider or another security provider to perform full chain validation, make sure that you register the intermediate and root CAs in each server's trust keystore, and the end certificates in the Certificate Registry.

The default security realm in WebLogic Server does not include a Certificate Registry. Once you configure a Certificate Registry, you can use the WebLogic Administration Console to add, remove, and view certificates in the registry. You can export a certificate from a keystore to a file, using the Java `keytool` utility. You can import a certificate that is a PEM or DER file in the file system into the Certificate Registry using the console. You can also use the console to view the contents of a certificate, including its subject DN, issuer DN, serial number, valid dates, fingerprints, etc.

For more information, see Configure Certification Path providers in the Administration Console online help.

Configuring a WebLogic Keystore Provider

Note: The WebLogic Keystore provider is deprecated in this release of WebLogic Server. It is only supported for backward compatibility. Use Java KeyStores (JKS) instead. All of the functionality that was supported by the WebLogic Keystore provider is available through use of Java KeyStores.

For information about configuring the WebLogic Keystore provider, see Configure keystores in the Administration Console online help.

Configuring Authentication Providers

WebLogic Server includes numerous Authentication security providers. Fundamentally, most of these Authentication providers work in the same way: given a username and password credential pair, the provider attempts to find a corresponding user in the provider's data store. These Authentication providers differ primarily in what they use as a data store: one of many available LDAP servers, a SQL database, or other data store. In addition to these username/password based security providers, WebLogic Server includes identity assertion Authentication providers, which use certificates or security tokens, rather than username/password pairs, as credentials.

The following sections describe how to configure the Authentication security providers supplied by WebLogic Server.

Choosing an Authentication Provider

Authentication is the process whereby the identity of users and system processes are proved or verified. Authentication also involves remembering, transporting, and making identity information available to various components of a system when that information is needed.

The WebLogic Server security architecture supports: certificate-based authentication directly with WebLogic Server; HTTP certificate-based authentication proxied through an external Web server; perimeter-based authentication (Web server, firewall, VPN); and authentication based on multiple security token types and protocols.

Authentication is performed by an Authentication provider. WebLogic Server offers the following types of Authentication providers:

- *The WebLogic Authentication provider* accesses user and group information in WebLogic Server's embedded LDAP server.
- *LDAP Authentication providers* access external LDAP stores. You can use an LDAP Authentication provider to access any LDAP server. WebLogic Server provides LDAP Authentication providers already configured for Open LDAP, Sun iPlanet, Microsoft Active Directory and Novell NDS LDAP servers.
- *RDBMS Authentication providers* access external relational databases. WebLogic Server provides three RDBMS Authentication providers: SQL Authenticator, Read-only SQL Authenticator, and Custom RDBMS Authenticator.
- *The WebLogic Identity Assertion provider* validates X.509 and IIOP-CSIv2 tokens and can use a user name mapper to map that token to a user in a WebLogic Server security realm.
- *The SAML Authentication provider*, which authenticates users based on Security Assertion Markup Language 1.1 (SAML) assertions.
- *The Negotiate Identity Assertion provider*, which uses Simple and Protected Negotiate (SPNEGO) tokens to obtain Kerberos tokens, validates the Kerberos tokens, and maps Kerberos tokens to WebLogic users.
- *The SAML Identity Assertion provider*, which acts as a consumer of SAML security assertions, allowing WebLogic Server to act as a destination site for using SAML for single sign-on.

In addition, you can use:

- Custom (non-WebLogic) Authentication providers, which offer different types of authentication technologies.

- Custom (non-WebLogic) Identity Assertion providers, which support different types of tokens.

Using More than One Authentication Provider

Each security realm must have one at least one Authentication provider configured. The WebLogic Security Framework is designed to support multiple Authentication providers (and thus multiple LoginModules) for multipart authentication. Therefore, you can use multiple Authentication providers as well as multiple types of Authentication providers in a security realm. For example, if you want to use both a retina-scan and a username/password-based form of authentication to access a system, you configure two Authentication providers.

The way you configure multiple Authentication providers can affect the overall outcome of the authentication process. Configure the JAAS Control Flag for each Authentication provider to set up login dependencies between Authentication providers and allow single-sign on between providers. See "Setting the JAAS Control Flag Option" on page 5-3.

Authentication providers are called in the order in which they were configured in the security realm. Therefore, use caution when configuring Authentication providers. You can use the WebLogic Administration Console to re-order the configured Authentication providers, thus changing the order in which they are called. See "Changing the Order of Authentication Providers" on page 5-4.

Setting the JAAS Control Flag Option

When you configure multiple Authentication providers, use the JAAS Control Flag for each provider to control how the Authentication providers are used in the login sequence. You can set the JAAS Control Flag in the WebLogic Administration Console. See "Set the JAAS control flag" in the Administration Console online help. You can also use the WebLogic Scripting Tool or Java Management Extensions (JMX) APIs to set the JAAS Control Flag for an Authentication provider.

The definitions for the JAAS Control Flag values are as follows:

- REQUIRED—The Authentication provider is always called, and the user must always pass its authentication test.
- SUFFICIENT—If the user passes the authentication test of the Authentication provider, no other Authentication providers are executed (except Authentication providers with the JAAS Control Flag set to REQUIRED) because the user was sufficiently authenticated.

- REQUISITE—If the user passes the authentication test of this Authentication provider, other providers are executed but can fail (except for Authentication providers with the JAAS Control Flag set to REQUIRED).
- OPTIONAL—The user is allowed to pass or fail the authentication test of this Authentication provider. However, if all Authentication providers configured in a security realm have the JAAS Control Flag set to OPTIONAL, the user must pass the authentication test of one of the configured providers.

When additional Authentication providers are added to an existing security realm, by default the Control Flag is set to OPTIONAL. If necessary, change the setting of the Control Flag so that the Authentication provider works properly in the authentication sequence.

Changing the Order of Authentication Providers

The order in which WebLogic Server calls multiple Authentication providers can affect the overall outcome of the authentication process. The Authentication Providers table lists the authentication providers in the order in which they will be called. By default, Authentication providers are called in the order in which they were configured. You can use the Administration Console to change the order of Authentication providers. Use the Reorder button on the Security Realms: Providers: Authentication page in the Administration Console to change the order in which Authentication providers are called by WebLogic Server and listed in the console.

For more information, see "Re-order Authentication Providers" in the Administration Console online help.

Configuring the WebLogic Authentication Provider

The WebLogic Authentication provider uses WebLogic Server's embedded LDAP server to store user and group membership information. This provider allows you to edit, list, and manage users and group membership. By default, most of the configuration options for the WebLogic Authentication provider are already defined. You should need to configure a WebLogic Authentication provider only when creating a new security realm. However, note the following:

- The WebLogic Authentication provider is configured in the default security realm with the name `DefaultAuthenticator`.
- User and group names in the WebLogic Authentication provider are case insensitive.
- Ensure that all user names are unique.

- Use the Minimum Password Length option on the Configuration: Provider Specific page to specify the length of passwords defined for users that are stored in the embedded LDAP server.
- If you are using multiple Authentication providers, set the JAAS Control Flag to determine how the WebLogic Authentication provider is used in the authentication process. See "Using More than One Authentication Provider" on page 5-3.

Configuring LDAP Authentication Providers

WebLogic Server provides the following LDAP Authentication providers:

- – iPlanet Authentication provider
- – Active Directory Authentication provider
- – Open LDAP Authentication provider
- – Novell Authentication provider
- – generic LDAP Authentication provider

Each of these LDAP Authentication providers works in basically the same way, storing user and group information in an external LDAP server. They differ primarily in how they are configured by default to match typical directory schemas for their corresponding LDAP server.

WebLogic Server does not support or certify any particular LDAP servers. Any LDAP v2 or v3 compliant LDAP server should work with WebLogic Server. The following LDAP directory servers have been tested:

- Sun iPlanet version 4.1.3
- Active Directory shipped as part of Windows 2000
- Open LDAP version 2.0.7
- Novell NDS version 8.5.1

An LDAP Authentication provider can also be used to access other LDAP servers. However, you must either use the LDAP Authentication provider (`LDAPAuthenticator`) or choose a pre-defined LDAP provider and customize it. See "Accessing Other LDAP Servers" on page 5-7.

Configuring an LDAP Authentication provider is a multi-step process. The steps are as follows.

1. Choose an LDAP Authentication provider that matches your LDAP server and create an instance of the provider in your security realm. See Configure Authentication and Identity Assertion providers in the Administration Console help.

2. Configure the provider-specific attributes of the LDAP Authentication provider, which you can do using the Administration Console. For each LDAP Authentication provider, there are attributes that:

   a. Enable communication between the LDAP server and the LDAP Authentication provider. For a more secure deployment, BEA recommends using the SSL protocol to protect communications between the LDAP server and WebLogic Server. Enable SSL with the `SSLEnabled` attribute.

   b. Configure options that control how the LDAP Authentication provider searches the LDAP directory.

   c. Specify where in the LDAP directory structure users are located.

   d. Specify where in the LDAP directory structure groups are located.

   e. Define how members of a group are located.

3. Configure performance options that control the cache for the LDAP server. Use the Configuration: Provider Specific and Performance pages for the provider in the Administration Console to configure the cache. See "Improving the Performance of WebLogic and LDAP Authentication Providers" on page 5-9.

For more information, see:

- "Requirements for Using an LDAP Authentication Provider" on page 5-6
- "Accessing Other LDAP Servers" on page 5-7
- "Configuring Failover for LDAP Authentication Providers" on page 5-7
- "Improving the Performance of WebLogic and LDAP Authentication Providers" on page 5-9

Requirements for Using an LDAP Authentication Provider

If an LDAP Authentication provider is the only configured Authentication provider for a security realm, you must have the `Admin` role to boot WebLogic Server and use a user or group in the LDAP directory. Do one of the following in the LDAP directory:

- By default in WebLogic Server, the `Admin` role includes the `Administrators` group. Create an `Administrators` group in the LDAP directory. Make sure the LDAP user who will boot WebLogic Server is included in the group.

The Active Directory LDAP directory has a default group called `Administrators`. Add the user who will be booting WebLogic Server to the `Administrators` group and define Group Base Distinguished Name (DN) so that the `Administrators` group is found.

- If you do not want to create an `Administrators` group in the LDAP directory (for example, because the LDAP directory uses the `Administrators` group for a different purpose), create a new group (or use an existing group) in the LDAP directory and include the user from which you want to boot WebLogic Server in that group. In the WebLogic Administration Console, assign that group the `Admin` role.

Accessing Other LDAP Servers

The LDAP Authentication providers in this release of WebLogic Server are configured to work readily with the SunONE (iPlanet), Active Directory, Open LDAP, and Novell NDS LDAP servers. You can use an LDAP Authentication provider to access other types of LDAP servers. Choose either the LDAP Authentication provider (`LDAPAuthenticator`) or the existing LDAP provider that most closely matches the new LDAP server and customize the existing configuration to match the directory schema and other attributes for your LDAP server.

Configuring Failover for LDAP Authentication Providers

You can configure an LDAP provider to work with multiple LDAP servers and enable failover if one LDAP server is not available. Use the Host attribute (found in the Administration Console on the Configuration: Provider Specific page for the LDAP Authentication provider) to specify the names of the additional LDAP servers. Each host name may include a trailing comma and a port number. In addition, set the Parallel Connect Delay and Connection Timeout attributes for the LDAP Authentication provider:

- Parallel Connect Delay—Specifies the number of seconds to delay when making concurrent attempts to connect to multiple servers. An attempt is made to connect to the first server in the list. The next entry in the list is tried only if the attempt to connect to the current host fails. This setting might cause your application to block for an unacceptably long time if a host is down. If the value is greater than 0, another connection setup thread is started after the specified number of delay seconds has passed. If the value is 0, connection attempts are serialized.
- Connection Timeout—Specifies the maximum number of seconds to wait for the connection to the LDAP server to be established. If the set to 0, there is no maximum time limit and WebLogic Server waits until the TCP/IP layer times out to return a connection failure. Set to a value over 60 seconds depending upon the configuration of TCP/IP.

The following examples present scenarios that occur when an LDAP Authentication provider is configured for LDAP failover.

LDAP Failover Example 1

In the following scenario, an LDAP Authentication provider is configured with three servers in its Host attribute: `directory.knowledge.com:1050`, `people.catalog.com`, and `199.254.1.2`. The status of the LDAP servers is as follows:

- `directory.knowledge.com:1050` is down
- `people.catalog.com` is up
- `199.254.1.2` is up

WebLogic Server attempts to connect to `directory.knowledge.com`. After 10 seconds, the connect attempt times out and WebLogic Server attempts to connect to the next specified host (`people.catalog.com`). WebLogic Server then uses `people.catalog.com` as the LDAP Server for this connection.

LDAP Option	Value

Host	`directory.knowledge.com:1050`
	`people.catalog.com`
	`199.254.1.2`

Parallel Connect Delay	0
Connect Timeout	10

LDAP Failover Example 2

In the following scenario, WebLogic Server attempts to connect to `directory.knowledge.com`. After 1 second (specified by the Parallel Connect Delay attribute), the connect attempt times out and WebLogic Server tries to connect to the next specified host (`people.catalog.com`) and `directory.knowledge.com` at the same time. If the connection to `people.catalog.com` succeeds, WebLogic Server uses `people.catalog.com` as the LDAP Server for this connection.WebLogic Server cancels the connection to `directory.knowledge.com` after the connection to `people.catalog.com` succeeds.

LDAP Option	Value

Host	`directory.knowledge.com:1050`
	`people.catalog.com`
	`199.254.1.2`

Parallel Connect Delay	1
Connect Timeout	10

Improving the Performance of WebLogic and LDAP Authentication Providers

You can improve the performance of the WebLogic and LDAP Authentication providers in the following ways:

- Optimize the configuration the group membership caches used by the WebLogic and LDAP Authentication providers.
- Expose the Principal Validator cache for the security realm and increase its thresholds.
- If you are using the Active Directory Authentication provider, configure it to perform group membership lookups using the `tokenGroups` option. The `tokenGroups` option holds the entire flattened group membership for a user as an array of system ID (SID) values. The SID values are specially indexed in the Active Directory and yield extremely fast lookup response.

The following sections describe how to implement these optimizations.

Optimizing the Group Membership Caches

To optimize the group membership caches for WebLogic and LDAP Authentication providers, set the following attributes (found in the Administration Console on the LDAP Authentication provider's Configuration: Provider Specific and Performance pages):

- Group Membership Searching—Controls whether group searches are limited or unlimited in depth. This option controls how deeply a search should recursive into nested groups. For configurations that use only the first level of nested group hierarchy, this option allows improved performance during user searches by limiting the search to the first level of the group.

  - If a limited search is defined, Max Group Membership Search Level must be defined.
  - If an unlimited search is defined, Max Group Membership Search Level is ignored.
- Max Group Membership Search Level—Controls the depth of a group membership search if Group Membership Searching is defined. Possible values are:
  - 0—Indicates only direct groups will be found. That is, when searching for membership in Group A, only direct members of Group A will be found. If Group B is a member of Group A, the members will not be found by this search.
  - Any positive number—Indicates the number of levels to search. For example, if this option is set to 1, a search for membership in Group A will return direct members of Group A. If Group B is a member of Group A, the members of Group B will also be found by this search. However, if Group C is a member of Group B, the members of Group C will not be found by this search.
- Enable Group Membership Lookup Hierarchy Caching—Indicates whether group membership hierarchies found during recursive membership lookup are cached. Each subtree found will be cached. The cache holds the groups to which a group is a member. This setting only applies if Group Membership is enabled. By default, it is disabled.
- Max Group Hierarchies in Cache—The maximum size of the Least Recently Used (LRU) cache that holds group membership hierarchies. This setting only applies if Enable Group Membership Lookup Hierarchy Caching is enabled.
- Group Hierarchy Cache TTL—The number of seconds cached entries stay in the cache. The default is 60 seconds.

Configuring Dynamic Groups in the iPlanet Authentication Provider to Improve Performance

Dynamic groups do not list the names of their members. Instead, the membership of the dynamic group is constructed by matching user attributes. Since group membership needs to be computed dynamically for dynamic groups, there is a risk of performance problems for large groups. Configuring the iPlanet Authentication provider appropriately can improve performance where dynamic groups are involved.

In the iPlanet Authentication provider, the User Dynamic Group DN Attribute attribute specifies the attribute of an LDAP user object that specifies the distinguished names (DNs) of dynamic groups to which this user belongs. If such an attribute does not exist, WebLogic Server determines if a user is a member of a group by evaluating the URLs on the dynamic group. By default, User Dynamic Group DN Attribute is null. If you set User Dynamic Group DN Attribute to some other value, to improve performance set the following attributes for the iPlanet Authentication provider:

```
UserDynamicGroupDNAttribute="wlsMemberOf"

DynamicGroupNameAttribute="cn"

DynamicGroupObjectClass=""

DynamicMemberURLAttribute=""
```

To set these attributes in the Administration Console:

1. Expand Security Realms-->*realm name*-->Providers-->Authentication.
2. On the Provider Specific tab for your iPlanet Authentication provider, set User Dynamic Group DN Attribute. Set Dynamic Group Object Class and Dynamic Member URL Attribute to null (delete anything in the fields) and leave Dynamic Group Name Attribute set to `cn`.

Optimizing the Principal Validator Cache

To improve the performance of a WebLogic or LDAP Authentication provider, the settings of the cache used by the WebLogic Principal Validation provider can be increased as appropriate. The Principal Validator cache used by the WebLogic Principal Validation provider caches signed WLSAbstractPrincipals. To optimize the performance of the Principal Validator cache, set these attributes for your security realm (found in the Administration Console on the Configuration: Performance page for the security realm):

- Enable WebLogic Principal Validator Cache—Indicates whether the WebLogic Principal Validation provider uses a cache. This setting only applies if Authentication providers in the security realm use the WebLogic Principal Validation provider and WLSAbstractPrincipals. By default, it is enabled.
- Max WebLogic Principals In Cache—The maximum size of the Last Recently Used (LRU) cache used for validated WLSAbstractPrincipals. The default setting is 500. This setting only applies if Enable WebLogic Principal Validator Cache is enabled.

Configuring the Active Directory Authentication Provider to Improve Performance

To configure an Active Directory Authentication provider to use the `tokenGroups` option, set the following attributes (found in the Administration Console on the Active Directory Authentication provider's Configuration: Provider Specific page):

- Use Token Groups for Group Membership Lookup—Indicates whether to use the Active Directory `tokenGroups` lookup algorithm instead of the standard recursive group membership lookup algorithm. By default, this option is not enabled.

  Note: Access to the `tokenGroups` option is required (meaning, the user accessing the LDAP directory must have privileges to read the `tokenGroups` option and the `tokenGroups` option must be in the schema for user objects).

- Enable SID to Group Lookup Caching—Indicates whether or not SID-to-group name lookup results are cached. This setting only applies if the Use Token Groups for Group Membership Lookup option is enabled.
- Max SID To Group Lookups In Cache—The maximum size of the Least Recently Used (LRU) cache for holding SID to group lookups. This setting applies only if both the Use Token Groups for Group Membership Lookup and Enable SID to Group Lookup Caching options are enabled.

Configuring RDBMS Authentication Providers

In WebLogic Server, an RDBMS Authentication provider is a username/password based Authentication provider that uses a relational database (rather than an LDAP directory) as its data store for user, password, and group information. WebLogic Server includes these RDBMS Authentication providers:

- SQL Authenticator—Uses a SQL database and allows both read and write access to the database. This Authentication provider is configured by default with a typical SQL database schema, which you can configure to match your database's schema. See "Configuring the SQL Authenticator" on page 5-13.
- Read-only SQL Authenticator—Uses a SQL database and allows only read access to the database. For write access, you use the SQL database's own interface, not the WebLogic security provider. See "Configuring the Read-Only SQL Authenticator" on page 5-14.
- Custom RDBMS Authenticator—Requires you to write a plug-in class. This may be a better choice if you want to use a relational database for your authentication data store, but the SQL Authenticator's schema configuration is not a good match for your existing database schema. See "Configuring the Custom DBMS Authenticator" on page 5-14.

For information about adding an RDBMS Authentication provider to your security realm, see Configure Authentication and Identity Assertion providers in the Administration Console help. Once you have created an instance of the RDBMS Authentication provider, configure it, using the RDBMS Authentication provider's Configuration: Provider Specific page in the Administration Console.

Common RDBMS Authentication Provider Attributes

Each of the three types of RDBMS Authentication provider include these configuration options:

Data Source Attribute

The Data Source Name specifies the WebLogic Server data source to use to connect to the database.

Group Searching Attributes

The Group Membership Searching and Max Group Membership Search Level attributes specify whether recursive group membership searching is unlimited or limited, and if limited, how many levels of group membership can be searched. For example, if you specify that Group Membership Searching is LIMITED, and the Max Group Membership Search Level is 0, then the RDBMS Authentication providers will find only groups that the user is a direct member of. Specifying a maximum group membership search level can greatly increase authentication performance in certain scenarios, since it may reduce the number of DBMS queries executed during authentication. However, you should only limit group membership search if you can be certain that the group memberships you require are within the search level limits you specify.

Group Caching Attributes

You can improve the performance of RDBMS Authentication providers by caching the results of group hierarchy lookups. Use of this cache can reduce the frequency with which the RDBMS Authentication provider needs to access the database. In the Administration Console, you can use the Performance page for your Authentication provider to configure the use, size, and duration of this cache. For detailed information, see Security Realms: Security Providers: SQL Authenticator: Performance in the Administration Console online help.

Configuring the SQL Authenticator

For detailed information configuring a SQL Authentication provider, see Security Realms: Security Providers: SQL Authenticator: Provider Specific in the Administration Console online help. In addition to the attributes described in "Common RDBMS Authentication Provider Attributes" on page 5-13, the SQL Authentication provider's configurable attributes include:

Password Attributes

The following attributes govern how the RDBMS Authentication provider and its underlying database handle user passwords:

- Plaintext Passwords Enabled
- Password Style Retained
- Password Style
- Password Algorithm

SQL Statement Attributes

The remaining attributes specify the SQL statements used by the provider to access and edit the username, password, and group information in the database. The default values in the SQL statement attributes assume that the database schema includes the following tables:

- users (username, password, [description])
- groupmembers (group name, group member)
- groups (group name, group description)

Note that the tables referenced by the SQL statements must exist in the database; the provider will not create them. You can modify these attributes as needed to match the schema of your database. However, if your database schema is radically different from this default schema, you may need to use a Custom DBMS Authentication provider instead.

Configuring the Read-Only SQL Authenticator

For detailed information configuring a Read-Only SQL Authentication provider, see Security Realms: Security Providers: Read-Only SQL Authenticator: Provider Specific in the Administration Console online help. In addition to the attributes described in "Common RDBMS Authentication Provider Attributes" on page 5-13, the Read-Only SQL Authentication provider's configurable attributes include attributes that specify the SQL statements used by the provider to list the username, password, and group information in the database. You can modify these attributes as needed to match the schema of your database.

Configuring the Custom DBMS Authenticator

The Custom DBMS Authentication provider, like the other RDBMS Authentication providers, uses a relational database as its data store for user, password, and group information. Use this provider if your database schema does not map well to the SQL schema expected by the SQL Authenticator. In addition to the attributes described in "Common RDBMS Authentication

Provider Attributes" on page 5-13, the Custom DBMS Authentication provider's configurable attributes include:

Plug-In Class Attributes

A Custom DBMS Authentication provider requires that you write a plug-in class that implements the `weblogic.security.providers.authentication.CustomDBMSAuthenticatorPlugin` interface. The class must exist in the CLASSPATH and must be specified in the Plugin Class Name attribute for the Custom DBMS Authentication provider. Optionally, you can use the Plugin Properties attribute to specify values for properties defined by your plug-in class.

Configuring a Windows NT Authentication Provider

The Windows NT Authentication provider uses account information defined for a Windows NT domain to authenticate users and groups and permit Windows NT users and groups to be listed in the WebLogic Administration Console.

To use the Windows NT Authentication provider, create the provider in the WebLogic Administration Console. In most cases, you should not need to do anything more to configure this Authentication provider. Depending on how your Windows NT domains are configured, you may want to set the attributes that control how the Windows NT Authentication provider interacts with the Windows NT domain.

Domain Controller Settings

Usernames in a Windows NT domain can take several different forms. You may need to configure the Windows NT Authentication provider to match the form of usernames you expect your users to sign on with. A simple username is one that gives no indication of the domain, such as `smith`. Compound usernames combine a username with a domain name and may take a form like `domain\smith` or `smith@domain`.

If the local machine is not part of a Microsoft domain, then no changes to the domain controller list are needed. On a standalone machine, the users and groups to be authenticated are defined only on that machine.

If the local machine is part of a Microsoft domain and is the domain controller for the local domain, then no changes are needed to the domain controller list are needed. Users defined on the local machine and the domain are the same in this case, so you can use the default domain controller setting.

If the local machine is part of a Microsoft domain, but is not the domain controller for the local domain, then a simple username might be found on either the local machine or in the domain. In this case, consider the following:

- Do you want the users and groups from the local machine to not be displayed in the console when the local machine is part of a domain?
- Do you want users from the local machine to be found and authenticated when a simple username is entered?

If the answer to either question is yes, then set the Domain Controller attribute to `DOMAIN`.

If you have multiple trusted domains, you may need to set the Domain Controller attribute to `DOMAIN` and specify a Domain Controller List. Do this if:

- you require the users and groups for other trusted domains to be visible in the console, or
- you expect that your users will be entering simple usernames and expect them to be located in the trusted domains? (that is, users will sign on with a simple username like enter `smith`, not `smith@domain` or `domain\Smith`).

If you answer yes to either question, then set the `DomainControllers` attribute to `LIST` and specify the names of the domain controllers in the `DomainControllerList` for the trusted domains that you want to be used. Consider also whether to use explicit names for the local machine and local domain controller or if you want to use placeholders in the list for those. You can use the following placeholders:

- [Local]
- [LocalAndDomain]
- [Domain]

LogonType Setting

The proper value of the `LogonType` attribute depends on the Windows NT logon rights of the users that you want to be able to authenticate:

- If users have the “logon locally” right assigned to them on the machines that will run WebLogic Server, then use the default value, `interactive`.
- If users have the “Access this computer from the Network” right assigned to them, then change the LogonType attribute to `network`.

You must assign one of these rights to users in the Windows NT domain or else the Windows NT Authentication provider will not be able to authenticate any users.

UPN Names Settings

UPN style usernames can take the form `user@domain`. You can configure how the Windows NT Authentication provider handles usernames that include the @ character, but which may not be UPN names, by setting the `mapUPNNames` attribute.

If none of your Windows NT domains or local machines have usernames that contain the @ character other than UPN usernames, then you can use the default value of the `mapUPNNames` attribute, FIRST. However, you may want to consider changing the setting to ALWAYS in order to reduce the amount of time it takes to detect authentication failures. This is especially true if you have specified a long domain controller list.

If your Windows NT domains do permit non-UPN usernames with the @ character in them, then:

- if a username with the @ character is more likely to be a UPN username than a simple username, set the `mapUPNNames` attribute to FIRST.
- if a username with the @ character is more likely to be a simple username than a UPN username, set the `mapUPNNames` attribute to LAST.
- if a username is never in UPN format, set the `mapUPNNames` attribute to NEVER.

Configuring Identity Assertion Providers

If you are using perimeter authentication, you need to use an Identity Assertion provider. In perimeter authentication, a system outside of WebLogic Server establishes trust through tokens (as opposed to simple authentication, where WebLogic Server establishes trust through usernames and passwords). An Identity Assertion provider verifies the tokens and performs whatever actions are necessary to establish validity and trust in the token. Each Identity Assertion provider is designed to support one or more token formats.

WebLogic Server includes the following Identity Assertion providers:

- WebLogic Identity Asserter
- LDAP X.509 Identity Asserter
- Negotiate Identity Asserter
- SAML Identity Asserter

Multiple Identity Assertion providers can be configured in a security realm, but none are required. Identity Assertion providers can support more than one token type, but only one token type per Identity Assertion provider can be active at a given time. In the Active Type field on the General page, define the active token type. The WebLogic Identity Assertion provider supports identity assertion with X.509 certificates and CORBA Common Secure Interoperability version 2 (CSI v2). If you are using CSI v2 identity assertion, define the list of client principals in the Trusted Principals field.

If multiple Identity Assertion providers are configured in a security realm, they can all support the same token type. However, the token can be active for only one only provider at a time.

When using the WebLogic Identity Assertion provider in a security realm, you can use a user name mapper to map the tokens authenticated by the Identity Assertion provider to a user in the security realm. For more information about configuring a user name mapper, see "Configuring a WebLogic Credential Mapping Provider" on page 4-15.

If the authentication type in a Web application is set to `CLIENT-CERT`, the Web Application container in WebLogic Server performs identity assertion on values from request headers and cookies. If the header name or cookie name matches the active token type for the configured Identity Assertion provider, the value is passed to the provider.

The Base64 Decoding Required value on the Details page determines whether the request header value or cookie value must be Base64 Decoded before sending it to the Identity Assertion provider. The setting is enabled by default for purposes of backward compatibility; however, most Identity Assertion providers will disable this option.

For more information see Configure Authentication and Identity Assertion providers in the Administration Console online help. In addition, see the following sections:

- "How an LDAP X509 Identity Assertion Provider Works" on page 5-19
- "Configuring an LDAP X509 Identity Assertion Provider:Main Steps" on page 5-19
- "Configuring a Negotiate Identity Assertion Provider" on page 5-21
- "Configuring a SAML Identity Assertion Provider" on page 5-21
- "Ordering of Identity Assertion for Servlets" on page 5-26
- "Configuring Identity Assertion Performance in the Server Cache" on page 5-27
- "Configuring a User Name Mapper" on page 5-28
- "Configuring a Custom User Name Mapper" on page 5-29

How an LDAP X509 Identity Assertion Provider Works

The LDAP X509 Identity Assertion provider receives an X509 certificate, looks up the LDAP object for the user associated with that certificate, ensures that the certificate in the LDAP object matches the presented certificate, and then retrieves the name of the user from the LDAP object.

The LDAP X509 Identity Assertion provider works in the following manner:

1. An application is set up to use perimeter authentication (in other words, users or system process use tokens to assert their identity).
2. As part of the SSL handshake, the application presents it certificate. The Subject DN in the certificate can be used to locate the object that represents the user in the LDAP server. The object contains the user's certificate and name.
3. The LDAP X509 Identity Assertion provider uses the certificate in the Subject DN to construct an LDAP search to find the LDAP object for the user in the LDAP server. It gets the certificate from that object, ensures it matches the certificate it holds, and retrieves the name of the user.
4. The username is passed to the authentication providers configured in the security realm. The authentication providers ensure the user exists and locates the groups to which the user belongs.

Configuring an LDAP X509 Identity Assertion Provider:Main Steps

Typically, if you use the LDAP X509 Identity Assertion provider, you also need to configure an LDAP Authentication provider that uses an LDAP server. The authentication provider ensures the user exists and locates the groups to which the user belongs. You should ensure both providers are properly configured to communicate with the same LDAP server.

To use an LDAP X509 Identity Assertion provider:

1. Obtain certificates for users and putting them in an LDAP Server. See Chapter 10, "Configuring Identity and Trust."

   There must be a correlation between the Subject DN in the certificate and the location of the object for that user in the LDAP server. The LDAP object for the user must also include configuration information for the certificate and the username that will be used in the Subject.

2. In your security realm, configure an LDAP X509 Identity Assertion provider. See Configure Authentication and Identity Assertion providers in the Administration Console help.
3. In the WebLogic Administration Console, configure the LDAP X509 Identity Assertion provider to find the LDAP object for the user in the LDAP directory given the certificate's Subject DN.
4. Configure the LDAP X509 Identity Assertion provider to search the LDAP server to locate the LDAP object for the user. This requires the following pieces of data.
   - A base LDAP DN from which to start searching. The Certificate Mapping option for the LDAP X509 Identity Assertion provider tells the identity assertion provider how to construct the base LDAP DN from the certificate's Subject DN. The LDAP object must contain an attribute that holds the certificate.
   - A search filter that only returns LDAP objects that match a defined set of options. The filter narrows the LDAP search. Configure User Filter Search to construct a search filter from the certificate's Subject DN.
   - Where in the LDAP directory to search for the base LDAP DN. The LDAP X509 Identity Assertion provider searches recursively (one level down). This value must match the values in the certificate's Subject DN.
5. Configure the Certificate Attribute attribute of the LDAP X509 Identity Assertion provider to specify how the LDAP object for the user holds the certificate. The LDAP object must contain an attribute the holds the certificate.
6. Configure the User Name Attribute attribute of the LDAP X509 Identity Assertion provider to specify which of the LDAP object's attributes holds the username that should appear in the Subject DN.
7. Configure the LDAP server connection for the LDAP X509 Identity Assertion provider. The LDAP server information should be the same as the information defined for the LDAP Authentication provider configured in this security realm.
8. Configure an LDAP Authentication provider for use with the LDAP X509 Identity Assertion provider. The LDAP server information should be the same the information defined for the LDAP X509 Identity Assertion provider configured in Step 7. See "Configuring LDAP Authentication Providers" on page 5-5.

Configuring a Negotiate Identity Assertion Provider

The Negotiate Identity Assertion provider enables single sign-on (SSO) with Microsoft clients. The identity assertion provider decodes Simple and Protected Negotiate (SPNEGO) tokens to obtain Kerberos tokens, validates the Kerberos tokens, and maps Kerberos tokens to WebLogic users. The Negotiate Identity Assertion provider utilizes the Java Generic Security Service (GSS) Application Programming Interface (API) to accept the GSS security context via Kerberos.

The Negotiate Identity Assertion provider is an implementation of the Security Service Provider Interface (SSPI) as defined by the WebLogic Security Framework and provides the necessary logic to authenticate a client based on the client's SPNEGO token.

For information about adding a Negotiate Identity Assertion provider to a security realm, see Configure Authentication and Identity Assertion providers in the Administration Console help. For information about using the Negotiate Identity Assertion provider with Microsoft client SSO, see "Configuring Single Sign-On with Microsoft Clients" on page 6-1.

Table 5-1 Negotiate Identity Asserter Attributes

Attribute	Description
Form Based Negotiation Enabled	Indicates whether the Negotiate Identity Assertion provider and servlet filter should negotiate when a Web application is configured for FORM authentication.
Active Types	The token type this Negotiate Identity Assertion provider uses for authentication. Available token types are `Authorization.Negotiate` and `WWW-Authenticate.Negotiate`. Ensure no other identity assertion provider configured in the same security realm has this attribute set to X509.

Configuring a SAML Identity Assertion Provider

The SAML Identity Assertion provider acts as a consumer of SAML security assertions, allowing WebLogic Server to act as a destination site for using SAML for single sign-on. The SAML Identity Assertion provider validates SAML 1.1 assertions by checking the signature and validating the certificate for trust in the certificate registry maintained by the provider. If so, identity is asserted based on the `AuthenticationStatement` contained in the assertion. The SAML Identity Assertion provider can be configured as a SAML Assertion Consumer Service.

The SAML Identity Assertion provider can also ensure that the assertion has not been previously used.

This section includes information about configuring the following aspects of the SAML Identity Assertion provider:

- "POST and ARTIFACT Profiles" on page 5-22
- "SAML Destination Site Configuration" on page 5-22
- "Limiting the Re-use of Assertions" on page 5-23
- "Certificate Registry" on page 5-23
- "Consumed Assertion Configuration" on page 5-24

For information about how to use the SAML Identity Assertion provider in a SAML single sign-on configuration, see "Configuring Single Sign-On with Web Browsers and HTTP Clients" on page 7-1. For general information about SAML support in WebLogic Server, see Security Assertion Markup Language (SAML) in *Understanding WebLogic Security*.

POST and ARTIFACT Profiles

Configure the SAML Identity Assertion provider to support POST profile, Artifact profile, or both.

To configure POST profile support:

1. Set the Post Enabled attribute.
2. Optionally, set the Recipient Check Enabled attribute. If true, this attribute requires that the recipient of the SAML Response must match the URL in the HTTP Request.
3. Optionally, set the Enforce One Use Policy attribute. See "Limiting the Re-use of Assertions" on page 5-23.

To configure Artifact profile support, set the Artifact Enabled attribute.

SAML Destination Site Configuration

The Assertion Consumer URIs attribute lists the URIs at which the SAML Identity Assertion provider listens for incoming assertions. These URIs provide the Assertion Consumer Service (ACS). The SAML destination site can also be configured to redirect unauthenticated users to remote SAML source sites for authentication based on the particular URI they tried to access. Configure these SAML source sites in the SAML Identity Assertion provider's Source Site

Redirects attribute. SAML Source Site Redirects are configured with Java-style properties, in key=value format, as in the example in Listing 5-1.

Listing 5-1 Example of Source Site Redirects Configuration

```
Redirects=mypost,myart

mypost.TargetURI=/targets/PostTarget.jsp

mypost.RedirectURL=https://www.source_site.com:7002/samlits_ba/post

myart.TargetURI=/targets/ArtTarget.jsp

myart.RedirectURL=https://www.source_site.com:7002/samlits_ba/artifact
```

Limiting the Re-use of Assertions

You can configure the SAML Identity Assertion provider to limit assertions to a single use only. To do this in the Administration Console:

1. Open the Provider Specific configuration page for the SAML Identity Assertion provider.
2. Set the Enforce One Use Policy attribute.
3. In order to track assertions that have already been used, the SAML Identity Assertion provider maintains a cache of used assertions. If you don't want to use the default class for this cache, specify the name of the custom Used Assertion Cache class to use. This class must implement `weblogic.security.providers.saml.SAMLUsedAssertionCache`.
4. If you are using a custom Used Assertion Cache class, you may be able to use the Used Assertion Cache Properties field to configure this cache. These properties, if provided, will be passed to the `initCache()` method of your custom Used Assertion Cache class.

Certificate Registry

The SAML Identity Assertion provider maintains a registry of trusted certificates. Whenever a certificate is received, it is checked against the certificates in the registry for validity. The certificates in this registry are used:

- To check trust when the SAML Identity Assertion provider receives a signed assertion.
- To check trust in the source site that sent a POST profile assertion (a signed SAML response object).

- To check trust in a destination site that is retrieving an artifact from the ARS, if the connection is an SSL connection.

When signing assertions or POST profile response objects, or contacting another site's ARS over SSL, we use the server's SSL certificate for signing and client authentication.

You can add trusted certificates to the certificate registry using the Administration Console:

1. In the console, navigate to the Security Realms > *your realm* > Providers > Authentication page.
2. Click the name of the SAML Identity Assertion provider and open the Management page

On the Management page, you can add, view, or delete certificates from the registry.

Consumed Assertion Configuration

You can configure how assertions will be consumed in the Assertion Configuration attribute of the SAML Identity Assertion provider. You configure assertions with Java-style properties, in key=value format. The key takes the form `assertionName.propertyName`.

Table 5-2 Consumed Assertion Configuration Properties

Property	Description
Assertions	The names assigned to the assertions in this configuration.
AssertionType	One of: `bearer`, `artifact`, `sender-vouches`, `holder-of-key`.
Target	For `bearer` and `artifact` assertions, the destination site URL for which authentication is desired. The Identity Asserter uses the target URL, in conjunction with the assertion type and issuer URL, to identify the assertion configuration against which the assertion should be validated. This attribute is ignored for `sender-vouches` and `holder-of-key` assertions.
IssuerURI	The issuer for this assertion. Used in conjunction with the Assertion Type and Target URL to identify the assertion configuration against which the assertion should be validated.

Table 5-2 Consumed Assertion Configuration Properties

Property	Description
SourceIdHex SourceIdBase64	Source site `SourceID`, in either hexadecimal or Base64 encoding. Used by Artifact profile, in conjunction with the Retrieval URL, to fetch the assertion corresponding to an Artifact.
RetrievalURL	Used by Artifact profile to fetch assertions for this Source ID.
AudienceURI	If this attribute is present, then the specified Audience URI must be present in the assertion.
Signed	If true, assertion must be signed. If false, signature elements will be ignored.
TrustedSender	If true, a SSL client certificate chain or message signer certificate must be provided when calling the Identity Asserter. Used with `sender-vouches` assertions. Do not use with `holder-of-key` assertions.
NameMapperClass	If this attribute is present, then the Identity Asserter will use the specified Name Mapper class instead of the default Name Mapper.
ProcessGroups	If `true`, presence of an Attribute Statement containing WebLogic Server groups is required. If `false`, Attribute Statements will not be processed.

Example of Consumed Assertion Configuration

Listing 5-2 is an example of how you might configure produced assertions. It includes both a POST profile assertion, named `mypost`, and an Artifact profile assertion, named `myart`.

Listing 5-2 Consumed Assertion Configuration

```
Assertions=mypost,myart

mypost.AssertionType=bearer

mypost.Target=http://www.destination_site.com:7001/targets/PostTarget.jsp
mypost.IssuerURI=http://www.source_site.com/issuer

mypost.ProcessGroups=true

myart.AssertionType=artifact

myart.Target=http://www.destination_site.com:7001/targets/Art*

myart.IssuerURI=http://www.source_site.com/issuer

myart.SourceIdHex=63A678A290A3DBADBA9F1E51135E225049FA5BB0

myart.RetrievalURL=https://www.source_site.com:7003/samlars/ars
```

Ordering of Identity Assertion for Servlets

When an HTTP request is sent, there may be multiple matches that can be used for identity assertion. However, identity assertion providers can only consume one of the active token types at a time. As a result there is no way to provide a set of tokens that can be consumed with one call. Therefore, the servlet contained in WebLogic Server is forced to choose between multiple tokens to perform identity assertion. The following ordering is used:

1. An X.590 digital certificate (signifies two-way SSL to client or proxy plug-in with two-way SSL between the client and the Web server) if X.509 is one of the active token types configured for the Identity Assertion provider in the default security realm.
2. Headers with a name in the form `WL-Proxy-Client-<`*`TOKEN`*`>` where <*`TOKEN`*> is one of the active token types configured for the Identity Assertion provider in the default security realm.

   Note: This method is deprecated and should only be used for the purpose of backward compatibility.
3. Headers with a name in the form <*`TOKEN`*> where <*`TOKEN`*> is one of the active tokens types configured for the Identity Assertion provider in the default security realm.
4. Cookies with a name in the form <*`TOKEN`*> where <*`TOKEN`*> is one of the active tokens types configured for the Identity Assertion provider in the default security realm.

For example, if an Identity Assertion provider in the default security realm is configured to have the `FOO` and `BAR` tokens as active token types (for the following example, assume the HTTP request contains nothing relevant to identity assertion except active token types), identity assertion is performed as follows:

- If a request comes in with a `FOO` header over a two-way SSL connection, X.509 is used for identity assertion.

- If a request comes in with a `FOO` header and a `WL-Proxy-Client-BAR` header, the `BAR` token is used for identity assertion.
- If a request comes in with a `FOO` header and a `BAR` cookie, the `FOO` token will be used for identity assertion.

The ordering between multiple tokens at the same level is undefined, therefore:

- If a request comes in with a `FOO` header and a `BAR` header, then either the `FOO` or `BAR` token is used for identity assertion, however, which one is used is unspecified.
- If a request comes in with a `FOO` cookie and a `BAR` cookie, then either the `FOO` or `BAR` token is used for identity assertion, however, which one is used is unspecified.

Configuring Identity Assertion Performance in the Server Cache

When using an Identity Assertion provider (either for an X.509 certificate or some other type of token), Subjects (a grouping of related information for a single entity including an identity and its security-related configuration options) are cached within the server. This greatly enhances performance for servlets and EJB methods with `<run-as>` tags as well as for other places where identity assertion is used but not cached in the HTTPSession (for example, signing and encrypting XML documents).

Note: Caching can violate the desired semantics.

You can change the lifetime of items in this cache by setting the maximum number of seconds a Subject can live in the cache via the `-Dweblogic.security.identityAssertionTTL` command-line argument. The default for this command-line argument is 300 seconds (that is, 5 minutes). Possible values for the command-line argument are:

- Less than 0—Disables the cache.
- 0—Caching is enabled and the identities in the cache never time out so long as the server is running. Any changes in the user database of cached entities requires a server reboot in order for the server to pick them up.
- Greater than 0—Caching is enabled and the cache is reset at the specified number of seconds.

To improve the performance of identity assertion, specify a higher value for this command-line argument. Note that as identity assertion performance improves, the Identity Assertion provider is less responsive to changes in the configured Authentication provider. For example, a change in the user's group will not be reflected until the Subject is flushed from the cache and recreated. Setting a lower value for the command-line argument makes authentication changes more responsive at a cost for performance.

Configuring a User Name Mapper

WebLogic Server verifies the digital certificate of the Web browser or Java client when establishing a two-way SSL connection. However, the digital certificate does not identify the Web browser or Java client as a user in the WebLogic Server security realm. If the Web browser or Java client requests a WebLogic Server resource protected by a security policy, WebLogic Server requires the Web browser or Java client to have an identity. The WebLogic Identity Assertion provider allows you to enable a user name mapper that maps the digital certificate of a Web browser or Java client to a user in a WebLogic Server security realm.

The user name mapper must be an implementation of the `weblogic.security.providers.authentication.UserNameMapper` interface. This interface maps a token to a WebLogic Server user name according to whatever scheme is appropriate for your needs. By default, WebLogic Server provides a default implementation of the `weblogic.security.providers.authentication.UserNameMapper` interface. You can also write your own implementation.

The WebLogic Identity Assertion provider calls the user name mapper for the following types of identity assertion token types:

- X.509 digital certificates passed via the SSL handshake
- X.509 digital certificates passed via CSIv2
- X.501 distinguished names passed via CSIv2

The default user name mapper uses the subject DN of the digital certificate or the distinguished name to map to the appropriate user in the WebLogic Server security realm. For example, the user name mapper can be configured to map a user from the Email attribute of the subject DN (`smith@example.com`) to a user in the WebLogic Server security realm (`smith`). Use Default User Name Mapper Attribute Type and Default Username Mapper Attribute Delimiter attributes of the WebLogic Identity Assertion provider to define this information:

- Default User Name Mapper Attribute Type—The subject distinguished name (DN) in a digital certificate used to create a username. Valid values are: `C`, `CN`, `E`, `L`, `O`, `OU`, `S and STREET`.

- Default User Name Mapper Attribute Delimiter—Ends the username. The user name mapper uses everything to the left of the value to create a username. The default delimiter is @.

For more information, see "Configure a user name mapper" in the Administration Console online help.

Configuring a Custom User Name Mapper

You can also write a custom user name mapper to map a token to a WebLogic Server user name according to whatever scheme is appropriate for your needs. The custom user name mapper must be an implementation of the `weblogic.security.providers.authentication.UserNameMapper` interface. You then configure the custom user name mapper in the active security realm, using the User Name Mapper Class Name attribute of the WebLogic Identity Assertion provider.

For more information, see "Configure custom user name mappers" in the Administration Console online help.

Configuring Single Sign-On with Microsoft Clients

This section explains how to set up single sign-on (SSO) with Microsoft clients, using Windows authentication based on the Simple and Protected Negotiate (SPNEGO) mechanism and the Kerberos protocol, together with the WebLogic Negotiate Identity Assertion provider.

Single Sign-on with Microsoft Clients: Main Steps

Single sign-on (SSO) with Microsoft clients allows cross-platform authentication between Web applications or Web Services running in a WebLogic Server domain and .NET Web Service clients or browser clients (for example, Internet Explorer) in a Microsoft domain. The Microsoft clients must use Windows authentication based on the Simple and Protected Negotiate (SPNEGO) mechanism.

Cross-platform authentication is achieved by emulating the negotiate behavior of native Windows-to-Windows authentication services that use the Kerberos protocol. In order for cross-platform authentication to work, non-Windows servers (in this case, WebLogic Server) need to parse SPNEGO tokens in order to extract Kerberos tokens which are then used for authentication.

Configuring SSO with Microsoft clients involves set up procedures in the Microsoft Active Directory, the client, and the WebLogic Server domains.

- The Kerberos protocol uses the Active Directory server in the Microsoft domain to store the necessary security information. Therefore, you need to define a principal in Active Directory to represent the WebLogic Server.
- Any Microsoft client you want to access in the Microsoft domain must be set up to use Windows Integrated authentication, sending a Kerberos ticket when available.
- In the WebLogic Server domain, a Negotiate Identity Assertion provider needs to be configured in the security realm. The Web application or Web Service used in SSO needs to have authentication set in a specific manner. A JAAS login file that defines the location of the Kerberos identification for WebLogic Server must be created.

The main configuration steps are as follows:

1. Configure your network domain to use Kerberos. See
2. Create a Kerberos identification for WebLogic Server.
   a. Create a user account in Active Directory for the host on which WebLogic Server is running.
   b. Create a Service Principal Name for this account.
   c. Create a user mapping and keytab file for this account.

   For more information, see "Creating a Kerberos Identification for WebLogic Server" on page 6-5.
3. Choose a Microsoft client (either a Web Service or a browser) and configure it to use Windows Integrated authentication. For more information, see "Configuring Microsoft Clients to Use Windows Integrated Authentication" on page 6-7.
4. Set up the WebLogic Server domain to use Kerberos authentication.

   a. Create a JAAS login file that points to the Active Directory server in the Microsoft domain and the keytab file created in Step 1. For more information, see "Creating a JAAS Login File" on page 6-9.

   b. Configure a Negotiate Identity Assertion provider in the WebLogic Server security realm. For more information, see "Configuring a Negotiate Identity Assertion Provider" on page 5-21.

5. Start WebLogic Server using specific start-up arguments. For more information, see "Startup Arguments for Using Kerberos Authentication with WebLogic Server" on page 6-10

The following sections describe these steps in detail.

System Requirements for SSO with Microsoft Clients

To use SSO with Microsoft clients, you must comply with these system requirements:

A host computer with:

- Windows 2000 or greater installed
- Fully-configured Active Directory authentication service. Specific Active Directory requirements include:
  - User accounts for mapping Kerberos services
  - Service Principal Names (SPNs) for those accounts
  - Key tab files created and copied to the start-up directory in the WebLogic Server domain
- WebLogic Server installed and configured properly to authenticate via Kerberos, as described in this section, Chapter 6, "Configuring Single Sign-On with Microsoft Clients."

Client systems with:

- Windows 2000 Professional SP2 or greater installed
- One of the following types of clients:
  - A properly configured Internet Explorer browser. Internet Explorer 6.01 or greater is supported.
  - .NET Framework 1.1 and a properly configured Web Service client.

  Clients must be logged on to a Windows 2000 domain and have Kerberos credentials acquired from the Active Directory server in the domain. Local logons will not work.

Configuring your Network Domain to Use Kerberos

A Windows domain controller can server as the Kerberos Key Distribution Center (KDC), using the Active Directory and the Kerberos services. On any domain controller, the Active Directory and the Kerberos services will be running automatically. To configure Kerberos in your Windows domain controller, you need to configure each machine that will access the KDC to locate the Kerberos realm and available KDC servers. For Windows machines, this configuration is in the `krb5.ini` file, in the `C:\winnt` folder. For UNIX machines, this configuration is in the `krb5.conf` file, the default location of which is `/etc/krb5/`. For example:

Listing 6-1 Sample krb5.ini File

```
[libdefaults]
default_realm = MYDOM.COM (Identifies the default realm. Set its value to
your Kerberos realm)
default_tkt_enctypes = des-cbc-crc
default_tgs_enctypes = des-cbc-crc
ticket_lifetime = 600

[realms]

MYDOM.COM = {
kdc = <IP address for MachineA> (host running the KDC)
(For Unix systems, you need to specify port88, as in <IP-address>:88)
admin_server = MachineA
default_domain = MYDOM.COM

[domain_realm]
.mydom.com = MYDOM.COM


[appdefaults]
autologin = true
forward = true
forwardable = true
encrypt = true
```

Creating a Kerberos Identification for WebLogic Server

Active Directory provides support for service principal names (SPN) which are a key component in Kerberos authentication. SPNs are unique identifiers for services running on servers. Every service that uses Kerberos authentication needs to have an SPN set for it so that clients can identify the service on the network. An SPN usually looks something like name@YOUR.REALM. You need to define an SPN to represent your WebLogic Server in the Kerberos realm. If an SPN is not set for a service, clients have no way of locating that service. Without correctly set SPNs, Kerberos authentication is not possible. Keytab files are the mechanism for storing the SPNs. Keytab files are copied to the WebLogic Server domain and are used in the login process. This configuration step describes how to create an SPN, user mapping, and keytab file for WebLogic Server.

This configuration step requires the use of the following Active Directory utilities:

- `setspn`–Windows 2000 Resource Kit
- `ktpass`–Windows 2000 distribution CD in `Program Files\Support Tools`

Note: The `setspn` and `ktpass` Active Directory utilities are products of Microsoft. Therefore, BEA Systems does not provide complete documentation for this utilities. For more information, see the appropriate Microsoft documentation.

To create a Kerberos identification for WebLogic Server:

1. Create a user account for the host computer on which WebLogic Server runs in the Active Directory server. (Select New > User, not New > Machine.)

   When creating the user account, use the simple name of the computer. For example, if the host is named `myhost.example.com`, create a user in Active Directory called `myhost`.

   Note the password you defined when creating the user account. You will need it in step 3. Do not select the `User must change password at next logon` option, or any other password options.

2. Configure the new user account to comply with the Kerberos protocol. The user account's encryption type must be DES and the account must require Kerberos pre-authentication.

   a. Right-click the name of the user account in the Users tree in the left pane and select Properties.

   b. Select the Account tab and check the box "Use DES encryption types for this account." Make sure no other boxes are checked, particularly the box "Do not require Kerberos pre-authentication."

   c. Setting the encryption type may corrupt the password. Therefore, you should reset the user password by right-clicking the name of the user account, selecting Reset Password, and re-entering the same password specified earlier.

3. Use the `setspn` utility to create the Service Principal Names (SPNs) for the user account created in step 1. Enter the following commands:

   ```
   setspn -a host/myhost.example.com myhost
   ```

   ```
   setspn -a HTTP/myhost.example.com myhost
   ```

4. Check which SPNs are associated with your user account, using the following command:

   ```
   setspn -L account name
   ```

   This is an important step. If the same service is linked to a different account in the Active Directory server, the client will not send a Kerberos ticket to the server.

5. Create a user mapping using the `ktpass` utility:

   Windows
   ```
   ktpass -princ host/myhost@Example.CORP -pass password -mapuser myhost
   -out c:\temp\myhost.host.keytab
   ```

6. Create a keytab file. On Windows, the `ktab` utility manages principal name and key pairs in the key table and allows you to list, add, update, or delete principal names and key pairs. On UNIX, it is preferable to use the `ktpass` utility.

   Windows

   a. Run the `ktab` utility on the host on which WebLogic Server is running to create the keytab file:

      ```
      ktab -k keytab-filename -a myhost@Example.CORP
      ```

   b. Copy the keytab file to the startup directory in the WebLogic Server domain.

   UNIX

   a. Create a user mapping using the `ktpass` utility:

      ```
      ktpass -princ HTTP/myhost@Example.CORP -pass password -mapuser myhost
      -out c:\temp\myhost.HTTP.keytab
      ```

      where *`password`* is the password for the user account created in step 1.

   b. Copy the keytab file created in Step a to the startup directory in the WebLogic Server domain.

c. Login as root and then merge them into a single keytab using the `ktutil` utility as follows:

   ```
   ktutil: "rkt myhost.host.keytab"
   ktutil: "rkt myhost.HTTP.keytab"
   ktutil: "wkt mykeytab"
   ktutil: "q"
   ```

7. Run the `kinit` utility to verify Kerberos authentication is working properly.

   ```
   kinit -k -t keytab-file account-name
   ```

   The output should be something similar to:

   ```
   New ticket is stored in cache file C:\Documents and
   Settings\Username\krb5cc_MachineB
   ```

Configuring Microsoft Clients to Use Windows Integrated Authentication

Ensure the Microsoft client you want to use for single sign-on is configured to use Windows Integrated authentication. The following sections describe how to configure a .NET Web server and an Internet Explorer browser to use Windows Integrated authentication.

Configuring a .NET Web Service

To configure a .NET Web Service to use Windows authentication, perform the following steps:

1. In the `web.config` file for the Web Service, set the authentication mode to Windows for IIS and ASP.NET as follows:

   ```
   <authentication mode="Windows" />
   ```

   This setting is usually the default.

2. Add the statement needed for the Web Services client to pass to the proxy Web Service object so that the credentials are sent through SOAP.

   For example, if you have a Web Service client for a Web Service that is represented by the proxy object `conv`, the syntax is as follows:

   ```
   /*
   * Explicitly pass credentials to the Web Service
   */
   conv.Credentials =
   System.Net.CredentialCache.DefaultCredentials;
   ```

Configuring an Internet Explorer Browser

To configure an Internet Explorer browser to use Windows authentication, follow these procedures in Internet Explorer:

Configure Local Intranet Domains

1. In Internet Explorer, select Tools > Internet Options.
2. Select the Security tab.
3. Select Local intranet and click Sites.
4. In the Local intranet popup, ensure that the "Include all sites that bypass the proxy server" and "Include all local (intranet) sites not listed in other zones" options are checked.
5. Click Advanced.
6. In the Local intranet (Advanced) dialog box, add all relative domain names that will be used for WebLogic Server instances participating in the SSO configuration (for example, `myhost.example.com`) and click OK.

Configure Intranet Authentication

1. Select Tools > Internet Options.
2. Select the Security tab.
3. Select Local intranet and click Custom Level....
4. In the Security Settings dialog box, scroll to the User Authentication section.
5. Select Automatic logon only in Intranet zone. This option prevents users from having to re-enter logon credentials, which is a key piece to this solution.
6. Click OK.

Verify the Proxy Settings

If you have a proxy server enabled:

1. Select Tools > Internet Options.
2. Select the Connections tab and click LAN Settings.
3. Verify that the proxy server address and port number are correct.

4. Click Advanced.
5. In the Proxy Settings dialog box, ensure that all desired domain names are entered in the Exceptions field.
6. Click OK to close the Proxy Settings dialog box.

Set Integrated Authentication for Internet Explorer 6.0

In addition to the previous settings, one additional setting is required if you are running Internet Explorer 6.0.

1. In Internet Explorer, select Tools > Internet Options.
2. Select the Advanced tab.
3. Scroll to the Security section.
4. Make sure that Enable Integrated Windows Authentication option is checked and click OK.
5. If this option was not checked, restart the computer.

Creating a JAAS Login File

If you are running WebLogic Server on either the Windows or UNIX platforms, you need a JAAS login file. The JAAS login file tells the WebLogic security framework to use Kerberos authentication and defines the location of the keytab file which contains Kerberos identification information for WebLogic Server. You specify the location of this file in the `java.security.auth.login.config` startup argument for WebLogic Server, as described in "Startup Arguments for Using Kerberos Authentication with WebLogic Server" on page 6-10.

Listing 6-2 contains a sample JAAS login file for Kerberos authentication.

Listing 6-2 Sample JAAS Login File for Kerberos Authentication

```
com.sun.security.jgss.initiate {

     com.sun.security.auth.module.Krb5LoginModule required
     principal="myhost@Example.CORP" useKeyTab=true
     keyTab=mykeytab storeKey=true;
};
```

```
com.sun.security.jgss.accept {

     com.sun.security.auth.module.Krb5LoginModule required
     principal="myhost@Example.CORP" useKeyTab=true
     keyTab=mykeytab storeKey=true;

};
```

Configuring the Identity Asssertion Provider

WebLogic Server includes a security provider, the Negotiate Identity Assertion provider, to support single sign-on (SSO) with Microsoft clients. This identity assertion provider decodes Simple and Protected Negotiate (SPNEGO) tokens to obtain Kerberos tokens, validates the Kerberos tokens, and maps Kerberos tokens to WebLogic users. You need to configure a Negotiate Identity Assertion provider in your WebLogic security realm in order to enable SSO with Microsoft clients. For information, see "Configuring a Negotiate Identity Assertion Provider" on page 5-21 and Configure Authentication and Identity Assertion providers in the Administration Console help.

Startup Arguments for Using Kerberos Authentication with WebLogic Server

To use Kerberos authentication with WebLogic Server, use the following start-up arguments when you start WebLogic Server:

```
-Djava.security.krb5.realm=Example.CORP
-Djava.security.krb5.kdc=ADhostname
-Djava.security.auth.login.config=krb5Login.conf
-Djavax.security.auth.useSubjectCredsOnly=false
-Dweblogic.security.enableNegotiate=true
```

where

- `java.security.krb5.realm` defines the Microsoft domain in which the Active Directory server runs.
- `java.security.krb5.kdc` defines the host name on which the Active Directory server runs.

- `java.security.auth.login.config` defines the location of the Kerberos login information.
- `javax.security.auth.useSubjectCredsOnly` specifies that it is permissible to use an authentication mechanism other than Subject credentials.

`weblogic.security.enableNegotiate` enables the Servlet container in WebLogic Server to support the Negotiate token used by SPNEGO.

Verifying that SSO with Microsoft Clients Works

To verify that SSO with Microsoft clients is properly configured, point a browser (that you have configured as described in "Configuring an Internet Explorer Browser" on page 6-8) to the Microsoft Web application or Web Service you want to use. If you are logged on to a Windows domain and have Kerberos credentials acquired from the Active Directory server in the domain, you should be able to access the Web application or Web Service without providing a username or password.

Configuring Single Sign-On with Web Browsers and HTTP Clients

This section explains how to set up single sign-on (SSO) with Web browsers or other HTTP clients, using authentication based on the Security Assertion Markup Language 1.1 (SAML).

Overview of SAML-Based Single Sign-On

You can use Authentication providers based on the Security Assertion Markup Language (SAML) to allow cross-platform authentication between Web applications or Web Services running in a WebLogic Server domain and Web browsers or other HTTP clients. This enables single sign-on (SSO): once users are authenticated in one site that participates in a single sign-on configuration, they don't need to log in separately at the other sites in the SSO configuration.

SAML SSO works as follows:

1. A Web user authenticates to a SAML source site.
2. The user then attempts to access a target resource at a destination site.
3. Through one or more steps (for example, redirection), the user arrives at an Intersite Transfer Service (ITS) at the source site. The WebLogic Server SAML Credential Mapping provider can serve as an ITS.

4. Through a sequence of HTTP exchanges, the user browser is transferred to an Assertion Consumer Service (ACS) at the SAML destination site. The WebLogic Server SAML Identity Assertion provider can serve as an ACS.

5. Information about the SAML assertion provided by the source site and associated with the user and the desired target is conveyed from the source site to the destination site by the protocol exchange.

6. The ACS at the destination site examines both the assertion and the target information to determine whether to allow access to the target resource thereby achieving web SSO for authenticated users originating from a source site.

For a general overview of how WebLogic Server implements SAML, see Single Sign-On: Web Sites and Web Applications and Single Sign-On with the WebLogic Security Framework in *Understanding WebLogic Security*. For more information about adding security to Web Services, see Configuring Security in *Programming WebLogic Web Services*.

For more information about SAML, see http://www.oasis-open.org.

Single Sign-on with SAML: Main Steps

To configure WebLogic Server to support single sign-on with SAML, you need to configure support for acting as a SAML source site and a SAML destination site. Although SAML support involves a variety of different servlets and services, SAML source site configuration is centralized in the SAML Credential Mapping provider, while SAML destination configuration is centralized in the SAML Identity Assertion provider.

The main steps in configuring SAML SSO in WebLogic Server are:

1. Determine if you want to support Artifact profile, POST profile, or both. See Single Sign-On with the WebLogic Security Framework for a discussion of how these profiles work.

2. Configure WebLogic Server as a SAML source site, by creating and configuring a SAML Credential Mapping provider in your security realm. See "Configuring a SAML Source Site for Single Sign-On" on page 7-3. For a reference-oriented description of SAML Credential Mapping provider configuration, see "Configuring a SAML Credential Mapping Provider" on page 4-19. As you configure the SAML Credential Mapping provider, you configure:
   - the Source Site URLs and Intersite Transfer URIs
   - attributes that enable and configure POST profile, Artifact profile, or both
   - the types of SAML assertions that the SAML Credential Mapping provider will produce

3. Configure WebLogic Server as a SAML destination site, by creating and configuring a SAML Identity Assertion provider in your security realm. See "Configuring a SAML Destination Site for Single Sign-On" on page 7-4. For a reference-oriented description of SAML Identity Assertion provider configuration, see "Configuring a SAML Identity Assertion Provider" on page 5-21. As you configure the SAML Credential Mapping provider, you configure:

- the Assertion Consumer URIs and Source Site Redirects attributes
- attributes that enable and configure POST profile, Artifact profile, or both
- the attributes of SAML assertions that the SAML Identity Assertion provider will consume

4. Establish trust by registering the source site's SSL certificate in the certificate registry maintained by the SAML Identity Assertion provider. See "Certificate Registry" on page 5-23.

Configuring a SAML Source Site for Single Sign-On

This section describes how to configure WebLogic Server as a SAML source site. SAML source site configuration is centralized in the SAML Credential Mapping provider. In your security realm, create a SAML Credential Mapping provider instance. The SAML Credential Mapping provider is not part of the default security realm. See "Configuring a SAML Credential Mapping Provider" on page 4-19.

Configure SAML Authority Attributes

Configure the SAML Credential Mapping provider as a SAML authority, using the Issuer URI, Name Qualifier, and other attributes.

Configure Source Site Attributes

Configure the SAML Credential Mapping provider as a SAML source site, using the Source Site URLs and Intersite Transfer URIs attributes.

Configure Supported Profiles

You can configure the SAML Credential Mapping provider to support Artifact profile, POST profile, or both, for the purposes of SAML SSO. Be sure to configure support for the profiles that the SAML destination sites support.

To configure support for Artifact profile:

1. In the SAML Credential Mapping provider, set Artifact Enabled to true.
2. Configure an ITS for Artifact profile in the Intersite Transfer URIs attribute.

To configure support for POST profile:

1. In the SAML Credential Mapping provider, set POST Enabled to true.
2. Optionally, create a default form to use in POST profile assertions and set the pathname of that form in the Default POST Form attribute.
3. Configure an ITS for POST profile in the Intersite Transfer URIs attribute.

Configure Produced Assertions

In the SAML Credential Mapping provider, you can configure the SAML assertions that will be provided by WebLogic Server when it acts as a SAML source site. You can configure any number of assertions, depending on the requirements of the SAML destination sites your users are connecting to. For information about configuring produced assertions, see "Produced Assertion Configuration" on page 4-22.

Configuring a SAML Destination Site for Single Sign-On

Configure WebLogic Server as a SAML destination. SAML destination configuration is centralized in the SAML Identity Assertion provider. See "Configuring a SAML Identity Assertion Provider" on page 5-21.

In your security realm, create a SAML Identity Assertion provider instance. The SAML Identity Assertion provider is not part of the default security realm.

Configure Supported Profiles

You can configure the SAML Identity Assertion provider to support Artifact profile, POST profile, or both, for the purposes of SAML SSO.

To configure support for Artifact profile, in the SAML Identity Assertion provider, set Artifact Enabled to true.

To configure support for POST profile:

1. In the SAML Identity Assertion provider, set POST Enabled to true.
2. Optionally, set the Recipient Check Enabled attribute. If true, this attribute requires that the recipient of the SAML Response must match the URL in the HTTP Request.

3. Optionally, set the Enforce One Use Policy attribute. See "Limiting the Re-use of Assertions" on page 5-23.

Configure Consumed Assertions

In the SAML Identity Assertion provider, you can configure how WebLogic Server consumes assertions it receives when it acts as a SAML destination site. You can configure any number of assertions. For information about configuring consumed assertions, see "Consumed Assertion Configuration" on page 5-24.

CHAPTER 8

Migrating Security Data

WebLogic Server provides methods for exporting security data from one security realm or security provider and importing the data into another realm or provider. The following sections provide information about exporting and importing security data.

Overview of Security Data Migration

Security realms persist different kinds of security data. Examples of security data used by security providers in a WebLogic security realm include users and groups (for the WebLogic Authentication provider), security policies (for the WebLogic Authorization provider), security roles (for the WebLogic Role Mapping provider), or credential maps (for the WebLogic Credential Mapping provider). When you configure a new security realm or a new security provider, you may prefer to use the security data from your existing realm or provider, rather than recreate all the users, groups, policies, roles, and credential maps. Several WebLogic security providers support security data migration. This means you can export security data from one security realm, and import it into a new security realm. You can migrate security data for each security provider individually, or migrate security data for all the WebLogic security providers at once (that is, security data for an entire security realm). You migrate security data through the WebLogic Administration Console or by using the WebLogic Scripting Tool (WLST).

Migrating security data may be helpful when:

- Transitioning from development to production mode.
- Copying production mode security configurations to the security realms in new WebLogic Server domains.
- Moving data from one security realm to a new security realm in the same WebLogic Server domain, where one or more of the default WebLogic security providers will be replaced with new security providers.

The remainder of this section describes the concepts you need to understand when migrating security data, the formats and constraints supported by the WebLogic security providers, and how to use WLST to migrate security data.

For more information about migrating security data using the WebLogic Administration Console, see the following topics in the online help:

- "Export data from security realms"
- "Import data into security realms"
- "Export data from security providers"
- "Import data into security providers"

Migration Concepts

A **format** is simply a data format that specifies how security data should be exported or imported. **Supported formats** are the list of data formats that a given security provider understands how to process.

Constraints are key/value pairs that specify options to the export or import process. Use constraints to control which security data is exported to or imported from the security provider's database (in the case of the WebLogic Server security providers, the embedded LDAP server). For example, you may want to export only users (not groups) from an Authentication provider's database. **Supported constraints** are the list of constraints you may specify during the migration process for a particular security provider. For example, an Authentication provider's database may be used to import users and groups, but not security policies.

Export files are the files to which security data is written (in the specified format) during the export portion of the migration process. **Import files** are files from which security data is read (also in the specified format) during the import portion of the migration process. Both export and import files are simply temporary storage locations for security data as it is migrated from one security provider's data store to another.

Formats and Constraints Supported by the WebLogic Security Providers

WebLogic Server does not provide any standard, public formats for developers of security providers. Therefore, in order for security data to be exported and imported from one security provider to another, both security providers must understand how to process the same format.

Notes: Because the data format used for the WebLogic Server security providers is unpublished, you cannot currently migrate security data from a WebLogic security provider to a custom security provider, or vice versa.

WebLogic security providers support the following formats and constraints.

Table 8-1 Formats and Constraints Supported by the WebLogic Security Providers

WebLogic Provider	Supported Format	Supported Constraints
WebLogic Authentication Provider	DefaultAtn	Users, groups
WebLogic Authorization Provider	DefaultAtz	None
WebLogic Role Mapping Provider	DefaultRoles	None
WebLogic Credential Mapping Provider	DefaultCreds	Passwords

In the WebLogic Administration Console, the constraints are only displayed for the WebLogic Authentication provider because you have the option of exporting or importing users and groups, only users, or only groups.

When exporting credential maps from the WebLogic Credential Mapping provider, you need to specify whether or not the passwords for the credentials are exported in clear text. The mechanism used to encrypt passwords in each WebLogic Server domain is different; therefore, you want to export passwords in clear text if you plan to use them in a different WebLogic Server domain. After the credential maps are imported into the WebLogic Credential Mapping provider in the new WebLogic Server domain, the passwords are encrypted. Carefully protect the directory and file in which you export credential maps in clear text as secure data is available on your system during the migration process.

Migrating Data Using WLST

You can use the WebLogic Scripting Tool (WLST) to export and import data from a security provider. The format of the WLST command to import data is:

```
cd("SecurityConfiguration/mydomain/DefaultRealm/myrealm/path-to-MBean/mbea
nname")
```

```
cmo.importData(format,filename,constraints)
```

where

mbeanname—Name of the security provider MBean.

format—`DefaultAtn`, `DefaultAtz`, `DefaultRoles`, or `DefaultCreds`

filename—The directory location and filename in which to export or import the security data. Remember that, regardless of whether you are using a UNIX or Windows operating system, you need to use a forward slash, not a back slash, as a path separator for pathname arguments in WLST commands.

constraints–The contraints that limit the data to be exported or imported

For more information, see *WebLogic Scripting Tool*.

Migrating Data Using weblogic.admin

Note: The `weblogic.Admin` utility is deprecated in this release of WebLogic Server. Use WLST instead.

You can also use the `weblogic.Admin` utility to export and import security data between security realms and security providers. The format of the command is:

```
java weblogic.Admin -username username -password password \
INVOKE -mbean mbeanname \
-method methodname dataformat filename constraints
```

where

username—Name of the Admin user

password—Password of the Admin user

mbeanname—Name of the security provider MBean.

methodname—`exportData` or `importData`

dataformat—`DefaultAtn`, `DefaultAtz`, `DefaultRoles`, or `DefaultCreds`

filename—The directory location and filename in which to export or import the security data `constraints`–The constraints that limit the data to be exported or imported

Note: The directory and file into which you export the security data should be carefully protected with operating system security as they contain secure information about your deployment.

For example:

```
java weblogic.Admin -username system -password weblogic INVOKE -mbean
Security:Name=myrealmDefaultAuthenticator -method importData DefaultAtn
d:\temp\security.info " "
```

Managing the Embedded LDAP Server

WebLogic Server includes an embedded LDAP server that acts as the default security provider data store for the WebLogic Authentication, Authorization, Credential Mapping, and Role Mapping providers.The following sections explain how to manage the embedded LDAP server.

Configuring the Embedded LDAP Server

The embedded LDAP server contains user, group, group membership, security role, security policy, and credential map information. By default, each WebLogic Server domain has an embedded LDAP server configured with the default values set for each type of information. The WebLogic Authentication, Authorization, Credential Mapping, and Role Mapping providers use the embedded LDAP server as their data store. If you use any of these providers in a new security realm, you may want to change the default values for the embedded LDAP server to optimize its use in your environment.

Note: The performance of the embedded LDAP server is best with fewer than 50,000 users. If you have more users, consider using a different LDAP server and Authentication provider.

For more information, see "Configure the embedded LDAP server" in the Administration Console online help.

Embedded LDAP Server Replication

The WebLogic Server embedded LDAP server for a domain consists of a master LDAP server, maintained in the domain's Administration Server, and a replicated LDAP server maintained in each Managed Server in the domain. When changes are made to the LDAP directory using the embedded LDAP server in Managed Server, updates are sent to the master LDAP server. The master LDAP server maintains a log of all changes. The master LDAP server also maintains a list replicated servers and the current change status for each one. The master LDAP server sends appropriate changes to each replicated server and updates the change status for each server. This process occurs when an update is made to the master LDAP server. However, depending on the number of updates, it may take several seconds or more for the change to be replicated to the Managed Server.

You can configure the behavior of the master LDAP server and the replicated LDAP servers in a domain using the Administration Console. On the Domain: Security: Embedded LDAP Server page in the Administration Console, you can set these attributes:

- Refresh Replica At Startup—Specifies whether the embedded LDAP server in a Managed Server should refresh all replicated data at boot time. This is useful if you have made a large amount of changes when the Managed Server was not active, and you want to download the entire replica instead of having the Administration Server push each change to the Managed Server.
- Master First—Specifies whether a Managed Server should always connect to the master LDAP server, instead of connecting to the local replicated LDAP server.

See "Configure the embedded LDAP server" in the Administration Console online help.

Note: Deleting and modifying the configured security providers using the WebLogic Administration Console may require manual clean up of the embedded LDAP server. Use an external LDAP browser to delete unnecessary information.

Viewing the Contents of the Embedded LDAP Server from an LDAP Browser

To view the contents of the embedded LDAP server through an LDAP browser:

1. Download and install an external LDAP browser. You can find one LDAP browser at the following location:

   `http://www-unix.mcs.anl.gov/~gawor/ldap/`

2. In the WebLogic Administration Console, change the credential for the embedded LDAP server.

   a. Expand Domain > Security > Embedded LDAP.

   b. In the Credential field, enter the new credential.

   c. In the Confirm Credential field, enter the new credential again.

   d. Click Save.

   e. Reboot WebLogic Server.

3. Start the LDAP browser. To start the LDAP Browser/Editor mentioned in step 1, use this command:

   ```
   lbe.sh
   ```

4. In the LDAP browser, configure a new connection in the LDAP browser:

   a. Set the host field to *`localhost`*.

   b. Set the port field to `7001` (`7002` if SSL is being used).

   c. Set the Base DN field to *`dc=mydomain`* where *`mydomain`* is the name of the WebLogic Server domain you are using.

   d. Uncheck the Anonymous Bind option.

   e. Set the User DN field to `cn=Admin`.

   f. Set the Password field to the credential you specified in Step 2.

5. Click the new connection.

   Use the LDAP browser to navigate the hierarchy of the embedded LDAP server.

Exporting and Importing Information in the Embedded LDAP Server

You can export and import data from the embedded LDAP server using either the WebLogic Administration Console or an LDAP browser. To export and import data with the console, use the Migration page of each security provider. See Export data from a security provider and Import data into a security provider in the Administration Console online help. This section describes how to use an LDAP browser to export and import data stored in the embedded LDAP server. Table 9-1 summarizes where data is stored in the hierarchy of the embedded LDAP server.

Table 9-1 Location of Security Data in the Embedded LDAP Server

Security Data	Embedded LDAP Server DN
Users	`ou=people,ou=`*`myrealm`*`,dc=`*`mydomain`*
Groups	`ou=groups,ou=`*`myrealm`*`,dc=`*`mydomain`*
Security roles	`ou=ERole,ou=`*`myrealm`*`,dc=`*`mydomain`*
Security policies	`ou=EResource,ou=`*`myrealm`*`,dc=`*`mydomain`*

To export security data from the embedded LDAP server using the LDAP Browser/Editor:

1. Enter the following command at a command prompt to start the LDAP Browser/Editor:

   ```
   lbe.sh
   ```

2. Specify the data to be exported (for example, to export users specify `ou=people,ou=`*`myrealm`*`,dc=`*`mydomain`*).

3. Select the LDIF > Export option.

4. Select Export all children.

5. Specify the name of the file into which the data will be exported.

To import security data into the embedded LDAP server using the LDAP Browser/Editor:

1. Enter the following command at a command prompt to start the LDAP browser:

   ```
   lbe.sh
   ```

2. Specify the data to be imported (for example, to import users, specify `ou=people,ou=`*`myrealm`*`,dc=`*`mydomain`*).

3. In the LDAP Browser/Editor, select the LDIF > Import option.
4. Select Update/Add.
5. Specify the name of the file from which the data will be imported.

LDAP Access Control Syntax

The embedded LDAP server supports the IETF LDAP Access Control Model for LDAPv3. This section describes how those access control is implemented within the embedded LDAP server. These rules can be applied directly to entries within the directory as intended by the standard or can be configured and maintained by editing the access control file (`acls.prop`).

Note: The default behavior of the embedded LDAP server is to allow access only from the Admin account in WebLogic Server. The WebLogic security providers use only the Admin account to access the embedded LDAP server. If you are not planning to access the embedded LDAP server from an external LDAP browser or if you are planning only to use the Admin account, you do not need to edit the `acls.prop` file. You may ignore the procedures in this section.

The Access Control File

The access control file (`acls.prop`) maintained by the embedded LDAP server contains the complete list of access control lists (ACLs) for an entire LDAP directory. Each line in the access control file contains a single access control rule. An access control rule is made up of the following components:

- Location in the LDAP directory where the rule applies
- Scope within that location to which the rule applies
- Access rights (either grant or deny)
- Permissions (either grant or deny)
- Attributes to which the rule applies
- Subject being granted or denied access

Listing 9-1 shows a sample access control file.

Listing 9-1 Sample acl.props File

```
[root]|entry#grant:r,b,t#[all]#public

ou=Employees,dc=octetstring,dc=com|subtree#grant:r,c#[all]#public:
ou=Employees,dc=octetstring,dc=com|subtree#grant:b,t#[entry]#public:
ou=Employees,dc=octetstring,dc=com|subtree#deny:r,c#userpassword#public:
ou=Employees,dc=octetstring,dc=com|subtree#grant:r#userpassword#this:
ou=Employees,dc=octetstring,dc=com|subtree#grant:w,o#userpassword,title,
description,
postaladdress,telephonenumber#this:
cn=schema|entry#grant:r#[all]#public:
```

Access Control Location

Each access control rule is applied to a given location in the LDAP directory. The location is normally a distinguished name (DN) but the special location `[root]` can be specified in the `acls.prop` file if the access control rule applies to the entire directory.

If an entry being accessed or modified on the LDAP server does not equal or reside below the location of the access control rule, the given access control rule is not evaluated further.

Access Control Scope

The following access control scopes are defined:

- Entry—An ACL with a scope of Entry is only evaluated if the entry in the LDAP directory shares the same DN as the location of the access control rule. Such rules are useful when a single entry contains more sensitive information than parallel or subentries entries.
- Subtree—A scope of Subtree is evaluated if the entry in the LDAP directory equals or ends with the location of this access control. This scope protects means the location entry and all subentries.

If an entry in the directory is covered by conflicting access control rules (for example, where one rule is an Entry rule and the other is a Subtree rule), the Entry rule takes precedence over rules that apply because of the Subtree rule.

Access Rights

Access rights apply to an entire object or to attributes of the object. Access can be granted or denied. Either of the actions `grant` or `deny` may be used when creating or updating the access control rule.

Each of the LDAP access rights are discrete. One right does not imply another right. The rights specify the type of LDAP operations that can be performed.

Attribute Permissions

The following permissions apply to actions involving attributes.

Table 9-2 Attribute Permissions

Permission	Description
`r` Read	Read attributes. If granted, permits attributes and values to be returned in a Read or Search operation.
`w` Write	Modify or add attributes. If granted, permits attributes and values to be added in a Modify operation.
`o` Obliterate	Modify and delete attributes. If granted, permits attributes and values to be deleted in a Modify operation.
`s` Search	Search entries with specified attributes. If granted, permits attributes and values to be included in a Search operation.
`c` Compare	Compare attribute values. If granted, permits attributes and values to be included in a Compare operation.
`m` Make	Make attributes on a new LDAP entry below this entry.

The `m` permission is required for all attributes placed on an object when it is created. Just as the `w` and `o` permissions are used in the Modify operation, the `m` permission is used in the Add operation. The `w` and `o` permissions have no bearing on the Add operation and `m` has no bearing on the Modify operation. Since a new object does not yet exist, the `a` and `m` permissions needed to create it must be granted to the parent of the new object. This requirement differs from `w` and `o` permissions which must be granted on the object being modified. The `m` permission is distinct and separate from the `w` and `o` permissions so that there is no conflict between the permissions needed to add new children to an entry and the permissions needed to modify existing children of the same entry. In order to replace values with the Modify operation, a user must have both the `w` and `o` permissions.

Entry Permissions

The following permissions apply to entire LDAP entries.

Table 9-3 Entry Permissions

Permission	Description
`a` Add	Add an entry below this LDAP entry. If granted, permits creation of an entry in the DIT subject to control on all attributes and values placed on the new entry at the time of creation. In order to add an entry, permission must also be granted to add at least the mandatory attributes.
`d` Delete	Delete this entry. If granted, permits the entry to be removed from the DIT regardless of controls on attributes within the entry.
`e` Export	Export entry and all subentries to new location. If granted, permits an entry and its subentries (if any) to be exported; that is, removed from the current location and placed in a new location subject to the granting of suitable permission at the destination. If the last RDN is changed, `Rename` permission is also required at the current location. In order to export an entry or its subentries, there are no prerequisite permissions to the contained attributes, including the RDN attribute. This is true even when the operation causes new attribute values to be added or removed as the result of the changes to the RDN.

Table 9-3 Entry Permissions

Permission	Description
`i` Import	Import entry and subentries from specified location. If granted, permits an entry and its subentries (if any) to be imported; that is, removed from one other location and placed at the specified location (if suitable permissions for the new location are granted). When importing an entry or its subentries, there are no prerequisite permissions for the contained attributes, including the RDN attributes. This is true even when the operation causes new attribute values to be added or removed as the result of the changes to RDN.
`n` RenameDN	Change the DN of an LDAP entry. Granting the `Rename` permission is necessary for an entry to be renamed with a new RDN, taking into account consequential changes to the DN of subentries. If the name of the superior entry is unchanged, the grant is sufficient. When renaming an entry, there are no prerequisite permissions to contained attributes, including the RDN attributes. This is true even when the operation causes new attribute values to be added or removed as the result of the changes of RDN.
`b` BrowseDN	Browse the DN of an entry. If granted, permits entries to be accessed using directory operations that do not explicitly provide the name of the entry.
`t` ReturnDN	Allows DN of entry to be disclosed in an operation result. If granted, allows the distinguished name of the entry to be disclosed in the operation result.

Attributes Types

The attribute types to which an access control rule applies should be listed where necessary. The following keywords are available:

- `[entry]` indicates the permissions apply to the entire object. This could mean actions such as delete the object, or add a child object.
- `[all]` indicates the permissions apply to all attributes of the entry.

If the keyword `[all]` and another attribute are both specified within an ACL, the more specific permission for the attribute overrides the less specific permission specified by the `[all]` keyword.

Subject Types

Access control rules can be associated with a number of subject types. The subject of an access control rule determines whether the access control rule applies to the currently connected session.

The following subject types are defined:

- `AuthzID`—Applies to a single user that can be specified as part of the subject definition. The identity of that user in the LDAP directory is typically defined as a DN.
- `Group`—Applies to a group of users specified by one of the following object classes:
  - `groupOfUniqueNames`
  - `groupOfNames`
  - `groupOfUniqueURLs`

  The first two types of groups contain lists of users, while the third type allows users to be included in the group automatically based on defined criteria.
- `Subtree`—Applies to the DN specified as part of the subject and all subentries in the LDAP directory tree.
- `IP Address`—Applies to a particular Internet address. This subject type is useful when all access must come through a proxy or other server. Applies only to a particular host, not to a range or subnet.
- `Public`—Applies to anyone connected to the directory, whether they are authenticated or not.
- `This`—Applies to the user whose DN matches that of the entry being accessed.

Grant/Deny Evaluation Rules

The decision whether to grant or deny a client access to the information in an entry is based on many factors related to the access control rules and the entry being protected. Throughout the decision making process, these guiding principles apply:

- More specific rules override less specific ones (for example, individual user entries in an ACL take precedence over a group entry).
- If a conflict still exists in spite of the specificity of the rule, the subject of the rule determines which rule will be applied. Rules based on an `IP Address` subject are given the highest precedence, followed by rules that are applied to a specific `AuthzID` or `This` subject. Next in priority are rules that apply to `Group` subjects. Last priority is given to rules that apply to `Subtree` and `Public` subjects.
- When there are conflicting ACL values, Deny takes precedence over Grant.
- Deny is the default when there is no access control information. Additionally, an entry scope takes precedence over a subtree scope.

Configuring Identity and Trust

This following sections describe how to configure identity and trust for WebLogic Server:

Before performing the steps in this chapter, review the "Identity and Trust" section in *Understanding WebLogic Security*.

Private Keys, Digital Certificates, and Trusted Certificate Authorities

Private keys, digital certificates, and trusted certificate authorities establish and verify server identity and trust.

SSL uses public key encryption technology for authentication. With public key encryption, a public key and a *private key* are generated for a server. The keys are related such that data encrypted with the public key can only be decrypted using the corresponding private key and vice versa. The private key is carefully protected so that only the owner can decrypt messages that were encrypted using the public key.

The public key is embedded into a *digital certificate* with additional information describing the owner of the public key, such as name, street address, and e-mail address. A private key and digital certificate provide *identity* for the server.

The data embedded in a digital certificate is verified by a certificate authority and digitally signed with the certificate authority's digital certificate. Well-know certificate authorities include Verisign and Entrust.net. The trusted certificate authority (CA) certificate establishes *trust* for a certificate.

An application participating in an SSL connection is authenticated when the other party evaluates and accepts the application's digital certificate. Web browsers, servers, and other SSL-enabled applications generally accept as genuine any digital certificate that is signed by a trusted certificate authority and is otherwise valid. For example, a digital certificate can be invalidated because it has expired or the digital certificate of the certificate authority used to sign it expired. A server certificate can be invalidated if the host name in the digital certificate of the server does not match the URL specified by the client.

Configuring Identity and Trust: Main Steps

To create identity and trust for a server:

1. Obtain digital certificates, private keys, and trusted CA certificates from the CertGen utility, Sun Microsystem's `keytool` utility, or a reputable vendor such as Entrust or Verisign. You can also use the digital certificates, private keys, and trusted CA certificates provided by the WebLogic Server kit. The demonstration digital certificates, private keys, and trusted CA certificates should be used in a development environment only.
2. Store the private keys, digital certificates, and trusted CA certificates. Private keys and trusted CA certificates are stored in a keystore.

   Note: The preferred keystore format is JKS (Java KeyStore). WebLogic Server supports private keys and trusted CA certificates stored in files or in the WebLogic Keystore provider for the purpose of backward compatibility only.

3. Configure the identity and trust keystores for WebLogic Server in the WebLogic Server Administration Console. See "Configure Keystores" in the *Administration Console Online Help*.

The remainder of this chapter describes these steps.

Supported Formats for Identity and Trust

The PEM (Privacy Enhanced Mail) format is the preferred format for private keys, digital certificates, and trusted certificate authorities (CAs). The preferred keystore format is the JKS (Java KeyStore) format.

A `.pem` format file begins with this line:

```
----BEGIN CERTIFICATE----
```

and ends with this line:

```
----END CERTIFICATE----
```

A `.pem` format file supports multiple digital certificates (for example, a certificate chain can be included). The order of certificates within the file is important. The server's digital certificate should be the first digital certificate in the file, followed by the issuer certificate, and so on. Each certificate in the chain is followed by its issuer certificate. If the last certificate in the chain is the self-signed (self-issued) root certificate of the chain, the chain is considered complete. Note that the chain does not have to be complete.

When using the deprecated file-based private keys, digital certificates, and trusted CAs, WebLogic Server can use digital certificates in either PEM or distinguished encoding rules (DER) format.

A `.der` format file contains binary data for a single certificate. Thus, a `.der` file can be used only for a single certificate, while a `.pem` file can be used for multiple certificates.

Microsoft is often used as a certificate authority. Microsoft issues trusted CA certificates in p7b format, which must be converted to PEM before they can be used with WebLogic Server. For more information, see "Converting a Microsoft p7b Format to PEM Format" on page 10-9.

Private key files (meaning private keys not stored in a keystore) must be in PKCS#5/PKCS#8 PEM format.

You can still use private keys and digital certificates used with other versions of WebLogic Server with this version of WebLogic Server. Convert the private key and digital certificate from distinguished encoding rules (DER) format to privacy-enhanced mail (PEM) format. For more information, see the description of the der2pem utility in "Using the WebLogic Server Java Utilities" in *WebLogic Server Command Reference*.

After converting the files, ensure the digital certificate file has the `-----BEGIN CERTIFICATE-----` header and the `-----END CERTIFICATE-----` footer. Otherwise, the digital certificate will not work.

Note: OpenSSL can add a header to the PEM certificate it generates. In order to use such certificates with WebLogic Server, everything in front of "`-----BEGIN CERTIFICATE-----`" should be removed from the certificate, which you can do using a text editor.

Obtaining Private Keys, Digital Certificates, and Trusted Certificate Authorities

Servers need a private key, a digital certificate containing the matching public key, and a certificate for at least one trusted certificate authority. WebLogic Server supports private keys, digital certificates, and trusted CA certificates from the following sources:

- The demonstration digital certificates, private keys, and trusted CA certificates in the *`WL_HOME`*`\server\lib` directory and the *`JAVA_HOME`*`\jre\lib\security` directory.

  The demonstration digital certificates, private keys, and trusted CA certificates should be used in a development environment only.

- Sun Microsystem's `keytool` utility can also be used to generate a private key, a self-signed digital certificate for WebLogic Server, and a Certificate Signing Request (CSR).
  - Submit the CSR to a certificate authority to obtain a digital certificate for WebLogic Server.
  - Use the keytool utility to update the self-signed digital certificate with a new digital certificate.
  - Use the keytool utility to obtain trust and identity when using WebLogic Server in a production environment.

  For more information about Sun's `keytool` utility, see the keytool-Key and Certificate Management Tool description at http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/keytool.html.

  Note: When using the keytool utility, the default key pair generation algorithm is DSA. WebLogic Server does not support the use of the Digital Signature Algorithm (DSA).

Specify another key pair generation and signature algorithm when using WebLogic Server.

- The CertGen utility generates digital certificates and private keys that should be used only for demonstration or testing purposes in a development environment, and not in a production environment. Use the CertGen utility if you want to set an expiration date in the digital certificate or specify a correct host name in the digital certificate so that you can use host name verification. (The demonstration digital certificate provided by WebLogic Server uses the machine's default host name as the host name.) For more information about using the CertGen utility to obtain private keys and digital certificates, see "Using the CertGen Utility" on page 10-6.

Note: The Certificate Request Generator servlet is deprecated in this release of WebLogic Server. Use the `keytool` utility from Sun Microsystems in place of the Certificate Request Generator servlet. For more information about `keytool`, see "Common Keytool Commands" on page 10-5.

Common Keytool Commands

Table 10-1 lists `keytool` commands you use when creating and using JKS keystores with WebLogic Server.

Note: The `keytool` utility is a product of Sun Microsystems. Therefore, BEA Systems does not provide complete documentation on the utility. For more information, see the keytool-Key and Certificate Management Tool description at http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/keytool.html.

Table 10-1 Commonly Used keytool Commands

Command	Description
`keytool -genkey -keystore` *`keystorename`* `-storepass` *`keystorepassword`*	Generates a new private key entry and self-signed digital certificate in a keystore. If the keystore does not exist, it is created.
`keytool -import -alias` *`aliasforprivatekey`* `-file` *`privatekeyfilename.pem`* `-keypass` *`privatekeypassword`* `-keystore` *`keystorename`* `-storepass` *`keystorepassword`*	Updates the self-signed digital certificate with one signed by a trusted CA.

Table 10-1 Commonly Used keytool Commands (Continued)

Command	Description
`keytool -import -alias` *`aliasfortrustedca`* `-trustcacerts -file` *`trustedcafilename.pem`* `-keystore` *`keystorename`* `-storepass` *`keystorepassword`*	Loads a trusted CA certificate into a keystore. If the keystore does not exist, it is created.
`keytool -certreq -alias alias -sigalg` *`sigalg`* `-file` *`certreq_file`* `-keypass` *`privatekeypassword`* `-storetype` *`keystoretype`* `-keystore` *`keystorename`* `-storepass` *`keystorepassword`*	Generates a Certificate Signing Request (CSR), using the PKCS#10 format, and a self-signed certificate with a private key. Stores the CSR in the specified *`certreq_file`*, and the certificate/private key pair as a key entry in the specified keystore under the specified alias.
`keytool -list -keystore` *`keystorename`*	Displays what is in the keystore.
`keytool -delete -keystore` *`keystorename`* `-storepass` *`keystorepassword`* `-alias` *`privatekeyalias`*	Deletes the entry identified by the specified alias from the keystore.
`keytool -help`	Provides online help for keytool.

Using the CertGen Utility

Note: The CertGen utility generates digital certificates and private keys that should only be used for demonstration or testing purposes and not in a production environment.

The CertGen utility provides command line options to specify a CA certificate and key to be used for issuing generated certificates. The digital certificates generated by the CertGen utility have the host name of the machine on which they were generated as the value for its common name field (`cn`) by default only. Command line options let you specify values for the `cn` and other Subject domain name (DN) fields, such as `orgunit`, `organization`, `locality`, `state`, and `countrycode`.

The CertGen utility generates public certificate and private key files in PEM and DER formats. On Windows, double-click `.der` files to view the details of the generated digital certificate. The

`.pem` files can be used when you boot WebLogic Server or use the digital certificates with a client.

By default, the `CertGen` utility uses the following demonstration digital certificate and private-key files: `CertGenCA.der` and `CertGenCAKey.der`. CertGen looks for these files in the current directory, or in the *`WL_HOME`*`/server/lib` directory, as specified in the `weblogic.home` system property or the `CLASSPATH`. If you want to use these files, you need not specify CA files on the command line. Alternatively, you can specify CA files on the command line, as shown in the following command syntax.

1. Use the `CertGen` utility to generate a certificate. See CertGen in the *WebLogic Server Command Reference*. The following example shows the syntax for the `CertGen` utility:

```
$ java utils.CertGen

[-cacert <ca_cert_file>] [-cakey <ca_key_file>]
[-cakeypass <ca_key_password>] [-selfsigned]
[-certfile <cert_file>] [-keyfile <private_key_file>]
[-keyfilepass <keyfile_pass>] [-strength <key_strength>]
[-cn <common_name>] [-ou <org_unit>] [-o <organization>]
[-l <locality>] [-s <state>] [-c <country_code>]
[-subjectkeyid <subject_key_identifier>]
[-subjectkeyidformat UTF-8|BASE64]
```

Argument	Definition
`ca_cert_file`	The file name of the issuer's CA public certificate.
`ca_key_file`	The file name of the issuer's CA private key.
`ca_key_password`	The password for the issuer's CA private key.
`selfsigned`	Generates a self-signed certificate that can be used as a trusted CA certificate. If this argument is specified, the *`ca_cert_filename`*, *`ca_key_filename`*, and *`ca_key_password`* arguments should not be specified.
`cert_file`	The name of the generated certificate file.
`private_key_file`	The name of the generated private key file.
`keyfile_pass`	The password for the private key.

Argument	Definition
`key_strength`	The length (in bits) of the keys to be generated. The longer the key, the more difficult it is for someone to break the encryption.
`common_name`	The name to be associated with the generated certificate.
`org_unit`	The name of the organizational unit associated with the generated certificate.
`organization`	The name of the organization associated with the generated certificate.
`locality`	The name of a city or town.
`state`	The name of the state or province in which the organizational unit (ou) operates if your organization is in the United States or Canada, respectively. Do not abbreviate.
`country_code`	Two-letter ISO code for your country. The code for the United States is US.
`subject_key_identifier`	Generates a certificate with the Subject Key identifier extension and the ID value specified on the command line.
`UTF-8|BASE64`	Format of the `subjectkeyid` value. Allowed values are UTF-8 or BASE64, with UTF-8 assumed by default.

2. Use the `ImportPrivateKey` utility to load the digital certificate and private key into a keystore. See ImportPrivateKey in the *WebLogic Server Command Reference*.

If you do not explicitly specify a hostname with the `-cn` option, the CertGen tool uses the JDK `InetAddress.getHostname()` method to get the hostname it puts in the Subject common name. The `getHostName()` method works differently on different platforms. It returns a fully qualified domain name (FQDN) on some platforms (for example, Solaris) and a short host name on other platforms (for example, Windows NT). On Solaris, the result of `InetAddress.getHostname()` depends on how the hosts entry is configured in the `/etc/nsswitch.conf` file.

If WebLogic Server is acting as a client (and by default host name verification is enabled), you need to ensure that the host name specified in the URL matches the Subject common name in the server certificate. Otherwise, connections will fail because the host names do not match.

Using Your Own Certificate Authority

Many companies act as their own certificate authority. To use those trusted CA certificates with WebLogic Server:

1. Ensure the trusted CA certificates are in PEM format.
   - If the trusted CA certificate is in DER format, use the `der2pem` utility to convert them.
   - If the trusted CA certificate was issued by Microsoft, see "Converting a Microsoft p7b Format to PEM Format" on page 10-9.
   - If the trusted CA certificate has a custom file type, use the steps in "Converting a Microsoft p7b Format to PEM Format" on page 10-9 to convert the trusted CA certificate to PEM format.
2. Create a trust keystore. For more information, see "How WebLogic Server Locates Trust" on page 10-14.
3. Store the trusted CA certificate in the trust keystore. For more information, see "How WebLogic Server Locates Trust" on page 10-14.
4. Configure WebLogic Server to use the trust keystore. For more information, see "Configuring Keystores For Production" on page 10-14.

Converting a Microsoft p7b Format to PEM Format

Digital certificates issued by Microsoft are in a format (`p7b`) that cannot be used by WebLogic Server. The following example converts a digital certificate in `p7b` (PKCS#7) format to PEM format on Windows XP:

1. In Windows Explorer, select the file (`*filename*.p7b`) you want to convert. Double-click on the file to display a Certificates window.
2. In the left pane of the Certificates window, expand the file.
3. Expand the Certificates folder to display a list of certificates.
4. Select a certificate to convert to PEM format. Right-click on the certificate, then choose All **Tasks > Export** to display the Certificate Export Wizard.
5. In the wizard, click Next
6. Select the `Base-64 encoded X.509 (.CER)` option. Then click Next. (Base-64 encoded is the PEM format.)

7. In the `File name:` field, enter a name for the converted digital certificate; then click Nest.

Note: The wizard appends a `.cer` extension to the output file The `.cer` extension is a generic extension which is appended to both base-64 encoded certificates and DER certificates. You can change the extension to `.pem` after you exit the wizard.

8. Verify that the settings are correct. If the settings are correct, click Finish; if they are not correct, click Back and make any necessary modifications.

Note: For p7b certificate files that contain certificate chains, you need to concatenate the issuer PEM digital certificates to the certificate file. The resulting certificate file can be used by WebLogic Server.

Obtaining a Digital Certificate for a Web Browser

Low-security browser certificates are easy to acquire and can be done from within the Web browser, usually by selecting the Security menu item in Options or Preferences. Go to the Personal Certificates item and ask to obtain a new digital certificate. You will be asked for some information about yourself.

The digital certificate you receive contains public information, including your name and public key, and additional information you would like authenticated by a third party, such as your E-mail address. Later you will present the digital certificate when authentication is requested.

As part of the process of acquiring a digital certificate, the Web browser generates a public-private key pair. The private key should remain secret. It is stored on the local file system and should never leave the Web browser's machine, to ensure that the process of acquiring a digital certificate is itself safe. With some browsers, the private key can be encrypted using a password, which is not stored. When you encrypt your private key, you will be asked by the Web browser for your password at least once per session.

Note: Digital certificates obtained from Web browsers do not work with other types of Web browsers or on different versions of the same Web browser.

Using Certificate Chains (Deprecated)

Note: The use of file-based certificate chains is deprecated in this release of WebLogic Server. Now the whole certificate chain is imported into a keystore. The steps in this section are provided for the purpose of backward compatibility only.

To use certificate chains with WebLogic Server:

1. Ensure that all the digital certificates are in PEM format. If they are in DER format, you can convert them using the der2pem utility. If you are using a digital certificate issued by

Microsoft, see "Converting a Microsoft p7b Format to PEM Format" on page 10-9. You can use the steps in the section to convert other types of digital certificates. Save the digital certificate in Base 64 format.

2. Open a text editor and include all the digital certificate files into a single file. The order is important. The server digital certificate should be the first digital certificate in the file. The issuer of that digital certificate should be the next in the file and so on until you get to the self-signed root certificate authority certificate. This digital certificate should be the last certificate in the file.

   You cannot have blank lines between digital certificates.

3. Specify the file in the Server Certificate File Name field on the **Configuration > SSL** page in the WebLogic Server Administration Console.

Listing 10-1 shows a sample certificate chain.

Listing 10-1 Sample File with Certificate Chain

```
-----BEGIN CERTIFICATE-----
MIICyzCCAjSgAwIBAgIBLDANBgkqhkiG9w0BAQQFADCBtjELMAkGA1UEBhMCVVMxEzARBgNVBA
gTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNhbiBGcmFuY2lzY28xFTATBgNVBAoTDEJFQSBXZWJM
b2dpYzERMA8GA1UECxMIU2VjdXJpdHkxLzAtBgNVBAMTJkRlbW8gQ2VydGlmaWNhdGUgQXV0aG
9yaXR5IENvbnN0cmFpbnRzMR8wHQYJKoZIhvcNAQkBFhBzZWN1cml0eUBiZWEuY29tMB4XDTAy
MTEwMTIwMDIxMloXDTA2MTAxNTIwMDIxMlowgZ8xCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYW
xpZm9ybmlhMRYwFAYDVQQHEw1TYW4gRnJhbmNpc2NvMRUwEwYDVQQKEwxCRUEgV2ViTG9naWMx
ETAPBgNVBAsTCFNlY3VyaXR5MRkwFwYDVQQDExB3ZWJsb2dpYy5iZWEuY29tMR4wHAYJKoZIhv
cNAQkBFg9zdXBwb3J0QGJlYS5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMJX8nKU
gsFej8pEu/1IVcHUkwY0c2JbBzOryu3sce4QjX+rGxiCjoPm2MY=yts2BvonuJ6CztdZf8B/LB
EWCz+qRrtdFn9mKSZWGvrAkmMPz2RhXEOThpoRo5kZz2FQ9XF/PxIJXTYCM7yooRBwXoKYjquR
wiZNtUiU9kYi6Z3prAgMBAAEwDQYJKoZIhvcNAQEEBQADgYEAh2eqQGxEMUnNTwEUD
0tBq+7YuAkjecEocGXvi2G4YSoWVLgnVzJoJuds3c35KE6sxBe1luJQuQkE9SzALG/6lDIJ5ct
PsHFmZzZxY7scLl6hWj5ON8oN2YTh5Jo/ryqjvnZvqiNIWe/gqr2GLIkajC0mz4un1LiYORPig
3fBMH0=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIC+jCCAmOgAwIBAgIBADANBgkqhkiG9w0BAQQFADCBtjELMAkGA1UEBhMCVVMxEzARBgNVBA
gTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNhbiBGcmFuY2lzY28xFTATBgNVBAoTDEJFQSBXZWJM
b2dpYzERMA8GA1UECxMIU2VjdXJpdHkxLzAtBgNVBAMTJkRlbW8gQ2VydGlmaWNhdGUgQXV0aG
9yaXR5IENvbnN0cmFpbnRzMR8wHQYJKoZIhvcNAQkBFhBzZWN1cml0eUBiZWEuY29tMB4XDTAy
MTEwMTIwMDIxMVoXDTA2MTAxNjIwMDIxMVowgbYxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYW
xpZm9ybmlhMRYwFAYDVQQHEw1TYW4gRnJhbmNpc2NvMRUwEwYDVQQKEwxCRUEgV2ViTG9naWMx
ETAPBgNVBAsTCFNlY3VyaXR5MS8wLQYDVQQDEyZEZW1vIENlcnRpZmljYXRlIEF1dGhvcml0eS
BDb25zdHJhaW50czEfMB0GCSqGSIb3DQEJARYQc2VjdXJpdHlAYmVhLmNvbTCBnzANBgkqhkiG
```

```
9w0BAQEFAAOBjQAwgYkCgYEA3ynD8l5JfLob4g6d94dNtI0Eep6QNl9bblmswnrjIYz1BVjjRj
NVal9fRs+8jvm85kIWlerKzIMJgiNsj50WlXzNX6orszggSsW15pqV0aYE9Re9K
CNNnORlsLjmRhuVxg9rJFEtjHMjrSYr2IDFhcdwPgIt0meWEVnKNObSFYcCAwEAAaMWMBQwEgY
DVR0TAQH/BAgwBgEB/wIBATANBgkqhkiG9w0BAQQFAAOBgQBS+0oqWxGyqbZO028zf9tQT2RKo
jfuwywrDoGW96Un5IqpFnBHIu5atliJo3OUpiH18KkwLN8DVP/3t3K3O3kXdIuLbqAL0i5xyBl
Ahr7gE5eVhIyeMg7ETBPLyGO2BF13Y24LlsO+MX9jW7fxMraPN608QeJXkZw0E0cGwrw2AQ==
-----END CERTIFICATE-----
```

Storing Private Keys, Digital Certificates, and Trusted Certificate Authorities

Once you have obtained private keys, digital certificates, and trusted CA certificates, you need to store them so that WebLogic Server can use them to find and verify identity. Private keys, their associated digital certificates, and trusted CA certificates are stored in keystores. The keystores can be configured through the WebLogic Server Administration Console or specified on the command line. Use the **Configuration > Keystore** page in the WebLogic Server Administration Console to configure identity and trust keystores for WebLogic Server. See Configuring Keystores in the *Administration Console Online Help*.

For the purpose of backward compatibility, private keys and trusted CA certificates can be stored in a file or in a JKS keystore accessed via the WebLogic Keystore provider. In addition, trusted CA certificates can be stored in a JKS keystore. Use the **Configuration > SSL** page of the WebLogic Server Administration Console to specify identity and trust options when using a file or a JKS keystore accessed via the WebLogic Keystore provider.

Guidelines for Using Keystores

When configuring SSL you have to decide how identity and trust will be stored. Although one keystore can be used for both identity and trust, BEA recommends using separate keystores for both identity and trust because the identity keystore (private key/digital certificate pairs) and the trust keystore (trusted CA certificates) may have different security requirements. For example:

- For trust, you only have to put the certificates (non-sensitive data) in the keystore while for identity, you have to put the certificate and private key (sensitive data) in the keystore.
- The identity keystore may be prohibited by company policy from ever being put in the network while the trust keystore can be distributed over the network.
- The identity keystore may be protected by the operating system for both reading and writing by non-authorized users while the trust keystore only needs to be write protected.

- The identity keystore password is generally known to fewer people than the password for the trust keystore.

In general, systems within a domain have the same trust rules (use the same set of trusted CAs), while they tend to have per-server identity. Identity requires a private key, and private keys should not be copied from one system to another. Therefore, you should maintain separate identity keystores for each system, each keystore containing only the server identity needed for that system. However, trust keystores can be copied from system to system; thus making it easier to standardize trust rules.

Identity is more likely to be stored in hardware keystores such as nCipher. Trust can be stored in a file-based JDK keystore without having security issues because a trust store contains only certificates, not private keys.

Creating a Keystore and Loading Private Keys and Trusted Certificate Authorities into the Keystore

A keystore is for the secure storage and management of private keys/digital certificate pairs and trusted CA certificates. Use the following mechanisms to create a keystore and load private keys and trusted CA certificates into the keystore:

- The WebLogic `ImportPrivateKey` utility. The `ImportPrivateKey` utility allows you to take private key and digital certificate files and load them into a keystore. For more information, see ImportPrivateKey in the *WebLogic Server Command Reference*.
- Sun Microsystem's `keytool` utility. Use the `keytool` utility to generate a private key/digital certificate pair and then import the signed private key into the keystore. For more information, see "How WebLogic Server Locates Trust" on page 10-14. While you can use the `keytool` utility to generate new private keys and digital certificates and add them to a keystore, the utility does not allow you to take an existing private key from a file and import it into the keystore. Instead, use the WebLogic `ImportPrivateKey` utility.

  Note: The `keytool` utility does allow you to import trusted CA certificates from a file into a keystore.

- Custom utilities. This release of WebLogic Server can use keystores created with custom tools or utilities. How to create and use these utilities is outside the scope of this document.

All private key entries in a keystore are accessed by WebLogic Server via unique aliases. You specify the alias when loading the private key into the keystore. Aliases are case-insensitive; the aliases `Hugo` and `hugo` would refer to the same keystore entry. Aliases for private keys are specified in the Private Key Alias field on the **Configuration > SSL** page in the WebLogic Server

Administration Console. Although WebLogic Server does not use the alias to access trusted CA certificates, the keystore does require an alias when loading a trusted CA certificate into the keystore.

All certificate authorities in a keystore identified as trusted by WebLogic Server are trusted.

How WebLogic Server Locates Trust

WebLogic Server uses the following algorithm when it loads its trusted CA certificates:

1. If the keystore is specified by the `-Dweblogic.security.SSL.trustedCAkeystore` command-line argument, load the trusted CA certificates from that keystore.
2. Else if the keystore is specified in the configuration file (`config.xml`), load trusted CA certificates from the specified keystore. If the server is configured with DemoTrust, trusted CA certificates will be loaded from the `WL_HOME\server\lib\DemoTrust.jks` and the JDK `cacerts` keystores.
3. Else if the trusted CA file is specified in the configuration file (`config.xml`), load trusted CA certificates from that file (this is only for compatibility with 6.x SSL configurations).
4. Else load trusted CA certificates from `WL_HOME\server\lib\cacerts` keystore.

Configuring Keystores For Production

By default, WebLogic Server is configured with two keystores:

- `DemoIdentity.jks`—Contains a demonstration private key for WebLogic Server. This keystore contains the identity for WebLogic Server.
- `DemoTrust.jks`—Contains the trusted certificate authorities from the `WL_HOME\server\lib\DemoTrust.jks` and the JDK `cacerts` keystores. This keystore establishes trust for WebLogic Server.

These keystores are located in the *`WL_HOME`*`\server\lib` directory and the *`JAVA_HOME`*`\jre\lib\security` directory. For testing and development purposes, the keystore configuration is complete. However, do not use the demonstration keystores in a production environment. Because the digital certificates and trusted CA certificates in the demonstration keystores are signed by a WebLogic Server demonstration certificate authority, a WebLogic Server installation using the demonstration keystores will trust **any** WebLogic Server installation that also uses the demonstration keystores. You want to create a secure environment where only your installations trust each other.

To configure keystores for use in a production environment:

1. Obtain private keys and digital certificates from a reputable certificate authority such as Verisign, Inc. or Entrust.net. For more information, see "Obtaining Private Keys, Digital Certificates, and Trusted Certificate Authorities" on page 10-4.
2. Create identity and trust keystores. For more information, see "Creating a Keystore and Loading Private Keys and Trusted Certificate Authorities into the Keystore" on page 10-13.
3. Load the private keys and trusted CAs into the identity and trust keystores. For more information, see "Creating a Keystore and Loading Private Keys and Trusted Certificate Authorities into the Keystore" on page 10-13.
4. Use the WebLogic Server Administration Console to configure the identity and trust keystores. See "Configure Keystores" in the *Administration Console Online Help*.

You can also use the WebLogic Scripting Tool or Java Management Extensions (JMX) APIs to create a new security configuration. For more information see the *WebLogic Scripting Tool* and *Developing Custom Management Utilities with JMX* manuals.

Configuring SSL

Configuring SSL is an optional step; however, BEA recommends using SSL in a production environment. The following sections describe how to configure SSL for WebLogic Server:

Notes: This chapter applies to WebLogic Server deployments using the security features in this release of WebLogic Server as well as deployments using Compatibility Security.

All machines must be kept up to date with the current set of recommended patches from the operating system vendors.

SSL: An Introduction

Secure Sockets Layer (SSL) provides secure connections by allowing two applications connecting over a network connection to authenticate the other's identity and by encrypting the data exchanged between the applications. Authentication allows a server and optionally a client to verify the identity of the application on the other end of a network connection. Encryption makes data transmitted over the network intelligible only to the intended recipient.

SSL in WebLogic Server is an implementation of the SSL 3.0 and Transport Layer Security (TLS) 1.0 specifications.

WebLogic Server supports SSL on a dedicated listen port which defaults to 7002. To establish an SSL connection, a Web browser connects to WebLogic Server by supplying the SSL listen port and the HTTPs protocol in the connection URL, for example, `https://myserver:7002`.

Using SSL is computationally intensive and adds overhead to a connection. Avoid using SSL in development environments when it is not necessary. However, always use SSL in a production environment.

One-Way and Two-Way SSL

SSL can be configured one-way or two-way:

- With one-way SSL, the server is required to present a certificate to the client but the client is not required to present a certificate to the server. To successfully negotiate an SSL connection, the client must authenticate the server, but the server will accept a connection from any client. One-way SSL is common on the Internet where customers want to create secure connections before they share personal data. Often, clients will also use SSL to log on in order for the server can authenticate them.
- With two-way SSL, the server presents a certificate to the client and the client presents a certificate to the server. WebLogic Server can be configured to require clients to submit valid and trusted certificates before completing the SSL connection.

Setting Up SSL: Main Steps

To set up SSL:

1. Obtain an identity (private key and digital certificates) and trust (certificates of trusted certificate authorities) for WebLogic Server. Use the digital certificates, private keys, and trusted CA certificates provided by the WebLogic Server kit, the CertGen utility, Sun

Microsystem's `keytool` utility, or a reputable vendor such as Entrust or Verisign to perform this step.

2. Store the identity and trust. Private keys and trusted CA certificates which specify identity and trust are stored in a keystore.

   Note: This release of WebLogic Server supports private keys and trusted CA certificates stored in files, or in the WebLogic Keystore provider for the purpose of backward compatibility only.

3. Configure the identity and trust keystores for WebLogic Server in the WebLogic Server Administration Console. See "Configure Keystores" in the *Administration Console Online Help*.

4. Set SSL configuration options for the private key alias and password in the WebLogic Server Administration Console. Optionally, set configuration options that require the presentation of client certificates (for two-way SSL). See "Configure SSL" and "Configure two-way SSL" in the *Administration Console Online Help*.

Note: When starting a WebLogic Server instance, you can specify the command line argument `-Dweblogic.security.ssl.nojce=true` to use a FIPS-compliant (FIPS 140-2) crypto module.

For information on configuring identity and trust for WebLogic Server, see "Obtaining Private Keys, Digital Certificates, and Trusted Certificate Authorities" on page 10-4 and "Storing Private Keys, Digital Certificates, and Trusted Certificate Authorities" on page 10-12.

Using Host Name Verification

A host name verifier ensures the host name in the URL to which the client connects matches the host name in the digital certificate that the server sends back as part of the SSL connection. A host name verifier is useful when an SSL client (or a WebLogic Server acting as an SSL client) connects to an application server on a remote host. It helps to prevent man-in-the-middle attacks.

By default, WebLogic Server has host name verification enabled. As a function of the SSL handshake, WebLogic Server compares the common name in the SubjectDN in the SSL server's digital certificate with the host name of the SSL server used to initiate the SSL connection. If these names do not match, the SSL connection is dropped. The SSL client is the actual party that drops the SSL connection if the names do not match.

If anything other than the default behavior is desired, either turn off host name verification or configure a custom host name verifier. Turning off host name verification leaves WebLogic

Server vulnerable to man-in-the-middle attacks. BEA recommends leaving host name verification on in production environments.

In this release of WebLogic Server, the host name verification feature is updated so that if the host name in the certificate matches the local machine's host name, host name verification passes if the URL specifies `localhost`, `127.0.01`, or the default IP address of the local machine.

For more information, see the following topics in the *Administration Console Online Help*:

- "Verify Host Name Verification is enabled"
- "Disable Host Name Verification"
- "Configure a Custom Host Name Verifier"
- "Configuring SSL"

Enabling SSL Debugging

SSL debugging provides more detailed information about the SSL events that occurred during an SSL handshake. The SSL debug trace displays information about:

- Trusted certificate authorities
- SSL server configuration information
- Server identity (private key and digital certificate)
- The encryption strength that is allowed by the license in use
- Enabled ciphers
- SSL records that were passed during the SSL handshake
- SSL failures detected by WebLogic Server (for example, trust and validity checks and the default host name verifier)
- I/O related information

Use the following command-line properties to enable SSL debugging:

```
-Dssl.debug=true -Dweblogic.StdoutDebugEnabled=true
```

The SSL debugging properties can be included in the start script of the SSL server, the SSL client, and the Node Manager. For a Managed Server started by the Node Manager, specify this command-line argument on the Remote Start page for the Managed Server.

SSL debugging dumps a stack trace whenever an ALERT is created in the SSL process. The types and severity of the ALERTS are defined by the TLS specification.

The stack trace dumps information into the log file where the ALERT originated. Therefore, when tracking an SSL problem, you may need to enable debugging on both sides of the SSL connection (on both the SSL client or the SSL server). The log file contains detailed information about where the failure occurred. To determine where the ALERT occurred, confirm whether there is a trace message after the ALERT. An ALERT received after the trace message indicates the failure occurred on the peer. To determine the problem, you need to enable SSL debugging on the peer in the SSL connection.

When tracking an SSL problem, review the information in the log file to ensure:

- The correct `config.xml` file was loaded
- The license (domestic or export) is correct
- The trusted certificate authority was valid and correct for this server.
- The host name check was successful
- The certificate validation was successful

Note: Sev 1 type 0 is a normal close ALERT, not a problem.

SSL Session Behavior

WebLogic Server allows SSL sessions to be cached. Those sessions live for the life of the server.

Clients that use SSL sockets directly can control the SSL session cache behavior. The SSL session cache is specific to each SSL context. All SSL sockets created by SSL socket factory instances returned by a particular SSL context can share the SSL sessions.

Clients default to resuming sessions at the same IP address and port. Multiple SSL sockets that use the same host and port share SSL sessions by default assuming the SSL sockets are using the same underlying SSL context.

Clients that do not want to use SSL sessions must call `setEnableSessionCreation(false)` on the SSL socket to ensure that no SSL sessions are cached. This setting only controls whether an SSL session is added to the cache, it does not stop an SSL socket from finding an SSL session that was already cached (for example, SSL socket 1 caches the session, SSL socket 2 sets `setEnableSessionCreation` to `false` but it can still reuse the SSL session from SSL socket 1 since that session was put in the cache.)

SSL sessions exist for the lifetime of the SSL context; they are not controlled by the lifetime of the SSL socket. Therefore, creating a new SSL socket and connecting to the same host and port can resume a previous session as long as the SSL socket is created using an SSL socket factory from the SSL context that has the SSL session in its cache.

By default, clients that use HTTPS URLs get a new SSL session for each URL because each URL uses a different SSL context and therefore SSL sessions can not be shared or reused. The SSL session can be retrieved using the `weblogic.net.http.HttpsClient` class or the `weblogic.net.http.HttpsURLConnection` class. Clients can also resume URLs by sharing a SSLSocket Factory between them.

Session caching is maintained by the SSL context, which can be shared by threads. A single thread has access to the entire session cache, not just one SSL session, so multiple SSL sessions can be used and shared in a single (or multiple) thread.

The following command-line arguments are ignored:

- `weblogic.security.SSL.sessionCache.size`
- `weblogic.security.SSL.sessionCache.ttl`

Configuring RMI over IIOP with SSL

Use SSL to protect Internet Interop-Orb-Protocol (IIOP) connections to Remote Method Invocation (RMI) remote objects. SSL secures connections through authentication and encrypts the data exchanged between objects.

To use SSL to protect RMI over IIOP connections, do the following:

1. Configure WebLogic Server to use SSL.
2. Configure the client Object Request Broker (ORB) to use SSL. Refer to the product documentation for your client ORB for information about configuring SSL.
3. Use the `host2ior` utility to print the WebLogic Server IOR to the console. The `host2ior` utility prints two versions of the interoperable object reference (IOR), one for SSL connections and one for non-SSL connections. The header of the IOR specifies whether or not the IOR can be used for SSL connections.
4. Use the SSL IOR when obtaining the initial reference to the CosNaming service that accesses the WebLogic Server JNDI tree.

For more information about using RMI over IIOP, see *Programming WebLogic RMI*.

SSL Certificate Validation

WebLogic Server ensures that each certificate in a certificate chain was issued by a certificate authority. All X509 V3 CA certificates used with WebLogic Server must have the Basic Constraint extension defined as CA, thus ensuring that all certificates in a certificate chain were issued by a certificate authority. By default, any certificates for certificate authorities not meeting this criteria are rejected. This section describes the command-line argument that controls the level of certificate validation.

Note: If WebLogic Server is booted with a certificate chain that will not pass the certificate validation, an information message is logged noting that clients could reject it.

Controlling the Level of Certificate Validation

By default WebLogic Server rejects any certificates in a certificate chain that do not have the Basic Constraint extension defined as CA. However, you may be using certificates that do not meet this requirement or you may want to increase the level of security to conform to the IETF RFC 2459 standard. Use the following command-line argument to control the level of certificate validation performed by WebLogic Server:

```
-Dweblogic.security.SSL.enforceConstraints=option
```

Table 11-1 describes the options for the command-line argument.

Table 11-1 Options for -Dweblogic.security.SSL.enforceConstraints

Option	Description

`strong` or `true`	Use this option to check that the Basic Constraints extension on the CA certificate is defined as CA.
	For example:
	`-Dweblogic.security.SSL.enforceConstraints=strong`
	or
	`-Dweblogic.security.SSL.enforceConstraints=true`
	By default, WebLogic Server performs this level of certificate validation.

`strict`	Use this option to check the Basic Constraints extension on the CA certificate is defined as CA and set to critical. This option enforces the IETF RFC 2459 standard. For example: `-Dweblogic.security.SSL.enforceConstraints=strict` This option is not the default because a number of commercially available CA certificates do not conform to the IETF RFC 2459 standard.
`off`	Use this option to turn off checking for the Basic Constraints extension. The rest of the certificate is still validated. CA certificates from most commercial certificate authorities should work with the default `strong` option. For example: `-Dweblogic.security.SSL.enforceConstraints=off` BEA does not recommend using this option in a production environment. Instead, purchase new CA certificates that comply with the IETF RFC 2459 standard.

Checking Certificate Chains

WebLogic Server provides a `ValidateCertChain` command-line utility to check whether or not an existing certificate chain will be rejected by WebLogic Server. The utility uses certificate chains from PEM files, PKCS-12 files, PKCS-12 keystores, and JKS keystores. A complete certificate chain must be used with the utility. The following is the syntax for the ValidateCertChain command-line utility:

```
java utils.ValidateCertChain -file pemcertificatefilename
java utils.ValidateCertChain -pem pemcertificatefilename
java utils.ValidateCertChain -pkcs12store pkcs12storefilename
java utils.ValidateCertChain -pkcs12file pkcs12filename password
java utils.ValidateCertChain -jks alias storefilename [storePass]
```

Example of valid certificate chain:

```
java utils.ValidateCertChain -pem zippychain.pem

Cert[0]: CN=zippy,OU=FOR TESTING
ONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US

Cert[1]: CN=CertGenCAB,OU=FOR TESTING
ONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US

Certificate chain appears valid
```

Example of invalid certificate chain:

```
java utils.ValidateCertChain -jks mykey mykeystore

Cert[0]: CN=corba1,OU=FOR TESTING ONLY,
O=MyOrganization,L=MyTown,ST=MyState,C=US

CA cert not marked with critical BasicConstraint indicating it is a CA
Cert[1]: CN=CACERT,OU=FOR TESTING ONLY,
O=MyOrganization,L=MyTown,ST=MyState,C=US

Certificate chain is invalid
```

Troubleshooting Problems with Certificate Validation

If SSL communications were working properly in a previous release of WebLogic Server and start failing unexpectedly, the problem is mostly likely because the certificate chain used by WebLogic Server is failing the validation.

Determine where the certificate chain is being rejected, and decide whether to update the certificate chain with one that will be accepted or change the setting of the `-Dweblogic.security.SSL.enforceConstraints` command-line argument.

To troubleshoot problems with certificates, use one of the following methods:

- If you know where the certificate chains for the processes using SSL communication are located, use the ValidateCertChain command-line utility to check whether the certificate chains will be accepted.
- Turn on SSL debug tracing on the processes using SSL communication. The syntax for SSL debug tracing is:

  ```
  -Dssl.debug=true -Dweblogic.StdoutDebugEnabled=true
  ```

  The following message indicates the SSL failure is due to problems in the certificate chain:

  ```
  <CA certificate rejected. The basic constraints for a CA certificate
  were not marked for being a CA, or were not marked as critical>
  ```

  When using one-way SSL, look for this error in the client log. When using two-way SSL, look for this error in the client and server logs.

Enabling SSL Debugging

SSL debugging provides more detailed information about the SSL events that occurred during an SSL handshake. The SSL debug trace displays information about:

- Trusted certificate authorities
- SSL server configuration information
- Server identity (private key and digital certificate)
- The encryption strength that is allowed by the license in use
- Enabled ciphers
- SSL records that were passed during the SSL handshake
- SSL failures detected by WebLogic Server (for example, trust and validity checks and the default host name verifier)
- I/O related information

Use the following command-line properties to enable SSL debugging:

```
-Dssl.debug=true -Dweblogic.StdoutDebugEnabled=true
```

The SSL debugging properties can be included in the start script of the SSL server, the SSL client, and the Node Manager. For a Managed Server started by the Node Manager, specify this command-line argument on the Remote Start page for the Managed Server.

SSL debugging dumps a stack trace whenever an ALERT is created in the SSL process. The types and severity of the ALERTS are defined by the TLS specification.

The stack trace dumps information into the log file where the ALERT originated. Therefore, when tracking an SSL problem, you may need to enable debugging on both sides of the SSL connection (on both the SSL client or the SSL server). The log file contains detailed information about where the failure occurred. To determine where the ALERT occurred, confirm whether there is a trace message after the ALERT. An ALERT received after the trace message indicates the failure occurred on the peer. To determine the problem, you need to enable SSL debugging on the peer in the SSL connection.

When tracking an SSL problem, review the information in the log file to ensure:

- The correct `config.xml` file was loaded
- The license (domestic or export) is correct
- The trusted certificate authority was valid and correct for this server.
- The host name check was successful
- The certificate validation was successful

Note: Sev 1 type 0 is a normal close ALERT, not a problem.

Using Certificate Lookup and Validation Providers

WebLogic Server SSL has built-in certificate validation, which given a set of trusted CAs:

- Verifies that the last certificate in the chain is either a trusted CA or is issued by a trusted CA.
- Completes the certificate chain with trusted CAs.
- Verifies the signatures in the chain.
- Ensures that the chain has not expired.

You can use certificate lookup and validation (CLV) providers to perform additional validation on the certificate chain. In this release, WebLogic Server has added two CLV providers:

- WebLogic CertPath Provider—Completes certificate paths and validates certificates using the trusted CA configured for a particular server instance, providing the same functionality as the built-in SSL certificate validation. This is configured by default.

- Certificate Registry—The system administrator makes a list of trusted CA certificates that are allowed access to the server; a certificate is valid if the end certificate is in the registry. The administrator revokes a certificate by removing it from the certificate registry providing an inexpensive mechanism for performing revocation checking. This is not configured by default.

Alternatively, you can write a custom CertPathValidator to provide additional validation on the certificate chain. For more information, see the Configuring WebLogic Security Providers chapter.

Outbound SSL and two-way inbound SSL in a WebLogic Server instance receive certificate chains during the SSL handshake that must be validated. An example of two-way inbound SSL is a browser connecting to a Web application over HTTPS where the browser sends the client's certificate chain to the Web application. The inbound certificate validation setting is used for all two-way client certificate validation in the server.

Examples of WebLogic Server using outbound SSL (that is, acting as an SSL client) include:

- Connecting to the Node Manager
- Connecting to another WebLogic Server over the administration port
- Connecting to an external LDAP server, such as the LDAPAuthenticator

Using the Administration Console or WLST, you can independently configure inbound and outbound SSL certificate validation using these `SSLMBean` attributes: `InboundCertificateValidation` and `OutboundCertificateValidation`.

Legal values for both attributes are:

- `BUILTIN_SSL_VALIDATION`: Use the built-in SSL certificate validation code to complete and validate the certificate chain. That is, configure SSL to work as has in previous releases. This is the default behavior.

- `BUILTIN_SSL_VALIDATION_AND_CERT_PATH_VALIDATORS`: Use the built-in trusted CA-based validation and the configured CertPathValidator providers to perform additional validation. That is, configure SSL to work as has in previous releases plus do extra validation.

For more information, see:

- `SSLMBean` in the *WebLogic Server MBean Reference*
- "Set Up SSL" in the *Administration Console Online Help*

Using the nCipher JCE Provider with WebLogic Server

Note: Java Cryptography Extension (JCE) providers are written using the application programming interfaces (APIs) in the JCE available in JDK 5.0. This type of provider is different from the providers written using the WebLogic Security Service Provider Interfaces (SSPIs). WebLogic Server does not provide a JCE provider by default.

SSL is a key component in the protection of resources available in Web servers. However, heavy SSL traffic can cause bottlenecks that impact the performance of Web servers. JCE providers like nCipher which use a hardware card for encryption, offload SSL processing from Web servers freeing the servers to process more transactions. They also provide strong encryption and cryptographic processes to preserve the integrity and secrecy of keys.

WebLogic Server supports the use of the following JCE providers:

- The JDK JCE provider (`SunJCE`) in the JDK 5.0. For more information about the features in the JDK JCE provider, see http://java.sun.com/products/jce.

  By default, the JCE provider in the JDK 5.0 has export strength jurisdiction policy files. After filling out the appropriate forms, the domestic strength jurisdiction policy files are downloadable from Sun Microsystems at http://java.sun.com/products/jce/javase.html#UnlimitedDownload.

  The BEA license will continue to control the strength of the cryptography used by the WebLogic Server Application Programming Interfaces (APIs). Client code without the appropriate domestic strength cryptography license will only be able to use the J2SE export strength default cryptography. On the server, there will always be a BEA license that will enable either export or domestic strength cryptography.

- The nCipher JCE provider. For more information about the nCipher JCE provider, see http://www.ncipher.com/solutions/sslsecurity.html.

To install the nCipher JCE provider:

1. Install and configure the hardware for the nCipher JCE provider per the product's documentation.

2. Install the files for the nCipher JCE provider. The following files are required:

   – Jurisdiction policy files—The JDK installs these files by default but they are of limited export strength.

   – Certificate that signed the JAR file

Note: This step may have been performed as part of installing the hardware for nCipher JCE provider. In that case, verify that the files are correctly installed.

– The JCE provider JAR files

The files are installed in one of the following ways:

– As an installed extension. Copy the files to one of the following locations:

```
JAVA_HOME/jre/lib/ext
```

For example:

```
BEA_HOME/jdk150_03/jre/lib/ext
```

– In the CLASSPATH of the server.

3. Edit the Java security properties file (`java.security`) to add the nCipher JCE provider to the list of approved JCE providers for WebLogic Server. The Java security properties file is located in:

   ```
   JAVA_HOME/jre/lib/security/java.security
   ```

   Specify the nCipher JCE provider as:

   ```
   security.provider.n=com.ncipher.provider.km.mCipherKM
   ```

   where

   n specifies the preference order that determines the order in which providers are searched for requested algorithms when no specific provider is requested. The order is 1-based; 1 is the most preferred, followed by 2, and so on.

   The nCipher JCE provider must follow the RSA JCA provider in the security properties file. For example:

   ```
   security.provider.1=sun.security.provider.Sun
   security.provider.2=com.sun.rsajca.Provider
   security.provider.3=com.ncipher.provider.km.mCipherKM
   ```

4. Boot WebLogic Server.

5. To ensure the nCipher JCE provider is working properly, enable debugging according to the nCipher product documentation.

Specifying the Version of the SSL Protocol

WebLogic Server supports both the SSL V3.0 and TLS V1.0 protocols. When WebLogic Server is acting as an SSL server, it will agree to use whichever of these protocols the client specifies as preferred in its client hello message. When WebLogic Server is acting as an SSL client, it will specify TLS1.0 as the preferred protocol in its SSL V2.0 client hello message, but will agree to SSL V3.0 as well, if that is the highest version that the SSL server on the other end supports. The peer must respond with an SSL V3.0 or TLS V1.0 message or the SSL connection is dropped.

While in most cases the SSL V3.0 protocol is acceptable there are circumstances (compatibility, SSL performance, and environments with maximum security requirements) where the TLS V1.0 protocol is desired. The `weblogic.security.SSL.protocolVersion` command-line argument lets you specify which protocol is used for SSL connections.

Note: The SSL V3.0 and TLS V1.0 protocols can not be interchanged. Only use the TLS V1.0 protocol if you are certain all desired SSL clients are capable of using the protocol.

The following command-line argument can be specified so that WebLogic Server supports only SSL V3.0 or TLS V1.0 connections:

- `-Dweblogic.security.SSL.protocolVersion=SSL3`—Only SSL V3.0 messages are sent and accepted.
- `-Dweblogic.security.SSL.protocolVersion=TLS1`—Only TLS V1.0 messages are sent and accepted.
- `-Dweblogic.security.SSL.protocolVersion=ALL`—This is the default behavior.

Configuring Security for a WebLogic Domain

The following sections describe how to set security configuration options for a WebLogic domain:

Note: This section applies to WebLogic Server deployments using the security features in this release of WebLogic Server as well as deployments using Compatibility Security.

Enabling Trust Between WebLogic Server Domains

Note: Enabling trust between WebLogic Server domains opens the servers up to man-in-the-middle attacks. Great care should be taken when enabling trust in a production environment. BEA recommends having strong network security such as a dedicated communication channel or protection by a strong firewall.

Trust between domains is established so that principals in a Subject from one WebLogic Server domain are accepted as principals in another domain. When this feature is enabled, identity is passed between WebLogic Server domains over an RMI connection without requiring authentication in the second domain (for example: login to Domain 1 as Joe, make an RMI call to Domain 2 and Joe is still authenticated). When inter-domain trust is enabled, transactions can commit across domains. A trust relationship is established when the Domain Credential for one domain matches the Domain Credential for another domain.

The domain credential is randomly created the first time a WebLogic Server domain is started. This process ensures that by default no two WebLogic Server domains have the same credential. To enable trust between two WebLogic Server domains, you must explicitly specify the same value for the credential in both WebLogic Server domains. Use the configuration options on the Security: Advanced page under the Domains node to set domain credentials.

By default, when you boot an Administration Server for the first time, the Domain Credential is not defined. As the Administration Server boots, it notices that the Domain Credential is not defined and generates a random credential. To enable trust between two WebLogic Server domains, for each domain, prevent the generation of a random credential by:

1. Unchecking Enable Generated Credential.
2. Enter a credential in the Credential and Confirm Credential fields, using the same credential for each domain.

WebLogic Server signs Principals with the Domain Credential as Principals are created. When a Subject is received from a remote source, its Principals are validated (the signature is recreated and if it matches, the remote domain has the same Domain Credential). If validation fails, an error is generated. If validation succeeds, the Principals are trusted as if they were created locally.

Note: Any credentials in clear text are encrypted the next time the `config.xml` file is persisted to disk.

If you want a WebLogic Server 6.x domain to interoperate with a WebLogic Server domain, change Domain Credential in the WebLogic Server domain to the password of the `system` user in the WebLogic Server 6.x domain.

If you are enabling trust between domains in a managed server environment, you must stop the Administration server and all the Managed Servers in both domains and then restart them. If this step is not performed, servers that were not rebooted will not trust the servers that were rebooted.

Keep the following points in mind when enabling trust between WebLogic Server domains:

- Because a domain will trust remote Principals without requiring authentication, it is possible to have authenticated users in a domain that are not defined in the domain's authentication database. This situation can cause authorization problems.

- Any authenticated user in a domain can access any other domain that has trust enabled with the original domain without re-authenticating. There is no auditing of this login and group membership is not validated. Therefore, if Joe is a member of the Administrators group in the original domain where he authenticated, he is automatically a member of the Administrators group for all trusted domains to which he makes RMI calls.
- If Domain 2 trusts both Domain 1 and Domain 3, Domain 1 and Domain 3 now implicitly trust each other. Therefore, members of the Administrators Group in Domain 1 are members of the Administrators group in Domain 3. This may not be a desired trust relationship.
- If you extended the WLSUser and WLSGroup Principal classes, the custom Principal classes must be installed in the server's classpath in all domains that share trust.

For more information, see "Enable trust between domains" in the Administration Console online help.

Note: You can also use the WebLogic Scripting tool or Java Management Extensions (JMX) APIs to modify your security configuration.

Using Connection Filters

Connection filters allow you to deny access at the network level. They can be used to protect server resources on individual servers, server clusters, or an entire internal network or intranet. For example, you can deny any non-SSL connections originating outside of your corporate network. Network connection filters are a type of firewall in that they can be configured to filter on protocols, IP addresses, and DNS node names.

Connection filters are particularly useful when using the Administration port. Depending on your network firewall configuration, you may be able to use a connection filter to further restrict administration access. A typical use might be to restrict access to the Administration port to only the servers and machines in the domain. An attacker who gets access to a machine inside the firewall, still cannot perform administration operations unless the attacker is on one of the permitted machines.

WebLogic Server provides a default connection filter called `weblogic.security.net.ConnectionFilterImpl`. This connection filter accepts all incoming connections and also provides static factory methods that allow the server to obtain the current connection filter. To configure this connection filter to deny access, simply enter the connection filters rules in the WebLogic Administration Console.

You can also use a custom connection filter by implementing the classes in the `weblogic.security.net` package. For information about writing a connection filter, see Using

Network Connection Filters in *Programming WebLogic Security*. Like the default connection filter, custom connection filters are configured in the WebLogic Administration Console.

To configure a connection filter:

1. Enable the logging of accepted messages. This Connection Logger Enabled option logs successful connections and connection data in the server. This information can be used to debug problems relating to server connections.
2. Choose which connection filter is to be used in the domain.
   - To configure the default connection filter, specify `weblogic.security.net.ConnectionFilterImpl` in Connection Filter.
   - To configure a custom connection filter, specify the class that implements the network connection filter in Connection Filter. This class must also be specified in the CLASSPATH for WebLogic Server.
3. Enter the syntax for the connection filter rules.

For more information:

- See "Configure connection filtering" in the Administration Console online help.
- For information about connection filter rules and writing a custom connection filter, see "Using Network Connection Filters" and "Developing Custom Connection Filters" in *Programming WebLogic Security.*
- You can also use the WebLogic Scripting tool or Java Management Extensions (JMX) APIs to create a new security configuration.

Using the Java Authorization Contract for Containers

The Java Authorization Contract for Containers (JACC) Standard can replace the EJB and Servlet container deployment and authorization provided by WebLogic Server. When you configure a WebLogic Server domain to use JACC, EJB and servlet authorization decisions are made by the classes in the JACC framework. All other authorization decisions within WebLogic Server are still determined by the WebLogic Security Framework. For information about the WebLogic JACC provider, see Using the Java Authorization Contract for Containers in *Programming WebLogic Security*.

You configure WebLogic Server to use JACC with a command line start option. For more information, see the description of the `-Djava.security.manager` option in the weblogic.Server Command-Line Reference.

Note that an Administration Server and all Managed Servers in a domain need to have the same JACC configuration. If you change the JACC setting on the Administration Server, you should shut down the Managed Server and reboot them with the same settings as the Administration Server to avoid creating a security vulnerability. Otherwise, it may appear that EJBs and servlets in your domain are protected by WebLogic Security Framework roles and policies, when in fact the Managed Servers are still operating under JACC.

Viewing MBean Attributes

The Anonymous Admin Lookup Enabled option specifies whether anonymous, read-only access to WebLogic Server MBeans should be allowed from the `MBean` API. With this anonymous access, you can see the value of any MBean attribute that is not explicitly marked as protected by the Weblogic Server MBean authorization process. This option is checked by default to sure backward compatibility.

To verify the setting of the Anonymous Admin Lookup Enabled option through the WebLogic Administration Console, see the Domain: Security: General page in the Administration Console or the `SecurityConfigurationMBean.AnonymousAdminLookupEnabled` attribute.

How Passwords are Protected in WebLogic Server

It is important to protect passwords that are used to access resources in a WebLogic Server domain. In the past, usernames and passwords were stored in clear text in a WebLogic security realm. Now all the passwords in a WebLogic Server domain are hashed. The `SerializedSystemIni.dat` file contains the hashes for the passwords. It is associated with a specific WebLogic Server domain so it cannot be moved from domain to domain.

If the `SerializedSystemIni.dat` file is destroyed or corrupted, you must reconfigure the WebLogic Server domain. Therefore, you should take the following precautions:

- Make a backup copy of the `SerializedSystemIni.dat` file and put it in a safe location.
- Set permissions on the `SerializedSystemIni.dat` file such that the system administrator of a WebLogic Server deployment has write and read privileges and no other users have any privileges.

Protecting User Accounts

WebLogic Server defines a set of configuration options to protect user accounts from intruders. In the default security configuration, these options are set for maximum protection. You can use the Administration Console to modify these options on the Configuration: User Lockout page.

As a system administrator, you have the option of turning off all the configuration options, increasing the number of login attempts before a user account is locked, increasing the time period in which invalid login attempts are made before locking the user account, and changing the amount of time a user account is locked. Remember that changing the configuration options lessens security and leaves user accounts vulnerable to security attacks. See “Set user lockout attributes” in the console online help.

Notes: The User Lockout options apply to the default security realm and all its security providers. The User Lockout options do not work with custom security providers in a security realm other than the default security realm. To use the User Lockout options with custom security providers, configure the custom security providers in the default security realm. Include the customer providers in the authentication process after the WebLogic Authentication provider and the WebLogic Identity Assertion provider. This ordering may cause a small performance hit.

If you are using an Authentication provider that has its own mechanism for protecting user accounts, disable Lockout Enabled.

If a user account becomes locked and you delete the user account and add another user account with the same name and password, the User Lockout configuration options will not be reset.

For information about unlocking a locked user account on the Administration Server, see “Unlock a user account” in the console online help. Unlocking a locked user account on a Managed Server cannot be done through the WebLogic Administration Console. The unlock information is propagated through a multicast message which is only configured in a cluster environment. Use the following command instead:

```
java weblogic.Admin -url url -username adminuser
-password passwordforadminuser -type
weblogic.mangement.security.authentication.UserLockoutManager -method
clearLockout lockedusername
```

You can also wait the time specified in the Lockout Duration attribute: the user account will be unlocked after the specified time.

Using Compatibility Security

Compatibility security refers to the capability to run security configurations developed under WebLogic Server 6.x in this release of WebLogic Server. In Compatibility security, you manage 6.x security realms, users, groups, and ACLs, protect user accounts, and configure the Realm Adapter Auditing provider and optionally the Identity Assertion provider in the Realm Adapter Authentication provider. The following sections describe how to configure Compatibility security:

Note: Compatibility security is deprecated in this release of WebLogic Server and will not be supported in future major releases. BEA strongly recommends upgrading your WebLogic Server deployment to the security features in this release of WebLogic Server. You should only use Compatibility security pending such an upgrade.

Running Compatibility Security: Main Steps

To set up Compatibility security:

1. Make a backup copy of your 6.x WebLogic domain (including your `config.xml` file) before using Compatibility security.
2. Add the following to the 6.x `config.xml` file if it does not exist, replacing the values with the actual names of your domain, security realm, and `FileRealm`:

   ```
   <Security Name="mydomain" Realm="mysecurity"/>
   <Realm Name="mysecurity" FileRealm="myrealm"/>
   <FileRealm Name="myrealm"/>
   ```

3. Install WebLogic Server in a new directory location. Do not overwrite your existing 6.x installation directory. For more information, see *Installation Guide*.
4. Modify the start script for your 6.x server to point to the new WebLogic Server installation. Specifically, you need to modify:
   – The classpath to point to the `weblogic.jar` file in the new WebLogic Server installation.
   – `The JAVA_HOME` variable to point to the new WebLogic Server installation.
5. Use the start script for your 6.x server to boot WebLogic Server.

To verify whether you are correctly running Compatibility security, open the WebLogic Administration Console. If you are running Compatibility security, a Compatibility Security node will appear in the Domain Structure pane at the left side of the WebLogic Administration Console.

Compatibility Security MBeans

All Compatibility security MBeans are marked excluded and therefore have limited visibility in the WebLogic Scripting Tool. For example, if you use a command like this:

```
java weblogic.WLST
connect()
ls()
```

then the attributes of the `DomainMBean` will be listed, excluding Compatibility security attributes such as `FileRealmMBean`. However, if you address a Compatibility MBean directly, you can access it, as in this example:

```
java weblogic.WLST
connect()
cmo.getFileRealms()
```

The Default Security Configuration in the CompatibilityRealm

By default, the `CompatibilityRealm` is configured with a Realm Adapter Adjudication provider, a Realm Adapter Authentication provider, a WebLogic Authorization provider, a Realm Adapter Authorization provider, a WebLogic Credential Mapping provider, and a WebLogic Role Mapping provider.

- In the `CompatibilityRealm`, the Realm Adapter Authentication provider is populated with users and groups from the 6.x security realm defined in the `config.xml` file.
  - If you were using the File realm in your 6.x security configuration, you can manage the users and groups in the Realm Adapter Authentication provider following the steps in "Define Users" and "Define Groups" topics of the "Compatiblility security" section of the Administration Console online help.
  - If you are using an alternate security realm (LDAP, Windows NT, RDBMS, or custom), you must use the administration tools provided by that realm to manage users and groups.

  For information about configuring a Realm Adapter Authentication provider, see "Configuring a Realm Adapter Authentication Provider" on page 13-4

  You can use implementations of the `weblogic.security.acl.CertAuthenticator` class in Compatibility security by configuring the Identity Assertion provider in the Realm Adapter Authentication provider. For more information, see "Configuring the Identity Assertion Provider in the Realm Adapter Authentication Provider" on page 13-4.

- Access Control Lists (ACLs) in the 6.x security realm are used to populate the Realm Adapter Authorization provider.
- The Realm Adapter Adjudication provider enables the use of both ACLs and security roles and security policies in Compatibility security. The Realm Adapter Adjudication provider resolves access decision conflicts between ACLs and new security policies set through the WebLogic Administration Console.
- The WebLogic Credential Mapping provider allows the use of credential maps in Compatibility security. For more information, see *Programming WebLogic Resource Adapters*.

- You can add a Realm Adapter Auditing provider to access implementations of the `weblogic.security.audit.AuditProvider` class from the `CompatibilityRealm`. For more information, see Configure a Realm Adapter Auditing Provider in the Administration Console online help.

Configuring a Realm Adapter Authentication Provider

When using Compatibility security, a Realm Adapter Authentication provider is by default configured for the `CompatibilityRealm`. For information about using the Realm Adapter Authentication provider in the `CompatibilityRealm`, see "The Default Security Configuration in the CompatibilityRealm" on page 13-3.

The Realm Adapter Authentication provider also allows use of implementations of the `weblogic.security.acl.CertAuthenticator` class with this release of WebLogic Server. The Realm Adapter Authentication provider includes an Identity Assertion provider that asserts identity based on X.509 tokens. For information about using a CertAuthenticator with WebLogic Server, "Configuring the Identity Assertion Provider in the Realm Adapter Authentication Provider" on page 13-4.

When you add a Realm Adapter Authentication provider to a security realm with an Authentication provider already configured, WebLogic Server sets the JAAS Control Flag on the Realm Adapter Authentication provider to OPTIONAL and checks for the presence of a `fileRealm.properties` file in the domain directory. WebLogic Server will not add the Realm Adapter Authentication provider to the security realm if the `fileRealm.properties` file does not exist.

Note: The subjects produced by the Realm Adapter Authentication provider do not contain principals for the groups to which a user belongs. Use the `weblogic.security.SubjectUtils.isUserInGroup()` method to determine whether a user is in a group. When you use subjects produced by the Realm Adapter Authentication provider there is no way to iterate the complete set of groups to which a user belongs.

Configuring the Identity Assertion Provider in the Realm Adapter Authentication Provider

The Realm Adapter Authentication provider includes an Identity Assertion provider.The Identity Assertion provider provides backward compatibility for implementations of the deprecated `weblogic.security.acl.CertAuthenticator` class. The identity assertion is performed on

X.509 tokens. By default, the Identity Assertion provider is not enabled in the Realm Adapter Authentication provider.

For information about how to enable the Identity Assertion provider, see Enable the Identity Assertion provider in the Administration Console online help.

Configuring a Realm Adapter Auditing Provider

The Realm Adapter Auditing provider allows you to use implementations of the `weblogic.security.audit.AuditProvider` interface when using Compatibility security. In order for the Realm Adapter Auditing provider to work properly, the implementation of the `AuditProvider` interface must have been defined. You can define the `AuditProvider` class using the Administration Console, in the Audit Provider Class field on the Domain: Compatibility Security: General page.

For information, see Configure a Realm Adapter Auditing provider in the Administration Console online help.

Protecting User Accounts in Compatibility Security

Password guessing is a common type of security attack. In this type of attack, a hacker attempts to log in to a computer using various combinations of usernames and passwords. Weblogic Server provides a set of lockout configuration options to protect user accounts from this kind of attack. By default, these options are set for maximum protection. As a system administrator, you have the option of turning off all the options, increasing the number of login attempts before a user account is locked, increasing the time period in which invalid login attempts are made before locking the user account, and changing the amount of time a user account is locked. Remember that changing the configuration options lessens security and leaves user accounts vulnerable to security attacks.

There are two sets of configuration options available to protect user accounts, one set at the domain and one set at the security realm. You may notice that if you set one set of configuration options (for example, the options for the security realm) and exceed any of the values, the user account is not locked. This happens because the user account lockout options at the domain override the user account options at the security realm. To avoid this situation, disable the user account lockout options at the security realm.

Warning: If you disable the user lockout configuration option at the security realm, you must set the user lockout configuration options on the domain otherwise the user accounts will not be protected.

For information, see Protect user accounts and Unlock user accounts in the Administration Console online help.

Accessing 6.x Security from Compatibility Security

Using Compatibility security assumes that you have an existing `config.xml` file with a security realm that defines users and groups and ACLs that protect the resources in your WebLogic Server domain. 6.x security management tasks such as configuring a security realm or defining ACLs should not be required and therefore those management tasks are not described in this section. However, if you corrupt an existing 6.x security realm and have no choice but to restore it, the following 6.x security management tasks are described in the Compatibility Security topic of the online help for the WebLogic Administration Console:

- Configure LDAP V1 security realms
- Configure LDAP V2 security realms
- Configure RDBMS security realms
- Configure Windows NT security realms
- Configure wlauth for UNIX security realms
- Configure UNIX security realms
- Configure Custom security realms
- Configure Caching realms
- Configure the File realm
- Define ACLs
- Define groups
- Delete groups
- Define users
- Delete users
- Change user passwords
- Change the system password
- Disable the Guest user

Warning: Compatibility security provides backward compatibility only and should not be considered a long-term security solution.

Security Configuration MBeans

This appendix describes MBeans used in configuring the WebLogic Security Framework. Each MBean attribute is marked either dynamic, meaning that the attribute value can be changed without requiring a server restart, or non-dynamic, meaning that if you change the attribute value, you need to restart the server for the change to take effect. Note also that if an edit is made to a non-dynamic attribute, no edits to dynamic attributes will take effect until after restart. This is to assure that a batch of updates having a combination of dynamic and non-dynamic attribute edits will not be partially activated.

Any security MBeans not listed are completely non-dynamic (create or destroy MBean, change any attribute).

For more information about WebLogic Security MBeans, see:

- Managing Security Realms with JMX in *Developing Custom Management Utilities with JMX*
- Security MBeans in the *WebLogic Server MBean Reference*

SSLMBean

Creating or destroying this bean is dynamic.

Dynamic attributes:

Enabled, TwoWaySSLEnabled, ClientCertificateEnforced, ListenPort

Ciphersuites, ExportKeyLifespan, SSLRejectionLoggingEnabled, LoginTimeoutMillis

ServerCertificateChainFileName, ServerKeyFileName, ServerCertificateFileName, TrustedCAFileName

ServerPrivateKeyAlias, ServerPrivateKeyPassPhrase

IdentityAndTrustLocations

InboundCertificateValidation, OutboundCertificateValidation

All other attributes are non-dynamic.

ServerMBean

Creating or destroying this bean is dynamic.

Dynamic attributes:

KeyStores

CustomIdentityKeyStoreFileName, CustomIdentityKeyStoreType, CustomIdentityKeyStorePassPhrase

CustomTrustKeyStoreFileName, CustomTrustKeyStoreType, CustomTrustKeyStorePassPhrase

JavaStandardTrustKeyStorePassPhrase

All other attributes are non-dynamic.

EmbeddedLDAPMBean

Dynamic attributes:

Credential

All other attributes are non-dynamic

SecurityMBean

Dynamic attributes:

ConnectionFilterRules

ConnectionLoggerEnabled

All other attributes are non-dynamic

SecurityConfigurationMBean

Dynamic attributes:

Credential

ConnectionFilterRules, ConnectionLoggerEnabled, CompatibilityConnectionFiltersEnabled

NodeManagerUsername, NodeManagerPassword

All other attributes are non-dynamic.

RealmMBean

Creating or destroying this MBean is non-dynamic.

Dynamic attributes:

DeployRoleIgnored, DeployPolicyIgnored, DeployCredentialMappingIgnored

FullyDelegateAuthorization

ValidateDDSecurityData, SecurityDDModel

CombinedRoleMappingEnabled

All other attributes are non-dynamic

WindowsNTAuthenticatorMBean

Creating or destroying this MBean is non-dynamic.

Dynamic attributes:

BadDomainControllerRetryInterval

MapUPNNames, LogonType

MapNTDomainName

All other attributes are non-dynamic.

CustomDBMSAuthenticatorMBean

Creating or destroying this MBean is non-dynamic. The ControlFlag and read-only provider attributes (such as ProviderClassName and Description) are non-dynamic. All other attributes are dynamic.

ReadonlySQLAuthenticatorMBean

Creating or destroying this MBean is non-dynamic.

The ControlFlag and read-only provider attributes (such as ProviderClassName and Description) are non-dynamic. All other attributes are dynamic.

SQLAuthenticatorMBean

Creating or destroying this MBean is non-dynamic.

The ControlFlag and read-only provider attributes (such as ProviderClassName and Description) are non-dynamic. All other attributes are dynamic.

DefaultAuditorMBean

Creating or destroying this MBean is non-dynamic.

Dynamic attributes:

Severity

All other attributes are non-dynamic

Compatibility Security MBeans

All MBeans used for Compatibility security are completely non-dynamic (create or destroy MBean, change any attribute). These MBeans include:

RealmMBean

FileRealmMBean

BasicRealmMBean

CachingRealmMBean

PasswordPolicyMBean

CustomRealmMBean

LDAPRealmMBean

NTRealmMBean

RDBMSRealmMBean

UnixRealmMBean

UserLockoutManagerMBean

This MBean is completely non-dynamic (create or destroy MBean, change any attribute).

Other Security Provider MBeans

All other security MBeans are completely non-dynamic (create or destroy MBean, change any attribute).

Index