Eclipse GlassFish Security Guide, Release 7

Eclipse GlassFish

Security Guide

Release 7

Contributed 2018 - 2024

This book provides instructions for configuring and administering Eclipse GlassFish security.

Eclipse GlassFish Security Guide, Release 7

Copyright © 2013, 2019 Oracle and/or its affiliates. All rights reserved.

This program and the accompanying materials are made available under the terms of the Eclipse Public License v. 2.0, which is available at http://www.eclipse.org/legal/epl-2.0.

SPDX-License-Identifier: EPL-2.0

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

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

Preface

This documentation is part of the Java Enterprise Edition contribution to the Eclipse Foundation and is not intended for use in relation to Java Enterprise Edition or Orace GlassFish. The documentation is in the process of being revised to reflect the new Jakarta EE branding. Additional changes will be made as requirements and procedures evolve for Jakarta EE. Where applicable, references to Jakarta EE or Java Enterprise Edition should be considered references to Jakarta EE.

Please see the Title page for additional license information.

The Eclipse GlassFish Security Guide provides instructions for configuring and administering Eclipse GlassFish security.

This preface contains information about and conventions for the entire Eclipse GlassFish (Eclipse GlassFish) documentation set.

Eclipse GlassFish 7 is developed through the GlassFish project open-source community at https://github.com/eclipse-ee4j/glassfish. The GlassFish project provides a structured process for developing the Eclipse GlassFish platform that makes the new features of the Jakarta EE platform available faster, while maintaining the most important feature of Jakarta EE: compatibility. It enables Java developers to access the Eclipse GlassFish source code and to contribute to the development of the Eclipse GlassFish.

The following topics are addressed here:

- Eclipse GlassFish Documentation Set
- Related Documentation
- Typographic Conventions
- Symbol Conventions
- Default Paths and File Names

Eclipse GlassFish Documentation Set

The Eclipse GlassFish documentation set describes deployment planning and system installation. For an introduction to Eclipse GlassFish, refer to the books in the order in which they are listed in the following table.

Book Title	Description
Release Notes	Provides late-breaking information about the software and the documentation and includes a comprehensive, table-based summary of the supported hardware, operating system, Java Development Kit (JDK), and database drivers.
Quick Start Guide	Explains how to get started with the Eclipse GlassFish product.

Book Title	Description
Installation Guide	Explains how to install the software and its components.
Upgrade Guide	Explains how to upgrade to the latest version of Eclipse GlassFish. This guide also describes differences between adjacent product releases and configuration options that can result in incompatibility with the product specifications.
Deployment Planning Guide	Explains how to build a production deployment of Eclipse GlassFish that meets the requirements of your system and enterprise.
Administration Guide	Explains how to configure, monitor, and manage Eclipse GlassFish subsystems and components from the command line by using the `asadmin`(1M) utility. Instructions for performing these tasks from the Administration Console are provided in the Administration Console online help.
Security Guide	Provides instructions for configuring and administering Eclipse GlassFish security.
Application Deployment Guide	Explains how to assemble and deploy applications to the Eclipse GlassFish and provides information about deployment descriptors.
Application Development Guide	Explains how to create and implement Java Platform, Enterprise Edition (Jakarta EE platform) applications that are intended to run on the Eclipse GlassFish. These applications follow the open Java standards model for Jakarta EE components and application programmer interfaces (APIs). This guide provides information about developer tools, security, and debugging.
Add-On Component Development Guide	Explains how to use published interfaces of Eclipse GlassFish to develop add-on components for Eclipse GlassFish. This document explains how to perform only those tasks that ensure that the add-on component is suitable for Eclipse GlassFish.
Embedded Server Guide	Explains how to run applications in embedded Eclipse GlassFish and to develop applications in which Eclipse GlassFish is embedded.
High Availability Administration Guide	Explains how to configure Eclipse GlassFish to provide higher availability and scalability through failover and load balancing.
Performance Tuning Guide	Explains how to optimize the performance of Eclipse GlassFish.
Troubleshooting Guide	Describes common problems that you might encounter when using Eclipse GlassFish and explains how to solve them.
Error Message Reference	Describes error messages that you might encounter when using Eclipse GlassFish.
Reference Manual	Provides reference information in man page format for Eclipse GlassFish administration commands, utility commands, and related concepts.
Message Queue Release Notes	Describes new features, compatibility issues, and existing bugs for Open Message Queue.

Book Title	Description
Message Queue Technical Overview	Provides an introduction to the technology, concepts, architecture, capabilities, and features of the Message Queue messaging service.
Message Queue Administration Guide	Explains how to set up and manage a Message Queue messaging system.
Message Queue Developer's Guide for JMX Clients	Describes the application programming interface in Message Queue for programmatically configuring and monitoring Message Queue resources in conformance with the Java Management Extensions (JMX).
Message Queue Developer's Guide for Java Clients	Provides information about concepts and procedures for developing Java messaging applications (Java clients) that work with Eclipse GlassFish.
Message Queue Developer's Guide for C Clients	Provides programming and reference information for developers working with Message Queue who want to use the C language binding to the Message Queue messaging service to send, receive, and process Message Queue messages.

Related Documentation

The following tutorials explain how to develop Jakarta EE applications:

- Your First Cup: An Introduction to the Jakarta EE Platform. For beginning Jakarta EE programmers, this short tutorial explains the entire process for developing a simple enterprise application. The sample application is a web application that consists of a component that is based on the Enterprise JavaBeans specification, a JAX-RS web service, and a JavaServer Faces component for the web front end.
- The Jakarta EE Tutorial. This comprehensive tutorial explains how to use Jakarta EE platform technologies and APIs to develop Jakarta EE applications.

Javadoc tool reference documentation for packages that are provided with Eclipse GlassFish is available as follows.

- The Jakarta EE specifications and API specification is located at https://jakarta.ee/specifications/.
- The API specification for Eclipse GlassFish 7, including Jakarta EE platform packages and nonplatform packages that are specific to the Eclipse GlassFish product, is located at https://glassfish.org/docs/.

For information about creating enterprise applications in the NetBeans Integrated Development Environment (IDE), see the NetBeans Documentation, Training & Support page.

For information about the Derby database for use with the Eclipse GlassFish, see the Derby page.

The Jakarta EE Samples project is a collection of sample applications that demonstrate a broad range of Jakarta EE technologies. The Jakarta EE Samples are bundled with the Jakarta EE Software Development Kit (SDK) and are also available from the repository (`https://github.com/eclipse-ee4j/glassfish-samples`).

Typographic Conventions

The following table describes the typographic changes that are used in this book.

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name% su`
		
		`Password:`

AaBbCc123	A placeholder to be replaced with a real name or value	The command to remove a file is `rm` filename.
AaBbCc123	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the User's Guide. A cache is a copy that is stored locally. Do not save the file.

Symbol Conventions

The following table explains symbols that might be used in this book.

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.
`-`	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
`+`	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
`>`	Indicates menu item selection in a graphical user interface.	File > New > Templates	From the File menu, choose New. From the New submenu, choose Templates.

Default Paths and File Names

The following table describes the default paths and file names that are used in this book.

Placeholder	Description	Default Value
as-install	Represents the base installation directory for Eclipse GlassFish. In configuration files, as-install is represented as follows: `${com.sun.aas.installRoot}`	• Installations on the Oracle Solaris operating system, Linux operating system, and Mac OS operating system: user's-home-directory`/glassfish7/glassfish` • Installations on the Windows operating system: SystemDrive`:\glassfish7\glassfish`
as-install-parent	Represents the parent of the base installation directory for Eclipse GlassFish.	• Installations on the Oracle Solaris operating system, Linux operating system, and Mac operating system: user's-home-directory`/glassfish7` • Installations on the Windows operating system: SystemDrive`:\glassfish7`
domain-root-dir	Represents the directory in which a domain is created by default.	as-install`/domains/`
domain-dir	Represents the directory in which a domain's configuration is stored. In configuration files, domain-dir is represented as follows: `${com.sun.aas.instanceRoot}`	domain-root-dir/domain-name
instance-dir	Represents the directory for a server instance.	domain-dir/instance-name

1 Administering System Security

This chapter describes general information about administering system security.

The following topics are addressed here:

- About System Security in Eclipse GlassFish
- Administering Passwords
- Administering Audit Modules
- Administering JSSE Certificates
- Administering JACC Providers

Instructions for accomplishing many of these tasks by using the Administration Console are contained in the Administration Console online help.

Information on application security is contained in "Securing Applications" in Eclipse GlassFish Application Development Guide.

About System Security in Eclipse GlassFish

Security is about protecting data, that is, how to prevent unauthorized access or damage to data that is in storage or in transit. The Eclipse GlassFish is built on the Java security model, which uses a sandbox where applications can run safely, without potential risk to systems or users. System security affects all the applications in the Eclipse GlassFish environment.

The Jakarta EE Security API specification defines portable, plug-in interfaces for authentication and identity stores, and a new injectable-type SecurityContext interface that provides an access point for programmatic security. You can use the built-in implementations of the plug-in SPIs, or write custom implementations.

System security features include the following:

- Authentication
- Authorization
- Auditing
- Firewalls
- Certificates and SSL
- Tools for Managing System Security

Authentication

Authentication is the way in which an entity (a user, an application, or a component) determines that another entity is who it claims to be. An entity uses security credentials to authenticate itself. The credentials might be a user name and password, a digital certificate, or something else. Usually, servers or applications require clients to authenticate themselves. Additionally, clients might

require servers to authenticate themselves. When authentication is bidirectional, it is called mutual authentication.

When an entity tries to access a protected resource, Eclipse GlassFish uses the authentication mechanism configured for that resource to determine whether to grant access. For example, a user can enter a user name and password in a web browser, and if the application verifies those credentials, the user is authenticated. The user is associated with this authenticated security identity for the remainder of the session.

Authentication Types

Within its deployment descriptors, an application can specify the type of authentication that it uses. The Jakarta EE Security API provides an alternative mechanism for configuring the type of authentication an application uses. See Jakarta EE Security API 1.0 specification. Eclipse GlassFish supports specifying the following types of authentication in deployment descriptors:

BASIC

Uses the standard Basic Authentication Scheme as described in RFC 2617. The communication protocol is HTTP (SSL optional). There is no encryption of user credentials unless using SSL. This type is not considered to be a secure method of user authentication unless used in conjunction with an encrypted communications channel, such as that provided by SSL.

FORM

The application provides its own custom login and error pages. The communication protocol is HTTP (SSL optional). There is no encryption of user credentials unless using SSL.

CLIENT-CERT

The server authenticates the client using a public key certificate. The communication protocol is HTTPS (HTTP over SSL).

DIGEST

The server authenticates a user based on a user name and a password. Unlike BASIC authentication, the plaintext password is never sent over the network, although a hash of the password combined with other parameters is sent. While Digest Authentication is more secure than Basic Authentication, user names and passwords are not strongly protected, and the use of SSL is still recommended.

JSR 375 Authentication Mechanisms and Identity Stores

The Jakarta EE Security API defines the `HttpAuthenticationMechanism` interface, the `IdentityStore` and `IdentityStoreHandler` interfaces.

The `HttpAuthenticationMechanism` interface defines an SPI for writing authentication mechanisms that can be provided with an application and deployed using CDI. Developers can write their own implementations of `HttpAuthenticationMechanism` to support specific authentication token types or protocols. There are also several built-in authentication mechanisms that perform BASIC, FORM, and Custom FORM authentication. The `HttpAuthenticationMechanism` interface defines three methods - `validateRequest()`, `secureResponse()`, and `cleanSubject()`. These methods align closely with the methods defined by the JASPIC `ServerAuth` interface.

The `IdentityStore` interface provides an abstraction of an identity store that holds user account information including name, password, group membership, and potentially other attributes. Implementations of the `IdentityStore` interface are used to validate caller credentials, typically username and password, and retrieve and group information. There are built-in implementations of this SPI that can validate credentials against external LDAP or Database identity stores.

`IdentityStore` is intended primarily for use by `HttpAuthenticationMechanism` implementations, but could be used by other authentication mechanisms, such as a JASPIC `ServerAuthModule`, or a container's built-in authentication mechanisms. Though `HttpAuthenticationMechanism` implementations can authenticate users in any manner they choose, the `IdentityStore` interface provides a convenient mechanism. A significant advantage of using `HttpAuthenticationMechanism` and `IdentityStore` over the declarative mechanisms defined by the Servlet specification is that it allows an application to control the identity stores that it authenticates against, in a standard, portable way. You can use the built-in implementations of the plug-in SPIs, or define custom implementations.

JSR 196 Server Authentication Modules

Eclipse GlassFish implements the Servlet Container Profile of JSR 196 Java Authentication Service Provider Interface for Containers specification.

JSR 196 defines a standard service-provider interface (SPI) for integrating authentication mechanism implementations in message processing runtimes. JSR 196 extends the concepts of the Java Authentication and Authorization Service (JAAS) to enable pluggability of message authentication modules in message processing runtimes. The standard defines profiles that establish contracts for the use of the SPI in specific contexts.

Passwords

Passwords are your first line of defense against unauthorized access to the components and data of Eclipse GlassFish. For Information about how to use passwords for Eclipse GlassFish, see Administering Passwords.

Master Password and Keystores

The master password is not tied to a user account and it is not used for authentication. Instead, Eclipse GlassFish uses the master password only to encrypt the keystore and truststore for the DAS and instances.

When you create a new Eclipse GlassFish domain, a new self-signed certificate is generated and stored in the domain keystore and truststore. The DAS needs the master password to open these stores at startup. Similarly, the associated server instances need the master password to open their copy of these stores at startup.

If you use a utility such as keytool to modify the keystore or truststore, you must provide the master password in that case as well.

The master password is a shared password and must be the same for the DAS and all instances in the domain in order to manage the instances from the DAS. However, because Eclipse GlassFish never transmits the master password over the network, it is up to you to keep the master password

in sync between the DAS and instances.

If you change the master password, you can choose to enter the master password manually when required, or save it in a file.

Understanding Master Password Synchronization

The master password is used encrypt the keystore and truststore for the DAS and instances. The DAS needs the master password to open these stores at startup. Similarly, the associated server instances need the master password to open their copy of these stores at startup.

Eclipse GlassFish keeps the keystore and truststore for the DAS and instances in sync, which guarantees that all copies of the stores are encrypted with the same master password at any given time.

However, Eclipse GlassFish does not synchronize the master password itself, and it is possible that the DAS and instances might attempt to use different master passwords.

Consider the following potential scenario:

1. You create a domain and instances, using the default master password (changeit). As a result, the DAS and instances have keystores and truststores encrypted using changeit.
2. You use the `change-master-password` subcommand on the DAS to change the master password to ichangedit. As a result, the DAS and instance keystores and truststores are encrypted using ichangedit.
3. Access to the keystore and truststore from an instance now requires the master password ichangedit. You are responsible for changing the master password as needed.

If you do not use a master password file, you assume the responsibility for using the `change-master-password` subcommand on the DAS and instances to keep the master passwords in sync. Be aware that not using a master password file has additional considerations for the start-instance and start-cluster subcommands, as described in Additional Considerations for the `start-instance` and `start-cluster` Subcommands.

If you do use a master password file, you assume the responsibility for using the `change-master-password` subcommand on the DAS and instances to keep the master password file in sync.

Using the Default Master Password

Eclipse GlassFish uses the known phrase "changeit" as the default master password. This master password is not stored in a file. The default password is a convenience feature and provides no additional security because it is assumed to be widely known.

All Eclipse GlassFish subcommands work as expected with the default master password and there are no synchronization issues.

Saving the Master Password to a File

The `change-master-password --savemasterpassword` option indicates whether the master password should be written to the file system in the `master-password` file for the DAS or a node. The default is

false.

For a domain, the master password is kept in domain-dir/`master-password`.

For a node, the master-password file is kept in `nodes/`node-name/`agent/master-password`. You can set a master password at the node level and all instances created on that node will use that master-password file. To do this, use the `--nodedir` option and provide a node name.

You might want to save the master password to the file so that the `start-domain` subcommand can start the server without having to prompt the user. There are additional considerations for using a master password with the `start-instance` and `start-cluster` subcommands, as described in Additional Considerations for the `start-instance` and `start-cluster` Subcommands.

The master-password file is encoded, not encrypted. You must use filesystem permissions to protect the file.

Using the Master Password When Creating a Domain

The `create-domain --usemasterpassword` option specifies whether the keystore is encrypted with a master password that is built into the system, or by a user-defined master password.

- If false (default), the keystore is encrypted with a well-known password (changeit) that is built into Eclipse GlassFish.
- If true, the subcommand obtains the master password from the `AS_ADMIN_MASTERPASSWORD` entry in the password file you specified in the `--passwordfile` option of the `asadmin` utility. Or, if none is defined, `--usemasterpassword` prompts the user for the master password.

Administration Password

An administration password, also known as the admin password, is used to invoke the Administration Console and the `asadmin` utility. As with the default admin username, the default admin password is usually set during installation but it can be changed. For instructions, see To Change an Administration Password.

Encoded Passwords

Files that contain encoded passwords need to be protected using file system permissions. These files include the following:

- domain-dir/`master-password`

  This file contains the encoded master password and should be protected with file system permissions 600.

- Any password file created to pass as an argument by using the `--passwordfile` argument to the `asadmin` utility should be protected with file system permissions. Additionally, any password file being used for a transient purpose, such as setting up SSH among nodes, should be deleted after it has served its purpose.

For instructions, see To Set a Password From a File.

Web Browsers and Password Storage

Most web browsers can save login credentials entered through HTML forms. This function can be configured by the user and also by applications that employ user credentials. If the function is enabled, then credentials entered by the user are stored on their local computer and retrieved by the browser on future visits to the same application. This function is convenient for users, but can also be a security risk. The stored credentials can be captured by an attacker who gains access to the computer, either locally or through some remote compromise. Further, methods have existed whereby a malicious web site can retrieve the stored credentials for other applications, by exploiting browser vulnerabilities or through application-level cross-domain attacks.

To prevent your web browser from saving login credentials for the Eclipse GlassFish Administration Console, choose "No" or "Never for this page" when prompted by the browser during login.

Password Aliases

To avoid storing passwords in the domain configuration file in clear text, you can create an alias for a password. This process is also known as encrypting a password. For more information, see Administering Password Aliases.

Single Sign-on

With single sign-on, a user who logs in to one application becomes implicitly logged in to other applications that require the same authentication information. Single sign-on is based on groups. Single sign-on applies to web applications configured for the same realm and virtual server. The realm is defined by the `realm-name` element in the `web.xml` file.

On Eclipse GlassFish, single sign-on behavior can be inherited from the HTTP Service, enabled, or disabled. By default, it is inherited from the HTTP Service. If enabled, single sign-on is enabled for web applications on this virtual server that are configured for the same realm. If disabled, single sign-on is disabled for this virtual server, and users must authenticate separately to every application on the virtual server.

Authorization

Authorization, also known as access control, is the means by which users are granted permission to access data or perform operations. After a user is authenticated, the user's level of authorization determines what operations the owner can perform. A user's authorization is based on the user's role.

Roles

A role defines which applications and what parts of each application users can access and what those users or groups can do with the applications. For example, in a personnel application, all employees might be able to see phone numbers and email addresses, but only managers have access to salary information. This application would define at least two roles: `employee` and `manager`. Only users in the `manager` role are allowed to view salary information.

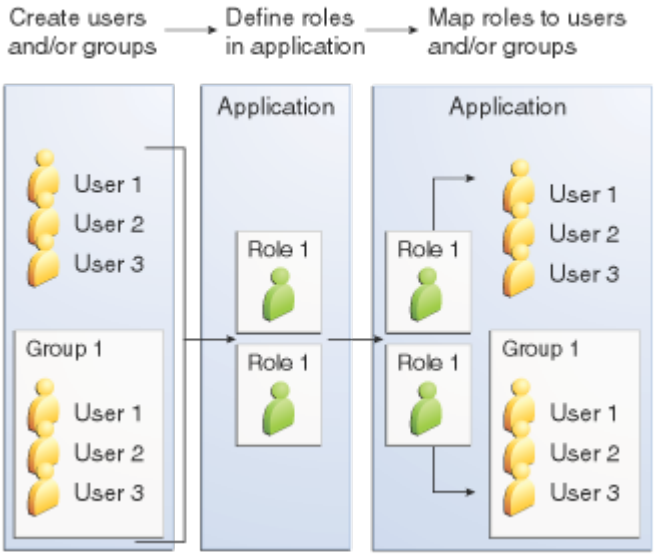
A role is different from a group in that a role defines a function in an application, while a group is a

set of users who are related in some way. For example, the personnel application specify groups such as `full-time`, `part-time`, and `on-leave`. Users in these groups are all employees (the `employee` role). In addition, each user has its own designation that defines an additional level of employment.

Roles are defined in the deployment descriptor for the application. The application developer or deployer maps roles to one or more groups in the deployment descriptor for each application. When the application is being packaged and deployed, the application specifies mappings between users, groups, and roles, as illustrated in Figure 1-1.

By default, group principal names are mapped to roles of the same name. Therefore, the Default Principal To Role Mapping setting is enabled by default on the Security page of the Eclipse GlassFish Administration Console. With this setting enabled, if the group name defined on Eclipse GlassFish matches the role name defined in the application, there is no need to use the runtime deployment descriptor to provide a mapping. The application server will implicitly make this mapping, as long as the names of the groups and roles match.

Figure 1-1 Role Mapping



Java Authorization Contract for Containers

Java Authorization Contract for Containers (JACC) is the part of the Jakarta EE specification that defines an interface for pluggable authorization providers. This enables you to set up third-party plug-in modules to perform authorization. By default, the Eclipse GlassFish provides a simple, file-based authorization engine that complies with the JACC specification.

This release includes Administration Console support and CLI subcommands to create (`create-jacc-provider`), delete (`delete-jacc-provider`), and list (`list-jacc-providers`) JACC providers. Administering JACC Providers for additional information.

You can also specify additional third-party JACC providers.

Working With the server.policy Policy File

Each Eclipse GlassFish domain has its own global Java SE policy file, located in domain-dir/`config`. The file is named `server.policy`.

This section covers the following topics:

- Contents of server.policy
- Changing the Default Permissions

Contents of server.policy

A sample server.policy file is as follows. Comments in the file describe why various permissions are granted. These permissions are described in more detail in the next section.

This server.policy file is presented for example purposes only and is subject to change.

```
// classes in lib get all permissions by default
grant codeBase "file:${com.sun.aas.installRoot}/lib/-" {
    permission java.security.AllPermission;
};

// Core server classes get all permissions by default
grant codeBase "file:${com.sun.aas.installRoot}/modules/-" {
    permission java.security.AllPermission;
};

// Felix classes get all permissions by default
grant codeBase "file:${com.sun.aas.installRoot}/osgi/felix/bin/-" {
    permission java.security.AllPermission;
};

// iMQ classes get all permissions by default
grant codeBase "file:${com.sun.aas.imqLib}/-" {
    permission java.security.AllPermission;
};

// Derby driver classes get all permissions by default
grant codeBase "file:${com.sun.aas.derbyRoot}/lib/-" {
    permission java.security.AllPermission;
};


// permission for JDK's tools.jar to enable webservice annotation processing
// at runtime by wsgen tool:
//       permission java.lang.RuntimePermission "createClassLoader";
//
```

```
// permission for JDK's tools.jar to sign JARs at runtime for
// Java Web Start support:
//       permissions java.security.AllPermission;
// on the advice of the JDK tools folks.  Should be refined later.
grant codeBase "file:${com.sun.aas.javaRoot}/lib/tools.jar" {
    permission java.security.AllPermission;
};

//Loading MBeans from anywhere, to take care of side effects of 6235678.
grant {
    permission javax.management.MBeanTrustPermission "register" ;
};
//Loading MBeans from anywhere, to take care of side effects of 6235678.


// Basic set of required permissions granted to all remaining code
// The permission FilePermission "<<ALL FILES>>", "read,write"
// allows all applications to read and write any file in the filesystem.
// It should be changed based on real deployment needs. If you know your
// applications just need to read/write a few directories consider removing
// this permission and adding grants indicating those specific directories.
// against the codebase of your application(s).
grant {
    //Workaround for bugs #6484935, 6513799
    permission java.lang.RuntimePermission "getProtectionDomain";
    permission com.sun.corba.ee.impl.presentation.rmi.DynamicAccessPermission
"access";
    permission java.util.PropertyPermission "*", "read,write";

    permission java.lang.RuntimePermission  "loadLibrary.*";
    permission java.lang.RuntimePermission  "queuePrintJob";
    permission java.net.SocketPermission    "*", "connect";
    permission java.io.FilePermission       "<<ALL FILES>>", "read,write";

        // work-around for pointbase bug 4864405
        permission java.io.FilePermission
          "${com.sun.aas.instanceRoot}${/}lib${/}databases${/}-",
           "delete";
        permission java.io.FilePermission "${java.io.tmpdir}${/}-", "delete";

    permission java.util.PropertyPermission "*", "read";

    permission java.lang.RuntimePermission    "modifyThreadGroup";
    permission java.lang.RuntimePermission    "getClassLoader";
    permission java.lang.RuntimePermission    "setContextClassLoader";
        permission javax.management.MBeanPermission
               "[com.sun.messaging.jms.*:*]", "*";
};


// Following grant block is only required by Connectors. If Connectors
```

```
// are not in use the recommendation is to remove this grant.
grant {
        permission javax.security.auth.PrivateCredentialPermission
               "javax.resource.spi.security.PasswordCredential * \"*\"","read";
};

// Following grant block is only required for Reflection. If Reflection
// is not in use the recommendation is to remove this section.
grant {
    permission java.lang.RuntimePermission "accessDeclaredMembers";
};

// Permissions to invoke CORBA objects in server
grant {
    permission com.sun.enterprise.security.CORBAObjectPermission "*", "*";
};
```

Changing the Default Permissions

The Eclipse GlassFish internal server code is granted all permissions. These grants are covered by the `AllPermission` grant blocks to various parts of the server infrastructure code. Do not modify these entries.

Application permissions are granted in the default grant block. These permissions apply to all code not part of the internal server code listed previously.

The last section, beginning with the comment "Basic set of required permissions..." provides the basic set of permissions granted to all remaining code.

Depending on your Eclipse GlassFish implementation, deleting or modifying these permissions might be appropriate.

Specifically, the following permission allows all applications to read and write all properties and read and write all files on the filesystem.

```
permission java.util.PropertyPermission "*", "read,write";
permission java.io.FilePermission       "<<ALL FILES\>>", "read,write";
```

While this grant provides optimum flexibility, it is inherently unsecure. For enhanced security, change this permission based on your real deployment needs.

For example, consider removing this permission and assign default read and write permissions only to the application's install directory (context-root). (This example uses `com.sun.aas.instanceRoot`, which specifies the top level directory for a server instance.)

```
grant codeBase "file:${com.sun.aas.instanceRoot}/applications/MyApp/-"
{
permission java.io.FilePermission "file:${com.sun.aas.instanceRoot}
/applications/MyApp/-", "read,write";
```

```
}
```

For any application that needs to read and write additional directories, you would then have to explicitly allow such permissions by adding specific grants. In general, you should add extra permissions only to the applications or modules that require them, not to all applications deployed to a domain.

Additional permissions (see the embedded comments in `server.policy`) are granted specifically for using connectors and reflection. If connectors or reflection are not used in a particular domain, you should remove these permissions, because they are otherwise unnecessary.

Auditing

Auditing is the means used to capture security-related events for the purpose of evaluating the effectiveness of security measures. Eclipse GlassFish uses audit modules to capture audit trails of all authentication and authorization decisions. Eclipse GlassFish provides a default audit module, as well as the ability to plug in custom audit modules. The scope of the audit module is the entire server, which means that all the applications on the server will use the same audit module.

For administration instructions, see Administering Audit Modules.

Firewalls

A firewall controls the flow of data between two or more networks, and manages the links between the networks. A firewall can consist of both hardware and software elements. The following guidelines pertain primarily to Eclipse GlassFish:

- In general, firewalls should be configured so that clients can access the necessary TCP/IP ports.

  For example, if the HTTP listener is operating on port 8080, configure the firewall to allow HTTP requests on port 8080 only. Likewise, if HTTPS requests are set up for port 8081, you must configure the firewalls to allow HTTPS requests on port 8081.

- If direct Remote Method Invocations over Internet Inter-ORB Protocol (RMI-IIOP) access from the Internet to EJB modules is required, open the RMI-IIOP listener port as well.

  Opening the RMI-IIOP listener port is strongly discouraged because it creates security risks.

- In double firewall architecture, you must configure the outer firewall to allow for HTTP and HTTPS transactions. You must configure the inner firewall to allow the HTTP server plug-in to communicate with Eclipse GlassFish behind the firewall.

Certificates and SSL

The following topics are addressed here:

- Certificates
- Certificate Chains

- Certificate Files
- Secure Sockets Layer
- Custom Authentication of Client Certificate in SSL Mutual Authentication

For administration instructions, see Administering JSSE Certificates.

Certificates

Certificates, also called digital certificates, are electronic files that uniquely identify people and resources on the Internet. Certificates also enable secure, confidential communication between two entities. There are different kinds of certificates:

- Personal certificates are used by individuals.
- Server certificates are used to establish secure sessions between the server and clients through secure sockets layer (SSL) technology.

Certificates are based on public key cryptography, which uses pairs of digital keys (very long numbers) to encrypt, or encode, information so the information can be read only by its intended recipient. The recipient then decrypts (decodes) the information to read it. A key pair contains a public key and a private key. The owner distributes the public key and makes it available to anyone. But the owner never distributes the private key, which is always kept secret. Because the keys are mathematically related, data encrypted with one key can only be decrypted with the other key in the pair.

Certificates are issued by a trusted third party called a Certification Authority (CA). The CA is analogous to a passport office: it validates the certificate holder's identity and signs the certificate so that it cannot be forged or tampered with. After a CA has signed a certificate, the holder can present it as proof of identity and to establish encrypted, confidential communications. Most importantly, a certificate binds the owner's public key to the owner's identity.

In addition to the public key, a certificate typically includes information such as the following:

- The name of the holder and other identification, such as the URL of the web server using the certificate, or an individual's email address
- The name of the CA that issued the certificate
- An expiration date

Certificates are governed by the technical specifications of the X.509 format. To verify the identity of a user in the `certificate` realm, the authentication service verifies an X.509 certificate, using the common name field of the X.509 certificate as the principal name.

Certificate Chains

A certificate chain is a series of certificates issued by successive CA certificates, eventually ending in a root CA certificate.

Web browsers are preconfigured with a set of root CA certificates that the browser automatically trusts. Any certificates from elsewhere must come with a certificate chain to verify their validity.

When a certificate is first generated, it is a self-signed certificate. A self-signed certificate is one for which the issuer (signer) is the same as the subject (the entity whose public key is being authenticated by the certificate). When the owner sends a certificate signing request (CSR) to a CA, then imports the response, the self-signed certificate is replaced by a chain of certificates. At the bottom of the chain is the certificate (reply) issued by the CA authenticating the subject's public key. The next certificate in the chain is one that authenticates the CA's public key. Usually, this is a self-signed certificate (that is, a certificate from the CA authenticating its own public key) and the last certificate in the chain.

In other cases, the CA can return a chain of certificates. In this situation, the bottom certificate in the chain is the same (a certificate signed by the CA, authenticating the public key of the key entry), but the second certificate in the chain is a certificate signed by a different CA, authenticating the public key of the CA to which you sent the CSR. Then, the next certificate in the chain is a certificate authenticating the second CA's key, and so on, until a self-signed root certificate is reached. Each certificate in the chain (after the first) thus authenticates the public key of the signer of the previous certificate in the chain.

Certificate Files

During Eclipse GlassFish installation, a certificate is generated in Java Secure Socket Extension (JSSE) format suitable for internal testing. (The certificate is self-signed.) By default, Eclipse GlassFish stores its certificate information in certificate databases in the domain-dir`/config` directory:

Keystore file

The `keystore.jks` file contains Eclipse GlassFish certificate, including its private key. The keystore file is protected with a password.
Each keystore entry has a unique alias. After installation, the Eclipse GlassFish keystore has a single entry with an alias of `s1as`.

Truststore file

The `cacerts.jks` file contains the Eclipse GlassFish trusted certificates, including public keys for other entities. For a trusted certificate, the server has confirmed that the public key in the certificate belongs to the certificate's owner. Trusted certificates generally include those of CAs.

By default, Eclipse GlassFish is configured with a keystore and truststore that will work with the example applications and for development purposes.

Secure Sockets Layer

Secure Sockets Layer (SSL) is the most popular standard for securing Internet communications and transactions. Secure web applications use HTTPS (HTTP over SSL). The HTTPS protocol uses certificates to ensure confidential and secure communications between server and clients. In an SSL connection, both the client and the server encrypt data before sending it. Data is decrypted upon receipt.

When a Web browser (client) wants to connect to a secure site, an SSL handshake happens, like this:

1. The browser sends a message over the network requesting a secure session (typically, by requesting a URL that begins with `https` instead of `http`).
2. The server responds by sending its certificate (including its public key).
3. The browser verifies that the server's certificate is valid and is signed by a CA whose certificate is in the browser's database (and who is trusted). It also verifies that the CA certificate has not expired.
4. If the certificate is valid, the browser generates a one time, unique session key and encrypts it with the server's public key. The browser then sends the encrypted session key to the server so that they both have a copy.
5. The server decrypts the message using its private key and recovers the session key.

After the handshake, the client has verified the identity of the Web site, and only the client and the Web server have a copy of the session key. From this point forward, the client and the server use the session key to encrypt all their communications with each other. Thus, their communications are ensured to be secure.

The newest version of the SSL standard is called Transport Layer Security (TLS). The Eclipse GlassFish supports the SSL 3.0 and the TLS 1.0 encryption protocols.

To use SSL, Eclipse GlassFish must have a certificate for each external interface or IP address that accepts secure connections. The HTTPS service of most web servers will not run unless a certificate has been installed. For instructions on applying SSL to HTTP listeners, see " To Configure an HTTP Listener for SSL" in Eclipse GlassFish Administration Guide.

Ciphers

A cipher is a cryptographic algorithm used for encryption or decryption. SSL and TLS protocols support a variety of ciphers used to authenticate the server and client to each other, transmit certificates, and establish session keys.

Some ciphers are stronger and more secure than others. Clients and servers can support different cipher suites. During a secure connection, the client and the server agree to use the strongest cipher that they both have enabled for communication, so it is usually sufficient to enable all ciphers.

Name-based Virtual Hosts

Using name-based virtual hosts for a secure application can be problematic. This is a design limitation of the SSL protocol itself. The SSL handshake, where the client browser accepts the server certificate, must occur before the HTTP request is accessed. As a result, the request information containing the virtual host name cannot be determined prior to authentication, and it is therefore not possible to assign multiple certificates to a single IP address.

If all virtual hosts on a single IP address need to authenticate against the same certificate, the addition of multiple virtual hosts probably will not interfere with normal SSL operations on the server. Be aware, however, that most browsers will compare the server's domain name against the domain name listed in the certificate, if any (applicable primarily to official, CA-signed certificates). If the domain names do not match, these browsers display a warning. In general, only address-based virtual hosts are commonly used with SSL in a production environment.

Custom Authentication of Client Certificate in SSL Mutual Authentication

Release 7 of Eclipse GlassFish extends the Certificate realm to allow custom authentication and group assignment based on the client certificate received as part of SSL mutual (two-way) authentication.

As in previous releases, you can create only one certificate realm. However, you can now use a convenient abstract base class to configure a JAAS LoginModule for the Certificate realm. Specifically, your LoginModule can now extend `com.sun.appserv.security.AppservCertificateLoginModule`. When you do this, you need to implement only the `authenticateUser` method and call the `commitUserAuthentication` method to signify success.

This section describes the following topics:

- Understanding the `AppservCertificateLoginModule` Class
- Example AppservCertificateLoginModule Code
- Setting the JAAS Context

Understanding the `AppservCertificateLoginModule` Class

The `AppservCertificateLoginModule` class provides some convenience methods for accessing the certificates, the application name and so forth, and for adding the group principals to the subject. The convenience methods include the following:

`getAppName()`
: Returns the name of the application to be authenticated. This may be useful when a single LoginModule has to handle multiple applications that use certificates.

`getCerts()`
: Returns the certificate chain as an array of `java.security.cert.X509Certificate` certificates.

`getX500Principal()`
: Returns the Distinguished principal from the first certificate in the chain.

`getSubject()`
: Returns the subject that is being authenticated.

`commitUserAuthentication(final String[] groups)`
: This method sets the authentication status to success if the groups parameter is non-null. Note that this method is called after the authentication has succeeded. If authentication failed, do not call this method.

You do not have to extend the convenience base class, you can extend the JAAS LoginModule `javax.security.auth.spi.LoginModule` instead if you so choose.

Example AppservCertificateLoginModule Code

Example 1-1 shows a sample instance of the AppservCertificateLoginModule class.

Take note of the following points from the example:

- The `getX500Principal()` method returns the subject (subject distinguished name) value from the first certificate in the client certificate chain as an `X500Principal`.
- From that `X500Principal`, the `getName()` method then returns a string representation of the X.500 distinguished name using the format defined in RFC 2253.
- The example uses the `getAppName()` method to determine the application name. It also determines the organizational unit (`OU`) from the distinguished name.
- The example concatenates the application name with the value of `OU`, and uses it as the group name in the `commitUserAuthentication` method.

Example 1-1 Sample AppservCertificateLoginModule Code

```
public class CertificateLM extends AppservCertificateLoginModule {

    @Override
    protected void authenticateUser() throws LoginException {
        // Get the distinguished name from the X500Principal.
        String dname = getX500Principal().getName();
        StringTokenizer st = new StringTokenizer(dname, "B \t\n\r\f,");
        while (st.hasMoreTokens()) {
            String next = st.nextToken();
            // Set the appname:OU as the group.
            // At this point, one has the application name and the DN of
            // the certificate. A suitable login decision can be made here.
            if (next.startsWith("OU=")) {
            commitUserAuthentication(new String[]{getAppName() + ":" + next.substring
(3)});
                return;
            }
        }
        throw new LoginException("No OU found.");
    }
}
```

Setting the JAAS Context

After you create your LoginModule, you must plug it in to a jaas-context, which you then specify as a parameter to the certificate realm in Eclipse GlassFish.

To do this, perform the following steps:

1. Specify a new jaas-context for the Certificate realm in the file domain-dir`/config/login.conf`. For example, using the `CertificateLM` class from Example AppservCertificateLoginModule Code:

```
certRealm {
    com.sun.blogs.certificate.login.CertificateLM required;
```

```
};
```

2. Specify this jaas-context as a parameter to the `set` subcommand in the `configs.config.server-config.security-service.auth-realm.certificate.property.jaas-context=`<jaas-context-name> property. For example:

```
asadmin> set configs.config.server-config.security-service.auth-realm.certificate.property.jaas-context=certRealm

configs.config.server-config.security-service.auth-realm.certificate.property.jaas-context=certRealm

Command set executed successfully.
```

3. Optionally, get the value you just set to make sure that it is correct.

```
asadmin> get configs.config.server-config.security-service.auth-realm.certificate.property.jaas-context

configs.config.server-config.security-service.auth-realm.certificate.property.jaas-context=certRealm

Command get executed successfully.
```

Tools for Managing System Security

Eclipse GlassFish provides the following tools for managing system security:

Administration Console

The Administration Console is a browser-based utility used to configure security for the entire server. Tasks include managing certificates, users, groups, and realms, and performing other system-wide security tasks. For a general introduction to the Administration Console, see "Administration Console" in Eclipse GlassFish Administration Guide.

The `asadmin` utility

The `asadmin` command-line utility performs many of the same tasks as the Administration Console. You might be able to do some things with the `asadmin` utility that you cannot do with the Administration Console. For a general introduction to `asadmin`, see "asadmin Utility" in Eclipse GlassFish Administration Guide.

The `keytool` utility

The `keytool` Java Platform, Standard Edition (Java SE) command-line utility is used for managing digital certificates and key pairs. For more information, see Administering JSSE Certificates.

The `policytool` utility

The `policytool` Java SE graphical utility is used for managing system-wide Java security policies. As an administrator, you rarely use `policytool`.

Administering Passwords

There are multiple ways to administer passwords. You can rely on administrators to keep

passwords secret and change the passwords regularly. You can set up files for storing passwords so that `asadmin` subcommands can access these files rather than having users type the commands. You can encrypt passwords by setting up aliases so that sensitive passwords are not visible in the `domain.xml` file.

The following topics are addressed here:

- To Change the Master Password
- Additional Considerations for the `start-instance` and `start-cluster` Subcommands
- Using `start-instance` and `start-cluster` With a Password File
- To Change an Administration Password
- To Set a Password From a File
- Administering Password Aliases

To Change the Master Password

The master password gives access to the keystore used with the domain. This password is not tied to a UNIX user. You should treat this overall shared password as sensitive data. Eclipse GlassFish never uses it for authentication and never transmits it over the network.

You can choose to type the password manually when required, or to obscure the password in a password file. If there is no password file, you are prompted for the master password. If there is a password file, but you want to change access to require prompting, remove the file. The default master password is `changeit`.

When changing the master password, it has to be changed on all nodes as well as on the DAS. The master password on nodes is only stored once in the node, for all instances that are on that node.

Use the `change-master-password` subcommand in local mode to modify the master password.

If you change the master password and are not using a master password file, the `start-instance` and `start-cluster` subcommands are not able to determine the master password. In this case, you must start those instances locally by using start-local-instance.

When the master password is saved, it is saved in the `master-password` file.

Before You Begin

This subcommand will not work unless the domain is stopped.

1. Stop the domain whose password you are changing.

   See "To Stop a Domain" in Eclipse GlassFish Administration Guide.

2. Change the master password for the domain by using the `change-master-password` subcommand.

   You are prompted for the old and new passwords. All dependent items are re-encrypted.

3. Start the domain.

   See "To Start a Domain" in Eclipse GlassFish Administration Guide.

Example 1-2 Changing the Master Password

The `change-master-password` subcommand is interactive in that you are prompted for the old master password as well as the new master password. This example changes the master password for `domain44ps`:

```
asadmin> change-master-password domain44ps
```

If you have already logged into the domain using the `login` subcommand, you are prompted for the new master password:

```
Please enter the new master password>
Please enter the new master password again>
```

If you are not logged into the domain, you are prompted for both the old and the new master passwords:

```
Please enter the master password>
Please enter the new master password>
Please enter the new master password again>
```

Information similar to the following is displayed:

```
Master password changed for domain44ps
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin --help change-master-password` at the command line.

Additional Considerations for the `start-instance` and `start-cluster` Subcommands

If you change the master password for DAS, the `start-domain` and `start-local-instance` subcommands allow you to provide it during domain or instance startup in one of three ways:

- Via the master-password file
- By entering it interactively
- Via the `asadmin passwordfile`

The `start-instance` and `start-cluster` subcommands are more problematic. If you create a domain

with a master password other than the default, an associated remote instance or cluster must have access to the master password in order to start. However, for security reasons Eclipse GlassFish never transmits the master password or the master password file over the network.

Consider the following scenario:

1. Change the master password on the DAS and save it with `--savemasterpassword`.
2. Create an instance on another host using the subcommand `create-instance`. Eclipse GlassFish copies the keystore and truststore from the DAS to the instance, but it does not copy the master password file.
3. Try to start the instance using the `start-instance` subcommand. An error results.

The `start-instance` command is looking for the file master-password in the node directory on the instance machine, and it is not there by default. Therefore, the subcommand fails.

You can use the `change-master-password` subcommand to make sure the correct password is used in this password file, as described in Using `start-instance` and `start-cluster` With a Password File.

The `start-instance` and `start-cluster` subcommands do not include any other way for you to provide the password. If you change the master password and are not using a master password file, the `start-instance` and `start-cluster` subcommands are not able to determine the master password. In this case, you must start the instances locally by using `start-local-instance`.

Using `start-instance` and `start-cluster` With a Password File

Assume that you have changed the master password on the DAS and you want to make the same change for all instances.

The `start-instance` and `start-cluster` subcommands automatically use the master password file if it exists in the instance filesystem. You can use the `change-master-password` subcommand to make sure the password file exists and that the correct password is used.

1. From the DAS, create a domain and set the master password.
   ```
   asadmin> create-domain --savemasterpassword true domain-name
   ```
2. Start the domain.
   ```
   asadmin> start-domain domain-name
   ```
3. Create a node that is enabled for communication over secure shell (SSH).
   ```
   asadmin> create-node-ssh --nodehost host-name --installdir/some-dir node-name
   ```
4. Create an instance on the node.
   ```
   asadmin> create-instance --node node-name instance-name
   ```
5. Before you start the instance, on the instance machine run `change-master-password` with the `---savemasterpassword` option to create a file called `master-password` in the agents directory to access the keystores. (The `start-instance` subcommand is looking for a file called `master-password` in the agents directory to access the stores.)
   ```
   asadmin> change-master-password --savemasterpassword true --nodedir /some-dir node-name
   ```
   You are prompted to enter the current and new master password:

```
Enter the current master password>
Enter the new master password>
Enter the new master password again>
Command change-master-password executed successfully.
```

Remember that when you created the domain you specified a new master password. This master password was then used to encrypt the keystore and truststore for the DAS, and these stores were copied to the instance as a result of the `create-instance` subcommand.

Therefore, enter the master password you set when you created the domain as both the current master password and again as the new master password. You enter it as the new master password because you do not want to change the master password for the instance and make it out of sync with the DAS.

6. Run start-instance from the DAS.

   `asadmin> start-instance` instance-name

   The master password file is associated with the node and not with an instance. After the master password file exists in the node directory on the instance machine, additional instances can be created, started and stopped from the DAS.

To Change an Administration Password

Use the `change-admin-password` subcommand in remote mode to change an administration password. The default administration user is `admin`. You are prompted for the old and new admin passwords, with confirmation. The passwords are not echoed to the display.

For the zip bundle of Eclipse GlassFish 7, the default administrator login is `admin`, with no password, which means that no login is required. For Eclipse GlassFish, you are prompted to provide a password for the `admin` user when you start the domain for the first time.

If there is a single user called `admin` that does not have a password, you are not prompted for login information. Any other situation requires login.

If secure administration is enabled as described in Running Secure Admin, you cannot change an administration password to a blank value.

Encrypting the admin password is strongly encouraged.

1. Change the admin password by using the `change-admin-password` subcommand.
2. Enter the old and new admin passwords when prompted.
3. Restart Eclipse GlassFish.
   See "To Restart a Domain" in Eclipse GlassFish Administration Guide.

Example 1-3 Changing the Admin Password

This example changes the admin password for user anonymous from `adminadmin` to `newadmin`:

```
asadmin> change-admin-password --username anonymous
```

You are prompted to enter the old and the new admin passwords:

```
Enter admin password>adminadmin
Enter new admin password>newadmin
Enter new admin password again>newadmin
```

Information similar to the following is displayed:

```
Command change-admin-password executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help change-admin-password` at the command line.

To Set a Password From a File

Instead of typing the password at the command line, you can access the password for a command from a file such as `passwords.txt`. The `--passwordfile` option of the `asadmin` utility takes the name of the file that contains the passwords. The entry for a password in the file must have the `AS_ADMIN_` prefix followed by the password name in uppercase letters.

Any password file created to pass as an argument by using the `--passwordfile` argument to the `asadmin` utility should be protected with file system permissions. Additionally, any password file being used for a transient purpose, such as setting up SSH among nodes, should be deleted after it has served its purpose.

For a list of the types of passwords that can be specified, see the `asadmin(1M)` help page.

```
AS_ADMIN_MASTERPASSWORD
AS_ADMIN_USERPASSWORD
AS_ADMIN_ALIASPASSWORD
```

1. Edit the password file.
   For example, to specify the password for the domain administration server (DAS), add an entry similar to the following to the password file, where `adminadmin` is the administrator password:

   ```
   AS_ADMIN_PASSWORD=adminadmin
   ```

2. Save the password file.

You can now specify the password file in an `asadmin` subcommand. In this example, `passwords.txt` is the file that contains the password:

```
asadmin>delete-jdbc-resource --user admin --passwordfile passwords.txt
jdbc/DerbyPool
```

Troubleshooting

If `AS_ADMIN_PASSWORD` has been exported to the global environment, specifying the `--passwordfile` option will produce a warning about using the `--passwordfile` option. To prevent this warning situation from happening, unset `AS_ADMIN_PASSWORD`.

Administering Password Aliases

A password alias is used to indirectly access a password so that the password itself does not appear in cleartext in the domain's `domain.xml` configuration file.

Storing passwords in cleartext format in system configuration files is common in many open source projects. In addition to Eclipse GlassFish, Apache Tomcat, Maven, and Subversion, among others, store and pass passwords in cleartext format. However, storing and passing passwords in cleartext can be a security risk, and may violate some corporate security policies. In such cases, you can use password aliases.

The following topics are addressed here:

- To Create a Password Alias
- To List Password Aliases
- To Delete a Password Alias
- To Update a Password Alias

To Create a Password Alias

Use the `create-password-alias` subcommand in remote mode to create an alias for a password in the domain's keystore. The password corresponding to the alias name is stored in an encrypted form in the domain configuration file. The `create-password-alias` subcommand takes both a secure interactive form, in which users are prompted for all information, and a more script-friendly form, in which the password is propagated on the command line.

You can also use the `set` subcommand to remove and replace the password in the configuration file. For example:

```
asadmin set --user admin server.jms-service.jms-host.default_JMS_host.
admin-password='${ALIAS=jms-password}'
```

1. Ensure that the server is running. Remote subcommands require a running server.
2. Go to the directory where the configuration file resides.

By default, the configuration file is located in domain-dir/config.

3. Create the password alias by using the create-password-alias subcommand.
4. Type the password for the alias when prompted.
5. Add the alias to a password file.
   For example, assume the use of a password file such as passwords.txt. Assume further that you want to add an alias for the AS_ADMIN_USERPASSWORD entry that is read by the create-file-user subcommand. You would add the following line to the password file: AS_ADMIN_USERPASSWORD=${ALIAS=user-password-alias}, where user-password-alias is the new password alias.
6. To continue the example of the previous step, you would then run the create-file-user subcommand.
   You could use this method to create several users (user1, user2, and so forth), all with the same password.
   asadmin> --passwordfile``passwords.txt create-file-user user1

Example 1-4 Creating a Password Alias

This example creates the new jms-password alias for the admin user:

```
asadmin> create-password-alias --user admin jms-password
```

You are prompted to type the password for the alias:

```
Please enter the alias password>secret-password
Please enter the alias password again>secret-password
Command create-password-alias executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing asadmin help create-password-alias at the command line.

To List Password Aliases

Use the list-password-aliases subcommand in remote mode to list existing the password aliases.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List password aliases by using the list-password-aliases subcommand.

Example 1-5 Listing Password Aliases

This example lists the existing password aliases:

```
asadmin> list-password aliases
jmspassword-alias
```

```
Command list-password-aliases executed successfully
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help list-password-aliases` at the command line.

To Delete a Password Alias

Use the `delete-password-alias` subcommand in remote mode to delete an existing password alias.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List all aliases by using the `list-password-aliases` subcommand.
3. Delete a password alias by using the `list-password-aliases` subcommand.

Example 1-6 Deleting a Password Alias

This example deletes the password alias `jmspassword-alias`:

```
asadmin> delete-password-alias jmspassword-alias
Command list-password-aliases executed successfully
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-password-alias` at the command line.

To Update a Password Alias

Use the `update-password-alias` subcommand in remote mode to change the password for an existing password alias. The `update-password-alias` subcommand takes both a secure interactive form, in which the user is prompted for all information, and a more script-friendly form, in which the password is propagated on the command line.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Update an alias by using the `update-password-alias` subcommand.
3. Type the password when prompted.

Example 1-7 Updating a Password Alias

This example updates the password for the `jmspassword-alias` alias:

```
asadmin> update-password-alias jsmpassword-alias
```

You are prompted to type the new password for the alias:

```
Please enter the alias password>new-secret-password
Please enter the alias password again>new-secret-password
Command update-password-alias executed successfully
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help update-password-alias` at the command line.

Administering Audit Modules

The following topics are addressed here:

- To Create an Audit Module
- To List Audit Modules
- To Delete an Audit Module

To Create an Audit Module

Use the `create-audit-module` subcommand in remote mode to create an audit module for the add-on component that implements the audit capabilities.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create an audit module by using the `create-audit-module` subcommand.

   Information about properties for this subcommand is included in this help page.

Example 1-8 Creating an Audit Module

This example creates an audit module named `sampleAuditModule`:

```
asadmin> create-audit-module
--classname com.sun.appserv.auditmodule --property defaultuser=
admin:Password=admin sampleAuditModule
Command create-audit-module executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help create-audit-module` at the command line.

To List Audit Modules

Use the `list-audit-modules` subcommand in remote mode to list the audit modules on one of the following targets:

- Server instance, `server` (the default)

- Specified server instance
- Specified configuration

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the audit modules by using the `list-audit-modules` subcommand.

Example 1-9 Listing Audit Modules

This example lists the audit modules on `localhost`:

```
asadmin> list-audit-modules
audit-module : default
audit-module : sampleAuditModule
Command list-audit-modules executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help list-audit-modules` at the command line.

To Delete an Audit Module

Use the `delete-audit-module` subcommand in remote mode to delete an existing audit module.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the audit modules by using the `list-audit-modules` subcommand.
3. Delete an audit module by using the `delete-audit-module` subcommand.

Example 1-10 Deleting an Audit Module

This example deletes `sampleAuditModule`:

```
asadmin> delete-audit-module sampleAuditModule
Command delete-audit-module executed successfully.
```

Administering JSSE Certificates

In the developer profile, the Eclipse GlassFish 7 uses the JSSE format on the server side to manage certificates and key stores. In all profiles, the client side (appclient or stand-alone) uses the JSSE format.

The Java SE SDK ships with the `keytool` utility, which enables you to set up and work with Java Secure Socket Extension (JSSE) digital certificates. You can administer public/private key pairs and associated certificates, and cache the public keys (in the form of certificates) of their communicating peers.

The following topics are addressed here:

- To Generate a Certificate by Using `keytool`
- To Sign a Certificate by Using `keytool`
- To Delete a Certificate by Using `keytool`

To Generate a Certificate by Using `keytool`

By default, the `keytool` utility creates a keystore file in the directory where the utility is run.

Before You Begin

To run the `keytool` utility, your shell environment must be configured so that the Java SE `/bin` directory is in the path, otherwise the full path to the utility must be present on the command line.

1. Change to the directory that contains the keystore and truststore files.
   Always generate the certificate in the directory containing the keystore and truststore files. The default is domain-dir`/config`.
2. Generate the certificate in the keystore file, `keystore.jks`, using the following command format:

   ```
   keytool -genkey -alias keyAlias-keyalg RSA
    -keypass changeit
    -storepass changeit
   keystore keystore.jks
   ```

   Use any unique name as your keyAlias. If you have changed the keystore or private key password from the default (`changeit`), substitute the new password for `changeit`. The default key password alias is `s1as`.

   A prompt appears that asks for your name, organization, and other information.

3. Export the generated certificate to the `server.cer` file (or `client.cer` if you prefer), using the following command format:

   ```
   keytool -export -alias keyAlias-storepass changeit
    -file server.cer
    -keystore keystore.jks
   ```

4. If a certificate signed by a certificate authority is required, see To Sign a Certificate by Using `keytool`.
5. Create the `cacerts.jks` truststore file and add the certificate to the truststore, using the following command format:

   ```
   keytool -import -v -trustcacerts
   -alias keyAlias
    -file server.cer
   -keystore cacerts.jks
   ```

```
 -keypass changeit
```

If you have changed the keystore or private key password from the default (`changeit`), substitute the new password.

Information about the certificate is displayed and a prompt appears asking if you want to trust the certificate.

6. Type `yes`, then press Enter.

   Information similar to the following is displayed:

   ```
   Certificate was added to keystore
   [Saving cacerts.jks]
   ```

7. To apply your changes, restart Eclipse GlassFish. See "To Restart a Domain" in Eclipse GlassFish Administration Guide.

Example 1-11 Creating a Self-Signed Certificate in a JKS Keystore by Using an RSA Key Algorithm

RSA is public-key encryption technology developed by RSA Data Security, Inc.

```
keytool -genkey -noprompt -trustcacerts -keyalg RSA -alias ${cert.alias}
-dname ${dn.name} -keypass ${key.pass} -keystore ${keystore.file}
-storepass ${keystore.pass}
```

Example 1-12 Creating a Self-Signed Certificate in a JKS Keystore by Using a Default Key Algorithm

```
keytool -genkey -noprompt -trustcacerts -alias ${cert.alias} -dname
${dn.name} -keypass ${key.pass} -keystore ${keystore.file} -storepass
${keystore.pass}
```

Example 1-13 Displaying Available Certificates From a JKS Keystore

```
keytool -list -v -keystore ${keystore.file} -storepass ${keystore.pass}
```

Example 1-14 Displaying Certificate information From a JKS Keystore

```
keytool -list -v -alias ${cert.alias} -keystore ${keystore.file}
-storepass ${keystore.pass}
```

See Also

To Sign a Certificate by Using `keytool`

After creating a certificate, the owner must sign the certificate to prevent forgery. E-commerce sites, or those for which authentication of identity is important, can purchase a certificate from a well-known Certificate Authority (CA).

If authentication is not a concern, for example if private secure communications are all that is required, you can save the time and expense involved in obtaining a CA certificate by using a self-signed certificate.

1. Delete the default self-signed certificate:

   ```
   keytool -delete -alias s1as -keystore keystore.jks -storepass <store_passwd>
   ```

   where <store_passwd> is the password for the keystore. For example, "mypass". Note that `s1as` is the default alias of the Eclipse GlassFish keystore.

2. Generate a new key pair for the application server:

   ```
   keytool -genkeypair -keyalg <key_alg> -keystore keystore.jks
   -validity <val_days> -alias s1as
   ```

   where <key_alg> is the algorithm to be used for generating the key pair, for example RSA, and <val_days> is the number of days that the certificate should be considered valid. For example, 365.

   In addition to generating a key pair, the command wraps the public key into a self-signed certificate and stores the certificate and the private key in a new keystore entry identified by the alias.

   For HTTPS hostname verification, it is important to ensure that the name of the certificate (CN) matches the fully-qualified hostname of your site (fully-qualified domain name). If the names do not match, clients connecting to the server will see a security alert stating that the name of the certificate does not match the name of the site.

3. Generate a Certificate Signing Request (CSR):

   ```
   keytool -certreq -alias s1as -file <certreq_file> -keystore keystore.jks
   -storepass <store_passwd>
   ```

   where <certreq_file> is the file in which the CSR is stored (for example, `s1as.csr`) and <store_passwd> is the password for the keystore. For example, changeit.

4. Submit the CSR to a Certificate Authority such as VeriSign (at http://www.verisign.com/ssl/buy-ssl-certificates/index.html). In response, you should receive a signed server certificate. Make sure to import into your browser the CA certificate of the CA (if not already present) and any intermediate certificates indicated by the CA in the reply.

5. Store the signed server certificate from the CA, including the markers `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`, into a file such as `s1as.cert`. Download the CA certificate and any intermediate CA certificates and store them in local files.
6. Import the CA certificate (if not already present) and any intermediate CA certificates (if not already present) indicated by the CA into the truststore `cacerts.jks`:

```
keytool -import -v -trustcacerts -alias <CA-Name> -file ca.cert
 -keystore cacerts.jks -storepass <store_passwd>
```

7. Replace the original self-signed certificate with the certificate you obtained from the CA, as stored in a file such as `s1as.cert`:

```
keytool -import -v -trustcacerts -alias s1as -file s1as.cert
   -keystore keystore.jks -storepass <store_passwd>
```

   When you import the certificate using the same original alias `s1as`, keytool treats it as a command to replace the original certificate with the certificate obtained as a reply to a CSR.

   After running the command, you should see that the certificate `s1as` in the keystore is no longer the original self-signed certificate, but is now the response certificate from the CA.

   Consider the following example that compares an original `s1as` certificate with a new `s1as` certificate obtained from VeriSign:

```
Original s1as (self-signed):

Owner: CN=FQDN, OU=Sun Java System Application Server, O=Sun
Microsystems, L=Santa Clara, ST=California, C=US
Issuer: CN=KUMAR, OU=Sun Java System Application Server, O=Su
n Microsystems, L=Santa Clara, ST=California, C=US
Serial number: 472acd34
Valid from: Fri Nov 02 12:39:40 GMT+05:30 2007 until: Mon Oct
30 12:39:40 GMT+05:30 2017

New s1as (contains signed cert from CA):

Owner: CN=FQDN, OU=Terms of use at www.verisign.com/cps/test
ca (c)05, OU=Sun Java System Application Server, O=Sun Micros
ystems, L=Santa Clara, ST=California, C=US
Issuer: CN=VeriSign Trial Secure Server Test CA, OU=Terms of
use at https://www.verisign.com/cps/testca (c)05, OU="For Test
Purposes Only. No assurances.", O="VeriSign, Inc.", C=US
Serial number: 1375de18b223508c2cb0123059d5c440
Valid from: Sun Nov 11 05:30:00 GMT+05:30 2007 until: Mon Nov
26 05:29:59 GMT+05:30 2007
```

8. To apply your changes, restart Eclipse GlassFish.

See "To Restart a Domain" in Eclipse GlassFish Administration Guide.

Example 1-15 Importing an RFC/Text-Formatted Certificate Into a JKS Keystore

Certificates are often stored using the printable encoding format defined by the Internet Request for Comments (RFC) 1421 standard instead of their binary encoding. This certificate format, also known as Base 64 encoding, facilitates exporting certificates to other applications by email or through some other mechanism.

```
keytool -import -noprompt -trustcacerts -alias ${cert.alias} -file
${cert.file} -keystore ${keystore.file} -storepass ${keystore.pass}
```

Example 1-16 Exporting a Certificate From a JKS Keystore in PKCS7 Format

The reply format defined by the Public Key Cryptography Standards #7, Cryptographic Message Syntax Standard, includes the supporting certificate chain in addition to the issued certificate.

```
keytool -export -noprompt -alias ${cert.alias} -file ${cert.file}
-keystore ${keystore.file} -storepass ${keystore.pass}
```

Example 1-17 Exporting a Certificate From a JKS Keystore in RFC/Text Format

```
keytool -export -noprompt -rfc -alias ${cert.alias} -file
${cert.file} -keystore ${keystore.file} -storepass ${keystore.pass}
```

See Also

To Delete a Certificate by Using `keytool`

Use the `keytool delete` command to delete an existing certificate.

Delete a certificate using the following command format:

```
keytool -delete
 -alias keyAlias
 -keystore keystore-name
 -storepass password
```

Example 1-18 Deleting a Certificate From a JKS Keystore

```
keytool -delete -noprompt -alias ${cert.alias} -keystore ${keystore.file}
-storepass ${keystore.pass}
```

See Also

Administering JACC Providers

The Java Authorization Contract for Containers (JACC) is part of the J2EE 1.4 specification that defines an interface for pluggable authorization providers. This enables the administrator to set up third-party plug-in modules to perform authorization.

Eclipse GlassFish includes Administration Console support and subcommands to support JACC providers, as follows:

- create `create-jacc-provider`
- delete `delete-jacc-provider`
- list `list-jacc-providers`

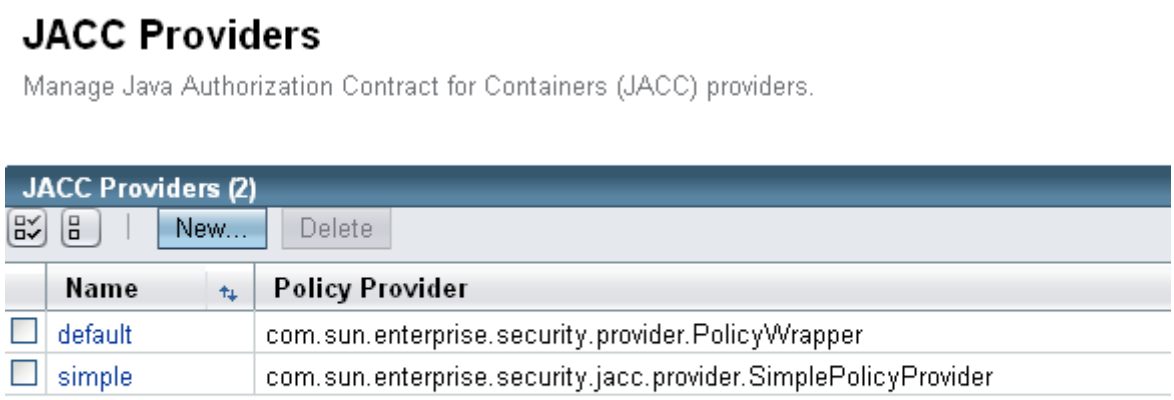
The default Eclipse GlassFish installation includes two JACC providers, named default and simple. You should not delete these default providers. Any JACC providers you create with the `create-jacc-provider` subcommand are in addition to these two default providers.

The Eclipse GlassFish creates a JSR-115-compliant JACC provider that you can use with third-party authorization modules for applications running in Eclipse GlassFish. The JACC provider is created as a `jacc-provider` element within the security-service element in the domain's `domain.xml` file.

Administering JACC Providers From the Administration Console

To use the Administration Console to administer JACC providers, perform the following steps:

1. Select Configurations and expand the entry.
2. Select the server configuration for which you want to administer JACC providers and expand the entry.
3. Select Security and expand the entry.
4. Select JACC Providers. The JACC Providers page is displayed. The existing JACC providers are shown on this page.



5. To create a new provider, click New.

Enter the Name, Policy Configuration (the class that implements the policy configuration factory) and the Policy Provider (the class that implements the policy factory) for the new JACC provider. You can also enter optional properties (name/value) for the provider.

6. To delete an existing JACC provider, select that provider and click Delete.

Administering JACC Providers from the Command Line

To use the command line to administer JACC providers, perform the following steps:

1. To create a JACC provider, use the `create-jacc-provider` subcommand. The following example shows how to create a JACC provider named testJACC on the default server target.

```
asadmin> create-jacc-provider
       --policyproviderclass
org.glassfish.exousia.modules.locked.SimplePolicyProvider
       --policyconfigfactoryclass com.sun.enterprise.security.provider.PolicyCon
figurationFactoryImpl
       testJACC
```

2. To delete a JACC provider, use the `create-jacc-provider` subcommand. The following example shows how to delete a JACC provider named testJACC from the default domain:

```
asadmin> delete-jacc-provider testJACC
```

3. To list the available providers, use the `list-jacc-providers` subcommand. The following example shows how to list JACC providers for the default domain:

```
asadmin> list-jacc-providers
default
simple
Command list-jacc-providers executed successfully.
```

2 Administering User Security

This chapter provides instructions for administering user security in the Eclipse GlassFish environment by using the `asadmin` command-line utility. Eclipse GlassFish enforces its authentication and authorization policies upon realms, users, and groups. This chapter assumes that you are familiar with security features such as authentication, authorization, and certificates. If you are not, see Administering System Security.

The following topics are addressed here:

- Administering Authentication Realms
- Administering File Users

Instructions for accomplishing these tasks by using the Administration Console are contained in the Administration Console online help.

JSR-375 defines the concept of an Identity Store, and an SPI interface for writing providers that can authenticate users against Identity Stores. It also provides two built-in providers. This mechanism is conceptually similar to Authentication Realms, but can be configured and managed by applications. See Working with Identity Stores in The Jakarta EE Tutorial for more information about Identity Stores.

Administering Authentication Realms

The following topics are addressed here:

- Overview of Authentication Realms
- To Create an Authentication Realm
- To List Authentication Realms
- To Update an Authentication Realm
- To Delete an Authentication Realm
- To Configure a JDBC or Digest Authentication Realm
- To Configure LDAP Authentication with OID
- To Configure LDAP Authentication with OVD
- To Enable LDAP Authentication on the Eclipse GlassFish DAS

Overview of Authentication Realms

An authentication realm, also called a security policy domain or security domain, is a scope over which the Eclipse GlassFish defines and enforces a common security policy. Eclipse GlassFish is preconfigured with the file, certificate, and administration realms. In addition, you can set up LDAP, JDBC, digest, Oracle Solaris, or custom realms. An application can specify which realm to use in its deployment descriptor. If the application does not specify a realm, Eclipse GlassFish uses its default

realm (`file`).

File realm

Eclipse GlassFish stores user credentials locally in a file named `keyfile`. The file realm is the initial default realm.

Administration realm

The administration realm is also a file realm and stores administrator user credentials locally in a file named `admin-keyfile`.

Certificate realm

Eclipse GlassFish stores user credentials in a certificate database. When using the certificate realm, the server uses certificates with the HTTPS protocol to authenticate web clients.

LDAP realm

Eclipse GlassFish can get user credentials from a Lightweight Directory Access Protocol (LDAP) server. LDAP is a protocol for enabling anyone to locate organizations, individuals, and other resources such as files and devices in a network, whether on the public Internet or on a corporate intranet.
See To Configure LDAP Authentication with OID for instructions on configuring Eclipse GlassFish to work with an OVD/OID LDAP provider.

By default, Eclipse GlassFish performs LDAP group search. If you have not created any groups in LDAP, the search fails.

To disable LDAP group search in LDAP user name search, set the `com.oracle.enterprise.security.auth.realm.ldap.DISABLEGROUP_SEARCH` Java system property to `true` in the required Eclipse GlassFish instance or cluster configurations:

```
asadmin> create-jvm-options --target=target
-Dcom.oracle.enterprise.security.auth.realm.ldap.DISABLEGROUP_SEARCH
=true
```

where target is the Eclipse GlassFish instance or cluster for which you are disabling LDAP group search. For more information about the `create-jvm-options` subcommand, see the Eclipse GlassFish Reference Manual.

JDBC realm

Eclipse GlassFish gets user credentials from a database. The server uses the database information and the enabled JDBC realm option in the configuration file.

Digest realm

Digest Authentication authenticates a user based on a user name and a password. However, the authentication is performed by transmitting the password in an encrypted form.

Oracle Solaris realm

Eclipse GlassFish gets user credentials from the Oracle Solaris operating system. This realm is supported on the Oracle Solaris 9 and Oracle Solaris 10 operating systems. Consult your Oracle Solaris documentation for information about managing users and groups in the Oracle Solaris realm.

PAM realm

A Pluggable Authentication Module (PAM) realm allows applications deployed on Eclipse GlassFish to authenticate users against a native Unix (Solaris/Linux/Mac OS) users list. PAM realms use the class name `com.sun.enterprise.security.ee.authentication.glassfish.pam.PamRealm` and the JAAS Context `pamRealm`.
This realm is supported on all Unix Operating Systems, including the Oracle Solaris 9 and Oracle Solaris 10 operating systems

Custom realm

You can create other repositories for user credentials, such as a relational database or third-party components. For more information about custom realms, see the Administration Console online help. For instructions on creating a custom realm, see " Creating a Custom Realm" in Eclipse GlassFish Application Development Guide.

The Eclipse GlassFish authentication service can govern users in multiple realms.

To Create an Authentication Realm

Use the `create-auth-realm` subcommand in remote mode to create an authentication realm.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create a realm by using the `create-auth-realm` subcommand.
   Information about properties for this subcommand is included in this help page.

Example 2-1 Creating a Realm

This example creates a realm named `db`.

```
asadmin> create-auth-realm --classname com.iplanet.ias.security.
auth.realm.DB.Database --property defaultuser=admin:Password=admin db
Command create-auth-realm executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help create-auth-realm` at the command line.

For information on creating a custom realm, see "Creating a Custom Realm" in Eclipse GlassFish Application Development Guide.

To List Authentication Realms

Use the `list-auth-realms` subcommand in remote mode to list the existing authentication realms.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List realms by using the `list-auth-realms` subcommand.

Example 2-2 Listing Realms

This example lists the authentication realms on `localhost`.

```
asadmin> list-auth-realms
db
certificate
file
admin-realm
Command list-auth-realms executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help list-auth-realms` at the command line.

To Update an Authentication Realm

Use the `set` subcommand to modify an existing authentication realm.

A custom realm does not require server restart.

1. List realms by using the `list-auth-realms` subcommand.
2. Modify the values for the specified thread pool by using the `set` subcommand.

   The thread pool is identified by its dotted name.

3. To apply your changes, restart Eclipse GlassFish.

   See "To Restart a Domain" in Eclipse GlassFish Administration Guide.

To Delete an Authentication Realm

Use the `delete-auth-realm` subcommand in remote mode to delete an existing authentication realm.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List realms by using the `list-auth-realms` subcommand.
3. If necessary, notify users that the realm is being deleted.
4. Delete the realm by using the `delete-auth-realm` subcommand.
5. To apply your changes, restart Eclipse GlassFish. See "To Restart a Domain" in Eclipse GlassFish

Administration Guide.

Example 2-3 Deleting a Realm

This example deletes an authentication realm named `db`.

```
asadmin> delete-auth-realm db
Command delete-auth-realm executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-auth-realm` at the command line.

To Configure a JDBC or Digest Authentication Realm

Eclipse GlassFish enables you to specify a user's credentials (user name and password) in the JDBC realm instead of in the connection pool. Using the `jdbc` type realm instead of the connection pool prevents other applications from browsing the database tables for user credentials.

By default, storage of passwords as clear text is not supported in the JDBC realm. Under normal circumstances, passwords should not be stored as clear text.

1. Create the database tables in which to store user credentials for the realm.
   How you create the database tables depends on the database that you are using.
2. Add user credentials to the database tables that you created.
   How you add user credentials to the database tables depends on the database that you are using.
3. Create a JDBC connection pool for the database.
   See " To Create a JDBC Connection Pool" in Eclipse GlassFish Administration Guide.
4. Create a JDBC resource for the database.
   " To Create a JDBC Resource" in Eclipse GlassFish Administration Guide.
5. Create a realm.
   For instructions, see To Create an Authentication Realm.

   The JAAS context should be `digestRealm` for digest authentication or `jdbcRealm` for other authentication types.

6. Modify the deployment descriptor to specify the `jdbc` realm.
   Modify the deployment descriptor that is associated with your application.
   - For an enterprise application in an Enterprise Archive (EAR) file, modify the `sun-application.xml` file.
   - For a web application in a Web Application Archive (WAR) file, modify the `web.xml` file.
   - For an enterprise bean in an EJB JAR file, modify the `sun-ejb-jar.xml` file.

For more information about how to specify a realm, see " How to Configure a Realm" in Eclipse GlassFish Application Development Guide.

7. Assign security roles to users in the realm.
   To assign a security role to a user, add a `security-role-mapping` element to the deployment descriptor that you modified.
8. Verify that the database is running.
   If needed, see "To Start the Database" in Eclipse GlassFish Administration Guide.
9. To apply the authentication, restart the server.
   See "To Restart a Domain" in Eclipse GlassFish Administration Guide.

Example 2-4 Assigning a Security Role

This example shows a `security-role-mapping` element that assigns the security role `Employee` to user `Calvin`

```
<security-role-mapping>
    <role-name>Employee</role-name>
    <principal-name>Calvin</principal-name>
  </security-role-mapping>
```

To Configure LDAP Authentication with OID

This procedure explains how to configure Eclipse GlassFish to use LDAP authentication with Oracle Internet Directory.

1. Install Oracle Enterprise Manager 11g and the latest Enterprise Manager patches, if they are not installed already.
   Instructions for installing Oracle Enterprise Manager are provided in the Oracle Enterprise Manager (`http://docs.oracle.com/cd/E11857_01/index.html`) documentation set.
2. Install the Oracle Identity Management Suite (IDM) 11g and Patch Set 2 or later, if they are not installed already.
   Instructions for installing the Oracle Identity Management suite are provided in Oracle Fusion Middleware Installation Guide for Oracle Identity Management (`http://docs.oracle.com/cd/E12839_01/install.1111/e12002.html`).
3. Configure SSL for Oracle Internet Directory (OID), if it is not configured already. Configure the OID instance in the server authentication mode and with the protocol version set to SSLv3
   Instructions for configuring SSL for OID are provided in the SSL chapter of Oracle Internet Directory Administrator's Guide (`http://docs.oracle.com/cd/B14099_19/idmanage.1012/b14082/ssl.html`).
4. Using Oracle Wallet Manager, export an SSL self-signed certificate you want to use with Eclipse GlassFish.
   Instructions for using Oracle Wallet Manager to create and export SSL certificates are provided in the "Configure Oracle Internet Directory for SSL" (`http://docs.oracle.com/cd/B14099_19/idmanage.1012/b14082/ssl.html#CHDCADIJ`) section of the SSL chapter in Oracle Internet Directory Administrator's Guide (`http://docs.oracle.com/cd/B14099_19/idmanage.1012/b14082/ssl.html`).

5. On the Eclipse GlassFish side, use the `keytool` command import the certificate you exported with Oracle Wallet Manager.
   The `keytool` command is available in the `$JAVA_HOME/bin` directory. Use the following syntax:

   ```
   keytool -importcert -alias "alias-name" -keystore domain-dir/config/cacerts.jks
   -file cert-name
   ```

   where the variables are defined as follows:

   alias-name

   Name of an alias to use for the certificate

   domain-dir

   Name of the domain for which the certificate is used

   cert-name

   Path to the certificate that you exported with Oracle Wallet Manager.

   For example, to import a certificate named `oi.cer` for a Eclipse GlassFish domain in `/glassfish7/glassfish/domains/domain1`, using an alias called "OID self-signed certificate," you would use the following command:

   ```
   keytool -importcert -alias "OID self signed certificate" -keystore \
   /glassfish7/glassfish/domains/domain1/config/cacerts.jks -file oid.cer
   ```

6. Restart the Eclipse GlassFish domain.
   See "To Restart a Domain" in Eclipse GlassFish Administration Guide.
7. Use the Oracle Enterprise Manager `ldapmodify` command to enable Anonymous Bind for OID.
   For example:

   ```
   ldapmodify -D cn=orcladmin -q -p portNum -h hostname -f ldifFile
   ```

   In this example, the LDIF file might contain the following:

   ```
   dn: cn=oid1,cn=osdldapd,cn=subconfigsubentry
   changetype: modify
   replace: orclAnonymousBindsFlag
   orclAnonymousBindsFlag: 1
   ```

   To disable all anonymous binds, you would use a similar LDIF file with the last line changed to:

   ```
   orclAnonymousBindsFlag: 0
   ```

   See "Managing Anonymous Binds" (http://docs.oracle.com/cd/E14571_01/oid.1111/e10029/

authentication.html#CACJEJDA) in Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory (http://docs.oracle.com/cd/E14571_01/oid.1111/e10029.html) for complete instructions on the `ldapmodify` command.

To Configure LDAP Authentication with OVD

This procedure explains how to configure Eclipse GlassFish to use LDAP authentication with Oracle Virtual Directory.

1. Create the OVD adapter, as described in the "Creating and Configuring Oracle Virtual Directory Adapters" (http://docs.oracle.com/cd/E12839_01/oid.1111/e10046/basic_adapters.html#BABCBGJA) chapter of Administrator's Guide for Oracle Virtual Directory (http://docs.oracle.com/cd/E12839_01/oid.1111/e10046.html).
2. Configure SSL for Oracle Virtual Directory (OVD), if it is not configured already. For instructions on configuring SSL for OVD, see the section "Enable SSL for Oracle Virtual Directory Using Fusion Middleware Control" in SSL Configuration in Oracle Fusion Middleware (http://docs.oracle.com/cd/E12839_01/core.1111/e10105/sslconfig.html#ASADM1800).

   Also, configure the SSL for the OVD listener in server authentication mode.
3. Export the certificate from JKS keystore you want to use with Eclipse GlassFish. See "Exporting a Keystore Using Fusion Middleware Control" (http://docs.oracle.com/cd/E16764_01/core.1111/e10105/wallets.html#CIHECAIB) for information.
4. On the Eclipse GlassFish side, use the `keytool` command to import the certificate you exported from the JKS keystore.
   The `keytool` command is available in the `$JAVA_HOME/bin` directory. Use the following syntax:

   ```
   keytool -importcert -alias "alias-name" -keystore domain-dir/config/cacerts.jks
   -file cert-name
   ```

   where the variables are defined as follows:

   alias-name

   Name of an alias to use for the certificate

   domain-dir

   Name of the domain for which the certificate is used

   cert-name

   Path to the certificate that you exported from the keystore.

   For example, to import a certificate named `ovd.cer` for a Eclipse GlassFish domain in `/glassfish7/glassfish/domains/domain1`, using an alias called "OVD self-signed certificate," you would use the following command:

   ```
   keytool -importcert -alias "OVD self signed certificate" -keystore \
   ```

```
/glassfish7/glassfish/domains/domain1/config/cacerts.jks -file ovd.cer
```

5. Restart the Eclipse GlassFish domain.
   See "To Restart a Domain" in Eclipse GlassFish Administration Guide.

To Enable LDAP Authentication on the Eclipse GlassFish DAS

This procedure explains how to enable LDAP authentication for logins to the Eclipse GlassFish Domain Administration Server (DAS). Logging in to the DAS is typically only performed by Eclipse GlassFish administrators who want to use the Eclipse GlassFish Administration Console or `asadmin` command. See To Configure LDAP Authentication with OID for instructions on enabling general LDAP authentication for Eclipse GlassFish.

Before you begin, ensure that you have followed the configuration instructions in To Configure LDAP Authentication with OID

Use the `asadmin configure-ldap-for-admin` subcommand to enable user authentication to the Eclipse GlassFish DAS.

Use the following syntax:

```
asadmin configure-ldap-for-admin --basedn "dn-list" --url [ldap|ldaps]://ldap-url
--ldap-group group-name
```

where the variables are defined as follows:

dn-list
: basedn parameters

ldap-url
: URL and port number for the LDAP server; can use standard (`ldap`) or secure (`ldaps`) protocol

group-name
: LDAP group name for allowed users, as defined on the LDAP server.

For example:

```
asadmin configure-ldap-for-admin --basedn "dc=red,dc=iplanet,dc=com" \
--url ldap://interopoel54-1:3060 --ldap-group sqestaticgroup

asadmin configure-ldap-for-admin --basedn "dc=red,dc=iplanet,dc=com" \
--url ldaps://interopoel54-1:7501 --ldap-group sqestaticgroup
```

See Also

See `configure-ldap-for-admin` for more information about the `configure-ldap-for-admin subcommand`.

Administering File Users

A user is an individual (or application program) identity that is defined in Eclipse GlassFish. A user who has been authenticated is sometimes called a principal.

As the administrator, you are responsible for integrating users into the Eclipse GlassFish environment so that their credentials are securely established and they are provided with access to the applications and services that they are entitled to use.

The following topics are addressed here:

- To Create a File User
- To List File Users
- To List File Groups
- To Update a File User
- To Delete a File User

To Create a File User

Use the `create-file-user` subcommand in remote mode to create a new user by adding a new entry to the `keyfile`. The entry includes the user name, password, and any groups for the user. Multiple groups can be specified by separating the groups with colons (:).

If secure administration is enabled as described in Running Secure Admin, you cannot create an administrative user with a blank password.

Creating a new `file` realm user is a dynamic event and does not require server restart.

1. Ensure that the server is running. Remote subcommands require a running server.
2. If the user will belong to a particular group, see the current groups by using the `list-file-groups` subcommand.
3. Create a file user by using the `create-file-user` subcommand.

Example 2-5 Creating a User

This example create user `Jennifer` on the default realm `file` (no groups are specified).

The `asadmin --passwordfile` option specifies the name of a file that contains the password entries in a specific format. The entry for a password must have the `AS_ADMIN_` prefix followed by the password name in uppercase letters, an equals sign, and the password. See `asadmin`(1M) for more information.

```
asadmin> create-file-user --user admin
--passwordfile=c:\tmp\asadminpassword.txt Jennifer
Command create-file-user executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help create-file-user` at the command line.

To List File Users

Use the `list-file-users` subcommand in remote mode to list the users that are in the `keyfile`.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List users by using the `list-file-users` subcommand.

Example 2-6 Listing File Users

This example lists file users on the default `file` realm file.

```
asadmin> list-file-users
Jennifer
Command list-file-users executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help list-file-users` at the command line.

To List File Groups

A group is a category of users classified by common traits, such as job title or customer profile. For example, users of an e-commerce application might belong to the `customer` group, and the big spenders might also belong to the `preferred` group. Categorizing users into groups makes it easier to control the access of large numbers of users. A group is defined for an entire server and realm. A user can be associated with multiple groups of users.

A group is different from a role in that a role defines a function in an application, while a group is a set of users who are related in some way. For example, in the personnel application there might be groups such as `full-time`, `part-time`, and `on-leave`. Users in these groups are all employees (the `employee` role). In addition, each user has its own designation that defines an additional level of employment.

Use the `list-file-groups` subcommand in remote mode to list groups for a file user, or all file groups if the `--name` option is not specified.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List file groups by using the `list-file-groups` subcommand.

Example 2-7 Listing Groups for a User

This example lists the groups for user `joesmith`.

```
asadmin> list-file-groups --name joesmith
staff
manager
Command list-file-groups executed successfully
```

To Update a File User

Use the `update-file-user` subcommand in remote mode to modify the information in the `keyfile` for a specified user.

If secure administration is enabled as described in Running Secure Admin, you cannot update an administrative user to have a blank password.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Update the user information by using the `update-file-user` subcommand.
3. To apply your changes, restart Eclipse GlassFish.
   See "To Restart a Domain" in Eclipse GlassFish Administration Guide.

Example 2-8 Updating a User

The following subcommand updates the groups for user `Jennifer`.

```
asadmin> update-file-user --passwordfile c:\tmp\asadminpassword.txt --groups
staff:manager:engineer Jennifer
Command update-file-user executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help update-file-user` at the command line.

To Delete a File User

Use the `delete-file-user` subcommand in remote mode to remove a user entry from the `keyfile` by specifying the user name. You cannot delete yourself, that is, the user you are logged in as cannot be deleted during your session.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List users by using the `list-file-users` subcommand.
3. Delete the user by using the `delete-file-user` subcommand.

Example 2-9 Deleting a User

This example deletes user `Jennifer` from the default `file` realm.

```
asadmin> delete-file-user Jennifer
```

```
Command delete-file-user executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-file-user` at the command line.

3 Administering Message Security

This chapter provides information and procedures on configuring the message layer security for web services in the Eclipse GlassFish environment.

Message security (JSR 196) is supported only in the Full Platform Profile of Eclipse GlassFish, not in the Web Profile.

The following topics are addressed here:

- About Message Security in Eclipse GlassFish
- Enabling Default Message Security Providers for Web Services
- Configuring Message Protection Policies
- Administering Non-default Message Security Providers
- Enabling Message Security for Application Clients
- Additional Information About Message Security

Some of the material in this chapter assumes a basic understanding of security and web services concepts. For more information about security, see About System Security in Eclipse GlassFish.

Instructions for accomplishing the tasks in this chapter by using the Administration Console are contained in the Administration Console online help.

About Message Security in Eclipse GlassFish

Message security enables a server to perform end-to-end authentication of web service invocations and responses at the message layer. Security information is inserted into messages so that it travels through the networking layers and arrives with the intact message at the message destination(s). Message security differs from transport layer security in that message security can be used to decouple message protection from message transport so that messages remain protected after transmission.

Web services deployed on Eclipse GlassFish are secured by binding SOAP layer message security providers and message protection policies to the containers in which the applications are deployed, or to web service endpoints served by the applications. SOAP layer message security functionality is configured in the client-side containers of Eclipse GlassFish by binding SOAP layer message security providers and message protection policies to the client containers or to the portable service references declared by client applications.

Message-level security can be configured for the entire Eclipse GlassFish or for specific applications or methods. Configuring message security at the application level is discussed in the Eclipse GlassFish Application Development Guide.

The following topics are addressed here:

- Security Tokens and Security Mechanisms

- Authentication Providers
- Message Protection Policies
- Application-Specific Web Services Security
- Message Security Administration
- Sample Application for Web Services

Security Tokens and Security Mechanisms

WS-Security is a specification that provides a communications protocol for applying security to web services. The security mechanisms implement the specification. Web Services Interoperability Technologies (WSIT) implements WS-Security so as to provide interoperable message content integrity and confidentiality, even when messages pass through intermediary nodes before reaching their destination endpoint. WS-Security as provided by WSIT is in addition to existing transport-level security, which can still be used.

The Simple Object Access Protocol (SOAP) layer message security providers installed with Eclipse GlassFish can be used to employ username/password and X.509 certificate security tokens to authenticate and encrypt SOAP web services messages.

- Username Tokens. Eclipse GlassFish uses username tokens in SOAP messages to authenticate the message sender. The recipient of a message containing a username token (within embedded password) validates that the message sender is authorized to act as the user (identified in the token) by confirming that the sender knows the password of the user.

  When using a username token, a valid user database must be configured on Eclipse GlassFish.

- Digital Signatures. Eclipse GlassFish uses XML digital signatures to bind an authentication identity to message content. Clients use digital signatures to establish their caller identity. Digital signatures are verified by the message receiver to authenticate the source of the message content (which might be different from the sender of the message.)

  When using digital signatures, valid keystore and truststore files must be configured on Eclipse GlassFish.

- Encryption. The purpose of encryption is to modify the data so that it can only be understood by its intended audience. This is accomplished by substituting an encrypted element for the original content. When based on public key cryptography, encryption can be used to establish the identity of the parties who are authorized to read a message.

  When using encryption, a Java Cryptography Extension (JCE) provider that supports encryption must be installed.

Authentication Providers

The authentication layer is the message layer on which authentication processing must be performed. Eclipse GlassFish enforces web services message security at the SOAP layer. The types of authentication that are supported include the following:

- Sender authentication, including username-password authentication
- Content authentication, including XML digital signatures

Eclipse GlassFish invokes authentication providers to process SOAP message layer security. The message security providers provide information such as the type of authentication that is required for the request and response messages. The following message security providers are included with Eclipse GlassFish:

- Client-side Provider. A client-side provider establishes (by signature or username/password) the source identity of request messages and/or protects (by encryption) request messages such that they can only be viewed by their intended recipients. A client-side provider also establishes its container as an authorized recipient of a received response (by successfully decrypting it) and validates passwords or signatures in the response to authenticate the source identity associated with the response. Client-side providers configured in Eclipse GlassFish can be used to protect the request messages sent and the response messages received by server-side components (servlets and EJB components) acting as clients of other services.

  The default client provider is used to identify the client—side provider to be invoked for any application for which a specific client provider has not been bound.

- Server-side Provider. A server-side provider establishes its container as an authorized recipient of a received request (by successfully decrypting it), and validates passwords or signatures in the request to authenticate the source identity associated with the request. A server-side provider also establishes (by signature or username/password) the source identity of response messages and/or protects (by encryption) response messages such that they can only be viewed by their intended recipients. Server-side providers are only invoked by server-side containers.

The default server provider is used to identify the server—side provider to be invoked for any application for which a specific server provider has not been bound.

Message Protection Policies

A request policy defines the authentication policy requirements associated with request processing performed by the authentication provider. Policies are expressed in message sender order such that a requirement that encryption occur after content would mean that the message receiver would expect to decrypt the message before validating the signature. The response policy defines the authentication policy requirements associated with response processing performed by the authentication provider.

Message protection policies are defined for request message processing and response message processing. The policies are expressed in terms of requirements for source and/or recipient authentication. The providers apply specific message security mechanisms to cause the message protection policies to be realized in the context of SOAP web services messages.

- Source Authentication Policy. A source authentication policy represents a requirement that the identity of the entity that sent a message or that defined the content of a message be established in the message such that it can be authenticated by the message receiver.
- Recipient Authentication Policy. A recipient authentication policy represents a requirement that the message be sent such that the identity of the entities that can receive the message can be

established by the message sender.

Request and response message protection policies are defined when a security provider is configured into a container. Application-specific message protection policies (at the granularity of the web service port or operation) can also be configured within the Eclipse GlassFish deployment descriptors of the application or application client. In any situation where message protection policies are defined, the request and response message protection policies of the client must be equivalent t) the request and response message protection policies of the server. For more information about defining application-specific message protection policies, see "Securing Applications" in Eclipse GlassFish Application Development Guide.

Application-Specific Web Services Security

Application-specific web services security functionality is configured (at application assembly) by defining the `message-security-binding` elements in the Eclipse GlassFish deployment descriptors of the application. These `message-security-binding` elements are used to associate a specific security provider or message protection policy with a web service endpoint or service reference, and might be qualified so that they apply to a specific port or method of the corresponding endpoint or referenced service.

For information about defining application-specific message protection policies, see "Securing Applications" in Eclipse GlassFish Application Development Guide.

Message Security Administration

When Eclipse GlassFish is installed, SOAP layer message security providers are configured in the client and server-side containers of Eclipse GlassFish, where they are available for binding for use by the containers, or by individual applications or clients deployed in the containers. During installation, the default providers are configured with a simple message protection policy that, if bound to a container, or to an application or client in a container, would cause the source of the content in all request and response messages to be authenticated by XML digital signature.

Eclipse GlassFish administrative interfaces can be used as follows:

- To modify the message protection policies enforced by the providers
- To bind the existing providers for use by the server-side containers of Eclipse GlassFish
- To create new security provider configurations with alternative message protection policies

Analogous administrative operations can be performed on the SOAP message layer security configuration of the application client container. If you want web services security to protect all web services applications deployed on Eclipse GlassFish. See Enabling Message Security for Application Clients.

By default, message layer security is disabled on Eclipse GlassFish. To configure message layer security for the Eclipse GlassFish see Enabling Default Message Security Providers for Web Services.

In most cases, you must restart Eclipse GlassFish after performing administrative tasks. This is especially true if you want the effects of the administrative change to be applied to applications that were already deployed on Eclipse GlassFish at the time the operation was performed.

Message Security Tasks

The general implementation tasks for message security include some or all of the following:

1. If you are using a version of the Java SDK prior to version 1.5.0, and using encryption technology, configuring a JCE provider
2. If you are using a username token, verifying that a user database is configured for an appropriate realm

   When using a username/password token, an appropriate realm must be configured and a user database must be configured for the realm.

3. Managing certificates and private keys, if necessary
4. Enabling the Eclipse GlassFish default providers
5. Configuring new message security providers

Message Security Roles

In Eclipse GlassFish, the administrator and the application deployer are expected to take primary responsibility for configuring message security. In some situations, the application developer might also contribute.

System Administrator

The system administrator is responsible for the following message security tasks:

- Administering server security settings and certificate databases
- Administering keystore and truststore files
- Configuring message security providers on Eclipse GlassFish
- Turning on message security
- (If needed) Installing the samples server

Application Deployer

The application deployer is responsible for the following message security tasks:

- Specifying (at application reassembly) any required application-specific message protection policies if such policies have not already been specified by the developer/assembler.
- Modifying Eclipse GlassFish deployment descriptors to specify application-specific message protection policies information (message-security-binding elements) to web service endpoint and service references.

Application Developer/Assembler

The application developer/assembler is responsible for the following message security tasks:

- Determining if an application-specific message protection policy is required by the application

If so, the developer ensures that the required policy is specified at application assembly time.

- Specifying how web services should be set up for message security

  Message security can be set up by the administrator so that all web services are secured, or by the application deployer when the security provider or protection policy bound to the application must be different from that bound to the container.

- Turning on message security if authorized to do so by the administrator

Sample Application for Web Services

Eclipse GlassFish includes a sample application named `xms`. The `xms` application features a simple web service that is implemented by both a Jakarta EE EJB endpoint and a Java servlet endpoint. Both endpoints share the same service endpoint interface. The service endpoint interface defines a single operation, `sayHello`, which takes a string argument, and returns a `String` composed by pre-pending `Hello` to the invocation argument.

The `xms` sample application is provided to demonstrate the use of Eclipse GlassFish WS-Security functionality to secure an existing web services application. The instructions which accompany the sample describe how to enable the WS-Security functionality of Eclipse GlassFish such that it is used to secure the `xms` application. The sample also demonstrates the binding of WS-Security functionality directly to the application as described in Application-Specific Web Services Security application.

For information about compiling, packaging, and running the `xms` sample application, " Securing Applications" in Eclipse GlassFish Application Development Guide.

The `xms` sample application is installed in the following directory: as-install`/samples/webservices/security/ejb/apps/xms/`

Enabling Default Message Security Providers for Web Services

By default, message security is disabled on Eclipse GlassFish. Default message security providers have been created, but are not active until you enable them. After the providers have been enabled, message security is enabled.

The following topics are addressed here:

- To Enable a Default Server Provider
- To Enable a Default Client Provider

To Enable a Default Server Provider

To enable message security for web services endpoints deployed in Eclipse GlassFish, you must specify a security provider to be used by default on the server side. If you enable a default provider for message security, you also need to enable providers to be used by clients of the web services deployed in Eclipse GlassFish.

1. Specify the default server provider by using the `set` subcommand.

   Use the following syntax:

   ```
   asadmin set --port admin-port
   server-config.security-service.message-security-config.SOAP.
   default_provider=ServerProvider
   ```

2. To apply your changes to applications that are already running, restart Eclipse GlassFish.

   See "To Restart a Domain" in Eclipse GlassFish Administration Guide.

To Enable a Default Client Provider

To enable message security for web service invocations originating from deployed endpoints, you must specify a default client provider. If you enabled a default client provider for Eclipse GlassFish, you must ensure that any services invoked from endpoints deployed in Eclipse GlassFish are compatibly configured for message layer security.

1. Specify the default client provider by using the `set` subcommand.

   Use the following syntax:

   ```
   asadmin set --port admin-port
   server-config.security-service.message-security-config.SOAP.
   default_client_provider=ClientProvider
   ```

2. To apply your changes to applications that are already running, restart Eclipse GlassFish.

   See "To Restart a Domain" in Eclipse GlassFish Administration Guide.

Configuring Message Protection Policies

Message protection policies are defined for request message processing and response message processing. The policies are expressed in terms of requirements for source and/or recipient authentication. The providers apply specific message security mechanisms to cause the message protection policies to be realized in the context of SOAP web services messages.

The following topics are addressed here:

- Message Protection Policy Mapping
- To Configure the Message Protection Policies for a Provider
- Setting the Request and Response Policy for the Application Client Configuration

Message Protection Policy Mapping

The following table shows message protection policy configurations and the resulting message

security operations performed by the WS-Security SOAP message security providers for that configuration.

Table 3-1 Message Protection Policy Mapping to WS-Security SOAP Operations

Message Protection Policy	Resulting WS-Security SOAP Message Protection Operations
auth-source="sender"	The message contains a `wsse:Security` header that contains a `wsse:UsernameToken` (with password).
auth-source="content"	The content of the SOAP message Body is signed. The message contains a `wsse:Security` header that contains the message Body signature represented as a `ds:`Signature`.
auth-source="sender" auth-recipient="before-content" OR auth-recipient="after-content"	The content of the SOAP message Body is encrypted and replaced with the resulting `xend:EncryptedData`. The message contains `a wsse:Security` header that contains a `wsse:UsernameToken (with password)` and an `xenc:EncryptedKey`. The `xenc:EncryptedKey` contains the key used to encrypt the SOAP message body. The key is encrypted in the public key of the recipient.
auth-source="content" auth-recipient="before-content"	The content of the SOAP message Body is encrypted and replaced with the resulting `xend:EncryptedData`. The `xenc:EncryptedData` is signed. The message contains `a wsse:Security` header that contains an `xenc:EncryptedKey` and a `ds:`Signature`. The `xenc:EncryptedKey` contains the key used to encrypt the SOAP message body. The key is encrypted in the public key of the recipient.
auth-source="content" auth-recipient="after-content"	The content of the SOAP message Body is signed, then encrypted, and then replaced with the resulting `xend:EncryptedData`. The message contains a `wsse:Security` header that contains an `xenc:EncryptedKey` and a `ds:Signature`. The `xenc:EncryptedKey` contains the key used to encrypt the SOAP message body. The key is encrypted in the public key of the recipient.
auth-recipient="before-content" OR auth-recipient="after-content"	The content of the SOAP message Body is encrypted and replaced with the resulting `xend:EncryptedData`. The message contains `a wsse:Security` header that contains an `xenc:EncryptedKey`. The `xenc:EncryptedKey` contains the key used to encrypt the SOAP message body. The key is encrypted in the public key of the recipient.
No policy specified.	No security operations are performed by the modules.

To Configure the Message Protection Policies for a Provider

Typically, you would not reconfigure a provider. However, if needed for your situation, you can modify a provider's message protection policies by changing provider type, implementation class, and provider-specific configuration properties. To understand the results of different combinations, see Table 3-1.

Use the `set` subcommand to set the response policy, then replace the word `request` in the following commands with the word `response`.

1. Add a request policy to the client and set the authentication source by using the `set` subcommand.

   For example:

   ```
   asadmin> set server-config.security-service.message-security-config.SOAP.
   provider-config.ClientProvider.request-policy.auth_source=[sender | content]
   ```

2. Add a request policy to the server and set the authentication source by using the `set` subcommand.

   For example:

   ```
   asadmin> set server-config.security-service.message-security-config.SOAP.
   provider-config.ServerProvider.request-policy.auth_source=[sender | content]
   ```

3. Add a request policy to the client and set the authentication recipient by using the `set` subcommand:

   For example:

   ```
   asadmin> set server-config.security-service.message-security-config.SOAP.
   provider-config.ClientProvider.request-policy.auth_recipient=[before-content |
   after-content]
   ```

4. Add a request policy to the server and set the authentication recipient by using the `set` subcommand:

   For example:

   ```
   asadmin> set server-config.security-service.message-security-config.SOAP.
   provider-config.ServerProvider.request-policy.auth_recipient=[before-content |
   after-content]
   ```

Setting the Request and Response Policy for the Application Client Configuration

The request and response policies define the authentication policy requirements associated with request and response processing performed by the authentication provider. Policies are expressed in message sender order such that a requirement that encryption occur after content would mean that the message receiver would expect to decrypt the message before validating the signature.

To achieve message security, the request and response policies must be enabled on both the server

and client. When configuring the policies on the client and server, make sure that the client policy matches the server policy for request/response protection at application-level message binding.

To set the request policy for the application client configuration, modify the Eclipse GlassFish-specific configuration for the application client container as described in Enabling Message Security for Application Clients.

Example 3-1 Message Security Policy Setting for Application Clients

In the application client configuration file, the `request-policy` and `response-policy` elements are used to set the request policy, as shown in the following code snippet. (Additional code in the snippet is provided as illustration and might differ slightly in your installation. Do not change the additional code.)

```
<client-container>
  <target-server name="your-host" address="your-host"
      port="your-port"/>
  <log-service file="" level="WARNING"/>
  <message-security-config auth-layer="SOAP"
      default-client-provider="ClientProvider">
    <provider-config
        class-name="com.sun.enterprise.security.jauth.ClientAuthModule"
        provider-id="clientprovider" provider-type="client">
      <request-policy auth-source="sender | content"
        auth-recipient="after-content | before-content"/>
      <response-policy auth-source="sender | content"
        auth-recipient="after-content | before-content"/>
       <property name="security.config"
           value="as-install/lib/appclient/wss-client-config.xml"/>
    </provider-config>
  </message-security-config>
</client-container>
```

Valid values for `auth-source` include `sender` and `content`. Valid values for `auth-recipient` include `before-content` and `after-content`. A table describing the results of various combinations of these values can be found in Configuring Message Protection Policies.

To not specify a request or response policy, leave the element blank, for example:

```
<response-policy/>
```

Administering Non-default Message Security Providers

The following topics are addressed here:

- To Create a Message Security Provider

- To List Message Security Providers
- To Update a Message Security Provider
- To Delete a Message Security Provider
- To Configure a Servlet Layer Server Authentication Module (SAM)

To Create a Message Security Provider

Use the `create-message-security-provider` subcommand in remote mode to create a new message provider for the security service. If the message layer does not exist, the message layer is created, and the provider is created under it.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create the message security provider by using the `create-message-security-provider` subcommand.

   Information about properties for this subcommand is included in the help page.
3. If needed, restart the server.

   Some properties require server restart. See "Configuration Changes That Require Restart" in Eclipse GlassFish Administration Guide. If your server needs to be restarted, see "To Restart a Domain" in Eclipse GlassFish Administration Guide.

Example 3-2 Creating a Message Security Provider

This example creates the new message security provider `mySecurityProvider`.

```
asadmin> create-message-security-provider
--classname com.sun.enterprise.security.jauth.ClientAuthModule
--providertype client mySecurityProvider
Command create-message-security-provider executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help create-message-security-provider` at the command line.

To List Message Security Providers

Use the `list-message-security-providers` subcommand in remote mode to list the message providers for the security layer.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the message security providers by using the `list-message-security-providers` subcommand.

Example 3-3 Listing Message Security Providers

This example lists the message security providers for a message layer.

```
asadmin> list-message-security-providers --layer SOAP
XWS_ClientProvider
ClientProvider
XWS_ServerProvider
ServerProvider
Command list-message-security-providers executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help list-message-security-providers` at the command line.

To Update a Message Security Provider

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the message security providers by using the `list-message-security-providers` subcommand.
3. Modify the values for the specified message security provider by using the `set` subcommand.

   The message security provider is identified by its dotted name.

To Delete a Message Security Provider

Use the `delete-message-security-provider` subcommand in remote mode to remove a message security provider.

1. Ensure that the server is running. Remote subcommands require a running server.
2. List the message security providers by using the `list-message-security-providers` subcommand.
3. Delete the message security provider by using the `delete-message-security-provider` subcommand.

Example 3-4 Deleting a Message Security Provider

This example deletes the `myServerityProvider` message security provider.

```
asadmin> delete-message-security-provider --layer SOAP myServerityProvider
Command delete-message-security-provider executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help delete-message-security-provider` at the command line.

To Configure a Servlet Layer Server Authentication Module (SAM)

You configure a JSR 196 Server Authentication Module (SAM) as an HttpServlet-layer message security provider, either through the Administration Console or with the `create-message-security-`

`provider` subcommand.

1. Ensure that the server is running. Remote subcommands require a running server.
2. Create the message security provider by using the `create-message-security-provider` subcommand.

   Information about properties for this subcommand is included in the help page.
3. Bind the message security provider for use with your application.

   You do this by defining the `httpservlet-security-provider` attribute in the `glassfish-web.xml` file corresponding to your application. Set the value of the attribute to the provider name you assigned to the message security provider. For example, if you use MySAM when you create the message security provider the entry would be httpservlet-security-provider="MySAM".
4. If needed, restart the server.

   Some properties require server restart. See "Configuration Changes That Require Restart" in Eclipse GlassFish Administration Guide. If your server needs to be restarted, see "To Restart a Domain" in Eclipse GlassFish Administration Guide.

Example 3-5 Creating a Message Security Provider

This example creates the new message security provider `mySAM`.

```
asadmin> create-message-security-provider --layer=HttpServlet
--classname com.sun.glassfish.oamsam.OAMAuthenticatorSAM
--providertype server
--property oam.resource.hostid.variation="your-host-system.com" mySAM
Creation of message security provider named mySAM completed successfully
Command create-message-security-provider executed successfully.
```

The subcommand results in the following `domain.xml` entry:

```
<message-security-config auth-layer="HttpServlet">
<provider-config provider-type="server" provider-id="mysam"
class-name="com.sun.glassfish.oamsam.OAMAuthenticatorSAM">
            <property name="oam.resource.hostid.variation" value="your-host-
system.com"></property>
            <request-policy></request-policy>
            <response-policy></response-policy>
          </provider-config>
</message-security-config>
```

To list the HttpServlet message security providers, use the `list-message-security-providers` subcommand:

```
asadmin> list-message-security-providers --layer HttpServlet
```

```
list-message-security-providers successful
GFConsoleAuthModule
mySAM
Command list-message-security-providers executed successfully.
```

See Also

You can also view the full syntax and options of the subcommand by typing `asadmin help create-message-security-provider` at the command line.

Enabling Message Security for Application Clients

The message protection policies of client providers must be configured such that they are equivalent to the message protection policies of the server-side providers they will be interacting with. This is already the situation for the providers configured (but not enabled) when Eclipse GlassFish is installed.

To enable message security for client applications, modify the Eclipse GlassFish specific configuration for the application client container. The process is analogous to the process in Configuring Message Protection Policies.

Additional Information About Message Security

For additional information about message security, see the following documentation:

- Introduction to Security in the Jakarta EE Platform in The Jakarta EE Tutorial
- "Securing Applications" in Eclipse GlassFish Application Development Guide

4 Administering Security in Cluster Mode

This chapter describes important information about administering security in a cluster.

The following topics are described:

- Configuring Certificates in Cluster Mode
- Dynamic Reconfiguration
- Understanding Synchronization

This chapter assumes that you are familiar with security features such as authentication, authorization, and certificates. If you are not, see Administering System Security.

Instructions for accomplishing the tasks specific to Eclipse GlassFish by using the Administration Console are contained in the Administration Console online help.

Configuring Certificates in Cluster Mode

The sections Certificates and SSL and Administering JSSE Certificates describe the relevant concepts and use of certificates in Eclipse GlassFish.

By default, Eclipse GlassFish uses self-signed certificates. The self-signed certificates that Eclipse GlassFish uses might not be trusted by clients by default because a certificate authority does not vouch for the authenticity of the certificate.

You can instead use your own certificates, as described in Using Your Own Certificates.

Dynamic Reconfiguration

Administrative commands that you execute on the domain administration server (DAS) must either be replicated on the affected server instances, or on all server instances that are part of the cluster. Eclipse GlassFish replicates the commands by sending the same administration command request that was sent to the DAS to the server instances. As a result of replicating the commands on the DAS and the individual instances, the DAS and the instances make the same changes to their respective copies of the domain's configuration.

Oracle recommends that you enable secure admin as described in Managing Administrative Security so that Eclipse GlassFish securely transfers these files on the network.

Dynamic reconfiguration refers to using the `--target` operand to CLI subcommands to make a change to a server instance (if the user-specified target is a server instance), or all server instances that are part of the cluster (if the user-specified target is a cluster). For example:`asadmin create-jdbc-resource some-options --target some-target`.

The `--target` operand allows the following values:

- `server` - Performs the command on the default server instance. This is the default value.

- configuration_name - Performs the command in the specified configuration.
- cluster_name - Performs the command on all server instances in the specified cluster.
- instance_name - Performs the command on a specified server instance.

If a command fails for a cluster, the status shows all server instances where dynamic reconfiguration failed, and suggests corrective next steps.

The command status also shows when a restart is required for each server instance.

The `--target` operand is supported for the following security-related CLI subcommands:

- create-jacc-provider
- delete-jacc-provider
- list-jacc-providers
- create-audit-module
- create-auth-realm
- create-file-user
- delete-audit-module
- delete-auth-realm
- delete-file-user
- update-file-user
- create-message-security-provider
- delete-message-security-provider
- list-audit-modules
- list-file-groups
- list-file-users
- login

Enabling Dynamic Configuration

Dynamic configuration is enabled by default and no additional action is required.

Use the following command to enable dynamic configuration from the command line:

`asadmin --user user --passwordfile password-file set cluster-name-config.dynamic-reconfiguration-enabled=true`.

To enable dynamic configuration from the Administration Console, perform the following steps:

1. Expand the Configurations node.
2. Click the name of the cluster's configuration.
3. On the Configuration System Properties page, check the Dynamic Reconfiguration Enabled box.

4. Click Save.

The dynamic reconfiguration feature applies only to server instances, not the DAS. Therefore, you cannot "disable" dynamic reconfiguration on the DAS. There is no way to make changes to the DAS configuration using `asadmin` commands, the Administration Console, or the REST interface without having those changes take effect immediately.

Understanding Synchronization

As described in " Resynchronizing Eclipse GlassFish Instances and the DAS" in Eclipse GlassFish High Availability Administration Guide, configuration data for a Eclipse GlassFish instance is stored in the repository of the DAS and in a cache on the host that is local to the instance. The configuration data in these locations must be synchronized. The cache is synchronized only when a user uses the administration tools to start or restart an instance.

See " Resynchronizing Eclipse GlassFish Instances and the DAS" in Eclipse GlassFish High Availability Administration Guide for information about default synchronization for files and directories, for the steps required to resynchronize an instance and the DAS, and for additional synchronization topics.

5 Managing Administrative Security

This chapter describes how to manage administrative security by using the secure administration feature.

This chapter assumes that you are familiar with security features such as authentication, authorization, and certificates. If you are not, first see Administering System Security.

Instructions for accomplishing the tasks specific to Eclipse GlassFish by using the Administration Console are contained in the Administration Console online help.

- Secure Administration Overview
- How Secure Admin Works: The Big Picture
- Considerations When Running Eclipse GlassFish With Default Security
- Running Secure Admin
- Additional Considerations When Creating Local Instances
- Secure Admin Use Case
- Upgrading an SSL-Enabled Secure GlassFish Installation to Secure Admin

Secure Administration Overview

The secure administration feature allows an administrator to secure all administrative communication between the domain administration server (DAS), any remote instances, and administration clients such as the `asadmin` utility, the administration console, and REST clients.

In addition, secure administration helps to prevent DAS-to-DAS and instance-to-instance traffic, and carefully restricts administration-client-to-instance traffic.

The secure administration feature, which is henceforth referred to as secure admin, provides a secure environment, in which you can be confident that rogue users or processes cannot intercept or corrupt administration traffic or impersonate legitimate Eclipse GlassFish components.

When you install Eclipse GlassFish or create a new domain, secure admin is disabled by default. When secure admin is disabled, Eclipse GlassFish does not encrypt administrative communication among the system components and does not accept administrative connections from remote hosts.

The following subcommands enable and disable secure admin:

- enable-secure-admin
  The `enable-secure-admin` subcommand turns on secure admin. Eclipse GlassFish uses SSL encryption to protect subsequent administrative traffic and will accept remote administrative connections. Enabling secure admin affects the entire domain, including the DAS and all instances. The DAS must be running, and not any instances, when you run `enable-secure-admin`. You must restart the DAS immediately after enabling secure admin, and then start any instances you want to run.

The `enable-secure-admin` subcommand fails if any administrative user has a blank password.

- disable-secure-admin
  The `disable-secure-admin` subcommand turns off secure admin. Eclipse GlassFish no longer encrypts administrative messages and will no longer accept remote administration connections. Disabling secure admin affects the entire domain, including the DAS and all instances. The DAS must be running , and not any instances, when you run `disable-secure-admin`. You must restart the DAS immediately after disabling secure admin, and then start any instances you want to run.

  If secure admin is not enabled, this subcommand has no effect.

This section describes how to use these commands to run secure admin, and the implications of doing so.

How Secure Admin Works: The Big Picture

Secure admin is a domain-wide setting. It affects the DAS and all instances and all administration clients. This section describes the following topics:

- Functions Performed by Secure Admin
- Which Administration Account is Used?
- What Authentication Methods Are Used for Secure Administration?
- Understanding How Certificate Authentication is Performed
- What Certificates Are Used?
- An Alternate Approach: Using Distinguished Names to Specify Certificates
- Guarding Against Unwanted Connections

Functions Performed by Secure Admin

The `enable-secure-admin` subcommand performs the following functions. Subsequent sections describe these functions in more detail.

- Enables the secure admin behavior, optionally setting which aliases are to be used for identifying the DAS and instance certificates.
- Adjusts all configurations in the domain, including default-config.
- Adjusts Grizzly settings:
  - SSL/TLS is enabled in the DAS's admin listener and the instances' admin listeners.
  - Port unification (that is, HTTP and HTTPS are handled by the same port), http—to—https redirection, and client authentication (client-auth=want) are enabled.
  - Configures SSL to use the administration truststore.
  - Configures SSL to use the administration keystore and the correct alias (for the self-signed

cert) for authenticating itself. (You can use your own certificate instead, as described in Using Your Own Certificates.

The Grizzly configuration on the DAS and each instance is identical, with the exception that the DAS uses the `s1as` alias for SSL/TLS authentication and the instances use the `glassfish-instance` alias. (These alias names are the default, and you can change them.)

A server restart is required to change the Grizzly adapter behavior.

The restart also synchronizes the restarted instances. When you start the instances, the DAS delivers the updated configuration to the instances.

Which Administration Account is Used?

If only one administration account exists in the realm, Eclipse GlassFish treats that account as the current default administration account. In this case, when you run an `asadmin` command, you do not need to specify the username. If a password for that username is required, you need to specify it, typically by using the `--passwordfile` option or by letting `asadmin` prompt you for it.

By default, Eclipse GlassFish includes a single account for user "admin" and an empty password. Therefore, if you make no other changes before you enable secure admin, "admin" is the initial default username and no password is required. You need to decide whether enabling secure admin without also requiring a password makes sense in your environment.

If multiple admin accounts exist, then Eclipse GlassFish does not recognize any admin username as the default. You must then specify a valid username via the `--user` option when you use the `asadmin` command (or by or defining the `AS_ASDMIN_USER` environment variable), and its associated password (if the associated password is not empty).

The username and password used for a login attempt must match the username and password (if required) for an account defined in the realm, and you must have set up the account as a member of the admin group.

What Authentication Methods Are Used for Secure Administration?

The secure admin feature enforces security via the following authentication methods:

- The DAS and instances authenticate to each other via mutual (two-way) SSL/TLS certificate authentication. The DAS authenticates to clients via one-way SSL/TLS certificate authentication.

  The domain creation process creates a default keystore and truststore, plus a default private key for the DAS. Secure admin uses this initial configuration to set up the truststore so that the DAS and instances always trust each other.

- Remote administration clients (`asadmin`, administration console, browsers, and IDEs) must accept the public certificate presented by the DAS. If accepted, remote administration clients then send a user name and password (HTTP Basic authentication) in the HTTP Authorization header. The receiving DAS or instance makes sure those credentials are valid in its realm, and authenticates and authorizes the user.
- A locally-running `asadmin` (that is, connecting to an instance on the same host) authenticates and authorizes to the co-located instance using a locally-provisioned password.

- Credentials or other sensitive information sent over the network are always encrypted if secure admin is enabled. No credentials are sent in the clear if secure admin is enabled. (If secure admin is disabled, credentials are sent in the clear.) Messages between administration clients and the DAS, between the DAS and remote instances, and between local administration clients and instances are encrypted using SSL/TLS. This is true even if you explicitly set the `asadmin --secure` option to false.

Table 5-1 shows which authentication methods are employed when secure admin is enabled or disabled.

Table 5-1 Authentication Methods Employed

Access Method	When Secure Admin is Disabled	When Secure Admin is Enabled
Remote administration access to the DAS	Rejected.	Username/password authentication. (Client must also accept server certificate.)
Communication between DAS and instances	Cleartext messages. No mutual authentication.	SSL-encrypted messages. SSL mutual authentication using certificates.
Communication between administration clients and DAS	Cleartext messages. No DAS authentication.	SSL-encrypted messages. DAS uses SSL certificate server authentication.
Local asadmin client to instance on same node	Cleartext messages. Locally-provisioned password mechanism is used.	SSL-encrypted messages. Locally-provisioned password mechanism is used.

Understanding How Certificate Authentication is Performed

The domain creation process creates a primary (private) key and a self-signed certificate for the DAS, and a separate private key and self-signed certificate for remote instances.

Then, when you enable secure admin, the following actions are performed:

- Both private keys are stored in the domain-wide DAS keystore file, keystore.jks.
- Both public certificates are stored in the domain-wide DAS truststore file, cacerts.jks.

When the DAS sends a message to an instance:

1. SSL on the instance asks the DAS to provide an SSL/TLS certificate.
2. The DAS sends the certificate with the alias you specified using the `--adminalias` option when you ran the `enable-secure-admin` subcommand.
3. SSL on the instance makes sure the certificate is valid and Eclipse GlassFish makes sure that the security Principal associated with the incoming request (provided automatically by Grizzly and the SSL/TLS Java implementation) matches the Principal associated with the adminalias from the instance's truststore.

What Certificates Are Used?

When you enable secure admin, you can optionally set the `--adminalias` and `--instancealias` options that tell secure admin which aliases to use for the DAS and instance certificates.

The DAS uses the alias associated with the `--instancealias` option to check incoming requests that use SSL/TLS cert authentication. Conversely, instances use the alias associated with the `--adminalias` option to check incoming requests with certificate authentication.

By default, `--adminalias` of the `enable-secure-admin` subcommand uses the `s1as` alias, and the `--instancealias` option uses the `glassfish-instance` alias, both of which identify the default self-signed certificates.

You can use your tool of choice, such as keytool, to list the default self-signed certificates in the keystore, similar to the following:

You can list the contents of the keystore without supplying a password. However, for a request that affects the private key, such as the keytool.exe `--certreq` option, the keystore password is required. This is the master password and has a default value of changeit unless you change it with the `change-master-password` subcommand.

```
keytool.exe -list -keystore keystore.jks
Enter keystore password:

*****************  WARNING WARNING WARNING  *****************
* The integrity of the information stored in your keystore  *
* has NOT been verified!  In order to verify its integrity, *
* you must provide your keystore password.                  *
*****************  WARNING WARNING WARNING  *****************

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 2 entries

glassfish-instance, Jan 3, 2011, PrivateKeyEntry,
Certificate fingerprint (MD5): 06:A4:83:84:57:52:9C:2F:E1:FD:08:68:BB:2D:ED:E8
s1as, Jan 3, 2011, PrivateKeyEntry,
Certificate fingerprint (MD5): 8B:7D:5A:4A:32:36:1B:5D:6A:29:66:01:B0:A3:CB:85
```

The `--adminalias` and `--instancealias` values are maintained. Because of this design, normal instance creation operations (create-instance over SSH and create-local-instance) apply the up-to-date keystore, truststore, and configuration to each instance.

Self-Signed Certificates and Trust

The self-signed certificates that Eclipse GlassFish uses might not be trusted by clients by default because a certificate authority does not vouch for the authenticity of the certificate. If you enable

secure admin and then contact the DAS using an administration client, that client will detect whether the certificate is automatically trusted.

Browsers will warn you, let you view the certificate, and ask you to reject the certificate, accept it once, or accept it indefinitely, as shown in Figure 5-1.

Figure 5-1 Sample Browser Response to Untrusted Certificate

Similarly, the first time `asadmin` receives an untrusted certificate, it displays the certificate and lets you accept it or reject it, as follows: (If you accept it, `asadmin` also accepts that certificate in the future.)

```
D:\glassfish7\glassfish\bin>asadmin enable-secure-admin
Command enable-secure-admin executed successfully.


D:\glassfish7\glassfish\bin>asadmin stop-domain domain1
Waiting for the domain to stop .......
Command stop-domain executed successfully.

D:\glassfish7\glassfish\bin>asadmin start-domain domain1
Waiting for domain1 to start ............................
Successfully started the domain : domain1
domain  Location: D:\glassfish7\glassfish\domains\domain1
Log File: D:\glassfish7\glassfish\domains\domain1\logs\server.log
Admin Port: 4848
Command start-domain executed successfully.

D:\glassfish7\glassfish\bin>asadmin list-domains
[
[
```

```
  Version: V3
  Subject: CN=machine.oracle.com, OU=GlassFish, O=Oracle Corporation, L=San
ta Clara, ST=California, C=US
  Signature Algorithm: SHA1withRSA, OID = 1.2.840.113549.1.1.5

  Key:  Sun RSA public key, 1024 bits
  modulus: 91604359507378444963235875637429733088161806229854910107270225245885674079656358328568800001548507219262910864311924824938195045822088563459253216383211006608196572047575238964156068334714995640712267224780564071023188627967974656245090519956376357288295037519504394674686082145398885236913866246525691704749
  public exponent: 65537
  Validity: [From: Tue Jan 04 14:30:08 EST 2011,
              To: Fri Jan 01 14:30:08 EST 2021]
  Issuer: CN=machine.oracle.com, OU=GlassFish, O=Oracle Corporation, L=Sant
a Clara, ST=California, C=US
  SerialNumber: [    4d237540]

Certificate Extensions: 1
[1]: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: AF 8B 90 1E 51 9A 80 1B   EB A4 D9 C6 01 8A A0 FD  ....Q...........
0010: DE EC 83 8A                                        ....
]
]

]
  Algorithm: [SHA1withRSA]
  Signature:
0000: 3F 2B 30 CE 97 0B 5E F3   72 0E 60 18 8D 3B 04 DC  ?+0...^.r.`..;..
0010: 26 E6 7A 6F D0 19 CC 26   1D 90 C0 DE 33 4E 53 FB  &.zo...&....3NS.
0020: DC E7 AE 78 9E BA EF 14   86 57 36 D4 3E 9B C9 FB  ...x.....W6.>...
0030: C0 B4 EF 72 27 D9 4F 79   1F 89 91 B8 96 26 33 64  ...r'.Oy.....&3d
0040: 9F 4B 04 4B 83 B9 BF 4D   54 B4 8F 75 17 1A 51 BD  .K.K...MT..u..Q.
0050: F3 69 94 CE 90 95 08 55   2C 07 D2 23 AC AE EC 6D  .i.....U,..#...m
0060: 84 B6 3D 00 FB FE 92 50   37 1A 2D 00 F1 21 5C E6  ..=....P7.-..!\.
0070: 1F 39 26 B2 5D C1 FD C8   B1 4F CC EE 26 84 B8 B5  .9&.]....O..&...

]
Do you trust the above certificate [y|N] -->
```

`asadmin` saves certificates you accept in the file `.asadmintruststore` in your log-in default directory. You do not generally need to work with the file directly, but if you delete or move the file, `asadmin` will prompt you again when it receives untrusted certificates.

Some `asadmin` commands such as run-script can contact an instance directly to retrieve information (but not to make configuration changes). The instances do not use the same certificate as the DAS, so in these cases `asadmin` then prompts you to accept or reject the instance certificate.

Using Your Own Certificates

By default, `--adminalias` of the `enable-secure-admin` subcommand uses the `s1as` alias, and the `--instancealias` option uses the `glassfish-instance` alias, both of which identify the default self-signed certificates.

You can instead have Eclipse GlassFish use your own certificates for this purpose by first adding your certificates to the keystore and truststore, and then running `enable-secure-admin` and specifying the aliases for your certificates.

It is also possible to use `s1as` and `glassfish-instance` as the alias names for your own certificates. A benefit of doing so is that you would not have to specify alias names with the `enable-secure-admin` subcommand.

In addition, your own certificate identified by the `s1as` alias would be used in all other cases within the domain where the `s1as` alias is used (by default), such as in the SSL configuration of the IIOP and http-listener-2 listeners, and as the `encryption.key.alias` and `signature.key.alias` used for provider configuration in the SOAP authentication layer for Message Security configuration.

You may find the wide-reaching effect of using the `s1as` alias with your own certificate to be either a useful feature or an unintended consequence. Therefore, you should understand the implications of using the `s1as` alias before doing so.

If you decide to use the `s1as` and `glassfish-instance` aliases with your own certificates, you will first need to disable secure admin (if enabled) and then change or delete the exiting `s1as` alias from both the `keystore.jks` keystore and `cacerts.jks` truststore for the DAS. You can use the `--changealias` or`--delete` option of `keytool` to accomplish this. Then, import your own certificates.

When you enable secure admin, the DAS and the instances then have copies of the same keystore and truststore

An Alternate Approach: Using Distinguished Names to Specify Certificates

By default, the DAS uses the alias associated with the `--instancealias` option to check incoming requests that use SSL/TLS cert authentication. Conversely, instances use the alias associated with the `--adminalias` option to check incoming requests with certificate authentication.

The `enable-secure-admin-principal` subcommand provides an alternate approach. `enable-secure-admin-principal` instructs Eclipse GlassFish to accept admin requests when accompanied by an SSL certificate with the specified distinguished name (DN).

Any certificate you specify with `enable-secure-admin-principal` must either be issued by a trusted certificate authority or, if it is self-signed, must already be in the Eclipse GlassFish truststore.

For example, assume that you write your own admin client that uses the REST interface. When your client establishes the connection, it can choose which certificate to use for its client cert. You would then specify the DN of this certificate to `enable-secure-admin-principal`.

You must specify either the DN or the `--alias` option of the `enable-secure-admin-principal`

subcommand.

If you specify the DN, Eclipse GlassFish records the value you specify as the DN. You specify the DN as a comma-separated list in quotes. For example, `"CN=system.amer.oracle.com,OU=GlassFish,O=Oracle Corporation,L=Santa Clara,ST=California,C=US"`.

The `enable-secure-admin-principal` subcommand accepts the string you enter and does not immediately validate it. However, secure admin must be able to match the DN you specify in order to use it.

If you have sufficient privileges to view the content of the keystore, you can use keytool to display the DN of a certificate:

```
keytool.exe -v -list -keystore keystore.jks
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 2 entries

Alias name: glassfish-instance
Creation date: Jul 7, 2011
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=systemname.amer.oracle.com-instance, OU=GlassFish,
O=Oracle Corporation, L=Santa Clara, ST=California, C=US
Issuer: CN=systemname.amer.oracle.com-instance, OU=GlassFish, O=Oracle Corporation,
 L=Santa Clara, ST=California, C=US
Serial number: 4e15d6e7
Valid from: Thu Jul 07 11:55:19 EDT 2011 until: Sun Jul 04 11:55:19 EDT 2021
Certificate fingerprints:
         MD5:  05:6E:01:D6:CE:9D:29:DA:55:D9:10:5E:BE:CC:55:05
         SHA1: 2A:6D:A2:52:A5:2B:ED:DE:CD:B4:76:4A:65:9D:B5:79:A6:EA:3C:10
         Signature algorithm name: SHA1withRSA
         Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 96 99 36 B6 CF 60 1E 8A   AE 25 75 4E C8 34 AA AB  ..6..`...%uN.4..
0010: E1 3B CF 03                                        .;..
]
]
```

If you use the "`--alias` aliasname" form, then Eclipse GlassFish looks in its truststore for a

certificate with the specified alias and uses the DN associated with that certificate. alias-name must be an alias associated with a certificate currently in the truststore. Therefore, you may find it most useful for self-signed certificates for which you know the alias.

If you have sufficient privileges to view the contents of the truststore, you can use keytool to display the alias of a certificate:

```
keytool.exe -v -list -keystore cacerts.jks
Enter keystore password:
:
:
Alias name: glassfish-instance
Creation date: Jul 7, 2011
Entry type: trustedCertEntry

Owner: CN=systemname.amer.oracle.com-instance, OU=GlassFish, O=Oracle Corporation,
L=Santa Clara, ST=California, C=US
Issuer: CN=systemname.amer.oracle.com-instance, OU=GlassFish, O=Oracle Corporation,
 L=Santa Clara, ST=California, C=US
Serial number: 4e15d6e7
Valid from: Thu Jul 07 11:55:19 EDT 2011 until: Sun Jul 04 11:55:19 EDT 2021
Certificate fingerprints:
         MD5:  05:6E:01:D6:CE:9D:29:DA:55:D9:10:5E:BE:CC:55:05
         SHA1: 2A:6D:A2:52:A5:2B:ED:DE:CD:B4:76:4A:65:9D:B5:79:A6:EA:3C:10
         Signature algorithm name: SHA1withRSA
         Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 96 99 36 B6 CF 60 1E 8A   AE 25 75 4E C8 34 AA AB  ..6..`...%uN.4..
0010: E1 3B CF 03                                        .;..
]
]
```

When you run `enable-secure-admin`, Eclipse GlassFish automatically records the DNs for the admin alias and the instance alias, whether you specify those values or use the defaults. You do not need to run `enable-secure-admin-principal` yourself for those certificates.

Other than these certificates, you must run `enable-secure-admin-principal` for any other DN that Eclipse GlassFish should authorize to send admin requests. This includes DNs corresponding to trusted certificates (those with a certificate chain to a trusted authority.)

You can run `enable-secure-admin-principal` multiple times so that Eclipse GlassFish accepts admin requests from a client sending a certificate with any of the DNs you specify.

The following example shows how to specify a DN for authorizing access in secure administration:

```
asadmin> enable-secure-admin-principal
"CN=system.amer.oracle.com,OU=GlassFish,
O=Oracle Corporation,L=Santa Clara,ST=California,C=US"

Command enable-secure-admin-principal executed successfully.
```

You can use the `disable-secure-admin-principal` subcommand to disable a specific certificate for authenticating and authorizing access in secure admin. You must specify either the DN or the `--alias` option of the `disable-secure-admin-principal` subcommand. To disable multiple certificates for authenticating and authorizing access in secure admin, run the `disable-secure-admin-principal` subcommand multiple times.

You can use the `list-secure-admin-principals` subcommand to list the certificates for which Eclipse GlassFish accepts admin requests from clients.

Guarding Against Unwanted Connections

Secure admin guards against unwanted connections in several ways:

- DAS-to-DAS, instance-to-instance:
  - The DAS and the instances have copies of the same truststore, which contains the public certificate of the DAS and the separate public certificate that is used by all instances. In addition, Eclipse GlassFish includes a unique, generated "domain ID" that servers use to ensure that admin requests from other Eclipse GlassFishs originate from the correct domain.
  - DAS-to-other-DAS communication is not authenticated because each different DAS will have its own self-signed certificate that is not in the truststore of the other DAS.
  - DAS-to-itself communication is unlikely unless you were to misconfigure the admin listener port for an instance on the same host so it is the same as for the DAS. Similarly, instance-to-instance traffic is unlikely unless you were to misconfigure listener ports for instances on the same host.

    To prevent both of these situations, both cases are handled by making sure that the connecting Principal (alias) is not the running Principal. secure admin ensures that if the client has authenticated using SSL/TLS client authentication that the Principal associated with the remote client is not the same as the current process. That is, the DAS makes sure that the Principal is not itself. Similarly, each instance ensures that the client is not an instance. (The instances share the same self-signed certificate and therefore are mapped to the same Principal.)
- Remote client-to-instance:

  Remote `asadmin` clients are unable to connect directly to instances. If the user on host "test1" runs a local command but specifies a remote instance on host "test2," `asadmin` on test1 will read and send that locally-provisioned password. The instance on "test2" will have a different locally-provisioned password and so the authentication attempt will fail.

Therefore, a user on "test1" will not be able to run a remote command targeting an instance on "test2."

Considerations When Running Eclipse GlassFish With Default Security

In Eclipse GlassFish, the default admin account is username "admin" with an empty password. Admin clients provide empty credentials or none at all, and all are authenticated and authorized as that default admin user. None of the participants (clients, DAS, or instances) encrypts network messages.

If this level of security is acceptable in your environment, no changes are needed and you do not need to enable secure administration. Imposing a heightened level of security is optional.

However, consider Table 5-2, which shows which operations are accepted and rejected when secure admin is disabled.

When secure admin is disabled, Eclipse GlassFish does allow remote monitoring (read-only) access via the REST interface.

Table 5-2 Accepted and Rejected Operations if Secure Admin is Disabled

Operation	Run From Same System as DAS	Run From Remote System
`start-local-instance`	Functions as expected	Cannot sync with DAS. The instance starts but cannot communicate with the DAS. DAS will not see the instance.
Any other `asadmin` subcommand	Functions as expected	Rejected. A user sees the username/password prompt, but even correct entries are rejected.
Commands that use SSH. For example, `create-instance`.	Functions as expected; requires prior SSH configuration.	Functions as expected; requires prior SSH configuration.

Running Secure Admin

This section describes how to run secure admin. The section begins with prerequisites for running secure admin.

Prerequisites for Running Secure Admin

Before running Eclipse GlassFish with secure admin enabled, you must make sure that:

1. The DAS is installed, initialized, and running.
2. If one or more remote instances are installed and initialized, they must not be running.

3. Any administration clients you require are installed.
4. The DAS communicates on the `--adminport` you configure when you create the domain, and defaults to 4848. An instance communicates on the `ASADMIN_LISTENER_PORT` system property you specify for the instance.
5. The user name and password sent by remote administration clients (`asadmin`, administration console, browsers, and IDEs) must exist in the realm and be in the admin group.
6. The keystore and truststore for the domain exist. (They are created by default when you create the domain or install Eclipse GlassFish.)

   If you are not using the default self-signed certificates, you must add your own valid certificates and CA root in the keystore and truststore, respectively.
7. If you are not using the default self-signed certificates, create two aliases corresponding to certificates in the keystore and truststore: one that the DAS will use for authenticating itself in administration traffic, and one that the instances will use for authenticating itself in administration traffic.

An Alternate Approach: Using A User Name and Password for Internal Authentication and Authorization

By default, secure admin uses the Eclipse GlassFish self-signed certificates, via the aliases corresponding to these certificates, to authenticate the DAS and instances with each other and to authorize secure admin operations. Specifically, the DAS uses the (`s1as`) alias for authenticating itself and authorizing access in administration traffic, and instances use the (`glassfish-instance`) alias for authenticating themselves and authorizing access in secure admin traffic.

As described in Using Your Own Certificates, you can instead use your own certificates and their associated aliases for authenticating and authorizing the DAS and instances in administration traffic.

As an alternative to this certificate-based authentication and authorization, you can instead use the `enable-secure-admin-internal-user` subcommand to instruct all servers in the domain to authenticate to each other, and to authorize admin operations submitted to each other, using an existing admin user name and password rather than SSL certificates.

If secure admin is enabled, all Eclipse GlassFish processes continue to use SSL encryption to secure the content of the admin messages, regardless of how they authenticate to each other.

You might want to use the `enable-secure-admin-internal-user` subcommand if your use case favors the use of a user name and password combination over the use of SSL certificates and aliases.

This generally means that you must:

1. Create a valid admin user.

```
asadmin> create-file-user --authrealmname admin-realm --groups
```

```
asadmin newAdminUsername
```

2. Create a password alias for the just-created password.

```
asadmin> create-password-alias passwordAliasName
```

3. Use that user name and password for inter-process authentication and admin authorization.

```
asadmin> enable-secure-admin-internal-user
--passwordalias passwordAliasName
newAdminUsername
```

The following example allows secure admin to use a user name and password alias for authentication and authorization between the DAS and instances, instead of certificates.

```
asadmin> enable-secure-admin-internal-user
--passwordalias passwordAliasName
newAdminUsername
```

If Eclipse GlassFish finds at least one secure admin internal user, then if secure admin is enabled Eclipse GlassFish processes will not use SSL authentication and authorization with each other and will instead use user name password pairs.

Most users who use this subcommand will need to set up only one secure admin internal user. If you set up more than one secure admin internal user, you should not make any assumptions about which user name and password pair Eclipse GlassFish will choose to use for any given admin request.

As a general practice, you should not use the same user name and password pair for internal admin communication and for admin user login. That is, create at least one admin account specifically for internal admin communication.

You can use the `disable-secure-admin-internal-user` subcommand to disable secure admin from using the user name (instead of SSL certificates) to authenticate the DAS and instances with each other and to authorize admin operations. To disable multiple user names for authenticating and authorizing access in secure admin, run the `disable-secure-admin-internal-user` subcommand multiple times.

You can use the `list-secure-admin-internal-users` subcommand to list the user names for which Eclipse GlassFish authenticate the DAS and instances with each other and authorizes admin operations.

Example of Running enable-secure-admin

The following example shows how to enable secure admin for a domain using the default admin alias and the default instance alias. You must restart the DAS immediately after enabling secure

admin.

The only indicator that secure admin is enabled is the successful status from the `enable-secure-admin` subcommand. When secure admin is running, the DAS and instances do not report the secure admin status.

```
asadmin> enable-secure-admin
Command enable-secure-admin executed successfully.
```

The following example shows how to enable secure admin for a domain using an admin alias adtest and an instance alias intest. You can also use this command to modify an existing secure admin configuration to use different aliases.

```
asadmin> enable-secure-admin --adminalias adtest --instancealias intest
```

The following example shows how to disable secure admin:

```
asadmin> disable-secure-admin
Command disable-secure-admin executed successfully.
```

You can use the following command to see the current state of secure admin in a domain:

```
asadmin> get secure-admin.enabled
secure-admin.enabled=false
Command get executed successfully.
```

Additional Considerations When Creating Local Instances

If you use `xxx-local-instance` commands to set up local instances, either leave secure admin disabled, or enable it before you create or start the instances and leave it that way.

However, if you use `xxx-instance` commands over SSH to manage remote instances, you can enable and disable secure admin, although this is not recommended because it can result in an inconsistent security model.

Secure Admin Use Case

This section describes a simple secure admin use case.

In the `asadmin --secure=false --user me --passwordfile myFile.txt cmd ⋯` use case, the user submits a command with `--secure` set to false, and supplies password credentials.

The important concept to note is that `asadmin` uses HTTPS because of the DAS redirection, even though the command sets `--secure` to false. `asadmin` sends the HTTP Authorization header along with the redirected request.

In addition to the flow described here, certificate authentication is also performed as described in Table 5-3. Also, the credentials that the user supplies are assumed to be valid administrator credentials for the DAS.

Table 5-3 asadmin --secure=false, With Username and Password

asadmin	**Grizzly**	**AdminAdapter**
Sends HTTP request, no authorization header (because the transport is not secure).		
	Returns 3xx status and redirects HTTP to HTTPS.	
Follows redirection, this time adding the Authorization header (because transport is now HTTPS).		
		Authenticates admin user and password from HTTP Authorization header in the realm Executes command, and responds with success status.

Upgrading an SSL-Enabled Secure GlassFish Installation to Secure Admin

If you enable secure admin on an SSL-enabled Eclipse GlassFish installation, secure admin uses the existing <ssl cert-nickname> value as the DAS adminalias for secure admin.

6 Running in a Secure Environment

This chapter describes important information about running Eclipse GlassFish in a secure environment.

This chapter assumes that you are familiar with security features such as authentication, authorization, and certificates. If you are not, see Administering System Security.

Instructions for accomplishing the tasks specific to Eclipse GlassFish by using the Administration Console are contained in the Administration Console online help.

The chapter describes the following topics:

- Determining Your Security Needs
- Installing Eclipse GlassFish in a Secure Environment
- Run on the Web Profile if Possible
- Securing the Eclipse GlassFish Host
- Securing Eclipse GlassFish
- Securing Applications

Determining Your Security Needs

Before you deploy Eclipse GlassFish and your Jakarta EE applications into a production environment, determine your security needs and make sure that you take the appropriate security measures, as described in the following sections:

- Understand Your Environment
- Read Security Publications

Understand Your Environment

To better understand your security needs, ask yourself the following questions:

- Which resources am I protecting?

  Many resources in the production environment can be protected, including information in databases accessed by Eclipse GlassFish and the availability, performance, applications, and the integrity of the Web site. Consider the resources you want to protect when deciding the level of security you must provide.

- From whom am I protecting the resources?

  For most Web sites, resources must be protected from everyone on the Internet. But should the Web site be protected from the employees on the intranet in your enterprise? Should your employees have access to all resources within the Eclipse GlassFish environment? Should the system administrators have access to all Eclipse GlassFish resources? Should the system administrators be able to access all data? You might consider giving access to highly confidential

data or strategic resources to only a few well trusted system administrators. Perhaps it would be best to allow no system administrators access to the data or resources.

- What will happen if the protections on strategic resources fail?

  In some cases, a fault in your security scheme is easily detected and considered nothing more than an inconvenience. In other cases, a fault might cause great damage to companies or individual clients that use the Web site. Understanding the security ramifications of each resource will help you protect it properly.

See Also: `http://www.oracle.com/us/products/ondemand/index.html`

Read Security Publications

Read about security issues:

- For the latest information about securing Web servers, Oracle recommends the "Security Practices & Evaluations" information available from the CERT Coordination Center operated by Carnegie Mellon University at `http://www.cert.org/`.

Installing Eclipse GlassFish in a Secure Environment

This section describes recommendations for installing Eclipse GlassFish in a secure environment. The Enable the Secure Administration Feature topic is described.

Enable the Secure Administration Feature

The secure administration feature allows an administrator to secure all administrative communication between the domain administration server (DAS), any remote instances, and administration clients such as the asadmin utility, the administration console, and REST clients. In addition, secure administration helps to prevent DAS-to-DAS and instance-to-instance traffic, and carefully restricts administration-client-to-instance traffic.

When you install Eclipse GlassFish or create a new domain, secure admin is disabled by default. Eclipse GlassFish does not encrypt administrative communication among the system components and does not accept administrative connections from remote hosts. Imposing a heightened level of security is optional.

See Managing Administrative Security for information on enabling the secure administration feature.

Run on the Web Profile if Possible

If your applications can run on the Web Profile, use that instead of the Full Platform.

Starting in Jakarta EE 6, Jakarta EE introduced the concept of profiles. A profile is a collection of Jakarta EE technologies and APIs that address specific developer communities and application types.

The following profiles are implemented through the distributions of Eclipse GlassFish:

- Full Platform -The full Jakarta EE platform is designed for developers who require the full set of Jakarta EE APIs for enterprise application development, and is installed when you install Eclipse GlassFish.
- Web Profile -This profile contains Web technologies that are a subset of the full Java platform, and is designed for developers who do not require the full set of Jakarta EE APIs.

For the list of APIs in each profile, see " Jakarta EE Standards Support" in Eclipse GlassFish Release Notes.

Securing the Eclipse GlassFish Host

A Eclipse GlassFish production environment is only as secure as the security of the machine on which it is running. It is important that you secure the physical machine, the operating system, and all other software that is installed on the host machine.

The following are recommendations for securing a Eclipse GlassFish host in a production environment. Also check with the manufacturer of the machine and operating system for recommended security measures.

The domain and server configuration files should be accessible only by the operating system users who configure or execute Eclipse GlassFish.

Table 6-1 Securing the Eclipse GlassFish Host

Security Action	Description
Physically secure the hardware.	Keep your hardware in a secured area to prevent unauthorized operating system users from tampering with the deployment machine or its network connections.
Log out of the Administration Console before navigating to a non-secure site.	If you are logged on to the Administration Console, be sure to log out completely before browsing to an unknown or non-secure Web site.
Secure networking services that the operating system provides.	Have an expert review network services such as e-mail programs or directory services to ensure that a malicious attacker cannot access the operating system or system-level commands. The way you do this depends on the operating system you use. Sharing a file system with other machines in the enterprise network imposes risks of a remote attack on the file system. Be certain that the remote machines and the network are secure before sharing the file systems from the machine.
Use a file system that can prevent unauthorized access.	Make sure that the file system on each Eclipse GlassFishhost can prevent unauthorized access to protected resources. For example, on a Windows computer, use only NTFS.

Security Action	Description
Set file access permissions for data stored on disk.	Set operating system file access permissions to restrict access to data stored on disk. This data includes, but is not limited to, the following: The database files. Eclipse GlassFish includes Apache Derby database, however, you can use any JDBC-compliant database. The directory and filename location of a private keystore, such as keystore.jks The directory and filename location of a Root Certificate Authority (CA) keystore, such as cacerts.jks. For example, operating systems provide utilities such as umask and chmod to set the file access permissions. At a minimum, consider using "umask 066", which denies read and write permission to Group and Others.
Set file access permission for the Eclipse GlassFish installation.	The directory structure in which Eclipse GlassFish is located, including all files, should be protected from access by unprivileged users. Taking this step helps ensure that unprivileged users cannot insert code that can potentially be executed by Eclipse GlassFish.
Limit the number of user accounts on the host machine.	Avoid creating more user accounts than you need on host machines, and limit the file access privileges granted to each account. On operating systems that allow more than one system administrator user, the host machine should have two user accounts with system administrator privileges and one user with sufficient privileges to run Eclipse GlassFish. Having two system administrator users provides a back up at all times. The Eclipse GlassFish user should be a restricted user, not a system administrator user. One of the system administrator users can always create a new Eclipse GlassFish user if needed. Important: Domain and server configuration files should be accessible only by the operating system users who configure or execute Eclipse GlassFish. Review active user accounts regularly and when personnel leave. Background Information: Configuration data and some URL (Web) resources, including Java Server Pages (JSPs) and HTML pages, are stored in clear text on the file system. A sophisticated user or intruder with read access to files and directories might be able to defeat any security mechanisms you establish with authentication and authorization schemes.
For your system administrator user accounts, choose names that are not obvious.	For additional security, avoid choosing an obvious name such as "system," "admin," or "administrator" for your system administrator user accounts.

Security Action	Description
Safeguard passwords.	The passwords for user accounts on production machines should be difficult to guess and should be guarded carefully. Set a policy to expire passwords periodically. Never code passwords in client applications. Do not deploy an application that can be accessed with the default username admin and no password.
Safeguard password files	The `-passwordfile` option of the `asadmin` command specifies the name of a file that contains password entries in a specific format. These password entries are stored in clear text in the password file, and rely on file system mechanisms for protection. Therefore, any password file created for use with the `-passwordfile` option should be protected with file system permissions. Additionally, any password file being used for a transient purpose, such as setting up SSH among nodes, should be deleted after it has served its purpose. To provide additional security, create a password alias.
Use a password alias	A password alias stores a password in encrypted form in the domain keystore, providing a clear-text alias name to use instead of the password. To provide additional security, use the `create-password-alias` subcommand to create an alias for the password. The password for which the alias is created is stored in an encrypted form. Then, specify the alias in the entry for the password in the password file as follows: In password files and the domain configuration file, use the form $\{alias=alias-name} to refer to the encrypted password.
Do not run Eclipse GlassFish as root	Eclipse GlassFish should run only as an unprivileged user, never as root. Taking this step helps ensure that code from other users cannot be executed by Eclipse GlassFish.
Consider use PAM Realm	The use of a PAM Realm requires Eclipse GlassFish to run as an account that has read-access to a shadow password file or the equivalent, and therefore may not be suitable in your environment.
Do not develop on a production machine.	Develop first on a development machine and then move code to the production machine when it is completed and tested. This process prevents bugs in the development environment from affecting the security of the production environment.

Security Action	Description
Do not install development or sample software on a production machine.	Do not install development tools on production machines. Keeping development tools off the production machine reduces the leverage intruders have should they get partial access to a production machine.
Enable security auditing.	If the operating system on which Eclipse GlassFish runs supports security auditing of file and directory access, Oracle recommends using audit logging to track any denied directory or file access violations. Administrators should ensure that sufficient disk space is available for the audit log.
Consider using additional software to secure your operating system.	Most operating systems can run additional software to secure a production environment. For example, an Intrusion Detection System (IDS) can detect attempts to modify the production environment. Refer to the vendor of your operating system for information about available software.
Apply operating system patch sets and security patches.	Refer to the vendor of your operating system for a list of recommended patch sets and security-related patches.
Apply the latest maintenance packs and critical patch updates.	Refer to the vendor of your operating system for a list of maintenance packs and critical patch updates.

Securing Eclipse GlassFish

Eclipse GlassFish provides a powerful and flexible set of software tools for securing the subsystems and applications that run on a server instance. The following table provides a checklist of essential features that Oracle recommends you use to secure your production environment.

Table 6-2 Securing Eclipse GlassFish

Security Action	Description
Enable Secure Admin.	The secure administration feature allows an administrator to secure all administrative communication between the domain administration server (DAS), any remote instances, and administration clients such as the `asadmin` utility, the administration console, and REST clients. In addition, secure administration helps to prevent DAS-to-DAS and instance-to-instance traffic, and carefully restricts administration-client-to-instance traffic. The secure administration feature provides a secure environment, in which you can be confident that rogue users or processes cannot intercept or corrupt administration traffic or impersonate legitimate Eclipse GlassFish components. See Managing Administrative Security.
Protect the `.asadminpass` file	If you create a domain with the `--savelogin` option, `create-domain` saves the administration user name and password in the `.asadminpass` file in the user's home directory. Make sure that this file remains protected. Information stored in this file will be used by `asadmin` commands to manage this domain.
Safeguard password files	The `-passwordfile` option of the `asadmin` command specifies the name of a file that contains password entries in a specific format. These password entries are stored in clear text in the password file, and rely on file system mechanisms for protection. Therefore, any password file created for use with the `-passwordfile` option should be protected with file system permissions. Additionally, any password file being used for a transient purpose, such as setting up SSH among nodes, should be deleted after it has served its purpose. To provide additional security, create a password alias.
Deploy production-ready security providers to the security realm.	Java Authorization Contract for Containers (JACC) is the part of the Jakarta EE specification that defines an interface for pluggable authorization providers. This enables you to set up third-party plug-in modules to perform authorization. By default, the Eclipse GlassFish provides a simple, file-based authorization engine that complies with the JACC specification. You can also specify additional third-party JACC providers. If you have purchased or written your own security providers, make sure that you have deployed and configured them properly.

Security Action	Description
Use SSL, but do not use the self-signed certificates in a production environment.	To prevent sensitive data from being compromised, secure data transfers by using HTTPS. By default, Eclipse GlassFish uses self-signed certificates. The self-signed certificates that Eclipse GlassFish uses might not be trusted by clients by default because a certificate authority does not vouch for the authenticity of the certificate. You can instead use your own certificates, as described in Using Your Own Certificates.
Restrict the size and the time limit of requests on external channels to prevent Denial of Service attacks.	To prevent some Denial of Service (DoS) attacks, restrict the size of a message as well as the maximum time it takes a message to arrive. The default setting for maximum post size is 2097152 bytes and 900 seconds for the request timeout.
Enable authentication and authorization auditing.	Auditing is the process of recording key security events in your Eclipse GlassFish environment. You use audit modules to develop an audit trail of all authentication and authorization decisions. To enable audit logging, two steps are required: 1. On the Security page, select the Audit Logging Enabled checkbox to enable audit logging. 2. Set the `auditOn` property for the active audit module to true. Review the auditing records periodically to detect security breaches and attempted breaches. Noting repeated failed logon attempts or a surprising pattern of security events can prevent serious problems.
Set logging for security and SSL messages.	Consider setting module log levels for table.jakarta.enterprise.system.ssl.security and jakarta.enterprise.system.core.security. You can set a level from Severe to Finest (the default is Info), but be aware that the finer logging levels may produce a large log file. By default, Eclipse GlassFish logging messages are recorded in the server log, and you can set the file rotation limit, as described in `rotate-log`(1)
Ensure that you have correctly assigned users to the correct groups.	Make sure you have assigned the desired set of users to the right groups. In particular, make sure that users assigned to the asadmin group need to be members of that group.
Create no fewer than two user accounts in the asadmin group.	The user admin is created when you install Eclipse GlassFish. For production environments, create at least one other account in the asadmin group in case one account password is compromised. When creating asadmin users give them unique names that cannot be easily guessed.

Security Action	Description
Assign a password to the admin account.	By default, Eclipse GlassFish includes a single account for user "admin" and an empty password. For production environments this default is inherently unsecure, and you should set a password for admin.
Run on the latest JDK update.	Refer to the vendor of your JDK for a list of security-related updates.
Disabling any public facing listeners if they are not used in any way.	By default, Eclipse GlassFish opens several listener ports upon startup, but you can restrict this by disabling unused listeners. This step helps prevent access from malicious attackers.

Securing Applications

Although much of the responsibility for securing the Eclipse GlassFish resources in a domain fall within the scope of the server, some security responsibilities lie within the scope of individual applications. For some security options, Eclipse GlassFish enables you to determine whether the server or individual applications are responsible. For each application that you deploy in a production environment, review the items in the following table to verify that you have secured its resources.

Table 6-3 Securing Applications

Security Action	Description
Use JSP comment tags instead of HTML comment tags.	Comments in JSP files that might contain sensitive data and or other comments that are not intended for the end user should use the JSP syntax of <%/* xxx */%> instead of the HTML syntax <!-- xxx -→. The JSP comments, unlike the HTML comments, are deleted when the JSP is compiled and therefore cannot be viewed in the browser.
Do not install uncompiled JSPs and other source code on the production machine.	Always keep source code off of the production machine. Getting access to your source code allows an intruder to find security holes. Consider precompiling JSPs and installing only the compiled JSPs on the production machine. To do this, set the `deploy` subcommand `-precompilejsp` option to true for the component. When set to true, the `deploy` and `redeploy` subcommands `-precompilejsp` option compiles JSPs during deploy time. If set to false (the default), JSPs are compiled during runtime.
Configure your applications to use SSL.	Set the transport-guarantee to CONFIDENTIAL in the user-data-constraint element of the web.xml file whenever appropriate.

Security Action	Description
Examine applications for security.	There are instances where an application can lead to a security vulnerability. Of particular concern is code that uses Java native interface (JNI) because Java positions native code outside of the scope of Java security. If Java native code behaves errantly, it is only constrained by the operating system. That is, the Java native code can do anything Eclipse GlassFish itself can do. This potential vulnerability is further complicated by the fact that buffer overflow errors are common in native code and can be used to run arbitrary code.
If your applications contain untrusted code, enable the Java security manager.	The Java security manager defines and enforces permissions for classes that run within a JVM. In many cases, where the threat model does not include malicious code being run in the JVM, the Java security manager is unnecessary. However, when third parties use Eclipse GlassFish and untrusted classes are being run, the Java security manager may be useful. See "Enabling and Disabling the Security Manager" in Eclipse GlassFish Application Development Guide.
Replace HTML special characters when servlets or JSPs return user-supplied data.	The ability to return user-supplied data can present a security vulnerability called cross-site scripting, which can be exploited to steal a user's security authorization. For a detailed description of cross-site scripting, refer to "Understanding Malicious Content Mitigation for Web Developers" (a CERT security advisory) at `http://www.cert.org/tech_tips/malicious_code_mitigation.html`. To remove the security vulnerability, before you return data that a user has supplied, scan the data for HTML special characters. If you find any such characters, replace them with their HTML entity or character reference. Replacing the characters prevents the browser from executing the user-supplied data as HTML.