IBM WebSphere Application Server, Version 5

Security

Note

Before using this information, be sure to read the general information under “Trademarks and service marks” on page v.

Compilation date: November 21, 2002

Contents

Trademarks and service marks v

Chapter 1. Welcome to Security 1

Chapter 2. Securing applications and their environment 7

Planning to secure your environment 8
- Security considerations when adding a Base Application Server node to Network Deployment 17

Implementing security considerations during installation 18
- Securing your environment before installation . . 18
- Securing your environment after installation . . 19

Migrating security configurations from previous releases 20
- Migrating custom user registries 21
- Migrating CORBA programmatic login to Java Authentication and Authorization Service . . . 25
- Migrating from CustomLoginServlet to servlet filters 28

Developing secured applications 29
- Developing with programmatic security APIs for Web applications 30
- Developing form login pages 37
- Developing with programmatic APIs for EJB applications 40
- Developing with JAAS to login programmatically 44
- Developing your own J2C principal mapping module 50
- Developing custom user registries 52
- Developing a custom interceptor for trust associations 61

Assembling secured applications 64
- EJB component security 65
- Securing EJB applications. 65
- Web component security 70
- Securing Web applications 70
- Role-based authorization 73
- Adding users and groups to roles 77
- Mapping users to RunAs roles 78

Deploying secured applications. 79
- Assigning users and groups to roles 80

Testing security 92

Managing secured applications 93
- Global security 93
- Configuring global security 94
- Administrative console and naming service authorization 100
- Assigning users to administrator roles 103
- Assigning users to naming roles 105
- Authentication mechanisms 107
- Configuring authentication mechanisms . . . 109
- User registries 126
- Configuring user registries 127
- Java Authentication and Authorization Service 170
- Configuring Java Authentication and Authorization Service login configuration . . . 173
- Authentication protocol for EJB security . . . 187
- Configuring CSIv2 and SAS authentication protocols 195
- Secure Sockets Layer 217
- Configuring Secure Sockets Layer 223
- Cryptographic token support 258
- Opening a cryptographic token using the IKeyman tool 259
- Configuring to use cryptographic tokens . . . 260
- Using JSSE and JCE with Servlets and EJB components 262
- Java 2 Security 267
- Configuring Java 2 Security 274

Troubleshooting security configurations 302

Tuning security configurations. 303
- Tuning CSIv2 303
- Tuning LDAP Authentication 304
- Tuning Web Authentication. 304
- Tuning Authorization. 304

Security: Resources for learning 305
- Planning, business scenarios and IT architecture 305
- Programming model and decisions 306
- Programming specifications 306
- Administration 307

Chapter 3. Integrating IBM WebSphere Application Server security with existing security systems 309

Interoperability issues for security 312
- Interoperability with C++ CORBA client support and limitations 312

Interoperating with C++ CORBA Client 313

Interoperating with previous product versions . . 314

Interoperating with WebSphere Application Server z/OS (Security Authentication Service) 315
- Creating login key files 316
- Preparing truststore files. 317
- Configuring the application server for interoperability 317

Security: Resources for learning 318
- Planning, business scenarios and IT architecture 318
- Programming model and decisions 318
- Programming specifications 319
- Administration 319

Trademarks and service marks

The following terms are trademarks of IBM Corporation in the United States, other countries, or both:

- Everyplace
- iSeries
- IBM
- Redbooks
- ViaVoice
- WebSphere
- zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

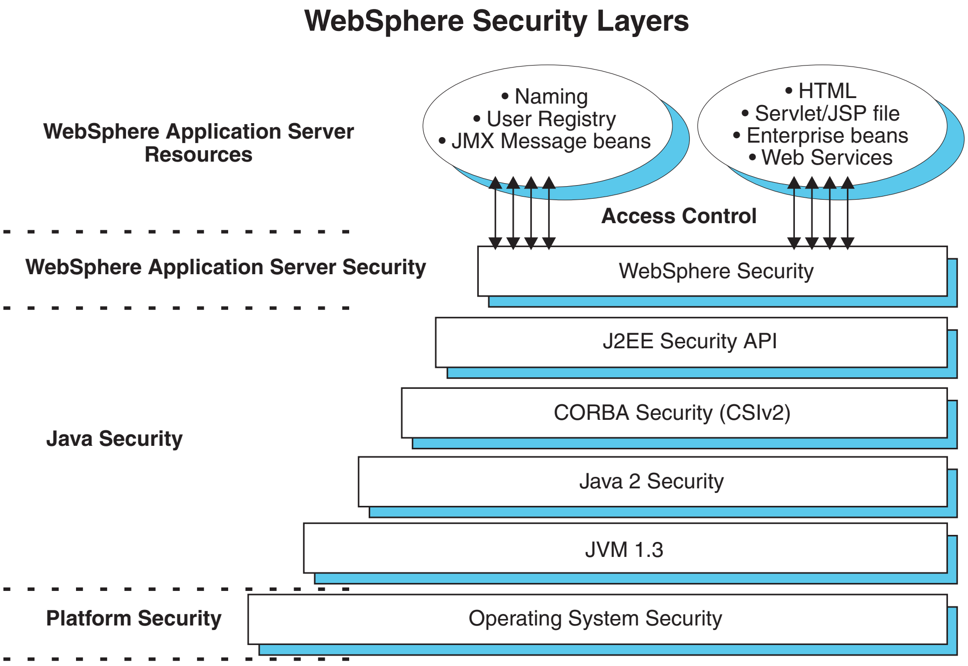
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product and service names may be trademarks or service marks of others.

Chapter 1. Welcome to Security

IBM WebSphere Application Server Version 5 provides security infrastructure and mechanisms to protect sensitive J2EE resources and administrative resources and to address enterprise end-to-end security requirements on authentication, on resource access control, on data integrity, confidentiality, and privacy, and on secure interoperability. IBM WebSphere Application Server security is based on industry standards. Version 5 has an open architecture that allows secure connectivity and interoperability with Enterprise Information System including DB2, CICS, MQ Series, Domino, IBM Directory and products from other vendors, with security providers including Tivoli Access Manager (Policy Director) and WebSeal secure proxy server, and with security products from other vendors.



Based on industry standards

The product provides a unified, policy-based, and permission-based model for securing Web resources and Enterprise Java according to J2EE specifications. Specifically Version 5 complies with J2EE specification Version 1.3.1 and has passed the J2EE Compatibility Test Suite. Product security is a layered architecture built on top of OS platform, JVM, and Java 2 security and employs a rich set of security technology including the:

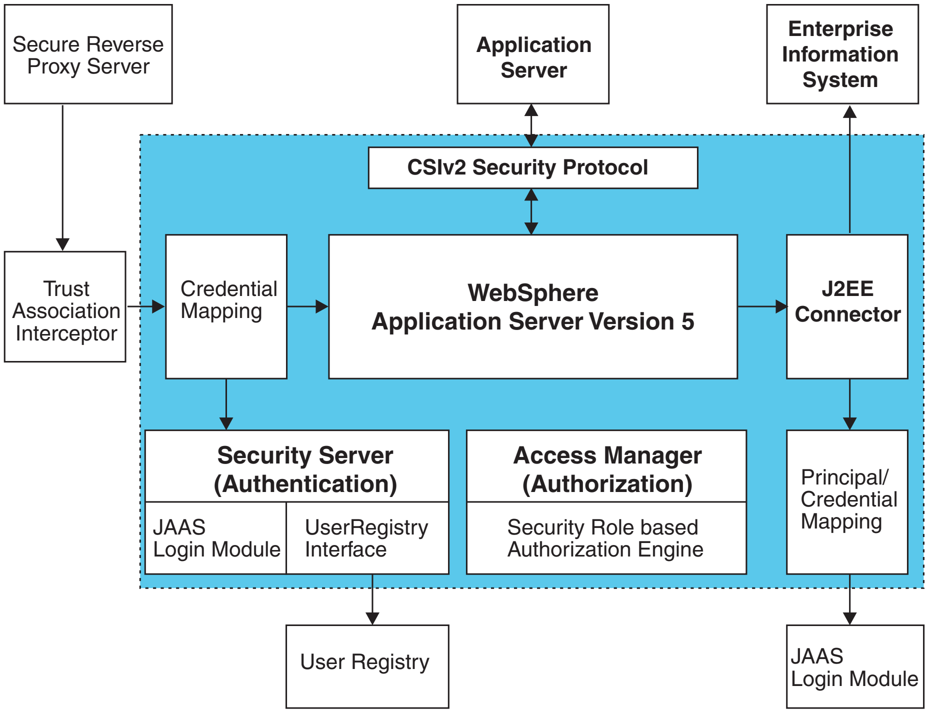
- Java 2 Security model, which provides policy-based, fine-grained, and permission-based access control to system resources.
- CSIv2 security protocol, in addition to the SAS (Secure Association Services) security protocol. Both protocols are supported by prior product releases. CSIv2 is an integral part of J2EE 1.3 Specification and is essential for interoperability among application servers from different vendors and with enterprise CORBA services.

- Java Authentication and Authorization Service (JAAS) programming model for Java applications, Servlets, and EJBs.
- J2EE Connector architecture for plugging in resource adapters that supports access to Enterprise Information Systems.

The standard security model and interface supported include Java Secure Socket Extension (JSSE) and Java Cryptographic Extension (JCE) provider for secure socket communication and for message and data encryption.

Open architecture paradigm

Application server plays an integral part of the multiple-tier enterprise computing framework. IBM WebSphere Application Server adopts the open architecture paradigm and provides many plug-in points to integrate with enterprise software components. Plug-in points are based on standard J2EE specifications wherever applicable.

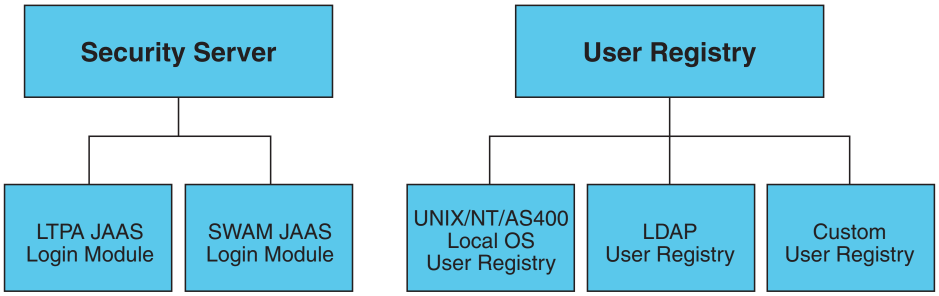

Secure Reverse Proxy Server
Application Server
Enterprise Information System
CSIv2 Security Protocol
Trust Association Interceptor
Credential Mapping
WebSphere Application Server Version 5
J2EE Connector
Security Server (Authentication)
JAAS Login Module
UserRegistry Interface
Access Manager (Authorization)
Security Role based Authorization Engine
Principal/ Credential Mapping
User Registry
JAAS Login Module

The dotted lines indicate the boundary between the product and other business application components.

The product provides SWAM (Simple WebSphere Authentication Mechanism) and LTPA (Light Weight Third Party) authentication mechanisms. Exactly one may be configured to be the active authentication mechanism for the security domain of the product. Exactly one user registry implementation may be configured to be the active user registry of the product security domain. The product provides the following UserRegistry implementations: Unix, NT, and AS400 LocalOSUserRegistry and LDAP UserRegistry. It also provides file-based and JDBC based UserRegistry reference implementations. It supports a flexible combination of authentication mechanisms and user registries. SWAM is simple to configure and is useful for a single application server environment. LTPA generates a security token for authenticated users which can be propagated to down stream servers and is suitable for distributed environment with multiple application servers. It is possible to use SWAM in a distributed environment if identity assertion is enabled. Note that identity assertion feature is available only on the CSIv2 security protocol.

The LTPA authentication mechanism is designed for distributed security. The security token can be validated by downstream servers. It also supports setting up trust association relationship with reverse secure proxy servers and Single SignOn (SSO), which will be discussed later. Besides the combination of LTPA and LDAP or Custom UserRegistry, Version 5 supports LTPA with LocalOSUserRegistry. The new configuration is particularly useful for a single node with multiple application servers. It may be used in a distributed environment if the local OS user registry implementation is a centralized user registry (such as NT Domain Controller) or can be kept in consistent state on multiple nodes.

Authentication Mechanism and User Registry



The product supports the J2EE Connector architecture and offers container managed authentication. It provides a default J2C principal/credential mapping module that basically maps any authenticated user credential to a PasswordCredential for the specified EIS security domain. The mapping module is a special JAAS LoginModule designed according to the Java 2 Connector and JAAS Specifications. Other mapping LoginModules can be plugged in.

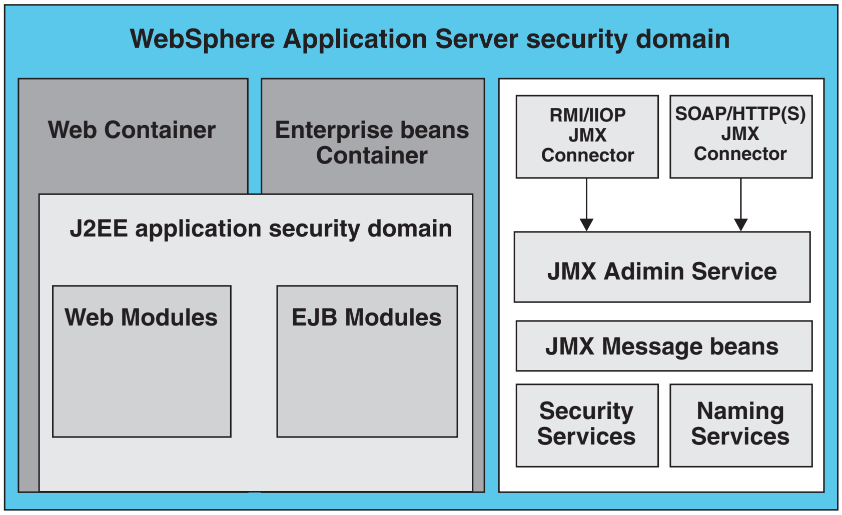
Backward compatibility

While adding new security functions and moving towards new industry standards, this version maintains backward compatibility with the 4.0.x and 3.5.x releases. Applications created in the Version 4.x development environment can be deployed in Version 5. When Java 2 Security is enforced in Version 5, special consideration needs to be given to Version 4.0.x applications because Version 4.0 applications may not be Java 2 Security ready. Please refer to the security migration section for steps to port Version 4.0.x to Version 5. See also the Security section of ″What is new in Version 5″ (not in this document).

Security for J2EE resources is provided by Web container and EJB container. Each container provides two kind of security: declarative security and programmatic security. In declarative security, the security structure of an application, including data integrity and confidentiality, authentication requirements, security roles, and access control, is expressed in a form external to the application. In particular the deployment descriptor is the primary vehicle for declarative security in the J2EE platform. The product maintains J2EE security policy including information derived from the deployment descriptor and specified by deployers and administrators in a set of XML descriptor files. At runtime, the container uses the security policy defined in the XML descriptor files to enforce data constraints and access control. When declarative security alone is not sufficient to express the security model of an application, programmatic security may be used by application code to make access decisions. The API for programmatic security consists of two methods of the EJB EJBContext interface (isCallerInRole, getCallerPrincipal) and two methods of the servlet HttpServletrequest interface (isUserInRole, getUserPrincipal).

Every application server process consists of a Web container, an EJB container, the administrative subsystem. Of course this is viewed only from the perspective of security. There are many other components that constitute a server process which will not be discussed here. The security services consists of authentication mechanism, user registry, and access control manager. Remote interface to the administrative subsystem, including the Admin Service interface via JMX connectors, user registry interface, and naming interface are protected by extended security role based access control. The product supports the Java 2 Security model. All the system code in the yellow box, including the administrative subsystem, the Web container, and the EJB container code, are running in the product security domain which in the present implementation are granted with AllPermission and can access all system resources. Application code are running in the application security domain which by default are granted with permissions according to J2EE specifications and can only access a restricted set of system resources. The product runtime classes are protected by the product classloader and are kept invisible to application code.

WebSphere Application Server Process


WebSphere Application Server security domain
Web Container
Enterprise beans Container
J2EE application security domain
Web Modules
EJB Modules
RMI/IIOP JMX Connector
SOAP/HTTP(S) JMX Connector
JMX Adimin Service
JMX Message beans
Security Services
Naming Services

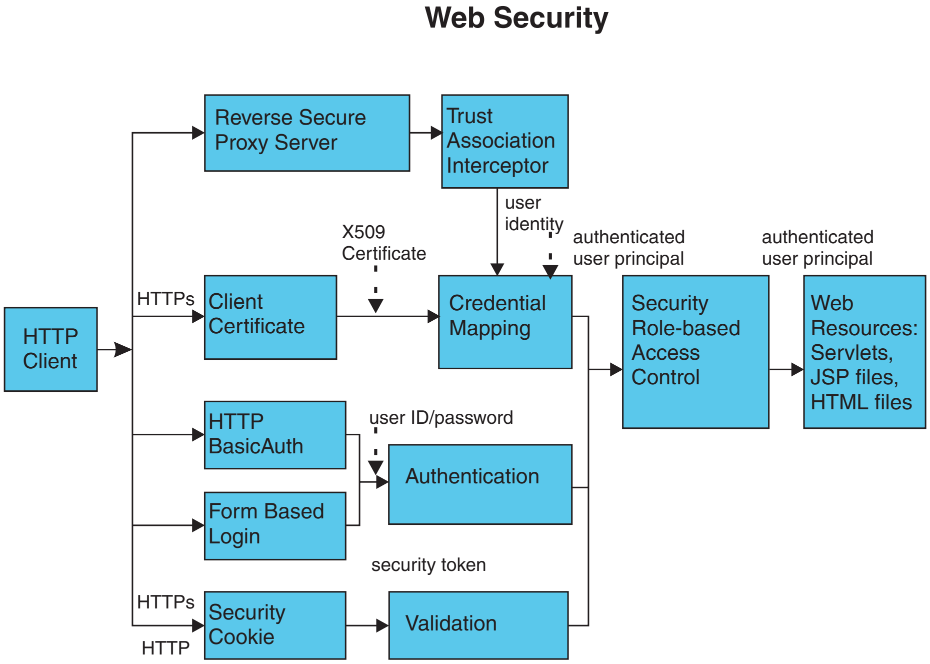
All of the application server processes by default share a common security configuration which is defined at a cell level security XML document. The security configuration determines whether product security is enforced, whether Java 2 Security is enforced, the authentication mechanism and user registry configuration, security protocol configurations, JAAS login configurations, and Secure Socket Layer configurations. Applications may have their unique security requirements. Each application server process may create a per server security configuration to address its own security requirement. Not all security configuration can be modified at application server level. Those can be modified at application server level include whether application security should be enforced, whether Java 2 Security should be enforced, and security protocol configurations. The administrative subsystem security configuration is always determined by the cell level security document. The Web container and EJB container security configuration are determined both by the optional per server level security document and by the cell level security document, while the former if exist has precedence over the latter.

Security configuration, both at cell level and at application server level, are managed either by the Web based administrative console application or by the wsadmin scripting application.

Web security

When security policy is specified for a Web resource and IBM WebSphere Application Server security is enforced, the Web container performs access control when the resource is requested by a Web client. The Web container would challenge the Web client for authentication data if none present according to the specified authentication method, ensure the data constraints are met, and determine whether the authenticated user has the required security role. The product supports the following login methods: HTTP Basic Authentication, HTTPS (SSL) Client Authentication, and Form Based Login. Mapping a client certificate to product security credential uses the UserRegistry implementation to perform the mapping. The LDAP UserRegistry supports the mapping function while LocalOS UserRegistry does not.

When LTPA authentication mechanism is configured and Single SignOn (SSO) is enabled, an authenticated client will be issued a security cookie which can be used to represent the user within the specified security domain. It is recommended to use SSL to protect the security cookie form being intercepted and from being replayed. When Trust Association is configured, the product can map an authenticated user identity to security credentials based on the trust relationship established with the secure reverse proxy server.



The Web security collaborator enforces role-based access control by using an access manager implementation. An access manager make authorization decision based on security policy derived from the deployment descriptor. An authenticated user principal is allowed to access the requested Servlet or JSP file if it has one of the required security roles. Servlets and JSP files can use the HttpServletRequest methods isUserInRole and getUserPrincipal. As an example the administrative console uses isUserInRole method to determine the proper set of administrative functionality to expose to a user principal.

When a servlet or JSP file access EJB methods, either the caller identity or a Run As identity is propagated to the EJB container depending on the Run As configuration. The product supports the JAAS programming model. Servlet and JSP file may also perform a JAAS login to the product security domain and execute code under the JAAS Subject identity by using WSSubject doAs or doAsPrivileged methods. Code in the doAs and doAsPrivileged PrivilegedAction block is executed under the JAAS Subject identity, otherwise it is executed under either the specified Run As identity or the caller identity depending on the Run As configuration.

EJB security

When security is enabled, EJB container will enforce access control on EJB method invocation. The authentication would take place regardless of whether method permission was defined for the specific EJB method.

A Java application client can provide the authentication data in several ways. Using the sas.client.props properties file, a Java client can specify whether to use user id and password to authenticate or to use SSL client certificate to authenticate. The client certificate is stored in the key file or in the hardware cryptographic card as defined in sas.clinet.props. The user ID and password may be optionally defined in the sas.client.props file. At runtime, the Java client may either perform a programmatic login or perform a ″lazy″ authentication. In lazy authentication when the Java client is accessing a protected EJB for the first time the security runtime would try to obtain the required authentication data. Depending on the configuration setting in sas.client.props the security runtime would either look up the authentication data from sas.client.props or prompt the user. Alternatively a Java client can use programmatic login. The product supports the JAAS programming model and the JAAS login (LoginContext) is the recommended way of programmatic login. The login_helper request_login helper function is deprecated in Version 5. Java clients programmed to the login_helper APT can run in this version.

The EJB security collaborator enforces role-based access control by using an access manager implementation. An access manager make authorization decision based on security policy derived from the deployment descriptor. An authenticated user principal is allowed to access the requested EJB method if it has one of the required security roles. EJB code may use the EJBContext methods isCallerInRole and getCallerPrincipal. EJB code may also use JAAS programming model performing JAAS login and WSSubject doAs and doAsPrivileged. The code in doAs and doAsPrivileged PrivilegedAction block is executed under the Subject identity. Otherwise EJB method is executed either under the Run As identity or the caller identity depending on the Rub As configuration. J2EE Run As specification is at EJB bean level. When Run As identity is specified, it applies to all bean methods. The method level IBM Run As extension introduced in Version 4.0 continues to be supported in this version.

Chapter 2. Securing applications and their environment

WebSphere Application Server supports the J2EE model for creating, assembling, securing, and deploying applications. This document provides a high-level description of what is involved in securing resources in a J2EE environment. Applications are often created, assembled and deployed in different phases and by different teams.

Consult the J2EE specifications for complete details.

Steps for this task

1. Plan to secure your applications and environment.

   Be sure to complete this step before you install the WebSphere Application Server.
2. Consider pre- and post- installation requirements.

   For example, during this step, you learn how to protect security configurations after you install the product.
3. Migrate your existing security systems.
4. Develop secured applications.
5. Assemble secured applications.

   Development tools, such as the WebSphere Application Assembly Tool (AAT) and the Deployment Tool for Enterprise JavaBeans (EJBDeploy) is used to assemble J2EE modules and to set the attributes in the deployment descriptors.

   Most of the steps in assembling J2EE applications involve deployment descriptors; the deployment descriptors play a central role in application security in a J2EE environment.

   Application assemblers combine J2EE modules, resolve references between them, and create from them a single deployment unit, typically an `.ear` file. Component providers and application assemblers can be the same people, but they do not have to be.
6. Deploy secured applications.

   Deployer link entities referred to in an enterprise application to the runtime environment. One of the important tasks the deployer performs is mapping actual users and groups to application roles. The deployer installs the enterprise application into the environment and makes the final adjustments needed to run the application.
7. Test secured applications.
8. Manage secured applications.
9. Improve performance by tuning security configurations.
10. Troubleshoot security configurations.

Results

Your applications and production environment are secured.

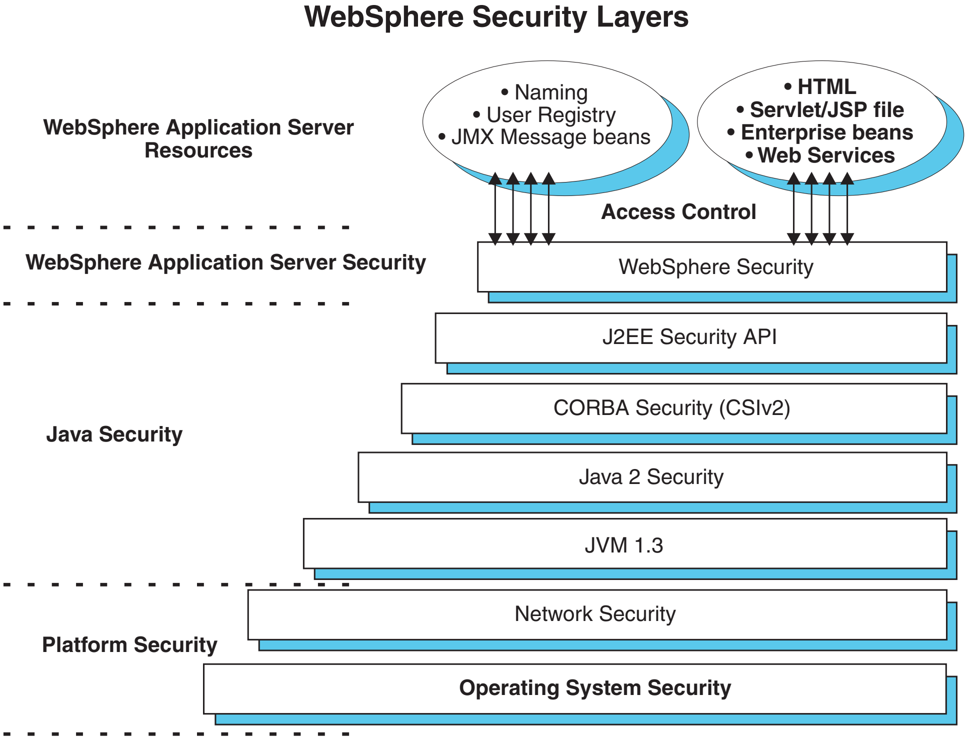
Usage scenario

See Security: Resources for Learning for more information on the WebSphere Application Server Security Architecture.

Planning to secure your environment

Before you begin

There are several communication links from a browser on the Internet, through web servers and product servers, to the enterprise data at the back end. In this section we will examine some typical configuration and common security practice. WebSphere Application Server security is built on a layered security architecture as showed below. In this section we will also examine the security protection offered by each security layer and common security practice for good quality of protection in end-to-end security. The following figure illustrates the building blocks that comprise the operating environment of WebSphere Security:



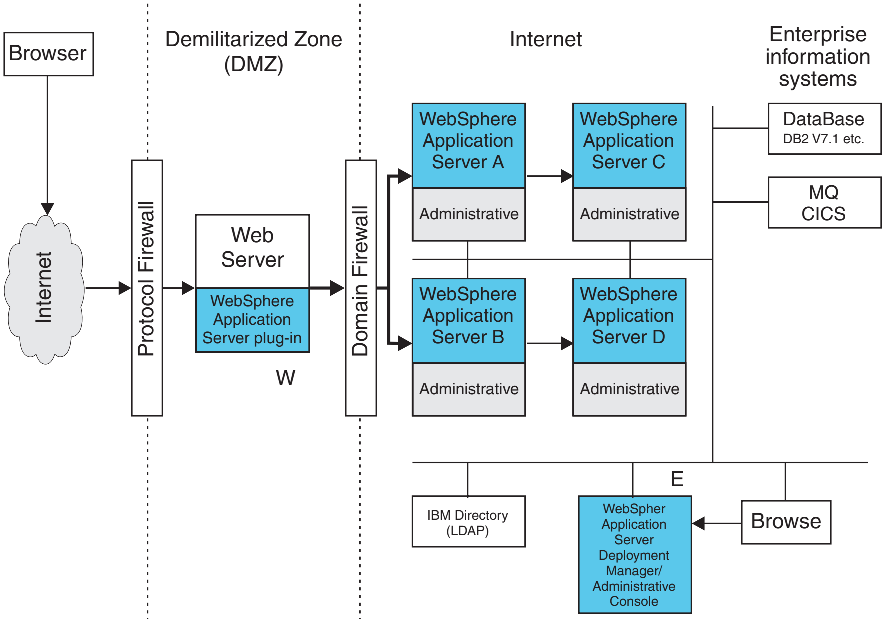
- **Operating System Security** - The security infrastructure of the underlying operating system provides certain security services to the WebSphere Security Application. This includes the file system security support to secure sensitive files in WebSphere product installation. The WebSphere system administrator can configure the product to obtain authentication information directly from the operating system user registry, for example the Windows NT system Security Access Manager (SAM).
- **Network Security** - The Network Security layers provides transport level authentication and message integrity and encryption. WebSphere Application Server inter-servers communications can be configured to use Secure Socket Layer (SSL) and HTTPS. Additionally IP Security and Virtual Private Network (VPN) may be used for added message protection.
- **JVM 1.3.1** - The JVM security model provides a layer of security above the operating system layer.
- **Java 2 Security** - The Java 2 Security model offers fine grained access control to system resources including file system, system property, socket connection,

threading, class loading, etc. Application code must be explicitly grant the required permission in order to access a protected resource.

- **CORBA Security** - Any calls made among secure ORBs are invoked over the Common Security Inter operability Version 2 security protocol that sets up the security context and the necessary quality of protection. After the session is established, the call is passed up to the enterprise bean layer. WebSphere Application Server continue to support the Secure Association Service (SAS) security protocol which was used in prior releases of WebSphere Application Server and other IBM products for backward compatibility.
- **J2EE Security** - The security collaborator enforces J2EE based security policies and supports J2EE security APIs.
- **WebSphere Security** - WebSphere security enforces security policies and services in a unified manner on access to Web resources, enterprise beans, and JMX administrative resources. It consists of WebSphere security technologies and features to support the needs of a secure enterprise environment.

The following picture shows a typical multiple-tier business computing environment. Shown in the picture is a WebSphere Application Server Network Deployment (ND) installation. Note that there is a Node Agent instance on every computer node which is omitted in the picture. Each product application server consists of a web container and an EJB container shown in the yellow shaded area and the administrative subsystem shown in red shaded area. The WebSphere Application Server deployment Manager contains only WebSphere administrative code and the administrative console application. Administrative console is a special J2EE Web Application that provides the GUI interface for performing administrative functions. WebSphere Application Server configuration data is stored in XML descriptor files. Those XML configuration files should be protected by operating system security. Passwords and other sensitive configuration data can be modified via administrative console. Hence, the administrative console Web application has setup data constraint which requires the administrative console servlets and JSP files to be accessed only through SSL connection when global security is enabled.

After installation, the administrative console HTTPS port is configured to use **DummyServerKeyFile.jks** and **DummyServerTrustFile.jks** with the default self signed certificate. Using the Dummy key and trust file certificate is not safe and you should generate your own certificate to replace dummy ones immediately. It is more secure if you first enable global security and complete other configuration tasks after global security is enforced.


Browser
Demilitarized Zone (DMZ)
Internet
Enterprise information systems
Internet
Protocol Firewall
Web Server
WebSphere Application Server plug-in
W
Domain Firewall
WebSphere Application Server A
Administrative
WebSphere Application Server C
Administrative
WebSphere Application Server B
Administrative
WebSphere Application Server D
Administrative
DataBase
DB2 V7.1 etc.
MQ
CICS
E
IBM Directory (LDAP)
WebSpher Application Server Deployment Manager/ Administrative Console
Browse

WebSphere Application Server servers interact with each other via CSIv2 and SAS security protocols as well as HTTP and or HTTPS protocols. Those protocols can be configured to use SSL when WebSphere Application Server global security is enabled. The WebSphere Application Server administrative subsystem in every server uses SOAP JMX connectors and or RMI/IIOP JMX connectors to pass administrative commands and configuration data. When global security is disabled, the SOAP JMX connector uses HTTP protocol and the RMI/IIOP connector uses TCP/IP protocol. When global security is enabled, the SOAP JMX connector always uses HTTPS protocol. When global security is enabled, the RMI/IIOP JMX connector may be configured to either use SSL or to use TCP/IP. Again it is recommended to enable global security and enable SSL to protect the sensitive configuration data.

Note: Global security and administrative security configuration is at the cell level.

While global security is enabled, application security at each individual application server may be disabled by disabling per server level security enable flag. Disabling application server security does not affect the administrative subsystem in that application server which is controlled only by the global security configuration. Both administrative subsystem and application code in an application server share the optional per server security protocol configuration. For more information, see the Configuring Server Security for Network Deployment article.

Security for J2EE resources is provided by Web container and EJB container. Each container provides two kind of security: declarative security and programmatic security.

In declarative security, an application security structure includes data integrity and confidentiality, authentication requirements, security roles, and access control is expressed in a form external to the application. In particular the deployment descriptor is the primary vehicle for declarative security in the J2EE platform. WebSphere maintains J2EE security policy including information derived from the deployment descriptor and specified by deployers and administrators in a set of

XML descriptor files. At runtime, the container uses the security policy defined in the XML descriptor files to enforce data constraints and access control.

When declarative security alone is not sufficient to express the security model of an application, (programmatic security) may be used by application code to make access decisions. When global security is enabled and application server security was not disabled at per server level, J2EE applications security will be enforced. When security policy is specified for a web resource, the web container performs access control when the resource is requested by a web client. The web container would challenge the web client for authentication data if none present according to the specified authentication method, ensure the data constraints are met, and determine whether the authenticated user has the required security role. The web security collaborator enforces role-based access control by using an access manager implementation. An access manager make authorization decision based on security policy derived from the deployment descriptor. An authenticated user principal is allowed to access the requested Servlet or JSP file if it has one of the required security roles. Servlets and JSP pages may use the `HttpServletRequest` methods `isUserInRole` and `getUserPrincipal`. When global security is enabled and application server security is not disabled, EJB container will enforce access control on EJB method invocation. The authentication would take place regardless of whether method permission was defined for the specific EJB method. The EJB security collaborator enforces role-based access control by using an access manager implementation. An access manager make authorization decision based on security policy derived from the deployment descriptor. An authenticated user principal is allowed to access the requested EJB method if it has one of the required security roles. EJB code may use the `EJBContext` methods `isCallerInRole` and `getCallerPrincipal`. The J2EE role based access control should be used to protect valuable business data from being accessed by unauthorized users from both the Internet and the Intranet. For enabling J2EE application security, please refer to Securing Web Applications and Securing EJB Applications. WebSphere extended security role based access control to administrative resources including the JMX system management subsystem, user registry, and JNDI name space. WebSphere administrative subsystem defines four administrative security roles:

- **Monitor role**, which can view configuration information and status but not anything more
- **Operator role**, which is a monitor that can trigger runtime state changes, such as start an application server or stop an application, but cannot change configuration
- **Configurator role**, which is a monitor that can modify configuration information but cannot change runtime state
- **Administrator role**, which is an operator as well as a configurator

A user with the configurator role can perform most administrative work including installing new applications and application servers. There are certain configuration tasks a configurator does not have sufficient authority to do when global security is enabled, including modifying WebSphere Application Server server identity and password, LTPA password and keys, and assigning users to administrative security roles. Those sensitive configuration tasks require the administrative role because the server id is mapped to the administrator role.

WebSphere Application Server administrative security is enforced when global security is enabled. It is recommended that WebSphere Application Server global security be enabled to protect administrative subsystem integrity. Application server security can be selectively disabled if there is no sensitive information to protect. For securing administrative security, refer to (Assigning Users to Admin

Roles Task) and (Assigning Users to Naming Roles) WebSphere Application Server uses Java 2 Security Model to create a secure environment to run application code. Java 2 Security provides a fine, grained and policy based access control to protect system resources such as files, system properties, opening socket connections, loading libraries, and so on. J2EE 1.3 Specification defines typical set of Java 2 security permissions that Web and EJB components should expect to have, which is shown in the table below.

J2EE Security Permissions set for Web components

Security Permission	**Target**	**Action**
java.lang.RuntimePermission	loadLibrary	
java.lang.RuntimePermission	queuePrintJob	
java.net.SocketPermission	*	connect
java.io.FilePermission	*	read, write
java.util.PropertyPermission	*	read

J2EE Security Permissions set for EJB components

Security Permission	Target	Action
java.lang.RuntimePermission	queuePrintJob	
java.net.SocketPermission	*	connect
java.util.PropertyPermission	*	read

WebSphere Application Server Java 2 Security implementation was based on J2EE 1.3 Specification. The Specification granted Web components read and write file access permission to any file in the file system, which may be too broad. WebSphere Application Server default policy gives Web components read and write permission to the sub directory and the sub tree where the Web module was installed. The default Java 2 security policy for all Java virtual machines and WebSphere Application Server server processes are contained in the following policy files:

- `${java.home}/jre/lib/security/java.policy` - default policy for JVM
- `${user.install.root}/properties/server.policy` - default policy for all product server processes

To simplify policy management, WebSphere Application Server policy is based on resource type rather than code base (location). Default policy for WebSphere Application Server subsystem that considered as an extension of WebSphere Application Server run time, which is referred to as *SPI*, for library shared by multiple applications, and for J2EE applications, are:

- `${was.install.root}/config/cells/<cellname>/nodes/<nodename>/spi.policy`

  , which is for embedded resources defined in resources.xml, such as JMS, JavaMail and JDBC drivers.
- `${was.install.root}/config/cells/<cellname>/nodes/<nodename>/library.policy`

  , which is for shared library defined by Web Admin Console.
- `${was.install.root}/config/cells/<cellname>/nodes/<nodename>/app.policy`

  , which is the default policy for J2EE applications.

In general applications should not require more permissions to run than those recommended by the J2EE Specification in order to be portable among various application servers. But some applications may require more permissions. WebSphere Application Server allows a per application policy file, was.policy, to be packaged together with each application from granting extra permissions to that application. Granting extra permissions to an application should be handled with great care because of the potential of compromising system integrity.

WebSphere Application Server uses a permission filtering policy file to alert users when an application requires permissions that are on the filter list during application installation and cause the offended application installation to fail. For example, the `java.lang.RuntimePermission` *`exitVM`* permission should not be given to an application so that no application code is allowed to terminate the WebSphere Application Server application server. The filtering policy is defined by the `filterMask` in `${was.install.root}/config/cells/<cellname>/filter.policy`. Moreover, WebSphere Application Server also performs runtime permission filtering based on the runtime filtering policy to ensure no application code has been granted any permission that is considered harmful to system integrity. Applying Java 2 Security model to application server is new.

WebSphere Application Server Version 4 supported Java 2 Security but only enforce three permission checking against `exitVM`, create and set the Security Manager. Other permission checking are disabled by default. Hence many applications developed for prior releases of WebSphere Application Server may not be Java 2 Security ready. To migrate those applications to WebSphere Application Server Version 5 quickly, you may temporarily give those applications `java.security.AllPermission` in the `was.policy` file. It is recommended to test or make those applications Java 2 Security ready, i.e., identity what extra permissions, if any, are required and to just grant those permissions to a particular application. Not granting applications `AllPermission` can certainly reduce the risk of compromising system integrity. For more information on migrating applications to WebSphere Application Server Version 5, refer to (Migrating Java 2 Security).

WebSphere Application Server runtime uses Java 2 Security to protect sensitive runtime functions and hence it is always a good idea to enforce Java 2 Security. Applications that are granted with `AllPermission` not only have access to sensitive system resources but also WebSphere Application Server runtime resources and can potential cause damage to both. In cases where an application can be trusted to be safe, WebSphere Application Server allows Java 2 Security to be disabled on a per application server basis. In other words, you can enforce Java 2 Security by default in security center and disable the per application server Java 2 Security flag to disable it at the particular application server.

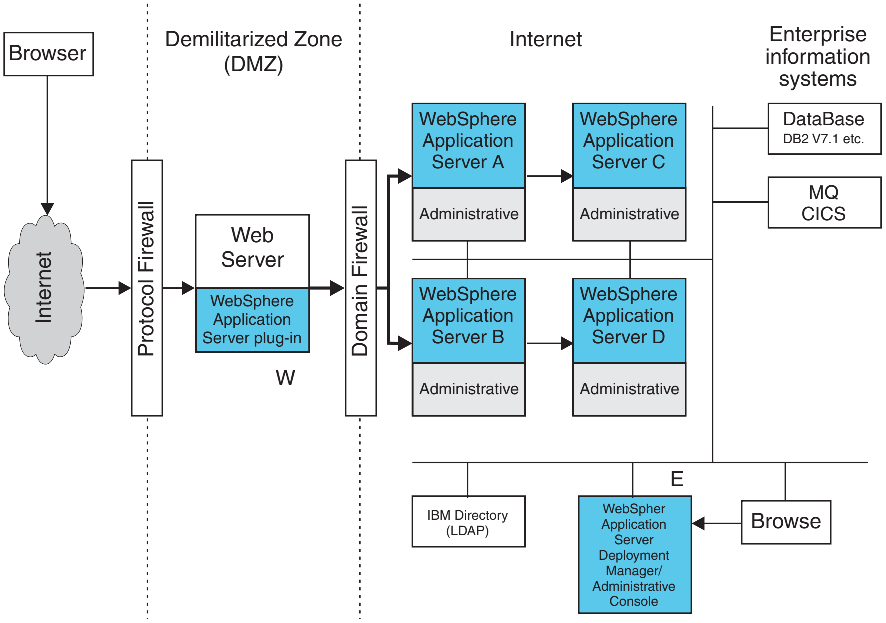
The global security enable flag and Java 2 Security enable flag along with other sensitive configuration data are stored in a set of XML configuration files. Both role based access control and Java 2 Security permission based access control are employed to protect the integrity of the configuration data. We will use configuration data protection as an example to illustrate how system integrity is maintained.

- When Java 2 Security is enforced, application code cannot access the WebSphere Application Server runtime classes that manages the configuration data unless it has been granted the required WebSphere Application Server runtime permissions.

- When Java 2 Security is enforced, application code cannot access the WebSphere Application Server configuration XML files unless it has been granted the required file read and write permissions.
- The JMX administrative subsystem provides SOAP over HTTP(S) and RMI/IIOP remote interface to allow application programs to extract and to modify configuration files and data. When global security is enabled, an application program can modify WebSphere Application Server configuration provided that the application program has presented valid authentication data and that the security identity has the required security roles.
- If a user is allowed to disable Java 2 Security, then that user can modify WebSphere Application Server configuration including the WebSphere Application Server security identity and authentication data along with other sensitive data. Hence, only users with the administrator security role is allowed to disable Java 2 Security.
- Because WebSphere Application Server security identity is given the administrator role, only users with administrator role are allowed to disable global security, to change server id and password, and to map users and groups to administrative roles, and so on.

Other WebSphere Application Server run time resources are protected by similar mechanism as described above. Hence it is very important to enable WebSphere Application Server global security and to enforce Java 2 Security. J2EE Specification defines four authentication method for Web components and WebSphere Application Server supports HTTP Basic Authentication, Form Based Authentication, and HTTPS Client Certificate Authentication. When using client certificate login, it is more convenient for the browser client if the web resources have integral or confidential data constraint. If a browser uses HTTP to access the web resource, the web container will automatically redirect it to HTTPS port. The CSIv2 security protocol also supports client certificate authentication. SSL client authentication can also be used to setup secure communication among selected set of servers based on trust relationship.

Starting from the WebSphere Application Server plug-in at web server, SSL mutual authentication may be configured between it and the WebSphere Application Server HTTPS server. When using self signed certificate, one can restrict the WebSphere Application Server plug-in to communicate with only the selected two WebSphere Application Server servers as shown in the picture. Suppose you want to restrict the HTTPS server in WebSphere Application Server server **A** and in WebSphere Application Server **B** to accept secure socket connection only from WebSphere Application Server plug-in **W**. You can generate three self-signed certificate using the IKEYMAN tool and certificate management utility, say certificate **W, A**, and **B**. WebSphere Application Server plug-in is configured to use certificate **W** and trust certificate **A** and **B**. The HTTPS server of WebSphere Application Server A is configured to use certificate **A** and to trust certificate **W**. The HTTPS server of WebSphere Application Server **B** is configured to use certificate **B** and to trust certificate **W**.



The trust relationship is shown in the following table.

Server	**Key**	**Trust**
WebSphere Application Server -plug-in	W	A, B
WebSphere Application Server Server A	A	W
WebSphere Application Server Server B	B	W

In a Network Deployment installation, the WebSphere Application Server deployment manager is a central point of administration. System management commands is sent from the Deployment manager to each individual WebSphere Application Server Server. When global security is enabled, all WebSphere Application Server servers may be configured to require SSL and mutual authentication. Suppose one wants to further restrict that WebSphere Application Server Server application A can only communicate to WebSphere Application Server application server C and WebSphere Application Server application server B can only communicate to WebSphere Application Server application server D. Note that as mentioned above, all WebSphere Application Server application servers must be able to communicate with WebSphere Application Server Deployment Manager E. Hence, when using self-signed certificates, one could setup CSIv2 and SOAP/HTTPS Key and trust relationship as shown in the following table.

Server	Key	Trust
WebSphere Application Server Server A	A	C, E
WebSphere Application Server Server B	B	D, E
WebSphere Application Server Server C	C	A, E
WebSphere Application Server Server D	D	B, E
WebSphere Application Server Deployment Manager E	E	A, B, C, D

When WebSphere Application Server is configured to use an LDAP user registry, SSL with mutual authentication may also be configured between every WebSphere Application Server server and the LDAP server with self-signed certificate so that no password will be passed in clear text from WebSphere Application Server to LDAP server. In this example Node Agent processes were not discussed. Each node agent needs to communicate with application servers on the same node and with the Deployment Manager. Node agents also need to communicate with LDAP servers when configured to use LDAP user registry. It is reasonable to let Deployment manager and node agents use the same certificate. Suppose application server **A** and **C** are on the same computer node. Node agent on that node needs to have certificates **A** and **C** in its trust file. WebSphere Application Server does not provide a user registry configuration or management utility. In addition, it does not dictate user registry password policy. It is recommended to use the password policy recommended by your user registry, including the password length and expiration period.

Steps for this task

1. Determine which versions of WebSphere Application Server you are using.
2. Review the WebSphere Application Server security architecture.
3. Review each of the following topics as also defined in Related reference.
   - Authentication Protocol for EJB Security
     - CSIv2 Features
     - Identity Assertion
     - Message layer authenticaiton
     - Secure Socket Layer Client Certificate authentication
     - Supported IBM Protocols
   - Authentication Mechanisms
     - SWAM
     - LTPA
     - Trust Associations
     - Single Sign-On (SSO)
   - User Registries
     - LocalOS
     - LDAP
   - Custom user registries
   - Java 2 Security
     - Dynamic Policy
   - Java Authentication and Authorization Service (JAAS)
     - Programmatic login
   - Java 2 Connector Security
   - Access Control
     - J2EE Role-Based Authorization
     - Administrative Console and Naming Service Authorization
   - Secure Socket Layer (SSL) Transport
     - Authenticity
     - Confidentiality
     - Integrity

Security considerations when adding a Base Application Server node to Network Deployment

Before you begin

At some point, you might decide to centralize the configuration of your standalone base application servers by adding them into a Network Deployment cell. If your base application server is currently configured with security, there are some issues to be considered. The major issue when adding a node to the cell is whether the user registries between the base application server and the Deployment Manager are the same. When adding a node to the cell, you automatically inherit both the user registry and the authentication mechanism of the cell. For distributed security, all servers in the cell must use the same user registry and authentication mechanism. In order to recover from a user registry change, you will need to modify your applications so that the user and group to role mappings are correct for the new user registry. To do this, see the article on (Mapping users and groups to J2EE application roles).

Another major issue is the SSL public-key infrastructure. Prior to performing `addNode` with the Deployment Manager, ensure that `addNode` can communicate as an SSL client to the Deployment Manager. This requires that the addNode truststore (configured in `sas.client.props`) contains the signer certificate of the Deployment Manager personal certificate as found in the keystore (specified in the Administrative Console). See the article,(Managing digital certificates).

The following are other issues to consider when running the `addNode` command with security:

Steps for this task

1. When attempting to run system management commands such as `addNode`, you need to explicitly specify administrative credentials to perform the operation. The `addNode` command accepts `-username` and `-password` parameters to specify the userid and password, respectively. The user ID and password, which are specified should be an administrative user, for example, a user that is a member of the console users with **Operator** or **Admistrator** privileges or the admistrative user ID configured in the User Registry. An example for `addNode`, `addNode CELL_HOST 8879 -includeapps -username user -password pass`. Remember that `-includeapps` is optional, but will attempt to include the server applications into the Deployment Manager. Again, if the user registries are not the same between the base application server and the Deployment Manager, there will be some mappings that need repair after the application has been added. See the addNode command for more information on addNode syntax.
2. The addNode command will fail in the following situation:
   - The base application server is started with security enabled.
   - The base application server is using a different user registry than the Deployment Manager.

   To resolve this problem, stop the application server prior to running the `addNode` command. This allows the `-username` and `-password` to apply to the Deployment Manager only.
3. Before running the `addNode` command, you must ensure that the truststore files on the nodes will communicate with the keystore files from the Deployment Manager and vice versa. When using the default `DummyServerKeyFile` and `DummyServerTrustFile`, you should not see this problem as these are already

able to communicate. However, you should never use these files in a production environment or anytime sensitive data is being transmitted.

4. After running addNode, the application server is in a new SSL domain. It might contain SSL configurations that point to keystore and truststore files that are not prepared to interoperate with other servers in the same domain. Consider which servers will be intercommunicating and ensure that the servers are trusted within your truststore files.

Results

Proper understanding of the security interactions between distributed servers greatly reduces problems encountered with secure communications. Security adds complexity because additional function needs to be managed. For security to function, it needs throrough consideration during the planning of your infrastructure. This document helps to reduce the problems that could occur due to inherent security interactions.

What to do next

When you have security problems related to the WebSphere Application Server Network Deployment environment, check the Security Troubleshooting section to see if you can get information about the problem. When trace is need to solve a problem, because servers are distributed, quite often it is required to gather trace on all servers simultaneously while recreating the problem. This trace can be enabled dynamically or statically, depending on the type problem occurring.

Implementing security considerations during installation

Steps for this task

1. Migrate your security configurations from previous releases.
2. Secure your environment before you install the WebSphere Application Server product.
3. Install the WebSphere Application Server. Refer to WebSphere Application Server product Installation documentation.

   During installation you will be prompted to migrate your security configurations.
4. Secure your environment after you install the WebSphere Application Server product.

Securing your environment before installation

Before you begin

To perform a product installation with proper authority, in the UNIX environment, log on as root and make sure unmask value is **022**. In the Windows environment, the log on user must be a member of the administrator group and with the rights of **Act as part of the operating system** and **Log on as a service**. If the Windows machine is a member of a Windows NT domain, then the log on user must also be created on the Active Directory and be a member of the administrator group and with the rights of **Act as part of the operating system** and **Log on as a service** in the Windows NT domain.

When Solaris or Linux is the operating system, make sure that the `/etc` directory contains shadow password file. If the shadow password file (file name is shadow) is missing, it can be created by using `pwconv` command from root ID.

Steps for this task

1. To add the rights to a user on Windows 2000 system:
   a. Click **Start** > **Programs** > **Administrative Tools** > **Local Security Policy** (for domain configuration, select Domain Security Policies, instead).
   b. Then on Local Security Settings Panel, click **Local Policies** > **User Rights Assignment** and add the following rights to the user ID:
      - Act as part of the operating system
      - Log on as a service
2. To create shadow password file on Solaris and Linux systems, login as root and execute `pwconv` command.
3. To check the umask value, execute `umask` command.
4. To set up the umask value as 022, execute `umask 022` command.

Securing your environment after installation

Before you begin

WebSphere Application Server depends on several configuration files created during installation. These files contain password information and should be protected accordingly. Although the files are protected to a limited degree during installation, this basic level of protection is probably not sufficient for your site. You should ensure that those files are protected in compliance with the policies of your site.

The files under the two directories, <*install_root*>\config and <**install_root**>\properties, except those in the following list, should be protected. For example, permission should only be given to the user who logs on the system for WebSphere Application Server primary administrative tasks. Other users or groups, such as WebSphere Application Server Console Users and Console Groups, who perform partial WebSphere Application Server administrative tasks, like configuring, starting servers and stopping servers, should be given permissions as well. The files, under <*install_root*>\properties, which should not be protected are the following:

- TraceSettings.properties
- client.policy
- client_types.xml
- implfactory.properties
- sas.client.props
- sas.stdclient.properties
- sas.tools.properties
- soap.client.props
- wsadmin.properties
- wsjaas_client.conf

Steps for this task

1. Secure properties files on a Windows 2000 system.
   a. Open the Windows Explorer for a view of the files and directories on the machine.
   b. Locate and right-click the file or the directory to be protected.
   c. On the resulting menu, click Properties.
   d. On the resulting dialog, click the Security tab.

e. Remove the Everyone entry and any other users or groups who should not be granted to access to the file.

f. Add the users who should be allowed to access the files with the proper permission.

2. Secure files on UNIX systems.

   This procedure applies only to the ordinary UNIX file system. If your site uses access-control lists, secure the files by using that mechanism. Any site-specific requirements can affect the desired owner, group and corresponding privileges. For example, on AIX,

   a. Go to the directory <*install_root*> and change the ownership of the directories configuration and properties to the user who logs on the system for WebSphere Application Server primary administrative works by executing the following command: `chown logon_user config properties`.

   b. Set up the permission by executing the following command: `chmod -R 770 config properties`.

   c. Go to the directory <*install_root*>\properties, set the following files' permission to everybody by executing the following command: `chmod 777 file_names`.

      where file_names are the following files:

      - TraceSettings.properties
      - client.policy
      - client_types.xml
      - implfactory.properties
      - sas.client.props
      - sas.stdclient.properties
      - sas.tools.properties
      - soap.client.props
      - wsadmin.properties
      - wsjaas_client.conf

   d. Create a group for WebSphere Application Server and put the users who will perform full or partial WebSphere Application Server' administrative works in the that group.

Results

After the protection, only the users given the permission can access the files. Failure to adequately secure these files can lead to a breach of security in your WebSphere applications.

What to do next

In the future, if there is any failure caused by file accessing permission, check if the permission set above are correct.

Migrating security configurations from previous releases

Before you begin

This task is about installing WebSphere Application Server Version 5 with the migration from a previous releases.

- Before migrating the installation, ensure that the administrative server of the previous release is running.
- If security is enabled in the previous release, prepare to have the previous server ID and password ready since it will need to log into the administrative server of the previous release during the migrating installation.
- You can optionally disable security in the previous release before migrating the installation. So, there is no logon required during the installation.

Steps for this task

1. Start the Installation Wizard by running the `install` command.
2. On the Installation Wizard panel, select **Specify previous version information** and **Migrate your applications and configuration from the previous version**. Complete the fields for **Install location** and **Configuration file** with corresponding information.
3. Follow the instructions provided by the Installation Wizard to complete the installation.

Results

The security configuration of previous WebSphere Application Server releases and its applications are migrated to the new installation of WebSphere Application Server Version 5.

Usage scenario

This task is for installation migrating.

What to do next

1. Re-enable the security configurations after the migration when security is disabled in the migrated version even if the previous version had security enabled.

   If Custom User Registry is used in the previous version, the migrating installation does not migrate the class files used by Custom User Registry under the directory of **<*previous_install_root*>\classes** to the product. Therefore, after the migration, copy the following 2 class files into **<*install_root*>\classes**:

   - `FileRegistrySample.class`
   - `RegExpSample.class`

Migrating custom user registries

Before you begin

Before you perform this task, it is expected that you already have a custom user registry implemented and working in WebSphere Application Server Version 4.0. The custom registry in WebSphere Application Server Version 4.0 is based on the CustomRegistry interface. For WebSphere Application Server Version 5, the interface is called the UserRegistry interface. Note that the WebSphere Application Server Version 4.0 based custom registry will work without any changes to the implementation in WebSphere Application Server Version 5 except when the implementation is using datasources to connect to a database during initialization. However, the WebSphere Application Server Version 4.0 version of the CustomRegistry is deprecated in WebSphere Application Server Version 5 and can be removed in the next release. Hence, it is expected that you move your implementation to the WebSphere Application Server Version 5 based interface.

In WebSphere Application Server Version 5, in addition to the UserRegistry interface, the custom user registry requires the Result object to handle users and groups information. This file is already provided in the package and you are expected to use it for the getUsers, getGroups and the getUsersForGroup methods.

In WebSphere Application Server Version 4.0, it might have been possible to use other WebSphere Application Server components to initialize the custom registry. For example, a datasource might have been used to connect to a database based user registry during the initialization of the custom registry. One could have also used EJB's deployed in WAS during initialization. This is no longer possible in WebSphere Application Server Version 5 since other components like the containers are initialized after security and hence are not available during the registry initialization. So in WebSphere Application Server Version 5, a custom registry implementation is expected to be a pure custom implementation and should not depend on other WebSphere Application Server components.

In WebSphere Application Server Version 4.0, if you had display names for users the EJB method getCallerPrincipal( ) and the servlet methods getUserPrincipal( ) and getRemoteUser( ) returned the display names. This behavior has changed in WebSphere Application Server Version 5. By default, these methods now return the security name instead of the display name. However, if you need the display names to be returned you need to set the property WAS_UseDisplayName to true. See the getUserDisplayName method description or the Javadoc for more information on this.

If the migration tool was used to migrate the WebSphere Application Server Version 4.0 configuration to WebSphere Application Server Version 5, you need to be aware that this migration does not involve any changes to your existing code. Since the WebSphere Application Server Version 4.0 custom registry works in WebSphere Application Server Version 5 without any changes to the implementation (except when using datasources) you should be able to use the 4.0 based custom registry after the migration without requiring any modifications to the code. Note that the migration tool may not copy your implementation files from Version 4.0 to Version 5. You might have to copy them to the classpath in the 5.0 setup (preferably to the classes subdirectoy, just like in Version 4.0). If you are using the Network Deployment version you should copy the files to the cell and each of the nodes classpath.

In Version 5, a case insensitive authorization can be performed when using the custom registry. This is new in Version 5.0 and will effect only the authorization check. This will be useful in cases where your custom registry returns inconsistent (in terms of case) results for users and groups Unique Ids.

Note: Setting this flag will not have any effect on the userNames or passwords, only the UniqueIds returned from the registry are lower-cased before comparing with the information in the authorization table (which is also converted to lowercase during runtime).

Also, before proceeding, take a look at the new UserRegistry interface. The section titled ″Developing custom user registries″ under the Developing secured applications describes each of these methods in detail and also indicate the changes from Version 4.0 if any.

The following steps will go through in detail all the changes required to move your WebSphere Application Server Version 4.0 based custom user registry to the 5.0 based custom user registry. The steps are very simple and involve minimal code changes. The sample implementation file is used as an example when describing some of the steps.

Steps for this task

1. Change your implementation to implement `UserRegistry` instead of `CustomRegistry`. Change

   ```
   public class FileRegistrySample implements CustomRegistry
   to
   public class FileRegistrySample implements UserRegistry
   ```
2. Throw the java.rmi.RemoteException in the constructors

   public FileRegistrySample() throws java.rmi.RemoteException
3. Change the mapCertificate method to take a certificate chain instead of a single certificate. Change

   ```
   public String mapCertificate(X509Certificate cert)
   to
   public String mapCertificate(X509Certificate[]cert)
   ```

   Having a certificate chain gives you the flexibility to act on the chain instead of one certificate. If you are only interested in the first certificate just take the first certificate in the chain before processing it. Note that in Version 5, the `mapCertificate` is called to map the user in a certificate to a valid user in the registry, when certificates are used for authentication by the Web and or the Java clients (transport layer certificates, Identity Assertion certificates). In Version 4.0, this was only called by Web clients since the CSIv2 protocol was not supported.
4. Remove the getUsers() method.
5. Change the signature of the getUsers(String) method to return a Result object and accept an additional parameter (int). Change

   ```
   public List getUsers(String pattern)
   to
   public Result getUsers(String pattern, int limit)
   ```

   In your implementation, construct the Result object from the List of the users obtained from the registry (whose number should be limited to the value of the limit parameter) and call the setHasMore() method on the Result object if the total number of users in the registry exceed the limit value.
6. Change the signature of the getUsersForGroup(String) method to return a Result object and accept an additional parameter (int) and throw a new exception called NotImplementedException. Change

   ```
    public List getUsersForGroup(String groupName)
            throws CustomRegistryException,
                   EntryNotFoundException {
   ```

   to

   ```
   public Result getUsersForGroup(String groupSecurityName, int limit)
            throws NotImplementedException,
                   EntryNotFoundException,
                   CustomRegistryException {
   ```

Note: In Version 5, this method is not called by WebSphere Application Server runtime. WebSphere Application Server clients like WorkFlow call this method. Since this would have been already implemented in WebSphere Application Server Version 4.0, you can change the implementation as explained in step 5 for the getUsers method.

7. Remove the getUniqueUserIds(String) method.
8. Remove the getGroups() method.
9. Change the signature of the getGroups(String) method to return a Result object and accept an additional parameter (int).

   change

   ```
   public List getGroups(String pattern)
   ```

   to

   ```
   public Result getGroups(String pattern, int limit)
   ```

   In your implementation, construct the Result object from the List of the groups obtained from the registry (whose number should be limited to the value of the limit parameter) and call the setHasMore() method on the Result object if the total number of groups in the registry exceed the limit value.
10. Add the createCredential method as shown below. You need to throw the NotImplementedException.

    (The following example has been split for publication.)

    ```
    public com.ibm.websphere.security.cred.WSCredential
             throws CustomRegistryException,
                    NotImplementedException,
                    EntryNotFoundException {
          NotImplementedException ex =
                         new NotImplementedException("createCredential not implemented");
          throw ex;
       }
    ```
11. To build the Version 5 implementation make sure you have the sas.jar and wssec.jar in your classpath.

    (The following example has been split for publication.)

    ```
    %WAS_HOME%\java\bin\javac -classpath
    %WAS_HOME%\lib\wssec.jar;%WAS_HOME%\lib\sas.jar FileRegistrySample.java
    ```

    Note: To build the Version 4 custom registry in Version 5, only the `wssec.jar` file is required.
12. Once the classes are built, they should be copied to a directory in the WAS classpath. The `%WAS_HOME%/classes` directory is the preferred location to copy these files. If you are using the Network Deployment product make sure to copy these files to the cell and all the nodes. Without the files in each of the node classpath the nodes and the application servers in those nodes will not start when security is on.
13. Use the administrative console GUI to setup the custom registry. Follow the instructions in the (Configuring custom user registries) article to set up the custom registry including the IgnoreCase flag.

    Make sure you add the the `WAS_UseDataSource` and `WAS_UseDisplayName` properties, if required.

Results

Migrates a Version 4.0 based custom registry to the 5.0 custom registry.

Usage scenario

This step is required to migrate a custom registry from WebSphere Application Server Version 4.0 to WebSphere Application Server Version 5.

What to do next

If you enabling security, make sure you complete the remaining steps. Once this is done, make sure you save the configuration and restart all the servers. Try accessing some J2EE resources to make sure that the custom registry migration was successful.

Migrating CORBA programmatic login to Java Authentication and Authorization Service

Before you begin

WebSphere Application Server Version 5.0 fully supports the Java Authentication and Authorization Service (JAAS) as the programmatic login APIs. See (Configuring Java Authentication and Authorization Service) and Developing with JAAS to login programmatically for more details on JAAS support. Customers are encourage to use JAAS for programmatic login.

This document outlines the deprecated CORBA programmatic login APIs and the alternatives provided by JAAS. The following are the deprecated CORBA programmatic login APISs:

- **${user.install.root}/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/LoginHelper.java**. The `sampleApp` is no included in Version 5.
- **${user.install.root}/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/ServerSideAuthenticator.java**. The `sampleApp` is not included in Version 5.
- **com.ibm.IExtendedSecurity._LoginHelper**. This API is included with the product, but is deprecated.
- **org.omg.SecurityLevel2.Credentials**. This API is included with the product, but not recommended to use.

The alternative APIs provided in WebSphere Application Server Version 5 are a combination of standard JAAS APIs and product implementation of standard JAAS interfaces (also some minor extension). The following is only a summary, refer to the JAAS documentation, which is included with the product (`${was.install.root}/web/docs/jaas/JaasDocs.zip`) and the product Javadoc (`${was.install.root}/web/apidocs/index.html`) for details.

- Programmatic login APIs:
  - javax.security.auth.login.LoginContext
  - javax.security.auth.callback.CallbackHandler interface: The WebSphere Application Server product provides the following implementation of the javax.security.auth.callback.CallbackHandler interface:
    - **com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl**: non-prompt CallbackHandler, application pushes Basic Authentication data (user ID, password and security realm) or Token data to product LoginModules. This is recommended for server side login.

    - **com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl**: GUI login prompt CallbackHandler to gather Basic Authentication data (user id, password and security realm). This is recommended for client side login.

      Note: If this is used on the server side, the server will be blocked for input.
    - **com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl**: stdin login prompt CallbackHandler to gather Basic Authentication data (user id, password and security realm). This is recommended for client side login.

      Note: If this is used on the server side, the server will be blocked for input.
  - javax.security.auth.callback.Callback interface:
    - **javax.security.auth.callback.NameCallback**: provided by JAAS to pass user name to LoginModules
    - **javax.security.auth.callback.PasswordCallback**: provided by JAAS to pass password to LoginModules
    - **com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl**: provided by the produc tto perform Token based login. This would allow application to pass a token byte array to product LoginModules.
  - **javax.security.auth.spi.LoginModule interface**: WebSphere Application Server provides LoginModules implementation for client and server side login, please refer to (Configuring Java Authentication and Authorization Service) for details.
- javax.security.Subject:
  - **com.ibm.websphere.security.auth.WSSubject**: extension provided by the product to invoke remote J2EE resources using the credentials in the javax.security.Subject
  - **com.ibm.websphere.security.cred.WSCredential**: after a successful JAAS login with the WebSphere Application Server LoginModules, a `com.ibm.websphere.security.cred.WSCredential` is created and stored in the Subject
  - **com.ibm.websphere.security.auth.WSPrincipal**: an authenticated principal, this is created and stored in a Subject that is authenticated by WebSphere LoginModules

Steps for this task

1. Use the following as an example of how to perform programmatic login using the CORBA-based programmatic login APIs:

   The CORBA-based programmatic login APIs have been replaced by JAAS login.

   ```
   public class TestClient {
   ...
   private void performLogin() {
   // Get the user's ID and password.
   String userid = customGetUserid();
   String password = customGetPassword();

   // Create a new security context to hold authentication data.
   LoginHelper loginHelper = new LoginHelper();
   try {
   // Provide the user's ID and password for authentication.
   org.omg.SecurityLevel2.Credentials credentials = loginHelper.login(userid, password);

   // Use the new credentials for all future invocations.
   loginHelper.setInvocationCredentials(credentials);
   // Retrieve the user's name from the credentials
   // so we can tell the user that login succeeded.
   ```

```
String username = loginHelper.getUserName(credentials);
System.out.println("Security context set for user: "+username);
} catch (org.omg.SecurityLevel2.LoginFailed e) {
// Handle the LoginFailed exception.
}
}
...
}
```

2. Use the following example to migrate the CORBA-based programmatic login APIs to the JAAS programmatic login.

   The following example assumed that the application code is granted for the required Java 2 Security permissions. See (Configuring Java Authentication and Authorization Service), (Configuring Java 2 Security) and JAAS documentation located in ${was.install.root}/web/docs/jaas/JaasDocs.zip for details. (Some lines of code in the following example have been split for publication.)

```
public class TestClient {
...
private void performLogin() {
// Create a new JAAS LoginContext.
javax.security.auth.login.LoginContext lc = null;

try {
// Use GUI prompt to gather the BasicAuth data.
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl());

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication date is collected by GUI login prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception:
" + e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS Login Configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resources using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is an protected EJB
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception:
" + e.getMessage());
e.printStackTrace();
}
return null;
}
}
);

// Retrieve the principal's name from the Subject
// so we can tell the user that login succeeded,
// should only be one WSPrincipal.
java.util.Set ps =
s.getPrincipals(com.ibm.websphere.security.auth.WSPrincipal.class);
java.util.Iterator it = ps.iterator();
while (it.hasNext()) {
com.ibm.websphere.security.auth.WSPrincipal p =
(com.ibm.websphere.security.auth.WSPrincipal) it.next();
System.out.println("Principal: " + p.getName());
}
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception:
" + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
}
...
}
```

Usage scenario

Migrating CORBA-based programmatic login application to JAAS-based applications.

Migrating from CustomLoginServlet to servlet filters

Before you begin

CustomLoginServlet is deprecated in Version 5.0. Those applications that are using CustomLoginServlet to perform authentication should use Form-based login. Using Form Based login mechanism, the look and feel of the login screen can be controlled. In Form Based login, a login page can be specified that gets displayed to retrieve the user ID and password information. An error page can also be specified that gets displayed when authentication fails.

If login and error page is not enough to implement the CustomLoginServlet, Servlet Filters can be used. Servlet Filters can be used to dynamically intercept requests and responses to transform or use the information contained in the requests or responses. One or more servlet filters can be attached to a Servlet or a group of Servlets. Servlet Filters can also be attached to JSP and HTML pages. All the attached servlet filters are called before invoking the servlet.

Both Form Based login and Servlet Filters are supported by any servlet 2.3 specification compliant Web container. Form login servlet performs the authentication and Servlet Filters can be used to perform additional authentication or auditing or logging information.

In order to perform pre-login and post-login actions using Servlet Filters, Servlet Filters can be configured for either Form login page or for /j_security_check URL. The j_security_check is posted by the form login page with the j_username parameter containing username and j_password parameter containing password. A Servlet Filter can make use of username and password information to perform more authentication or other special needs.

Steps for this task

1. Develop Form login page and Error page for the application as described in Developing form login pages.
2. Configuring Form login page and Error page for the application as described in Securing web applications.

3. Developing Servlet Filters if additional processing is required before and after form login authentication. Refer to Developing servlet filters for form login processing for details.
4. Configuring the Servlet Filters developed in previous step for either the form login page URL or for **/j_security_check** URL. Assembly tool or development tools like WebSphere Application Development Studio can be used to configure filters.

   After configuring servlet filters, the web-xml file will contain two stanzas. The first one contains Servlet Filter configuration, the Servlet Filter and its implementation class. The second one contains the Filter Mapping section, mapping of the servlet filter to URL. In this case Servlet Filter is mapped to /j_security_check.

   ```
   <filter id="Filter_1">
      <filter-name>LoginFilter</filter-name>
      <filter-class>LoginFilter</filter-class>
      <description>Performs pre-login and post-login operation</description>
      <init-param>
         <param-name>ParamName</param-name>
         <param-value>ParamValue</param-value>
      <init-param>
    </filet>

    <filter-mapping>
       <filter-name>LoginFilter</filter-name>
       <url-pattern>/j_security_check</url-pattern>
       </filter-mapping>
   ```

Results

This migration results in an application that uses form-based login and servlet filters without the use of CustomLoginServlet.

Usage scenario

The use of form-based login and servlet filters by the new application should be used to replace CustomLoginServlet. Servlet filters can also be used to perform additional authentication, auditing and logging.

Developing secured applications

IBM WebSphere Application Server provides security components that provide or collaborate with other services to provide authentication, authorization, delegation, and data protection. WebSphere Application Server also supports the security features described in the Java 2 Enterprise Edition (J2EE) specification. An application goes through three stages before it is ready to run:

- Development
- Assembly
- Deployment

Most of the security is configured for an application during the assembly stage. The security configured during assembly stage is called *declarative security* because the security is *declared* or *defined* in the deployment descriptors. The declarative security is enforced by the security run time of which an application developer need not be aware. For some applications, declarative security alone is not sufficient to express the security model of the application. For those applications, you can use programmatic security.

Steps for this task

1. Develop secure Web applications.
2. Develop servlet filters for form login processing.
3. Develop form login pages.
4. Develop EJB component applications.
5. Develop with JAAS to log in programmatically.
6. Develop your own Java 2 security mapping module.
7. Develop custom user registries.
8. Develop a custom interceptor for trust associations.

Developing with programmatic security APIs for Web applications

Before you begin

Programmatic security is used by security aware applications when declarative security alone is not sufficient to express the security model of the application. Programmatic security consists of the following methods of the HttpServletRequest interface.

- **getRemoteUser()**: returns user name the client used for authentication. returns null if no user has been authenticated.
- **isUserInRole** (String rolename): returns true if the remote user is granted the specified security role. If remote user is not granted the specified role or if no user is authenticated, it returns false.
- **getUserPrincipal()**: returns java.security.Principal object containing the remote user name. If no user is authenticated, it returns null.

When `isUserInRole()` is used, a security-role-ref element should be declared in the deployment descriptor with a `role-name` sub-element containing the rolename passed to this method. Since actual roles are created during assembly stage of the application, developer can use logical role as role name and provide enough hints to the assembler in the description of the security-role-ref element to link that role to actual role. During assembly, assembler creates a role-link sub element to link the role-name to actual role. Creation of security-role-ref element is possible if developing tools such as WebSphere Studio Application Developer is used. The security-role-ref element can also be created during assembly stage using assembly tool.

Steps for this task

1. Add required security methods in the servlet code.
2. Create security-role-ref element with role-name field. This is optional. If security-role-ref is not created during development, make sure it is created during assembly stage.

Results

A programmatically secured Servlet application.

Usage scenario

This step is required to secure an application programmatically. An example where it is useful is when a Web application wants to access external resources and wants to control the access to external resources using its own authorization table (external-resource to remote-user mapping). In this case, getUserPrincipal() or getRemoteUser() can be used to get the remote user and then it can consult its own Authorization Table to perform authorization. The remote user information can also be used to retrieve the corresponding users information from an external source such as database or from an EJB. isUserInRole() can also be used similarly.

After development a security-role-ref may be created as show below:

```
<security-role-ref>
<description>Provide hints to assembler for linking this role-name to actual                  role
<role-name>Mgr<\role-name>
</security-role-ref>
```

During assembly, assembler creates role-link as shown below:

```
<security-role-ref>
<description>Hints provided by developer to map role-name to role-link</description>
<role-name>Mgr</role-name>
<role-link>Manager</role-link>
</security-role-ref>
```

Programmatic servlet security methods can be added inside any of the servlet's doGet(), doPost(), doPut(), doDelete() service methods. An example of usage of programmatic security APIs:

```
public void doGet(HttpServletRequest request, HttpServletResponse response) {

                  ....

                  // to get remote user using getUserPrincipal()
                  java.security.Principal principal = request.getUserPrincipal();
                  String remoteUser = principal.getName();

                  // to get remote user using getRemoteUser()
                  remoteUser = request.getRemoteUser();

                 // to check if remote user is granted Mgr role
                 boolean isMgr = request.isUserInRole("Mgr");

                 // use the above information in any way as needed by the application
                 ....

}
```

What to do next

After developing an application, the Application Assembly Tool (AAT) should be used to create roles and link actual role to role-name in the security-role-ref elements.

Example: Web applications code

The following example illustrates a Web application (a servlet) using the programmatic security model. The following example is one usage and not necessarily the only usage of the programmatic security model. The application can use the information returned by getUserPrincipal(), isUserInRole() and getRemoteUser() in any other way that is meaningful to that application. But, it is strongly recommended that the declarative security model be used when ever possible.

```
File : HelloServlet.java

public class HelloServlet extends javax.servlet.http.HttpServlet {

 public void doPost(
  javax.servlet.http.HttpServletRequest request,
  javax.servlet.http.HttpServletResponse response)
  throws javax.servlet.ServletException, java.io.IOException {
 }
public void doGet(
  javax.servlet.http.HttpServletRequest request,
  javax.servlet.http.HttpServletResponse response)
  throws javax.servlet.ServletException, java.io.IOException {

        String s = "Hello";


        // get remote user using getUserPrincipal()
        java.security.Principal principal = request.getUserPrincipal();
        String remoteUserName = "";
        if( principal != null )
          remoteUserName = principal.getName();
// get remote user using getRemoteUser()
        String remoteUser = request.getRemoteUser();

        // check if remote user is granted Mgr role
        boolean isMgr = request.isUserInRole("Mgr");

        // display Hello username for managers and bob.
        if ( isMgr || remoteUserName.equals("bob") )
            s = "Hello " + remoteUserName;

   String message =  "<html> \n" +
              "<head><title>Hello Servlet</title></head>\n" +
         "<body> /n +"
    "<h1> "  +s+ </h1>/n " +
  byte[] bytes = message.getBytes();

  // displays "Hello" for ordinary users
        // and displays "Hello username" for managers and "bob".
        response.getOutputStream().write(bytes);
 }

}
```

After developing the Servlet, you can create a security role reference for the HelloServlet as show below:

```
<security-role-ref>
<description> </description>
<role-name>Mgr</role-name>
</security-role-ref>
```

Developing servlet filters for form login processing

Using the form-based login mechanism, the look and feel of the login screen can be controlled. In Form Based login, a login page can be specified that gets displayed to retrieve the user ID and password information. An error page can also be specified that gets displayed when authentication fails.

If additional authentication or additional processing before and after authentication is required, Servlet Filters can be used. Servlet Filters can be used to dynamically intercept requests and responses to transform or use the information contained in the requests or responses. One or more servlet filters can be attached to a Servlet or a group of servlets. Servlet Filters can also be attached to JSP and HTML pages. All the attached servlet filters are called before invoking the servlet.

Both Form Based login and Servlet Filters are supported by any servlet 2.3 specification complaint web container. Form login servlet performs the authentication and Servlet Filters can be used to perform additional authentication or auditing or logging information.

In order to perform pre-login and post-login actions using Servlet Filters, Servlet Filters can be configured for either Form login page or for `/j_security_check` URL. j_security_check is posted by form login page with `j_username` parameter containing `username` and `j_password` parameter containing password. A Servlet Filter can make use of `username` and password information to perform more authentication or other special needs.

Steps for this task

1. A Servlet Filter should implement javax.servlet.Filter class. There are three methods in the Filter class that needs to be implemented:
   - **init(javax.servlet.FilterConfig cfg)**. This method is called by the container exactly once when the servlet filter is placed into service. The `FilterConfig` passed to this method contains the init-params of the servlet filter. The init-params can be specified for a Servlet Filter during configuration using assembly tool.
   - **destroy()**. This method is called by the container when the servlet filter is taken out of service. Any cleanup required a
   - **doFilter(ServletRequest req, ServletResponse res, FilterChain chain)**. This method is called by the container for every servlet request that is mapped to this filter before invoking the servlet itself. FilterChain passed to this method can be used to invoke the next filter in the chain of filters. The original requested servlet gets executed when the last filter in the chain calls the `chain.doFilter()` method. Therefore, all filters should call chain.doFilter() in order for the original servlet to get executed after filtering. If additional authentication check is implemented in the filter code and that results in failure, the original servlet need not be executed. In this case chain.doFilter() need not be called, instead it can be redirected to some other error page.

   If a servlet is mapped to many servlet filters, servlet filters are called in the order that is listed in the deployment descriptor of the application(web.xml).

   An example of Servlet Filter is shown below: This Login Filter can be mapped to /j_security_check to perform pre-login and post-login actions. (Some lines of code in the following example have been split for publication.)

```
import javax.servlet.*;

                            public class LoginFilter implements Filter {

                                      protected FilterConfig filterConfig;

                                     // Called once when this filter is instantiated.
If mapped to j_security_check, called very first time j_security_check is invoked.
                                     public void init(FilterConfig filterConfig)
throws ServletException {
                                                    this.filterConfig = filterConfig;
                                      }

                                     public void destroy() {
                                                    this.filterConfig = null;
                                     }

                                    // Called for every request that is
```

```
               mapped to this filter.
If mapped to j_security_check, called for every  j_security_check action
                                    public void doFilter(ServletRequest request,
ServletResponse response, FilterChain chain)
                                                  throws java.io.IOException,
                  ServletException   {

                                                   // perform pre-login action here

                                                   chain.doFilter(request, response);
// calls the next filter in chain.  j_security_check if this filter is mapped
   to j_security_check.

                                                   // perform post-login action here.

                                     }
                           }
```

The Servlet Filter class file should be places in the WEB-INF/classes directory of the application.

Configuring servlet filters: WebSphere Application Development Studio or the Application Assembly Tool (AAT) can be used to configure the servlet filters. There are two steps in configuring a servlet filter.

Steps for this task

1. Configuring Servlet Filter : In this step a name can be assigned to servlet filter and the corresponding implementation class can be assigned to the servlet filter. Optionally, initialization parameters that gets passed to the `init()` method of the servlet filter can also be assigned.

   After configuring the Servlet Filter, the application deployment descriptor, `web.xml`, contains Servlet Filter configuration similar to one shown below:

   ```
   <filter id="Filter_1">
      <filter-name>LoginFilter</filter-name>
      <filter-class>LoginFilter</filter-class>
      <description>Performs pre-login and post-login operation</description>
      <init-param>// optional
        <param-name>ParamName</param-name>
        <param-value>ParamValue</param-value>
      </init-param>
   </filter>
   ```

2. Maping Servlet Filter to URL or Servlet : In this step a servlet or a URL pattern can be mapped to the servlet filter.

   After mapping the servlet filter to a servlet or a URL, the application deployment descriptor (web.xml) contains Servlet Mapping similar to one shown below:

   ```
   <filter-mapping>
      <filter-name>LoginFilter</filter-name>
      <url-pattern>/j_security_check</url-pattern>
    // can be servlet <servlet>servletName</servlet>
   </filter-mapping>
   ```

Usage scenario

Servlet filters can be used to replace CustomLoginServlet. Servlet filters can also be used to perform additional authentication, auditing and logging.

Example: Servlet filters: This example illustrates one usage of the servlet filters to perform pre-login and post-login processing during form login.

Servlet Filter source code: LoginFilter.java

```
/**
 * A Servlet filter example: This example filters j_security_check and
 * performs pre-login action to determine if the user trying to log in
 * is in the revoked list. If the user is in revoked list, an error is
 * sent back to the browser.
 *
 * This filter reads the revoked list file name from the FilterConfig
 * passed in the init() method. Reads the revoked user list file and
 * creates a revokedUsers list.
 *
 * When doFilter method is called, the user being logged in is checked
 * to make sure that the user is not in the revoked User list.
 *
 */

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class LoginFilter implements Filter {

   protected FilterConfig filterConfig;

   java.util.List revokeList;


   /**
    * init() : init() method called when the filter is instantiated.
    * This filter is instantiated first time j_security_check is invoked for
    * the application(That is when a protected servlet in the application is accessed).
    */
   public void init(FilterConfig filterConfig) throws ServletException {
      this.filterConfig = filterConfig;

      // read revoked user list
      revokeList = new java.util.ArrayList();
      readConfig();
   }


   /**
    * destroy() : destroy() method called when the filter is taken out of service.
    */
   public void destroy() {
      this.filterConfig = null;
      revokeList = null;
   }

   /**
    * doFilter() : doFilter() method called before the servlet that this filter
    * is mapped is invoked. Since this filter is mapped to j_security_check,
    * this method is called before j_security_check action is posted.
    */
   public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws java.io.IOException, ServletException {


      HttpServletRequest req = (HttpServletRequest)request;
      HttpServletResponse res = (HttpServletResponse)response;

      // pre login action

      // get username
      String username = req.getParameter("j_username");

      // if user is in revoked list send error
      if ( revokeList.contains(username) ) {
          res.sendError(javax.servlet.http.HttpServletResponse.SC_UNAUTHORIZED);
          return;
      }

      // call next filter in the chain : let j_security_check authenticate user
      chain.doFilter(request, response);

      // post login action

   }

   /**
    * readConfig() : Reads revoked user list file and creates a revoked user list.
    */
   private void readConfig() {
      if ( filterConfig != null ) {

         // get the revoked user list file and open it.
         BufferedReader in;
         try {
               String filename = filterConfig.getInitParameter("RevokedUsers");
               in = new BufferedReader( new FileReader(filename));
         } catch ( FileNotFoundException fnfe) {
               return;
         }

         // read all the revoked users and add to revokeList.
         String userName;
         try {
               while ( (userName = in.readLine()) != null )
                   revokeList.add(userName);
         } catch (IOException ioe) {
         }

      }

   }


}
```

Portion of web.xml showing LoginFilter configured and mapped to j_security_check:

```
<filter id="Filter_1">
    <filter-name>LoginFilter</filter-name>
    <filter-class>LoginFilter</filter-class>
    <description>Performs pre-login and post-login operation<
    /description>
    <init-param>
      <param-name>RevokedUsers</param-name>
<param-value>c:\WebSphere\AppServer\installedApps\<app-name>
\revokedUsers.lst</param-value>
    </init-param>
  </filter-id>

  <filter-mapping>
    <filter-name>LoginFilter</filter-name>
    <url-pattern>/j_security_check</url-pattern>
  </filter-mapping>
```

An example of revoked user list file:

```
user1
cn=user1,o=ibm,c=us
user99
cn=user99,o=ibm,c=us
```

Developing form login pages

Before you begin

A Web client (Browser) can authenticate a user to a web server using one of the following mechanisms:

- **HTTP Basic Authentication**: A web server requests the web client to authenticate and web client passes user and password in the HTTP header.
- **HTTPS Client Authentication**: This mechanism requires a user (Web Client) to possess a Public Key Certificate. Web client sends this certificate to a web server that requests for client certificate. This is a strong authentication mechanism and uses HTTPS protocol.
- **Form-Based Authentication**: Using this authentication mechanism a developer can control the look and feel of the login screens.

The HTTP Basic authentication transmits user password from Web client to Web server in simple base64 encoding. Form based authentication transmits user password from browser to web server in plain text. Therefore, both HTTP Basic authentication and Form Based authentication are not very secure unless HTTPS is used.

The Web application deployment descriptor contains information about what authentication mechanism to use. When Form based authentication is used, the deployment descriptor also contains entries for login and error pages. A login page can be either a HTML page or a JSP page. This login page will be displayed on the Web Client side when a secured resource(Servlet,JSP,HTMLpage) is accessed from the application. On authentication failure, error page will be displayed. Developer can write login and error page to suite the application needs and control the look and feel of those pages. During assembly of the application, an assembler can set the authentication mechanism for the application and set the login and error pages in the deployment descriptor.

See the Example: Form login article for sample form login pages.

Steps for this task

1. Create Form login page with the required look and feel including the required elements to perform Form Based Authentication.
2. Create Error page. Error page can be programmed to retry authentication or display an error message that is appropriate.
3. Place the login page and error page in the war file. The login page and error page should be placed relative to the top directory of the war file. For example if the login page is configured as `/login.html` in the deployment descriptor, it should be placed in the top directory of the war as shown in the previous section. An assembler can also perform this step using the assembly tool.
4. Create a Form logout page and insert it to the application only if required.

Usage scenario

This step is required when a Web application requires Form based authentication mechanism.

What to do next

After developing login and error page, they should be added to the Web application. Assembly tool should be used to configure authentication mechanism and insert the developed login page and error page in the deployment descriptor of the application.

Example: Form login

In order for the authentication to proceed appropriately, the action of the login form must always be `j_security_check`. The following is an example of how the form should be coded into the HTML page:

```
<form method="POST" action="j_security_check">
<input type="text" name="j_username">
<input type="text" name="j_password">
<\form>
```

The j_username input field should be used to get the user name and the j_password input field should be used to get the user's password.

On receiving a request from a Web client, the Web server sends the configured Form page to the client and preserves the original request. When Web server receives the completed Form page from the Web client, it extracts the username and password from the form and authenticates the user. On successful authentication, Web server redirects the call to original request. If authentication fails, the web server redirects the call to the configured error page.

Here is an example login page in HTML (login.html):

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
<META HTTP-EQUIV = "Pragma" CONTENT="no-cache">
<title> Security FVT Login Page </title>
<body>
<h2>Form Login</h2>
<FORM METHOD=POST ACTION="j_security_check">
<p>
<font size="2"> <strong> Please Enter user ID and password: </strong></font>
<BR>
<strong> User ID</strong> <input type="text" size="20" name="j_username">
<strong> Password </strong>  <input type="password" size="20" name="j_password">
<BR>
<BR>
<font size="2">  <strong> And then click this button: </strong></font>
<input type="submit" name="login" value="Login">
</p>

</form>
</body>
</html>
```

Here is an example of error page in JSP (error.jsp): (Some lines in the following example have been split for publication.)

```
<sp:useBean id="ErrorReport" scope="request" class=
"com.ibm.websphere.servlet.error.ServletErrorReport"/>
<DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
<head><title>Error <%=ErrorReport.getErrorCode()%>
</head></title>
<body>

<H1>SecFVTServlet1 Error <%= ErrorReport.getErrorCode() %> </H1>
<% if(ErrorReport.getErrorCode() >= 500 && ErrorReport.getErrorCode()
 != response.SC_SERVICE_UNAVAILABLE) {%>
<H4>An error has occured while processing request: <%=
HttpUtils.getRequestURL(request) %></H4>
<B>Message: </B><%= ErrorReport.getMessage() %><BR>
<B>StackTrace:</B><%= ErrorReport.getStackTrace() %>
<%}else if(ErrorReport.getErrorCode() == response.SC_NOT_FOUND){%>
Document Not Found
<%}%>

</html>
```

After assembler configures the Web application to use Form based authentication the deployment descriptor will contain the login configuration as shown below:

```
<ogin-config id="LoginConfig_1">
<auth-method>FORMauth-method>FORM>
<realm-name>Example Form-Based Authentication Area</realm-name>
<form-login-config id="FormLoginConfig_1">
<form-login-page>/login.html</form-login-page>
<form-error-page>/error.jsp</form-error-page>
</form-login-config>
</fogin-config>
```

A sample war(Web Application Archive) file directory structure showing login and error pages for the above login configuration:

```
META-INF
     META-INF/MANIFEST.MF
     login.html
     error.jsp
     WEB-INF/
     WEB-INF/classes/
     WEB-INF/classes/aServlet.class
```

Form logout

Form logout is a mechanism to log out without having to close all Web-browser sessions. After logging out with form logout, access to a protected Web resource requires reauthentication. This feature is not required by J2EE specifications but is provided as an additional feature in WebSphere security.

Suppose that it is desirable to log out after logging into a Web application and performing some actions. A form logout works in the following manner:

1. The logout-form URI is specified in the Web browser and loads the form.
2. The user clicks on the submit button of the form to logout.
3. The WebSphere security code logs the user out.
4. Upon logout, the user is redirected to a logout exit page.

Form logout does not require any attributes in any deployment descriptor. It is simply an HTML or JSP file that is included with the Web application. The form-logout page is like most HTML forms except that, like the form-login page, it has a special post action that is recognized by the Web container, which dispatches it to a special internal WebSphere form-logout servlet. The post action in the form-logout page must be ibm_security_logout.

A logout-exit page can be specified in the logout form, and the exit page can be a HTML or JSP file within the same Web application that the user is redirected to after logging out. The logout-exit page is simply specified as a parameter in the form-logout page. If no logout-exit page is specified, a default logout HTML message is returned to the user. Here is a sample form logout HTML form. This form configures the logout-exit page to redirect the user back to the login page after logout.

```
<!DOCTYPE HTML PUBliC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
  <META HTTP-EQUIV = "Pragma" CONTENT="no-cache">
  <title>Logout Page </title>
  <body>
  <h2>Sample Form Logout</h2>
      <FORM METHOD=POST ACTION="ibm_security_logout" NAME="logout">
      <p>
      <BR>
      <BR>
      <font size="2"><strong> Click this button to logout: </strong></font>
      <input type="submit" name="logout" value="Logout">
      <INPUT TYPE="HIDDEN" name="logoutExitPage" VALUE="/login.html">
      </p>
      </form>
  </body>
</html>
```

Developing with programmatic APIs for EJB applications

Before you begin

Programmatic security is used by security aware applications when declarative security alone is not sufficient to express the security model of the application. The `javax.ejb.EJBContext` interface provides two methods that allow the Bean Provider to access security information about the enterprise bean's caller.

- **IsCallerInRole**(String rolename): returns true if the bean's caller is granted the specified security role(specified by rolename). If caller is not granted the specified role or if caller is not authenticated, it returns false. If the specified role is granted Everyone access, it always returns true.
- **getCallerPrincipal()**: returns java.security. Principal object containing the bean's caller name. If caller is not authenticated, it returns an a principal containing UNAUTHENTICATED name.

When `isCallerInRole()` is used, a security-role-ref element should be declared in the deployment descriptor with a role-name sub-element containing the `rolename` passed to this method. Since actual roles are created during assembly stage of the application, developer can use logical role as role name and provide enough hints to the assembler in the description of the security-role-ref element to link that role to actual role. During assembly, assembler creates a role-link sub element to link the role-name to actual role. Creation of security-role-ref element is possible if developing tools such as WebSphere Studio Application De is used. The security-role-ref element can also be created during assembly stage using assembly tool.

Steps for this task

1. Add required security methods in the EJB module code.
2. Create security-role-ref element with role-name field for all the role-names used in `isCallerInRole()`. This is optional. If security-role-ref is not created during development, make sure it is created during assembly stage.

Results

A secure aware EJB application.

Usage scenario

Hard coding security policies into applications is strongly discouraged. The J2EE model security capabilities to declaratively specify security policies is encouraged to be used wherever possible . These APIs can be used to develop security-aware EJB applications. An example where it is useful is when a EJB application wants to access external resources and wants to control the access to external resources using its own authorization table (external-resource to user mapping). In this case, `getCallerPrincipal()` can be used to get the caller's identity and then it can consult its own Authorization Table to perform authorization. The caller identification can also be used to retrieve the corresponding users information from an external source such as database or from another EJB. isCallerInRole() can also be used similarly.

After development a security-role-ref may be created as show below:

```
<security-role-ref>
<description>Provide hints to assembler for linking
this role-name to actual role here<\description>
<role-name>Mgr<\role-name>
</security-role-ref>
```

During assembly, assembler creates role-link as shown below:

```
<security-role-ref>
<description>Hints provided by developer to map role-name to role-link</description>
<role-name>Mgr</role-name>
<role-link>Manager</role-link>
</security-role-ref>
```

Programmatic EJB component security methods(isCallerInRole() and getCallerPrincipal()) can be added inside any business methods of an ejb. The following is an example of programmatic security APIs. A session bean is used in the example:

```
public class aSessionBean implements SessionBean {

                 .....

                 // SessionContext extends EJBContext. If it is entity bean use EntityContext
                 javax.ejb.SessionContext context;

                 // The following method will be called by the EJB container automatically
                 public void setSessionContext(javax.ejb.SessionContext ctx) {
                              context = ctx; // save the session bean's context
                 }

                  ....

                  private  void aBusinessMethod()  {
                            ....

                           // to get  bean's caller using getCallerPrincipal()
                           java.security.Principal principal = context.getCallerPrincipal();
                           String  callerId= principal.getName();

                          // to check if  bean's caller is granted Mgr role
                         boolean isMgr = context.isCallerInRole("Mgr");

                         // use the above information in any way as needed by the application

                          ....
                 }

                 ....
}
```

What to do next

After developing an application, assembly tool should be used to create roles and link actual role to role-name in the security-role-ref elements.

Example: EJB applications code

The following EJB component example illustrates usage of isCallerInRole() and getCallerPrincipal() in an EJB module. It is recommended that the declarative security be used if possible. The following example is one way of using isCallerInRole() and getCallerPrincipal() result and not the only way. The application can make use of this result in any way that is suitable for the application.

Remote Interface

Usage scenario

```
File : Hello.java

package tests;
import java.rmi.RemoteException;
/**
 * Remote interface for Enterprise Bean: Hello
 */
public interface Hello extends javax.ejb.EJBObject {
      public abstract String getMessage()throws RemoteException;
     public abstract void setMessage(String s)throws RemoteException;
}
```

Home Interface

Usage scenario

```
File : HelloHome.java
package tests;
/**
 * Home interface for Enterprise Bean: Hello
 */
public interface HelloHome extends javax.ejb.EJBHome {
 /**
  * Creates a default instance of Session Bean: Hello
  */
 public tests.Hello create() throws javax.ejb.CreateException, java.rmi.RemoteException;
}
```

Bean Implementation

Usage scenario

```
File : HelloBean.java

package tests;
/**
 * Bean implementation class for Enterprise Bean: Hello
 */
public class HelloBean implements javax.ejb.SessionBean {
 private javax.ejb.SessionContext mySessionCtx;
 /**
  * getSessionContext
  */
 public javax.ejb.SessionContext getSessionContext() {
  return mySessionCtx;
 }
 /**
  * setSessionContext
  */
 public void setSessionContext(javax.ejb.SessionContext ctx) {
  mySessionCtx = ctx;
 }
 /**
  * ejbActivate
  */
 public void ejbActivate() {
 }
 /**
  * ejbCreate
  */
 public void ejbCreate() throws javax.ejb.CreateException {
 }
 /**
  * ejbPassivate
  */
 public void ejbPassivate() {
 }
 /**
  * ejbRemove
  */
 public void ejbRemove() {
 }

 public java.lang.String message;


      //bussiness methods

      // all users can call getMessage()
      public String getMessage() throws java.rmi.RemoteException {
          return message;
      }

      // all users can call setMessage() but only few users can set new message.
      public void setMessage(String s) throws java.rmi.RemoteException {

        // get  bean's caller using getCallerPrincipal()
        java.security.Principal principal = mySessionCtx.getCallerPrincipal();
        java.lang.String  callerId= principal.getName();

        // check if  bean's caller is granted Mgr role
        boolean isMgr = mySessionCtx.isCallerInRole("Mgr");

        // only set supplied message if caller is "bob" or caller is granted Mgr role
        if ( isMgr || callerId.equals("bob") )
            message = s;
        else
            message = "Hello";
      }

}
```

After development of the Entity bean, create a security role reference in the deployment descriptor under the session bean, Hello, as shown below:

```
<security-role-ref>
<description>Only Managers can call setMessage() on this bean (Hello)</description>
<role-name>Mgr</role-name>
</security-role-ref>
```

Developing with JAAS to login programmatically

Before you begin

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server Version 5. It is also mandates by J2EE 1.3 Specification. Java Authentication and Authorization Service represents the strategic APIs for authentication and it will replace of the CORBA programmatic login APIs. WebSphere Application Server has provides some extension to JAAS:

- Rrefer to Developing applications that use CosNaming (CORBA Naming interface) article for details on how to set up setup the environment for thin client applications to access remote resources on a server.
- If the application using custom JAAS Login Configuration, please make sure that the custom JAAS Login Configuration is properly defined See (Configuring Java Authentication and Authorization Service (JAAS) Login Configuration) section for details.
- Some of the JAAS APIs are protected by Java 2 Security permissions, if these APIs are used by application code, please make sure that these permissions are added to the application `was.policy` file See (Adding the was.policy file to Application), (Using the Policytool to edit policy file) and (Configure was.policy) articles for details. For more details of which APIs are protected by Java 2 Security permissions, please check the JDK, JAAS and WebSphere Application Server public APIs javadoc in (Resources for learning) for more details. The following lists some of the APIs used in the sample code in this documentation and the Java 2 Security permissions required by these APIs:
  - javax.security.auth.login.LoginContext constructors are protected by javax.security.auth.AuthPermission ″createLoginContext″.
  - javax.security.auth.Subject.doAs() and com.ibm.websphere.security.auth.WSSubject.doAs() are protected by javax.security.auth.AuthPermission ″doAs″.
  - javax.security.auth.Subject.doAsPrivileged() and com.ibm.websphere.security.auth.WSSubject.doAsPrivileged() are protected by javax.security.auth.AuthPermission ″doAsPrivileged″.
- **`com.ibm.websphere.security.auth.WSSubject`** . Due to a design oversight in the JAAS 1.0, javax.security.auth.Subject.getSubject() does not return the Subject associated with the thread of execution inside a java.security.AccessController.doPrivileged() code block. This can present a inconsistent behavior that is problematic and causes undesireable effort. com.ibm.websphere.security.auth.WSSubject provides a workaround to associate Subject to thread of execution. com.ibm.websphere.security.auth.WSSubject extends the JAAS model to J2EE resources for authorization checks. If the *Subject* associates with the thread of execution within `com.ibm.websphere.security.auth.WSSubject.doAs()` or if the `com.ibm.websphere.security.auth.WSSubject.doAsPrivileged()` code block contains product credentials, the Subject will be used for J2EE resources authorization checks.
- **UI support for defining new JAAS Login Configuration**. JAAS Login Configuration can be configured in the administrative console and stored in WebSphere Common Configuration Model. Application can define new JAAS login configuration in the administrative console and the the data is persisted in the configuration repository (stored in the WebSphere Common Configuration Model). However, WebSphere still support the default JAAS login configuration format (plan text file) provided by the JAAS default implementation. But if there are duplication login configurations defined in both the WebSphere Common

Configuration and the plan text file format, the one in the WebSphere Common Configuration takes precedence. There are advantages to define the login configuration in the WebSphere Common Configuration :

- UI support in defining JAAS login configuration.
- The JAAS configuration login configuration can be managed centrally.
- The JAAS configuration login configuration is distributed in a Network Deployment installation.

- **WebSphere JAAS Login Configurations**. WebSphere provides JAAS Login Configurations for application to perform programmatic authentication to the WebSphere security runtime. These WebSphere JAAS Login Configurations perform authentication to the WebSphere configured authentication mechanism (SWAM or LTPA) and user registry (LocalOS, LDAP or Custom) based on the authentication data supplied. The authenticated Subject from these JAAS Login Configurations contain the required Principal and Credentials that can be used by WebSphere security runtime to perform authorization checks on J2EE role-based protected resources. Here is the JAAS Login Configurations provided by WebSphere:
  - **WSLogin JAAS Login Configuration**. This is a generic JAAS Login Configuration can be used by Java Client, Client Container application, Servlets, JSP, and EJB components, for example. to perform authentication based on a user ID and password, or a token to the the WebSphere security runtime. However, this does not honors the CallbackHandler specified in the Client Container deployment descriptor.
  - **ClientContainer JAAS Login Configuration**. This JAAS Login Configuration honors the CallbackHandler specified in the Client Container deployment descriptor. The Login Module of this Login Configuration will use the CallbackHandler in the Client Container deployment descriptor if one specified, even if the application code specified one CallbackHandler in the LoginContext. This is for Client Container application.

    Note: Subject authenticated with the above JAAS Login Configurations contains a com.ibm.websphere.security.auth.WSPrincipal and a com.ibm.websphere.security.auth.WSCredential. If the authenticated Subject is passed in com.ibm.websphere.security.auth.WSSubject.doAs() (or the other doAs() methods), the WebSphere security runtime can perform authorization checks on J2EE resources based on the Subject com.ibm.websphere.security.auth.WSCredential.
- **Customer defined JAAS Login Configurations**. Customer can define other JAAS Login Configurations. See the (Managing Java Authentication and Authorization Service (JAAS) Login Configuration) article for details. Use these Login Configurations to perform programmatic authentication to the customer's authentication mechanism. However, the Subjects from these customer defined JAAS Login Configurations might not be able to be used by WebSphere security runtime to perform authorization checks if it does not contains the required Principal and Credentials.

Steps for this task

1. Log in programmatically through a non-prompt implementation.

   The product provides an non-prompt implementation of the javax.security.auth.callback.CallbackHandler interface, which is called com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl. This com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl allows application to ″push″ authentication data to the WebSphere LoginModule to perform authentication. This can be useful for server side application code to authenticate an identity and want to use that identity to invoke down stream J2EE resources.

```
javax.security.auth.login.LoginContext lc = null;

try {
lc = new javax.security.auth.login.LoginContext("WSLogin",
  new
  com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl
  ("user","securedpassword"));

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determin how authentication data is collected
// in this case, the authentication data is "push" to the authentication mechanism
//   implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception:
" + e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS Login Configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resources using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is an protected EJB
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception:
" + e.getMessage());
e.printStackTrace();
}
return null;
}
}
);
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

Note: The com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl callback handler can be used by pure Java Client, Client Application Container, EJB components, JSPs, servlets, or any other J2EE resources.

2. Log in programmatically from the graphical user interface implementation.

   WebSphere also provide GUI implementation of javax.security.auth.callback.CallbackHandler interface to collect authentication data from user through GUI login prompt, which is called com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl. This callback handler presents a GUI login panel to prompt user for authentication data.

```
javax.security.auth.login.LoginContext lc = null;

try {
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl());

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determin how authentication data is collected
// in this case, the authentication date is collected by GUI login prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception:
" + e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS Login Configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resources using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is an protected EJB
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception:
" + e.getMessage());
e.printStackTrace();
}
return null;
}
}
);
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

Note: The com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl callback handler should not be used at all for server side resources (like EJB components, servlets, JSPs, or any other server side resouces). The GUI login prompt will block the server for user input. This is not a desirable behavior for a server process.

3. Log in programmatically from the `Stdin` implementation.

   WebSphere also provide `stdin` implementaion of javax.security.auth.callback.CallbackHandler interface to collect authentication data from user through `stdin`, which is called com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl. This callback handler prompt user in the `stdin` for authentication data. (Some lines in the following example have been split for publication.)

```
javax.security.auth.login.LoginContext lc = null;

try {
```

```
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl());

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determin how authentication data is collected
// in this case, the authentication date is collected by
stdin prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception:
" + e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS Login Configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resources using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is an protected EJB
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception:
" + e.getMessage());
e.printStackTrace();
}
return null;
}
}
);
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

Note: The com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl callback handler should not be used at all for server side resources (like EJB components, servlets, JSP, or any other server side resouces). The `stdin` prompt does not make send in the server environment, most server usually run in the background do not have a console. However, if the server does have a console, the `stdin` prompt will block the server for user input. This is not a desirable behavior for a server process.

Results

Application get a authenticated JAAS Subject from successful authentication. If the authentication is performed using WebSphere JAAS Login Module, the authenticate Subject can be used by WebSphere security runtime to perform authorization checks on J2EE role-based protected resources.

Usage scenario

Application wants to perform programmatic login and use the authenticated Subject for downstream call.

What to do next

Use the authenticated Subject to access protected resources, if the authenticated Subject granted the required permissions (pure JAAS authorization) or the required roles (J2EE role-base authorization), access is granted, else the access is denied.

Example: JAAS programmatic login

The following example illustrates how application programs may perform a programmatic login using JAAS authentication.

```
LoginContext lc = null;

 try {
       lc = new LoginContext("WSLogin",
                  new WSCallbackHandlerImpl("userName", "realm", "password"));
 } catch (LoginException le) {
        System.out.println("Cannot create LoginContext. " + le.getMessage());
        // insert error processing code
 } catch(SecurityException se) {
        System.out.printlin("Cannot create LoginContext." + se.getMessage();
        // Insert error processing
  }

  try {
         lc.login();
  } catch(LoginExcpetion le) {
         System.out.printlin("Fails to create Subject. " + le.getMessage());
          // Insert error processing code
  }
```

Shown in the example, the new LoginContext is initialized with the WSLogin login configuration and the WSCallbackHandlerImpl CallbackHandler. The WSCallbackHandlerImpl is suitable for use on server side application where prompt is not desirable. A WSCallbackHandlerImpl instance is initialized by the specified user id, password, and the realm information. The present WSLoginModuleImpl class implementation that is specified by WSLogin can only retrieve authentication information form the specified CallbackHandler. A LoginContext may be constructed with a Subject object but the Subject will be disregarded by the present WSLoginModuleImpl implementation. For WebSphere client container applications, replace WSLogin by ClientContainer login configuration which specifies the WSClientLoginModuleImpl implementation that is tailored for client container requirements.

For a pure Java application client, WebSphere provides two other CallbackHandler implementation: WSStdinCallbackHandlerImpl and WSGUICallbackHandlerImpl which prompt user name, password, and realm information on command line and pop up panel, respectively. One may choose either one of the WebSphere CallbackHandler depending on the particular application environment. You may develop a new CallbackHandler for your if none of those implementation fit your particular application requirement.

In addition, you can also develop your own LoginModule if the default WSLoginModuleImpl implementation cannot address all your requirements. WebSphere provides utility functions that can be used by custom LoginModule which are described in the next section.

Developing your own J2C principal mapping module

Before you begin

WebSphere Application Server provides principal mapping when Java 2 Connector (J2C) connection factory is configured to perform container managed sign-on. For example, the applicaiton server can map the caller principal to a resource principal in order to open a new coonection to the backend server. With the container-managed sign-on, WebSphere Application Server creates a Subject instance that contains EIS security domain credentials. A Subject object returned by a principal mapping module contains a Principal object represents the caller identity and a `PasswordCredential` or a `GenericCredential`. WebSphere Application Server provides a default principal mapping module that maps any authenticated user credentials to password credentials for the EIS security domain. The default mapping module is defined in the Application Login Configuration panel in the `DefaultPrincipalMapping` entry. The user ID and password for the EIS security domain is defined under each connection factory by an `authDataAlias` attribute *container-managed authentication alias* in the administrative console. The `authDataAlias` attribute does not actually contain the user name and password. An `authDataAlias` attribute contains an alias that refers to a user name and password pair that is defined in the security configuration document. Since it contains sensitive data, the security configurtion document requires the most privileged **administrator** role for both read and write access. This indirection avoids saving sensitive user name and password in configuration documents other than the security document.

The J2C Connection Factory configuration contains a mapping module which defines a principal mapping module alias (`mappingConfigAlias` attribute) and an authentication data alias (`authDataAlias` attribute). At runtime the J2C managed connection factory code passes a reference of the `ManagedConnectionFactory` and an `authDataAlias` object to the configured principal mapping module via the `WSPrincipalMappingCallbackHandler` object. WebSphere Application Server allows users to plug-in a custom principal mapping module for a connection factory if the any-authenticated-to-one mapping provided by the default principal mapping module is insufficient. A custom mapping module is a special purpose JAAS LoginModule that perform principal or credential mapping in the login method. The `WSSubject.getCallerPrincipal()` method can be used to retrieve the application client identity. Plugging in a custom mapping module is very simple. Change the value of the `mappingConfigAlias` to the custom mapping module. However, the configuration cannot be done via the administrative console and must be done through the wsadmin scripting tool.

The following steps are needed to perform this task. The first few steps can be performed using the administrative console. The remaining configuration needs to be performed using wsadmin scriptings.

Steps for this task

1. Start the administrative console. To add a custom mapping module for an application server, click **Servers** > **Application Servers**. Click the particular server on the right navigation panel.
2. Click **Security** > **JAAS Configuration**.
3. Select **JAAS Configuration** and **Application Login**. Click **New**.
4. Enter a unique alias for the new mapping module, and click **Apply**.
5. Click **JAAS Login Modules** to define the custom mapping module class.
6. Click **New**, and complete mapping LoginModule class name.

7. Click **Apply**. Click **Save** to save the new configuration.
8. Configure J2C Connection Factory to use the new mapping module using `wsadmin`.
   a. At the wsadmin prompt, type the following command to show a list of `J2CConnectionFactory` objects: `wsadmin>$AdminConfig list J2CConnectionFactory`.
   b. Select the J2C Connection Factory, type in the following command to show all the attributes. For example,

      ```
      wsadmin>$AdminConfig show PetStore_CF(cells/hillsideNetwork/nodes/hillside/servers
      /server1: resources.xml#CMPConnectorFactory_4)
      ```

      .
   c. Type the following command to examine the current mapping module configuration:

      (Lines in the following example have been split for publication.)

      ```
      wsadmin>$AdminConfig show {mapping (cells/hillsideNetwork/nodes/
      hillside/servers/server1:
      resources.xml#MappingModule_7)}
      ```

      The following shows sample results of the above command: `{authDataAlias {}} {mappingConfigAlias DefaultPrincipalMapping}`. As shown in the previous example, the J2C Connection factory is configured to use the DefaultPrincipalMapping login configuration.
   d. Type the following command to modify the mapping module configuration to use the new mapping module:

      (Lines in the following example have been split for publication.)

      ```
      wsadmin>$AdminConfig modify {mapping (cells/hillsideNetwork/nodes/
      hillside/servers/server1:
      resources.xml#MappingModule_7)} { {mappingConfigAlias myMappingModule}}
      ```

   You may check the result by typing: (Lines in the following example have been split for publication.)

   ```
   wsadmin>$AdminConfig show {mapping (cells/hillsideNetwork/nodes/
   hillside/servers/server1:resources.xml#MappingModule_7)}
   {authDataAlias {}} {mappingConfigAlias myMappingModule}
   ```

   Note: The authDataAlias is left undefined. In practice, the authDataAlias will be passed at runtime to the custom mapping module. But using the authDataAlias to look up user id and password requires the WebSphere Common Configuration Model (WCCM) programming interface which is not available at this time.
9. Enter save to save your changes.
   a. Enter the following command: `wsadmin>save`.

Results

A mapping module is defined and is configured for the specified J2C Connection factory.

Usage scenario

This task allows you to use your own mapping module to fit your application environment. The WebSphere Application Server default principal mapping module maps all authenticated user credentials to the same user id and password credentials of the EIS security domain. The user ID and password are stored in the security configuration document and is looked up using the configured alias as a key. Your mapping module may be programmed to perform more sophisticated mapping and store passwords in other persistent storage or from a remote service.

What to do next

To develop ones own principal and credential mapping LoginModule, please refer to the JAAS documentation for general information. The JAAS documents are shipped with WebSphere, it is located in ${install_root}/web/docs/jaas/JaasDocs.zip, please refer to the login.html in the JaasDocs.zip zip file for details of how to develop JAAS login module.

In particular, a mapping module needs to obtain the security identity of the caller. The `WSSubject.getCallerPrincipal()` static method returns an `com.ibm.websphere.security.auth.WSPrincipal` object which represents the security identity of an authenticated caller.

Developing custom user registries

Before you begin

WebSphere Application Server security supports the use of custom registries in addition to LocalOS and LDAP registries for authentication and authorization purposes. A Custom User Registry is a customer implemented user registry which implements the UserRegistry Java interface as provided by WebSphere Application Server. A custom implemented user registry can support virtually any type or notion of an accounts repository from a relational database, flat file, etc. The Custom User Registry provides considerable flexibility in adapting WebSphere Application Server security to various environments where some notion of a user registry, other than LDAP or LocalOS, already exist in the operational environment.

Implementing a Custom User Registry is a software development effort. One must use the methods defined in the UserRegistry interface to make calls to the desired registry to obtain user and group information. The interface defines a very general set of methods, so it can be used to encapsulate a wide variety of registries. A custom user registry can be configured as the active User Registry when configuring WebSphere Application Server global security.

For backward compatibility, the WebSphere Application Server Version 4.0 custom registry is also supported. Refer to the Migrating custom user registries article for more information on migrating. However, it is recommended that you use the new interface to implement your custom registry.

Note: If the custom registry implementation is using data sources to connect to a database you need to set the property `WAS_UseDataSource` to **true** in the custom registry properties in the Network Deployment environment. This is required because the node agent process does not contain datasources and hence has to use the remote registry from the cell process. In this case, if the cell process goes down for any reason, the node agent process(es) should be restarted after the cell process is started back up.

Steps for this task

1. If not familiar with the Custom User Registry concept, refer to the article, (Custom user registries).

This section explains each of the methods in the interface in detail and the changes for these methods from the 4.0 release.

2. Implement all the methods in the interface except for the `CreateCredential` method which is implemented by WebSphere Application Server.

   A simple sample that implements this interface is provided for reference.
3. Build your implementation.

   You need the `%WAS_HOME%/lib/sas.jar` and `%WAS_HOME%/lib/wssec.jar` files in your classpath. For example: `%WAS_HOME%\java\bin\javac -classpath %WAS_HOME%\lib\wssec.jar;%WAS_HOME%\lib\sas.jar yourImplementationFile.java`.
4. Copy the class files generated in the above step to the product classpath.

   The preferred location is %WAS_HOME%/classes directory. This should be copied to all the product processes (cell, all NodeAgents) classpath.
5. Follow the steps in (Configuring custom user registries) to configure your implementation using the administrative console.

   Note: If data sources are being used in your custom registry, you need set the property `WAS_UseDataSource` to **true**.

Usage scenario

This step is required to implement custom user registries in Version 5.

What to do next

If you enabling security, make sure you complete the remaining steps. Once this is done, make sure you save and sync the configuration and restart all the servers. Try accessing some J2EE resources to make sure that the custom registry implementation is successful.

Example: Custom user registries

A Custom User Registry is a customer implemented user registry which implements the `UserRegistry` Java interface as provided by WebSphere Application Server. A custom implemented user registry can support virtually any type or notion of an accounts repository from a relational database, flat file, etc. The custom user registry provides considerable flexibility in adapting WebSphere Application Server security to various environments where some notion of a user registry, other than LDAP or LocalOS, already exist in the operational environment.

Implementing a Custom User Registry is a software development effort. One must use the methods defined in the UserRegistry interface to make calls to the desired registry to obtain user and group information. The interface defines a very general set of methods, so it can be used to encapsulate a wide variety of registries. A Custom User Registry can be configured as the active User Registry when configuring the product global security.

If you are using the WebSphere Application Server Version 4.0 custom registry you should be able to plugin your registry without any changes. However, it is recommended that you use the new interface to implement your custom registry.

To view a sample custom registry, refer to the following articles:

- (FileRegistrySample.java files)
- (Users.prop file)
- (Groups.prop file)

UserRegistry interface methods

Implementing this interface enables WebSphere Application Server security to use custom registries. This capability should extend the`java.rmi` file. With a remote registry, you can complete this process remotely.

Implementation of this interface must provide implementations for:

- initialize(java.util.Properties)
- checkPassword(String,String)
- mapCertificate(X509Certificate[])
- getRealm
- getUsers(String,int)
- getUserDisplayName(String)
- getUniqueUserId(String)
- getUserSecurityName(String)
- isValidUser(String)
- getGroups(String,int)
- getGroupDisplayName(String)
- getUniqueGroupId(String)
- getUniqueGroupIds(String)
- getGroupSecurityName(String)
- isValidGroup(String)
- getGroupsForUser(String)
- getUsersForGroup(String,int)
- createCredential(String)

```
public void initialize(java.util.Properties props)
      throws CustomRegistryException,
             RemoteException;
```

This method is called to initialize the UserRegistry. All the properties defined in the Custom User Registry panel propagate to this method.

For the sample the initialize method retrieves the names of the registry files containing the user and group information.

This method is called during server bringup to initialize the registry. This method is also called when validation is performed by the administrative console when security is on. This method remains the same as in Version 4.0.

```
public String checkPassword(String userSecurityName, String password)
      throws PasswordCheckFailedException,
             CustomRegistryException,
             RemoteException;
```

The checkPassword method is called to authenticate users when they log in using a name (or ID) and a password. This method returns a string which, in most cases is the user being authenticated. A credential is then created for the user for authorization purposes. This user name is also returned for the EJB call getCallerPrincipal() and the servlet calls getUserPrincipal() and getRemoteUser(). See also the getUserDisplayName method for more information if you have display names in your registry. In some situations if you return a user other than the one who has logged in, ensure that the user is valid in the registry.

For the sample, the mapCertificate method gets the distinguished name (DN) from the certificate chain and makes sure it is a valid user in the registry before returning the user. For the sample, the checkPassword method simply checks the name and password combination in the registry, and if they match, returns the user who is being authenticated.

This method is called for various scenarios. It is called by the administrative console to validate the user information once the registry is initialized. It is also called when you access protected resources in the product for authenticating the user and before proceeding with the authorization. This method is the same as in Version 4.0.

```
public String mapCertificate(X509Certificate[] cert)
      throws CertificateMapNotSupportedException,
             CertificateMapFailedException,
             CustomRegistryException,
             RemoteException;
```

The mapCertificate method is called to obtain a user name from a X509 certificate chain supplied by the browser. The complete certificate chain is passed to this method and the implementation can validate the chain if needed and get the user information. A credential is created for this user for authorization purposes. If browser certificates are not supported in your configuration you can throw the exception, CertificateMapNotSupportedException. The consequence of not supporting certificates is that the authentication will fail if the challenge type is certificates, even if they have valid certificates in the browser.

This method is called when certificates are provided for authentication. For Web applications when the auth-constraints are set to CLIENT-CERT in the `web.xml` of the application this method is called to map a certificate to a valid user in the registry. For Java clients, this method is called to map the client certificates in the transport layer, when using the transport layer authentication. Also, when the Identity Assertion Token (when using the CSIv2 authentication protocol) is set to contain certificates, this method is called to map the certificates to a valid user.

In Version 4.0, the input parameter was the X509Certificate certificate itself. In Version 5, this parameter has been changed to accept an array of X509Certificate certificates (for example, certificate chain). Also, note that unlike in Version 4.0 (where this parameter was called only for Web Applications), you can call this method for both Web and Java clients in Version 5.

```
public String getRealm()
      throws CustomRegistryException,
             RemoteException;
```

The getRealm method is called to get the name of the security realm. The name of the realm identifies the security domain for which the registry authenticates users. If this method returns a null value, a default name of customRealm is used.

For the sample, the getRealm method returns the string customRealm. One of the calls to this method is when the registry information is validated. This method is same as in Version 4.0.

```
public Result getUsers(String pattern, int limit)
      throws CustomRegistryException,
             RemoteException;
```

The getUsers method returns the list of users from the registry. The names of users depend on the pattern parameter. The number of users are limited by the limit parameter. In a registry that has many users, getting all the users is not practical. So the limit parameter is introduced to limit the number of users retrieved from the registry. A limit of 0 indicates to return all the users that match the pattern and can cause problems for large registries. Use this limit with care. The Custom Registry implementations are expected to support at least the wildcard search *. For example, a pattern of (*) returns all the users and a pattern of (b*) returns the users starting with *b*.

The return parameter is an object of type com.ibm.websphere.security.Result. This object contains two attributes, a java.util.List and a java.lang.boolean. The list should contain the list of users returned and the boolean flag indicates if there are more users available in the registry for the searche pattern. This boolean flag is used to indicate to the client whether more users are available in the registry.

In the sample, the getUsers retrieves the required number of users from the registry and sets them as a list in the Result object. To find out if there are more users than requested the sample gets one more user than requested and if it finds the additional user it sets the boolean flag to true. For pattern matching the match method in the RegExpSample class is used, which supports wildcards like *, ?.

This method is called by administrative console to add users to roles in the various map users to roles panels. The administrative console uses the boolean set in the Result object to indicate that more entries matching the pattern are available in the registry.

In Version 4.0 this method accepted only the pattern parameter. The return was a list. In Version 5, this method is changed to take one additional parameter, the limit. Ideally, your implementation should change to take the limit value and limit the users returned. The return is changed to return a Result object which consists of the list (as in Version 4.0) and a flag indicating if more entries exist. So as in Version 4.0 when the list returns, use the Result.setList(List) to set the List in the Result object. If there are more entries than requested in the Limit parameter, set the boolean attribute to `true` in the Result object using Result.setHasMore(). The default for the boolean attribute in the Result object is `false`.

```
public String getUserDisplayName(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

The getUserDisplayName method returns a display name for a user, if one exists. The display name is an optional string that describes the user that you can set in some registries. This is a descriptive name for the user and need not be unique in the registry. For example in Windows systems, you can display the full name of the user. If you do not need display names in your registry, return null or an empty string for this method.

Note: In WebSphere Application Server Version 4.0, if display names existed for any user they were useful for the EJB method call getCallerPrincipal() and the servlet calls getUserPrincipal() and getRemoteUser(). So if the display names were not the same as the security name for any user, the display names are returned for the above mentioned enterprise beans and servlet methods. Returning display names for these methods might become problematic is some situations since the display names might not be unique in the registry. Avoid this problem by changing the default behavior to return the user's security name instead of the user's display name in this version of the product. However, if you want to have the same behavior as in Version 4.0, set the property WAS_UseDisplayName to `true` in the Custom Registry properties in the administrative console. For more information on how to set properties for the Custom Registry see the section on Setting Properties for Custom Registries.

In the sample, this method returns the display name of the user whose name matches the user name provided. If the display name does not exist this returns an empty string.

This method can be called by the product to present the display names in the administrative console or through the command line using the wsadmin tool. Use this method only for displaying. This method is the same as in Version 4.0.

```
public String getUniqueUserId(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the uniqueId of the user given the security name.

In the sample, this method returns the uniqueId of the user whose name matches the supplied name. This method is called when forming a credential for a user and also when creating the authorization table for the application.

```
public String getUserSecurityName(String uniqueUserId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the security name of a user given the uniqueId. In the sample, this method returns the security name of the user whose uniqueId matches the supplied ID.

This method is called to make sure a valid user exists for a given uniqueUserId. One of the situations this method is called is to get the security name of the user is when the uniqueUserId is obtained from a token. This method is the same as in Version 4.0.

```
public boolean isValidUser(String userSecurityName)
      throws CustomRegistryException,
             RemoteException;
```

This method indicates whether the given user is a valid user in the registry.

In the sample, this method returns `true` if the user is found in the registry, otherwise this method returns `false`. This method is primarily called in situations where knowing if the user exists in the directory or not would prevent problems later. For example, in the mapCertificate call, once the name is obtained from the certificate if the user is found to be an invalid user in the registry, you can avoid trying to create the credential for the user. This method is the same as in 4.0.

```
 public Result getGroups(String pattern, int limit)
      throws CustomRegistryException,
             RemoteException;
```

The getGroups method returns the list of groups from the registry. The names of groups depend on the pattern parameter. The number of groups is limited by the limit parameter. In a registry that has many groups, getting all the groups is not practical. So the limit parameter is introduced to limit the number of groups retrieved from the registry. A limit of 0 implies to return all the groups that match the pattern and can cause problems for large registries. Use this limit with care. The custom registry implementations are expected to support at least the wildcard search (*). For example, a pattern of (*) returns all the users and a pattern of (b*) returns the users starting with *b*.

The return parameter is an object of type com.ibm.websphere.security.Result. This object contains two attributes, a java.util.List and a java.lang.boolean. The list contains the list of groups returned and the boolean flag indicating whether there are more groups available in the registry for the pattern searched. This boolean flag is used to indicate to the client if more groups are available in the registry.

In the sample, the getUsers retrieves the required number of groups from the registry and sets them as a list in the Result object. To find out if there are more groups than requested, the sample gets one more user than requested and if it finds the additional user, it sets the boolean flag to `true`. For pattern matching, the match method in the RegExpSample class is used. It supports wildcards like *, ?.

This method is called by the administrative console to add groups to roles in the various map groups to roles panels. The administrative console will use the boolean set in the Result object to indicate that more entries matching the pattern are available in the registry.

In Version 4.0, this method used to take only the pattern parameter. The return was a list. In Version 5, this method is changed to take one additional parameter, the limit. Ideally, your implementation should change to take the limit value and limit the users returned. The return is changed to return a Result object which consists of the list (as in Version 4.0) and a flag indicating whether more entries exist. So as in Version 4.0 when the list is returned, use the Result.setList(List) to set the list in the Result object. If there are more entries than requested in the limit parameter set the boolean attribute to `true` in the Result object using Result.setHasMore(). The default for the boolean attribute in the Result object is `false`.

```
public String getGroupDisplayName(String groupSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

The getGroupDisplayName method returns a display name for a group if one exists. The display name is an optional string describing the group that you can set in some registries. This name is a descriptive name for the group and need not be unique in the registry. If you do not need to have display names for groups in your registry return null or an empty string for this method.

In the sample, this method returns the display name of the group whose name matches the group name provided. If the display name does not exist this method returns an empty string.

The product can call this method to present the display names in the administrative console or through command line using the wsadmin tool. This method is only used for displaying. This method is the same as in Version 4.0.

```
public String getUniqueGroupId(String groupSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the uniqueId of the group given the security name.

In the sample, this method returns the uniqueId of the group whose name matches the supplied name. This method is called when creating the authorization table for the application. This method is the same as in Version 4.0.

```
public List getUniqueGroupIds(String uniqueUserId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the uniqueIds of all the groups to which a user whose uniqueId matches the supplied Id belongs.

In the sample, this method returns the uniqueId of all the groups that contain this uniqueUserId. This method is called when creating the credential for the user. As part of creating the credential, all the groupUniqueIds that this user is apart of are collected and put in the credential for authorization purposes when groups are given access to a resource. This method is the same as in Version 4.0.

```
public String getGroupSecurityName(String uniqueGroupId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the security name of a group given its uniqueId.

In the sample, this method returns the security name of the group whose uniqueId matches the supplied Id. This method is called to make sure a valid group exists for a given uniqueGroupId. This method is the same as in Version 4.0.

```
public boolean isValidGroup(String groupSecurityName)
      throws CustomRegistryException,
             RemoteException;
```

This method indicates if the given group is a valid group in the registry.

In the sample, this method returns `true` if the group is found in the registry, otherwise the method returns `false`. This method can be used in situations where knowing whether the group exists in the directory would help prevent problems later. This method is the same as in Version 4.0.

```
public List getGroupsForUser(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns all the groups to which a user whose name matches the supplied name belongs. This method is similar to the getUniqueGroupIds method with the except that the securityNames are used, instead of the uniqueIds.

In the sample, this method returns all the group securityNames that contain the userSecurityName.

This method is called by the administrative console or the scripting tool to verify that the users entered for the RunAs roles are already part of that role in the users and groups to role mapping. This check is required to ensure that a user cannot be added to a RunAs role unless that user is assigned to the role in the users and groups to role mapping either directly or indirectly (through a group which contains this user). Since a group that this user belongs to can be part of the role in the users and groups to role mapping, this method is called to check if any of the groups that this user belongs to is mapped to that role. This method is the same as in Version 4.0.

```
public Result getUsersForGroup(String groupSecurityName, int limit)
      throws NotImplementedException,
             EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method is not called by WebSphere Application Server for either authenticating or authorization purposes. This method, however, can be used by some product clients, like Workflow, where this information is used. The documentation in those products in the WebSphere product line explains the usage of this method in detail. This method is deprecated.

This method retrieves users from the specified group. The number of users returned is limited by the limit parameter. A limit of 0 indicates to return all the users in that group that can cause problems for some large groups. Also, this method might not be practical for implementation for registries that have many users in a group. In these cases you can throw the NotImplementedException.The return parameter is an object of type com.ibm.websphere.security.Result. This object contains two attributes, a java.util.List and a java.lang.boolean. The list contains the list of users returned and the boolean flag indicates whether there are more users available in the registry for the search pattern. This boolean flag is used to indicate to the client whether more users are available in the registry.

In the sample, this method gets one more user than requested if the limit parameter is not set to 0. If it succeeds in getting one more user it sets the boolean flag to `true`. As an example, if the limit exceeds a certain number (50 in this case) it throws the NotImplementedException.

This method is NOT used by product. Rather, other WebSphere Application Server clients like Workflow use this method.

In Version 4.0, this method was mandatory for the product. For Version 5, this method is optional and can throw the NotImplementedException if you choose not to implement it. In Version 4.0, this method accepted only the pattern parameter, and the return was a list. In Version 5, this method accepts one additional parameter, the limit. Ideally, your implementation should change to take the limit value and limit the users returned. The return changes to return a Result object which consists of the list (as in Version 4.0) and a flag indicating whether more entries exist. So as in Version 4.0, when the list is returned use the Result.setList(List) to set the list in the Result object. If there are more entries than requested in the limit parameter, set the boolean attribute to `true` in the Result object using Result.setHasMore(). The default for the boolean attribute in the Result object is `false`.

```
 public com.ibm.websphere.security.cred.WSCredential
createCredential(String userSecurityName)
```

```
throws NotImplementedException,
       EntryNotFoundException,
       CustomRegistryException,
       RemoteException;
```

In this release of the WebSphere Application Server, the custom registry should NOT implement this method. Throw the NotImplementedException exception instead.

In the sample, the NotImplementedException exception is thrown. This method did not exist in Version 4.0. In Version 5, your implementation throws the NotImplementedException exception.

Developing a custom interceptor for trust associations

Before you begin

If you are using a third-part reverse proxy server other than Tivoli WebSeal Version 3.6, you must provide an implementation class for the WebSphere interceptor interface for your proxy server. This file describes the interface you must implement.

Steps for this task

1. Define the interceptor class method.

   WebSphere Application Server provides the interceptor Java interface, com.ibm.websphere.security.TrustAssociationInterceptor, which defines the following methods:

   - `public boolean isTargetInterceptor(HttpServletRequest req) throws WebTrustAssociationException;`

     The isTargetInterceptor method is used to determine whether the request originated with the proxy server associated with the interceptor. The implementation code must examine the incoming request object and determine if the proxy server forwarding the request is a valid proxy server for this interceptor. The result of this method determines whether the interceptor processes the request or not.
   - `public void validateEstablishedTrust(HttpServletRequest req) throws WebTrustAssociationException;`

     The validateEstablishedTrust method determines if the proxy server from which the request originated is trusted or not. This method is called after the isTargetInterceptor method. The implementation code must authenticate the proxy server. The authentication mechanism is proxy-server-specific. For example, in the WebSphere-provided implementation for the WebSeal server, this method retrieves the basic-authentication information from the HTTP header and validates the information against the user registry used by WebSphere Application Server. If the credentials are invalid, the code throws the WebTrustAssociationException exception, indicating that the proxy server is not trusted and the request is to be denied.
   - `public String getAuthenticatedUsername(HttpServletRequest req) throws WebTrustAssociationException;`

     The `getAuthenticatedUsername` method is called after trust has been established between the proxy server and WebSphere Application Server. WebSphere Application Server has accepted the proxy server's authentication of the request and must now authorize the request. To authorize the request, the name of the original requestor must be subjected to an authorization policy to determine if the requestor has the necessary privilege. The implementation code for this method must extract the user name from the HTTP request header and determine if that user is entitled to the requested resource. For example, in the WebSphere-provided implementation for the WebSeal server, the method looks for an iv-user attribute in the HTTP request header and extracts the user ID associated with it for authorization.

Interceptor class methods have been defined.

2. Configure WebSphere Application Server by setting properties in the `trustedservers.properties` file. This procedure is described for the WebSeal interceptor in ″Example: Trust association interceptor″, and the procedure described there varies as follows:
   a. Establish a name for your proxy to use in the WebSphere Application Server configuration properties. Use this name when you set the property com.ibm.websphere.security.trustassociation.types. For example, if you call your proxy myProxy, then set the property as follows: `com.ibm.websphere.security.trustassociation.types=myproxy`.
   b. Based on the name you specified as the type of the proxy, WebSphere Application Server looks for a property that names the implementation class. Set the value of this property to the name of your implementation class. The implementation class must be locatable from the information on the class path.

      The name of the property is based on the name you assigned to your proxy according to this pattern: com.ibm.websphere.trustassociation..interceptor.

      For example, for the proxy called `myProxy`, the property name is com.ibm.websphere.trustassociation.myproxy.interceptor, and for the proxy type webseal36, the property name is com.ibm.websphere.trustassociation.webseal36.interceptor.

The WebSphere Application Server has been configured.

Usage scenario

Refer to the WebSphere Application Server Version 5 Security Redbook for an example of a custom interceptor for JAAS.

Making your custom interceptor configurable

To allow configuration of your custom interceptor by reading a configuration file, you can subclass the WebSphere-provided class com.ibm.websphere.security.WebSphereBaseTrustAssociationInterceptor and provide implementations of the following methods:

- `abstract public int init(String propsfile);`

  The `init` method reads the configuration file specified for the interceptor. The configuration file is specified in the `trustedservers.properties` file by using a property, the name of which is determined by this pattern: `com.ibm.websphere.trustassociation.`*`<proxyname>`*`.config`. For example, the property name for the proxy called `myProxy`, is `com.ibm.websphere.trustassociation.myproxy.config`, and for the proxy type `webseal36`, the property name is `com.ibm.websphere.trustassociation.webseal36.config`. The value of the property is the name of the configuration file for the interceptor.
- `abstract public void cleanup();`

  The cleanup method does any necessary termination work for the interceptor.

Example: Trust association interceptor

If you are using a third party reverse proxy server other than Tivoli WebSeal product, you must provide an implementation class for the trust association interceptor interface for your proxy server.

Using the TrustAssociation Interceptor interface

WebSphere Application Server (WAS) provides the interceptor Java interface, com.ibm.websphere.security.TrustAssociationInterceptor, which defines the following methods:

- **`public boolean isTargetInterceptor`** (`HttpServletRequest req`) throws `WebTrustAssociationException;`
- **`public void validateEstablishedTrust`** (`HttpServletRequest req`) throws `WebTrustAssociationException;`
- **`public string getAuthenticatedUsername`** (`HttpServletRequest req`) throws `WebTrustAssociationException;`

The isTargetInterceptor method is used to determine whether the request originated with the proxy server associated with the Interceptor. The implementation code must examine the incoming request object and determine if the proxy server forwarding the request is a valid proxy server for this interceptor. The result of this method determines whether the interceptor processes the request or not.

The validateEstablishedTrust method determines if the proxy server from which the request originated is trusted or not. This method is called after the isTargetInterceptor method. The implementation code must authenticate the proxy server. The authentication mechanism is proxy server specific. For example, in the WebSphere provided implementation for the WebSeal server, this method retrieves the basic authentication from the HTTP header and validates the information against the user registry used by WAS. If the credentials are invalid then the code throws the WebTrustAssociationException exception, indicating that the proxy server is not trusted and the request is to be denied.

The getAuthenticatedUsername method is called after trust has been established between the proxy server and WAS. WAS has accepted the proxy server's authentication of the request and must now authorize the request. To authorize the request, the name of the original requestor must be subjected to an authorization policy to determine if the requestor has the necessary privilege. The implementation code for this method must extract the user name from the HTTP request header and determine if that user is entitled to the requested resource. For example, in the WebSphere provided implementation for the WebSeal server, the method looks for an iv-user attribute in the HTTP request header and extracts the user ID associated with it for authorization.

After the interceptor class has been created, WAS must be configured to use it by providing the following properties. Use the (Security > LTPA panel of the administrative console) to provide these properties.

In the LTPA panel click on Trust Association link. This will take you to the ″Trust Association″ panel.

1. Check mark the Enable Trust Association Box Then click on ″Interceptors″ link .
2. The following interceptor is the default value. If you are using WebSeal Interceptor then check mark the following com.ibm.ws.security.web.WebSealTrustAssociationInterceptor.

3. Otherwise click on New button to add the Interceptors. WAS looks for the implementation class using the classpath. Then for each Interceptor click on Additional Properties to enter the property name and value pairs. Please check mark the box labeled ″Required″ for each property.

The properties are as follows:

- Establish a name for your proxy. Use this name when you set the property `com.ibm.websphere.security .trustassociation.types`. For example, if you call your proxy server myProxy, then set the property as follows:com.ibm.websphere.security.trustassociation.types=myproxy.
- Value for com.ibm.websphere.security.<proxyname>.hostnames is the hostname of the machine where <proxyname> server is running.
- Value for com.ibm.websphere.security.<proxyname>.loginId is the <proxyname> server ID.
- Value for com.ibm.websphere.security.<proxyname>.Id is a special header field that will be sent by <proxyname> server with the request to WebSphere.
- Value for com.ibm.websphere.security.<proxyname>.ports is the port where <proxyname> server will receive the user requests. You can have a different port number depending on your <proxyname> server's configuration.

The name and value pairs for WebSeal are as follows:

- Value for **com.ibm.websphere.security.trustassociation.types** is Webseal
- Value for **com.ibm.websphere.security.webseal.hostnames** is hostname of the machine where WebSeal server is running.
- Value for **com.ibm.websphere.security.webseal.loginId** is the WebSeal server id
- Value for **com.ibm.websphere.security.webseal.id** is iv-user This is a special header field that will be sent by WebSeal with the request to WebSphere.
- Value for **com.ibm.websphere.security.webseal.ports** is 443. This is the port where WebSeal will receive the user requests. You can have a different port number depending on your WebSeal configuration.

Assembling secured applications

The Application Assembly Tool (AAT) is a graphical user interface for assembling enterprise (J2EE) applications. You can use this tool to assemble an application and secure EJB and Web modules in that application. An EJB module consists of one or more beans. You can enforce security at the EJB method level. A Web module consists of one or more Web resources (an HTML page, a JSP file or the Servlet). You can also enforce security for each Web resource. You can use the AAT to secure an EJB module (`.jar` file) or a Web module (`.war` file) or an application (`.ear` file). You can create an application, an EJB module or a Web Module using development tools like the IBM WebSphere Studio Application Developer, for example.

Steps for this task

1. Secure EJB applications.
2. Secure Web applications.
3. Add users/groups to roles while assembling secured application components.
4. Map users to RunAs roles while assembling secured application components.
5. Add the was.policy file to applications for Java 2 Security.
6. **(Optional)** Assemble the application components that you just secured. Refer to the WebSphere Application Server Applications documentation.

7. Specify method permissions and security roles for the application.
8. Save the application (EAR file) that you have just assembled.

Results

After securing an application, the resulting `.ear` file contains security information in its deployment descriptor. The EJB modules security information is stored in `ejb-jar.xml` file and Web Modules security information is stored in `web.xml` file. The `application.xml` file of the application EAR contains all the roles used in the application. The user and group to roles mapping is stored in `ibm-application-bnd.xmi` file of the application EAR. The `was.policy` file of the application EAR contains the permissions granted for the application to access system resources.

Usage scenario

This step is required to secure EJB modules and Web modules in an application. This step is also required for applications to run properly when Java 2 security is enabled. If was.policy file is not created and it does not contain required permissions, the application might not be able to access system resources.

What to do next

After securing an application using the AAT, you can install an application using administrative console. When you install a secured application, refer to the Deploying Secured Applications article to complete this task.

EJB component security

An EJB module consists of one or more beans. You can use development tools such as WebSphere Studio Application Developer to develop an EJB module. You can also enforce security at the EJB method level.

You can assign a set of EJB methods to a set of one or more roles. When an EJB method is secured by associating a set of roles, grant at least one role in that set so that you can access that method. To exclude a set of EJB methods from being accessed by anyone mark them **excluded**. You can give everyone access to a set of enterprise beans methods by unchecking those methods. You can run enterprise beans as a different identity (runAs identity) before invoking other enterprise beans.

Securing EJB applications

Before you begin

EJB methods can be protected by assigning security roles to them. So, you need to know what EJB methods needs protecting and how.

Steps for this task

1. Open the EJB application file.

   This file can be an EJB `.jar` file or an application `.ear` file that contains one or more EJB modules. To open the EJB application file click **File** > **Open** and browse. Select the EJB application file.
2. Create security roles.

   You can create security roles at the application level or at the EJB module level. If you create a security role at the EJB module level, the role displays in the application level. If a security role is created at application level, the role will not show up in all the EJB modules. You can copy and paste one or more EJB module security roles that you create at application level.

a. Create a role at application level by right-clicking **Security Roles** under the application folder. Click **New**. Type the role name. If the role created for the application is required for a EJB module, select that role from the application, copy it and right-click the EJB module **Security Roles**, and click **Paste**.

b. To create a role at an EJB module level, open the corresponding EJB module folder. Right-click **Security Roles** under the EJB module and click **New**. Type the role name.

3. Create Method Permissions.

   Method permissions is a mapping of one or more methods to a set of roles. An EJB bean has four types of methods: Home methods, Remote methods, LocalHome methods and Local methods.

   a. To create a new method permission in a EJB module, open the EJB module folder. Right-click **MethodPermissions** and click **New**. A new panel displays.

   b. Type the method permission name and description.

   c. Add methods by clicking **Add** under Methods. Browse and select the required methods. An (*) indicates all methods.

   d. Add the required roles for those methods by clicking **Add** under Roles. Browse and click the required roles. If a set of methods needs to be unprotected, click the checkbox. Click **OK** when done.

4. Exclude methods user access.

   Users cannot access excluded methods. Any method in enterprise beans that is not assigned to a role or is not excluded, is marked deselected during the application installation by the deployer.

   a. Exclude one or more methods by right-clicking **Exclude List** under the EJB module folder. Click **New**. A new panel displays.

   b. Type the description explaining why these methods are excluded.

   c. Add methods to be excluded by clicking **Add**. Browse and click the methods to be excluded. Click **OK** when done.

5. Map security-role-ref and role-name to role-link.

   During development of enterprise beans, you may create the security-role-ref element using development tools such as WebSphere Studio Application Developer. The security-role-ref element contains only the role-name field at this stage. The role-name field determines if the caller is in a specified `role(isCallerInRole()` and contains the name of the role that is referenced in the code. Since you create security roles during the assembly stage, the developer uses a *logical rolename* in the role-name field and provides enough information in the description field for the assembler to map the actual role (role-link). The security-role-ref element is located at the EJB level. Enterprise beans can have zero or more security-role-ref elements.

   a. Open the required EJB folder and click **Security Role References** to map role-name to role-link for a security-role-ref element.

   b. Click each role-name on the right navigation panel and click the role that you intend to map to that role-name by selecting a role from the drop down list of the link.

   c. Right-click **Security Role References** and click **New** if you did not create the security-role-ref element during development. A new panel displays.

d. You can enter the role-name in the **Name** field and the role-link in the **Link** field by selecting a proper role from the drop down list. You can also add a proper description in the **Description** field.

e. Mapp every role-name used during development to the role (role-link) using the previous steps.

6. Specify RunAs Identity for enterprise beans components.

   The RunAs Identity of the enterprise beans is used to invoke the next enterprise beans in the chain of EJB invocations. When the next enterprise beans are invoked, the `RunAsIdentity` is passed to the next enterprise beans for performing an authorization check on the next enterprise beans. If the RunAs Identity is not specified, the client identity is propagated to the next enterprise beans. The RunAs Identity can represent each of the enterprise beans or can represent each method in the enterprise beans.

   a. Set the RunAs Identity for the enterprise beans component by clicking the enterprise beans. Click the security tab in the right navigation.

   b. Select the **Security Identity** selection box.

   c. Click **Run-As mode** from the drop down list.

   d. Click the role name from the dropdown list if **UseSpecifiedId** is selected. Click **Apply** when done.

   e. Set the RunAs Identity at the method level by opening the EJB folder. Click **Method Extensions**.

   f. Select the **Advanced** tab in the right navigation panel.

   g. Select the required methods from the top of the panel and select the **Security Identity** checkbox. Click **Run-As Mode**. Selecting System Identity implies that the invocation is done using the WebSphere Application Server security server ID and should be used with caution as this ID has more privileges.

   h. Click the **Role Name** from the dropdown list if the specified identity is selected.

Results

After securing an EJB application, the resulting `.jar` file contains security information in its deployment descriptor. The EJB modules security information is stored in `ejb-jar.xml` file.

What to do next

After securing an EJB application using assembly tool, you can install the EJB application using the administrative console. During the installation of a secured EJB application, follow the steps in the (Deploying Secured Applications) article to complete the task of securing the EJB application.

Security permissions assembly settings

Specifies the supplementary permissions to include in your EJB module.

Security permissions: A permission represents access to a system resource.

See ″Security: Resources for learning″ for more information on security permissions.

Security settings

Use the Security Center to modify global and default security settings for all applications:

- Global settings apply to existing and future applications and cannot be customized.
- Default settings apply only to future applications and can be customized.

The default settings are used as a template or starting point for configuring individual applications. The administrator should still explicitly configure security settings for each application.

The following are security settings that are specified during application assembly.

- ″Security role assembly settings″
- ″Security constraint assembly settings″ (not in this document)

Security role assembly settings: A security role is a logical grouping of principals. Access to operations (such as EJB methods) is controlled by granting access to a role. In WebSphere Application Server Version 5.0, if a role is added at the EAR application level, it will be removed when you save the archive file, since it is not associated with any module.

Role Name: Specifies the name of a security role that is unique to an application. This setting applies only when you are specifying security roles at the application level (EAR file).

Description: Contains text that describes the application-specific security role. This setting applies only when you are specifying security roles at the application level (EAR file).

Binding -- Groups -- Name: Specifies the user groups that are granted the application-specific security role. This setting applies only when you are specifying security roles at the application level (EAR file).

Binding -- Users -- Name: Specifies the users that are granted the application-specific security role. This setting applies only when you are specifying security roles at the application level (EAR file).

Binding -- Special Subjects -- Name: Specifies one of two special categories of users to which roles can be granted: Everyone or All authenticated users.

Specifies one of two special categories of authenticate users to which application-specific security roles can be granted: **Everyone** or **All**. This setting applies only when you are specifying security roles at the application level (EAR file).

If the special subject **All** is granted a role, any user who can authenticate by using a valid user ID and password is considered to be granted that role.

If the special subject **Everyone** is granted a role, all users, including those who did not authenticate, are granted the role. In other words, a method on Enterprise Beans or a URI is unprotected if any of the required roles for that method are granted to the special subject **Everyone**.

Security role references: Web application developers or EJB providers that use the available programmatic security J2EE APIs, isUserInRole(String roleName) or isCallerInRole(String roleName), use a role-name in the code.

The actual roles used in the deployed runtime environment may not be known until the Web application and EJB components (for example, `.war` files and ejb-jar.jar files) are assembled into an .ear file. Therefore, the role-names used in the web application or EJB component code are ″logical″ role-names which the application assembler maps to the actual runtime environment roles during application assembly. The security role references provide a level of indirection that insulate web application component and EJB developers from having to know the actual roles in the runtime environment.

The definition of the ″logical″ roles and the mapping to the actual runtime environment roles are specified in the security-role-ref element of both the web application and the EJB jar file deployment descriptors, web.xml and ejb-jar.xml respectively. The Application Assembly Tool (AAT) can be used to both define the role-name and map them to the actual runtime roles in the environment with the role-link element.

The following is an example of a security-role-ref from an EJB ejb-jar.xml deployment descriptor.

```
... <enterprise-beans>
... <entity>
ejb-name>AardvarkPayroll</ejb name>
<ejb-class>com.aardvark.payroll.PayrollBeanejb-class
>com.aardvark.payroll.PayrollBean>
...
<security-role-ref>
<description>
This role should be assigned to the employees of the payroll department.
Members of this role have access to the payroll record of everyone.
The role has been linked to the payroll-department role. This role should be
assigned to the employees of the payroll department. Members of this role have
access to the payroll record of everyone. The role has been linked to the
payroll-department role. </description>
role-name>payroll</role-name>
<role-link>payroll-department</role-link>
</security-role-ref>
...
</entity>
...
</enterprise-beans>
```

In the example above, the string *payroll* with appears in the <role-name> element, is what the EJB provider would use as the argument to isCallerInRole() API. The <role-link> element is what ties the logical role to the actual role used in the runtime environment.

Note that for EJBs, the security-role-ref must appear in the deployment descriptor even if the logical role-name is the same as the actual role name in the environment.

The rules for web application components are slightly different. If no security-role-ref element matching a security-role element has been declared, the container must default to checking the role-name element argument against the list of security-role elements for the web application. The isUserInRole method references the list to determine whether the caller is mapped to a security role. The developer must be aware that the use of this default mechanism may limit the flexibility in changing role names in the application without having to recompile the servlet making the call.

See the EJB 2.0 and Servlet 2.3 specification in ″Security: Resources for learning″ for complete details on this specification.

Security role references assembly settings: The Application Assembly Tool (AAT) can be used to both define the role-name and map them to the actual runtime roles in the environment with the role-link element.

Name: Specifies the name of a security role reference used in the application code.

For example, if the name is *boss*, then the AccountBean can make a decision based on whether the user executing a method is granted the role of a *boss*.

Data type	String

Link: Specifies the name of a security role defined in the encompassing application.

The role reference will be linked to this name. For example, the AccountBean code uses a role named *boss*. The Account Bean is a part of an enterprise application, FinanceApp, that has a role named *Manager*. If the link specifies ″Manager,″ then when the bean makes a call to isCallerInRole(″boss″), the result will be true if and only if the user, who invoked the method, has been granted the FinanceApp's Manager role. The security role reference is the name used by an application component (module), and the link name is the name defined in the deployment descriptor of the encompassing application. The link maps the name used in the component to a corresponding name in the application.

Data type	String

Description: Contains text describing the security role.

Data type	String

Web component security

A Web module consists of one or more HTML resources, JSP files and servlets. You can use development tools such as IBM WebSphere Studio Application Developer to develop a Web module and enforce security at the method level of each Web resource.

You can identify a Web resource by URI pattern. A Web resource method can be any HTTP method (GET, POST, DELETE, PUT, for example). A set of URI patterns and a set of HTTP methods can be grouped together and assigned a set of roles. When a Web resource method is secured by associating a set of roles, grant a user at least one role in that set to access that method. You can exclude anyone from accessing a set of Web resources by assigning empty set of roles. A Servlet or a JSP file can be made to run as a different identity (RunAs identity) before invoking another enterprise beans component. All the secured Web resources require the user to login by using configured login mechanism. There are three types of Web login (authentication) mechanisms: basic authentication, form-based authentication and client certificate-based authentication.

For more detailed information on Web security see the (product architectural overview) article.

Securing Web applications

Before you begin

There are three types Web login (authentication) mechanisms that can be configured on a Web application: Basic authentication, Form-based authentication and Client certificate-based authentication. Web resources in a Web application can be protected by assigning security roles to those resources. So, you need to know in advance what Web Resources need protecting and how.

Steps for this task

1. Open the Web application file.

   The application file can be a Web `.war` file or an application `.ear` file that contains one or more Web modules. To open the Web application file click **File** > **Open**. Browse and select the Web application file.
2. Create security roles.

   You can create security roles at the application level or at Web module level. If a security role is created at Web module level, the role also displays in the application level. If a security role is created at the application level, the role does not display in all the Web modules. You can copy and paste a security role at the application level to one or more Web module security roles.
   a. To create a role at the application level, right-click **Security Roles** under the application folder. Click **New**. Type the role name. If the role created for the application is required for a Web module, select that role from the application. Copy the role and select the Web module security roles. Right-click the Web module security role and click Paste.
   b. Create a role at Web module level by opening the corresponding Web module folder. Right-click **Security Roles** under the Web module and click **New**. Type the role name.
3. Create security constraints. Security constraints are a mapping of one or more Web resources to a set of roles.
   a. Create new security constraints by opening the Web module folder and right-click **Security Constraints**. Click **New**. A new panel displays.
   b. Type the security constraints name and description.
   c. Add roles required by clicking **Add** under Roles. Browse and click the required roles. The (*) indicates all roles. An empty role list indicates that no user can have access to the Web Resources specified under these security constraints.
   d. Set user data constraints by clicking **Transport Guarantee** from the drop-down list. A transport guarantee of **NONE** indicates that the communication between the Web Client (Browser) and the Server (Web Server) is transported over HTTP. A transport guarantee of **CONFIDENTIAL** or **INTEGRAL** guarantees that the communication between the Web Client and the Web Server is secured and is transported over HTTP/S.
   e. Click **OK** when done. This creates a new Security Constraints folder for the Web Module.
   f. Open the security constraints created from previous steps and right-click **Web Resources Collection**. Click **New**. A new panel displays.
   g. Type a Web resource collection name and description.
   h. Click **Add** under Methods and select HTTP methods. Click **OK**. If no methods are selected, all methods are selected by default.
   i. Click **Add** under URLs and type the URL pattern (for example: `- /*, *.jsp, /hello`). Consult the servlet specification version 2.3 for specifications of mapping URL patterns to servlets. Security run time uses the exact match first to map the incoming URL with URL patterns. If the exact match is not

present, the security run time uses the longest match. The wild card (`*.,*.jsp`) URL pattern matching is used last.

   j. Click **OK** when done.

   k. Repeat the ten previous steps to create multiple security constraints.

4. Map security-role-ref and role-name to the role-link. During development of Web application, you can create the security-role-ref element using development tools such as WebSphere Studio Application Developer. The security-role-ref element contains only the role-name field at this stage. The role-name field contains the name of the role that is referenced in the servlet or JSP code to determine if the caller is in a specified role (isUserInRole()). Since security roles are created during the assembly stage, the developer uses a *logical rolename* in the role-name field and provides enough description in the description field for the assembler to map the role actual (role-link). The Security-role-ref element is at the servlet level. A servlet or JSP file can have zero or more security-role-ref elements.

   a. Map the role-name to the role-link for a security-role-ref element by opening the required servlet folder. Click **Security Role References**.

   b. Select each role-name on the right navigation panel and click the actual role to be mapped to that role-name. Select a role from the drop-down list of the link.

   c. If the security-role-ref is not created already during development, right-click **Security Role References**. Click **New**. A new panel displays.

   d. Enter the role-name in the **Name** field and role-link in the **Link** field by selecting a proper role from the drop-down list. Add a description in the Description field.

   e. Using the previous steps, every role-name used during development maps to the actual role (role-link).

5. Specify RunAs Identity for servlets and JSP files. The RunAs Identity of a servlet is used to invoke enterprise beans from within the servlet code. When enterprise beans are invoked, the RunAsIdentity is passed to the EJB for performing an authorization check on the enterprise beans. If the RunAs Identity is not specified, the client identity is propagated to the enterprise beans. The RunAs Identity is assigned at the servlet level.

   a. Set the RunAs Identity for a servlet by clicking the servlet folder. Select the **Security** tab on the right navigation panel.

   b. Select the Role name from the drop-down list. Click **Apply** when done.

6. Configure the login mechanism. Configure the login mechanism only at the Web module level. This configured login mechanism applies to all the servlets, JSP files and HTML resources in the Web Module.

   a. Configure the login mechanism for the Web Module by clicking the Web Module folder.

   b. Click the **Advanced** tab on the right navigation panel.

   c. Click the **Login Configuration** checkbox.

   d. Select the required authentication method from the drop-down list.

   e. Type the Login page and Error page URLs if you select form-based authentication (for example: `/login.jsp and /error.jsp`). The specified login and error pages are present in the `.war` file.

   f. Install the client certificate on the browser (Web Client) and place the client certificate in the server trust keyring file, if `ClientCert` is selected.

Results

After securing a Web application, the resulting `.war` file contains security information in its deployment descriptor. The Web module security information is stored in `web.xml` file.

What to do next

After using AAT to secure a Web application, you can install the Web application using the administrative console. During the Web application installation, complete the steps in the Deploying Secured Applications article to finish securing the Web application.

Role-based authorization

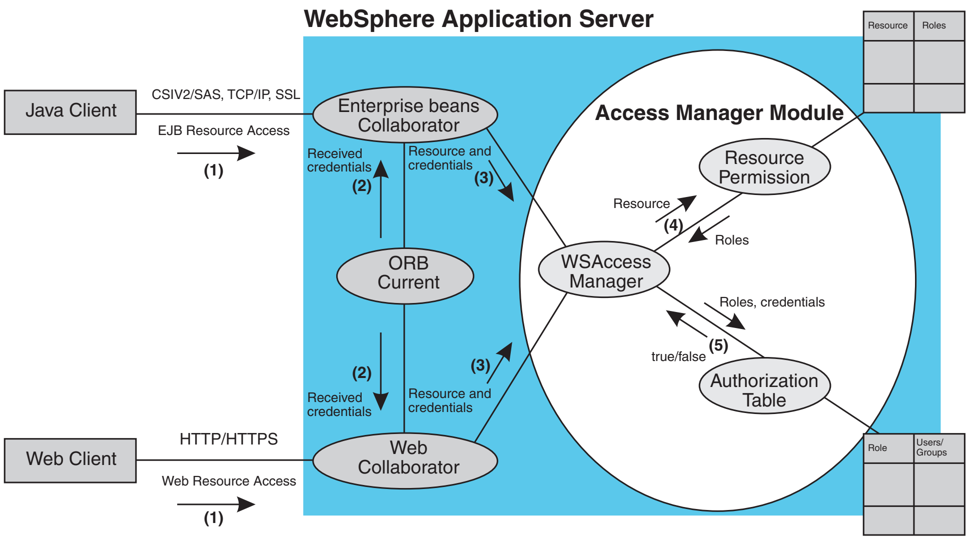
Use authorization information to determine whether a caller has the necessary privilege to request a service.

The following diagram illustrates the flow during an authorization process. Web resource access from a Web Client is handled by a Web Collaborator. The EJB resource access from a Java Client (can be enterprise beans or a Servlet) is handled by an EJB Collaborator. The EJB collaborator and the Web Collaborator extract the client credentials from the *ORB* Current Object. The client credentials are set during the authentication process as received credentials in the ORB Current. The resource and the received credentials are presented to WEAccessManager to check whether access is permitted to the client to access the requested resource.

AccessManager contains two main modules.

- Resource Permission module helps to determine the required roles for a given resource. It uses Resource to Roles mapping table that is built by the security runtime during the application startup. The Resource to Role mapping table is built by reading the deployment descriptor of enterprise beans or Web module(`ejb-jar.xml` or `web.xml`)
- Authorization Table module consults a role to user or group table to determine whether a client is granted one of the required roles. The Role to User or Group mapping table, also known as the *authorization table*, is created by the security run time during the application startup. The authorization table is built from reading the application binding file (`ibm-application-bnd.xmi` file) for the application.

Authorization



Use authorization information to determine whether a caller has the necessary privilege to request a service. You can store authorization information many ways. For example, with each resource, you can store an an *access-control list*, which contains a list of users and user privileges. Another way to store the information is to associate a list of resources and the corresponding privileges with each user. This is called a *capability list*.

WebSphere Application Server uses the Java 2 Enterprise Edition (J2EE) authorization model. In this model, authorization information is organized as follows:

- During the assembly of an application, permission to execute methods is granted to one or more roles. A role is a set of permissions; for example, in a banking application, roles can include Teller, Supervisor, Clerk, and other industry-related positions. The Teller role is associated with permissions to run methods related to managing the money in an account, for example, the withdraw and deposit methods. The Teller role is not granted permission to close accounts; this permission is given to the Supervisor role. The application assembler defines a list of method permissions for each role; this list is stored in the deployment descriptor for the application.

There are two special subjects that are not defined by J2EE but are worth mentioning, AllAuthenticatedUsers, Everyone and a special role. A special subject is a product-defined entity independent of the user registry. It is used to generically represent a class of users or groups in the registry.

- AllAuthenticatedUsers is a special subject that permits all authenticated users to access protected methods. As long as the user can authenticate successfully, the user is permitted access to the protected resource.
- Everyone is a special subject that permits unrestricted access to a protected resource. Users do not have to authenticate to get access; this special subject allows access to protected methods as if the resources are unprotected.

During the deployment of an application, real users or groups of users are assigned to the roles. The application deployer does not need to understand the individual methods. By assigning roles to methods, the application assembler simplifies the job of the application deployer; instead of working with a set of methods, the deployer works with the roles, which represent semantic groupings of the methods. When a user is assigned to a role, the user gets all the method permissions that are granted to that role. Users can be assigned to more than one role; the permissions granted to the user are the union of the permissions granted to each role. Additionally, if the authentication mechanism supports the grouping of users, these groups can be assigned to roles. Assigning a group to a role has the same effect as assigning each individual user to the role.

A best practice during deployment is to assign groups, rather than individual users to roles for the following reasons:

- Improves performance during the authorization check. There are typically far fewer groups than users
- Provides greater flexibility, by using group membership to control resource access
- Allows the addition and deletion of users from groups outside of the product environment. This action is preferred to adding and removing them to WebSphere Application Server roles. Stop and restart the enterprise application for these changes to take effect. This action can be very disruptive in a production environment

At execution time, WebSphere Application Server authorizes incoming requests based on the user's identification information and the mapping of the user to roles. If the user belongs to any role that has permission to execute a method, the request is authorized. If the user does not belong to any role that has permission, the request is denied.

The J2EE approach represents a declarative approach to authorization, but it also recognizes that you cannot deal with all siturations declaratively. For these situations, methods are provided for determining user and role information programmatically. For Enterprise JavaBeans, the following two methods are supported by WebSphere Application Server:

- **getCallerPrincipal**: This method retrieves the user identification information.
- **isCallerInRole**: This method checks the user identification information against a specific role.

For servlets, the following methods are supported by WebSphere Application Server:

- getRemoteUser
- isUserInRole
- getUserPrincipal

These methods correspond in purpose to the enterprise bean methods.

For more information on the J2EE security authorization model see the Security: Resources for Learning article.

Admin roles

The J2EE role-based authorization concept has been extended to protect the WebSphere Administrative subsystem. A number of administrative roles have been defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the web based admin console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following table describes the Admin roles:

Role	Description
monitor	Least privileged that basically allows a user to view the WebSphere Application Server configuration and current state.
configuration	Monitor privilege plus the ability to change the WebSphere Application Server configuration.
operator	Monitor privilege plus the ability to change runtime state, such as starting or stopping services for example.
administrator	Operator plus configuration privilege.

The identity specified when enabling global security is automatically mapped to the administrator role. Users, groups, can be added or removed to or from the Admin roles from the WebSphere Application Server administrative console at anytime. However, a server restart is required for the changes to take effect. A best practice is to map a group(s), rather than specific users, to admin roles because it is more flexible and easier to administer in the long run. By mapping a group to an Admin role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

In addition to mapping user or groups, a special-subject can also be mapped to the Admin roles. A special-subject is a generalization of a particular class of users. The AllAuthenticated special subject means that the access check of the admin role ensures that the user making the request has at least been authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action, as if no security were enabled.

Naming roles

The J2EE role-based authorization concept has been extended to protect the WebSphere CosNaming service.

CosNaming security offers increased granularity of security control over CosNaming functions. CosNaming functions are available on CosNaming servers such as the WebSphere Application Server. They affect the content of the WebSphere Name Space. There are generally two ways in which client programs will result in CosNaming calls. The first is through the JNDI interfaces. The second is CORBA clients invoking CosNaming methods directly.

If a user is assigned a particular naming role and that user is a member of a group that has been assigned a different naming role, the user will be granted the most permissive access between the role he was assigned and the role his group was assigned. For example, assume that user MyUser has been assigned the CosNamingRead role. Also, assume that group MyGroup has been assigned the CosNamingCreate role. If MyUser is a member of MyGroup, MyUser will be assigned the CosNamingCreate role because he is a member of MyGroup. If MyUser were not a member of MyGroup, he would be assigned the CosNamingRead role.

Four security roles are introduced: **CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete**. The name of the four roles are the same with WebSphere Advanced Edition Version 4.0.2. However, the roles now have authority level from low to high as follows:

- **CosNamingRead**. Users who have been assigned the CosNamingRead role will be allowed to do queries of the WebSphere Name Space, such as through the JNDI ″lookup″ method. The special-subject Everyone is the default policy for this role.
- **CosNamingWrite**. Users who have been assigned the CosNamingWrite role will be allowed to do write operations such as JNDI ″bind″, ″rebind″, or ″unbind″, plus CosNamingRead operations. The special-subject AllAuthenticated is the default policy for this role.
- **CosNamingCreate**. Users who have been assigned the CosNamingCreate role will be allowed to create new objects in the Name Space through such operations as JNDI ″createSubcontext″, plus CosNamingWrite operations. The special-subject AllAuthenticated is the default policy for this role.
- **CosNamingDelete**. And finally users who have been assigned CosNamingDelete role will be able to destroy objects in the Name Space, for example using the JNDI ″destroySubcontext″ method, as well as CosNamingCreate operations. The special-subject AllAuthenticated is the default policy for this role.

Users, groups, or the special subjects AllAuthenticated and Everyone can be added or removed to or from the Naming roles from the WebSphere web based administrative console at anytime. However, a server restart is required for the changes to take effect. A best practice is to map a group(s) or one of the special-subjects, rather than specific users, to Naming roles because it is more flexible and easier to administer in the long run. By mapping a group to an Naming role, adding or removing users to or from the group occurs outside of WebSphere and doesn't require a server restart for the change to take effect.

The CosNaming authorization policy is only enforced when global security is enabled. When global security is enabled, attempts to do CosNaming operations without the proper role assignment will result in a org.omg.CORBA.NO_PERMISSION exception from the CosNaming Server.

In the previous release (Version 4.0.2) of the product, each CosNaming function is assigned to only one role. Therefore, users who have been assigned CosNamingCreate role will not be able to query the Name Space unless they have also been assigned CosNamingRead. In most cases a creator would need to be assigned three roles: **CosNamingRead, CosNamingWrite, and CosNamingCreate**. This has been changed in the release. The **CosNamingRead** and **CosNamingWrite** roles assignment for the creator example in above have been included in **CosNamingCreate** role. In most of the cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from previous one.

Although the ability exist to greatly restrict access to the Name space by changing the default policy, doing so may result in unexpected org.omg.CORBA.NO_PERMISSION exceptions at runtime. Typically, J2EE applications access the Name space and the identity they use is that of the user that authenticated to WebSphere Application Server when they access the J2EE application. Unless the J2EE application provider clearly communicates the expected Naming roles, care should be taken when changing the default naming authorization policy.

Adding users and groups to roles

Before you begin

Before you perform this task, you should have already completed the steps in the Securing Web applications and Securing EJB applications articles where you created new roles and assigned those roles to EJB and Web resources. Complete these steps during application installation. This is because the environment (user registry) under which the application is running is not known until deployment. If you already know the environment in which the application is running and the user registry that is used, then you can use the Application Assembly Tool (AAT) to assign users and groups to roles. Using the administrative console to assign users and groups to roles is recommended.

Steps for this task

1. Open the application file. Open the application file by clicking **File** > **Open**. Browse and select the application file.
2. Open the application folder.
3. Click **Security Roles**.
4. Click the **Bindings** tab on the right hand side panel.
5. Select a role from the right navigation top panel.
6. Add a group to role by clicking **Add** under Groups and type in a group name. Click **OK**. Repeat this operation to add more groups.
7. Add a user to a role by clicking **Add** under Users. Type a user name and click **OK**. Repeat this operation to add more users.
8. Add a special subject (**All authenticated users** or **Everyone**) to a role. Click **Add** under Special Subjects and select **All authenticated users** or **Everyone** as required. Click **OK**. When All authenticated users or Everyone special subjects is assigned to a role, you can skip steps 6 and 7 for that role.
9. Repeat steps 5 through 8 for all the roles.
10. Click **Apply** when done.

Results

The `ibm-application-bnd.xmi` file in the application contains the users and groups to roles mapping table (*authorization table*).

Usage scenario

This step is required to secure an application.

What to do next

After securing an application, use the Application Assembly Tool (AAT). You can install an application using the administrative console.

Mapping users to RunAs roles

Before you begin

RunAs roles are used for delegation. The RunAs role is used by a Servlet or EJB component to invoke another EJB be impersonating as that role. Before you perform this task, you should already have completed Securing Web applications and Securing EJB applications where new roles were created and assigned to EJB/Web resources. Also you should have assigned users and groups to roles. This step should actually be done during the installation of the application. This is because, the environment (user registry) under which the application is going to be run is not known until deployment. If you already know what environment the application is going to be run and the user registry that is going to be used then you can use Application Assembly Tool to assign users to RunAs roles.

Steps for this task

1. Opening the Application file. To open the application file Click File->Open and browse and select the application file.
2. Click the application folder.
3. Click on the Bindings tab on the right hand side panel.
4. Click Add button under RunAs Bindings.
5. Select a role from the dropdown list of the Security Role.
6. Type in User Id and Password and Click OK. Make sure the User Id entered is part of the Security Role selected. If a special subject AllAuthenticated is assigned to the Security Role, you can use any valid User Id and Password. If a special subject Everyone is assigned to Security Role, you need not map any user to that role.
7. Repeat the above steps (4 to 6) for all the RunAs roles in the application.
8. Click Apply when done.

Results

The ibm-application-bnd.xmi file in the application contains the user to RunAs role mapping table.

Usage scenario

This step is required to secure an application. This step is required when a Servlet or an EJB in an application is configured with RunAs settings.

What to do next

After securing an application using assembly tool, an application can be installed using the administrative console.

Deploying secured applications

Before you begin

Before you perform this task, ensure that you have already designed, developed and assembled an application with all the relevant security configurations. For more information on these steps refer to the (Designing and developing secured application) and (Assembling secured applications) articles. In this context, deploying and installing an application are considered the same task. You can interchange them without any effect in the document.

Deploying applications that have security constraints (secured applications) is not much different than deploying applications without any security constraints. The only difference is that you might need to assign users and groups to roles for a secured application, which requires that you have the correct active registry. To deploy a new secured application click **Applications** > **Install New Application** in the navigation panel on the left and follow the steps as you are prompted. If you are installing a secured application, roles would have been defined in the application. If delegation was required in the application, RunAs roles would also have been defined.

One of the steps required to deploy secured applications is to assign users and groups to roles defined in the application. This task is completed as part of the step titled *Map security roles to users and groups*. This assignment might have already been done through the AAT tool. In that case you can confirm the mapping by going through this step. You can add new users and groups and modify existing information during this step.

If the applications support delegation, then a RunAs role is already defined in the application. If the Delegation policy is set to **Specified Identity** (during assembly) the intermediary invokes a method using an identity setup during deployment. Use the RunAs role to specify the identity under which the downstream invocations are made. For example, if the RunAs role is assigned user ″bob″ and the client ″alice″ is invoking a servlet (which has delegation set) which in turn calls the enterprise beans, then the method on the enterprise beans is invoked with ″bob″ as the identity. As part of the deployment process one of the steps is to assign or modify users to the RunAs roles. This step is titled ″Map RunAs roles to users″. Use this step to assign new users or modify existing users to RunAs roles when the delegation policy is set to Specified Identity.

These steps are common for both installing an application and modifying an existing application. If the application contains roles, you will see the ″Map security roles to users and groups″ link during application installation(as one of the steps) and also during managing applications (as a link in the Additional Properties section at the bottom).

Steps for this task

1. Click **Applications** > **Install New Application**. Complete the steps (non-security related) required prior to the step titled **Map security roles to users/groups**.
2. Map security roles to users and groups.
3. Map users to RunAs roles if RunAs roles exist in the application.
4. Click **Correct use of System Identity** to specify RunAs roles if needed. Complete this action if the application has delegation set up to use System Identity (applicable to enterprise beans only).

   System Identity would use the WebSphere Application Server security server ID to invoke downstream methods and should be used with caution as this ID has more privileges than other identities in terms of accessing WebSphere Application Server internal methods. This task has been provided to make sure that the deployer is aware that the methods listed in the panel have System Identity set up for delegation and to correct them if necessary. If no changes are necessary, skip this task.
5. Complete the remaining (non-security related) steps to finish installing and deploying the application.

What to do next

Once a secured application is deployed, make sure you can access the resources in the application with the correct credentials. For example, if your application has a protected Web module, make sure you only use the users listed in the roles for that Web resource to access.

Assigning users and groups to roles

Before you begin

Before you perform this task, ensure you already have completed securing Web applications and securing EJB applications where new roles were created and assigned to Web and EJB resources. This step assumes that all the roles are already created in your application. Also, you need to make sure that the user registry used is the current or active user registry. It is preferable to have the security turned on with the user registry of your choice before you begin this process. Make sure that if you have changed anything in the security configuration (for example, enabled security or changed user registry) save the configuration and restart the server before the changes become effective.

Since the default active registry is LocalOS it is not necessary (though recommended) to enable security if you want to use the LocalOS registry as your registry to assign users and groups to roles. You can enable security once the users and groups are assigned in this case. The advantage of enabling security with the appropriate registry before proceeding with this task is that you can make sure you have a valid security setup (which includes checking the user registry configuration) and avoiding any problems using the registry.

These steps are common for both installing an application and modifying an existing application. If the application contains roles, you see the ″Map security roles to users/groups″ link during installation application (as one of the steps) and also during application management (as a link in the Additional Properties section at the bottom).

Steps for this task

1. Click ″Map security roles to users/groups″ link. This should list all the roles that belong to this application. If the roles already had users or special subjects (All Authenticated, Everyone) assigned, they show up here.
2. To assign the special subjects click on either the Everyone or the ″All Authenticated″ checkbox for the appropriate roles.
3. To assign users or groups, select the role (multiple roles can be selected at the same time if the same users or groups will be assigned to all the roles), click on ″Lookup Users″ or ″Lookup groups″.
4. Get the appropriate users and groups from the registry by filling in the ″limit (number of items)″ and the ″Search String″ fields and clicking on search. The limit field will limit the number of users that are obtained and displayed from the registry. The pattern is a searchable pattern matching one or more users/groups. For example, user* will list users like user1, user2. A pattern of * indicates all users or groups. A word of caution here. Use the limit and the search strings cautiously so as not to overwhelm the registry. When using large registries (like LDAP) where thousands of users/groups information resides, a search for a large number of users or groups can make the system very slow and may even fail. When there are more entries than requested for, a message will indicate that on the top of the panel. You can refine your search (limit or the search pattern) until you have the required list.
5. Select the users/groups that should be members of this role(s) from the Available box and click ″>>″ button to add them to the role(s).
6. To remove existing users/groups select them from the Selected box and click ″<<″ to remove them. When removing existing users/groups from a role(s) care should be taken if that same role(s) is used as RunAs role. For example, if user1 is assigned to RunAs role role1 and you are trying to remove user1 from role1, the GUI validation will not allow you to delete the user since a user can only be a part of a RunAs role if the user is already in role (user1 should be in role1 in this case) either directly or indirectly (through a group). For more

information on the validation checks that are performed between RunAs role mapping and user/group mapping to roles, see the Mapping users to RunAs roles section.

7. Click **OK**.

   If there are any validation problems between the role assignments and the RunAs role assignments the changes will not be committed and an error message indicating the problem will be shown in the top of the screen. If there is a problem make sure that the user in RunAs role is also a member of the regular role. If the regular role contains a group which contains the user in the RunAs role make sure that the group is assigned to the role using the administrative console GUI (using the step 4 and 5 above - and not through the application assembly tool (AAT) or any other manual process where the complete name of the group - hostname\groupName or Distinguished Name(DN) - was not used.

Results

The users and groups information is added to the binding file in the application. This will be later used for authorization purposes.

Usage scenario

This step is required to assign users and groups to roles so that the application is secured to be called by the designated users.

What to do next

If you are installing the application complete the rest of the steps. Once the application is installed and running you should be able to access your resources according to the user/group mapping you did in this task. If you are managing applications and have modified the users/groups to role(s) mapping, make sure you save, stop and start the application so that the changes would be effective. Try accessing the J2EE resources in the application to make sure that changes are effective.

Security role to user (and group) mappings

Use this page to map security roles to users. You can map roles to specific users and or groups, or different categories.

To view this administrative console page, click **Application** > **Install New Application**. During the Application Installation Wizard, you are prompted to map security roles to users or groups. Role to user(s) or group(s) mapping can be changed for deployed applications. Click **Application** > **Install New Application** > *deployed_application* > **Map security roles to users/groups**.

Users: Specifies the users for role mapping. Ensure the users are defined in your chosen user registry.

To change the roles to users mapping, click **Manage Application** > *application* > **Map security roles to users**.

Data type	String

Groups: Specifies the groups for role mapping. Ensure the groups are defined in your chosen user registry.

To change the roles to users mapping, click **Manage Application** > *application* > **Map security roles to groups**.

Data type	String

Roles: Specifies the roles to which you want to map users and groups. Role privileges give users and groups permission to run as specified.

Select the check boxes to select a role or a set of roles. Click **Look-up Users** to map users to the roles that you have selected. Click **Look-up Groups** to map groups to the selected roles. Use the check boxes to map roles to **Everyone** or **All Authenticated special subject**.

Data type	String

Everyone: Specifies to map roles to everyone. Mapping a role to everyone means that anyone can access resources protected by this role, and essentially, there is no security.

Data type	Boolean

All Authenticated: Specifies to authenticate all users. Roles are mapped to all authenticated users, and all authenticated users in the selected User Registry are granted access to the role.

Data type	Boolean

Security role to user and group mappings

Use this page to select users and groups for security roles.

To view this administrative console page, click **Application** > **Install New Application**.

During the Install New Application Wizard, you are prompted to map security roles to users. You can also configure security roles to user mappings of deployed applications. Different roles can have different security authorizations. Mapping users or groups to a role authorizes those users or groups to access applications defined by the role. Users, groups and roles are defined when an application is installed or configured.

You can also select role to user and group mappings while you are deploying applications. After you deploy, in additional properties, select (Map Security roles to users). This link allows you to change user and group mappings to a role.

Look up users: Specifies whether the server looks up selected users.

Choose the role by selecting the check box beside the role, then click **Lookup users**. Fill in the limit and the pattern fields. The limit field contains the number of entries that should to be returned by the search. The pattern field contains the search pattern used for searching entries. For example, `bob*` searches all users or groups starting with `bob`. A limit of zero returns all the entries that match the pattern. Use the value only when a small number users or groups match that pattern in the registry. If the registry contains more entries that match the pattern than requested, a message appears in the console to indicate that there are more entries in the registry. You can either increase the limit or refine the search pattern to get all the entries.

Look up groups: Specifies whether the server looks up selected groups.

Choose the role by selecting the check box beside the role, then click **Lookup groups**. Complete the limit and the pattern fields. The limit field contains the number of entries that the search should return. The pattern field contains the search pattern used to search for entries. For example, `bob*` searches all users or groups starting with `bob`. A limit of zero returns all the entries that match the pattern. Use this value only when a small number users or groups match that pattern in the registry. If the registry contains more entries that match the pattern than requested, a message appears in the console to indicate that there are more entries in the registry. You can either increase the limit or refine the search pattern to get all the entries.

Role: Specifies user roles.

A number of administrative roles are defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following roles are valid:

- **Monitor—**least privileged that basically allows a user to view the server configuration and current state
- **Configurator—**monitor privilege plus the ability to change the server configuration
- **Operator—**monitor privilege plus the ability to change the run time state, such as starting or stopping services
- **Administrator—**operator plus configurator privilege

Range	Monitor, Configurator, Operator, Administrator

Everyone: Specifies to authenticate everyone.

Range	Monitor, Configurator, Operator, Administrator

All authenticated:

Range	Monitor, Configurator, Operator, Administrator

Look up users and groups settings: Use this page to select users and groups for security roles.

To view this administrative console page, click **Applications** > **Applications** > *application_name* > **Map security roles to users** > *role_name* > **Look up users or groups** button.

Different roles can have different security authorizations. Mapping users or groups to a role authorizes those users or groups to access applications defined by the role. Users, groups and roles are defined when an application is installed or configured. Use the Search field to display users in the Available Users list. Click the arrows to add users from the Available Users list to the Selected Users list.

Limit: Specifies the maximum number of users/groups that can be returned when assigning users/groups to roles.

A value of 0 implies a return of all users/groups that match the pattern. You can either increase the limit or refine the search pattern to get all the entries.

Data type	Integer
Units	User name
Default	20
Range	0 or more

Pattern: Indicates the search pattern used to search for the entries.

The pattern field should contain the search pattern that should be used to search for the entries. For example, bob* will search all users or groups starting with bob. A limit of 0 gets all the entries that match the pattern and should be used only when a small number users/groups match that pattern in the registry. If the registry contains more entries that match the pattern than requested for, a message shows in the console to indicate that there are more entries in the registry.

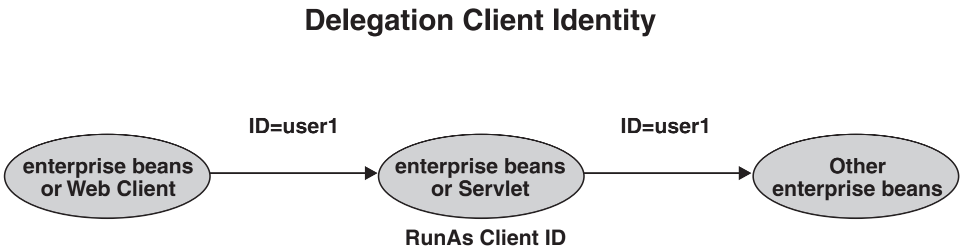
Data type	String
Units	Number of users
Default	20
Range	A-Z with *

Delegations: Delegation is a process security identity propagation from caller to called object. As per the J2EE specification, a servlet and enterprise beans can propagate either the client (remote user) identity when invoking enterprise beans or it can use another specified identity as specified in the corresponding deployment descriptor.

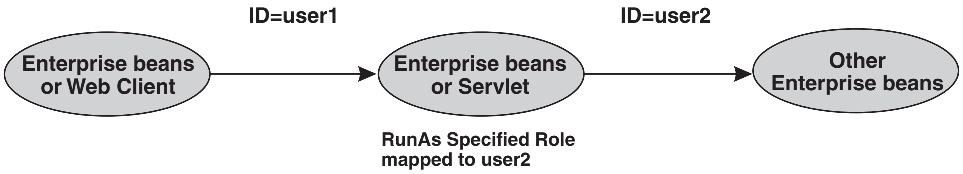
The IBM extension supports Enterprise JavaBeans (EJB) to propagate to the server ID when invoking other entity beans. There are three types of delegation:

- Delegate(RunAs) ClientIdentity
- Delegate(RunAs) SpecifiedIdentity
- Delegate(RunAs) System Identity

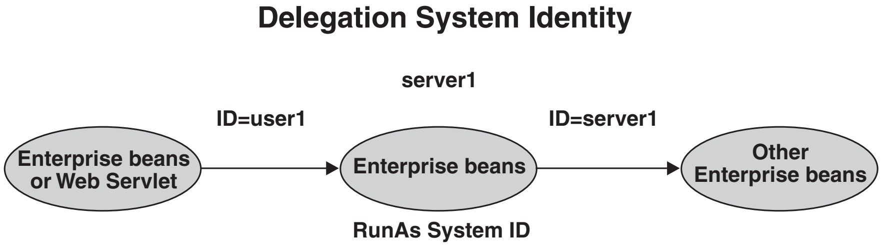
Delegate (RunAs) Client Identity


Delegation Client Identity
ID=user1
ID=user1
enterprise beans or Web Client
enterprise beans or Servlet
Other enterprise beans
RunAs Client ID

Delegate (RunAs) Specified Identity


ID=user1
ID=user2
Enterprise beans or Web Client
Enterprise beans or Servlet
Other Enterprise beans
RunAs Specified Role mapped to user2

Delegate (RunAs) System Identity

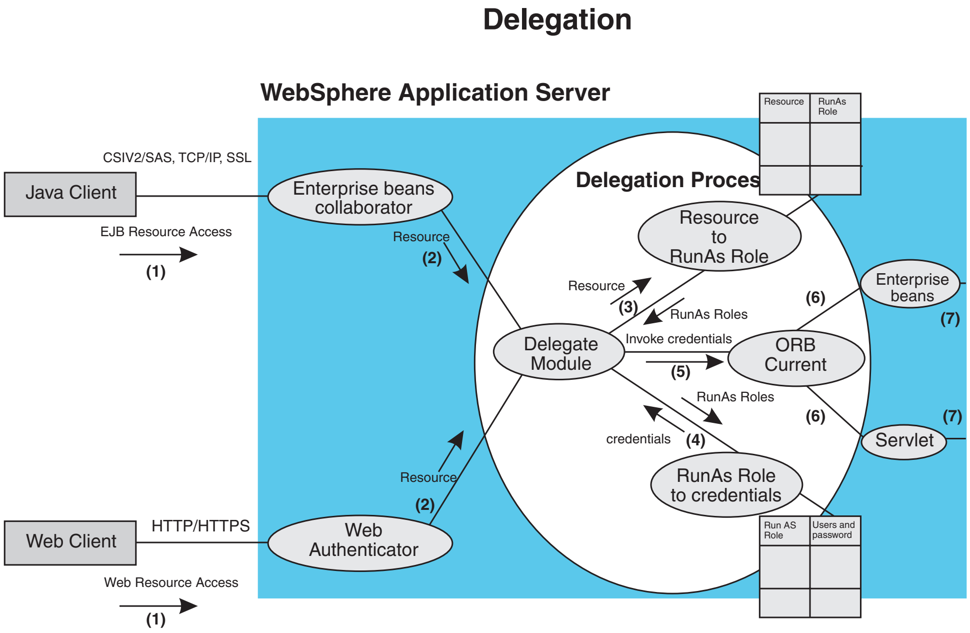


The EJB Specification only supports delegation (RunAs) at the EJB level. But an IBM extension allows EJB method level RunAs specification. Method EJB method level runAs specification allows one to specify a different RunAs role for different methods within the same enterprise beans.

The RunAs Specification is specified in the deployment descriptor (the `ejb-jar.xml` file in the EJB module and the `web.xml` file in the Web module). The IBM extension to RunAs Specification is specified in `ibm-ejb-jar-ext.xmi` file.

There is also an IBM specific binding file for each application that contains a mapping from RunAs role to user. This is specified in `ibm-application-bnd.xmi` file.

These specifications are read by the run time during application startup. The following figure illustrates the delegation mechanism as implemented in the WebSphere Application Server security model.



Delegation Process

There are two tables that help in the delegation process:

- Resource to RunAs Role mapping table
- RunAs Role to User ID and Password mapping table

Use the Resource to RunAs Role mapping table to get the role that is used by Servlet or enterprise beans to propagate to the next enterprise beans call.

Use the RunAsRole to User ID and Password mapping table to get the user ID that belongs to the RunAs role and its password.

Delegation is performed after successful authentication and authorization. During this process, delegation module consults the Resource to RunAsRole mapping table to get the RunAs role(3). The delegation module consults the RunAs role to user ID and password mapping table to get the user that belongs to the RunAs role(4). The user ID and password is used to create a new credential using the authentication module which is not shown in figure. The resulting credential is stored in the ORB Current as invocation credential(5). Servlet and enterprise beans when invoking other enterprise beans picks up the invocation credential from the ORB Current(6) and call the next enterprise beans(7).

Assigning users to RunAs roles:

Before you begin

Before you perform this task, you should already have completed securing Web applications and/or securing EJB applications where new RunAs roles were created and assigned to Web/EJB resources. This step assumes that all the RunAs roles are already created in your application. Since the user in the RunAs role can only be entered if that user (or a group that the user belongs to) is already part of the regular role the assigning users and groups to security roles task should be completed before this. Also, the user registry requirements are same as in the case of assigning users and groups to security roles task.

As mentioned above, you can only add a user to RunAs role if that user (or a group that the user belongs to) is already a member of that role. For example, if role1 is one of the roles which is also used as a RunAs role, the user user1 can be added to RunAs role role1 if user1 (or a group that user1 belongs to) is already assigned to the role role1. The GUI (AdminConsole) checks this logic when the Apply or OK buttons are pressed. If the check fails, the change will not be made and an error is indicated at the top of the screen.

If the special subjects ″Everyone″ or ″All Authenticated″ are assigned to a role, then no check takes place for that role.

The checking is done every time the Apply button in this panel is clicked or when the OK button in the ″Map security roles to users/groups″ is clicked. The check will make sure that all the users in all the RunAs role roles do exist directly or indirectly (through a group) in those roles in the ″Map security roles to users/groups″ panel. Note that if a role has been assigned both a user and a group that the user belongs then either the user or the group (not both) can be deleted from ″Map security roles to users/groups″.

One other thing to be noted here is that if the RunAs role user belongs to a group and if that groups has been assigned to that role, make sure that the assignment of this group to the role is done through AdminConsole (and not through the application assembly tool (AAT) or any other method) to avoid problems in some situations. This is because when using the AdminConsole the full name of the group is used (for example,. hostname\groupName in Windows, DN in LDAP). During the check, all the groups that the RunAs role user belongs are obtained from the registry. Since the list of groups obtained from the registry are the full names of the groups the check will work correctly. If a short name of the groups was entered using AAT (for example, group1 instead of CN=group1, o=myCompany.com) then this check would fail.

These steps are common to both installing an application and modifying an existing application. If the application contains RunAs roles, you will see the ″Map RunAs roles to users″ link during installing applications (as one of the steps) and also during managing applications (as a link in the Additional Properties section at the bottom).

Steps for this task

1. Click on ″Map RunAs roles to users″ link. This should list all the RunAs roles that belong to this application. If the roles already had users assigned, they show up here.
2. To assign a user, select the role (multiple roles can be selected at the same time if the same user will be assigned to all the roles), enter the user's name and password in the user name and password fields respectively. The user name entered can be either the short name (preferred) or the full name (as seen when getting users/groups from the registry). Click **Apply**.
3. The user will be authenticated using the active user registry. If authentication is successful, a check will made to make sure that this user (or a group that he belongs to) is mapped to the role in the ″Map security roles to users/groups″ panel. If authentication fails, make sure that the user/password is correct and the active registry configuration is correct.
4. To remove a user from RunAs role, select the role(s) and click on Remove.

Results

The RunAs role user is added to the binding file in the application. This will be used for delegation purposes when accessing J2EE resources.

Usage scenario

This step is required to assign users to RunAs roles so that during delegation the appropriate user will be used to invoke the EJB methods.

What to do next

If you are installing the application complete the rest of the steps. Once the application is installed and running you should be able to access your resources according to the RunAS role mapping you did in this task. Make sure you save the configuration.

If you are managing applications and have modified the RunAs roles to users mapping, make sure you save, stop and start the application so that the changes would be effective. Try accessing your J2EE resources to make sure that the new changes are in effect.

User and group selection settings: Use this page to select users and or groups for mapping security roles.

You may also encounter this panel in the administrative console as part of the wizard for installing an enterprise application or module.

Note: You must install an application that already has security roles defined before selecting users/groups for mapping.

<h3/>

Steps for this task

1. Click **Applications** > **Manage Applications** from the left navigation pane.

   A collection of installed application names appear.
2. Click the application name for which you wish to map security roles to users/groups.
3. Click the **Map security roles to users/groups** link.

   At runtime, the authorization checking will grant access in the following order: Everyone, All authenticated users, and then Select users/groups. If a user or group is in more than one of these roles, the first match will grant access.
4. To select a user or group for the role, enter a name in the search field or enter a search pattern.

   After a result of the search is displayed in the **Available Users/Groups** tree view, select one or more users or groups and click **Add**.
5. Click **OK** to commit the role to user or group mapping.
6. Repeat the steps for each role that needs to be mapped.

Unprotected EJB 2.0 methods protection settings: Use this page to ensure all unprotected EJB 2.0 methods have the correct level of protection.

To view this administrative console page, click **Application** > **Install New Application**. During the Install New Application Wizard, you are prompted to map security roles to users.

Ensure that all unprotected EJB 2.0 methods have the correct level of protection before you map users to roles.

Exclude: Specifies that the method is completely protected.

Data type	Check box
Default	Unchecked

Uncheck: Specifies that everyone can access the security method.

Data type	Check box
Default	Uncheck

Specify role: Specifies the EJB level of protection based on security role.

The roles listed in this drop-down menu are gleaned from the Application Scope. If the selected role is not in the module, then it is added to the modules or JAR files.

Data type	String

Units	Role

Module name: Specifies the name of the module.

If a module name appears in this list, then the module has some unprotected EJB methods.

Data type	String
Units	Module name

Protection: Specifies the level of protection assigned to a particular module name.

Data type	String
Default	Unchecked

EJB 1.0 method protection level settings: Use this page to ensure all unprotected EJB 1.0 methods have the correct level of protection.

To view this administrative console page, click **Application** > **Install New Application**. During the Install New Application Wizard, you are prompted to ensure that all unprotected EJB 1.0 methods have the correct level of protection.

Ensure that all unprotected EJB 1.0 methods have the correct level of protection before you map users to roles.

EJB Module: Specifies the EJB module name.

Data Type	String
Units	EJB module name

Module URI: Specifies the JAR file name.

Data Type	String
Units	JAR file name

Method protection: Specifies the level of protection assigned to the EJB module.

A selected box means to *Deny All* and that the method is completely protected.

Data Type	Check box
Default	Unchecked
Range	Yes or No

RunAs roles to users mapping: Use this page to map RunAs roles to users. You can change the RunAs settings after an application deploys.

To view this administrative console page, click **Applications** > **Install New Application**. As you complete the application installation wizard, you are prompted to map RunAs roles to users. You can change RunAs roles to users mappings for deployed applications. Click **Applications** > *application_name* > **Map RunAs roles to users** in the Additional Properties section.

The enterprise beans you are installing contain predefined RunAs roles. RunAs roles are used by enterprise beans that need to run as a particular role for recognition while interacting with another enterprise bean.

User name: Specifies a user name for the RunAs role user.

This user should already map to the role specified in the Mapping users and groups to roles panel. You can map the user to its appropriate role by either mapping the user to that role directly or mapping a group that contains the user to that role.

Data type	String

Password: Specifies the password for the Run As user.

Data type	String

Confirm password: Specifies the confirmed password of the administrative user.

Data type	String

Role: Specifies administrative user roles.

A number of administrative roles have been defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the web based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following roles are valid:

- **Monitor—**least privileged that basically allows a user to view the WebSphere configuration and current state
- **Configurator**—monitor privilege plus the ability to change the WebSphere configuration
- **Operator—**monitor privilege plus the ability to change runtime state, such as starting or stopping services for example
- **Administrator—**operator plus configurator privilege

Updating and redeploying secured applications:

Before you begin

Before you perform this task, ensure you have already completed securing Web applications, securing EJB applications, and deploying them in WebSphere Application Server. This section addresses the way to update existing applications.

Steps for this task

1. Once an application deploys, you can use the administrative console to modify the existing users and groups mapping to roles. The task titled Mapping users and groups to roles details the required steps. You can also modify the users for the RunAs roles using the administrative console. The task titled Mapping users to RunAs roles details the required steps. Once you complete the changes make sure to save the changes. Stop and start the application for the changes to become effective.

2. If you need to update any other security related information, use the Application Assembly Tool (AAT). Use the AAT to modify roles, method permissions, auth-constraints, data-constraints and other security related information. Once the changes are done, save the *EAR* file, uninstall the old application, deploy the modified application and start the application to make the changes effective. If information about roles is modified make sure you update the user and group information using the administrative console.

Results

The applications are modified and redeployed.

Usage scenario

This step is required to modify existing secured applications.

What to do next

Once the secured applications are modified and either restarted or redeployed, make sure that the changes are effective by accessing the resources in the application.

Testing security

Before you begin

After configuring global security and restarting all of your servers in a secure mode, it is best to validate that security is properly enabled. There are a few techniques that you can use to test the various security login types. For example, you can test Web FormLogin, Web BasicAuth login, and Java Client BasicAuth login. These basic tests show the fundamental security components working properly. Here are the steps you can take to validate your security configuration. After completing these basic tests, thoroughly test your secured applications.

Steps for this task

1. Test Web-based BasicAuth with *snoop*, by accessing the following URL: *http://hostname.domain:9080/snoop*. A login panel appears; if it does not, there is a problem. If the panel appears, type in any valid userID and password in your configured user registry.

   Note: In a Network Deployment environment, the snoop servlet is only available in the domain if you included the *DefaultApplication* option when adding the application server to the cell. The option *-includeapps* for the *addNode* command migrates the *DefaultApplication* to the cell. Otherwise, skip this step.
2. Test Web-based FormLogin by bringing up the administrative console: *http://hostname.domain:9090/admin*. A form-based login page appears, if it does not, there is a problem. If the panel appears, type in the administrative userid and password used for configuring your user registry when configuring security. Note that, when the authentication mechanism is set as LTPA, the host name should be a fully qualified host name (that is, *myhost.mycompany.com:9090* rather than just *myhost:9090*).
3. Test Java Client BasicAuth with *dumpNameSpace* by executing *<WAS_HOME>\bin\dumpNameSpace.bat*. A login panel appears, if it does not, there is a problem. If the panel appears, type in any valid userid and password in your configured user registry.

Results

The results of these tests, if successful, indicate that security is fully enabled and working properly. This test is just a start to security verification. Thoroughly test all of your applications in secure mode. After enabling security, you want to ensure that your system comes up in secure mode.

What to do next

If all tests pass, proceed with more rigorous testing of your secured applications. If you have any problems, review the output logs in the *WAS /logs/nodeagent* or *WAS /logs/server_name directory*, respectively. Then check the security troubleshooting article to see if it references any common problems.

Managing secured applications

Before you begin

Administering secure applications requires access to the WebSphere administrative console. The URL to access the console is `http://localhost:9080/admin`, and `http://localhost:9090/admin` for a Network Deployment environment. If security is not yet enabled, you are only prompted for a user name for change control. Otherwise, you need to log in with a valid user ID and password which is given administrative access. Once you log in, the page shown below is the first thing you see. To administer security, complete the Security component tasks.

Steps for this task

1. Configure global security
2. Assign users to administrative roles.
3. Assign users to naming roles.
4. Configure authentication mechanisms.
5. Configure LTPA.
6. Configure trust association interceptors.
7. Configure Single Sign-On.
8. Configure user registries.
   a. Configure LocalOS user registries.
   b. Configure LDAP user registries.
   c. Configure custom user registries.
9. Configure Java Authentication and Authorization Service (JAAS).
10. Configuring CSIv2 and SAS authentication protocols.
11. Secure Socket Layer configuration repertoire settings.
12. Configure Java 2 Security Manager.

Global security

Global security applies to all applications running in the environment and determines whether security is used at all, the type of registry against which authentication takes place, and other values, many of which act as defaults.

The term *global security* implies the security configuration which is effective for the entire security domain. A security domain consists of all servers configured with the same user registry *realm* name. In some cases, the realm can be the machine name of a LocalOS user registry. In this case, all application servers must reside on the same physical machine. In other cases, the realm can be the machine name of an LDAP user registry. Since LDAP is a distributed user registry, a multiple node configuration is supported, such as the case for a Network Deployment environment. The basic requirement for a security domain is that the access ID returned by the registry from one server within the security domain is the same access ID as that returned from the registry on any other server within the same security domain. The access ID is the unique identification of a user and is used during authorization to determine if access is permitted to the resource.

Configuration of global security for a security domain consists of configuring the common user registry, the authentication mechanism, and other security information, which defines the behavior of a security domain. The other security information which you can configure includes Java 2 Security Manager, Java Authentication and Authorization Service (JAAS), Java 2 Connector authentication data entries, CSIv2/SAS authentication protocol (RMI/IIOP security), and other miscellaneous attributes. The global security configuration usually applies to every server within the security domain.

Configuring global security

Before you begin

It is helpful to understand security from an infrastructure standpoint so that you know the advantages of different authentication mechanisms, user registries, authentication protocols, an so on. Picking the right security components to meet your needs is apart of configuring global security. The following sections help you make these decisions. Read the following articles before continuing with the security configuration.

- Global Security
- Welcome to Security

Once you understand the security components, you can proceed to configure global security in WebSphere Application Server.

Steps for this task

1. Start the WebSphere Application Server administrative console by going to **http://yourhost.domain:9090/admin** after the WebSphere Application Server has been started. If security is currently disabled, log in with any user ID. If security is currently enabled, log in with a predefined administrative ID and password (this is typically the server user ID specified when you configured the user registry).
2. Once inside the administrative console, click **Security** from the left navigation menu. Configure the authentication mechanism, user registry, and so on. It is not too important which order you do this in, however, when you select the **Enabled** flag in the global security panel, you should have completed all of these tasks. When you click **Apply** or **OK** and the **Enabled** flag is set, a verification occurs to see if the administrative user ID and password can be authenticated to the configured user registry. If you do not configured these, the validation fails.
3. Choose a user registry and configure it. Details about configuring user registries can be found in the following article, Configuring User Registries. Here you need to decide which user registry to configure (LocalOS, LDAP, or Custom), and then specify details about the one you decide upon. One of these details that is common to all user registries is the administrative user ID. This ID is a member of the chosen user registry, but has special privileges in

WebSphere Application Server and must have special privileges for the user registry it represents. Specifically, if the user registry is a LocalOS registry, the administrative user ID that you choose must have ″Act as Part of Operating System″ privileges on a Windows NT system or ″root″ privileges on a UNIX system. Also, on Windows NT systems, the administrative user ID you choose must not be the same name as the machine name of your system, since the registry sometimes returns machine specific information when you query a user of the same name. Also, on LDAP user registries, ensure that the administrative user ID is a member of the registry and not just the LDAP administrative ID. The entry must be searchable.

4. Configure the authentication mechanism. To get details about configuring authentication mechanisms, go to the following article, Configuring Authentication Mechanisms. There are two authentication mechanisms to choose from in the Global Security panel: Simple WebSphere Authentication Mechanism (SWAM) and Lightweight Third-Party Authentication (LTPA). However, only LTPA requires any additional configuration parameters. Use the SWAM option for single server requirements. Use the LTPA option for multi-server distributed requirements. SWAM credentials are not forwardable to other machines and for that reason do not expire. Credentials for LTPA are forwardable to other machines and for security reasons are expirable. This expiration time is configurable. If you do choose to go with LTPA, you should also visit the following link if you want single sign-on support. This support permits browsers to visit different product servers without having to authenticate multiple times. The article describing this feature is Configuring Single Sign-On (SSO).
5. For special security requirements from Java clients, you might want to configure the authentication protocol. This entails choosing a protocol, either CSIv2 or SAS. The CSIv2 protocol is new to WebSphere Application Server Version 5 and has many new and improved features. The SAS protocol is still provided as a backwards compatibility to previous product releases but is being deprecated. For details on configuring CSIv2 or SAS, see the article, Configuring Authentication Protocols (CSIv2 and SAS).
6. It is important that you change the default SSL keystore and truststore with which WebSphere Application Server ships.

   This action protects the integrity of the messages being sent across the Internet. The product provides a single location where you can specify SSL configurations that can be used among the various WebSphere Application Server features that use SSL including the LDAP user registry, Web Container, and the Authentication Protocol (CSIv2 and SAS). Create a new keystore and truststore, by referring to the Creating KeyStore files and Creating TrustStore files articles. You can create different keystores and truststores for different uses or you can create just one set for everything that the server uses SSL for. Once you create these new keystores and truststores, specify them in the SSL Configuration Repertoire. To get to the SSL Configuration Repertoire, click **Security** > **SSL**. You can either edit the `DefaultSSLConfig` or create a new SSL configuration with a new alias name. If you create a new alias name for your new keystore and truststore files, change every location that references the SSL configuration alias `DefaultSSLConfig`. A list of these locations follow:

   - **Security** > **User Registries** > **LDAP** (at the bottom of the panel)
   - **Security** > **Authentication Protocol** > **CSIv2 Inbound Transport**
   - **Security** > **Authentication Protocol** > **CSIv2 Outbound Transport**
   - **Security** > **Authentication Protocol** > **SAS Inbound Transport**
   - **Security** > **Authentication Protocol** > **SAS Outbound Transport**

- **Servers** > **Application Servers** > *app_server_name* > **Web Container** > **HTTP transports** > *host_link*

7. Click **Security** > Global Security to configure the rest of the security settings and enable security. The panel is the last to visit because this panel does a final validation of the security configuration. When you click on **OK** or **Apply** from this panel, the security validation routine is performed and any problems are reported at the top of the page. At this panel, the following values are considered. When you complete all of the fields here, click **OK** or **Apply** to accept the selected settings. Click **Save** above to persist these settings out to a file. If you see any informational messages in *red* text color, than a problem has occurred with the security validation. Typically, the message indicates the problem. So, review your configuration to ensure the user registry settings are accurate and the correct registry is selected. In some cases the LTPA configuration may not be fully specified. See the Global security settings article for detailed information.
8. If after selecting **OK** or **Apply** on the **Security** > **Global Security** panel and there are no validation problems, then you can proceed to store the configuration for use by the server the next time it is restarted. To save the configuration, click the **Save** option in the menu bar at the top. This action writes the settings out to the configuration repository. If you do not select **Apply** or **OK** in the Global Security panel before selecting **Save** on the main menu, your changes are not written to the repository.

Enabling and removing global security

IBM WebSphere Application Server security can be enabled or not enabled. If security is not enabled, all other security settings are ignored.

Steps for this task

1. Enable global security in the WebSphere Application Server.

   It is important that you have just enabled the **Security** > **Global Security** > Enabled flag to **ON** so that security gets enabled upon a server restart. Once you have changed the configuration so that security is enabled in the configuration, you need to restart the server so that the configuration becomes effective.
2. Before restarting the server, log off of the administrative console. You can log off by selecting Logout at the top menu bar.
3. Stop the server by going to the command line in the WebSphere Application Server /bin directory and issue a `stopServer server_name` command.
4. Now restart the server in secure mode by issuing the command `startServer server_name`. Once the server is secure, you will not be able to stop the server again without specifying an administrative user name and password. To stop the server once security is enabled, issue the following command: `stopServer server_name -username userid -password password`. Alternatively, you can edit the sas.client.props file in the WAS /properties directory and edit the com.ibm.SOAP.loginUserid or com.ibm.SOAP.loginPassword properties to contain these administrative IDs.
5. If you have any problems restarting the server, review the output logs in the WAS /logs/server_name directory. Then check the Troubleshooting security article to see if it references any common problems.

Disabling global security:

Steps for this task

1. When disabling global security, it is assumed that you have just flipped the **Security** > **Global Security** > Enabled flag to **OFF** so that security gets disabled upon a server restart. Once you have changed the configuration so that security is enabled in the configuration, you need to restart the server so that the configuration becomes effective.
2. Before restarting the server it's best to log off of the administrative console. You can logout by selecting log off at the top menu bar.
3. Stop the server by going to the command line in the WebSphere Application Server /bin directory and issue a `stopServer server_name -username adminuser -password -adminpw` command. You have to include the administrative user ID to stop the server when security is enabled. Alternatively, you can edit the sas.client.props file in the WAS /properties directory and edit the com.ibm.SOAP.loginUserid or com.ibm.SOAP.loginPassword properties to contain these administrative IDs. In this case if you just enter `stopServer server_name`, the administrative userid will get picked up from the sas.client.props file.
4. Now start the server back up in secure mode by issuing the command `startServer server_name`.
5. If you have any problems restarting the server, review the output logs in the WAS /logs/server_name directory.

Usage scenario

This scenario is specifically for a standalone setup where you have a single application server. You will likely utilize your LocalOS registry for your repository of users. The authentication mechanism will likely be SWAM. The application server cannot communicate securely to other application servers as the SWAM authentication mechanism does not contain a forwardable token which can be sent to downstream servers.

What to do next

Upon restart of the server in secure mode, there are a couple of simple tests to verify that most facets of security are working properly.

1. Test Basic Auth with snoop by accessing the following URL: **http://hostname.domain:9080/snoop**. A login panel should appear, if it does not, there's a problem. If it does, type in any valid userid/password in your configured user registry.
2. Test Java Client with dumpNameSpace by executing, WAS_Home>\bin\dumpNameSpace.bat. A login panel should appear, if it does not, there's a problem. If it does, type in any valid userid/password in your configured user registry.
3. Test Form Login by bringing up the administrative console: **http://hostname.domain:9090/admin**. A form-based login page should appear, if it does not, there's a problem. If it does, type in the administrative userid/password which was used for configuring your user registry when configuring security above. When the Authentication Mechanism is set as LTPA, the host name should be a fully qualified host name (for example, **myhost.mycompany.com:9090** rather than just **myhost:9090**).

If you encountered a problem with any of these tests, check the WebSphere Application Server `/logs/server_name/SystemOut.log` file for hints about the problems that occurred. Also refer to (Troubleshooting Security) for solutions.

Global security settings

Use this page to configure global security.

To view this administrative console page, click **Security** > **Global Security**.

If you are configuring security for the first time, complete the steps in Configuring global security in the InfoCenter to avoid problems. Once security is configured, validate any changes to the registry or authentication mechanism panels. Click **Apply** to validate the user registry settings. An attempt is made to authenticate the server ID to the configured user registry. Validating the user registry settings after enabling global security can avoid problems when you restart the server for the first time.

Enabled: Specifies for the server to enable security subsystems.

This flag is commonly referred to as the *global security flag* in WebSphere Application Server literature. When enabling security, set the authentication mechanism configuration and specify a valid user ID and password in the selected user registry configuration.

Data type	Boolean
Default	Disable

Enforce Java 2 Security: Specifies whether to enable or disable Java 2 Security permission checking. By default, Java 2 security is disabled. However, if you enabled global security, this automatically enables Java 2 security. You can choose to disable Java 2 security, even when global security is enabled.

When Java 2 Security is enabled and if an application requires more Java 2 security permissions then are granted in the default policy, then the application might fail to run properly until the required permissions are granted in either the *app.policy* file or the *was.policy* file of the application. AccessControl exceptions are generated by applications that donot have all the required permissions. Consult the InfoCenter and review the *Java 2 Security and Dynamic Policy* sections if you are unfamiliar with Java 2 security.

If your sever does not restart after you enable global security, you can disable security. Go to your `${was_install_root}\bin` directory. Excecute the command `wsadmin -conntype NONE`. At the `wsadmin>` prompt, enter `securityoff`. Type `exit` to get back to a command prompt. Now you should be able to start the server again, with security disabled. This enables you to check what might not be set correctly through the administrative console.

Data type	Boolean
Default	Disabled
Range	Enabled or Disabled

Use Domain Qualified User Names: Specifies the user names to qualify with the security domain within which they reside.

Data type	Boolean
Default	Disabled
Range	Enable or Disable

Cache Timeout: Specifies the timeout value in seconds for security cache. This value is a relative timeout.

Data type	Integer
Units	Seconds
Default	600
Range	Greater than 30 seconds

Issue Permission Warning: Specifies that when the *Issue permission warning* is enabled, during application deployment and application start, the security run time emits a warning if applications are granted any custom permissions. Custom permissions are permissions defined by the user applications, not JDK permissions. JDK permissions are permissions in `package java.*` and `javax.*`.

The WebSphere product provides support for policy file management. There are a number of policy files in this product, some of them are static and some of them are dynamic. Dynamic policy is a template of permissions for a particular type of resource. There is no code base defined or relative code base are used in the dynamic policy template. The real code base is dynamically created from the configuration and runtime data. The `filter.policy` file contains a list of permissions that an application should not have according to the J2EE 1.3 Specification. For more information on permissions, see the Java 2 Security Policy Management article in the InfoCenter.

Data type	Boolean
Default	Disabled
Range	Enable or Disable

Active Protocol: Specifies the active authentication protocol for RMI/IIOP requests when security is enabled. In previous releases the SAS protocol was the only available protocol.

This release includes an OMG protocol called CSIv2 which supports increased vendor interoperability and additional features. If all servers in your entire security domain are Version 5.0 servers, it is best to specify **CSI** as your protocol. If some servers are 3.x or 4.x servers, specify **CSI and SAS**.

Data type	String
Default	BOTH
Range	CSI and SAS, CSI

Active Authentication Mechanism: Specifies the active authentication mechanism, when security is enabled.

In WebSphere Application Server, Version 5 SWAM and LTPA are the supported authentication mechanism.

Data type	String
Default	SWAM (WebSphere Application Server)
Range	SWAM, LTPA

Active User Registry: Specifies the active user registry, when security is enabled.

You can configure settings for one of the following user registries:

- Local operating system
- LDAP user registry. The LDAP User Registry settings are used when users and groups reside in an external LDAP directory. When security is enabled and any of these properties are changed, go to the **Global Security** panel and click **Apply** or **OK** to validate the changes.
- Custom user registry

Data type	String
Default	LocalOS
Range	LocalOS, LDAP, Custom

Administrative console and naming service authorization

Administrative Console

WebSphere Application Server extended J2EE security role based access control to protect the WebSphere Application Server administrative subsystem. Four administrative roles have been defined to provide degrees of authority needed to perform certain WebSphere Application Server administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The four administrative security roles are defined in the following table:

Role	Description
monitor	Least privileged that basically allows a user to view the WebSphere Application Server configuration and current state.
configuration	Monitor privilege plus the ability to change the WebSphere Application Server configuration.
operator	Monitor privilege plus the ability to change runtime state, such as starting or stopping services for example.
administrator	Operator plus configuration privilege and th epermission required to access sensitive data including server password, LTPA password and keys, and so on.

When WebSphere Application Server global security is enabled, the administrative subsystem role based access control is enforced. The administrative subsystem includes Security Server, UserRegistry, and all the JMX MBeans. When security is enabled, both the web based administrative console and the administrative scripting tool will require users to provide the required authentication data. Moreover, the administrative console is designed such that the control functions that are displayed on the GUI pages are adjusted according to the security roles a user has. For example, a user who has only the monitor role will only be able to see non-sensitive configuration data. A user with the operator role would have available to him on GUI pages buttons to change the system state.

The server identity specified when enabling global security is automatically mapped to the administrative role. Users, groups, can be added or removed to or from the administrative roles from the WebSphere Application Server Web-based administrative console at anytime. However, a server restart is required for the changes to take effect. A best practice is to map a group(s), rather than specific users, to administrative roles because it is more flexible and easier to administer in the long run. By mapping a group to an administrative role, adding or removing users to or from the group occurs outside of WebSphere Application Server and doesn’t require a server restart for the change to take effect.

In addition to mapping user or groups, a special-subject can also be mapped to the administrative roles. A special-subject is a generalization of a particular class of users. The AllAuthenticated special subject means that the access check of the administrative role ensures that the user making the request has at least been authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action, as if no security were enabled.

When global security is enabled, WebSphere Application Server servers run under the server identity which is defined under the active user registry configuration. Although it is not shown on administrative console and other tools, a special Server subject is mapped to the administrator role. This is why the WebSphere Application Server server runtime code, which runs under the server identity, would have the required authorization to execute runtime operations. If no other user has been assigned administrative roles, one can login to administrative console or to wsadmin scripting using server identity to perform administrative operations and to assign other users or groups to administrative roles. Because the server identity is assigned to the administrative role by default, the administrative security policy requires administrative role to perform the following operations:

- Change server ID and server password
- Enable or disable WebSphere Application Server global security
- Enforce or disable Java 2 Security
- Change LTPA password or generate keys
- Assign users and groups to administrative roles

When enabling security for the first time, you may perform the following steps to assign one or more users and groups to administrative roles. When global security is enabled, the following steps can be performed by users who have the administrative role. Before performing the following steps, one must configure the active user registry because user and group validation in the following steps depends on active user registry.

Naming service authorization

WebSphere Application Server extended J2EE security role based access control to protect the WebSphere Application Server naming subsystem. CosNaming security offers increased granularity of security control over CosNaming functions. CosNaming functions are available on CosNaming servers such as the WebSphere Application Server. They affect the content of the WebSphere Application Server Name Space. There are generally two ways in which client programs will result in CosNaming calls. The first is through the JNDI interfaces. The second is CORBA clients invoking CosNaming methods directly.

Four security roles are introduced : CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The name of the four roles are the same with WebSphere Application Server Advanced Edition v4.0.2 However, the roles now have authority level from low to high as follows:

- **CosNamingRead**. Users who have been assigned the CosNamingRead role will be allowed to do queries of the WebSphere Application Server Name Space, such as through the JNDI ″lookup″ method. The special-subject Everyone is the default policy for this role.
- **CosNamingWrite**. Users who have been assigned the CosNamingWrite role will be allowed to do write operations such as JNDI ″bind″, ″rebind″, or ″unbind,″ plus CosNamingRead operations. The special-subject AllAuthenticated is the default policy for this role.
- **CosNamingCreate**. Users who have been assigned the CosNamingCreate role will be allowed to create new objects in the Name Space through such operations as JNDI ″createSubcontext,″ plus CosNamingWrite operations. The special subject AllAuthenticated is the default policy for this role.
- **CosNamingDelete**. And finally users who have been assigned CosNamingDelete role will be able to destroy objects in the Name Space, for example using the JNDI ″destroySubcontext″ method, as well as CosNamingCreate operations. The special-subject AllAuthenticated is the default policy for this role.

Additionally, a ″Server″ special subject is assigned to all the four CosNaming roles by default. The Server special subject allows a WebSphere Application Server server process, which runs under the server identity, to have access to all the CosNaming operations. Note that the Server special subject is not displayed and cannot be modified via the administrative console nor other administrative tools.

Users, groups, or the special subjects AllAuthenticated and Everyone can be added or removed to or from the Naming roles from the WebSphere web based administrative console at anytime. However, a server restart is required for the changes to take effect. A best practice is to map groups or one of the special-subjects, rather than specific users, to Naming roles because it is more flexible and easier to administer in the long run. By mapping a group to an Naming role, adding or removing users to or from the group occurs outside of WebSphere Application Server and doesn't require a server restart for the change to take effect.

The CosNaming authorization policy is only enforced when global security is enabled. When global security is enabled, attempts to do CosNaming operations without the proper role assignment will result in a org.omg.CORBA.NO_PERMISSION exception from the CosNaming Server.

In WebSphere Application Server Version 4.0.2, each CosNaming function is assigned to only one role. Therefore, users who have been assigned CosNamingCreate role will not be able to query the Name Space unless they have also been assigned CosNamingRead. And in most cases a creator would need to be assigned three roles: CosNamingRead, CosNamingWrite, and CosNamingCreate. This has been changed in the release. The CosNamingRead and CosNamingWrite roles assignment for the creator example in above have been included in CosNamingCreate role. In most of the cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from previous one.

Although the ability exist to greatly restrict access to the Name space by changing the default policy, doing so may result in unexpected org.omg.CORBA.NO_PERMISSION exceptions at runtime. Typically, J2EE applications access the Name space and the identity they use is that of the user that authenticated to WebSphere when they access the J2EE application. Unless the

J2EE application provider clearly communicates the expected Naming roles, care should be taken when changing the default naming authorization policy.

Assigning users to administrator roles

Before you begin

The following steps are needed to perform this task. In administrative console, expand the folder ″System Administration″ and click the link **Console Users** or **Console Groups**.

Steps for this task

1. To add a user or a group, click the **Add** button on the Console users or Console groups panel.
2. To add a new administrative user, enter a user identity to the user text box and high light one administrative role, and then click the **OK** button. If there is no validation error, the specified user will be displayed with the assigned security role. To add a new administrative group, either enter a group name or select either EVERYONE or ALLAUTHENTICATED special subject, and then click the **OK** button. If there is no validation button, the specified group or special subject will be displayed with the assigned security role.
3. To remove a user or group assignment, click the **remove** button on the ″Console Users″ or ″Console Groups″ panel. On the users or groups panel, click the check box of the user or group to be removed and then click **OK**.
4. To manage the set of users or groups to be displayed, expand the filter folder on the right hand side panel, and modify the filter text box. For example, setting the filter to ″user*″ will allow only users with the ″user″ prefix to be displayed.
5. After modifications have been made, click the ″save″ link to save the mappings.

   Modifications will take effect after the server is restarted.

Usage scenario

This task, mapping user and groups to administrative role, is performed to assign users to perform WebSphere Application Server administrative functions. Users and groups assigned to the administrator roles can perform all administrative operations and can setup both J2EE role based and Java 2 security policy. Users assigned to the configurator role can perform all day to day configuration tasks including installing and uninstalling applications, assigning users and groups to role mapping for applications, setting run-as configurations, setting up Java 2 security permissions for applications, and customizing CSIv2, SAS, and SSL configurations.

What to do next

If you are setting up administrative users and groups in preparation to enable security, you may proceed to restart the server for the modification to take effect. After server is restarted, all administrative resources will be protected. Because the administrative security configuration is at cell level, you will need to restart all the servers.

Console users settings

Use this page to give users specific authority to administer WebSphere Application Server using tools such as the administrative console or wsadmin scripting. The authority requirements are only effective when global security is enabled.

To view the **Console Users** administrative console page, click **System Administration** > **Console Users**. To view the **CORBA Naming Service Users** administrative console page, refer to the CORBA Naming Service Users article.

User: Specifies users.

The users entered must exist in the configured active user registry.

Data type	String

Role: Specifies user roles.

A number of administrative roles are defined to provide degrees of authority needed to perform certain product administrative functions from either the web based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled.

When you add a valid user, you can associate a role with them to give specific access rights. There are four different levels of authority for which you may grant access:

- **Monitor—**least privileged that basically allows a user to view the product configuration and current state
- **Configurator—**monitor privilege plus the ability to change the product configuration
- **Operator—**monitor privilege plus the ability to change the run time state, such as starting or stopping services
- **Administrator—**operator plus configurator privilege and the permission to access sensitive configuration data including server password, LTPA password and keys, and so on.

Data type	String
Range	Monitor, Configurator, Operator, Administrator

Login Status: Specifies the login status of users.

This column appears regardless of whether you are logged in or not.

Console groups settings

Use this page to give groups specific authority to administer the WebSphere Application Server using tools such as the administrative console or wsadmin scripting. The authority requirements are only effective when global security is enabled.

To view the **Console Groups** administrative console page, click **System Administration** > **Console Groups**. To view the **CORBA Naming Service Groups** administrative console page, refer to the (CORBA Naming Service Groups) article.

Configuration tab

Group Specifies groups.

Data type	String

Role Specifies user roles.

A number of administrative roles are defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following roles are valid:

- **Monitor—**least privileged that basically allows a user to view the server configuration and current state
- **Configurator—**monitor privilege plus the ability to change the server configuration
- **Operator—**monitor privilege plus the ability to change the run time state, such as starting or stopping services
- **Administrator—**operator plus configurator privilege and the permission to access sensitive configuration data including server password, LTPA password and keys, and so on

Data type	String
Range	Monitor, Configurator, Operator, Administrator

Assigning users to naming roles

The following steps are needed to perform this task. In admin console, expand the folder ″Environment″ and then expand ″Naming,″ and click the link ″CORBA Naming Service Users″ or ″CORBA Naming Service Groups.″

Steps for this task

1. To add a user or a group, click the **Add** button on the Console users or Console groups panel.
2. To add a new administrative user, enter a user identity to the user text box and high light one administrative role, and then click the **OK** button. If there is no validation error, the specified user will be displayed with the assigned security role. To add a new administrative group, either enter a group name or select either EVERYONE or ALLAUTHENTICATED special subject, and then click the **OK** button. If there is no validation button, the specified group or special subject will be displayed with the assigned security role.
3. To remove a user or group assignment, click the **remove** button on the ″Console user″ or ″Console group″ panel. On the user or group panel, click the check box of the user or group to be removed and then click **OK**.
4. To manage the set of users or groups to be displayed, expand the filter folder on the right hand side panel, and modify the filter text box. For example, setting the filter to ″user*″ will allow only users with the ″user″ prefix to be displayed.
5. After modifications have been made, click the ″save″ link to save the mappings. You must restart the server for the changes to take effect.

Usage scenario

The default naming security policy is to grant all users read access to CosNaming space and to grant any valid user the privileged to modify the contents of CosNaming space. The above steps may be performed to restrict user access to CosNaming space. However, great caution must be taken when changing the naming security policy. Unless an J2EE application has clearly specified its naming space access requirements, changing the default policy may result in unexpected org.omg.CORBA.NO_PERMISSION exceptions at runtime.

CORBA Naming Service users settings

Use this page to manage CORBA Naming Service users settings.

To view this administrative console page, click **Environment** > **Naming** > **CORBA Naming Service Users**.

Configuration tab

User Specifies administrative users.

The users entered must exist in the configured active user registry.

Data type	String

Role Specifies administrative users' roles.

A number of defined administrative roles provide degrees of authority needed to perform certain WebSphere administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following roles are valid: **CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete**.

The name of the four roles are the same with WebSphere Advanced Edition Version 4.0.2. However, the roles now have authority level from low to high as follows:

- **CosNamingRead**. Users who have been assigned the CosNamingRead role will be allowed to do queries of the WebSphere Name Space, such as through the JNDI ″lookup″ method. The special-subject Everyone is the default policy for this role.
- **CosNamingWrite**. Users who have been assigned the CosNamingWrite role will be allowed to do write operations such as JNDI ″bind″, ″rebind″, or ″unbind″, plus CosNamingRead operations. The special-subject AllAuthenticated is the default policy for this role.
- **CosNamingCreate**. Users who have been assigned the CosNamingCreate role will be allowed to create new objects in the Name Space through such operations as JNDI ″createSubcontext″, plus CosNamingWrite operations. The special-subject AllAuthenticated is the default policy for this role.
- **CosNamingDelete**. And finally users who have been assigned CosNamingDelete role will be able to destroy objects in the Name Space, for example using the JNDI ″destroySubcontext″ method, as well as CosNamingCreate operations. The special-subject AllAuthenticated is the default policy for this role.

Data type	String
Range	CosNamingRead, CosNamingWrite, CosNamingCreate and CosNamingDelete

CORBA Naming Service groups settings

Use this page to manage CORBA Naming Service groups settings.

To view this administrative console page, click **Environment** > **Naming** > **CORBA Naming Service Groups**.

Configuration tab

Group Identifies administrative groups.

The ALL_AUTHENTICATED group has the following role privileges: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The EVERYONE group indicates that the users in this group have CosNamingRead privileges only.

Data type	String
Range	ALL_AUTHENTICATED, EVERYONE

Role Identifies administrative users' roles.

A number of administrative roles are defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the web based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled.

Four security roles are introduced : **CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete**. The name of the four roles are the same with WebSphere Advanced Edition Version 4.0.2. However, the roles now have authority level from low to high as follows:

- **CosNamingRead**. Users who have been assigned the CosNamingRead role will be allowed to do queries of the WebSphere Name Space, such as through the JNDI ″lookup″ method. The special-subject Everyone is the default policy for this role.
- **CosNamingWrite**. Users who have been assigned the CosNamingWrite role will be allowed to do write operations such as JNDI ″bind″, ″rebind″, or ″unbind″, plus CosNamingRead operations. The special-subject ALL_AUTHENTICATED is the default policy for this role.
- **CosNamingCreate**. Users who have been assigned the CosNamingCreate role will be allowed to create new objects in the Name Space through such operations as JNDI ″createSubcontext″, plus CosNamingWrite operations. The special-subject AllAuthenticated is the default policy for this role.
- **CosNamingDelete**. And finally users who have been assigned CosNamingDelete role will be able to destroy objects in the Name Space, for example using the JNDI ″destroySubcontext″ method, as well as CosNamingCreate operations. The special-subject ALL_AUTHENTICATED is the default policy for this role.

Data type	String
Range	CosNamingRead, CosNamingWrite, CosNamingCreate and CosNamingDelete

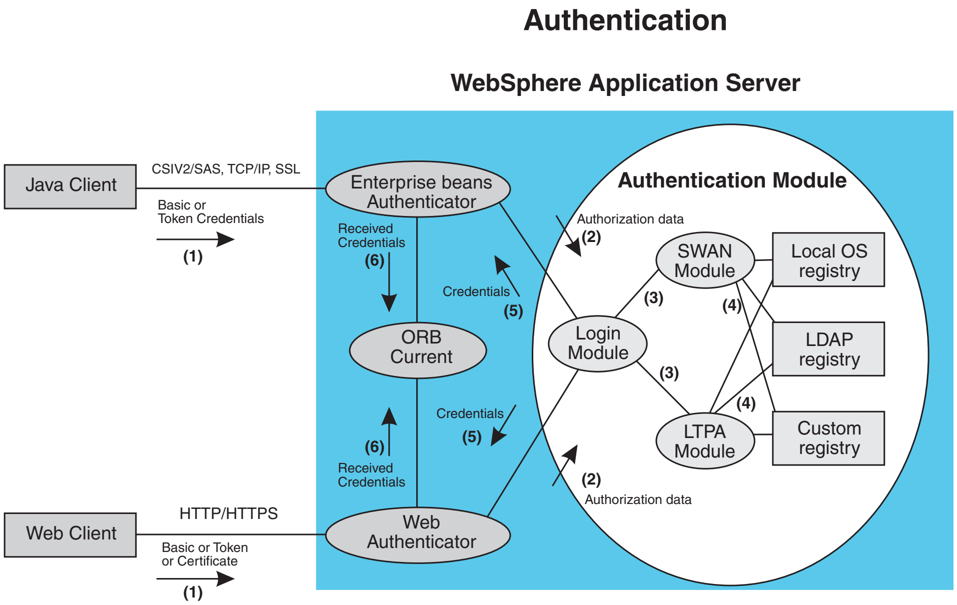
Authentication mechanisms

An authentication mechanism defines rules about security information (for example, whether a credential is forwardable to another Java process), and the format of how security information is stored in both credentials and tokens.

Authentication is the process of establishing whether a client is valid in a particular context. A client can be either an end user, a machine, or an application.

An authentication mechanism in WebSphere Application Server typically collaborates closely with a User Registry. The User Registry is the user and groups accounts repository that the authentication mechanism consults with when performing authentication. The authentication mechanism is responsible for creating a credential which is an internal product representation of successfully authenticated client user. Not all credentials are created equal. The abilities of the credential are determined by the configured authentication mechanism.

Although this product provides several authentication mechanisms, only a single *active* authentication mechanism can be configured at once. The active authentication mechanism is selected when configuring WebSphere global security.



Authentication Process

The figure demonstrates the authentication process in brief. Basically, authentication is required for enterprise beans clients and Web clients when they are accessing protected resources. Enterprise beans clients (a servlet or other enterprise beans or a pure client) sends the authentication information to Web application server using the CSIV2 or SAS protocol. Web clients use the HTTP or HTTPS protocol to send the authentication information as shown in figure 1. The authentication information can be either BasicAuth (user ID and password) or Credential Token (in case of LTPA) or Client Certificate. The Web authentication is performed by the WebAuthentication module and the EJB authentication is performed by the EJB Authentication Module which resides in CSIV2/SAS layer itself.

The Authentication module is implemented using JAAS login module. WebAuthenticator and EJBAuthenticator pass the authentication data to the login module(2) which can be either LTPA (Lightweight Third Party Authentication) or Simple WebSphere Authentication Mechanism (SWAM).

The authentication module uses Registry that is configured on the system to perform the authentication(4). There are three types of registries supported namely, LocalOS, LDAP and CustomRegistry. External registry implementation following the registry interface specified by IBM can replace either LocalOS or LDAP registry.

The login module creates a JAAS subject after authentication and stores the CORBA credential derived from the authentication data in the public credentials list of the subject. The credential is returned to WebAuthenticator or EJB Authenticator(5).

The EJB Authenticator and WebAuthenticator store the received credentials in the ORB Current(6) for the Authorization service to use it to perform further access control check.

The WebSphere Application Server provides two authentication mechanisms: Simple WebSphere Authentication Mechanism (SWAM) and Lightweight Third Party Authentication (LTPA). These two authentication mechanisms differ primarily in the distributed security features each supports.

Configuring authentication mechanisms

Configure authentication mechanisms through the Security Center in the administrative console. See Lightweight Third Party Authentication configuration settings article.

Steps for this task

1. Configure authentication mechanisms by selecting Authentication Mechanisms under Security in the administrative console.
   - If you are using **SWAM**, there is no setup needed. Follow the instructions in Configuring LTPA to set up LTPA. If you choose LTPA, Configure Single Sign-On (SSO) for most situations. If Trust Association is required follow the steps in Configuring a trust association interceptor.

Simple WebSphere Application Server Authentication Mechanism (SWAM)

The SWAM authentication mechanism is intended for simple, non-distributed, single application server type runtime environments. The single application server restriction is due to the fact that SWAM does not support *forwardable* credentials. What this means is that if a servlet or enterprise bean in application server process 1, invokes a remote method on an enterprise bean living in another application server process 2, the identity of the caller identity in process 1 is not transmitted to server process 2. What is transmitted is an unauthenticated credential, which, depending on the security permissions configured on the EJB methods, may cause authorization failures.

Since SWAM is intended for a single application server process, single-sign-on (SSO) is not supported.

The SWAM authentication mechanism is suitable for simple environments, software development environments, or other environments that do not require a distributed security solution.

Configuring Lightweight Third Party Authentication

The following steps are needed to perform this task initially when setting up security for the first time.

Steps for this task

1. Click **Security** > **Authentication mechanisms** > **LTPA** in the Navigation panel on the left.
2. Enter the password and confirm it in the password fields. This password is used to encrypt and decrypt the LTPA keys during export and import of the keys. This password needs to be remembered as it needs to be entered again when the keys from this cell are exported to another cell.
3. Enter a positive integer value in the Timeout field. This timeout refers to how long a LTPA token is valid in minutes. The token contains this expiration time so that any server that receives this token can make sure that this token is valid

before proceeding further. When the token expires, the user will be prompted to login. An optimal value for this depends on your configuration. The default value is 30 minutes.

4. Click **Apply** or **OK**. The LTPA configuration is now set. You should not generate the LTPA keys in this step becauese they will be automatically generated later. Proceed with the rest of the steps required to enabled security, starting with SSO (if SSO is required).
5. Complete the information in the Global Security panel and press OK. When **OK** or **Apply** is clicked in the Global Security panel the LTPA keys are generated automatically the first time, and therefore, you should not generate the keys manually.

Results

Sets LTPA configuration. Generates new set of LTPA keys. Export and Import LTPA keys.

What to do next

1. Generate key files.
2. Export key files.
3. Import key files.
4. If you enabling security, make sure you complete the remaining steps starting with enabling SSO.
5. If you have generated a new set of keys or imported a new set of keys, make sure the keys are saved by clicking on the save at the top of the panel. Since LTPA authentication uses time sensitive tokens, ensure the time, date, and timezone is synchronized among all product servers that are participating in the protection domain. If the clock skew is too high between servers, the LTPA token will appear prematurely expired and cause authentication or validation failures.

Configuring Lightweight Third Party Authentication keys:

Generating keys: LTPA keys are automatically generated when a password change is detected. The first time you set the LTPA password, (as part of enabling security) the LTPA keys are automatically generated once the **OK** or the **Apply** is clicked in the LTPA panel. You do ***not*** have to click **Generate Keys** in this situation. Complete the following steps in the administrative console to generate a new set of LTPA keys.

Steps for this task

1. Make sure all the WebSphere Application Server processes are running (Cell, Nodes and all the Application Servers). If any of the servers are down at the time of generating the keys, they would not be able to come up later because they would contain old keys. You would have to copy the new set of keys to these servers to bring them back up.
2. Click **Security** > **Authentication mechanisms** > **LTPA** in the navigation panel on the left.
3. Click **Generate Keys** if you want to use the existing password. This action generates a new set of keys that will be encrypted with the same password as the old set of keys.

Note: Regardless of the password change, a new set of keys are generated when **Generate Keys** is clicked. Since these new set of keys are not propagated to run time unless saved, save the files immediatley.

4. Enter the new password and confirm it, to use a new password to generate keys. Click **OK** or **Apply**.

   A new set of keys are generated. A message indicating that a new set of keys are generated shows up on the console. Do not click **Generate Keys**. These new keys will be propagated to the run time once you save.
5. Click **Save** to save the keys.

   Once a new set of keys are generated and saved, the key propagation is dynamic. All the processes running at that time (cell, node agents, application servers) will be updated with the new set of keys. The next sections describe the process of exporting and importing the keys.

Exporting keys: To support single sign on (SSO) in WebSphere Application Server across multiple WebSphere Application Server domains (cells) the LTPA keys and the password should be shared among the domains. The times on the domains should be similar to prevent the tokens from appearing as expired between the cells. The **Export Keys** button can be used to export the LTPA keys to other domains or cells. Complete the following steps in the administrative console to export key files for LTPA.

Steps for this task

1. Click **Security** > **Authentication mechanisms** > **LTPA** in the Navigation panel on the left.
2. In the **Key File Name** field, enter the full path of a file where the keys need to be stored. The file should have write permissions.
3. Click **Save** to save the file.
4. Click **Export Keys**. A file will be created with the LTPA keys in it.

   Exporting keys will fail if a new set of keys was generated or imported and not saved prior to exporting. To avoid failure, make sure you save the new set of keys (if any) prior to exporting them.
5. Click **Save** to save the configuration.

Importing keys: To support single sign on (SSO) in WebSphere Application Server across multiple WebSphere Application Server domains (cells) the LTPA keys and the password should be shared among the domains. The **Import Keys** button can be used to import the LTPA keys from other domains. The key files should have been exported from one of the cells involved into a file. Complete the following steps in the administrative console to import key files for LTPA.

Importing keys is a dynamic operation. All the servers that are running at this time will be updated with the new set of keys and any back-level tokens signed with the back-level keys will fail validation and the user will be prompted to login again.

Steps for this task

1. Click **Security** > **Authentication mechanisms** > **LTPA** in the Navigation panel on the left.
2. Change the password in the password fields to match the password in the cell that you are importing the keys from.
3. Click **Save** to save the new set of keys in the repository.

This is an important step to be completed before importing the keys. If the password and the keys do not match, the servers will fail to come up. In that case, you would have to turn off security and complete this process again.

4. In the **Key File Name** field, enter the full path of a file where the keys need to be stored. The file should have read permissions.
5. Click **Import Keys**. The keys are now imported into the system.
6. Click **Save** to save the new set of keys in the repository. It is important to save the new set of keys to match the new password so that there will not be any problems starting the servers up later.

Lightweight Third Party Authentication settings: Use this page to configure Lightweight Third Party Authentication (LTPA) settings.

To view this administrative console page, click **Security** > **Authentication Mechanisms** > **LTPA**.

If you are configuring security for the first time only the password is required. Once the password is entered click **Apply**. Click **Single Sign On** (SSO) and enter the domain name. Make sure that SSO is enabled. Click **Apply**. To complete the security setup ensure that the appropriate registry is setup and click **Apply** from the Global Security panel. When security is enabled and any of these properties change, go to the Global Security panel and click **Apply** to validate the changes.

Generate Keys: Specifies whether the server will generate new LTPA keys.

When security is turned on for the first time with LTPA as the authentication mechanism the LTPA keys are automatically generated with the password entered in the panel. If you need a new set of keys to be generated using the previously set password click **Generate Keys**. If a new password is used do not click this button. Once the new password is entered and **OK** or **Apply** is clicked, a new set of keys are generated. *Whenever a new set of keys are generated, they will not be used until you save them.*

Import Keys: Specifies whether the server will import new LTPA keys.

To support Single Sign-On (SSO) in the WebSphere product across multiple WebSphere domains (cells), share the LTPA keys and the password among the domains. The **Import Keys** option can be used to import the LTPA keys from other domains. The LTPA keys should have been previously exported from one of the cells to a file. In order to import a new set of LTPA keys, enter the appropriate password and the file name where the LTPA keys are located. Click **Import Keys** only and ***do not click*** **OK** or **Apply**.

Export Keys: Specifies whether the server will export LTPA keys.

To support single sign on (SSO) in the WebSphere product across multiple WebSphere domains (cells), share the LTPA keys and the password among the domains. The **Export Keys** option can be used to export the LTPA keys to other domains.

To export the LTPA keys, make sure that the system is running with security enabled, and using LTPA. Enter the file name in **Key File Name** field and click **Export Keys**. The encrypted keys will be stored in the file specified.

Password: Specifies the password to encrypt and decrypt the LTPA keys. Use this password when importing these keys into other WebSphere Application Server administrative domain configurations (if any) and when configuring SSO for Domino server.

Once the keys are generated or imported they are used to encrypt and decrypt the LTPA token. Whenever the password is changed, a new set of LTPA keys are automatically generated when you click **OK** or **Apply**. These new set of keys will be used only when you save.

Data type	String

Confirm Password: Specifies the confirmed password used to encrypt and decrypt the LTPA keys.

Use this password when importing these keys into other WebSphere Application Server administrative domain configurations (if any) and when configuring SSO for Domino Server.

Data type	String

Timeout: Specifies the time period in minutes at which an LTPA token will expire. Ensure this time period is longer than cache timeout configured in the Global Security panel.

Data type	Integer
Units	Minutes
Default	120

Key File Name: Specifies the name of the file used when importing or exporting keys.

Enter the full path of the file name to import or export keys. When importing keys make sure that the file can be read. When exporting keys, make sure that file can be written to. Once the required fields are completed, click **Export Keys** or **Import Keys**.

Data type	String

Trust Associations

Trust Association enables the integration of IBM WebSphere Application Server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials passed by the proxy server.

Demand for such an integrated configuration has become more and more compelling especially when a single product cannot meet all customer needs or when migration is not a viable solution. This document gives a conceptual background behind the approach.

There is a growing demand to provide customers with a trust association solution between IBM WebSphere Application Server and other Web authentication servers that act as reverse proxy security server (IBM Tivoli Security Manager - WebSeal for Policy Director, ECommerce Server) as an entry point to all service requests

(See Figure 1). This implementation design intends to have the proxy server as the only exposed entry point. It authenticates all requests that come in and provides coarse, granularity junction point authorization.

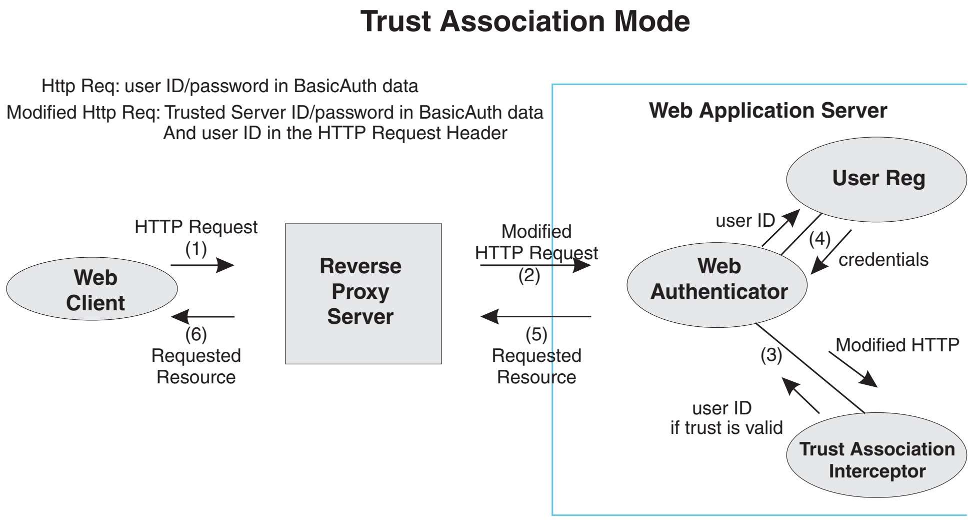
In this setup, the WebSphere Application Server is used as a back-end server to further exploit its fine-grained access control. The reverse proxy server passes to WebSphere Application Server the HTTP request that includes the credentials of the authenticated user. WebSphere Application Server then uses these credentials to authorize the request.

Trust Association model

The idea that WebSphere Application Server can support trust association implies that the product application security recognizes and processes HTTP requests received from a reverse proxy server. WebSphere Application Server and the proxy server engage in a contract in which the product gives its full trust to the proxy server and the proxy server applies its authentication policies on every Web request that is dispatched to WebSphere Application Server. This trust is validated by the interceptors that reside in the product environment for every request received. The method of validation is agreed upon by the proxy server and the interceptor.

Running in trust association mode does not prohibit WebSphere Application Server from accepting requests that did not pass through the proxy server. In this case, no interceptor is needed for validating trust. It should be possible, however, to configure WebSphere Application Server to strictly require all HTTP requests to go through a reverse proxy server, in which case all requests not coming from a proxy server are immediately denied by WebSphere Application Server.

Trust association



IBM WebSphere Application Server—WebSeal Integration

The integration of WebSeal and WebSphere Application Server security is achieved by placing WebSeal at the front-end as a reverse proxy server, see Figure 2. From a WebSeal management perspective, a junction is created with WebSeal on one end, and the product Web server on the other end. A junction is a logical connection created to establish a path from WebSeal to another server.

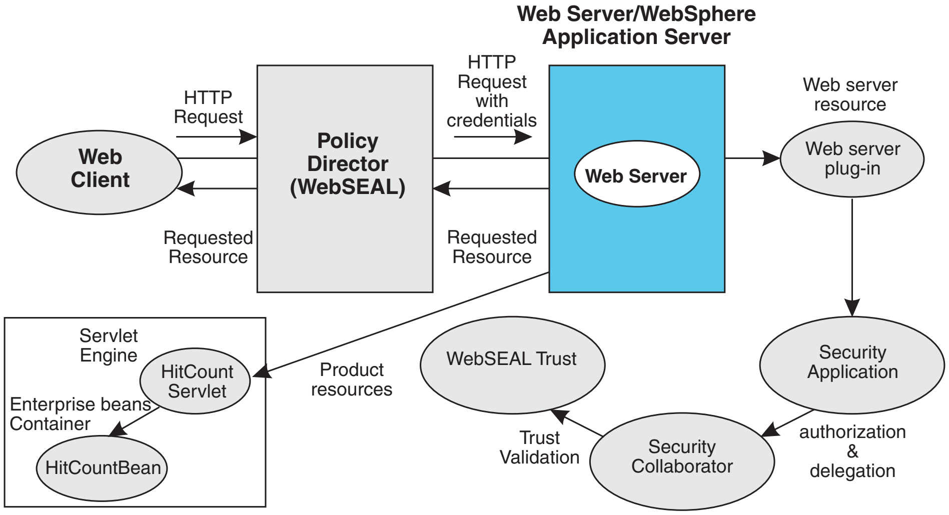
In this setup, a request for Web resources stored in protected domain of the product is submitted to WebSeal where it is authenticated against the WebSeal security realm. If the requesting user has access to the junction, the request is transmitted to the WebSphere Application Server HTTP server through the junction, and then to the application server.

Meanwhile, the WebSphere Application Server validates every request that comes through the junction to ensure that the source is a trusted party. This process is referenced as *validating the trust* and it is performed by a WebSeal product-designated interceptor. If the validation is successful, the WebSphere Application Server authorizes the request by checking whether the client user has the required permissions to access the Web resource. If so, the Web resource is delivered to WebSeal, through the Web server, which then gives it to the client user.

WebSeal

The Policy Director delegates all Web requests to its Web component, the WebSeal server. One of the major functions of the server is to perform authentication of the requesting user. The WebSeal server consults an LDAP directory. It can also map the original user ID to another user ID, such as when global single sign-on (GSO) is used.

Policy Director (WebSeal)

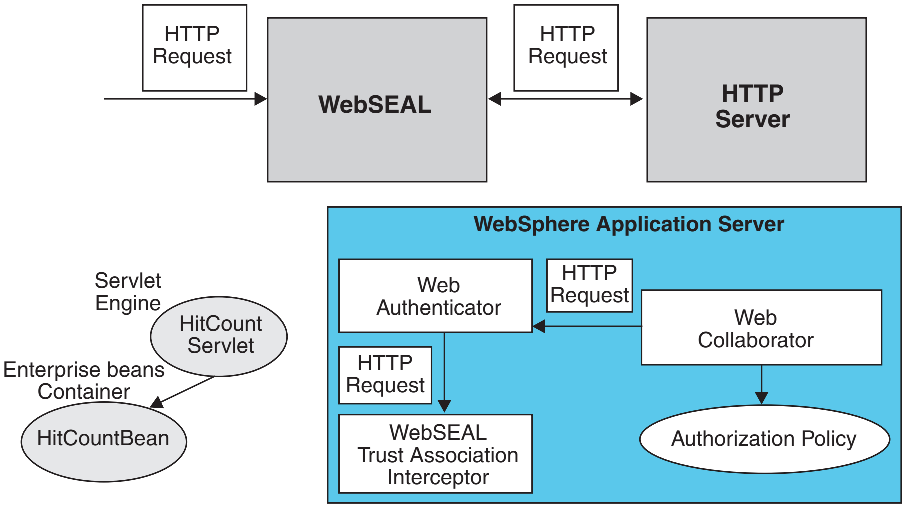


For successful authentication, the server plays the role of a client to WebSphere Application Server when channeling the request. As such, it would need its own user ID and password to identify itself to WebSphere Application Server. This identity must be valid in the security realm of WebSphere Application Server. Thus, WebSeal replaces the Basic Auth information in the HTTP request with its own user ID and password. In addition, WebSphere Application Server would need to know the user ID of the requesting client so it can base its authorization decision from this user ID, and not from a WebSeal user ID. This information is transmitted through the HTTP request by creating a header called `iv-user` with the client user ID as its value.

HTTP Server

The junction created in WebSeal must get to the HTTP server that serves as the product front end. The HTTP Server is shielded however, from knowing that trust association is used. As far as it is concerned, WebSeal is just another HTTP client, and as part of its normal routines, it sends the HTTP request to the product. The only requirement on the HTTP Server is an SSL configuration using server authentication only. This requirement protects the requests that flow within the junction.

HTTP Server



Web Collaborator

When Trust Association is enabled, the Web collaborator manages the interceptors that are configured in the system. It loads and initializes these interceptors when you restart your servers. When a request is passed to WebSphere Application Server by the Web server, the Web collaborator will eventually receives the request for a security check. Two things must be performed:

1. The request must be authenticated
2. The request must be authorized

The Web authenticator is called to authenticate the request by passing the HTTP request. If successful, a good credential record is returned by the authenticator, which the web collaborator uses to base its authorization on the requested resource. If the authorization succeeds, the Web collaborator indicates to WebSphere Application Server that the security check has succeeded and the requested resource can be served.

Web Authenticator

The Web authenticator is asked by the Web collaborator to authenticate a given HTTP request. Knowing that trust association is enabled, the task of the Web authenticator is to find the appropriate trust association interceptor to direct the request for processing. It does this by querying every available interceptor. If no target interceptor can be found, the Web authenticator processes the request as though trust association is not enabled.

For an HTTP request sent by WebSeal, the WebSeal trust association interceptor replies with a positive response to the Web authenticator. Subsequently, the interceptor is asked to validate its trust association with the WebSeal server and retrieve the user ID of the original user client.

Trust Association Interceptor feature

The intent of the trust association interceptor feature is to have reverse proxy security servers (RPSS) exist as the exposed entry points to perform authentication and coarse-grained authorization, while the WebSphere Application Server enforces further fine-grained access control. Trust associations improve security by reducing the scope and risk of exposure.

In a typical e-business infrastructure, a distributed environment of a company consists of Web application servers, Web servers, legacy systems and one or more RPSS, such as the WebSeal product from Tivoli. Such reverse proxy servers, front-end security servers or security plug-ins registered within Web servers, guard the HTTP access requests to the Web servers and Web application servers. While protecting access to the Uniform Resource Identifiers (URIs), these RPSS perform authentication, coarse-grained authorization and route the request to the target application server. The credential formed because of the user authentication passing as a part of the request. Unfortunately, most back-end services, including WebSphere Application Server, do not understand the format of the credential information passed to them by the RPSS.

Using the Trust Association Interceptor feature:

The following further describes the benefits of the Trust Association Interceptor (TAI) feature:

- RPSS can authenticate WebSphere Application Server users up front and send credential information about the authenticated user to the product so that the product can *trust* the RPSS to have performed authentication and not prompt the end user for authentication data another time. The strength of the trust relationship between RPSS and the product is based on the criteria of trust association that is particular to a RPSS and enforced through the TAI implementation. This level of trust might need relaxing based on the environment, but the WebSphere Application Server user should be aware of the vulnerabilities in cases where the RPSS is not trusted, based on a security technology.
- The end user credentials most likely are sent in a special format as part of the Hypertext Transfer Protocol (HTTP) headers in the case of RPSS authentication. It may be a special header or a cookie. What data is passed is implementation specific, and the TAI feature considers this fact and accommodates the idea. The TAI implementation works with the credential data and returns a string that represents the end user that WebSphere Application Server usees to enforce security policies.

Configuring trust association interceptors

The following steps are needed to perform this task initially when setting up security for the first time. These steps are required to use either WebSEAL Trust Association Interceptor or your own trust association interceptor with a reverse proxy security server.

Steps for this task

1. Click **Security** > **Authentication mechanisms** > **LTPA** in the left navigation panel. Click **Trust Association** in additional properties.
2. Click the **Enable Trust Association** checkbox.

3. Click **Interceptors**.
4. The following Interceptor is the default value. If you are using WebSeal Interceptor then check mark the following com.ibm.ws.security.web.WebSealTrustAssociationInterceptor.
   a. Otherwise click **New** to add the Interceptors.
   b. For each Interceptor click **Additional Properties** to enter the property name and value pairs.
5. The name and value pairs for WebSeal follow:
   - **com.ibm.websphere.security.trustassociation.types—**WebSeal
   - **com.ibm.websphere.security.webseal.loginId—**the ID of the WebSeal server
   - **com.ibm.websphere.security.webseal.id—**iv-user. This is a special header field that is sent by WebSeal with the request to the WebSphere Application Server

   It is unnecessary to specify the host name of the machine where a WebSeal server is running or the port where WebSEAL receives the user requests. If you want to restrict the hostname and or the port where a user can send requests, then add the following properties:
   - **com.ibm.websphere.security.webseal.hostnames—**the hostname of the machine where WebSEAL server is running
   - **com.ibm.websphere.security.webseal.ports—**the port where WebSEAL server receives user requests
6. Click **OK**.

Results

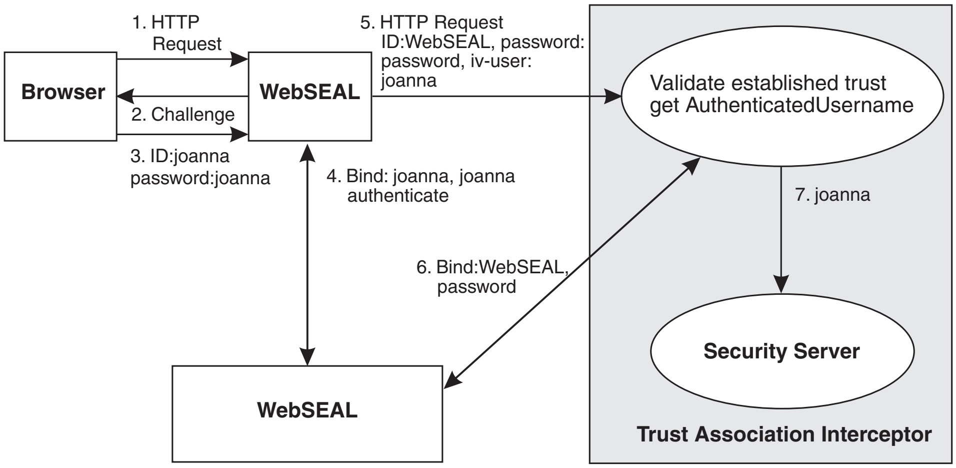
Enables Trust Association.

Usage scenario

A typical scenario where the trust association interceptor (TAI) is used is better understood based on an environment where IBM Tivoli WebSeal product is deployed and used with WebSphere Application Server. For WebSeal, there is an implementation of the TAI already provided with the product. The following steps outline the typical flow of an HTTP request for a secured WebSphere Application Server resource authenticated by WebSeal, through a Web trust association.
1. The browser makes a request for a secured WebSphere resource.
2. WebSeal sends back a challenge, either an HTTP Basic authentication or form-based challenge.
3. User name and password are supplied. WebSeal authenticates the user.
4. The modified request is forwarded by WebSeal to the WebSphere Application Server.
5. The plug-in (TAI) establishes that the WebSphere Application Server trusts the WebSeal server. It uses the validateEstablishedTrust method to do this.
6. The plug-in extracts the end-user name from the `iv-user` header field and passes it to the WebSphere Application Server to handle authorization.

Note: WebSeal 3.9 and later does not flow the user ID and password to the server. Trust is based on mutual SSL established between WebSeal and the WebSphere Application Server. Therefore steps 5 and 6 do not apply to WebSeal 3.9 and later.

Web Trust Association authentication flow


1. HTTP Request
Browser
WebSEAL
2. Challenge
3. ID:joanna password:joanna
5. HTTP Request ID:WebSEAL, password: password, iv-user: joanna
Validate established trust get AuthenticatedUsername
4. Bind: joanna, joanna authenticate
7. joanna
6. Bind:WebSEAL, password
Security Server
WebSEAL
Trust Association Interceptor

What to do next

1. Ensure that you complete the remaining steps if you enable security.
2. Save, stop and restart all the product servers (cell, nodes and all the application servers) for the changes to take effect.

Trust association settings: Use this page to configure trust association settings.

To view this administrative console page, click **Security Center** > **Authentication Mechanisms** > **LTPA** > **Trust Association**.

When security is enabled and any of these properties change, go to the Global Security panel and click **Apply** to validate the changes.

Enabled: Specifies whether trust association is enabled.

Data type	Boolean
Default	Disable
Range	Enable or Disable

Trust association interceptor settings: Use this page to specify trust information for reverse security proxy servers.

To view this administrative console page, click **Security** > **Authentication Mechanisms** > **LTPA** > **Trust Association** > **Interceptors**.

When security is enabled and any of these properties are changed, go to the Global Security panel and click **Apply** to validate the changes.

Interceptor Class Name: Specifies the trust association interceptor class name.

Data type	String
Default	com.ibm.ws.security.web.WebSealTrustAssociationInterceptor

Single Sign-On

With Single sign-on (SSO) support, Web users can authenticate once when accessing both WebSphere Application Server resources, such as HTML, JSP files, servlets, enterprise beans, and Domino resources, such as documents in a Domino database, or accessing resources in multiple WebSphere domains.

Web users can authenticate once to a WebSphere Application Server or Domino server and then access any other WebSphere Application Servers or Domino servers in the same DNS domain that are enabled for single sign-on (SSO) without logging on again. This authentication is accomplished by configuring the WebSphere Application Servers and the Domino servers to share authentication information.

Enable SSO among WebSphere Application Servers by you configuring SSO for WebSphere Application Server. To enable SSO between WebSphere Application Servers and Domino servers, you must configure SSO for both WebSphere Application Server and for Domino.

Prerequisites and conditions: To take advantage of support for single sign-on between WebSphere Application Servers or between WebSphere Application Server and Domino, applications must meet the following prerequisites and conditions:

- Ensure all servers are configured as part of the same DNS domain. For example, if the DNS domain is specified as *mycompany.com*, then SSO will be effective with any Domino or WebSphere Application Server on a host that is part of the *mycompany.com* domain, for example, *a.mycompany.com* and *b.mycompany.com*.
- Ensure all servers share the same user registry. This registry can be either a supported LDAP directory server or, if SSO is configured between two WebSphere Application Servers, a custom user registry. Domino does not support the use of custom registries, but you can use a Domino-supported registry as a custom registry within WebSphere Application Server. For more information on custom registries, see *Introduction to custom registries*.

  You can use a Domino Directory (configured for LDAP access) or other LDAP directory for the user registry. The LDAP directory product must be supported by WebSphere Application Server. Supported products include both Domino and all IBM SecureWay LDAP directory servers. Regardless of the choice to use an LDAP or custom registry, the SSO configuration is the same. The difference is in the configuration of the registry.
- Define all users in a single LDAP directory. Using LDAP referrals to connect more than one directory together is not supported. Using multiple Domino directory assistance documents to access multiple directories is not supported.
- Enable HTTP cookies in browsers because the authentication information that is generated by the server is transported to the browser in a cookie. The cookie is then used to propagate the user's authentication information to other servers, exempting the user from entering the authentication information for every request to a different server.
- For Domino:
  - Domino R5.0.6a for iSeries 400 (or later) and Domino R5.0.5 (or later) for other platforms are supported.
  - A Lotus Notes client R5.0.5 (or later) is required for configuring the Domino server for SSO.
  - You can share authentication information across multiple Domino domains.
- For WebSphere Application Server:
  - WebSphere Application Server V3.5 (or later) for all platforms is supported.
  - You can use any HTTP Web server supported by WebSphere Application Server.

- You can share authentication information across multiple product administrative domains.
- Basic authentication (user ID and password) using the basic and form-login mechanisms is supported.
- By default WebSphere Application Server does a case sensitive comparison for authorization. This implies that the a user who is authenticated by Domino should match exactly the entry (including the base distinguished name) in the WebSphere Application Server authorization table. If case sensitivity should not be considered for the authorization, the **Ignore Case** property should be enabled in the LDAP user registry settings.

Configuring Single Sign On (SSO)

With single sign-on (SSO) support, Web users can authenticate once when accessing Web resources across multiple WebSphere Application Servers. This authentication is supported only when LTPA is the authentication mechanism. SSO uses HTTP cookies to achieve this functionality. When SSO is enabled, a cookie is created with the LTPA token in it. When the user accesses some other Web resource in any other WebSphere Application Servers process in the same DNS domain, the cookie is sent in the request. The LTPA token is then extracted from the cookie and is validated. If the request is between different cells of WebSphere Application Servers, the LTPA keys and the user registry should be shared between the cells for SSO to work.

The LTPA authentication mechanism requires that SSO is enabled if any of the Web applications have form login as the authentication method.

The following steps are needed when setting up security for the first time.

Steps for this task

1. Click **Security** > **Authentication mechanisms** > **LTPA** in the Navigation panel on the left. Click SSO in the additional properties section.
2. SSO is enabled by default. If it has been disabled, click **Enable**.
3. Enable the Requires SSL field if all the requests are expected to come over HTTPS.
4. Enter the domain name. This is the DNS domain name where SSO is effective since the cookie is sent for all the servers in this domain only.

   For example, if the domain is ibm.com, SSO works between the domains austin.ibm.com, raleigh.ibm.com and not austin.otherCompany.com.

   Note: The domain field is optional, and, if left blank, the Web browser defaults to the domain name of the SSO cookie to the WebSphere Application Server that created it. In this case, SSO is only be valid for the server that created the cookie. This behavior may be desirable when multiple virtual hosts have been defined and they each need to have their own or separate domain specified in the SSO cookie.
5. Click **OK**.

Usage scenario

This step is required to set up SSO configuration. Form login mechanism for Web applications require that SSO is enabled.

What to do next

1. If you enabling security, make sure you complete the remaining steps.

2. For the changes to be effective, you need to save, stop and restart all the WAS servers (Cell, Nodes and all the WebSphere Application Servers).

Single Sign-on settings: Use this page to set the configuration values for Single Sign-on (SSO).

To view this administrative console page, click **Security** > **Authentication Mechanisms** > **LTPA** > **Single Sign-On**.

Requires SSL: Specifies that Single Sign-On function is enabled only when requests are over HTTPS Secure Socket Layer (SSL) connections.

Data type	Boolean
Default	Disable
Range	Enable or Disable

Domain Name: Specifies the domain name (.ibm.com, for example) for all Single Sign-on hosts.

If no value is specified, the user's Web browser will default the value to the host name where the Web application is running. This restricts the HTTP cookie (generated for SSO purposes) only to the host that originated it. Restricting the HTTP cookie may be undesirable if there is more than one host participating in the SSO domain. Leaving the domain name attribute empty is only desirable if multiple virtual hosts with different domain names are running on the same physical host. Leaving this field empty allows your Web browser to default the domain name to each different virtual host. If a domain name is explicitly specified in this field, then that value is used for all virtual host, and thereby restricting them to a single domain, which may be undesirable in some situations.

If a domain name is explicitly specified, then all URLs used to access protected Web resources should contain the server DNS hostname. For example, once global security has been configured for LTPA and an explicit SSO domain name has been specified, then the administrative console can be accessed with the following URL: **http://yourhost.austin.ibm.com:9090/admin**, where *yourhost.austin.ibm.com* should be replaced with your server DNS hostname.

Data type	String

Enabled: Specifies that the Single Sign-on function is enabled.

Web applications that use J2EE FormLogin style login pages (such as the WebSphere Application Server administrative console) require single sign-on (SSO) to be enabled. You should only disable SSO for certain advanced configurations where LTPA SSO type cookies are not required.

Data type	Boolean
Default	Enabled
Range	Enabled or Disabled

Troubleshooting SSO configurations:

Before you begin

This article describes common problems in configuring single sign-on between WebSphere Application Server and Domino and suggests possible solutions. The problems include the following:

Steps for this task

1. Failure to save the Domino Web SSO Configuration document

   The client must be able to find Domino Server documents for the participating SSO Domino servers. The Web SSO Configuration document is encrypted for the servers that you specify, so the home server indicated by the client's location record must point to a server in the Domino domain where the participating servers reside. This ensures that lookups can find the public keys of the servers.

   If you receive a message that states that one or more of the participating Domino servers cannot be found, then those servers will not be able to decrypt the Web SSO Configuration document or perform SSO.

   When the Web SSO Configuration document is saved, the status bar indicates how many public keys were used to encrypt the document by finding the listed servers, authors, and administrators on the document.

2. Domino server console fails to load the Web SSO Configuration document upon Domino HTTP server start-up

   During configuration of SSO, the Server document is configured for Multi-Server in the Session Authentication field. Therefore, the Domino HTTP server tries to find and load a Web SSO Configuration document during startup. The Domino server console reports the following if a valid document is found and decrypted: `HTTP: Successfully loaded Web SSO Configuration`.

   If a server cannot load the Web SSO Configuration document, SSO does not work. Such a server reports the following message: `HTTP: Error Loading Web SSO configuration`. Reverting to single-server session authentication.

   Make sure that there is only one Web SSO Configuration document in the Web Configurations view of the Domino Directory and in the $WebSSOConfigs hidden view. You cannot create more than one, but additional documents can be inserted during replication.

   If there is only one Web SSO Configuration document, another condition that can elicit the same error message is that the public key of the Server document does not match the public key in the ID file. In this case, attempts to decrypt the Web SSO Configuration document fail and the error message is generated.

   This situation can occur when the ID file is created multiple times but the Server document is not updated correctly. Usually, there is an error message displayed on the Domino Server Console that states that the public key does not match the server ID. If this happens, then SSO does not work because the document is encrypted with a public key for which the server does not possess the corresponding private key.

   To correct a key-mismatch problem, do the following:

   a. Copy the public key from the server ID file and paste it into the Server document.
   b. Re-create the Web SSO Configuration document.

3. Authentication fails when accessing a protected resource.

   If a Web user is repeatedly prompted for a user ID and password, SSO is not working because either the Domino or WebSphere security server is not able to authenticate the user with the LDAP server. Check the following possibilities:

- Verify that the LDAP server can be accessed from the Domino server machine. Use the **TCP/IP ping** utility to verify TCP/IP connectivity and that the host machine is running.
- Verify that the LDAP user is defined in the LDAP directory. Use the **ldapsearch** utility to confirm that the user ID exists and that the password is correct. For example, the following command, entered as a single line, can be run from the OS/400 Qshell, a UNIX shell, or a Windows DOS prompt:

  ```
  % ldapsearch -D "cn=John Doe, ou=Rochester, o=IBM, c=US" -w mypassword
  -h myhost.mycompany.com -p 389
  -b "ou=Rochester, o=IBM, c=US" (objectclass=*)
  ```

  (The percent character (%) indicates the prompt and is not part of the command.) A list of directory entries is expected. Possible error conditions and causes follow:
  - No such object: This error indicates that the directory entry referenced by either the user's DN value, which is specified after the -D option, or the base DN value, which is specified after the -b option, does not exist.
  - Invalid credentials: This error indicates that the password is invalid.
  - Can't contact LDAP server: This error means that the host name or port specified for the server is invalid or that the LDAP server is not running.
  - An empty list means that the base directory specified by the -b option does not contain any directory entries.
- If you are using the user's short name (or user ID) instead of the Distinguished Name, ensure that the directory entry is configured with the short name. For a Domino Directory, this is the **Short name/UserID** field of the Person document. For other LDAP directories, this is the userid property of the directory entry.
- If Domino authentication fails when using an LDAP directory other than Domino Directory, verify the configuration settings of the LDAP server in the Directory Assistance document in the Directory Assistance database. Also verify that the Server document refers to the correct Directory Assistance document. The following LDAP values specified in the Directory Assistance document must match the values specified for the user registry in the WebSphere administrative domain:
  - Domain name
  - LDAP host name
  - LDAP port
  - Base DN

  Additionally, the rules defined in the Directory Assistance document must refer to the base DN of the directory containing the directory entries of the users.

  You can trace the Domino server's requests to the LDAP server by adding the following line to the server's notes.ini file:

  ```
  webauth_verbose_trace=1
  ```

  After restarting the Domino server, trace messages are displayed in the Domino server's console as Web users attempt to authenticate to the Domino server.

4. Authorization fails when accessing a protected resource.

   After authenticating successfully, if a Web user is shown an authorization error message, security is not configured correctly. Check the following possibilities:

- For Domino databases, verify that the user is defined in the access-control settings for the database. Refer to the Domino Administrative documentation for the correct way to specify the user's DN. For example, for the DN `cn=John Doe, ou=Rochester, o=IBM, c=US`, the value on the access-control list must be set as `John Doe/Rochester/IBM/US`.
- For resources protected by WebSphere Application Server, verify that the security permissions are set correctly.
  - If granting permissions to selected groups, make sure that the user attempting to access the resource is a member of the group. For example, you can verify the members of the groups by using the following URL to display the directory contents:
    `Ldap://myhost.mycompany.com:389/ou=Rochester, o=IBM, c=US??sub`
  - If you have changed the LDAP configuration information (host, port, and base DN) in a WebSphere Application Server administrative domain since the permissions were set, the existing permissions are probably invalid and need to be re-created.

5. SSO fails when accessing protected resources.

   If a Web user is prompted to authenticate each time he or she accesses a resource, SSO is not configured correctly. Check the following possibilities:

   a. Both WebSphere Application Server and Domino must be configured to use the same LDAP directory. The HTTP cookie used for SSO stores the full Distinguished Name (DN) of the user, for example, `cn=John Doe, ou=Rochester, o=IBM, c=US`, and the DNS domain.

   b. If the Domino Directory is being used, Web users must be defined by hierarchical names. For example, update the **User name** field in the Person document to include names of this format as the first value: **John Doe/Rochester/IBM/US**.

   c. URLs issued to Domino and WebSphere application servers configured for SSO must specify the full DNS server name, not just the host name or TCP/IP address. For browsers to be able to send cookies to a group of servers, the DNS domain must be included in the cookie, and the DNS domain in the cookie must match the URL. (This is why cookies cannot be used across TCP/IP domains.)

   d. Domino and WebSphere Application Server must be configured to use the same DNS domain. Verify that the DNS domain value is exactly the same, including capitalization. The DNS domain value can be found on the Configure Global Security Settings panel of the WebSphere administrative console and in the Web SSO Configuration document of a Domino server. If you make a change to the Domino Web SSO Configuration document, replicate the modified document to all Domino servers participating in SSO.

   e. Clustered Domino servers must have the host name populated with the full DNS server name in the Server document for Domino ICM (Internet Cluster Manager) to redirect to cluster members using SSO. If this field is not populated, by default, ICM redirects URLs to clustered Web servers by using only the host name. It cannot send the SSO cookie because the DNS domain is not included in the URL. To correct the problem, do the following:

      1) Edit the Server document.

      2) Select the **Internet Protocols — > HTTP** tab.

      3) Enter the server's full DNS name in the **Host names** field.

   f. If a port value for an LDAP server was specified for a WebSphere Application Server administrative domain, the Domino Web SSO

Configuration document must be edited and a backslash character (\)must be inserted into the value of the **LDAP Realm** field before the colon character (:). For example, replace myhost.mycompany.com:389 with myhost.mycompany.com\:389.

User registries

A user registry is where the users and groups information reside. WebSphere Application Server uses a user registry to authenticate a user and also to retrieve information about users and groups to perform all security related functions including authentication and authorization. Implementation is provided to support multiple operating system based user registries (Windows systems, AIX, Solaris, Linux) and most of the major LDAP based user registries. Also, one can use the custom LDAP feature to support any LDAP server by setting up the correct configuration (user and group filters). However, support is not extended to these custom LDAP servers since there are many possibilities that cannot be tested.

In addition to LocalOS and LDAP registries, WebSphere Application Server also provides a plug-in to support any registry by using the Custom Registry feature (also referred as Custom User Registry). The Custom Registry feature allows the use of any user registry that is not implemented by WebSphere Application Server. The possibilities are endless in that any registry can be made to work in the product environment by implementing an interface called the UserRegistry interface. This is very helpful in situations where the current user and group information exists in some other formats (for example, a database) and cannot be moved to LocalOS or LDAP. In such a case one should implement the UserRegistry interface so that WebSphere Application Server can use the existing registry for all the security related operations. Implementing a custom registry is a software implementation effort and it is expected that the implementation does not depend on other WebSphere Application Server resources (for example, data sources) for its operation.

Note: Even though the product supports different types of user registries only one can be active. This active registry is shared by all the product servers (processes). If the product processes in one cell need to communicate with other product processes in other cells using LTPA, it is a requirement that all the cells share the same user registry. Since LocalOS registries are restricted to the local machines, it is expected that either LDAP or a Custom Registry is used when inter-cell communication is required.

By default, the registry is local to the all the product processes. The performance is higher this way (no need for remote calls) and it also ensures high availability (any process failing will not effect other processes). This implies that when using LocalOS as the registry, every product process needs to be running with privilege access (root in UNIX, Act as part of opearting system in Windows systems). If this is not practical in some situations one can use a remote registry from the node or the cell. *Using a remote registry will effect performance and will create a single point of failure, and should be used only in rare situations.* The node and the cell processes are meant for manipulating configuration information and using them to host the registry for all the application servers will create more traffic than normal. One can set up to use a remote registry by setting the property `WAS_UseRemoteRegistry` in the Global Security panel using the **Custom Properties** link at the bottom of the panel. The value should be either Cell or Node (case insensitive). If the value is cell, the cell registry will be used by all the product processes (all Node Agents, all application servers). If the Cell process is down for any reason, all the processes need to be restarted (after the Cell is restarted). If the Node Agent registry needs to be used for the remote registry, set the value, `WAS_UseRemoteRegistry`, to **node**. In this case all the application server processes would use their respective Node Agent registry. In this case, if any of the Node Agents fail and does not come up automatically, all the application servers depending on those Node Agent(s) may need to be restarted once those Node Agent(s) are started.

Configuring user registries

Before configuring the user registry a decision should have been made on which registry to use. Though different types of registries are supported, only one active registry can be used by all the processes in WebSphere Application Server. Configuring the correct registry is a prerequiste to assigning users and groups to roles for applications. By default, when no registry is configured the LocalOS registry will be used. So if your choice of registry is not LocalOS you need to first configure the registry which is normally done as part of enabling security, restart the servers and then assign users and groups to roles for all your applications.

Once the applications are assigned users and groups, and you need to change the registries (say from LDAP to Custom), it is recommended that you delete all the users and groups (including any RunAs role) from the applications and reassign them after changing the registry. Deleting all the users and groups can be done through GUI or the wsadmin scripting. The following `wsadmin` command will remove all the users and groups (including the RunAs role) from any application. `$AdminApp deleteUserAndGroupEntries yourAppName` where *<yourAppName>* is the name of the application. Backing up the old application is adviced before performing this operation. However, if you have all the user and group names (including the password for the RunAs role users) in all the applications same in both the registries and if the application bindings file does not contain the `accessIDs` (which are unique for each registry, even for the same user or group name) you might be able switch the registries without having to delete the users and groups information. By default, an application does not contain `accessIDs` in the bindings file (they are generated on the fly when the applications are started). However, if you have migrated an existing application from an earlier release or if you had used the wsadmin script to add accessIDs for the applications (to improve performance) you will have to remove the existing user and group information and add them after configuring the new registry.

Steps for this task

1. Configure local operating system user registry.
2. Configure the LDAP user registry.
3. Configure the custom user registry.

Usage scenario

This step is required as part of enabling security in WebSphere Application Server.

What to do next

1. If you enabling security, make sure you complete the remaining steps. Also make sure that the Active User Registry in the Global Security panel is set to the appropriate registry. As the final step make sure that you validate the user and password by clicking OK or Apply in the Global Security panel. Save, stop and start all the WAS servers.
2. For any changes in user registry panels to be effective, first make sure to validate the changes by clicking the OK or Apply buttons in the Global Security panel. Once validated, save the configuration, stop and start all the WAS

servers (Cell, Nodes and all the AppServers). Also, in order to avoid inconsistencies between the WAS processes make sure any changes to the registry are done when all the processes are up and running. If any of the processes are down, one needs to do a force sync to make sure that process can come up later.

3. If the server(s) comes up without any problems the set up is correct.

Local operating system user registries

With the local operating system, or LocalOS, user registry implementation, the WebSphere authentication mechanism can use the user accounts database of the local operating system. WebSphere Application Server provides implementations for Windows NT and Windows 2000 local accounts registry and domain registry, as well as implementations for the Linux, Solaris and AIX user accounts registries. Note that Windows Active Directory is supported through the LDAP user registry implementation discussed later.

A LocalOS user registry is not a centralized user registry like LDAP. There are exceptions though, a Windows domain registry is a centralized registry. Typically, a LocalOS user registry is not the best choice in a distributed WebSphere environment, where application servers are dispersed across several machines, because each machine has its own user registry. Keeping the users (and their passwords) and groups in synchronization across several machines is problematic and an administrative burden.

Also, as mentioned previously, the access-IDs fetched from the user registry are used during authorization checks. Since these IDs are typically unique identifiers, they vary from machine to machine even if the exact users and passwords exist on each machine.

Using Windows NT systems or Windows 2000 operating system registries: When enabling security on Windows NT or Windows 2000 systems, if local operating system (LocalOS) is selected as the authentication mechanism, keep the following in mind:

- The ID that is running the WebSpher Application Server process should have the **Administrator** and **Act as part of the operating system** privilege in order to call the Windows systems API for authenticating and obtaining user or group information from the Windows operating system. This is normally the user who logs into the machine or if running as a service it is the *Log on as* user. You might need to reboot the machine the first time you give this access.
- WebSphere Application Server dynamically determines whether the machine is a member of a Windows domain.
- WebSphere Application Server does not support Windows NT trusted domains.
- If a machine is a member of a Windows domain, both the domain user registry and the local user registry of the machine participate in authentication and security role mapping.
- When using a Windows system domain, the ID that is running the WebSphere Application Server process should have enough privilege for authenticating and obtaining user or group information from the domain. This implies that this user should be a member of the domain administrator group in the domain and should have **Act as part of operating system** privilege in the local machine.
- The domain user registry takes precedence over the local user registry of the machine and might have undesirable implications if users with the same password exist in both user registries.

When LocalOS is selected, the user registry used for authentication depends on whether the machine is a member of a Windows domain. When WebSphere Application Server is started, the security run time initialization process dynamically attempts to determine if the local machine is a member of a Windows domain. WebSphere Application Server relies on the Windows computer browser service to help determine which domain the machine is a member of.

If the machine is not a member of a Windows domain, the user registry local to that machine is used for authentication.

If the machine is a member of a Windows domain, you can use both the domain user registry and the local user registry for authorization. The Windows domain registry is used for authentication first. If this registry cannot authenticate the user, authentication is attempted at the local user registry of the machine.

Authorizing with the domain user registry first can cause problems if a user exists in both the domain and local user registries with the same password. Role-based authorization can fail in this situation because the user is first authenticated within the domain user registry. This authentication produces a unique domain security ID that is used in WebSphere Application Server during the authorization check. However, the local user registry is used for role assignment. The domain security ID does not match the unique security ID associated with the role. To avoid this problem, map security roles to domain users instead of local users.

Using UNIX system registries

When using UNIX system registries the process ID that runs the WebSphere Application Server process should have the root authority to call the local operating system APIs for authentication and obtaining user or group information.

Note: In UNIX only the local machine's registry is used. NIS (Yellow Pages) is not supported.

Configuring local operating system user registries

For security purposes, the WebSphere Application Server provides and supports the implementation for Windows NT systems and Windows 2000 operating system registries, AIX, Solaris and multiple versions of Linux operating systems. The respective operating system APIs are called by the product processes (servers) for authenticating a user and other security-related tasks (for example, getting user/group information). Access to these APIs are restricted to users who have special privileges. These privileges depend on the operating system and are described below.

Before configuring the LocalOS registry you need to know the user name (ID) and password that will be used here. This user can be any valid user in the registry. This user will be referred to as either a product security server ID, a server ID or a server user ID in the documentation. Having a server ID means that a user to has special privileges when calling protected internal methods. Normally, this ID and password are used to log into the administrative console once security is turned on. You can use other users to log in if those users are part of the administrative roles. When security is enabled in the product, this server ID and password are authenticated with the registry during product startup. If authentication fails the server does not come up. So it is important to choose an ID and password that do not expire or change often. If the product server user ID or password need to be changed in the registry, ensure that the changes are performed when all the product servers are up and running. Once the changes are completed in the registry, use the steps described below to change the ID and the password information. Save, stop and restart all the servers so that the product can use the new ID or password. If there is any problem starting the product because of authentication problems (that cannot be fixed), disable security before the server can start up. To avoid this step, make sure the changes are validated in the Global Security panel. Once the server is up, change the ID and password information and enable security.

When using the Windows operating system, keep the following in mind:

- The server ID should not be the same as the Windows machine name where the product is installed. For example, if the Windows machine name is *vicky* and the security server ID is *vicky*, Windows treats the machine *vicky* to have an account similar to user *vicky* and hence will fail when getting the information (group information, for example) for user *vicky*.
- WebSphere Application Server dynamically determines whether the machine is a member of a Windows system domain.
- WebSphere Application Server does not support Windows trusted domains.
- If a machine is a member of a Windows domain, both the domain user registry and the local user registry of the machine participate in authentication and security role mapping.
- The domain user registry takes precedence over the local user registry of the machine and can have undesirable implications if users with the same password exist in both user registries.
- The user that the product processes run under should have the `Administrative` and `Act as part of the operating system` privileges to call the Windows operating system APIs that authenticate or collect user and group information. The process needs special authority, which is given by these privileges. The user in this example may not be the same as the security server ID (the requirement for which is a valid user in the registry). This user logs into the machine (if using the command line to start the product process) or the `Log On User` setting in the services panel if the product processes have started using the services. If the machine is also part of a domain, this user should be part of the `Domain Admin` group in the domain to call the operating system APIs in the domain in addition to having the `Act as part of opeating system` privilege in the local machine.

When using the UNIX operating systems (AIX, Solaris, Linux), consider the following:

- The user that the product processes run under should have the `root` privilege. This privilege is needed to call the UNIX operating system APIs to authenticate or to collect user and group information. The process needs special authority, which is given by the `root` privilege. This user may not be the same as the security server ID (the requirement is that it should be a valid user in the registry). This user logs into the machine and is running the product processes.
- When using the Linux operating system, you might need to have the password shadow file in your system.

The following steps are needed to perform this task initially when setting up security for the first time.

Steps for this task

1. Click **Security** > **User Registries** > **LocalOS** in the left navigation panel of the administrative console.

2. Enter a valid user name in the **Server User ID** field.
3. Enter the user password in the **Server User Password** field.
4. Click **OK**. Validation of the user and password does not happen in this panel. Validation is only done when you click **OK** or **Apply** in the Global Security panel. If you are in the process of enabling security for the first time, complete the other steps and go to the Global Security panel, make sure that Local OS is the Active User Registry. If security was already enabled and you had changed either the user or the password information in this panel, make sure to go to the Global Security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated the server may not be able to come up.

Results

The Local OS registry has been configured.

What to do next

1. If you are enabling security, complete the remaining steps. As the final step, ensure that you validate the user and password by clicking **OK** or **Apply** in the Global Security panel. Save, stop and start all the product servers.
2. For any changes in this panel to be effective, you need to save, stop and start all the product servers (cell, nodes and all the application servers).
3. If the server comes up without any problems the set up is correct.

Local operating system user registry settings: Use this page to configure local operating system user registry settings.

To view this administrative console page, click **Security** > **User Registries** > **Local OS User Registry**.

Server user ID: Specifies a valid user ID in the LocalOS registry.

This ID is the security server ID, which is only used for WebSphere Application Server security and is not associated with the system process that runs the server. The server calls the LocalOS registry to authenticate and obtain privilege information about users by calling the native APIs in that particular registry. Access to native APIs is normally restricted to users having special privilege (for example, root in UNIX and **Act as part of operating system** in Windows systems). To use security in the application server, the process ID (not the security server ID) on which WebSphere Application Server runs should have enough privilege to call the system APIs. The special privilege means that the process running the WebSphere Application Server needs to be part of the **Administrators** groups and has the **Act as part of operating system** privilege on Windows systems, and be **root**, or have root authority in UNIX.

- When using a Windows system registry, this ID cannot be the same as name of the Windows machine. This is because, Windows systems treat the machine name **bob** as having an acoount similar to user **bob**.

Data type	String
Units	Alphanumeric characters

Server user password: Specifies the password of the server user ID.

Data type	String

Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (LDAP) is a user registry in which authentication is performed using an LDAP binding.

WebSphere Application Server security provides and supports implementation for most major LDAP directory servers (LDAP servers) which can be used as the repository for user and group information. These LDAP servers are called by the product processes (servers) for authenticating a user and other security related tasks (for example, getting user or group information). This support is provided by using different user and group filters to obtain the user and group information. These filters have default values which can be modified to fit your needs. Also, the Custom LDAP feature enables one to use any other LDAP server (which is not in the product supported list of LDAP servers) for their user registry by using the appropriate filters. *Supporting new LDAP servers using the Custom LDAP feature is left to the customer.* In order to accomplish this one needs to understand how the filters are used by the product to obtain the user and group information. See Configuring LDAP search filters for more information. It is expected that the customer is responsible for the validity of the filters and testing the configuration and not depend on IBM for support.

In order to use LDAP as the user registry, one needs to know a valid user name (ID), the user password, the server host and port, the base DN and if necessary the bind DN and the bind password. The user chosen can be any valid user in the registry and should be searchable. In some LDAP servers some of the admin users are not searchable and hence cannot be used (for example, cn=root in SecureWay). This user will be referred to as WebSphere Application Server security server ID or server ID or server user ID in the documentation. Being a server ID entails a user to have special privileges when calling some protected internal methods. Normally, this ID and password is used to log into the administrative console once security is turned on (you can use other users to login if those users are part of the administrative roles).

When security is enabled in the product, this server ID and password will be authenticated with the registry during the product startup. If authentication fails the server will not come up. So it is important to choose an ID and password that would not expire or change often. If the product server user ID or password need to be changed in the registry, make sure the changes are performed when all the product servers are up and running. Once the changes are done in the registry, use the steps described in Configuring LDAP user registries change the ID, password and other configuration information if any, save, stop and restart all the servers so that the new ID or password will be used by the product. If there are any problems starting the product when security is enabled, security should be disabled before the server can start up (to avoid this in the first place, make sure any changes in this panel are validated in the Global Security panel). Once the server is up, one can change the ID, password and other configuration information and then enable security.

Supported directory services: WebSphere Application Server security supports the following LDAP servers: SecureWay, IBM_Directory_Server, iPlanet, Netscape, Domino and Active_Directory. Though it is expected that other LDAP servers that follow the LDAP specification would function, the support is limited to these specified directory servers only. You can use any other directory server by using the custom directory type in the drop-down list and by filling in the filters required for that directory.

In order to improve performance for LDAP searches, the default filters for IBM_Directory_Server, iPlanet and Active_Directory have been defined such that when you search for a user, the result contains all the relevant information about the user (user ID, groups, and so on). This prevents the product from going to the LDAP server multiple times and improves performance. This is possible only in these directory types because these support searches where the complete information about a user can be obtained.

Also, if you use the IBM_Directory_Server, enable the IgnoreCase flag. This is required because when the groups information is obtained from the user object attributes, the case is not the same as the one obtained when you get the groups information directly. In order for the authorization to work in this case, perform a case insensitive check and also check the requirement for the IgnoreCase flag.

Lightweight Directory Access Protocol settings: Use this page to configure Lightweight Third Party Authentication (LDAP) settings when users and groups reside in an external LDAP directory.

To view this administrative console page, click **Security** > **User Registries** > **LDAP User Registry** > **LDAP Settings**.

When security is enabled and any of these properties change, go to the **Global Security** panel and click **Apply** to validate the changes.

Server User ID: Specifies the user ID under which the server runs, for security purposes.

Although this ID is not the LDAP administrator user ID, it should be a valid entry in the LDAP directory located under the Base Distinguished Name.

Server User Password: Specifies the password corresponding to the security server ID.

Type: Specifies the type of LDAP server to which you connect.

The type is used to preload default LDAP properties. IBM Directory Server users can choose either `IBM_Directory_Server` or `SecureWay` as the directory type. Using the type of `IBM_ Directory_server` should have better performance. iPlanet users can choose either iPlanet or NetScape as the directory type. Using the type of iPlanet might have better performance, however, you must configure iPlanet to use `role (nsRole)` as the grouping method.

Host: Specifies the host ID (IP address or DNS name) of the LDAP server.

Port: Specifies the host port of the LDAP server.

If multiple WebSphere Application Servers are installed and configured to run in the same single sign-on domain, or if the WebSphere Application Server interoperates with a previous version of the WebSphere Application Server, then it is important that the port number match all configurations. For example, if the LDAP port is explicitly specified as **389** in a Version 4.0.x configuration, and a WebSphere Application Server Version 5 is going to interoperate with the Version 4.0.x server, then ensure port **389** is specified explicitly for the Version 5 server.

Default	389

Note: If the port (including the default port number) is specified explicitly in one server configuration, then ensure it is specified explicitly in all server configurations.

Base Distinguished Name: Specifies the base distinguished name of the directory service, indicating the starting point for LDAP searches of the directory service.

For example, for a user with a distinguished name (DN) of cn=John Doe, ou=Rochester, o=IBM, c=US, you can specify the base DN as (assuming a suffix of c=us): ou=Rochester, o=IBM, c=us o=IBM, c=us c=us. For authorization purposes. this field is case sensitive. This implies that if a token is received (for example, from another cell or Domino) the base DN in the server should match exactly the base DN from the other cell or Domino. If case sensitivity should not be considered for authorization, enable the **Ignore Case** field. This field is required for all LDAP directories except for the Domino Directory, where it is optional.

Bind Distinguished Name: Specifies the distinguished name for the application server to use when binding to the directory service.

If no name is specified, the application server binds anonymously. See the Base Distinguished Name field description for examples of distinguished names.

Bind Password: Specifies the password for the application server to use when binding to the directory service.

Search Timeout: Specifies the timeout value in seconds for an LDAP server to respond before aborting a request.

Default	300

Reuse connection: Specifies whether the server should reuse the LDAP connection. Uncheck this option only in rare situations where a router is used to spray requests to multiple LDAP servers and when the router does not support affinity.

Default	Enabled
Range	Enabled or Disabled

Ignore Case: Specifies that a case insensitive authorization check is performed.

This is a required field when **IBM_Directory_Server** is selected as the LDAP directory server. Otherwise, this field is optional and can be enabled when case sensitive authorization check needs to be performed. Some examples when it can be enabled include, if you are using certificates and the certificate contents may not match the case of the entry in the LDAP server. You can also enable the **Ignore Case** field when using SSO between the product and Domino.

Default	Disabled
Range	Enabled or Disabled

SSL Enabled: Specifies whether secure socket communications is enabled to the LDAP server. When enabled, the LDAP Secure Sockets Layer settings are used if specified.

SSL Configuration: Specifies the Secure Sockets Layer configuration to use for the LDAP connection. This configuration will be used only when SSL is enabled for LDAP.

Default	DefaultSSLSettings

Lightweight Directory Access Protocol advanced settings: Use this page to configure advanced LDAP user registry settings when users and groups reside in an external LDAP directory.

To view this administrative page, click **Security** > **User Registries** > **LDAP User Registry Advanced LDAP settings**.

Default values for all the user and group related filters are already completed in the appropriate fields. You can change these values depending on your requirements. These default values are based on the type of LDAP server selected in the **LDAP settings** panel. If this type is changed (for example from NETSCAPE to SECUREWAY) the default filters automatically get changed. When the default filter values change the LDAP server type changes to **Custom** to indicate that custom filters are used. When security is enabled and any of these properties change, go to the Global Security panel and click **Apply** or **OK** to validate the changes.

User Filter: Specifies the LDAP user filter that searches the registry for users.

This option is typically used for Security Role to User assignments. It specifies the property by which to look up users in the directory service. For example, to look up users based on their user IDs, specify `(ampersand(uid=%v)(objectclass=inetOrgPerson)` where `ampersand` is the ampersand symbol. For more information about this syntax, see the LDAP directory service documentation.

Data type	String

Group Filter: Specifies the LDAP group filter that searches the registry for groups

This option typically used for Security Role to Group assignments. It specifies the property by which to look up groups in the directory service. For more information about this syntax, see the LDAP directory service documentation.

Data type	String

User ID Map: Specifies the LDAP filter that maps the short name of a user to an LDAP entry.

Specifies the piece of information that represents users when users appear. For example, to display entries of the type object class = inetOrgPerson by their IDs, specify `inetOrgPerson:uid`. This field takes multiple objectclass:property pairs delimited by a semicolon (";").

Data type	String

Group ID Map: Specifies the LDAP filter that maps the short name of a group to an LDAP entry.

Specifies the piece of information that represents groups when groups appear. For example, to display groups by their names, specify `*:cn`. The `*` is a wildcard character that searches on any object class in this case. This field takes multiple `objectclass:property` pairs delimited by a semicolon (″;″).

Data type	String

Group Member ID Map: Specifies the LDAP filter which identifies user to group relationships.

For a directory type of SecureWay, NetScape, and Domino, this field takes multiple objectclass:property pairs delimited by a semicolon (″;″). In objectclass:property pair, the objectclass is the same objectclass defined in Group Filter, and the property is the member attribute. If objectclass does not match objectclass in Group Filter, authorization might fail if groups are mapped to security roles. For more information about this syntax, see your LDAP directory service documentation.

For IBM Directory Server, iPlanet and Active Directory, this field takes multiple (group attribute:member attribute) pairs delimited by a semicolon (″;″). They are used to find the group memberships of a user by enumerating all the group attributes possessed by a given user. For example, attribute pair (`memberof:member`) is used by Active Directory, and ″(ibm-allGroup:member)″ is used by IBM Directory Server. This field also specifies which property of an objectclass stores the list of members belonging to the group represented by the objectclass.

Data type	String

Certificate Map Mode: Specifies whether to map X.509 certificates into an LDAP directory by EXACT_DN or CERTIFICATE_FILTER. Specify CERTIFICATE_FILTER to use the specified certificate filter for the mapping.

Data type	String

Certificate Filter: Specifies whether to use the filter Certificate Mapping property to specify the LDAP filter, which is used to map attributes in the client certificate to entries in the LDAP registry.

To enable this field, click **CERTIFICATE_FILTER** for the certificate mapping. If more than one LDAP entry matches the filter specification at run time, then authentication will fail because it results in an ambiguous match. The syntax or structure of this filter is: `LDAP attribute=${Client certificate attribute}` (for example, `uid=${SubjectCN}`). The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. The right side must begin with `${ and end with }`. The following certificate attribute values may be used on the right side of the filter specification. The case of the strings is important:

- `${UniqueKey}`
- `${PublicKey}`
- `${PublicKey}`
- `${Issuer}`
- `${NotAfter}`
- `${NotBefore}`

- `${SerialNumber}`
- `${SigAlgName}`
- `${SigAlgOID}`
- `${SigAlgParams}`
- `${SubjectCN}`
- `${Version}`

Data type	String

Using specific directory servers as the LDAP server:

Using IBM Directory Server as the LDAP server: You can choose the directory type of either IBM Directory Server or SecureWay for the IBM Directory Server. The difference between these two types is group membership lookup. It is recommended that you choose the IBM Directory Server for optimum performance during run time. In the IBM Directory Server, the group membership is an operational attribute. An entry can be a member directly by the member, or uniqueMember. With this attribute, a group membership lookup is done by enumerating the `ibm-allGroups` attribute for the entry, rather than selecting a group and browsing through the members list. To utilize this attribute in a security authorization application, use case-insensitive match so that attribute values returned by `ibm-allGroups` are all in upper case. Lower-case values are stored in the directory server.

Using iPlanet Directory Server as LDAP server: You can choose the directory type of either iPlanet or NetScape for your iPlanet Directory Server. The difference between these two types is group membership lookup. The iPlanet directory type is selected to use the iPlanet new grouping mechanism only. The new grouping mechanism is called *roles* in iPlanet, and the attribute is *nsRole*. Roles are a new entry grouping mechanism in iPlanet that unify entries. Roles are designed to be more efficient and easier to use for applications. For example, an application can locate the role of an entry by enumerating all the roles possessed by a given entry, rather than selecting a group and browsing through the members list. With the iPlanet directory, the WebSphere Application Server security only supports groups defined by *nsRole*. If you are planning to use traditional grouping methods to group entries in the iPlanet server, select **NetScape** as the directory type.

Using MS Active Directory server as the LDAP server:

Before you begin

To use Microsoft Active Directory as the LDAP server for authentication with WebSphere Application Server, there are specific steps you must take. By default, Microsoft Active Directory does not permit anonymous LDAP queries. To create LDAP queries or browse the directory, an LDAP client must bind to the LDAP server using the distinguished name (DN) of an account that belongs to the **administrator** group of the Windows system. Group membership search in the Active Directory is done by enumerating the `memberof` attribute possessed by a given user entry, rather than browsing through the member list in each group. If you change this default behavior to browse each group, you can change the **Group Member ID Map** field from `memberof:member` to `group:member`.

To set up Microsoft Active Directory as your LDAP server, complete the following steps.

Steps for this task

1. Determine the full DN and password of an account in the **administrators** group.

   For example, if the Active Directory administrator creates an account in the Users folder of the Active Directory Users and Computers Windows NT or Windows 2000 systems control panel and the DNS domain is `ibm.com`, the resulting DN has the following structure:

   ```
   cn=<adminUsername>, cn=users, dc=ibm,
   dc=com
   ```
2. Determine the short name and password of any account in the Microsoft Active Directory.

   This password does not have to be the same account as used in the previous step.
3. Use the WebSphere Application Server administrative console to set up the information needed to use Microsoft Active Directory:
   a. Start the administrative server for the domain, if necessary.
   b. On the administrative console, click **Security** on the left navigation panel.
   c. Click the **Authentication mechanisms** tabbed page. Select Lightweight Third Party Authentication (LTPA) as the authentication mechanism.
   d. Enter the following information in the LDAP settings fields:
      - **Security Server ID**: The short name of the account chosen in 2
      - **Security Server Password**: The password of the account chosen in step 2
      - **Directory Type**: Active Directory
      - Host: The DNS name of the machine running Microsoft Active Directory
      - **Base Distinguished Name**: The domain components of the DN of the account chosen in step 1. For example: dc=ibm, dc=com Bind
      - **Distinguished Name**: The full DN of the account chosen in step 1. For example: cn=<*adminUsername*>, cn=users, dc=ibm, dc=com
      - **Bind Password**: the password of the account chosen in step 1
   e. Click **OK** to save the changes.
   f. Stop and restart the administrative server so that changes take effect.

Configuring Lightweight Directory Access Protocol user registries

Review the article on Lightweight Directory Access Protocol (LDAP) before beginning this task.

Steps for this task

1. In the administrative console, click **Security** > **User Registries** > **LDAP** in the left navigation panel.
2. Enter a valid user name in the **Server User ID** field. You can either enter the complete distinguished name (DN) of the user or the shortname of the user as defined by the **User Filter** in the Advanced LDAP settings panel. For example, for Netscape enter the user ID.
3. Enter the password of the user in the **Server User Password** field.
4. Select the type of LDAP server that is used from the **Type** drop-down list. The type of LDAP server determines the default filters that are used by the WebSphere Application Server. When these default filters are changed the **Type** field changes to **Custom**, which indicates that custom filters are used. This action occurs once **OK** or **Apply** is clicked in the LDAP advance settings

panel. Choose the **Custom** type from the list and modify the user and group filters to use other LDAP servers, if required. However, it is the customer responsibility to come with the filters and validate them for other LDAP servers. Also, if either the IBM_Directory_Server or iPlanet is selected, the **Ignore Case** field should also be selected.

5. Enter the fully qualified host name of the LDAP server in the **Host** field.
6. Enter the LDAP server port number in the **Port** field. The host name along with the port number represent the realm for this LDAP server in the WebSphere Application Server cell. So, if servers in different cells are communicating with each other (using LTPA tokens), these realms should match exactly in all the cells.
7. Enter the Base distinguished name (DN) in the **Base Distinguished Name** field. The Base DN indicates the starting point for searches in this LDAP directory server. For example, for a user with a DN of cn=John Doe, ou=Rochester, o=IBM, c=US, the Base DN can be specified as any of (assuming a suffix of c=us): ou=Rochester, o=IBM, c=us or o=IBM c=us or c=us. This field can be case sensitive, and it is recommended that they match the case in your directory server. This field is required for all LDAP directories except the Domino Directory. The Base DN field is optional for the Domino Server.
8. Enter the Bind DN name in the **Bind Distinguished Name** field, if necessary. The Bind DN is required if anonymous binds cannot be performed on the LDAP server to obtain user and group information. If the LDAP server is set up to use anonymous binds, leave this field blank.
9. Enter the password corresponding to the Bind DN in the **Bind password** field, if necessary.
10. Modify the **Search Time Out** value if required. This time out value is the maximum amount of time the LDAP server waits to send a response to the product client before aborting the request. The default is 120 seconds.
11. Disable the **Reuse Connection** field only if you are using routers to spray requests to multiple LDAP servers, and if the routers do not support affinity. Leave this field enabled for all other situations.
12. Enable the **Ignore Case** flag, if required. When this is enabled, the authorization check is case insensitive. Normally, an authorization check involves checking the complete DN of a user (which is unique in the LDAP server) and is case sensitive. However, when using either the IBM_Directory_Server or the iPlanet LDAP servers, this flag needs enabling because the group information obtained from the LDAP servers is not consistent in terms of case. This inconsistency only effects the authorization check.
13. Enable single sockets layer (SSL) if the communication to the LDAP server is through SSL. For more information on setting up LDAP for SSL, refer to (Configuring SSL for LDAP clients).
14. If SSL is enabled, select the appropriate SSL alias configuration from the drop-down list in the **SSL configuration** field.
15. Click **OK**.

    The validation of the user and password and the setup does not take place in this panel. Validation is only done when you click **OK** or **Apply** in the Global Security panel. If you are enabling security for the first time, complete the remaining steps and go to the Global Security panel. Select **LDAP** as the Active User Registry. If security is already enabled but information on this panel is changed, make sure to go to the Global Security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not come up.

Results

Sets the LDAP registry configuration.

Usage scenario

This step is required to set up the LDAP registry. This step is required as part of enabling security in the WebSphere Application Server.

What to do next

1. If you are enabling security, complete the remaining steps. As the final step, ensure that you validate this setup by clicking **OK** or **Apply** in the Global Security panel. Save, stop and start all the product servers.
2. For changes in this panel to take effect, save, stop and restart all the product servers (cell, nodes and all the application servers).
3. If the server comes up without any problems the setup is correct.

Configuring Lightweight Directory Access Protocol search filters:

Before you begin

Lightweight Directory Access Protocol (LDAP) filters are used by the WebSphere Application Server to search and obtain information about users and groups from a LDAP directory server. A default set of filters are provided for each LDAP server that the product supports. These filters can be modified to fit your LDAP configuration. Once the filters are modified (and **OK** or **Apply** is clicked) the directory type in the LDAP registry panel changes to custom, which indicates that custom filters are being used. Also, you can develop filters to support any additional type of LDAP server. The effort to support additional LDAP directories is optional, and IBM does not provide support for other LDAP directory types.

Steps for this task

1. In the administrative console, click **Security** > **User Registries** > **LDAP** in the left navigation panel. Click **Advanced LDAP Setting** in Additional Properties.
2. Modify the user filter, if necessary.

   The user filter is used for searching the registry for users. It is typically used for Security Role to User assignment. It is also used to authenticate a user using the attribute specified in the filter. It specifies the property for which to look up users in the directory service. For example, to look up users based on their user IDs and using the object class inetOrgPerson, specify: (&(uid=%v)(objectclass=inetOrgPerson). For more information about this syntax, see the LDAP directory service documentation.
3. Modify the group filter if necessary.

   The group filter is used for searching the registry for groups. It is typically used for Security Role to Group assignment. It specifies the property by which to look up groups in the directory service. For example, to look up groups based on their common names (CN) and using the object class of either groupOfNames or groupOfUniqueNames, specify

(&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames))). For more information about this syntax, see the LDAP directory service documentation.

4. Modify the User ID Map filter. if necessary.

   This filter maps the short name of a user to an LDAP entry. This specifies the piece of information that should represent users when users are displayed using their short names. For example, to display entries of the type object class = inetOrgPerson by their IDs, specify inetOrgPerson:uid. This field takes multiple objectclass:property pairs delimited by a semicolon (";"). To provide a consistent value for methods like getCallerPrincipal( ), getUserPrincipal() the short name obtained by using this filter is used. For example the user CN=Bob Smith, ou=austin.ibm.com, o=IBM, c=US can log in using any attributes that were defined for him (for example, e-mail address, social security number, and so on) but when the above methods are called, the user ID **bob** is returned no matter how he logs in.

5. Modify the **Group ID Map** filter, if necessary.

   This filter maps the short name of a group to an LDAP entry. This specifies the piece of information that should represent groups when groups are displayed. For example, to display groups by their names, specify `*:cn`. The * is a wildcard character that searches on any object class in this case. This field takes multiple `objectclass:property` pairs delimited by a semicolon (";").

6. Modify the **Group Member ID Map** if necessary.

   This filter identifies User to Group memberships. For SecureWay, Netscape and Domino directory types, this field is used to query all the groups that match the specified object class(es) to find if the user is contained in the attribute specified. For example, to get all the users belonging to groups whose object class is `groupOfNames` and the users are contained in the member attributes, specify `groupOfNames:member`. This specifies which property of an objectclass stores the list of members belonging to the group represented by the objectclass. This field takes multiple objectclass:property pairs delimited by a semicolon (";"). For more information about this syntax, see the LDAP directory service documentation. For the IBM Directory Server, iPlanet and Active Directory, this is used to query all users in a group by using the information stored in the user object (instead of querying all the groups individually to find if the user exists in that group). For example, the filter `memberof:member` (for Active Directory) is used to get the `memberof` attribute of the user object to get all the groups that the user belongs to. The *member* attribute is used to get all the users in a group using the group object. Using the user object to obtain the group information is expected to improve performance.

7. Modify the certificate Map Mode, if necessary.

   The X.590 certificates can be used for user authentication when LDAP is selected as the user registry. This field is used to indicate whether to map the X.509 certificates into an LDAP directory user by **EXACT_DN** or **CERTIFICATE_FILTER**. If **EXACT_DN** is selected, the DN in the certificate should exactly match the user entry in the LDAP server (including case and spaces). One can use the **Ignore Case** field in the LDAP settings to make the authorization case insensitive. If CERTIFICATE_FILTER is selected, fill in the appropriate certificate filter (in the next field) that should be used for mapping the certificate to a user in the LDAP.

8. If you specified the filter certificate mapping in step 7, use this property to specify the LDAP filter to use to map attributes in the client certificate to entries in LDAP.

If more than one LDAP entry matches the filter specification at run time, then authentication fails because it results in an ambiguous match. The syntax or structure of this filter is: `LDAP attribute=${Client certificate attribute}` (for example, `uid=${SubjectCN}`). The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. Note that the right side must begin with ${and end with}. Use the following certificate attribute values on the right side of the filter specification. The case of the strings is important.

- ${UniqueKey}
- ${PublicKey}
- ${Issuer}
- ${NotAfter}
- ${NotBefore}
- ${SerialNumber}
- ${SigAlgName}
- ${SigAlgOID}
- ${SigAlgParams}
- ${SubjectDN}
- ${Version}

To enable this field, select **CERTIFICATE_FILTER** for the certificate mapping.

9. Click **OK**.

   The validation of the changes (if any) does not take place in this panel. Validation is only done when the **OK** or **Apply** buttons are pressed in the Global Security panel. If you are in the process of enabling security for the first time, complete the remaining steps and go to the Global Security panel, select **LDAP** as the Active User Registry. If security was already enabled and any information on this panel is changed, make sure to go to the Global Security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated the server may not be able to come up.

Results

Sets the LDAP search filters.

Usage scenario

This step is required to modify existing user and group filters for a particular LDAP directory type. It is also used to set up certificate filters to map certificates to entries in the LDAP server.

What to do next

1. If you are enabling security, complete the remaining steps. As the final step make sure that you validate this setup by clicking **OK** or **Apply** in the Global Security panel. Save, stop and start all the product servers.
2. For any changes in this panel to be effective, you need to save, stop and start all the product servers (cell, nodes and all the applicatin servers).
3. Once the server comes up, make sure to go through all the security related tasks (getting users, getting groups and so on) to make sure the changes to the filters function.

Custom user registries

A custom user registry is a customer implemented user registry, which implements the UserRegistry Java interface as provided by the product. A custom implemented user registry can support virtually any type or notion of an accounts repository from a relational database, flat file, etc. The custom user registry provides considerable flexibility in adapting product security to various environments where some notion of a user registry, other than Lightweight Directory Access Protocol (LDAP) or Local Operating System (LocalOS), already exists in the operational environment.

WebSphere Application Server security provides an implementation that uses various local operating system based registries (Windows, AIX, Solaris, Linux) and various Lightweight Directory Access Protocol (LDAP) based registries. However, there might be situations where your user and group data resides in other repositories (a database, for example) and moving this information to either the LocalOS or LDAP might not be feasible. For these situations the WebSphere Application Server security provides an SPI that you can implement to interact with your current registry. The SPI is the UserRegistry interface. This interface has a set of methods that need implementing in order for the product security to interact with your registries for all security-related tasks. The LocalOS and LDAP registry implementations that are provided also implement this interface. Custom user registries are sometimes called the pluggable user registries or *custom registries* for short.

The article on ("UserRegistry interface methods") describes each of the methods in the UserRegistry interface that need implementing. An explanation of each of the methods along with their usage in the sample and any changes from the Version 4.0 interface are provided. The Related reference section provides links to all other custom user registries documentation, including a simple file based registry sample. The sample provided is very simple and is intended to familiarize you with this feature. Do not use this sample in an actual production environment.

Configuring custom user registries

Before you begin this task, implement and build the UserRegistry interface. For more information on the developing custom user registries refer to the article, (Developing custom user registries). The following steps are required to configure custom user registries through the administrative console.

You must also decide which registry to use. Though different types of registries are supported only one active registry can be used by all the processes in WebSphere Application Server. Configuring the correct registry is a prerequiste to assigning users and groups to roles for applications. By default, when no registry is configured, the LocalOS registry is used. So if your choice of registry is not LocalOS, configure the registry that is normally done as part of enabling security. Restart the servers, and then assign users and groups to roles. Complete this process for all your applications.

If you need to change the registries (say from LDAP to Custom), once the applications are assigned users and groups remember to delete all the users and groups (including any RunAs role) from the applications and reassign them after changing the registry. You can delete all the users and groups through the administrative console or by using the `wsadmin` script. The `wsadmin` command removes all the users and groups (including the RunAs role) from any application.

Issue `$AdminApp deleteUserAndGroupEntries yourAppName` where `yourAppName` is the name of the application. Backing up the previous application before performing this operation. However, if you have all the same user and group names (including the password for the RunAs role users) in all the applications in both registries, and if the application bindings file does not contain the `accessIDs` (which are unique for each registry, even for the same user or group name) you might be able to switch the registries without having to delete the users and groups information. By default, an application does not contain `accessIDs` in the bindings file. They are generated on the when the applications are started. However, if you migrated an existing application from a previous release or if you used the `wsadmin` script to add `accessIDs` for the applications (for improving performance), remove the existing user and group information, and add them after configuring the new registry.

Steps for this task

1. In the administrative console, click on **Security** > **User Registries** > **Custom** in the left navigation panel.
2. Enter a valid user name in the **Server User ID** field.
3. Enter the password of the user in the **Server User Password** field.
4. Enter the full name of the location of the implementation class file in the **Custom Registry Classname** field. This should be a dot separated file name. For the sample this is `com.ibm.websphere.security.FileRegistrySample`. The file exists in the WebSphere Application Server class path (preferably in the `%WAS_HOME%/classes` directory).

   This file exists in all the product processes. So, if you are operating in a Network Deployment environment, this file exists in the cell class path as well as all the nodes class path.
5. Select the **Ignore Case** checkbox for the authorization to perform a case insensitive check. Enabling this option is necessary only when your registry is case insensitive and does not provide a consistent case when queried for users and groups.
6. Click **Apply** if you have any other additional properties to enter for the registry initialization. Otherwise click **OK** and complete the steps required to turn on security.
7. If you need to enter additional properties to initialize your implementation, click **Custom Properties** at the bottom of the panel. Click **New**. Enter the property name and value. Click **OK**. Repeat this step to add other additional properties.

   For the sample, enter the following two properties: (assuming the `users.props` and `groups.props` are in `myDir` directory under the product installation directory).

Property name	**Property value**
usersFile	${USER_INSTALL_ROOT}/myDir/users.props
groupsFile	${USER_INSTALL_ROOT}/myDir/groups.props

The **Description**, **Required** and **Validation Expression** fields are not used and can be left blank.

Note: In a Network Deployment environment where multiple WebSphere Application Server processes exist (cell, and multiple nodes in different machines), these properties are available for each process. So, use the relative name `${USER_INSTALL_ROOT}` to locate any files, as this expands to the product installation directory. If this name is not used, ensure that the files exist in the same location in all the nodes.

Results

This step is required to set up the custom user registry. This step is required as part of enabling security in WebSphere Application Server.

What to do next

1. If you enable security, you complete the remaining steps. As the final step, validate the user and password by clicking **OK** or **Apply** in the Global Security panel. Save, synchronize (in the CELL environment), stop and start all the product servers.
2. Once security is turned on, for any changes in this panel to take effect, you need to save, stop and start all the product servers (cell, nodes and all the application servers).
3. If the server comes up without any problems, the setup is correct.

UserRegistry.java files:

Usage scenario

```
// @(#) 1.19 src/en/ae/rsec_usereg.xml, WEBSJAVA.INFO.DOCSRC, ASVINFO1 10/17/02 16:43:05
[10/18/02 07:31:31]
// 5639-D57, 5630-A36, 5630-A37, 5724-D18 (C) COPYRIGHT International Business Machines Corp.
1997, 2002
// All Rights Reserved * Licensed Materials - Property of IBM
//
// DESCRIPTION:
//
//    This is the UserRegistry interface that Custom Registries in WebSphere
//    should implement to enable WebSphere Security to use the Custom Registry.
//

package com.ibm.websphere.security;

import java.util.*;
import java.rmi.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.cred.WSCredential;

/**
 * Implementing this interface enables WebSphere Security to use Custom
 * Registries. This should extend java.rmi.Remote as the registry can be in
 * a remote process.
 *
 * Implementation of this interface must provide implementations for:
```

- initialize(java.util.Properties)
- checkPassword(String,String)
- mapCertificate(X509Certificate[])
- getRealm
- getUsers(String,int)
- getUserDisplayName(String)
- getUniqueUserId(String)
- getUserSecurityName(String)
- isValidUser(String)
- getGroups(String,int)
- getGroupDisplayName(String)
- getUniqueGroupId(String)

- getUniqueGroupIds(String)
- getGroupSecurityName(String)
- isValidGroup(String)
- getGroupsForUser(String)
- getUsersForGroup(String,int)
- createCredential(String)

```
**/

public interface UserRegistry extends java.rmi.Remote
{

  /**
   * Initializes the registry. This method is called when creating the
   * registry.
   *
   * @param props the registry-specific properties with which to
   *               initialize the  custom registry
   * @exception CustomRegistryException
   *                    if there is any registry specific problem
   * @exception RemoteException
   *    as this extends java.rmi.Remote
   **/
   public void initialize(java.util.Properties props)
      throws CustomRegistryException,
             RemoteException;

  /**
   * Checks the password of the user. This method is called to authenticate a
   * user when the user's name and password are given.
   *
   * @param userSecurityName the name of user
   * @param password the password of the user
   * @return    a valid userSecurityName. Normally this is
   *   the name of same user whose password was checked but if the
   *  implementation wants to return any other valid
   *  userSecurityName in the registry it can do so
   * @exception CheckPasswordFailedException if userSecurityName/
   *  password combination does not exist in the registry
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String checkPassword(String userSecurityName, String password)
      throws PasswordCheckFailedException,
             CustomRegistryException,
             RemoteException;

  /**
   * Maps a Certificate (of X509 format) to a valid user in the Registry.
   * This is used to map the name in the certificate supplied by a browser
   * to a valid userSecurityName in the registry
   *
   * @param cert the X509 certificate chain
   * @return the mapped name of the user userSecurityName
   * @exception CertificateMapNotSupportedException if the particular
   *           certificate is not supported.
   * @exception CertificateMapFailedException if the mapping of the
   *           certificate fails.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String mapCertificate(X509Certificate[] cert)
      throws CertificateMapNotSupportedException,
             CertificateMapFailedException,
             CustomRegistryException,
             RemoteException;

  /**
   * Returns the realm of the registry.
   *
   * @return the realm. The realm is a registry-specific string indicating
   *            the realm or domain for which this registry
   *            applies.  For example, for OS400 or AIX this would be the
   *            host name of the system whose user registry this object
   *            represents.
   *            If null is returned by this method realm defaults to the
   *            value of "customRealm".
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getRealm()
      throws CustomRegistryException,
             RemoteException;

  /**
   * Gets a list of users that match a pattern in the registy.
   * The maximum number of users returned is defined by the limit
   * argument.
   * This method is called by GUI(adminConsole) and Scripting(Command Line) to
   * make available the users in the registry for adding them (users) to roles.
   *
   * @param pattern the pattern to match. (For e.g., a* will match all
   *   userSecurityNames starting with a)
   * @param limit the maximum number of users that should be returned.
   *  This is very useful in situations where there are thousands of
   *            users in the registry and getting all of them at once is not
   *            practical. A value of 0 implies get all the users and hence
   *            must be used with care.
   * @return a Result object that contains the list of users
   *   requested and a flag to indicate if more users exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public Result getUsers(String pattern, int limit)
      throws CustomRegistryException,
             RemoteException;

  /**
   * Returns the display name for the user specified by userSecurityName.
   *
   * This method may be called only when the user information is displayed
   * (i.e information purposes only, for example, in GUI) and hence not used
   * in the actual authentication or authorization purposes. If there are no
   * display names in the registry return null or empty string.
   *
   * In WAS 4.0 custom registry, if you had a display name for the user and
   * if it was different from the security name, the display name was
   * returned for the EJB methods getCallerPrincipal() and the servlet methods
   * getUserPrincipal() and  getRemoteUser().
   * In WAS 5.0 for the same methods the security name will be returned by
   * default. This is the recommended way as the display name is not unique
   * and might create security holes.
   * However, for backward compatability if one needs the display name to
   * be returned set the property WAS_UseDisplayName to true.
   *
   * See the Infocenter documentation for more information.
   *
   * @param userSecurityName the name of the user.
   * @return the display name for the user. The display name
   *   is a registry-specific string that represents a descriptive, not
   *  necessarily unique, name for a user. If a display name does
   *            not exist return null or empty string.
   * @exception EntryNotFoundException if userSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getUserDisplayName(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Returns the UniqueId for a userSecurityName. This method is called when
   * creating a credential for a user.
   *
   * @param userSecurityName the name of the user.
   * @return the UniqueId of the user. The UniqueId for an user is
   *   the stringified form of some unique, registry-specific, data
   *  that serves to represent the user.  For example, for the UNIX
   *  user registry, the UniqueId for a user can be the UID.
   * @exception EntryNotFoundException if userSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getUniqueUserId(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Returns the name for a user given its uniqueId.
   *
   * @param uniqueUserId the UniqueId of the user.
   * @return the userSecurityName of the user.
   * @exception EntryNotFoundException if the uniqueUserId does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getUserSecurityName(String uniqueUserId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Determines if the userSecurityName exists in the registry
   *
   * @param userSecurityName the name of the user
   * @return true if the user is valid. false otherwise
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public boolean isValidUser(String userSecurityName)
      throws CustomRegistryException,
             RemoteException;

  /**
   * Gets a list of groups that match a pattern in the registy.
   * The maximum number of groups returned is defined by the limit
   * argument.
   * This method is called by GUI(adminConsole) and Scripting(Command Line) to
   * make available the groups in the registry for adding them (groups) to
   * roles.
   *
   * @param pattern the pattern to match. (For e.g., a* will match all
   *   groupSecurityNames starting with a)
   * @param limit the maximum number of groups that should be returned.
   *  This is very useful in situations where there are thousands of
   *           groups in the registry and getting all of them at once is not
   *           practical. A value of 0 implies get all the groups and hence
   *           must be used with care.
   * @return a Result object that contains the list of groups
   *   requested and a flag to indicate if more groups exist.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public Result getGroups(String pattern, int limit)
      throws CustomRegistryException,
             RemoteException;

  /**
   * Returns the display name for the group specified by groupSecurityName.
   *
   * This method may be called only when the group information is displayed
   * (for example, GUI) and hence not used in the actual authentication or
   * authorization purposes. If there are no display names in the registry
   * return null or empty string.
   *
   * @param groupSecurityName the name of the group.
   * @return the display name for the group. The display name
   *   is a registry-specific string that represents a descriptive, not
   *  necessarily unique, name for a group. If a display name does
   *           not exist return null or empty string.
   * @exception EntryNotFoundException if groupSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getGroupDisplayName(String groupSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Returns the Unique id for a group.

   * @param groupSecurityName the name of the group.
   * @return the Unique id of the group. The Unique id for
   *   a group is the stringified form of some unique,
   *   registry-specific, data that serves to represent the group.
   *           For example, for the Unix user registry, the Unique id could
   *  be the GID.
   * @exception EntryNotFoundException if groupSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getUniqueGroupId(String groupSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;



  /**
   * Returns the Unique ids for all the groups that contain the UniqueId of
   * a user.
   * Called during creation of a user's credential.
   *
   * @param uniqueUserId the uniqueId of the user.
   * @return a List of all the group UniqueIds that the uniqueUserId
   *   belongs to. The Unique id for an entry is the stringified
   *  form of some unique, registry-specific, data that serves
   *  to represent the entry.  For example, for the
   *   Unix user registry, the Unique id for a group could be the GID
   *  and the Unique Id for the user could be the UID.
   * @exception EntryNotFoundException if uniqueUserId does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public List getUniqueGroupIds(String uniqueUserId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Returns the name for a group given its uniqueId.
   *
   * @param uniqueGroupId the UniqueId of the group.
   * @return the name of the group.
   * @exception EntryNotFoundException if the uniqueGroupId does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getGroupSecurityName(String uniqueGroupId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Determines if the groupSecurityName exists in the registry
   *
   * @param groupSecurityName the name of the group
   * @return true if the groups exists, false otherwise
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public boolean isValidGroup(String groupSecurityName)
      throws CustomRegistryException,
             RemoteException;

  /**
   * Returns the securityNames of all the groups that contain the user
   *
   * This method is called by GUI(adminConsole) and Scripting(Command Line)
   * to verify the user entered for RunAsRole mapping belongs to that role
   * in the roles to user mapping. Initially, the check is done to see if
   * the role contains the user. If the role does not contain the user
   * explicitly, this method is called to get the groups that this user
   * belongs to so that check can be made on the groups that the role contains.
   *
   * @param userSecurityName the name of the user
   * @return a List of all the group securityNames that the user
   *   belongs to.
   * @exception EntryNotFoundException if user does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public List getGroupsForUser(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

```
  /**
   * Gets a list of users in a group.
   *
   * The maximum number of users returned is defined by the limit
   * argument.
   *
   * This method is not used by WebSphere Application Server (WAS) for
   * authenticating or authorization purposes. This is, however, used by some
   * of the WAS clients like Workflow.
   *
   * If you are working with a registry where getting all the users from
   * any of your groups is not practical (for example if there are a large
   * number of users) you can through the NotImplementedException. Also,
   * if you implement this method, you can still throw this exception if
   * the limit exceeds some practical value.
   * When the NotImplementedException is thrown the client program should fall
   * back to some default implementation which should be documented by the
   * client.
   *
   * @param groupSecurityName the name of the group
   * @param limit the maximum number of users that should be returned.
   *  This is very useful in situations where there are lot of
   *            users in the registry and getting all of them at once is not
   *            practical. A value of 0 implies get all the users and hence
   *            must be used with care.
   * @return a Result object that contains the list of users
   *   requested and a flag to indicate if more users exist.
   * @deprecated This method will be deprecated in future.
   * @exception NotImplementedException throw this exception if it is not
   *            pratical to get this information from your registry.
   * @exception EntryNotFoundException if the group does not exist in
   *            the registry
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   *
   **/
   public Result getUsersForGroup(String groupSecurityName, int limit)
      throws NotImplementedException,
             EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Throw the NotImplementedException for this method.
   *
   * Create Credential for a user.
   *
   * This will be implemented internally by WebSphere code and should NOT be
   * implemented by the Custom Registry implementations.
   *
   * @exception NotImplementedException Always throw this.
   **/
   public com.ibm.websphere.security.cred.WSCredential
   createCredential(String userSecurityName)
      throws NotImplementedException,
      EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
}
```

FileRegistrySample.java files:

Following is the contents of the FileRegistrySample.java file:

```
package com.ibm.websphere.security;
//
// 5639-D57, 5630-A36, 5630-A37, 5724-D18 (C) COPYRIGHT International Business Machines Corp.
1997, 2002
// All Rights Reserved * Licensed Materials - Property of IBM
//

//----------------------------------------------------------------------
// This program may be used, executed, copied, modified and distributed
// without royalty for the purpose of developing, using, marketing, or
// distributing.
//----------------------------------------------------------------------
//

// This sample is for the Custom User Registry feature in WebSphere

//----------------------------------------------------------------------
// The main purpose of this sample is to demonstrate the use of the
// Custom Registry feature available in WebSphere. This sample is a very
// simple File based registry sample where the users and the groups information
// is listed in files (users.props and groups.props). As such simplicity and
// not the performance was a major factor behind this. This sample should be
// used only to get familiarized with this feature. An actual implementation
// of a realistic registry should consider various factors like performance,
// scalability etc.
//----------------------------------------------------------------------
import java.util.*;
import java.io.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.*;

public class FileRegistrySample implements UserRegistry {

   private static String USERFILENAME = null;
   private static String GROUPFILENAME = null;

   // Default Constructor
   public FileRegistrySample() throws java.rmi.RemoteException {
   }

  /**
   * Initializes the registry. This method is called when creating the
   * registry.
   *
   * @param     props   the registry-specific properties with which to
   *                    initialize the  custom registry
   * @exception CustomRegistryException
   *                    if there is any registry specific problem
   **/
   public void initialize(java.util.Properties props)
         throws CustomRegistryException {
      try {
          /* try getting the USERFILENAME and the GROUPFILENAME from
          * properties that are passed in (i.e from GUI).
          * These values should be set in the security center GUI in the
          * Special Custom Settings in the Custom User Registry section of
          * the Authentication panel.
          * For example:
          * usersFile   c:/temp/users.props
          * groupsFile  c:/temp/groups.props
          */
          if (props != null) {
             USERFILENAME = props.getProperty("usersFile");
             GROUPFILENAME = props.getProperty("groupsFile");
          }

      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      }

      if (USERFILENAME == null || GROUPFILENAME == null) {
         throw new CustomRegistryException("users/groups information missing");
      }

   }

  /**
   * Checks the password of the user. This method is called to authenticate a
   * user when the user's name and password are given.
   *
   * @param     userSecurityName the name of user
   * @param     password the password of the user
   * @return    a valid <code>userSecurityName. Normally this is
   *            the name of same user whose password was checked but if the
   *            implementation wants to return any other valid
   *            <code>userSecurityName in the registry it can do so
   * @exception CheckPasswordFailedException if <code>userSecurityName/
   *            <code>password combination does not exist in the registry
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public String checkPassword(String userSecurityName, String passwd)
      throws PasswordCheckFailedException,
             CustomRegistryException {
      String s,userName = null;
      BufferedReader in = null;

      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.indexOf(":",index+1);
               // check if the userSecurityName:passwd combination exists
               if ((s.substring(0,index)).equals(userSecurityName) &&
                       s.substring(index+1,index1).equals(passwd)) {
                  // Authentication successful, return the userId.
                  userName = userSecurityName;
                  break;
               }
            }
         }
      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }


      if (userName == null) {
         throw new PasswordCheckFailedException(userSecurityName);
      }

      return userName;
   }

  /**
   * Maps a Certificate (of X509 format) to a valid user in the Registry.
   * This is used to map the name in the certificate supplied by a browser
   * to a valid <code>passworduserSecurityName in the registry
   *
   * @param     cert the X509 certificate chain
   * @return    the mapped name of the user <code>userSecurityName
   * @exception CertificateMapNotSupportedException if the particular
   *            certificate is not supported.
   * @exception CertificateMapFailedException if the mapping of the
   *            certificate fails.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public String mapCertificate(X509Certificate[] cert)
      throws CertificateMapNotSupportedException,
             CertificateMapFailedException,
             CustomRegistryException {
      String name=null;
      X509Certificate cert1 = cert[0];
      try {
         // map the SubjectDN in the certificate to a userID.
         name = cert1.getSubjectDN().getName();
      } catch(Exception ex) {
         throw new CertificateMapNotSupportedException(ex.getMessage());
      }

      if(!isValidUser(name)) {
         throw new CertificateMapFailedException(name);
      }
      return name;
   }

  /**
   * Returns the realm of the registry.
   *
   * @return    the realm. The realm is a registry-specific string indicating
   *            the realm or domain for which this registry
   *            applies.  For example, for OS400 or AIX this would be the
   *            host name of the system whose user registry this object
   *            represents.
   *            If null is returned by this method realm defaults to the
   *            value of "customRealm".
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public String getRealm()
      throws CustomRegistryException {
      String name = "customRealm";
      return name;
   }

  /**
   * Gets a list of users that match a <code>pattern in the registy.
   * The maximum number of users returned is defined by the <code>limit
   * argument.
   * This method is called by GUI(adminConsole) and Scripting(Command Line) to
   * make available the users in the registry for adding them (users) to roles.
   *
   * @param     pattern the pattern to match. (For e.g., a* will match all
   *            userSecurityNames starting with a)
   * @param     limit the maximum number of users that should be returned.
   *            This is very useful in situations where there are thousands of
   *            users in the registry and getting all of them at once is not
   *            practical. The default is 100. A value of 0 implies get all the
   *            users and hence must be used with care.
   * @return    a <code>Result object that contains the list of users
   *            requested and a flag to indicate if more users exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public Result getUsers(String pattern, int limit)
      throws CustomRegistryException {
      String s;
      BufferedReader in = null;
      List allUsers = new ArrayList();
      Result result = new Result();
      int count = 0;
      int newLimit = limit+1;
      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               String user = s.substring(0,index);
               if (match(user,pattern)) {
                  allUsers.add(user);
                  if (limit !=0 && ++count == newLimit) {
                     allUsers.remove(user);
                     result.setHasMore();
                     break;
                  }
               }
            }
         }
      } catch (Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }

      result.setList(allUsers);
      return result;
   }

  /**
   * Returns the display name for the user specified by userSecurityName.
   *
   * This method may be called only when the user information is displayed
   * (i.e information purposes only, for example, in GUI) and hence not used
   * in the actual authentication or authorization purposes. If there are no
   * display names in the registry return null or empty string.
   *
   * In WAS 4.0 custom registry, if you had a display name for the user and
   * if it was different from the security name, the display name was
   * returned for the EJB methods getCallerPrincipal() and the servlet methods
   * getUserPrincipal() and  getRemoteUser().
   * In WAS 5.0 for the same methods the security name will be returned by
   * default. This is the recommended way as the display name is not unique
   * and might create security holes.
   * However, for backward compatability if one needs the display name to
   * be returned set the property WAS_UseDisplayName to true.
   *
   * See the Infocenter documentation for more information.
   *
   * @param     userSecurityName the name of the user.
   * @return    the display name for the user. The display name
   *            is a registry-specific string that represents a descriptive, not
   *            necessarily unique, name for a user. If a display name does
   *            not exist return null or empty string.
   * @exception EntryNotFoundException if userSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public String getUserDisplayName(String userSecurityName)
      throws CustomRegistryException,
             EntryNotFoundException {

      String s,displayName = null;
      BufferedReader in = null;
```

```
       if(!isValidUser(userSecurityName)) {
          EntryNotFoundException nsee = new EntryNotFoundException(userSecurityName);
          throw nsee;
       }

       try {
          in = fileOpen(USERFILENAME);
          while ((s=in.readLine())!=null)
          {
             if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                int index = s.indexOf(":");
                int index1 = s.lastIndexOf(":");
                if ((s.substring(0,index)).equals(userSecurityName)) {
                   displayName = s.substring(index1+1);
                   break;
                }
             }
          }
       } catch(Exception ex) {
          throw new CustomRegistryException(ex.getMessage());
       } finally {
          fileClose(in);
       }

       return displayName;
    }

   /**
    * Returns the UniqueId for a userSecurityName. This method is called when
    * creating a credential for a user.
    *
    * @param     userSecurityName the name of the user.
    * @return    the UniqueId of the user. The UniqueId for an user is
    *            the stringified form of some unique, registry-specific, data
    *            that serves to represent the user.  For example, for the UNIX
    *            user registry, the UniqueId for a user can be the UID.
    * @exception EntryNotFoundException if userSecurityName does not exist.
    * @exception CustomRegistryException if there is any registry specific
    *            problem
    **/
    public String getUniqueUserId(String userSecurityName)
       throws CustomRegistryException,
              EntryNotFoundException {

       String s,uniqueUsrId = null;
       BufferedReader in = null;
       try {
          in = fileOpen(USERFILENAME);
          while ((s=in.readLine())!=null)
          {
             if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                int index = s.indexOf(":");
                int index1 = s.indexOf(":", index+1);
                if ((s.substring(0,index)).equals(userSecurityName)) {
                   int index2 = s.indexOf(":", index1+1);
                   uniqueUsrId = s.substring(index1+1,index2);
                   break;
                }
             }
          }
       } catch(Exception ex) {
          throw new CustomRegistryException(ex.getMessage());
       } finally {
          fileClose(in);
       }
```

```
      if (uniqueUsrId == null) {
         EntryNotFoundException nsee = new EntryNotFoundException(userSecurityName);
         throw nsee;
      }

      return uniqueUsrId;
   }

  /**
   * Returns the name for a user given its uniqueId.
   *
   * @param     uniqueUserId the UniqueId of the user.
   * @return    the userSecurityName of the user.
   * @exception EntryNotFoundException if the uniqueUserId does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public String getUserSecurityName(String uniqueUserId)
      throws CustomRegistryException,
             EntryNotFoundException {
      String s,usrSecName = null;
      BufferedReader in = null;
      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.indexOf(":", index+1);
               int index2 = s.indexOf(":", index1+1);
               if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
                  usrSecName = s.substring(0,index);
                  break;
               }
            }
         }
      } catch (Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }

      if (usrSecName == null) {
         EntryNotFoundException ex =
            new EntryNotFoundException(uniqueUserId);
         throw ex;
      }

      return usrSecName;

   }

  /**
   * Determines if the <code>userSecurityName exists in the registry
   *
   * @param     userSecurityName the name of the user
   * @return    true if the user is valid. false otherwise
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public boolean isValidUser(String userSecurityName)
      throws CustomRegistryException {
      String s;
      boolean isValid = false;
      BufferedReader in = null;
      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               if ((s.substring(0,index)).equals(userSecurityName)) {
                  isValid=true;
                  break;
               }
            }
         }
      } catch (Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }

      return isValid;
   }


  /**
   * Gets a list of groups that match a <code>pattern in the registy.
   * The maximum number of groups returned is defined by the <code>limit
   * argument.
   * This method is called by GUI(adminConsole) and Scripting(Command Line) to
   * make available the groups in the registry for adding them (groups) to
   * roles.
   *
   * @param     pattern the pattern to match. (For e.g., a* will match all
   *            groupSecurityNames starting with a)
   * @param     limit the maximum number of groups that should be returned.
   *            This is very useful in situations where there are thousands of
   *            groups in the registry and getting all of them at once is not
   *            practical. The default is 100. A value of 0 implies get all the
   *            groups and hence must be used with care.
   * @return    a <code>Result object that contains the list of groups
   *            requested and a flag to indicate if more groups exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public Result getGroups(String pattern, int limit)
      throws CustomRegistryException {
      String s;
      BufferedReader in = null;
      List allGroups = new ArrayList();
      Result result = new Result();
      int count = 0;
      int newLimit = limit+1;
      try {
         in = fileOpen(GROUPFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               String group = s.substring(0,index);
               if (match(group,pattern)) {
                  allGroups.add(group);
                  if (limit !=0 && ++count == newLimit) {
                     allGroups.remove(group);
                     result.setHasMore();
                     break;
                  }
               }
            }
         }
      } catch (Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }

      result.setList(allGroups);
      return result;
   }

  /**
   * Returns the display name for the group specified by groupSecurityName.
   * For this version of WebSphere the only usage of this method is by the
   * clients (GUI and Scripting) to present a descriptive name of the user
   * if it exists.
   *
   * @param     groupSecurityName the name of the group.
   * @return    the display name for the group. The display name
   *            is a registry-specific string that represents a descriptive, not
   *            necessarily unique, name for a group. If a display name does
   *            not exist return null or empty string.
   * @exception EntryNotFoundException if groupSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public String getGroupDisplayName(String groupSecurityName)
      throws CustomRegistryException,
             EntryNotFoundException {
      String s,displayName = null;
      BufferedReader in = null;

      if(!isValidGroup(groupSecurityName)) {
         EntryNotFoundException nsee = new EntryNotFoundException(groupSecurityName);
         throw nsee;
      }

      try {
         in = fileOpen(GROUPFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.lastIndexOf(":");
               if ((s.substring(0,index)).equals(groupSecurityName)) {
                  displayName = s.substring(index1+1);
                  break;
               }
            }
         }
      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }

      return displayName;
   }

  /**
   * Returns the Unique id for a group.

   * @param     groupSecurityName the name of the group.
   * @return    the Unique id of the group. The Unique id for
   *            a group is the stringified form of some unique,
   *            registry-specific, data that serves to represent the group.
   *            For example, for the Unix user registry, the Unique id could
   *            be the GID.
   * @exception EntryNotFoundException if groupSecurityName does not exist.
```

```
 * @exception CustomRegistryException if there is any registry specific
 *           problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getUniqueGroupId(String groupSecurityName)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,uniqueGrpId = null;
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             if ((s.substring(0,index)).equals(groupSecurityName)) {
                uniqueGrpId = s.substring(index+1,index1);
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage());
    } finally {
       fileClose(in);
    }

    if (uniqueGrpId == null) {
       EntryNotFoundException nsee = new EntryNotFoundException(groupSecurityName);
       throw nsee;
    }

    return uniqueGrpId;
 }

/**
 * Returns the Unique ids for all the groups that contain the UniqueId of
 * a user. Called during creation of a user's credential.
 *
 * @param     uniqueUserId the uniqueId of the user.
 * @return    a List of all the group UniqueIds that the uniqueUserId
 *           belongs to. The Unique id for an entry is the stringified
 *           form of some unique, registry-specific, data that serves
 *           to represent the entry.  For example, for the
 *           Unix user registry, the Unique id for a group could be the GID
 *           and the Unique Id for the user could be the UID.
 * @exception EntryNotFoundException if uniqueUserId does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *           problem
 **/
 public List getUniqueGroupIds(String uniqueUserId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,uniqueGrpId = null;
    BufferedReader in = null;
    List uniqueGrpIds=new ArrayList();
    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             int index2 = s.indexOf(":", index1+1);
             if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
                int lastIndex = s.lastIndexOf(":");
                String subs = s.substring(index2+1,lastIndex);
                StringTokenizer st1 = new StringTokenizer(subs, ",");
                while (st1.hasMoreTokens())
                   uniqueGrpIds.add(st1.nextToken());
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage());
    } finally {
       fileClose(in);
    }

    return uniqueGrpIds;
 }

/**
 * Returns the name for a group given its uniqueId.
 *
 * @param     uniqueGroupId the UniqueId of the group.
 * @return    the name of the group.
 * @exception EntryNotFoundException if the uniqueGroupId does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 **/
 public String getGroupSecurityName(String uniqueGroupId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,grpSecName = null;
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {

          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             if ((s.substring(index+1,index1)).equals(uniqueGroupId)) {
                grpSecName = s.substring(0,index);
                break;
             }
          }
       }
    } catch (Exception ex) {
       throw new CustomRegistryException(ex.getMessage());
    } finally {
       fileClose(in);
    }

    if (grpSecName == null) {
       EntryNotFoundException ex =
          new EntryNotFoundException(uniqueGroupId);
       throw ex;
    }

    return grpSecName;

 }

/**
 * Determines if the <code>groupSecurityName exists in the registry
 *
 * @param     groupSecurityName the name of the group
 * @return    true if the groups exists, false otherwise
 * @exception CustomRegistryException if there is any registry specific
 *           problem
 **/
 public boolean isValidGroup(String groupSecurityName)
    throws CustomRegistryException {
    String s;
    boolean isValid = false;
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             if ((s.substring(0,index)).equals(groupSecurityName)) {
                isValid=true;
                break;
             }
          }
       }
    } catch (Exception ex) {
       throw new CustomRegistryException(ex.getMessage());
    } finally {
       fileClose(in);
    }

    return isValid;
 }

/**
 * Returns the securityNames of all the groups that contain the user
 *
 * This method is called by GUI(adminConsole) and Scripting(Command Line)
 * to verify the user entered for RunAsRole mapping belongs to that role
 * in the roles to user mapping. Initially, the check is done to see if
 * the role contains the user. If the role does not contain the user
 * explicitly, this method is called to get the groups that this user
 * belongs to so that check can be made on the groups that the role contains.
 *
 * @param     userSecurityName the name of the user
 * @return    a List of all the group securityNames that the user
 *           belongs to.
 * @exception EntryNotFoundException if user does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *           problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public List getGroupsForUser(String userName)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s;
    List grpsForUser = new ArrayList();
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             StringTokenizer st = new StringTokenizer(s, ":");
             for (int i=0; i<2; i++)
                st.nextToken();
             String subs = st.nextToken();
             StringTokenizer st1 = new StringTokenizer(subs, ",");
             while (st1.hasMoreTokens()) {
                if((st1.nextToken()).equals(userName)) {
                   int index = s.indexOf(":");
                   grpsForUser.add(s.substring(0,index));
                }
             }
          }
       }
    } catch (Exception ex) {
       if (!isValidUser(userName)) {
          throw new EntryNotFoundException(userName);
       }
       throw new CustomRegistryException(ex.getMessage());
    } finally {
       fileClose(in);
    }

    return grpsForUser;
 }

/**
 * Gets a list of users in a group.
 *
 * The maximum number of users returned is defined by the limit
 * argument.
 *
 * This method is not used by WebSphere Application Server (WAS) for
 * authenticating or authorization purposes. This is, however, used by some
 * of the WAS clients like Workflow.
 *
 * If you are working with a registry where getting all the users from
 * any of your groups is not practical (for example if there are a large
 * number of users) you can through the NotImplementedException. Also,
 * if you implement this method, you can still throw this exception if
 * the limit exceeds some practical value.
 * When the NotImplementedException is thrown the client program should fall
 * back to some default implementation which should be documented by the
 * client.
 *
 * @param groupSecurityName the name of the group
 * @param limit the maximum number of users that should be returned.
 *  This is very useful in situations where there are lot of
 *           users in the registry and getting all of them at once is not
 *           practical. A value of 0 implies get all the users and hence
 *           must be used with care.
 * @return a Result object that contains the list of users
 *   requested and a flag to indicate if more users exist.
 * @deprecated This method will be deprecated in future.
 * @exception NotImplementedException throw this exception if it is not
 *           pratical to get this information from your registry.
 * @exception EntryNotFoundException if the group does not exist in
 *           the registry
 * @exception CustomRegistryException if there is any registry specific
 *           problem
 **/
 public Result getUsersForGroup(String groupSecurityName, int limit)
    throws NotImplementedException,
           EntryNotFoundException,
           CustomRegistryException {
    String s, user;
    BufferedReader in = null;
    List usrsForGroup = new ArrayList();
    int count = 0;
    int newLimit = limit+1;
    Result result = new Result();

    // As mentioned in the javadoc if the registry cannot handle a
    // large limit value it can throw the NotImplementedException.
    // For eg.
    if (limit > 50)
       throw new NotImplementedException("Limit exceeds 50");
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             if ((s.substring(0,index)).equals(groupSecurityName))
             {
                StringTokenizer st = new StringTokenizer(s, ":");
                for (int i=0; i<2; i++)
                   st.nextToken();
                String subs = st.nextToken();
                StringTokenizer st1 = new StringTokenizer(subs, ",");
                while (st1.hasMoreTokens()) {
                   user = st1.nextToken();
                   usrsForGroup.add(user);
                   if (limit !=0 && ++count == newLimit) {
                      usrsForGroup.remove(user);
                      result.setHasMore();
                      break;
                   }
                }
             }
          }
       }
    } catch (Exception ex) {
       if (!isValidGroup(groupSecurityName)) {
          throw new EntryNotFoundException(groupSecurityName);
       }
       throw new CustomRegistryException(ex.getMessage());
    } finally {
       fileClose(in);
    }

    result.setList(usrsForGroup);
    return result;
 }

/**
 * Create Credential for a user. For this version of WebSphere one should
 * throw an NotImplementedException. This will be implemented internally
 * by WebSphere code and should not be implemented by the Custom Registry.
 *
 * @param     userSecurityName the name of the user.
 * @return    com.ibm.websphere.security.cred.WSCredential
 * @exception CustomRegistryException if there is any problem.
 * @exception EntryNotFoundException if the uniqueGroupId does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 **/
 public com.ibm.websphere.security.cred.WSCredential createCredential(String userSecurityName)
       throws CustomRegistryException,
              NotImplementedException,
              EntryNotFoundException {
    NotImplementedException ex =
                   new NotImplementedException("createCredential not implemented");
    throw ex;
 }

 // private methods
 private BufferedReader fileOpen(String fileName)
    throws FileNotFoundException {
    try {
       return new BufferedReader(new FileReader(fileName));
    } catch(FileNotFoundException e) {
       throw e;
    }
```

```
   }

   private void fileClose(BufferedReader in) {
      try {
         if (in != null) in.close();
      } catch(Exception e) {
         System.out.println("Error closing file" + e);
      }
   }

   private boolean match(String name, String pattern) {
      RegExpSample regexp = new RegExpSample(pattern);
      boolean matches = false;
      if(regexp.match(name))
          matches = true;
      return matches;
   }
}


//----------------------------------------------------------------------
// The program provides the Regular Expression implementation used in the
// Sample for the Custom User Registry (FileRegistrySample). The pattern
// matching in the sample uses this program to search for the pattern (for
// users and groups).
//----------------------------------------------------------------------

class RegExpSample
{

    private boolean match(String s, int i, int j, int k)
    {
        for(; k < expr.length; k++)
label0:
            {
                Object obj = expr[k];
                if(obj == STAR)
                {
                    if(++k >= expr.length)
                        return true;
                    if(expr[k] instanceof String)
                    {
                        String s1 = (String)expr[k++];
                        int l = s1.length();
                        for(; (i = s.indexOf(s1, i)) >= 0; i++)
                            if(match(s, i + l, j, k))
                                return true;

                        return false;
                    }
                    for(; i < j; i++)
                        if(match(s, i, j, k))
                            return true;

                    return false;
                }
                if(obj == ANY)
                {
                    if(++i > j)
                        return false;
                    break label0;
                }
                if(obj instanceof char[][])
                {
                    if(i >= j)
                        return false;
                    char c = s.charAt(i++);
                    char ac[][] = (char[][])obj;
                    if(ac[0] == NOT)
                    {
                        for(int j1 = 1; j1 < ac.length; j1++)
                            if(ac[j1][0] <= c && c <= ac[j1][1])
                                return false;

                        break label0;
                    }
                    for(int k1 = 0; k1 < ac.length; k1++)
                        if(ac[k1][0] <= c && c <= ac[k1][1])
                            break label0;

                    return false;
                }
                if(obj instanceof String)
                {
                    String s2 = (String)obj;
                    int i1 = s2.length();
                    if(!s.regionMatches(i, s2, 0, i1))
                        return false;
                    i += i1;
                }
            }

        return i == j;
    }

    public boolean match(String s)
    {
        return match(s, 0, s.length(), 0);
    }

    public boolean match(String s, int i, int j)
    {
        return match(s, i, j, 0);
    }

    public RegExpSample(String s)
    {
        Vector vector = new Vector();
        int i = s.length();
        StringBuffer stringbuffer = null;
        Object obj = null;
        for(int j = 0; j < i; j++)
        {
            char c = s.charAt(j);
            switch(c)
            {
            case 63: /* '?' */
                obj = ANY;
                break;

            case 42: /* '*' */
                obj = STAR;
                break;

            case 91: /* '[' */
                int k = ++j;
                Vector vector1 = new Vector();
                for(; j < i; j++)
                {
                    c = s.charAt(j);
                    if(j == k && c == '^')
                    {
                        vector1.addElement(NOT);
                        continue;
                    }
                    if(c == '\\')
                    {
                        if(j + 1 < i)
                            c = s.charAt(++j);
                    }
                    else
                    if(c == ']')
                        break;
                    char c1 = c;
                    if(j + 2 < i && s.charAt(j + 1) == '-')
                        c1 = s.charAt(j += 2);
                    char ac1[] = {
                        c, c1
                    };
                    vector1.addElement(ac1);
                }

                char ac[][] = new char[vector1.size()][];
                vector1.copyInto(ac);
                obj = ac;
                break;

            case 92: /* '\\' */
                if(j + 1 < i)
                    c = s.charAt(++j);
                break;

            }
            if(obj != null)
            {
                if(stringbuffer != null)
                {
                    vector.addElement(stringbuffer.toString());
                    stringbuffer = null;
                }
                vector.addElement(obj);
                obj = null;
            }
            else
            {
                if(stringbuffer == null)
                    stringbuffer = new StringBuffer();
                stringbuffer.append(c);
            }
        }

        if(stringbuffer != null)
            vector.addElement(stringbuffer.toString());
        expr = new Object[vector.size()];
        vector.copyInto(expr);
    }

    static final char NOT[] = new char[2];
    static final Integer ANY = new Integer(0);
    static final Integer STAR = new Integer(1);
    Object expr[];

}
```

Users.prop file: Following is the format for the `users.prop` file:

Usage scenario

```
# 5639-D57, 5630-A36, 5630-A37, 5724-D18 (C) COPYRIGHT International Business Machines Corp.
1997, 2002
# All Rights Reserved * Licensed Materials - Property of IBM
```

```
#
# Format:
# name:passwd:uid:gids:display name
# where name   = userId/userName of the user
#       passwd = password of the user
#       uid    = uniqueId of the user
#       gid    = groupIds of the groups that the user belongs to
#       display name = a (optional) display name for the user.
bob:bob1:123:567:bob
dave:dave1:234:678:
jay:jay1:345:678,789:Jay-Jay
ted:ted1:456:678:Teddy G
jeff:jeff1:222:789:Jeff
vikas:vikas1:333:789:vikas
bobby:bobby1:444:789:
```

Groups.prop file: Following is the format for the `groups.prop` file:

Usage scenario

```
# 5639-D57, 5630-A36, 5630-A37, 5724-D18 (C) COPYRIGHT International Business Machines Corp.
1997, 2002
# All Rights Reserved * Licensed Materials - Property of IBM
#
# Format:
# name:gid:users:display name
# where name   = groupId of the group
#       gid    = uniqueId of the group
#       users  = list of all the userIds that the group contains
#       display name = a (optional) display name for the group.
admins:567:bob:Administrative group
operators:678:jay,ted,dave:Operators group
users:789:jay,jeff,vikas,bobby:
```

Result.java file: Usage scenario

```
//  @(#) 1.20 src/en/ae/rsec_result.xml, WEBSJAVA.INFO.DOCSRC, ASVINFO1 10/17/02 16:43:01
[10/18/02 07:31:30]
//  5639-D57, 5630-A36, 5630-A37, 5724-D18 (C) COPYRIGHT International Business Machines Corp.
1997, 2002
//  All Rights Reserved * Licensed Materials - Property of IBM
//
//  DESCRIPTION:
//
//    This module is used by User Registries in WebSphere when calling the
//    getUsers and getGroups method. The user registries should use this
//    to set the list of users/groups and to indicate if there are more
//    users/groups in the registry than requested
//
package com.ibm.websphere.security;

import java.util.List;

/**
    This module is used by User Registries in WebSphere when calling the
    getUsers and getGroups method. The user registries should use this
    to set the list of users/groups and to indicate if there are more
    users/groups in the registry than requested
 */

public class Result implements java.io.Serializable {
    /**
      Default constructor
    */
    public Result() {
    }
    /**
       Returns the list of users/groups
       @return the list of users/groups
    */
    public List getList() {
      return list;
    }

    /**
       indicates if there are more users/groups in the registry
    */
    public boolean hasMore() {
      return more;
    }
    /**
       Set the flag to indicate that there are more users/groups in the registry to true
    */
    public void setHasMore() {
      more = true;
    }

    /*
      Set the list of user/groups
      @param list   list of users/groups
    */
    public void setList(List list) {
      this.list = list;
    }

    private boolean more = false;
    private List list;
}
```

Custom user registry settings: Use this page to configure the custom user registry.

To view this administrative console page, click **Security** > **User Registries** > **Custom User Registry**.

Once the properties are set in this panel, click **Apply**. Additional properties that the custom registry requires should be added using the properties panel. When security is enabled and any of these properties are changed, go to the **Global Security** panel and click **Apply** to validate the changes.

Server ID: Specifies the user ID under which the server runs, for security purposes.

This server ID should be a valid user in the custom registry.

Data type	String

Server Password: Specifies the password corresponding to the security server ID.

Data type	String

Custom Registry ClassName: Specifies a dot-separated class name that implements the `com.ibm.websphere.security.UserRegistry interface`.

The custom registry class name should be in the classpath. A suggested location is the %WAS_HOME%/classes directory. Although the custom registry implements the `com.ibm.websphere.security.UserRegistry` interface, for backward compatibility, a user registry can alternately implement the `com.ibm.websphere.security.CustomRegistry` interface.

Data type	String
Default	com.ibm.websphere.security.FileRegistrySample

Ignore Case: Specifies that a case insensitive authorization check will be performed.

Default	Enabled
Range	Enabled or Disabled

Use the Custom Properties link to add any additional properties required to initialize the custom registry. The following two properties are pre-defined by the product, so set them only when required:

- WAS_UseDisplayName—When set to **true**, the methods `getCallerPrincipal()`, `getUserPrincipal()`, `getRemoteUser()` will return the display name. By default, the `securityName` of the user is returned. This is primarily introduced to support backward compatibility with the Version 4.0 custom registry.
- WAS_UseDataSource—Set this property to **true** if the data source is used to connect to the database in the custom registry implementation.

Java Authentication and Authorization Service

The standard Java 2 Security API provides a means to enforce access control, based on the location of the code and who signed it. The current principal of the thread of execution is not considered in the Java 2 Security authorization. There are instances where it is useful for the authorization to be based on the principal, rather the code base and the signer. The Java Authentication and Authorization Services is a standard Java API that allows the Java 2 Security authorization to be extend to the code base on the principal as well as the code base and signers.

The Java Authentication and Authorization Service (JAAS) Version 1.0 extends the Java 2 Security architecture of the Java 2 Platform with additional support to authenticate and enforce access control upon users. It implements a Java version of the standard Pluggable Authentication Module (PAM) framework, and extends the access control architecture of the Java 2 platform in a compatible fashion to support user-based authorization. WebSphere Application Server fully supports the JAAS architecture and extends the access control architecture to support role-based authorization for J2EE resources including servlets, JSP files, and EJB components.

This following sections cover the product JAAS authentication and authorization implementation and programming model.

- Java Authentication and Authorization Service login configuration
- Programmatic login
- Java Authentication and Authorization Service custom login module
- Java Authentication and Authorization Service authorization

Additionally the accompany product Javadoc contains detailed descriptions of the WebSphere Application Server programming APIs and the JAAS Javadoc is shipped with the product. Refer to the `${was.install.root}/web/docs/jaas` directory.

Java Authentication and Authorization Service authorization

Java 2 security architecture uses a security policy to specify which access rights are granted to executing code. That architecture is *code-centric*. That is, the permissions are granted based on code characteristics: where the code is coming from and whether it is digitally signed and by whom. JAAS authorization augments the existing code-centric access controls with new user-centric access controls. Permissions can be granted based not just on what code is running, but also on who is running it.

When using JAAS authentication to authenticate a user, a subject is created, which represents the authenticated user. A subject is comprised of a set of principals, where each principal represents an identity for that user. You can grant permissions in the policy to specific principals. After the user has been authenticated, the application can associate the subject with the current access control context. For each subsequent security-checked operation, the Java run time automatically determines whether the policy grants the required permission only to a specific principal. If so, the operation is be allowed only if the subject associated with the access control context contains the designated principal.

Associate a subject with the current access control context, by calling the static doAs method from the subject class, passing it an authenticated subject and java.security.PrivilegedAction or java.security.PrivilegedExceptionAction. The doAs method associates the provided subject with the current access control context and then invokes the run method from the action. The run method implementation contains all the code executed as the specified subject. The action executes as the specified subject.

In the J2EE programming model, when invoking the EJB method from an enterprise bean or servlet, the execution is under the user identity that is determined by the run-as setting. The J2EE V 1.3 Specification does not specify which user identity to use when invoking an enterprise bean from a Subject.doAs action block within either the EJB code or the servlet code. A logical extension is to use the proper identity specified in the subject when invoking the EJB method within the Subject doAs action block.

This simple rule of letting Subject.doAs overwrite the run-as identity setting would have been an ideal way to integrate the JAAS programming model with the J2EE run time environment. However, a design oversight was introduced into JDK V1.3 when integrating the JAAS V 1.0 implementation with the Java 2 security architecture. A subject, which is associated with the access control context is cut off by a doPrivileged call when a doPrivileged occurs within the Subject.doAs action block. The same behavior is also observed in JDK V 1.4 where JAAS becomes a standard feature. Until this problem is corrected, there is no reliable and run-time efficient way to guarantee the correct behavior of Subject.doAs in aJ2EE run-time environment.

The problem can be explained better with the following example: (Some lines have been split for publication.)

```
Subject.doAs(subject, new java.security.PrivilegedAction() {
       Public Object run() {
           // Subject is associated with the current thread context
           java.security.AccessController.doPrivileged( new
               java.security.PrivilegedAction() {
                                   public Object run() {
                                        // Subject was cut off from the current
              thread context
       return null;
```

```
                }
            });
            // Subject is associated with the current thread context
            return null;
        }
    });
```

At line three, the subject object is associated with the context of the current execution thread. As indicated on line 7 within the run method of a doPrivileged action block, the subject object is removed from the thread context. After leaving the doPrivileged block, the subject object is restored to the current thread context. Since doPrivileged blocks can be placed anywhere along the execution path and instrumented quite often in a server environment, the run-time behavior of a doAs action block becomes difficult to manage.

To resolve this difficulty, WebSphere Application Server provides a helper class, WSSubject, to extend the JAAS authorization to a J2EE EJB method invocation as described above. WSSubject class provides static doAs and doAsPrivileged methods that have identical signatures to the subject class. The WSSubject.doAs method basically associates the WSPrincipal, WSCredential, and the CORBA credential to the current execution thread. The credential is used by the SAS run time for EJB invocation. The WSSubject.doAs and WSSubject.doAsPrivileged methods then invoke the corresponding Subject.doAs and Subject.doAsPrivileged methods. The original credential is restored and associated with the execution thread upon leaving the WSSubject.doAs and WSSubject.doAsPrivileged methods.

Note that WSSubject is not a replacement of the subject object, but rather a helper class to ensure consistent run-time behavior as long as EJB method invocation is a concern.

The following example illustrates the runtime behavior of the WSSubject.doAs method:

```
WSSubject.doAs(subject, new java.security.PrivilegedAction() {
                        // Subject's CORBA Credentials
         is associated with SAS thread local storage
       Public Object run() {
           // Subject is associated with the current thread context
           java.security.AccessController.doPrivileged( new
               java.security.PrivilegedAction() {
                                   public Object run() {
                                        // Subject was cut off from the
               current thread context, but
                                       //  nonetheless its CORBA Credentials
               is still associated with
                                       //  SAS thread local storage
       return null;
                 }
            });
            // Subject is associated with the current thread context and its CORBA
                            //  Credentials is still associated with SAS thread
            local storage
            return null;
        }
    });
              // Subject's CORBA Credential is removed from SAS thread local storage and the
              // original CORBA Credentials is restored.
```

The Subject.doAs and Subject.doAsPrivileged methods are not integrated with the J2EE run-time environment. EJB methods that are invoked within the Subject.doAs and Subject.doAsPrivileged action blocks are executed under the identity specified by the run-as setting and not by the subject identity.

Configuring Java Authentication and Authorization Service login configuration

Before you begin

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server. It is also mandated by J2EE 1.3 Specification. JAAS is a collection of WebSphere Application Server strategic authentication APIs for and replace the CORBA programmatic login APIs. WebSphere Application Server has provided some extensions to JAAS:

- **com.ibm.websphere.security.auth.WSSubject**. Due to a design oversight in JAAS V1.0, javax.security.auth.Subject.getSubject() does not return the subject associated with the thread of execution inside a java.security.AccessController.doPrivileged() code block. This presents an inconsistent behavior that is problematic and causes undesirable effort. The com.ibm.websphere.security.auth.WSSubject API provides a workaround to associate the subject to thread of execution. The com.ibm.websphere.security.auth.WSSubject API extends the JAAS authorization model to J2EE resources.
- You can configure JAAS login in the administrative console and stored in WebSphere Common Configuration Model (WCCM). However, WebSphere Application Server still supports the default JAAS login configuration format (plan text file) provided by the JAAS default implementation. If there are duplicate login configurations defined in both the WCCM and the plan text file format, the one in the WCCM takes precedence. There are advantages to defining the login configuration in the WCCM:
  - User Interface support in defining JAAS login configuration
  - You can manage the JAAS configuration login configuration centrally
  - The JAAS configuration login configuration is distributed in a Network Deployment product installation
- **Proxy LoginModule**. The default JAAS implementation does not use the thread context class loader to load classes. The LoginModule can not load if the LoginModule class file is not in the application class loader or the Java extension class loader class path. Due to this class loader visibility problem, WebSphere Application Server provides a proxy LoginModule to load JAAS LoginModule using the thread context class loader. You do not need to place the LoginModule implementation on the application class loader or the Java extension class loader classpath with this proxy LoginModule.

Note: Do not remove or delete the pre-defined JAAS login configurations (ClientContainer, WSLogin and DefaultPrincipalMapping). Deleting or removing them could cause other enterprise applications to fail.

Steps for this task

1. Delete a JAAS login configuration.
   a. Click **Security** in the navigation tree.
   b. Click **JAAS Configuration** > **Application Logins**.
      The Application Login Configuration panel appears.
   c. Select the checkbox for the login configurations to be deleted and click **Delete**.
2. Create a new JAAS login configuration.
   a. Click **Security** in the navigation tree.
   b. Click **JAAS Configuration** > **Application Logins**.

c. Click **New**.

   The Application Login Configuration panel appears.

d. Specify the alias name of the new JAAS login configuration and click **Apply**.

   This is the name of the login configuration that you pass in the javax.security.auth.login.LoginContext for creating a new LoginContext.

e. Click **JAAS Login Modules**

f. Click **New**.

g. Specify the **Module Classname**.

   It is recommended that you specify WebSphere Proxy LoginModule because of the limitation of the class loader visibility problem

h. Specify the **LoginModule** implementation as the delegate property of the Proxy LoginModule.

   The WebSphere Proxy LoginModule classname is com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy.

i. Select **Authentication Strategy** from the drop down list and click **Apply**.

j. Click **Custom Properties**.

   This will navigate to the Custom Properties panel for the selected LoginModule

k. Create a new property with the name `delegate` with the value being the real LoginModule implementation.

   You can specify other properties like `debug` with value true. These properties are passed to the LoginModule as options to initialize() method of the LoginModule.

l. Click **Save**.

   For Network Deployment installation, make sure a file sync operation is performed to propagate the changes to other nodes.

3. Change the plain text file.

   WebSphere Application Server supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. However, there is no tool provided to edit plain text file in this format. You can define the JAAS login configuration in the plain text file (`${user.install.root}/properties/wsjaas.conf`), any syntax errors can cause the plain JAAS Login Configuration text file to not parse correctly. This could cause other applications to fail.

   Note: Do not remove or delete the pre-defined JAAS login configurations (ClientContainer, WSLogin, system.SWAM and system.LTPA). Deleting or removing them could cause other enterprise applications to fail.

Results

Created new JAAS login configuration or old JAAS login configuration is removed. An enterprise application can use newly created JAAS login configuration without restarting the application server process.

However, if the new JAAS login configurations are defined in `${user.install.root}/properties/wsjaas.conf`, it is not refreshed automatically. Restart the application servers to validate changes. These JAAS login configurations are per node and are not available for other application servers running on other nodes.

Usage scenario

To create new JAAS login configurations used by enterprise applications to perform custom authentication.

What to do next

Use these newly defined JAAS login configurations to perform programmatic login.

Java Authentication and Authorization Service login configuration

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server. It is mandated by J2EE 1.3 Specification. JAAS is WebSphere strategic APIs for authentication and it will replace of the CORBA programmatic login APIs. WebSphere Application Server provides some extensions to JAAS:

- com.ibm.websphere.security.auth.WSSubject: Due to a design oversight in the JAAS 1.0, javax.security.auth.Subject.getSubject() does not return the Subject associated with the thread of execution inside a java.security.AccessController.doPrivileged() code block. This can present a inconsistent behavior that is problematic and causes undesirable effort. com.ibm.websphere.security.auth.WSSubject provides a work around to associate Subject to thread of execution. com.ibm.websphere.security.auth.WSSubject extends the JAAS authorization model to J2EE resources.
- JAAS Login Configuration can be configured in Admin Console and stored in WCCM (WebSphere Common Configuration Model): Application can define new JAAS login configuration in the Admin Console and the the data is persisted in the configuration respository (stored in the WCCM). However, WebSphere still support the default JAAS login configuration format (plan text file) provided by the JAAS default implementation. But if there are duplication login configurations defined in both the WCCM and the plan text file format, the one in the WCCM takes precedence. There are advantages to define the login configuration in the WCCM:
  - UI support in defining JAAS login configuration.
  - The JAAS configuration login configuration can be managed centrally.
  - The JAAS configuration login configuration is distributed in a Network Deployment installation.
- Proxy LoginModule: The default JAAS implementation does not use the thread context class loader to load classes, the LoginModule could not be loaded if the LoginModule class file is not in the application class loader or the Java extension class loader classpath. Due to this class loader visibility problem, WebSphere provides a proxy LoginModule to load JAAS LoginModule using the thread context class loader. The LoginModule implementation does not have to be placed on the application class loader or the Java extension class loader classpath with this proxy LoginModule.

Note: Please do not remove or delete the pre-defined JAAS Login Configurations (ClientContainer, WSLogin and DefaultPrincipalMapping). Deleting or removing them could cause other enterprise applications to fail.

A system administrator determines the authentication technologies, or LoginModules, to be used for each application and configures them in a login configuration. The source of the configuration information (for example, a file or a database) is up to the current javax.security.auth.login.Configuration implementation. The WebSphere Application Server implementation permits the login configuration to be defined in both the WebSphere Common Configuration Model (WCCM) security document and in a JAAS configuration file where the former takes precedence.

Two JAAS login configurations are defined in the WCCM security document for applications to use. They may be found in the left navigation pane at **Security** > **JAAS Configuration** > **Application Login Config**: **WSLogin** and **ClientContainer**. The **WSLogin** defines a login configuration and LoginModule implementation that may be used by applications in general. The **ClientContainer** defines a login configuration and LoginModule implementation that is similar to that of WSLogin but enforces the requirements of the WebSphere Application Server Client Container. The third entry, **DefaultPrincipalMapping**, defines a special LoginModule that is typically used by Java 2 Connector to map an authenticated WebSphere user identity to a set of user authentication data (user ID and password) for the specified back end Enterprise Information System (EIS). For more information about Java 2 Connector and the DefaultMappingModule please refer to the Java 2 Security section.

New JAAS login configuration may be added and modified using Security Center. The changes are saved in the cell level security document and are available to all managed application servers. An application server restart is required for the changes to take effect at runtime.

WebSphere Application Server also reads JAAS Configuration information from the wsjaas.conf file under the properties sub directory of the root directory under which WebSphere Application Server is installed. Changes made to the wsjaas.conf file is used only by the local application server and will take effect after restarting the application server. Note that JAAS configuration in the WCCM security document takes precedence over that defined in the `wsjaas.conf` file. In other words, a configuration entry in `wsjaas.conf` will be overridden by an entry of the same alias name in the WCCM security document.

Java Authentication and Authorization Service configuration entry settings

Use this page to specify a list of Java Authentication and Authorization Service (JAAS) login configurations for the application code to use, including enterprise beans, Java Server Pages (JSP) files, servlets and resource adapters.

To view this administrative console page, click **Security** > **JAAS Configuration** > **Application Login Configuration**.

Reading the JAAS documentation in the InfoCenter before you begin defining additional login modules that are used for authenticating to the WebSphere Application Server security run time is strongly recommended. You can define additional login configurations for your applications. However, if the WebSphere Application Server LoginModule (`com.ibm.ws.security.common.auth.module.WSLoginModuleImpl`) is not used or the LoginModule does not produce a credential that is recognized by WebSphere Application Server, then the WebSphere Application Server security run time cannot use the authenticated subject from these login configurations for an authorization check for resource access.

ClientContainer: Specifies the login configuration used by the client container application, which uses the CallbackHandler API defined in the client container deployment descriptor.

ClientContainer is the default login configuration for the WebSphere Application Server. Do not remove this default, as other applications that use them fail.

Default	ClientContainer

DefaultPrincipalMapping: Specifies the login configuration used by Java 2 Connectors to map users to principals that are defined in the **J2C Authentication Data Entries.**

DefaultPrincipalMapping is the default login configuration for the WebSphere Application Server. Do not remove this default, as other applications that use them fail. The DefaultPrincipalMapping login configuration authenticates users for the WebSphere Application Server security run time. Use credentials from the authenticated subject returned from this login configuration as an authorization check for access to WebSphere Application Server resources.

Default	ClientContainer

WSLogin: Specifies whether all applications can use the WSLogin configuration to perform authentication for the WebSphere Application Server security run time.

This login configuration does not honor the CallbackHandler defined in the client container deployment descriptor. To use this functionality, use the ClientContainer login configuration.

WSLogin is the default login configuration for the WebSphere Application Server. Do not remove this default, as other applications that use them fail. This login configuration authenticates users for the WebSphere Application Server security run time. Use credentials from the authenticated subject returned from this login configurations as an authorization check for access to WebSphere Application Server resources.

Default	ClientContainer

Java Authentication Authorization Service login module settings

Use this page to define the login module for a Java Authentication Authorization Service (JAAS) login configuration.

To view this administrative page, click **Security** > **JAAS Configuration** > **Application Login Configuration** > *alias_name* > **JAAS Login Modules**.

Module Class Name: Specifies the class name of the given login module.

The default login modules defined by the WebSphere product use a proxy LoginModule class, `com.ibm.ws.security.common.auth.module.WSLoginModuleProxy`. This proxy class loads the WebSphere login module with the thread context class loader and delegates all the operations to the *real* login module implementation. The real login module implementation is specified as the delegate option in the option configuration. The proxy class is needed because the Developer Kit application class loaders do not have visibility of the WebSphere product class loaders.

Data type	String

Authentication Strategy: Specifies the authentication behavior as authentication proceeds down the stack of login modules.

A JAAS authentication provider supplies the authentication strategy. In JAAS, an authentication strategy is implemented through the LoginModule interface.

Data type	String
Default	Required
Range	Required, Requisite, Sufficient and Optional

Specify additional options in **Options Additional Properties**. These name and value pairs are passed to the login modules during initialization. This is one of the mechanism used to passed information to login modules.

Application login configuration settings

Use this page to configure application login configurations.

To view this administrative console page, click **Security** > **JAAS Configuration** > **Application Login Configuration** > *alias_name*.

Alias: Specifies the alias name of the application login.

Data type	String

Java 2 Connector security

Java 2 Connection Authentication Data Entries are use by resource adapters and JDBC data source. A Java 2 Connection Authentication Data Entry contains authentication data.

The connector architecture defines a standard architecture for connecting the Java 2 Platform, Enterprise Edition (J2EE) to heterogeneous EISs. Examples of EIS include ERP, mainframe transaction processing (TP) and database systems.

The connector architecture enables an EIS vendor to provide a standard resource adapter for its EIS. A resource adapter is a system-level software driver that is used by a Java application to connect to an EIS. The resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application. Information in EIS must be protected from unauthorized access. The Java 2 Connector Security architecture is designed to extend the end-to-end security model for J2EE-based applications to include integration with EISs. An application server and an EIS collaborate to ensure the proper authentication of a resource principal, which establishes a connection to an underlying EIS. The connector architecture identifies the following as the commonly-supported authentication mechanisms:

- BasicPassword: Basic user-password-based authentication mechanism specific to an EIS
- Kerbv5: Kerberos Version 5-based authentication mechanism

WebSphere Application Server implementation of a Java 2 connection supports basic password authentication mechanisms.

The user ID and password for the target EIS is either supplied by applications, or by the application server. WebSphere Application Server uses a JAAS pluggable authentication mechanism to perform principal mapping to convert a WebSphere principal to a resource principal. WebSphere Application Server provides a

DefaultPrincipalMapping LoginModule, which basically converts any authenticated principal to the pre-configured EIS resource principal and password. Subsequently, you can plug in their principal mapping LoginModule through the JAAS plug-in mechanism.

J2C mapping module configuration

When a Java 2 Connection Factory is configured to container-managed sing-on, WebSphere Application Server uses the configured principal mapping module to create a Subject instance that contains a user ID and password for the target EIS.

Mapping modules are special JAAS login modules that provide principal and credential mapping functionality. You can define and configure custom mapping modules through the administrative console. Associated with the mapping module configuration is a set of user IDs and passwords that you can define in the security configuration with a specified alias name. The WebSphere Application Server run time passes the user ID, password and a reference of the connection factory manager to the configured mapping module to create a subject.

For more information about mapping module requirements, please refer to the Javadoc of the `WSDefaultPrincipalMapping` class. For more detailed information about developing a mapping module, refer to the (Developing your own Java 2 security mapping module) article.

J2C mapping module programming reference

You can develop your own mapping module if your application requires more sophisticated mapping functions. You can use the `WSSubject.getCallerPrincipal()` mehtod to retrieve the application client identity. The subject instance contains a `WSPrincipal` instance in the principals set and a `WSCredential` instance in the set of public credentials.

Managing J2C authentication data entries

This task creates and deletes Java 2 Connection Authentication data entries. Java 2 Connection Authentication data entries are use by resource adapters and JDBC data source. A Java 2 Connection Authentication Data Entry contains authentication data, in this case, the user identity and password. It contains the following information:

Alias An identifier used to identify the authenticated data entry. When configuring resource adapters or JDBC data sources, the administrator can specify which authentication data to choose for the corresponding alias.

User ID
A user identity of the intended security domain. For example, if a particular authentication data entry is intended to be used to open a new connection to DB2, this entry would contain a DB2 user identity.

Password
The password of the user identity is encoded in the configuration respository.

Description
A short text description.

Steps for this task

1. Delete a J2C Authentication data entries.

a. Click **Security** in the navigation tree, then click **JAAS Configuration** > **J2C Authentication Data**. This will navigate to the **J2C Authentication Data Entries** panel.
b. Select the check boxes for the entries to delete and click **Delete**.

   Note: Before deleting or removing an authentication data entry, make sure that the authentication data entry is not used or referenced by any resource adapter or JDBC data source. If the deleted authentication data entry is used or referenced by a resource adapter or JDBC data source, the application that uses the resource adapter or JDBC data source fails to connect to the resources.

2. Create a new J2C Authentication data entry.
   a. Click **Security** in the navigation tree, then click **JAAS Configuration** > **J2C Authentication Data**. This will navigate to the **J2C Authentication Data Entries** panel.
   b. Click **New**.
   c. Enter a unique alias, a value user ID, a valid password and a short description (optional).
   d. Click **OK** or **Apply**. Note, there is no validation for the user ID and password.
   e. Click **Save**. For a Network Deployment installation, make sure a file synchronized operation is performed to propagate the changes to other nodes.

Results

A new J2C Authentication data entry is created or an old entry is removed. You can use the newly created entry without restarting the application server process. Specifically, the authentication data is loaded by an application server when starting an application and is shared among applications in the same application server.

Usage scenario

This step defines authentication data that you can share among resource adapters and JDBC data sources.

What to do next

Use the authentication data entry defined in the resource adapters or JDBC data sources.

Java 2 Connector authentication data entry settings: Use this page as a central place for administrators to define authentication data, which includes user identities and passwords. These can reference authentication data entries by resource adapters, data sources and other configurations that require authentication data using an alias.

To view this administrative page, click **Security** > **JAAS Configuration** > **J2C Authentication Data Entries**.

Note: Be careful when deleting authentication data entries. If the deleted authentication data is used by other configurations, the initializing resources process fails.

Define a new authentication data entry by clicking **New**.

Alias: Specifies the name of the authentication data entry.

Data type	String
Units	String
Default	None

User ID: Specifies the user identity.

Data type	String

Password: Specifies the password.

Data type	String

Description: Specifies an optional description of the authentication data entry. For example, this authentication data entry is used to connect to DB2.

Data type	String

Programmatic login

A type of form login that supports application presentation site-specific login forms for the purpose of authentication.

When Java enterprise bean client applications require the user to provide identifying information, the writer of the application must collect that information and authenticate the user. You can broadly classify the work of the programmer in terms of where the actual user authentication is performed:

- In a client program
- In a server program

Users of Web applications can be prompted for authentication data in many ways. The login-config element in the Web application deployment descriptor defines the mechanism used to collect this information. Programmers who want to customize login procedures, rather than relying on general purpose devices like a 401 dialog window in a browser, can use a form based login to provide an application specific HTML form for collecting login information.

No authentication occurs unless WebSphere global security is enabled. Additionally, if you want to use form-based login for Web applications, you must specify `FORM` in the `auth-method` tag of the `login-config` element in the deployment descriptor of each Web application.

Applications can present site-specific login forms by using the WebSphere form-login type. The J2EE Specification defines form login as one of the authentication methods for Web applications. However, the Servlet 2.2 Specification does not define a mechanism for logging out. WebSphere Application Server extends J2EE by also providing a form-logout mechanism.

Java Authentication and Authorization Service programmatic login

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server. It is also mandated by the J2EE 1.3 Specification.

JAAS is a collection of WebSphere strategic authentication APIs and replace of the CORBA programmatic login APIs. WebSphere Application Server provides some extensions to JAAS:

Before you begin developing with programmatic login APIs, consider the following:

- For the pure Java client application or client container application, make sure that the host name and the port number of the target JNDI bootstrap properties are specified properly. See the Developing applications that use CosNaming (CORBA Naming interface) section for details.
- If the application uses custom JAAS login configuration, make sure that the custom JAAS login configuration is properly defined. See the Java Authentication and Authorization Service (JAAS) Login Configuration section for details.
- Some of the JAAS APIs are protected by Java 2 security permissions, if these APIs are used by application code, make sure that these permissions are added to the application `was.policy` file. See (Adding the was.policy file) to the application, (Using policytool to edit policy file) and (Configuring was.policy) sections for details. For more details of which APIs are protected by Java 2 Security permissions, check the IBM Developer Kit, Java edition; JAAS and the WebSphere public APIs Javadoc for more details. The following lists only the APIs used in the samples code provided in this documentation.
  - javax.security.auth.login.LoginContext constructors are protected by javax.security.auth.AuthPermission ″createLoginContext″.
  - javax.security.auth.Subject.doAs() and com.ibm.websphere.security.auth.WSSubject.doAs() are protected by javax.security.auth.AuthPermission ″doAs″.
  - javax.security.auth.Subject.doAsPrivileged() and com.ibm.websphere.security.auth.WSSubject.doAsPrivileged() are protected by javax.security.auth.AuthPermission ″doAsPrivileged″.
- com.ibm.websphere.security.auth.WSSubject: Due to a design oversight in the JAAS 1.0, javax.security.auth.Subject.getSubject() does not return the Subject associated with the thread of execution inside a java.security.AccessController.doPrivileged() code block. This can present an inconsistent behavior that is problematic and causes undesirable effort. The com.ibm.websphere.security.auth.WSSubject API provides a work around to associate Subject to thread of execution. The com.ibm.websphere.security.auth.WSSubject API extends the JAAS model to J2EE resources for authorization checks. The Subject associated with the thread of execution within com.ibm.websphere.security.auth.WSSubject.doAs() or com.ibm.websphere.security.auth.WSSubject.doAsPrivileged() code block is used for J2EE resources authorization checks.
- UI support for defining new JAAS login configuration: You can configure JAAS login configuration in the administrative console and store it in WCCM (WebSphere Common Configuration Model). Applications can define new JAAS login configuration in the administratiave console and the the data is persisted in the configuration respository (stored in the WCCM). However, WebSphere Application Server still supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. But if there are duplication login configurations defined in both the WCCM and the plain text file format, the one in the WCCM takes precedence. There are advantages to defining the login configuration in the WCCM:
  - UI support in defining JAAS login configuration.

- You can manage the JAAS configuration login configuration centrally.
- The JAAS configuration login configuration is distributed in a Network Deployment installation.

- WebSphere JAAS login configurations: WebSphere provides JAAS login configurations for application to perform programmatic authentication to the WebSphere security runtime. These WebSphere JAAS login configurations perform authentication to the WebSphere configured authentication mechanism (SWAM or LTPA) and user registry (Local OS, LDAP or Custom) based on the authentication data supplied. The authenticated Subject from these JAAS login configurations contain the required Principal and Credentials that can be used by WebSphere security runtime to perform authorization checks on J2EE role-based protected resources. Here is the JAAS login configurations provided by WebSphere:
  - WSLogin JAAS login configuration: A generic JAAS login configuration that a Java Client, client container application, servlet, JSP file, enterprise bean, and so on, can use to perform authentication based on a user ID and password, or a token to the the WebSphere security run time. However, this does not honor the CallbackHandler specified in the Client Container deployment descriptor.
  - ClientContainer JAAS login configuration: This JAAS login configuration honors the CallbackHandler specified in the client container deployment descriptor. The login module of this login configuration uses the CallbackHandler in the client container deployment descriptor if one is specified, even if the application code specified one CallbackHandler in the LoginContext. This is for client container application.

    Note: Subject authenticated with the previously mentioned JAAS login configurations contain a com.ibm.websphere.security.auth.WSPrincipal and a com.ibm.websphere.security.auth.WSCredential. If the authenticated Subject is passed the in com.ibm.websphere.security.auth.WSSubject.doAs() (or the other doAs() methods), the WebSphere security run time can perform authorization checks on J2EE resources, based on the Subject com.ibm.websphere.security.auth.WSCredential.
- Customer defined JAAS login configurations: You can define other JAAS login configurations. See Configuring Java Authentication and Authorization Service (JAAS) login configuration section for details. Use these login configurations to perform programmatic authentication to the customer authentication mechanism. However, the subjects from these customer-defined JAAS login configurations might not be used by WebSphere security run time to perform authorization checks if it does not contain the required principal and credentials.

Non-prompt programmatic login:

WebSphere Application Server provides a non-prompt implementation of the javax.security.auth.callback.CallbackHandler interface, which is called `com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl`. This com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl interface allows the application to push authentication data to the WebSphere LoginModule to perform authentication. This capability can be useful for server side application code to authenticate an identity and want to use that identity to invoke downstream J2EE resources. (Some lines in the following example have been split for publication).

```
javax.security.auth.login.LoginContext lc = null;

try {
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl
("user", "securityrealm", "securedpassword"));

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication data is "push" to the authentication mechanism
//   implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception:
" + e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS login configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resources using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is an protected EJB
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception:
" + e.getMessage());
e.printStackTrace();
}
return null;
}
}
);
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

Note: You can use the com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl callback handler by a pure Java client, a client application container, enterprise bean, JSP files, servlet, or other J2EE resources.

GUI prompt programmatic login

WebSphere Application Server also provides a GUI implementation of the javax.security.auth.callback.CallbackHandler interface to collect authentication data from user through GUI login prompt, which is called `com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl`. This callback handler presents a GUI login panel to prompt users for authentication data.

```
javax.security.auth.login.LoginContext lc = null;

try {
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl());

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication date is collected by GUI login prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception:
" + e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS login configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resources using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is a protected enterprise bean
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception:
" + e.getMessage());
e.printStackTrace();
}
return null;
}
}
);
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

Note: Avoid using the com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl callback handler for server side resources (like enterprise bean, servlet, JSP file, or any other server side resources). The GUI login prompt blocks the server for user input. This behavior is not desirable for a server process.

Stdin Prompt Programmatic Login

WebSphere Application Server also provide stdin implementation of the javax.security.auth.callback.CallbackHandler interface to collect authentication data from auser through stdin, which is called `com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl`. This callback handler prompts a user in the stdin for authentication data.

```
javax.security.auth.login.LoginContext lc = null;

try {
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl());

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication date is collected by stdin prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception:
```

```
" + e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS login configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resource using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is a protected enterprise bean
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception:
" + e.getMessage());
e.printStackTrace();
}
return null;
}
}
);
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

Note: Avoid using the com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl callback handler for server side resources (like enterprise bean, servlet, JSP file, or other server side resources). The stdin prompt does not make send in the server environment, most servers run in the background do not have a console. However, if the server does have a console, the stdin prompt blocks the server for user input. This behavior is not desirable for a server process.

Java Authentication and Authorization Service custom login module

The **WSLoginHelperImpl** (**com.ibm.websphere.security.auth.WSLoginHelperImpl**) class contains a set of utility functions that bridges between the JAAS Programming model and the WebSphere Security Association Service (SAS) run time. The WSLoginModuleImpl and WSClientLoginModuleImpl methods are implemented using the WSLoginHelperImpl helper functions. In the following example, the WSLoginHelperImpl method is used as an example to illustrate how helper functions can buid a LoginModule.

WSLoginHelperImpl.login method

The login method of the WSLoginHelperImpl class can handle both BasicAuth (user ID and password) authentication and security credential (security token) validation. In the present WebSphere implementation, the login method passes a CallBack array to the specified CallbackHandler to retrieve the authentication data or security token. If user ID, password, and realm information is returned from the CallbackHandler, the login method invokes the WSLoginHelperImpl.authenticate method to authenticate the specified user. If a security token is returned from CallbackHandler, the login method invokes the WSLoginHelperImpl.validate method to validate the security token. On a successful authentication or validation, a WSCredential object is returned to the login method. The WSCredential object is saved as a private class variable credential.

WSLoginHelperImpl.commit method

The commit method of WSLoginHelperImpl class takes the WSCredential object, which is created after a successful login, and constructs the corresponding Subject object. You can use the following sample code in your own implementation. (Some lines in the following example have been split for publication.)

```
private Subject subject; // initialized by the initialize() method

String securityName = com.ibm.ws.security.common.auth.util.CredentialsHelper.getSecurityName
( com.ibm.ws.security.common.auth.util.CredentialsHelper.getPublicAttributes
( credential.getCORBACred()));
WSPrincipal principal = new com.ibm.ws.security.common.auth.WSPrincipalImpl
( securityName, credential);
try {
     java.security.AccessController.doPrivileged( new java.security.PrivilegedAction() {
 public Object run() {
                          if (!subject.getPrincipals().contains(principal)) {
  subject.getPrincipals().add(principal);
         }
         if (!subject.getPublicCredentials().contains(credential)) {
  subject.getPublicCredentials().add(credential);
  If ( !subject.getPrivateCredentials().contains( credential.getCORBACred())) {
       subject.getPrivateCredentials().add(credential.getCORBACred());
  }
  credential.setPrivateCredentialList(subject.getPrivateCredentials());
         }
         return null;
 }
     });
} catch (Exception e) {
      cleanup();
}
```

Note: Lines 13 to 15 are required because the SAS run-time code relies on finding the CORBA credential in the private credential set of the specified Subject.

If you look closer into the JAAS login configuration, you notice that all LoginModule entries are actually referring to a WSLoginModuleProxy class, which in turn delegates to the actual LoginModule implementation, namely WSLoginModuleImpl and WSClientLoginModuleImpl. WSLoginModuleImpl as well as other LoginModule implementation are under either the `lib` or the `classes` sub-directory under the WebSphere Application Server installation directory. Classes in the WebSphere Application Server `lib` and `classes` directories are not visible to the IBM Developer Kit, Java edition extension loader which loads the JAAS classes. Hence, a proxy class, WSLoginModuleProxy, is used after it is loaded by the IBM Developer Kit, Java edition extension loader, invokes the thread context class loader to load the actual LoginModule and delegate the requests.

Authentication protocol for EJB security

In WebSphere Application Server Version 5, there are two authentication protocols to choose from: Secure Association Service (SAS) and Common Secure Interoperability Version 2 (CSIv2). SAS is the authentication protocol used by all previous releases of WebSphere Application Server and thus is maintained for backwards compatibility. The Object Management Group (OMG) has defined a new authentication protocol, called CSIv2, so that vendors can interoperate securely. CSIv2 has been implemented in WebSphere Application Server Version 5 with more features than SAS and is considered the strategic protocol.

Invoking EJB methods in secure WebSphere Application Servers requires an authentication protocol to determine the level of security and type of authentication, which occurs between any given client and server for each request. It is the job of the authentication protocol during a method invocation to coalesce the server authentication requirements (determined by the object IOR) with the client authentication requirements (determined by the client configuration) and come up with an authentication policy specific to that client and server pair.

The authentication policy makes the following decisions, among others, all of which are based on both the client and server configuration.

- What kind of connection can you make to this server—SSL or TCP/IP?
- If single sockets layer (SSL) is chosen, how strong is the encryption of the data?
- If SSL is chosen, should the client be authenticated using client certificates?
- Should the client be authenticated using a user ID and password? Is there an existing credential to use?
- Should the client identity be asserted to downstream servers?
- Given the configuration of the client and server, should a secure request proceed?

You can configure both protocols (SAS and CSIv2) to work simultaneously. If a server supports both protocols, it exports an IOR containing tagged components describing the configuration for SAS and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and SAS. If the client supports both and the server supports both, CSIv2 is used. However, if the server supports SAS (for example, it is a previous WebSphere Application Server release). If the client supports both, the client chooses SAS for this request. Choose a protocol by specifying the `com.ibm.CSI.protocol` property on the client side and is configured through the administrative console on the server side (more details are included in the SAS and CSIv2 properties articles).

Common Secure Interoperability Specification, Version 2

The Common Secure Interoperability Specification, Version 2 (CSIv2) defines the Security Attribute Service (SAS) that enables interoperable authentication, delegation and privileges. The CSIv2 protocol also supports SSL and interoperability with the EJB Specification Version 2.0.

Security Attribute Service

The Security Attribute Service (SAS) protocol is designed to exchange its protocol elements in the service context of GIOP request and reply messages that are communicated over a connection-based transport. The protocol is intended to be used in environments where transport layer security, such as that available through SSL and TLS, is used to provide message protection (that is, integrity and or confidentiality) and server-to-client authentication. The protocol provides client authentication, delegation and privilege functionality that might be applied to overcome corresponding deficiencies in an underlying transport. The SAS protocol facilitates interoperability by serving as the higher-level protocol under which secure transports can be unified.

Connection and Request interceptors

The authentication protocols used by WebSphere Application Server are add-on Interoperable Inter-ORB Protocol (IIOP) services. IIOP is a request-and-reply communications protocol used to send messages between two Object Request Brokers (ORBs). For each request made by a client ORB to a server ORB, there is an associated reply made by the server ORB back to the client ORB. Prior to any request flowing, a connection between the client ORB and server ORB must be established over the TCP/IP transport (SSL is a secure version of TCP/IP). The client ORB invokes the authentication protocol client connection interceptor, which is used to read the tagged components in the IOR of the object located on the server. As mentioned previously, this is where the authentication policy is established for the request. Given the authentication policy (a coalescing of the server configuration with the client configuration), the strength of the connection is returned to the ORB. The ORB makes the appropriate connection, usually over SSL.

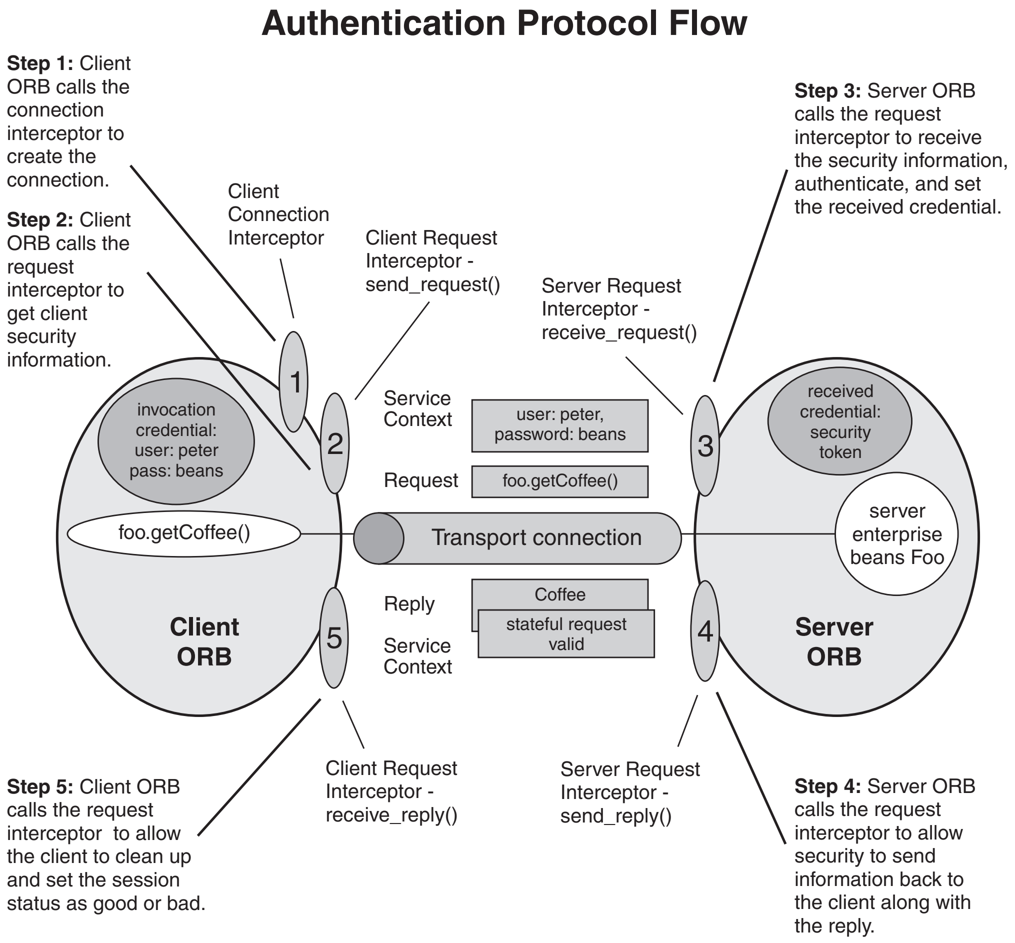
Once the connection is established, the client ORB invokes the authentication protocol client request interceptor, which is used to send security information other than what is established by the transport. This includes the user ID and password token (authenticated by the server), an authentication mechanism-specific token (validated by the server), or an identity assertion token (identity assertion is a way for one server to trust another server without the need to re-authenticate or re-validate the originating client. However, some work is required to trust the upstream server). This additional security information is sent along with the message in a service context. A service context has a registered identifier so that the server ORB can identity which protocol is sending the information. The fact that a service context contains a unique identity is another way for WebSphere Application Server to support both SAS and CSIv2 simultaneously, since both protocols have different service context IDs. Once the client request interceptor finishes adding the service context to the message, the message is sent to the server ORB.

When the message is received by the server ORB, the ORB invokes the authentication protocol server request interceptor. This interceptor looks for the service context ID known by the protocol. When both SAS and CSIv2 are supported by a server, two different server request interceptors are invoked at this point and both interceptors look for different service context IDs. However, only one finds a service context for any given request. When the server request interceptor finds a service context, it reads the information in the service context. A method is invoked to the security server to authenticate or validate client identity. The security server either rejects the information or returns a credential. A credential contains additional information about the client, retrieved from the user registry so that authorization can make the appropriate decision. Authorization is the process of determining if the user should be able to invoke the request based on the roles applied to the method and the roles given to the user. If the request is rejected by the security server, a reply is sent back to the client without ever invoking the business method.

If no service context is found by the CSIv2 server request interceptor, it then looks at the transport connection to see if a client certificate chain was sent. This is done when SSL client authentication is configured between the client and server. If a client certificate chain is found, the distinguished name (DN) is extracted from the certificate and is used to map to an identity in the user registry. If the user registry is LDAP, the search filters defined in the LDAP registry configuration determine how the certificate maps to an entry in the registry. If the user registry is LocalOS, the first attribute of the DN maps to the user ID of the registry. This attribute is typically the common name. If the certificate does not map, no credential is created and the request is rejected. When invalid security information is presented, the method request is rejected and a NO_PERMISSION exception is sent back with the reply. However, when no security information is presented, an unauthenticated credential is created for the request and the authorization engine determines if the method gets invoked or not. For an unauthenticated credential to invoke an EJB method, either no security roles are defined for the method or a special **Everyone** role is defined for the method.

When the method invocation is completed in the EJB container, the server request interceptor is again invoked to complete server authentication and a new reply service context can be created to inform the client request interceptor of the outcome. This process is typically for making the request *stateful*. When a stateful request is made, only the first request between a client and server requires security information to be sent. All subsequent method requests need only send a unique context ID so that the server may look up the credential stored in a session table. The context ID is unique within the connection between a client and server.

Finally, the method request cycle is completed by the client request interceptor receiving a reply from the server with a reply service context providing information so the client side stateful context ID can be confirmed and reused. Specifying a stateful client is done through the property `com.ibm.CSI.performStateful (true/false)`. Specifying a stateful server is done through the administrative console configuration.


Authentication Protocol Flow
Step 1: Client ORB calls the connection interceptor to create the connection.
Step 2: Client ORB calls the request interceptor to get client security information.
Client Connection Interceptor
Client Request Interceptor - send_request()
Server Request Interceptor - receive_request()
Step 3: Server ORB calls the request interceptor to receive the security information, authenticate, and set the received credential.
invocation credential: user: peter pass: beans
1
2
3
Service Context
user: peter, password: beans
Request
foo.getCoffee()
received credential: security token
foo.getCoffee()
Transport connection
server enterprise beans Foo
Reply
Coffee
Service Context
stateful request valid
Client ORB
5
4
Server ORB
Step 5: Client ORB calls the request interceptor to allow the client to clean up and set the session status as good or bad.
Client Request Interceptor - receive_reply()
Server Request Interceptor - send_reply()
Step 4: Server ORB calls the request interceptor to allow security to send information back to the client along with the reply.

Authentication Policy for Each Request

The authentication policy of a given request determines what is performed in terms of security protection between a client and a server. A client and or server authentication protocol configuration can describe required features, supported features and non-supported features. When a client requires a feature, it can only talk to servers that either require or support that feature. When a server requires a feature, it can only talk to clients that either require or support that feature. When a client supports a feature, it may talk to a server that supports or requires that feature, but can also talk to servers that do not support the feature. When a server supports a feature, it can talk to a client that supports or requires the feature, but can also talk to clients that do not support the feature (or chose not to support the feature).

For example, for a client to support client certificate authentication, some setup is required to either generate a self-signed certificate or get one from a certificate authority (CA). Some clients may not want the trouble of doing these actions, therefore, they can configure this feature as *not supported*. By making this decision, it cannot communicate with a secure server requiring client certificate authentication. Instead, this client may choose to use the user ID and password as the method of authenticating itself to the server.

Typically, supporting a feature is the most common way of configuring features. It is also the most successful during run time since it is more forgiving than requiring a feature. Knowing how secure servers are configured in your domain, you can choose the right combination for the client to ensure successful method invocations and still get the most security. If you know that all of your servers support both client certificate and user ID and password authentication for the client, you might want to require one and not support the other. If both the user ID and password and the client certificate are supported on the client and server, both are performed but user ID and password take precedence at the server. This action is based on the CSIv2 specification requirements.

CSIv2 features

The following CSIv2 features are available in IBM WebSphere Application Server: SSL client certificate authentication, message layer authentication and identity assertion.

- Secure Socket Layer Client Certificate authentication (Transport Layer Authentication) .

  An additional way to authenticate a client to a server using SSL client authentication.
- Message layer authentication.

  Authenticates credential information and sends that information across the network so that a receiving server can interpret it.
- Identity assertion.

  Allows a downstream server to accept the client identity established on an upstream server, without having to reauthenticate. The downstream server trusts the upstream server.

Identity assertion

Identity assertion is the invocation credential that is asserted to the downstream server.

When a client authenticates to a server, the received credential is set. When authorization checks the credential to determine whether access is permitted, it will also set the *invocation* credential so that if the EJB method calls another EJB method located on other servers, the invocation credential can be used as the identity to invoke the downstream method. Depending on the RunAs mode for the enterprise beans, the invocation credential is set as the originating client identity, the server identity, or a specified different identity. Regardless of the identity that is set, when Identity Assertion is enabled, it is the invocation credential that is asserted to the downstream server.

The invocation credential identity is sent to the downstream server in an identity token. In addition, the sending server identity, including password or token, is sent in the client authentication token. Both are needed by the receiving server to accept the asserted identity. The receiving server completes the following to accept the asserted identity.

First, the server determines whether the sending server identity is on the trusted principal list of the receiving server. In other words, can the sending server send an identity token to the receiving server. Once you determine that the sending server is on the trusted list, ensure that the server is the sending server by authenticating it. Authenticate the server by comparing the user ID and password from the sending server to that of the receiving server, or it could require a real authenticate call. If the credentials of the sending server are authenticated and on the trusted principal list, then proceed to evaluate the identity token. Evaluation of the identity token consists of the following.

There are four formats of identities which can be present in an identity token—a principal name, a distinguished name, a certificate chain and an anonymous identity. The product server that receive authentication information typically support all four identity types. The sending server decides which one is chosen based on how the original client authenticated. The type that is present depends on how the client originally authenticates to the sending server. For example, if the client uses SSL client authentication to authenticate to the sending server, then the identity token to the downstream server contains the certificate chain. This information is important because it permits the receiving server to perform its own certificate chain mapping. It enables more interoperability with other vendors and platforms.

Once the identity format is understood and parsed, the identity maps to a credential. For an `ITTPrincipal` identity token, this maps one-to-one with the user ID fields. For an `ITTDistinguishedName` identity token, the mapping depends on the user registry. For LDAP, the configured search filter determines how the mapping occurs. For LocalOS, the first attribute of the DN (typically the same as the common name) maps to the user ID of the registry. For an `ITTCertChain` identity token, see the section, (Map certificates to users) for details on how this action is performed for the LDAP user registry. For LocalOS, the first attribute of the DN in the certificate is used to map to the user ID in the registry.

Some user registry methods are called to gather additional credential information used by authorization. In a stateful server, this is done one time for the sending server and receiving server pair where the identity tokens are the same. Subsequent requests are made through a session ID.

Identity Assertion is only available using the CSIv2 protocol.

Message layer authentication

Defines the credential information and sends that information across the network so that a receiving server can interpret it.

Defines the credential information and sends that information across the network so that a receiving server can interpret it.

When sending authentication information across the network using a token (whether the token be a user ID and password token , i.e., GSSUP, or a mechanism-specific format token, LTPA, for example, this is considered message layer authentication because the data is sent along with the message inside a service context.

A pure Java client will use BasicAuth (GSSUP) as the authentication mechanism to establish the client's identity. However, a servlet can use either BasicAuth (GSSUP) or the authentication mechanism of the server (LTPA) to send security information in the message layer. In order to use LTPA the BasicAuth credentials need to be authenticated or mapped to the security mechanism of the server.

The security token contained in a token-based credential is authentication mechanism specific. That is, the way the token is interpreted is only known by the authentication mechanism. Therefore, each authentication mechanism has an object ID (OID) representing it and sent along with the token from the client to the server, so that the server knows which mechanism needs to read the token and validate it. The following lists the object IDs for each mechanism:

```
BasicAuth (GSSUP):  oid:2.23.130.1.1.1
LTPA:    oid:1.3.18.0.2.30.2
SWAM:     No OID since it is not forwardable
```

On the server, the authentication mechanisms can interpret the token and create a credential, or they can authenticate BasicAuth data coming over from the client, and create a credential. Either way, the credential created will be the "received" credential to be used by the authorization check to determine if the user has access to invoke the method. The authentication mechanism can be specified using the com.ibm.CORBA.authenticationTarget property on the client side (the only valid value currently is BasicAuth). The server is configured through the GUI.

While this property tells you which authentication mechanism to use, you also need to specify whether you want to perform authentication over the message layer (i.e., get a `BasicAuth or token-based` credential). To do this you can specify the `com.ibm.CSI.performClientAuthenticationRequired`(**True** or **False**) and `com.ibm.CSI.performClientAuthenticationSupported` (**True** or **False**) properties. To say that client authentication is required indicates that it must be done for every request. To say that it is supported indicates it may be done but does not need to be done. For some servers, this is good if no resources are protected. In most cases it's good to say that this is supported. This way it will be performed if both client and server support it and it won't be performed when communicating with certain servers that don't want security but yet the method request will still succeed.

Configuring Authentication Retries: There are situations where you want a prompt to reappear if you have entered your user ID and password incorrectly or you want a method to retry when a particular error occurs back at the client. If the error is one which can be corrected by information at the client side, we will automatically perform a retry without the client seeing the failure if it is configured to do so.

Some of these errors include entering an invalid user ID and password, having an expired credential on the server, not finding the stateful session on the server, etc. By default, authentication retries are enabled and it will perform 3 retries before finally returning the error to the client. The property to enable/disable authentication retries is com.ibm.CORBA.authenticationRetryEnabled (True/False). The property to specify the number of retry attempts is com.ibm.CORBA.authenticationRetryCount.

Immediate Validating of a BasicAuth Login: In WebSphere Application Server Version 5.0, a new behavior is defined during request_login for a BasicAuth login. In prior releases, a BasicAuth login would take the user ID and password entered via the loginSource method and create a BasicAuth credential. If the user ID or password was invalid, the client program would not find out until the first method request was attempted. Now, when the user ID or password is specified during a prompt or programmatic login, by default this will be authenticated with the security server with a **True** or **False** being returned as the result. If **False**, n org.omg.SecurityLevel2.LoginFailed exception will be returned to the client indicating the user ID and password are invalid. If **True**, then the BasicAuth credential will be returned to the caller of request_login. To disable this feature on the pure client, specify com.ibm.CORBA.validateBasicAuth=false. By default this is set to **True**. On the server side, specify this property in the security dynamic properties.

Secure Socket Layer Client Certificate authentication (Transport Layer Authentication)

An additional way to authenticate a client to a server is using SSL client authentication.

Using SSL client authentication is another way of authenticating a client to a server. This form of authentication does not occur at the message layer as described above (that is., using user ID and password or tokens), instead it occurs during the connection handshake using SSL certificates. When the client is configured with a personal certificate in the SSL keystore and it indicates that SSL client authentication is desired and the server supports SSL client authentication, then the following happens to establish the identity on the client side.

When a method request is invoked in the client code to a remote enterprise bean, the ORB invokes the client connection interceptor to establish a connection with the server. Since the configuration specifies SSL, and in addition SSL client authentication, the connection type is SSL and the SSL handshake sends the client's certificate to the server to validate. If the client certificate does not validate, the connection does not get established and an exception will get sent back to the client code where the method gets invoked, indicating the failure. If the client certificate does get validated, then a connection gets opened between the client and the server.

The ORB proceeds to call the client request interceptor, which may or may not have work to do. If, for example, BasicAuth is also configured, then the user may be prompted for a user ID and password. Obviously this is not necessary, so it should be disabled in the configuration if the SSL certificate is the desired identity to invoke the method against. If there is not message layer security, then no security context will be created and associated with the request.

Once the server receives the request, the server side request interceptor will first check for a security context. Since the server will not find a service context, it will then check the server socket for a client certificate chain which contains the client identity. In this case, it will find the certificate chain from the client. At this point, the identity in the certificate chain is valid since the connection was made. All that needs to be done to create a credential, is to map the identity from the certificate to the user registry. This is done differently based on the type of authentication mechanism. Mapping a certificate to a credential is done differently based on the User Registry type. See the article, (Map certificates to users) for details on how this is performed for the LDAP user registry. For LocalOS, the first attribute of the DN in the certificate is used to map to the user ID in the registry.

One benefit of SSL client certificate authentication is that it's probably the best performing way to authenticate since an SSL connection is typically created anyway. The extra overhead of sending the client's certificate is minimal. The client side request interceptor really does nothing and the server side request interceptor just maps the certificate to a credential. The drawback might be the complexity of setting up the keystore on each client system.

To enable SSL client certificate authentication on the client side, you must enable the following properties. First you must enable SSL. This is done using the following two properties com.ibm.CSI.performTransportAssocSSLTLSRequired (true or false) and com.ibm.CSI.performTransportAssocSSLTLSSupported (true or false). To say that SSL is required indicates that every request must generate an SSL connection key. If a server does not support SSL, then the request will fail. One you have enabled SSL by either supporting it or requiring it, you can enable some of the SSL features.

To enable SSL client authentication, you can specify the following two properties com.ibm.CSI.performTLClientAuthenticationRequired (true or false) and com.ibm.CSI.performTLClientAuthenticationSupported (true/false). The TL stands for transport layer. Again, if you say that SSL client authentication is required, then you limit only being able to communicate with servers which support SSL client authentication. For a server to support SSL client authentication it must have configured similar properties through the GUI and thus an SSL listener port will be opened which can handle mutual authentication handshakes. Configuration of server properties are always done through the GUI.

SSL client certificate authentication from a Java client is only available using the CSIv2 protocol.

Supported IBM Protocols: Secure Association Service and Common Secure Interoperability Version 2

There are two authentication protocols supported by IBM. Secure Association Service (SAS) is the authentication protocol used by all previous releases of the WebSphere product. Common Secure Interoperability Version 2 (CSIv2) is implemented in WebSphere Application ServerVersion 5.0 and is considered the strategic protocol.

You can configure both protocols to work simultaneously. If a server supports both protocols, it exports an IOR containing tagged components describing the configuration for SAS and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and SAS. If the client supports both and the server supports both, CSIv2 is used. However, if the server supports SAS (for example, is a previous WebSphere release) and the client supports both, the client chooses SAS for this request. Choosing a protocol is specified using the com.ibm.CSI.protocol property on the client side and configured through the GUI on the server side.

Configuring CSIv2 and SAS authentication protocols

Steps for this task

1. Determine how security should be configured inbound and outbound at each point in your infrastructure.

   For example, you might have a Java client communicating with an EJB application server, which in turn communicates to a downstream EJB application server. The Java client will utilize the sas.client.props file to configure outbound security (pure clients only need to configure outbound security). The upstream EJB application server will configure inbound security to handle the right type of authentication from the Java client. The upstream EJB application server will utilize the outbound security configuration when going to the downstream EJB application server. This type of authentication might be different than what you would expect from the Java client into the upstream EJB application server. Security might be *tighter* between the pure client and the first EJB server, depending on your infrastructure. The downstream EJB server will utilize the inbound security configuration to accept requests from the upstream EJB server. These two have to have similiar configuration options as well. If the downstream EJB application server communicates to other downstream servers, then the outbound security might need to be configured a special way.
2. Specify the type of authentication.

   By default, authentication using Userid/Password is performed. Both Java client certificate authentication and Identity Assertion are disabled by default. Therefore, if you want this type of *Basic Authentication* to be performed at every tier, than you probably do not need to modify the CSIv2 authentication protocol configuration. However, if you have any special requirements where some servers authenticate differently from other servers, than you will want to understand how to configure CSIv2 to best take advantage of the features it offers.
3. Configure clients and servers.

   Configuring a pure Java client is always done through the `sas.client.props` file where properties are modified. Configuring servers will always be done from the WebSphere Administrative Console either from Security Center for Cell level configurations or from the application server Server Security for server level configurations. If you want some servers to authenticate differently from others, you would probably need to modify some of the server level configurations. Anytime you modify the server level configurations, you are overriding the cell level configurations.

Configuring CSIv2 inbound authentication

Inbound authentication refers to the configuration which determines what type of authentication is accepted for inbound requests. This is advertised in the IOR that the client will pickup from the name server.

Steps for this task

1. Start the administrative console. Navigate to **Security** > **Authentication Protocol** > **CSI Authentication** > **Inbound**.
2. Consider the following three layers of security:
   - **Identity Assertion** (attribute layer). When selected, this server will accept identity tokens from upstream servers. If it does receive an identity token, the identity will be from an originating client (for example, the identity in the same form as the originating client presented it to the first server). An upstream server sends the identity of the originating client. The format of the identity can be either a principal name, distinguished name, or certificate chain. In some cases, the identity is anonymous. It is important to trust the upstream server that sends the identity token since the identity is not going

to be authenticated on this server. The server ID is sent in the client authentication token along with the identity token and the server ID is checked against the trusted server ID list. If the server ID is on the trusted server list, the server ID is authenticated. If the server ID is valid, then the identity token identity is put into a credential and used for authorization of the request.

- **User ID/Password** (message layer). This type of authentication is the most typical. The user ID/password OR authenticated token is sent either from a pure client or from an upstream server. Usually, a token is sent from an upstream server and a user ID/password is sent from a client (including a servlet). When a user ID/password is received at the server, it is authenticated with the user registry. When a token is received at the server, it is validated (to check if the token is expired or has been tampered with).
- **SSL Client certificate authentication** (transport layer). This type of authentication typically occurs from pure clients for the purpose of using the certificate identity and from servers for the purpose of trusting the upstream server. Usually, when a server delegates an identity to a downstream server, this will come from either the message layer (a client authentication token) or the attribute layer (an identity token), not from the transport layer through the client certificate authentication. A client has an SSL client certificate stored in the keystore in the client configuration. When SSL client authentication is enabled on this server, the server will request the client to send it when the connection is established. The certificate chain is available on the socket whenever a request is sent to the server. The server request interceptor will get the certificate chain from the socket and map it to a user in the registry. This type of authentication is good directly from a client to a server, however, when you have to go downstream, the identity typically flows over the Message Layer or via Identity Assertion.

3. Consider the following when deciding what type of authentication to accept:
   - A server can receive multiple layers simultaneously, therefore, an order of precedence rule decides which identity to use. The identity assertion layer has top priority, then the message layer and finally the transport layer. SSL client certificates will only be used if that is the only layer provided. If the message layer and transport layer are provided, the message layer will be used to establish the identity used for authorization. If the identity assertion layer is provided, that will always be used to establish precedence.
   - Will this server usually receive requests from a client or a server or both? If always receiving requests from a client, this rules out the need for identity assertion. You can then choose either the message layer or transport layer or both. Also, you can decide when one of them is required or just supported. To select a layer as required, the sending client must supply this layer or the request will be rejected. However, if the layer is only supported, the layer may or may not be supplied.
   - What kind of client identity has been supplied? If the client identity is client certificates and you want the certificate chain to flow downstream so that it may be mapped to the downstream servers user registries, then identity assertion is the appropriate choice. Identity Assertion will preserve the format of the originating client. If the originating client authenticated with user ID and password, then a principal identity will be sent. If it is with a certificate, then the certificate chain will be sent. In some cases, if the client authenticated with a token and an LDAP server is the user registry, then a distinguished name will be sent.
4. Configure a trusted server list.

When identity assertion is selected for inbound requests, you need to type a comma separated list of server administrator ids which this server will be allowing to submit identity tokens to. If you choose to allow any server to send an identity token, you can put just an asterisk in this field. This is called presumed trust. In this case, you should use SSL client certificate authentication between servers to establish the trust.

5. Configure session management.

   You can choose either stateful or stateless security. Performance is best when choosing stateful sessions. The first method request between a client and server is authenticated. All subsequent requests (or until the credential token expires) will reuse the session information including the credential. A client will send just a context ID for subsequent requests. The context ID is scoped to the connection for uniqueness.

Results

When you have completed configuring this panel, you will have configured most of the information which a client will coalesce when determining what to send to this server. A client or server outbound configuration along with this server inbound configuration, will determine the security which will be applied. When you always know what clients will be sending, the configuration is simple. However, if you have a disparate set of clients with differing security requirements, your server will need to consider various layers of authentication.

Usage scenario

For an enterprise beans server, the authentication will usually be either Identity Assertion or Message Layer. This is because you want to have the identity of the originating client be delegated downstream. You cannot easily delegate a client certificate using an SSL connection. It's ok to enable the transport layer for additional server security as the additional client certificate portion of the SSL handshake adds just a little overhead to the overall SSL connection establishment.

What to do next

Once you determine what type of authentication data this server might received, you can then determine what to select for outbound security. Proceed to Configuring CSIv2 Outbound Authentication.

Identity Assertion: The invocation credential that is asserted to the downstream server.

When a client authenticates to a server, the received credential is set. When authorization checks the credential to see if it is allowed access, it will also set the *invocation* credential so that if the EJB method calls another EJB method located on other servers, the invocation credential can be used as the identity to invoke the downstream method. Depending on the RunAs mode for the enterprise beans, the invocation credential will be set as the originating client identity, the server identity, or a specified different identity. Regardless of the identity that is set, when Identity Assertion is enabled, it is the invocation credential that is asserted to the downstream server.

The invocation credential identity is sent to the downstream server in an identity token. In addition, the sending server identity including password or token, is sent in the client authentication token. Both are needed by the receiving server to accept the asserted identity. The receiving server does the following to accept the asserted identity.

First it is determined whether the sending server identity is on the trusted principal list of the receiving server. In other words, is the sending server one allowed to send an identity token to the receiving server.

Second, once we have determined that the sending server is on the trusted list, we need to make sure it truly is the sending server by authenticating it. This could be simply comparing the user ID and password from the sending server to that of the receiving server. Or it could require a real authenticate call. If the sending server credentials are authenticated and on the trusted principal list, then evaluation of the identity token can proceed. Evaluation of the identity token consists of the following.

There are four formats of identities which can be present in an identity token. A principal name, a distinguished name, a certificate chain, and an anonymous identity. The product servers that receive authentication information typically support all four identity types. The sending server decides which one will be chosen based on how the original client authenticated. The type that is present depends on how the client originally authenticates to the sending server. For example, if the client uses SSL client authentication to authenticate to the sending server, then the identity token to the downstream server will contain the certificate chain. This is important because it allows the receiving server to perform its own mapping of the certificate chain. It enables more interoperability with other vendors and platforms.

Once the identity format is understood and parsed, the identity is simply mapped to a credential. All identity token types map to the user ID field of the active user registry. For an `ITTPrincipal` identity token, this maps one-to-one with the user ID fields. For an `ITTDistinguishedName` identity token, the value is taken from the first equal sign and mapped to the user ID field. For an `ITTCertChain identity` token, the value from the first equal sign of the Distinguished Name is taken from the certificate and mapped to the user ID field. Future enhancements may include mapping Distinguished Name to Distinguished Name and using filters to allow administrators to control the mapping.

Some user registry methods are called to gather additional credential information used by authorization. In a stateful server, this is done one time for the sending server or receiving server pair where the identity tokens are the same. Subsequent requests will be made via a session ID.

Identity Assertion is only available using the CSIv2 protocol.

Common Secure Interoperability inbound authentication settings: Use this page to specify the features that a server supports for a client needing to access its resources.

To view this administrative console page, click **Security** > **Authentication Protocol** > **CSI Authentication** > **Inbound**.

CSI inbound authentication settings are for configuring the type of authentication information contained in an incoming request or transport.

Authentication features include three layers of authentication, which you can use simultaneously:

- **Transport layer**. The transport layer, the lowest layer, might contain a Secure Socket Layer (SSL) client certificate as the identity.
- **Message layer**. The message layer might contain a user ID and password or expirable authenticated token.
- **Attribute layer**. The attribute layer might contain an identity token which is an identity from an upstream server and is already authenticated. The identity layer has the highest priority followed by the message layer and then the transport layer. If a client sends all three, only the identity layer is used. The only way to use the SSL client certificate as the identity is if it is the only information presented during the request. The client picks up the IOR from the name space and reads the values from the tagged component to determine what the server needs for security.

Basic Authentication: Specifies that basic authentication occurs over the message layer.

In the message layer, Basic Authentication (user ID and password) takes place. This type of authentication typically involves sending a user ID and password from the client to the server for authentication. This also involves delegating a credential token from an already authenticated credential provided the credential type is forwardable (for example, Lightweight Third Party Authentication (LTPA)). If **Basic Authentication** is selected for the server, you specify both user ID and password authentication as well as token-based authentication.

When selecting **Basic Authentication**, you need to decide whether it is **Required** or **Supported**. Selecting **Required**, indicates only clients which are configured to authenticate to this server through the message layer are allowed to invoke requests on the server. Selecting supported, indicates that this server accepts **Basic Authentication**. However, other methods of authentication can occur if configured and anonymous requests are accepted. Selecting **Never**, indicates that the server is not configured to accept message layer authentication from any client.

Data type	String

Client Certificate Authentication: Specifies that authentication occurs when the initial connection is made between the client and server during a method request.

In the transport layer, Secure Socket Layer (SSL) client certificate authentication takes place. In the message layer, Basic Authentication (user ID and password) is performed. Client certificate authentication typically performs better than message layer authentication but requires some additional setup steps. These additional steps involve ensuring the server has the signer certificate of each client to which it is connected. If the client uses a certificate authority (CA) to create its personal certificate, then you need only the CA root certificate in the server signer section of the SSL trust file. When the certificate is authenticated to an LDAP user registry, the DN is mapped based on the filter specified when configuring LDAP. When the certificate is authenticated to a LocalOS user registry, the first attribute of the DN in the certificate (typically the CN) is mapped to the user ID in the registry. The identity from client certificates is used only if no other layer of authentication is presented to the server.

When selecting **Client Certificate Authentication**, you need to decide whether it is **Required** or **Supported**. When selecting **Required**, only clients which are configured to authenticate to this server through SSL client certificates are allowed to invoke requests on the server. When selecting **Supported**, this server accepts SSL client certificate authentication, however, other methods of authentication can occur (if configured) and anonymous requests are accepted. When selecting **Never**, this server is not configured to accept client certificate authentication from any client.

Data type	String

Identity Assertion: Specifies that identity assertion is a way to assert identities from one server to another during a downstream EJB invocation.

Identity assertion is performed in the attribute layer and is only applicable on servers. The principal determined at the server is based on precedence rules. If identity assertion is performed, the identity is always derived from this layer. If basic authentication is performed without identity assertion, the identity is always derived from this layer. Finally, if SSL client certificate authentication is performed without either basic authentication, or identity assertion, then the identity is derived from this layer.

The identity asserted is the invocation credential which is determined by the RunAs mode for the enterprise bean. If the RunAs mode is **Client**, the identity is the client identity. If the RunAs mode is **System**, the identity is the server identity. If the RunAs mode is **Specified**, the identity is the one specified. The receiving server receives the identity in an identity token and also receives the sending server identity in a client authentication token. The receiving server validates the sending server identity is as a trusted identity through the **Trusted Server IDs** entry box. Enter a list of comma-separated principal names, for example, `serverid1, serverid2, serverid3`.

When authenticating to a LocalOS user registry, all identity token types map to the user ID field of the active user registry. For an `ITTPrincipal` identity token, this maps one-to-one with the user ID fields. For an `ITTDistinguishedName` identity token, the value from the first equal sign is mapped to the user ID field. For an `ITTCertChain` identity token, the value from the first equal sign of the distinguished name is mapped to the user ID field.

When authenticating to an LDAP user registry, the LDAP filters determine how an identity of type `ITTCertChain` and `ITTDistinguishedName` gets mapped to the registry. If the token type is `ITTPrincipal`, then the principal gets mapped to the UID field in the LDAP registry.

Data type	String

Trusted Server User IDs: Specifies a comma-separated list of server user IDs, which are trusted to perform identity assertion to this server.

Use this list to quickly rule out whether or not a server is trusted. Even if the server is on the list, the sending server must still authenticate with the receiving server to accept the identity token of the sending server.

Data type	String

Stateful Sessions: Specifies stateful sessions, used mostly for performance improvements.

The first contact between a client and server must fully authenticate. However, all subsequent contacts, while the sessions are still valid, reuse the security information. The client passes a context ID to the server, and the ID is used to look up the session. The context ID is scoped to the connection, which guarantees uniqueness. Whenever the security session is invalid, if the authentication retry is enabled (it is by default), the client-side security interceptor invalidate the client-side session and resubmits the request without the user awareness. This might occur if the session does not exist on the server (the server failed and resumed operation). When this value is disabled, every method invocation must re-authenticate.

Data type	String

Configuring CSIv2 outbound authentication

Outbound authentication refers to the configuration which determines what type of authentication is performed for outbound requests to downstream servers. There are several layers/methods of authentication that can occur. The downstream server inbound authentication configuration must support at least one choice made in this server outbound authentication configuration. If nothing is supported, the request may go outbound as `unauthenticated`. This does not create a security problem as the authorization runtime is responsible for preventing access to resources which are protected. However, if you choose to prevent an unauthenticated credential to go outbound, you might want to choose one of the authentication layers to be `required` rather than just `supported`. When something is required, if the downstream server does not support it, the method request will fail to go outbound.

The choices in the CSIv2 outbound authentication panel is as follows:

Steps for this task

1. Identity Assertion (attribute layer). When selected, this server will submit an identity token to a downstream server if the downstream server supports Identity Assertion. When an originating client authenticated to this server, the authentication information supplied is preserved in the outbound identity token. That is, if the client authenticated to this server using client certificate authentication, the identity token format will be a certificate chain containing the exact client certificate chain on the socket. The same is true for other mechanisms of authentication. Click on this link to read more about (Identity Assertion).
2. User ID and Password (message layer). This type of authentication is the most typical. The user and password (if BasicAuth credential) or authenticated token (if authenticated credential) is sent outbound to the downstream server if the downstream server supports message layer authentication in the inbound authentication panel. Refer to the article,(Message Layer Authentication).
3. SSL Client certificate authentication (transport layer). The main reason to enable outbound SSL client authentication from one server to a downstream server is to create a trusted environment between those servers. For delegating client credentials you should use one of the two layers above. However, you might want to create SSL personal certificates for all servers in your domain and only trust those servers in your SSL truststore. This way no other servers or clients could even make a connection to the servers in your domain, except for just at the tiers where you want them to. This is a good way to protect your enterprise

beans servers from being accessed by anything but your servlet servers. Refer to the article,(SSL Client Certificate Authentication).

A server can send multiple layers simultaneously, therefore, an order of precedence rule decides which identity to use. The identity assertion layer has top priority, then the message layer and finally the transport layer. SSL client certificates will only be used as the identity for invoking method requests if that is the only layer provided (although it is useful for trust purposes even if the identity is not used for the request). If only the message layer and transport layer are provided, the message layer will be used to establish the identity used for authorization. If the identity assertion layer is provided (regardless of whatever else is provided), then the identity from the identity token will always be used by the authorization engine as the identity for that request

Configuring session management: You can choose either stateful or stateless security. Performance is best when choosing stateful sessions. The first method request between this server and the downstream server is authenticated. All subsequent requests (or until the credential token expires) will reuse the session information including the credential. A unique session entry is defined as a unique client auth token and identity token combined, scoped to the connection.

Results

When you have completed configuring this panel, you will have configured the information which this server will use to make decisions about the type of authentication to perform with downstream servers. If the downstream server is configured in such a way as to not support this server's outbound configuration, the exception that will likely occur is the following: (Some lines in the following example have been split for publication.)

```
Exception received: org.omg.CORBA.INITIALIZE:
JSAS1477W: SECURITY CLIENT/SERVER CONFIG MISMATCH:  The client security configuration
(sas.client.props or outbound settings in GUI) does not support the server security
configuration for the following reasons:
ERROR 1: JSAS0607E: The client requires SSL Confidentiality but the server does not support it.
ERROR 2: JSAS0610E: The server requires SSL Integrity but the client does not support it.
ERROR 3: JSAS0612E: The client requires client (e.g., userid/password or token),
but the server does not support it.
minor code: 0  completed: No
        at com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor.
   getConnectionKey (SecurityConnectionInterceptor.java:1770)
        at com.ibm.ws.orbimpl.transport.WSTransport.getConnection(Unknown Source)
        at com.ibm.rmi.iiop.TransportManager.get(TransportManager.java:79)
        at com.ibm.rmi.iiop.GIOPImpl.locate(GIOPImpl.java:167)
        at com.ibm.CORBA.iiop.ClientDelegate._createRequest(ClientDelegate.java:2088)
        at com.ibm.CORBA.iiop.ClientDelegate.createRequest(ClientDelegate.java:1264)
        at com.ibm.CORBA.iiop.ClientDelegate.createRequest(ClientDelegate.java:1177)
        at com.ibm.CORBA.iiop.ClientDelegate.request(ClientDelegate.java:1726)
        at org.omg.CORBA.portable.ObjectImpl._request(ObjectImpl.java:245)
        at com.ibm.WsnOptimizedNaming._NamingContextStub.get_compatibility_level
   (Unknown Source)
        at com.ibm.websphere.naming.DumpNameSpace.getIdlLevel(DumpNameSpace.java:300)
        at com.ibm.websphere.naming.DumpNameSpace.getStartingContext(DumpNameSpace.java:329)
        at com.ibm.websphere.naming.DumpNameSpace.main(DumpNameSpace.java:268)
        at java.lang.reflect.Method.invoke(Native Method)
        at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:163)
```

Note: The reasons for the mismatch will be spelled out in the exception and the corrections can be made at either this server's outbound configuration or the downstream server's inbound configuration. If there are multiple reasons for a failure, they should all be spelled out at once as message text in the exception.

Usage scenario

Typically, the outbound authentication configuration is for an upstream server to communicate with a downstream server. Most likely, the upstream server is a servlet server and the downstream server is an EJB server. On a servlet server, the authentication performed by the client to access the servlet can be one of many different types of authentication including client certificate and basic auth. When receiving basic auth data, whether that be via a prompt login or a form based login, the basic auth information is typically authenticated to form a credential of the mechanism type supported by the server such as LTPA or LocalOS. When LTPA is the mechanism, a forwardable token exists in the credential in which case you would want to choose the message layer (BasicAuth) authentication to propagate the client's credentials. If the credential was created using a certificate login and you want to preserve sending the certificate downstream, you might want to go outbound with Identity Assertion.

What to do next

For the changes to take effect, restart the server after having saved the configuration.

Common Secure Interoperability outbound authentication settings: Use this page to specify features that a server will support when acting as a client to another downstream server.

To view this administrative console page, click **Security** > **Authentication Protocol** > **CSI Authentication** > **Outbound**.

Authentication features include three layers of authentication, which could be used simultaneously:

- **Transport layer**. The transport layer, the lowest layer, might contain an SSL client certificate as the identity.
- **Message layer**. The message layer might contain a user ID and password or authenticated token.
- **Attribute layer**. The attribute layer might contain an identity token which is an identity from an upstream server and is already authenticated. The identity layer has the highest priority followed by the message layer and then the transport layer. If this server sends all three, only the identity layer is used by the downstream server. The only way to use the SSL client certificate as the identity is if it is the only information presented during the outbound request.

Basic Authentication: Specifies whether to send a user ID and password from the client to the server for authentication.

This type of authentication occurs over the message layer. Basic Authentication also involves delegating a credential token from an already authenticated credential provided the credential type is forwardable (for example, Lightweight Third Party Authentication (LTPA)). Basic authentication refers to any authentication over the message layer and indicates both user ID and password and token-based authentication.

Selecting Basic Authentication determines whether it is required or supported. Selecting Required indicates that when the server goes outbound to downstream servers, the downstream server must support Basic Authentication for this server to connect to it. Selecting supported indicates that this server may or may not perform Basic Authentication to a downstream server, however, other methods of authentication can occur (if configured). Selecting never, indicates that this server will never send a message layer token outbound to a downstream server. If the downstream server requires Basic Authentication then the connection is not attempted.

Data type	String

Client Certificate Authentication: Specifies whether a client certificate from the configured keystore will be used to authenticate to the server when the SSL connection is made between this server and a downstream server (provided that the downstream server supports client certificate authentication).

Typically client certificate authentication performs better than message layer authentication but requires some additional setup steps. These additional steps involve ensuring this server has a personal certificate and the downstream server has the signer certificate of this server.

When selecting Client Certificate Authentication, decide whether it is a required or supported. Selecting required indicates that this server can only connect to downstream servers with Client Certificate Authentication also configured. Selecting supported indicates that this server will perform Client Certificate Authentication with any downstream server but may or may not use Client Certificate Authentication depending on whether the downstream server supports it. Selecting never indicates that this client will not perform Client Certificate Authentication to any downstream server. This limitation prevents access to any downstream server that requires Client Certificate Authentication.

Data type	String

Identity Assertion: Specifies whether to assert identities from one server to another during a downstream enterprise bean invocation.

The identity asserted is the invocation credential which is determined by the RunAs mode for the enterprise bean. If the RunAs mode is **Client**, the identity is the client identity. If the RunAs mode is **System**, the identity is the server identity. If the RunAs mode is **Specified**, the identity is the identity specified. The receiving server receives the identity in an identity token and also receives the sending server identity in a client authentication token. The receiving server validates the identity of the sending server to ensure a trusted identity.

Data type	String

Stateful Sessions: Specifies whether to reuse security information during authentication. This option is usually used to increase performance.

The first contact between a client and server must fully authenticate. However, all subsequent contacts, while the sessions are still valid, reuse the security information. The client passes a context ID to the server, and that ID is used to look up the session. The context ID is scoped to the connection, which guarantees uniqueness. Whenever the security session is invalid, if authentication retry is enabled (it is by default), the client-side security interceptor invalidates the client-side session and resubmits the request without you being aware of it. For example the session does not exist on the server (the server failed and resumed operation).

When this value is disabled, each and every method invocation must re-authenticate.

Data type	String

Configuring inbound transports

Inbound transports refers to the types of listener ports and their attributes that will be opened to receive requests for this server. Both CSIv2 and SAS have the ability to configure the transport to some degree. CSIv2 is much more flexible than SAS. SAS requires SSL while CSIv2 does not. SAS does not support SSL Client Certificate Authentication while CSIv2 does. CSIv2 can require SSL connections while SAS only supports SSL connections. SAS will always have two listener ports opened, TCP/IP and SSL. CSIv2 can have as few as one listener port and as many as three listener ports. One will be opened for the cases of just TCP/IP or when SSL is required. Two will be opened when SSL is supported and three will be opened when SSL is supported and SSL client certificate authentication is supported. There are some other combinations for CSIv2, but this just shows the flexibility of the configuration.

The following are things to keep in mind when configuring the Inbound Transport panels.

Steps for this task

1. Click **Security** > **Authentication Protocol** > **CSIv2 Inbound Transport** to select the type of transport and the SSL settings. By selecting the type of transport, as noted above, you are choosing which listener ports you want to open. In addition, you will be disabling the SSL client certificate authentication feature if you choose TCP/IP as the transport.
2. Select the SSLSettings that will correspond to an SSL transport. These SSL settings are defined in the **Security** -> **SSL** panel and define the SSL configuration including keystore, truststore, file formats, security level, ciphers, cryptographic token selections, etc.
3. You might want to fix the listener ports which you have configured. Doing this is done in a different panel but this is the time to think about it. Most endpoints are managed at a single location which is why they don't appear right here. This helps you to avoid conflicts in your configuration when assigning them. The location for SSL end points is at each server, there's an End Points panel under additional properties for that server. For example, for an application server, go to **Servers** > **Application Servers** > *server_name* > **End Points**. For a Node Agent, go to **System Administration** > **Node Agents** > *node_name* > **End Points**. The end points for the Node Agent and Deployment Manager have already been fixed, but you might want to reassign the ports. For the Deployment Manager, go to **System Administration** > **Deployment Manager** > **End Points**. The following port names are defined in the End Points panels and are used for ORB security.
   - CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS - CSIv2 Client Authentication SSL Port
   - CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS - CSIv2 SSL Port
   - SAS_SSL_SERVERAUTH_LISTENER_ADDRESS - SAS SSL Port
   - ORB_LISTENER_PORT - TCP/IP Port

4. Click **Security** > **Authentication Protocol** > **SAS Inbound** to select the SSL settings used for inbound requests from SAS clients. Keep in mind that the SAS protocol is used to interoperate with previous releases. When configuring the KeyStore and TrustStore files in the SSL configuration, these files should have the right information for interoperating with previous releases of WebSphere Application Server. For example, a previous release has a different TrustStore file than this release. If you use the KeyStore from this release, the signer needs to be added to the TrustStore of the previous release for those clients connecting to this server.

Results

The inbound transport configuration is completed.

Usage scenario

The configuration allows you to configure a different transport for inbound security versus outbound security. For example, if the application server is the first server used by end users, the security configuration might be more secure. When requests go to backend enterprise beans servers, you might lighten up on the security for performance reasons when you go outbound. This flexibility allows you to design the right transport infrastructure to meet your needs.

What to do next

Once finished configuring security, you will need to perform the following steps to save, synchronize, and restart the servers.

1. Click on Save above in the administrative console to save any modifications to the configuration.
2. Synchronize the configuration with all node agents (Network Deployment only).
3. Once synchronized, stop all servers and restart them.

Common Secure Interoperability transport inbound settings: Use this page to specify which listener ports to open and which SSL settings to use. These specifications will determine which transport a client or upstream server will use to communicate with this server for incoming requests.

To view this administrative console page, click **Security** > **Authentication Protocol** > **CSI Transport** > **Inbound**.

Transport: Specifies whether client processes connect to the server using one of its connected transports.

You can choose to use either Secure Socket Layer (SSL), TCP/IP or both as the inbound transport which a server supports. If you specify **TCPIP**, the server only supports TCP/IP and cannot accept SSL connections. If you specify **SSL Supported**, either TCP/IP or SSL connections can be handled by this server. If you specify **SSL required**, then any server communicating with this one must use SSL.

If you specify **SSL Supported** or **SSL Required**, you need to decide which set of SSL configuration settings you want to use for the inbound configuration. This decision determines which key file and trust file will be used for inbound connections to this server.

By default, SSL ports for CSIv2 and SAS are dynamically generated. In cases where you need to fix the SSL ports on application servers, click **Servers** > **Application Servers** > *server_name* > **End Points**. You should configure the following ports to be fixed. A zero port number indicates that a dynamic assignment will be made at runtime.

```
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS
```

- **TCP/IP**: only a TCP/IP listener port will be opened, and all requests inbound will not have SSL protection.
- **SSL Supported**: both a TCP/IP and SSL listener port will be opened, and most requests will come inbound by way of SSL.
- **SSL Required**: only an SSL listener port is opened, and all requests will come through SSL connections. If you choose **SSL Required**, you must also choose **CSI** as the Active Authentication Protocol. If you choose **CSI and SAS**, SAS will require a TCP/IP socket to be open for some special requests.

Default	SSL-Supported
Range	TCP/IP, SSL-Required, SSL-Supported

SSL settings: Specifies a list of predefined SSL settings to choose from for inbound connections. These settings are configured at the SSL Repertoire panel.

Data type	String
Default	DefaultSSLSettings
Range	Any SSL settings configured in the SSL Configuration Repertoire

Secure Association Service transport inbound settings: Use this page to specify transport settings for connections which are accepted by this server using the Secure Association Service (SAS) authentication protocol. The SAS protocol is used to communicate securely to enterprise beans with previous releases of the WebSphere Application Server.

To view this administrative console page, click **Security** > **Authentication Protocol** > **SAS Transport** > **Inbound**.

SSL Settings: Specifies a list of predefined SSL settings to choose from for inbound connections. These settings are configured at the SSL Repertoire panel.

Data type	String
Default	DefaultSSLSettings

Configuring outbound transports

Before you begin

Outbound transports refers to the transport used to connect to a downstream server. When you configure the outbound transport, you should consider the transports which the downstream servers support. If SSL, you need to consider including the downstream servers signers in this server’s trustfile in order for the handshake to succeed. When you select an SSL configuration, the keystore and truststore pointed to by this configuration should contain the necessary signers. If you have configured client certificate authentication for this server in the **Security**

> **Authentication Protocols** > **CSIv2 Outbound Authentication** panel, then the downstream servers should contain the signer certificate of this server's personal certificate.

The following are things to keep in mind when configuring the Outbound Transport panels.

Steps for this task

1. In the **Security** > **Authentication Protocol** > **CSIv2 Outbound Transport** panel, you can select the type of transport and the SSL settings. By selecting the type of transport, as noted above, you are choosing transport to use when connecting to downstream servers. The downstream servers should support the transport you choose. If you choose SSL-Supported, the transport used is negotiated during the connection. If both the client and server supports SSL, that will always be chosen unless the request is considered a special request that does not require SSL such as an ORB is a request.
2. Pick the SSLSettings which will correspond to an SSL transport. These SSL settings are defined in the **Security** > **SSL** panel and define the SSL configuration including keystore, truststore, file formats, security level, ciphers, cryptographic token selections, etc. Ensure that the truststore in the SSL configuration selected contains the signers for any downstream servers. Ensure that the downstream servers contain the signer certificates of this server when client certificate authentication is used outbound.
3. In the **Security** > **Authentication Protocol** > **SAS Outbound** panel, you can select the SSL settings used for outbound requests to downstream SAS servers. Keep in mind that the SAS protocol is used to interoperate with previous releases. When configuring the keystore and truststore files in the SSL configuration, these files should have the right information for interoperating with previous releases of WebSphere Application Server. For example, a previous release has a different personal certificate than this release. If you use the keystore from this release, the signer needs to be added to the truststore of the previous release. Also, the signer for this release should be extracted and imported into the truststore of the previous release.

Results

The outbound transport configuration is completed.

Usage scenario

The configuration allows you to configure a different transport for inbound security versus outbound security. For example, if the application server is the first server used by end users, the security configuration might be more secure. When requests go to backend enterprise beans servers, you might lighten up on the security for performance reasons when you go outbound. This flexibility allows you to design the right transport infrastructure to meet your needs.

What to do next

Once finished configuring all security, you will need to perform the following steps to save, sync and restart the servers.

- Click on **Save** above in the Administrative Console to save any modifications to the configuration.
- Synchronize the configuration with all node agents (Network Deployment only).

- Once synchronized, stop all servers and restart them.

Common Secure Interoperability transport outbound settings: Use this page to specify which transports and SSL settings this server uses when communicating with downstream servers for outbound requests.

To view this administrative console page, click **Security** > **Authentication Protocol** > **CSI Transport** > **Outbound**.

Transport: Specifies whether the client processes connect to the server using one of the server connected transports.

You can choose to use either **SSL**, **TCP/IP** or **Both** as the outbound transport which a server supports. If you specify **TCP/IP**, the server only supports TCP/IP and cannot initiate SSL connections with downstream servers. If you specify **SSL Supported**, this server can initiate either TCP/IP or SSL connections. If you specify **SSL required**, then this server must use SSL to initiate connections to downstream servers. When you do specify SSL, decide which set of SSL configuration settings you want to use for the outbound configuration. This decision determines which key file and trust file to use for outbound connections to downstream servers. For example, consider the following:

- If **TCP/IP**, then this server only opens TCP/IP connections with downstream servers.
- If **SSL Supported**, then this server opens SSL connections with any downstream servers supporting them, and TCP/IP connections with any downstream servers not supporting them.
- If **SSL Required**, then this server always opens SSL connections with downstream servers.

Default	SSL-Supported
Range	TCP/IP, SSL-Required, SSL-Supported

SSL settings: Specifies a list of predefined SSL settings for outbound connections. These settings are configured at the SSL Configuration Repertoires panel.

Data type	String
Default	DefaultSSLSettings
Range	Any SSL settings configured in the SSL Configuration Repertoires panel

Secure Association Service transport outbound settings: Use this page to specify transport settings for connections which are accepted by this server using the Secure Association Service (SAS) authentication protocol. Use the SAS protocol to communicate securely to enterprise beans with previous releases of WebSphere Application Server.

To view this administrative console page, click **Security** > **Authentication Protocol** > **SAS Transport** > **Outbound**.

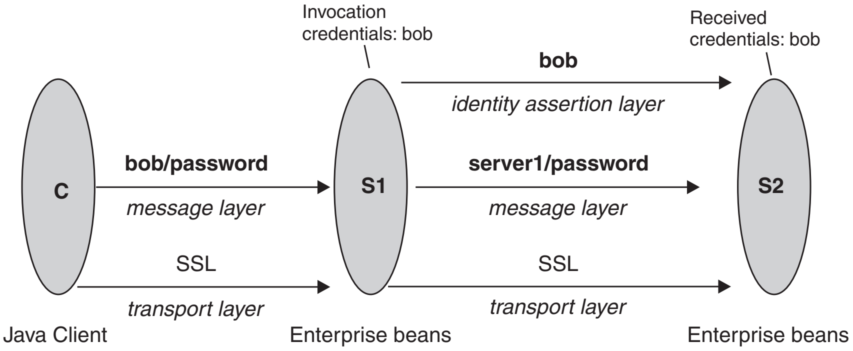
SSL Settings: Specifies a list of predefined Secure Sockets Layer (SSL) settings to choose from for outbound connections. These settings are configured at the SSL Repertoire panel.

Data type	String
Default	DefaultSSLSettings

Example: CSIv2 scenarios

The articles included in this section are intended to demonstrate how to configure specific CSIv2 configuration examples.

Scenario 1: BasicAuth and Identity Assertion:



Scenario Explanation: This is an example of a pure Java client, C, accessing a secure enterprise bean on S1 through user ″bob.″ The enterprise bean code on S1 accesses another enterprise bean on S2. This configuration uses Identity Assertion to propagate the identity of ″bob″ to the downstream server S2. S2 will trust that ″bob″ has already been authenticated by S1 because it trusts S1. To gain this trust, the identity of S1 also flows to S2 simultaneously and S2 will validate the identity by checking the trustedPrincipalList to ensure it is a valid server principal. S2 also authenticates S1. The steps for configuring C, S1 and S2 follow.

Configuring C1: C1 requires message layer authentication with an SSL transport. To accomplish this:

Steps for this task

1. Point the client to the `sas.client.props` file using the property `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props`. All further configuration involves setting properties within this file.
2. Enable SSL. In this case, SSL is supported but not required: `com.ibm.CSI.performTransportAssocSSLTLSSupported=true`, `com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Enable client authentication at the message layer. In this case, client authentication is supported but not required: `com.ibm.CSI.performClientAuthenticationRequired=false`, `com.ibm.CSI.performClientAuthenticationSupported=true`
4. Use all of the remaining defaults in the `sas.client.props` file.

Configuring S1: In the administrative console, S1 is configured for incoming requests to support message layer client authentication and incoming connections to support SSL without client certificate authentication. S1 is configured for outgoing requests to support identity assertion.

Steps for this task

1. Configure S1 for incoming connections.
   a. Disable identity assertion.
   b. Enable user ID/password authentication.
   c. Enable SSL.
   d. Disable SSL client certificate authentication.

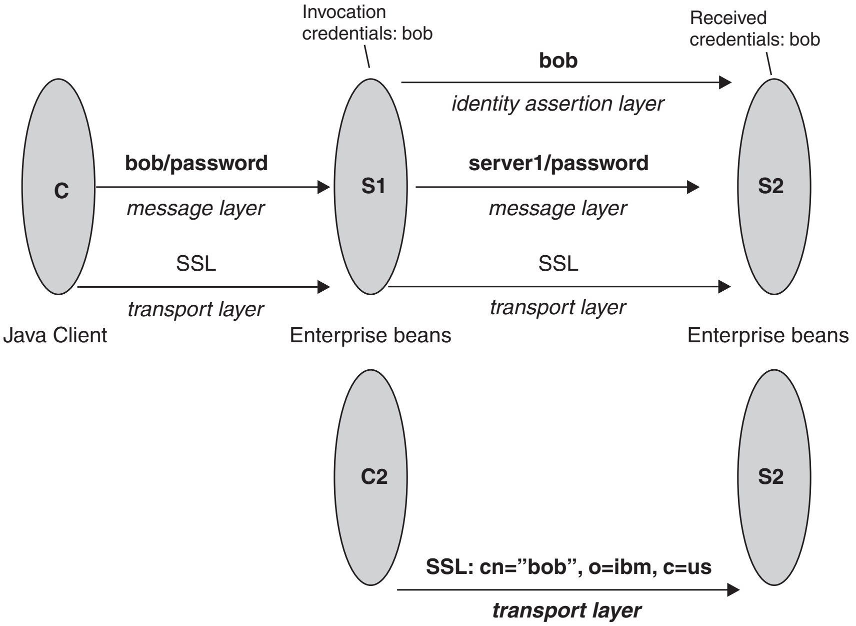
2. Configure S1 for outgoing connections.
   a. Enable Identity Assertion
   b. Disable user ID and password authentication
   c. Enable SSL.
   d. Disable SSL client certificate authentication.

Configuring S2: In the administrative Console, S2 is configured for incoming requests to support identity assertion and to accept SSL connections. Complete the following steps to configure incoming connections. Configuration for outgoing requests and connections are not relevant for this scenario.

Steps for this task

1. Enable identity assertion.
2. Disable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication authentication.

Scenario 2: BasicAuth, Identity Assertion and Client certificates:



Scenario Explanation: This scenario is the same as Scenario 1 except for the interaction from client C2 to server S2. Therefore, the configuration of Scenario 1 still needs to be in place, but you have to modify server S2 slightly and add a configuration for client C2. There is no modification of the configuration for C1 or S1.

Configuring C2: C2 requires transport layer authentication (SSL client certificates). To accomplish this:

Steps for this task

1. Point the client to the `sas.client.props` file using the property `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props`. All further configuration involves setting properties within this file. All further configuration involves setting properties within this file.

2. Enable SSL. In this case, SSL is supported but not required: `com.ibm.CSI.performTransportAssocSSLTLSSupported=true`, `com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Disable client authentication at the message layer. `com.ibm.CSI.performClientAuthenticationRequired=false`, `com.ibm.CSI.performClientAuthenticationSupported=false`
4. Enable client authentication at the transport layer. Here it is supported but not required: `com.ibm.CSI.performTLClientAuthenticationRequired=false`, `com.ibm.CSI.performTLClientAuthenticationSupported=true`

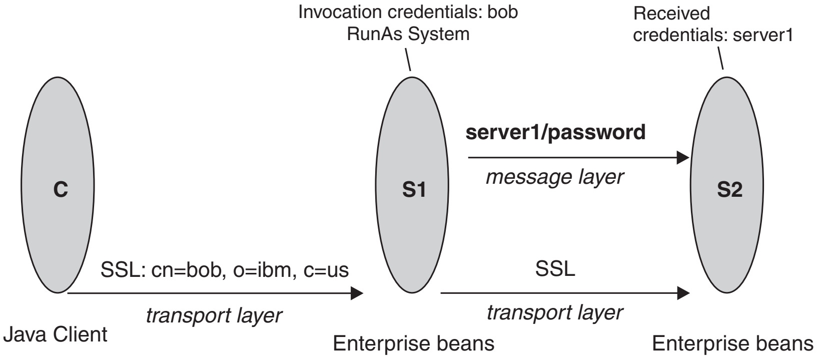
Configuring S2: In the administrative console, S2 is configured for incoming requests to SSL client authentication and identity assertion. Configuration for outgoing requests is not relevant for this scenario.

<u>Steps for this task</u>

1. Configure S2 for incoming connections.
   a. Enable identity assertion.
   b. Disable user ID and password authentication.
   c. Enable SSL.
   d. Enable SSL client authentication authentication.

   Note: You can mix and match these configuration options. However, there is a precedence to which authentication features will become the identity in the received credential. The order of precedence is as follows:
   a. Identity assertion
   b. Message layer client authentication (BasicAuth or token)
   c. Transport layer client authentication (SSL certificates)

Scenario 3: Client certificate authentication and RunAs system:



Scenario Explanation: This is an example of a pure Java client, C, accessing a secure enterprise bean on S1. C authenticates to S1 using SSL client certificates. S1 maps the cn of the DN in the certificate to a user in the local registry. The user in this case is ″bob.″ The enterprise bean code on S1 accesses another enterprise bean on S2. Because the RunAs mode is system, the invocation credential is set as ″server1″ for any outbound requests.

Configuring C1: C1 requires transport layer authentication (SSL client certificates). To accomplish this:

<u>Steps for this task</u>

1. Point the client to the `sas.client.props` file using the property `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props`. All further configuration involves setting properties within this file.
2. Enable SSL. In this case, SSL is supported but not required: `com.ibm.CSI.performTransportAssocSSLTLSSupported=true`, `com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Disable client authentication at the message layer. `com.ibm.CSI.performClientAuthenticationRequired=false`, `com.ibm.CSI.performClientAuthenticationSupported=false`
4. Enable client authentication at the transport layer. It is supported but not required: `com.ibm.CSI.performTLClientAuthenticationRequired=false`, `com.ibm.CSI.performTLClientAuthenticationSupported=true`

Configuring S1: In the administrative console, S1 is configured for incoming connections to support SSL with client certificate authentication. The S1 is configured for outgoing requests to support message layer client authentication.

Steps for this task

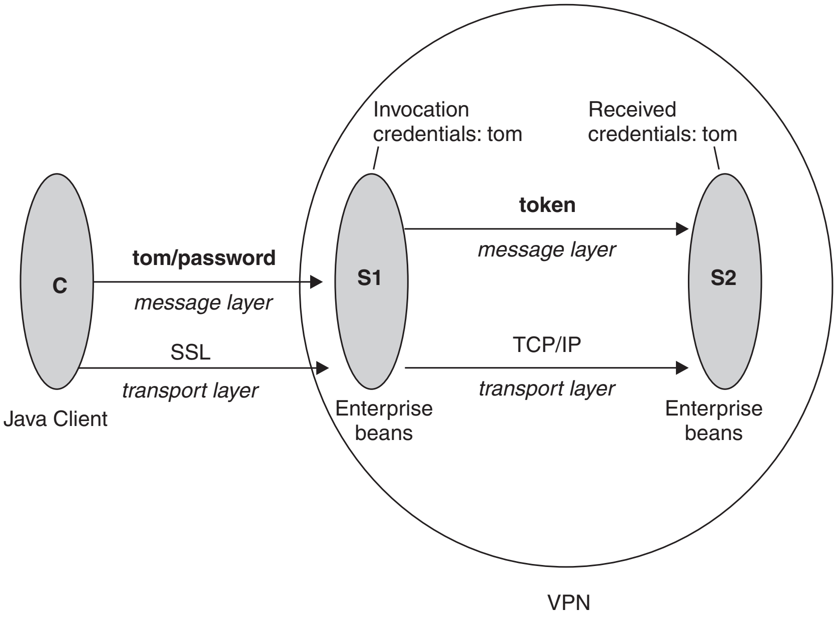
1. Configure S1 for incoming connections.
   a. Disable identity assertion.
   b. Disable user ID and password authentication.
   c. Enable SSL.
   d. Enable SSL client certificate authentication.
2. Configure S1 for outgoing connections.
   a. Disable identity assertion.
   b. Disable user ID and password authentication.
   c. Enable SSL.
   d. Enable SSL client certificate authentication.

Configuring S2: In the administrative console, the S2 will be configured for incoming requests to support message layer authentication over SSL. Configuration for outgoing requests is not relevant for this scenario.

Steps for this task

1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.

Scenario 4: TCP/IP Transport Using VPN:



Scenario Explanation: This scenario illustrates the ability to choose TCP/IP as the transport when it is appropriate. In some cases, when two servers are on the same VPN, it may be appropriate to select TCP/IP as the transport for performance reasons since the VPN already encrypts the message.

Configuring C1: C1 requires message layer authentication with an SSL transport. To accomplish this:

Steps for this task

1. Point the client to the `sas.client.props` file using the property `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props`. All further configuration involves setting properties within this file.
2. Enable SSL. In this case, SSL is supported but not required: `com.ibm.CSI.performTransportAssocSSLTLSSupported=true`, `com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Enable client authentication at the message layer. In this case, client authentication is supported but not required: `com.ibm.CSI.performClientAuthenticationRequired=false`, `com.ibm.CSI.performClientAuthenticationSupported=true`
4. Use all of the rest of the defaults in the `sas.client.props` file.

Configuring S1: In the administrative console, S1 is configured for incoming requests to support message layer client authentication and incoming connections to support SSL without client certificate authentication. S1 is configured for outgoing requests to support identity assertion.

Steps for this task

1. Configure S1 for incoming connections.
   a. Disable identity assertion.
   b. Enable user ID andpassword authentication.
   c. Enable SSL.
   d. Disable SSL client certificate authentication.

2. Configure S1 for outgoing connections.
   a. Disable identity assertion.
   b. Enable user ID andpassword authentication.
   c. Disable SSL.

   Note: It is possible to enable SSL for inbound connections and disable SSL for outbound connections. The same is true in reverse.

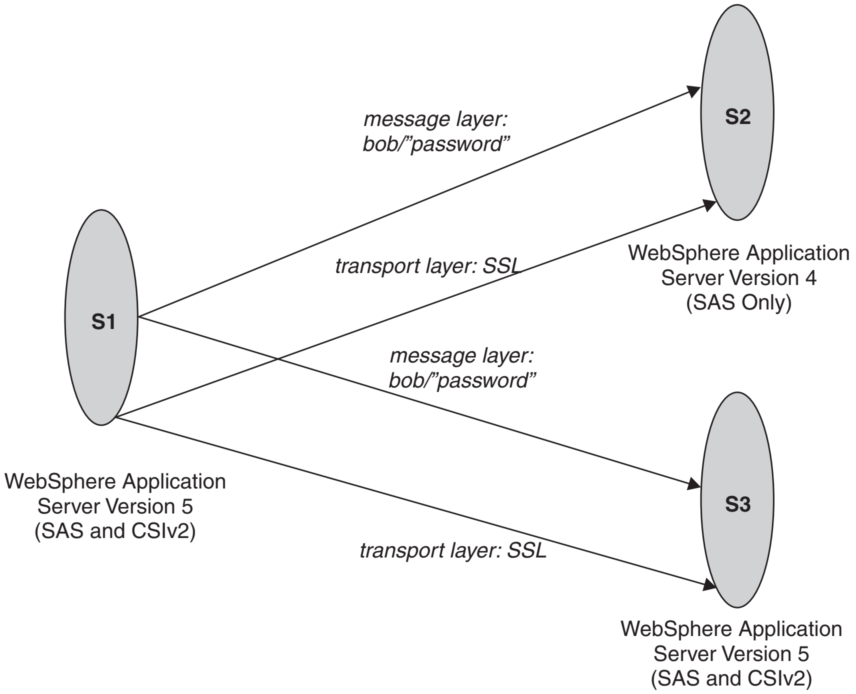
Configuring S2: In the administrative console, S2 is configured for incoming requests to support identity assertion and to accept SSL connections. Configuration for outgoing requests and connections are not relevant for this scenario.

Steps for this task

1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Disable SSL.

Scenario 5: Interoperability with WebSphere Application Server Version 4.x:

Interoperability with WebSphere Application Server Version V4.x


S2
message layer:
bob/"password"
transport layer: SSL
WebSphere Application Server Version 4 (SAS Only)
S1
message layer:
bob/"password"
WebSphere Application Server Version 5 (SAS and CSIv2)
S3
transport layer: SSL
WebSphere Application Server Version 5 (SAS and CSIv2)

Scenario Explanation: The purpose of this scenario is to show how secure interoperability can take place between different releases simultaneously while using multiple authentication protocols (SAS and CSIv2). For a WebSphere Application Server v5 server to communicate with a WebSphere Application Server V4 server, the WebSphere Application Server v5 server must support either IBM or BOTH as the protocol choice. By choosing BOTH, that WebSphere Application Server v5 server can also communicate with other WebSphere Application Server v5 servers which support CSI. If the only servers in your security domain are WebSphere Application Server V5, it is recommended that you choose CSI as the protocol because this prevents the IBM interceptors from loading. However, if there is a chance that any server will need to communicate with a previous release of WebSphere Application Server, select the protocol choice of BOTH.

Configuring S1: S1 requires message layer authentication with an SSL transport. Also, the protocol for S1 must be BOTH. Configuration for incoming requests for S1 is not relevant for this scenario.

Steps for this task

1. Configure S1 for outgoing connections.
   a. Disable identity assertion.
   b. Enable user ID and password authentication.
   c. Enable SSL.
   d. Disable SSL client certificate authentication.
   e. Set authentication protocol to BOTH in the global security settings.

Configuring S2: All previous releases of WebSphere Application Server only support the SAS authentication protocol. There are no special configuration steps needed other than enabling global security on server (S2).

Configuring S3: In the administrative console, S3 is configured for incoming requests to message layer authentication and to accept SSL connections. Configuration for outgoing requests and connections are not relevant for this scenario.

Steps for this task

1. Enable identity assertion.
2. Disable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.
5. Set authentication protocol to either CSI or BOTH.

   Note: If S3 is only communicating with WebSphere Application Server Version 5 servers, it is recommended to choose the **CSI** protocol. Otherwise, choose a protocol of **BOTH**.

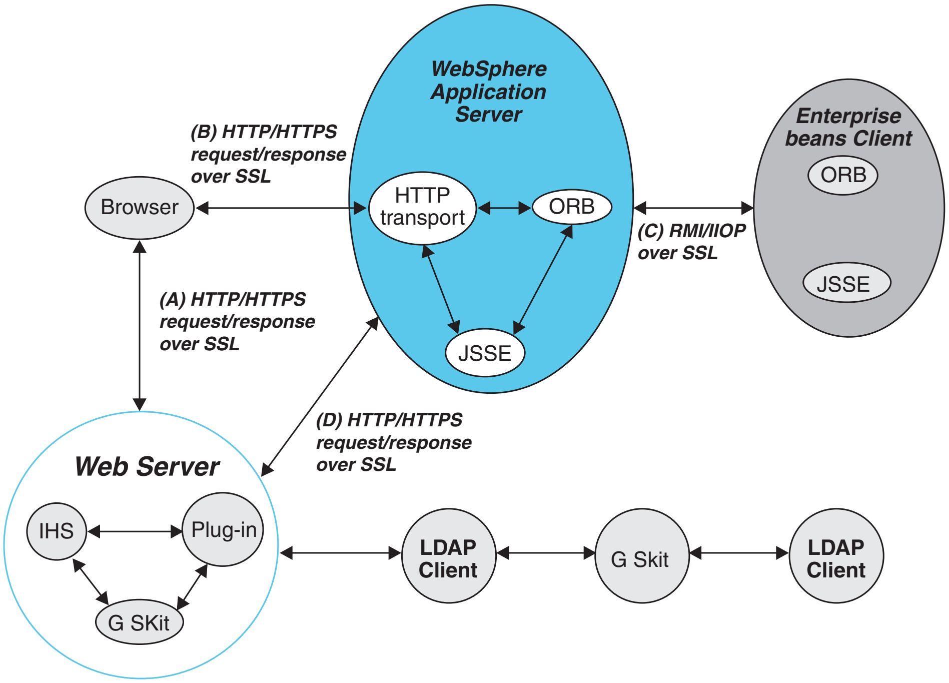
Secure Sockets Layer

The Secure Sockets Layer (SSL) protocol provides transport layer security: authenticity, intergrity, and confidentiality, for a secure connection between a client and server in the WebSphere Application Server. The protocol runs above TCP/IP and below application protocols such as HTTP, LDAP, and Internet Inter-ORB Protocol (IIOP), and provides trust and privacy for the transport data.

Depending upon the SSL configurations of both the client and server, various levels of trust, data integrity, and privacy can be established. Understanding the basic operation of SSL is very important to properly configure and to therefore achieve the desired protection level for both client and application data.

Some of the security features provided by SSL are data encrypted to prevent seeing sensitive information while data flows across the wire. Data signing prevents unauthorized modification of data while data flows across the wire. Client and server authentication ensures that you talk to the appropriate person or machine. Secure Socket Layer is used with many different protocols within the WebSphere Application Server infrastructure including HTTP, LDAP and IIOP. Hence, SSL can be effective in securing an enterprise environment.

SSL is used by multiple components within WebSphere Application Server to provide trust and privacy. These components are the built-in HTTP Transport, the ORB (client and server), and the secure LDAP client.


WebSphere Application Server
Enterprise beans Client
ORB
JSSE
(B) HTTP/HTTPS request/response over SSL
Browser
HTTP transport
ORB
(C) RMI/IIOP over SSL
(A) HTTP/HTTPS request/response over SSL
JSSE
(D) HTTP/HTTPS request/response over SSL
Web Server
IHS
Plug-in
G SKit
LDAP Client
G Skit
LDAP Client

- The built-in HTTP transport in a WebSphere Application Server accepts HTTP requests over SSL from a Web client like a browser.
- The Object Request Broker used in WebSphere Application Server can perform Internet Inter-ORB Protocol (IIOP) over SSL to secure the message.
- The secure LDAP client uses LDAP over SSL to securely connect to an LDAP user registry and is present only when LDAP is configured as the user registry.

WebSphere Application Server and IBM Java Secure Sockets Extension (JSSE)

The SSL implementation used by the WebSphere Application Server is IBM Java Secure Sockets Extension (IBM JSSE). The IBM JSSE contains a reference implementation supporting SSL and TLS protocols and an application programming interface (API) framework. The IBM JSSE also comes with a standard provider which supplies RSA support for the signature-related JCA features of the Java 2 platform, common SSL and TLS cipher suites, hardware cryptographic token device, X.509-based key and trust managers, and PKCS12 implementation for a JCA *keystore*. A graphical tool called IBM Key Management Tool (IKeyman) is also provided to manage digital certificates. With IKeyman, you can create a new key database or a test digital certificate, add CA roots to the database, copy certificates from one database to another, as well as request and receive a digital certificate from a CA.

Configuring JSSE is very similar to configuring most other SSL implementations (for example, GSKit); however, a couple of differences are worth noting.

- JSSE supports both signer and personal certificates storage in an SSL key file, but it also allows a separate file to specify called a *trust file*. A trust file can contain only signer certificates. Therefore, you can put all of your personal certificates in an SSL key file and your signer certificates in a trust file. This might be desirable, for example, if you have an inexpensive hardware cryptographic device with only enough memory to hold a personal certificate. In this case, the key file refers to the hardware device and the trust file to a file on disk containing all of the signer certificates.

- JSSE does not recognize the proprietary SSL key file format which is used by the plug-in (*.kdb* files); instead, it recognizes standard file formats such as Java Key Store (JKS). As such, SSL key files might not be shared between the plug-in and application server. Furthermore, a different implementation of the key management utility (called IKeyMan) must be used to manage application server key and trust files.

There are also certain limitations with using JSSE in Version 5:

- Customer code using JSSE/JCE APIs must reside within a WebSphere Application Server environment. This includes applications deployed in WebSphere Application Server and client applications in the J2EE Application Client environment.
- Only `com.ibm.crypto.provider.IBMJCE`, `com.ibm.jsse.IBMJSSEProvider`, `com.ibm.security.cert.IBMCertPath`, and `com.ibm.crypto.pkcs11.provider.IBMPKCS11` are provided as the cryptography package providers.
- Interoperability of the IBM JSSE implementation with other SSL implementations by vendors is limited to tested implementations. The tested implementations include Microsoft Internet Information Services (IIS), BEA WebLogic Server, IBM z/OS, IBM AIX, and IBM AS/400.
- Hardware token support is limited to supported cryptographic token devices.
- The SSL protocol of Version 2.0 is not supported. In addition, the JSSE and JCE APIs are not supported with Java applet applications.

Authenticity

Authenticity of client and server identities during an SSL connection is validated by both communicating parties using public key cryptography or asymmetric cryptography, to prove the claimed identity from each other.

The public key cryptography is a cryptographic method that uses public and private keys to encrypt and decrypt messages. The public key as a public key certificate is distributed while the private key is kept private. The public key is also a cryptographic inverse of the private key. Well known public key cryptographic algorithms such as Rivest Shamir Adleman (RSA) algorithm and Diffie-Hellman (DH) algorithm are supported in the WebSphere Application Server.

Public key certificate can either be issued by a trusted organization like a Certificate Authority(CA) or extracted from a self-signed personal certificate by IBM Key Management Tool (IKeyman).

Note: A self-signed certificate is less secure and is not recommended for production environment.

The public key certificate includes the following information:

- Issuer of the certificate
- Expiration date
- Subject that the certificate represents
- The Subject's public key
- Signature by the Issuer

Multiple key certificates can be linked into a certificate chain. In a certificate chain, the first is always that of the client while the final is the certificate for a root CA. In between, each certificate is that of the authority that issued the previous one.

During the SSL connection, digital signature is also applied to avoid forged keys. The digital signature is an encrypted hash and cannot be reversed. It is very useful to validate the public keys.

Being optional during the handshake, SSL allows the client to authenticate the server and the server to authenticate the client. By defaults, a WebSphere Application Server client always authenticates its server during the SSL connection. For further protection, a WebSphere application server can also be configured for client authentication.

Please refer to the IBM JSSE documentation located at *install_root*web/docs/jsse/JSSERefGuide.html and the TLS specification at **http://www.ietf.org/rfc/rfc2246.txt** for further information.

Confidentiality

Secure Sockets Layer (SSL) uses private (secret) key cryptography or symmetric cryptography to support message confidentiality or privacy. After an initial handshake (a negotiation process by message exchange), a secret key and a cipher suite are decided between the client and server. Between the communicating parties, each message is then encrypted and decrypted using the secret key based on the cipher suite.

Private key cryptography requires the two communicating parties to use the same key for the encryption and decryption. Both parties must have the key and keep the key private. Well-known secret key cryptographic algorithms include the Data Encryption Standard (DES), triple-strength DES (3DES), and Rivest Cipher 4 (RC4) that are all supported in WebSphere Application Server. These algorithms provide excellent security and quick encryption.

A cryptographic algorithm is called a *cipher* while a set of ciphers is called a *cipher suite*. A cipher suite is a combination of cryptographic parameters that define the security algorithms and key sizes used for authentication, key agreement, encryption strength and integrity protection.

The following cipher suites are supported in WebSphere Application Server:

- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- SSL_DH_anon_WITH_RC4_128_MD5
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5

- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA

where

Name	Description
SSL	Secure Socket Layer
RSA	• A public key algorithm developed by Rivest, Shamir, and Adleman. • Requires RSA or DSS key exchange
DH	• Diffie-Hellman public key algorithm • The server certificate contains the Diffie-Hellman parameters signed by the certificate authority (CA).
DHE	• Ephemeral Diffie-Hellman public key algorithm • The Diffie-Hellman parameters are signed by a DSS or RSA certificate, which is signed by the certificate authority (CA).
DSS	Digital Signature Standard, using Digital Signature Algorithm for digital signatures.
DES	• Data Encryption Standard, an symmetric encryption algorithm • Block cipher • Performance cost is high when using only software (without hardware crypto device support)
3DES	• Triple DES, increasing the security of DES by encrypting three times with different keys • Strongest of the ciphers • Performance cost is very high when using only software (without hardware crypto device support)
RC4	• A stream cipher designed for RSA. • Variable key-size stream cipher with key length from 40 to 128 bits.
EDE	Encrypt-decrypt-encrypt for the triple DES algorithm
CBC	• Cipher block chaining • A mode in which every plaintext block encrypted with the block cipher is first exclusive-ORed with the previous ciphertext block
128	128 bits key size
40	40 bits key size
EXPORT	Exportable

MD5	• Secure hashing function that converts an arbitrarily long data stream into a digest of fixed size • Produce 128-bits hash
SHA	• Secure Hash Algorithm, same as SHA-1 • Produce 160-bits hash
anon	For anonymous connections
NULL	No encryption
WITH	The cryptographic algorithm is defined after this key word

All the above cipher suites provide data integrity protection by using the hash algorithm like MD5 and SHA-1. The cipher suite names ending with _SHA indicates the SHA-1 algorithm is used. SHA-1 is considered a stronger hash, while MD5 provides better performance.

The SSL_DH_anon_xxx cipher suites (for example, those which begin with SSL_DH_anon_, where, anon is *anonymous*) are not enabled on the WebSphere client-side. Since the IBM JSSE client trust manager does not allow anonymous connections, the IBM JSSE client must always establish trust in the server. However, the SSL_DH_anon_xxx cipher suites are enabled on the server-side to allow another type of client to connect which might not require that trust is established in the server. These cipher suites are vulnerable to *man-in-the-middle* attacks and is therefore, **strongly discouraged**.

Refer to the IBM JSSE documentation located at *install_root/web/docs/jsse/JSSERefGuide.html* and the TLS specification at *http://www.ietf.org/rfc/rfc2246.txt* for further information.

Integrity

SSL uses cryptographic hash function which is similar to checksum, to ensure data integrity in transit. The cryptographic hash function is used to detect accidental alternations in the data and does not require a cryptographic key. Once a cryptographic hash is created, the hash is encrypted with a secret key. The encrypted hash is then encrypted by the sender’s private key for digital signature of the message

When secret key information is included with the cryptographic hash, the resulting hash is known as a HMAC (Key-Hashing Message Authentication Code) value. HMAC is a mechanism for message authentication using cryptographic hash functions. It can be used with any iterative cryptographic hash function in combination with a secret shared key.

In WebSphere, both well known one-way hash algorithms of MD5 and SHA-1 are supported. One-way hash is an algorithm that converts processing data into a string of bits known as a hash value or a message digest. The ″one-way″ means that is extremely difficult to turn the fixed string back into the original data.

MD5 is a hash algorithm designed for 32-bit machine. It takes as input a message of arbitrary length and produces as output a 128-bit hash value. Although it is less secure than SHA-1, MD5 does provide better performance.

SHA-1 is a secure hash algorithm specified in the Secure Hash Standard. It is designed to produce a 160-bit hash. Although it is slightly slower than MD5, the larger message digest makes it more secure against attack like brute-force collision.

Please refer to the IBM JSSE documentation located at *install_root*/web/docs/jsse/JSSERefGuide.html and the TLS specification at http://www.ietf.org/rfc/rfc2246.txt for further information.

Configuring Secure Sockets Layer

Secure Sockets Layer (SSL) is used by multiple components within WebSphere Application Server to provide trust and privacy. These components are the built-in HTTP Transport, the ORB (client and server), and the secure LDAP client. Configuring SSL is different between client and server with WebSphere Application Server.

Steps for this task

1. Use the `sas.client.props` file located in the `${was_install_root}/properties` directory.

   The `sas.client.props` file is a configuration file that contains lists of property-value pairs, using the syntax *<property>* = *<value>*. The property names are case sensitive, but the values are not; the values are converted to lower case when the file is read. By default, the *sas.client.props* file is located in the properties directory under the *<install_root>* of your WebSphere Application Server installation. Properties to be specify for an SSL connection include the following:

   - For the CSI and SAS authentication protocol:
     - `com.ibm.ssl.protocol`
     - `com.ibm.ssl.keyStoreType`
     - `com.ibm.ssl.keyStore`
     - `com.ibm.ssl.keyStorePassword`
     - `com.ibm.ssl.trustStoreType`
     - `com.ibm.ssl.trustStore`
     - `com.ibm.ssl.trustStorePassword`
     - `com.ibm.ssl.enabledCipherSuites`
     - `com.ibm.ssl.contextProvider`
     - `com.ibm.ssl.keyStoreServerAlias`
     - `com.ibm.ssl.keyStoreClientAlias`
   - For the SAS authentication protocol only:
     - `com.ibm.CORBA.standardPerformQOPModels`
   - For the cryptographic token device:
     - `com.ibm.ssl.tokenType`
     - `com.ibm.ssl.tokenLibraryFile`
     - `com.ibm.ssl.tokenPassword`

2. Use the administrative console to configure an application server that makes SSL connections. To start the administrative console, specify URL: *http://<server_hostname>:9090/admin*.

   You must first create an SSL Configuration Repertoires alias or entry. You can then select the alias later when a component is configured for SSL support. An SSL configuration repertoires entry contains the following fields:

   - Typical configuration settings:

  - Alias
  - Key File Name
  - Key File Password
  - Key File Format
  - Trust File Name
  - Trust File Password
  - Trust File Format
  - Client Authentication
  - Security Level
  - Cipher Suites
- For the cryptographic token device:
  - Cryptographic Token (Create the alias first so you can configure these fields).
    - Token Type
    - Library File
    - Password
- For additional Java properties:
  - Custom Properties (Create the alias first so you can configure these fields).
    - `com.ibm.ssl.contextProvider`
    - `com.ibm.ssl.protocol`

Configuring Secure Sockets Layer for Web client authentication

To enable client side certificate-based authentication, you must modify the authentication method defined on the J2EE Web Module that you wish to manage. It might be that the Web Module has already been configured to use the basic challenge authentication method. In this case, you will simply need to modify the challenge type to Client cert. As such, this functionality is delivered to the WebSphere Application Server administrator in the Application Assembly Tool (AAT). However, it is envisaged that developers will use the WebSphere Application Server Studio Application Development (WSAD) environment to achieve the same result.

Steps for this task

1. Launch the WebSphere Application Assembly Tool (AAT).

   This step can be undertaken either before an enterprise application archive .ear file has been deployed into the WebSphere Application Server or after deployment into WebSphere Application Server. The latter option is discouraged in a production environment, as it involves opening the expanded archive correlating to the enterprise application archive in question, found under the installedApps directory.
2. Locate and expand the Web Module package under the Application for which you wish to enable the client side certificate authentication method.
3. Select the appropriate Web Application, switch to the Advanced tab and modify the Authentication method to read Client Cert. The Realm name is the scope of the login operation and should be same for all participating resources.
4. Click **OK** and then save the changes made with AAT.
5. If the modification was made to a resource already deployed into WebSphere Application Server, stop and start the associated application server containing the resource, so that the security modification can be included in the runtime.

Results

Now your enterprise application will prompt the user to prove its identity with a certificate.

Note: The Web Server must also be configured to request a client certificate. If the Web Server is an external Web Server, you need to refer to documentation for steps to configure it. If the Web Server being used is the Web Container transport (for example, 9043) within WebSphere Application Server, you need to ensure that the **client authentication** flag is checked in the referenced SSL configuration.

Refer to the article, Map certificates to users, to determine how a certificate is then authenticated within the product.

Usage scenario

To enable a user to login using certificates.

Configuring SSL for the LDAP client

This topic describes how to establish a Secure Sockets Layer (SSL) connection between WebSphere Application Server and an Lightweight Directory Access Protocol (LDAP) server. This page gives an overview; refer to the linked pages for more details. To understand SSL concepts, refer to (Secure Sockets Layer (SSL)).

Setting up an SSL connection between WebSphere Application Server and an LDAP server requires the following steps:

Steps for this task

1. Set up an LDAP server with users.

   The server configured in this example is IBM Directory Server Version 4.1. Other servers are configured differently. You should refer to the documentation of the directory server you are using for details on SSL enablement.
2. Set up certificates for LDAP Server using the IKeyMan that is shipped with the IBM HTTP Server (IHS) product.
3. Click **Key Database File** > **New**.
4. Type `LDAPkey.kdb` as the file name and a proper path.
5. Select **Personal Certificates**. Click **New Self-Signed Certificate**. This action brings up the Create New Self-Signed Certificate panel, type in the following information in the fields:
   - Key Label —- LDAP_Cert
   - Common Name —- droplet.austin.ibm.com This common name is the host name where the WebSphere Application Server plug-in runs.
   - Organization—- ibm
   - Country —- US

   a. Click **OK**.
6. Return to the Personal Certificates panel, and click **Extract Certificate**.
7. Choose the Data type as **Base64-encoded ASCII data**. Type `LDAP_cert.arm` as the file name and a proper path. Click **OK**.
8. Enable SSL on the LDAP Server with the following steps:

   a. Copy the LDAPkey.kdb , `LDAPkey.sth`, LDAPkey.rdb, and `LDAPkey.crl` files created above to the LDAP server's system, say in `\Program Files\IBM\LDAP\ssl\` directory.

b. Open the LDAP's Web administrator from a browser (`http://secnt3.austin.ibm.com/ldap`, for example). IBM HTTP Server should be running on `secnt3`.

c. Click **SSL properties** to open the SSL setting window.

d. Click **SSL On** > **Server Authentication** and type an SSL port (**636**, for example) and a full path to the `LDAPkey.kdb` file.

e. Click **Apply**, and restart the LDAP server.

9. Manage certificates for WebSphere Application Server using the default SSL key files.

   a. Open `<`*`install_root`*`>\etc\DummyServerTrustFile.jks` using the IKeyMan that shipped with WebSphere Application Server. The password is `WebAS`.

   b. Click **Personal Certificates** with the pull-down tab. Click **Import**.

      The Import Key panel appears. In the File Name, specify `LDAP_cert.arm`. Complete this step for all the servers including the cell manager.

10. Establish a connection between WebSphere Application Server and LDAP server.

    a. In the administrative console, click **User Registry** > **LDAP User Registry** > **LDAP Settings**. Fill in the **Server ID**, **Server Password**, **Type**, **Host**, **Port** and **Base Distinguished Name** fields. Select the **SSL Enabled** check box. The port is the one that the LDAP server is using for SSL (636, for example). Then click **Apply**.

    b. Click **Authentication Mechanisms** > **LTPA** > **Single SignOn (SSO)**. Type in a domain name (`austin.ibm.com`, for example). Then click **Apply**.

11. Enable global security.

    a. Click **Security** > **Global Security**. Select the **Enabled** checkbox. Choose **LTPA** as the active authentication mechanism and **LDAP** as the active user registry. Then click **Apply** and **Save**.

       Note: The default security level is **HIGH** (128-bit). Ensure the security level for the LDAP server is set to **HIGH**. Check the file `<`*`LDAP_install_root`*`>\etc\slapd32.conf`, the parameter ibm-slapdSSLCipherSpecs should have the value 15360 instead of 12288.

    b. Restart the servers. In case of Network Deployment setup, see the article Enable and removing global security in a Network Deployment environment. Restarting the servers ensures that the Security settings are synchronized between the cell manager and the application servers.

Results

You can test the setup by accessing `https://<`*`fully_qualified_hostname`*`>:9443/snoop`. You are then presented with a login challenge.

Usage scenario

This can be beneficial when using LDAP as your user registry. Sensitive information can flow between the WebSphere Application Server and the LDAP server including passwords. Using SSL to encrypt the data will protect this sensitive information.

What to do next

1. If you enabling security, make sure you complete the remaining steps. As the final step make sure that you validate this setup by clicking **OK** or **Apply** in

the Global Security panel. Save, stop and start all WebSphere Application Servers. Refer to the article, Configuring Global Security for detailed steps on enabling global security

2. For any changes in this panel to be effective, save, stop and start all WebSphere Application Servers (Cell, Nodes and all the application servers).
3. Once the server comes up, make sure to go through all the security related tasks (getting users, getting groups, and so on.) to make sure the changes to the filters are functioning.

Configuring IBM HTTP Server for SSL Mutual Authentication

IBM HTTP Server, as of version 1.3.24, supports SSL version 3 and version 2 and TLS version 1. IBM HTTP Server is based on the Apache Web server, still for SSLconfiguration it is necessary to use the IBM-supplied SSL modules, rather than the OpenSSL varieties. This document will describe configuration of IBM HTTP Server, although it is entirely possible that another supported Web server is used in its place.

SSL is disabled by default and it is necessary to modify a configuration file and generate a server side certificate using the IKeyMan tool provided with IBM HTTP Server in order to enable SSL.

Steps for this task

1. For a single server, enable SSL on IBM HTTP Server (port 443,for example).
2. To set up certificates complete the following:

   Start the IKeyMan tool. To start the tool, click **Start** > **Programs** > **IBM HTTP Server** > **Start Key Management Utility**. Refer to Requesting a CA-signed personal certificate, Creating a certificate signing request (CSR), Receiving a CA-signed personal certificate, Extracting a Public Certificate for use in a Truststore
3. Create a key database. Click **Key Database File** >**New**.
4. Type a file name, `serverkey.kdb`, for example, and the path for location. Click **OK**.
5. Type a password, check the **Stash the password to a file** box, and then click **OK**.
6. The Verisign Test CA Root Certificate is in the set of signer certificates shipped with the IKeyMan for IBM HTTP Server.
7. Obtain a personal certificate for IBM HTTP Server : choose **Personal Certificate Requests** in the pull-down menu of IKeyMan. Click **New**. The Create New Key and Certificate Request panel appears. Fill in the following information:
   - Key Label — Server_Cert
   - Common Name — droplet.austin.ibm.com
   - Organization — IBM
   - Country — US
   - file name — Server_certreq.arm

   a. Go to URL `http://www.verisign.com`, click on Get Free Trial SSL ID. Complete the profile information, click **Submit**, and click **Continue** twice.
8. Use Notepad to edit the request file `Server_certreq.arm`, and copy the entire contents of the file into the browser request panel. Click **Continue**.
9. VeriSign displays the Personal Certificate in the browser. Copy and paste this certificate into a file, for example `Server_Cert.arm`. Choose the Personal

Certificate option from the pull-down menu in IKeyMan. Click **Receive**. Specify the filename as `Server_Cert.arm`, and click **OK**. Close `serverkey.kdb`.

10. To allow IBM HTTP Server to support https, say at port 443, enable SSL on IBM HTTP Server. Modify the configuration file of IBM HTTP Server, `<IHS_HOME>/conf/httpd.conf`, although SSL can be enabled through the HTTP Server Administration console also. Open the file `<IHS_HOME>/conf/httpd.conf` and then add the following lines into the file above the line:

    ```
    Alias  /IBMWebAS/   "<install_root>/web"
    LoadModule  ibm_ssl_module   modules/IBMModuleSSL128.dll
    LoadModule  ibm_app_server_http_module
    <install_root>/bin/mod_ibm_app_server_http.dll
    Listen 443
    <VirtualHost  droplet.austin.ibm.com:443>
    ServerName  droplet.austin.ibm.com
    DocumentRoot  <install_root>\htdocs
    SSLEnable
    #SSLClientAuth  required
    SSLDisable
    Keyfile <IHS_HOME>/serverkey.kdb
    ```

    Note: You need to change the host name and the path for the key file accordingly. Also, you must modify the Web server to support client certificates. Uncomment the directive shown above in `httpd.conf file`.

    ```
    SSLClientAuth   required
    ```

11. Restart IBM HTTP Server.
12. Test SSL between a browser and IBM HTTP Server by accessing the following URL: `https://droplet.austin.ibm.com`.

    ″Welcome to the IBM HTTP Server″ appears on the browser.
13. If SSLClientAuth directive is set to required, your browser should prompt you to select a personal certificate.
14. To enable the Application Server to communicate with IBM HTTP Server using port 443, by adding the host alias on the default_host. Click **Environment** > **Virtual Hosts** > *default host* > **Host Aliases** > **New**.

    Enter the following in the appropriate fields:
    - host name — *
    - port type — 443
15. Click **Apply** and **Save** to write to `security.xml`.
16. Click **Update Web Server Plugin**, and then click **OK**.
17. Restart WebSphere Application Server.
18. Test your connection by accessing the following: **https://droplet.austin.ibm.com:443/snoop**.

Results

You can connect to the Snoop servlet.

Usage scenario

Enable Secure Sockets Layer communication between IBM HTTP Server and the WebSphere Application Server.

Configuring IHS plug-in and the internal Web server for SSL

This section documents the configuration necessary to instantiate a secure connection between the Web server plug-in and the embedded HTTP server in the WebSphere Application Server Web Container. By default, this connection is not secure, even when Global Security is enabled. This document will cover the configuration for the IBM HTTP Server 1.3.24; however, the Web server related configuration in this situation is not specific to any Web server.

Steps for this task

1. Create a self-signed certificate for the Web server plug-in.

   The Web server plug-in requires a keyring to store its own private and public keys and to store the public certificate from the Web container's keyfile. The following steps are required to generate a self-signed certificate for the Web server plug-in.

   a. Create a suitable directory on the Web server host for storing the keyring file referenced by the plug-in and associated files; for example: `<IHS_install_root>\conf\keys`.

   b. Launch the IKeyman tool that comes with the IBM HTTP server.

   c. From the ikeyman menu, select **Key Database File** > **New**

   d. Set the following settings:
      - Key Database file: **CMS Key Database File**
      - File name: **WASplugin.kdb**
      - Location: **C:\http1324\conf\keys\** (or file of your choice)

   e. Click **OK**.

   f. At the password prompt, set the password of your choice. Set the Stash the Password to a File box and the password will be saved to a stash file, so the plug-in can use the password to gain access to the certificates contained in the key database.

   g. From the ikeyman menu, select **Create** > **New Self-Signed Certificate** to create a new self-signed certificate key pair. The following options need to be specified, you may choose to complete all of the remaining fields for the sake of completeness:
      - Key Label: **WASplugin**
      - Version: **X509 V3**
      - Key Size: **1024**
      - Common Name: **droplet.austin.ibm.com**
      - Organisation: **IBM**
      - Country: **US**
      - Validity Period: **365**

   h. Click **OK**.

   i. Extract the public self-signed certificate key, as this will be used later by the embedded HTTP server peer to authenticate connections originating from the plug-in.

   j. Select **Personal Certificates** in the drop-down menu and select the WASplugin certificate that was just created.

   k. Click **Extract Certificate**. Extract the certificate to a file:
      - Data type: **Base64-encoded ASCII data**
      - Certificate file name: **WASpluginPubCert.arm**
      - Location: **C:\http1324\conf\keys** (or directory of your choice)

   l. Click **OK**.

   m. Close the key database and exit the Ikeyman tool when you are finished.

2. Generate a self-signed certificate for the Web Container.

a. Launch the IBM JKS capable ikeyman version that ships under the WebSphere bin directory.

b. From the ikeyman menu, select **Key Database File** > **New**.

c. Set the following settings:

   - Key database file: **JKS**
   - File name: **WASWebContainer.jks**
   - Location: **C:\WebSphere\AppServer\etc\** (or directory of your choice)

d. Click **OK**.

e. At the password prompt window, enter the password of your choice.

f. From the ikeyman menu, select: **Create** > **New Self-Signed Certificate**. The following values were used in this example:

   - Key Label: **WASWebContainer**
   - Version: **X509 V3**
   - Key Size: **1024**
   - Common Name: **droplet.austin.ibm.com**
   - Organization: **IBM**
   - Country: **US**
   - Validity Period: **365**

g. Click **OK**.

h. Extract the public self-signed certificate key, as it will be used later by the Web server plug-in peer to authenticate connections originating from the embedded HTTP server in WebSphere Application Server.

i. Select Personal Certificates from the drop-down list, then select the WASWebContainer certificate that was just created. Click **Extract Certificate**. Extract the certificate to a file:

   - Data Type: **Base64-encoded ASCII data**
   - Certificate file name: **WASWebContainerPubCert.arm**
   - Location: **C:\WebSphere\AppServer\etc\**

j. Click **OK**.

k. Close the database and exit the IKeyman tool.

3. Exchange the public certificates.

   a. Copy the WASpluginPubCert.arm from the Web server machine to the WebSphere Application Server machine. The source directory in this case is `C:\http1324\conf\keys`, while the destination is `C:\WebSphere\Appserver\etc`.

   b. Copy the WASWebContainerPubCert.arm from the WebSphere machine to the Web server machine. The source directory in this case is `C:\WebSphere\Appserver\etc`, while the destination is `C:\http1324\conf\keys`.

4. Import the certificate into the Web server plug-in keyfile.

   a. On the Web server machine launch the ikeyman utility that supports the CMS key database format.

   b. From the ikeyman menu, select **Key Database File** > **Open** and select the previously created key database file: `WASplugin.kdb`.

   c. In the password prompt window, enter the password, then click **OK**.

   d. Select Signer Certificates from the drop-down list, click **Add**. This will allow you to import the public certificate previously extracted from the embedded HTTP server (Web Container) keystore.

   - Data type: **Base64-encoded ASCII data**
   - Certificate file name: **WASWebContainerPubCert.arm**
   - Location: **C:\WebSphere\Appserver\etc\**

   e. Click **OK**.

   f. You will be prompted for a label name by which the trusted signer public certificate will be known. Enter a label for the certificate: `WASWebContainer`.

   g. Close the key database and exit IKeyman when you are finished.

5. Import the certificate into the Web Container keystore

   a. On the WebSphere Application Server machine, launch the IBM JKS capable ikeyman version that ships under the WebSphere Application Server bin directory.

   b. From the ikeyman menu, select: **Key Database File** > **Open** and select the previously created `WASWebContainer.jks` file.

   c. In the password prompt window, enter the password, and then click **OK**.

   d. Select Signer Certificates from the drop-down list, click **Add**. This will allow you to import the public certificate previously extracted from the embedded HTTP server (Web Container) keystore.
   - Data type: **Base64-encoded ASCII data**
   - Certificate file name: **WASWebContainerPubCert.arm**
   - Location: **C:\WebSphere\Appserver\etc\**

   e. Click **OK**.

   f. You will be prompted for a label name by which the trusted signer public certificate will be known. Enter a label for the certificate: `WASWebContainer`.

   g. Close the key database and exit IKeyman when you are finished.

   h. You will be prompted for a label name by which the trusted signer public certificate will be known. Enter a label for the certificate: WASplugin.

   i. Close the key database and exit IKeyman when you are finished.

6. Modify the Web server plug-in file.

   It will be useful for production environment to replace the original plugin-key.kdb file with your own key file for the secure transport definition, port 9443.

7. Modify the Web Container to support SSL.

   To complete the configuration between Web server plug-in and Web Container, the WebSphere Application Server Web Container must be modified to use the previously created self-signed certificates.

   a. Start the WebSphere administration console.

   b. Click **Security** > **SSL Configuration Repertoires**.

   c. Click **New** to create a new entry in the repertoire. Provide the following values to fill out the form:
   - Alias: WebContainerSSLSettings
   - Key File Name: **C:\WebSphere\Appserver\etc\WASWebContainer.jks**
   - Key File Password: ***<key_file_password>***
   - Key File Format: **JKS**
   - Trust File Name: C**:\WebSphere\Appserver\etc\WASWebContainer.jks**
   - Trust File Password: ***<trust_file_password>***
   - Trust File Format: **JKS**
   - Client Authentication:

- Security Level: **HIGH**

d. Click **OK**.

e. In case you want mutual SSL between the two parties, click the Client Authentication check-box.

f. Save the configuration in the WebSphere Application Server administration console.

g. Select the **Servers** > **Application Servers**, and then select the server you want to work with, in this case server1.

h. Select the Web Container under the server.

i. Select HTTP Transport under the Web Container.

j. Select the entry for the transfer you want to secure, click the item under the Host column. Select the * (asterisk) in this case in the line where 9443 is the Port.

k. On the configuration panel, click the Enable SSL box, select the desired SSL entry from the repertoire from the SSL drop down list in this example, the WebContainerSSLSettings.

l. Click **OK**.

8. Test the secure connection.

   a. To test the secure connection access a Web application on WebSphere Application Server using the port 9443, for example: https://droplet.austin.ibm.com:9443/snoop.

9. In order to test the secured connection when client side certification is required, import the right certificate with public and private key into the browser.

   a. On the Web server machine, launch the ikeyman utility that can handle the CMS key database file.

   b. Open the keyfile for the plug-in, `C:\http1324\conf\keys\WASplugin.kdb`. Provide the password when you are prompted.

   c. Select **WASplugin certificate** under the Personal Certificates, then click on **Export**.

   d. Save the certificate in PKCS12 format to a file, C:\http1324\conf\keys\WASplugin.p12. Provide a password to secure the PKCS12 certificate file.

   e. Close the keyfile and exit IKeyman.

   f. Copy the saved WASplugin.p12 file to the client machine where you want to access the WebSphere server from.

   g. Import the PKCS12 file into your browser. Then access **https://droplet.austin.ibm.com:9443/snoop**.

   h. The browser will ask which personal certificate to use for the connection. Select the certificate then continue the connection.

   i. Once the browser test with the direct WebSphere access is successful, test the connection through the Web server using port 9443, using client certificate, **https://droplet.austin.ibm.com:443/snoop**.

Results

The IBM HTTP Server plug-in and the internal Web server are configured for SSL.

Usage scenario

To enable Secure Socket Layer communication between IHS plug-in and the embedded HTTP server (Web Container) in the WebSphere Application Server

Configuring SSL for Java client authentication

WebSphere Application Server supports Java client authentication using digital certificate when the client attempts to make a SSL connection. The authentication occurs during a SSL handshake. The SSL handshake is a series of messages exchanged over the SSL protocol to negotiate for a connection-specific protection. During the handshake, the secure server requests the client to send back a certificate or certificate chain for the authentication.

Before you begin

To configure SSL for Java client authentication, you need to first consider the following:

- Have you enabled security with your WebSphere Application Server? Refer to Configuring global security for more details.
- Have you configured CSI authentication protocol for your target application server?

  Note: The SAS authentication protocol does not support Java client authentication with SSL transport.

  Refer to Configuring global security for more details.
- Have you configured your server to support secure transport for the inbound CSI authentication protocol?
- Have you configured your server to support client authentication at the transport layer for the inbound CSI authentication protocol?
- If you are using a self-signed personal certificate, have you exported the public certificate from your client application Java Keystore file or cryptographic token device?
- If you are using a CA-signed (certificate authority) personal certificate, have you received the root certificate of the CA?
- If you are using a self-signed personal certificate, have you imported the public certificate into your target Java Truststore file as a signer certificate?
- If you are using a CA-signed (certificate authority) personal certificate, have you imported the CA root certificate into your target Java TrustStore file as a signer certificate?
- Does the Common Name (CN) specified in your personal certificate name exist in your configured user registry?

If you answer yes for all the above questions, you can proceed to configure the SSL for Java client authentication.

Note: Java client authentication using digital certificates is supported only by the CSIv2 authentication protocol.

Steps for this task

1. Configure CSIv2 for SSL client authentication.
2. Add Keystore files.
3. Add Truststore files.
4. Save changes.
5. Restart the server if you have configured the server.

Usage scenario

Secure client connects to a secure IIOP server that requires client authentication at transport layer.

What to do next

If a connection problem occurs, you can set a Java property, `javax.net.debug=true`, before you run your client or your server to generate debugging information. See (Troubleshooting security configurations) for further information about how to debug a IBM JSSE problem.

Configuring CSIv2 for SSL client authentication:

Before you begin

The configuration for the SSL client authentication can be done by using `sas.client.props` configuration file or the administrative console. To configure a Java client application, use the `sas.client.props` configuration file. By default, the `sas.client.props` is located in the properties directory under the <*product_installation_root*> of your WebSphere Application Server installation.

To configure a WebSphere Application Server, use the administrative console. To start the administrative console, specify URL: `http://<server hostname>:9090/admin`.

To configure a Java client application, complete <samp/>the following steps, which explain how to edit the <samp>sas.client.props file.

Steps for this task

1. To require SSL client authentication, set property `com.ibm.CSI.performTLClientAuthenticationRequired=true`.

   Note: Do not set this property unless you know your target server also supports SSL client authentication for the inbound CSI authentication protocol.
2. To support SSL client authentication, set the property `com.ibm.CSI.performTLClientAuthenticationSupported=true`.
3. To specify CSI protocol, set the property `com.ibm.CSI.protocol=csiv2`.
4. To match the SSL protocol configured with your server, set the property, `com.ibm.ssl.protocol`, accordingly.
5. Specify the property com.ibm.CORBA.ConfigURL with the fully-qualified path of your Java property file when you run your application.

   For example, `-Dcom.ibm.CORBA.ConfigURL=file:/c:/WebSphere/AppServer/properties/sas.client.props`.

To configure a WebSphere Application Server, complete the following steps

Steps for this task

1. Starts the administrative console.
2. Expand **Security** > **Authentication Protocol**.
3. Click **CSIv2 Inbound Authentication**.
4. Select **Supported** or **Required** for Client Certificate Authentication.
5. Click **OK**.
6. If you checked Required in step 4, you need to configure the CSIv2 outbound authentication as well to support the client certificate authentication. Otherwise,

you can skip this step. To configure, click **CSIv2 Outbound Authentication** and check either **Supported** or **Required** for Client Certificate Authentication.

7. Click **CSIv2 Outbound Transport**. Select an SSL setting from the SSLSettings list for keystore, truststore, cryptographic token, SSL protocol, and ciphers to be used. If you have not created yet, you need to create an alias from the SSL Configuration Repertoires panel for a SSL setting. Note that you should update the SSL setting selected in CSIv2 Inbound Transport accordingly.
8. Save your configuration.
9. Restart the server for the changes to take effect.

Results

Client authentication using digital certificates is performed during SSL connection.

Usage scenario

Secure client connects using SSL to a secure IIOP server with client authentication at the transport layer.

What to do next

Specify the Keystore and Truststore files in your configuration.

Adding keystore files: Keystore file contains both the public keys and private keys. Public keys are stored as signer certificates while private keys are stored in the personal certificates. In WebSphere Application Server, adding keystore files to the configuration is different between client and server. For client, keystore file is added to a property file like sas.client.props. For server, keystore file is added through the WebSphere Application Server administration console.

Before you begin

Before you add the keystore file to your configuration, the following need to be considered:

- Has a self-signed or a CA-signed personal certificate been created in the keystore?
- If you configure for client authentication using digital certificate, has the public key of the signed personal certificate been imported as a signer certificate into the server's truststore?

Steps for this task

1. To add a keystore file into a client configuration, edit the sas.client.props and set the following properties:
   - **com.ibm.ssl.keyStoreType** for the keystore format. Range: JKS (Default), PKCS12KS, JCEK.
   - **com.ibm.ssl.keyStore** for a fully qualified path to the keystore file. The keystore file contains private keys and perhaps public keys.
   - **com.ibm.ssl.keyStorePassword** for the password to access the keystore file.
2. To add a keystore file into a server configuration, you need to do the following:
   a. Starts the WebSphere admin console by specifying URL: http://<server hostname>>:9090/admin.
   b. Expand **Security** > **SSL Configuration Repertoires**.
   c. Create a new SSL setting alias if you have not done it yet.

d. Select the alias that you want to add the keystore into.
e. Type the Key File Name for the path of the keystore file.
f. Type in Key File Password for the password to access the keystore file.
g. Select the Key File Format for the keystore type. Range: JKS (Default), PKCS12KS, JCEK.
h. Click **OK** and **Save** to save the configuration.

Results

The SSL configuration alias now has a valid keystore for SSL connection.

Note: If the Cryptographic Token field is checked and you only want to use cryptographic token for your keystore, you should leave the fields of **Key File Name** and **Key File Password** blank.

Usage scenario
- SSL connection for IIOP
- SSL connection for LDAP
- SSL connection for HTTP

Adding truststore files:

Before you begin

A truststore file is a key database file that contains public keys. The public key is stored as a signer certificate. The keys are used for a variety of purposes, including authentication and data integrity. In WebSphere Application Server, adding truststore files to the configuration is different between client and server. For the client, truststore file is added to a property file, like `sas.client.props`. For the server, truststore file is added through the WebSphere Application Server administrative console.

Before you add the truststore file to your configuration, the following need to be considered:
- If you configure for client authentication using digital certificate, has the public key of the client personal certificate been imported as a signer certificate into the server truststore file?
- Does the truststore file contain all the required signer certificates with respect to the keystore files of the target servers?

Steps for this task
1. To add a truststore file into a client configuration, edit the sas.client.props and set the following properties:
   - **com.ibm.ssl.trustStoreType** for the truststore format. Range: JKS (Default), PKCS12KS, JCEK.
   - **com.ibm.ssl.trustStore** for a fully qualified path to the truststore file. The truststore file contains the public keys.
   - **com.ibm.ssl.trustStorePassword** for the password to access the truststore file.
2. Add a truststore file into a server configuration.
   a. Start the WebSphere admin console by specifying URL: **http://<server hostname>:9090/admin**.
   b. Expand **Security** > **SSL Configuration Repertoires**.

c. Create a new SSL setting alias if you have not done it yet.

d. Select the alias that you want to add the truststore into.

e. Type the Trust File Name for the path of the truststore file.

f. Type the Trust File Password for the password to access the truststore file.

g. Select the Trust File Format for the truststore type. JKS (Default), PKCS12KS, JCEK.

h. Click **OK**.

i. Save the configuration.

Results

The SSL configuration alias has now contained a valid truststore for SSL connection.

Usage scenario

- SSL connection for IIOP
- SSL connection for LDAP
- SSL connection for HTTP

Creating an SSL repertoire configuration entry

The first step in configuring SSL is to define an SSL configuration repertoire. A repertoire contains the details necessary for building an SSL connection, such as the location of the key files, their type and the available ciphers. WebSphere Application Server provides a default repertoire called DefaultSSLSettings. To view this page in the administrative console, click **Security** > **SSL** to see the list of SSL repertoire settings.

The appropriate repertoire is referenced during the configuration of a service that sends and receives requests encrypted using SSL, such as the Web and enterprise beans containers. Before deleting SSL configurations from the repertoire, remember that if an SSL configuration alias is referenced somewhere, and it is deleted here, an SSL connection will fail if the deleted alias is accessed.

The SSL configuration repertoire allows administrators to define any number of SSL settings which can be used to make HTTPS, IIOPS or LDAPS connections. You can pick one of the SSL settings defined here from any location within the administrative console which allows SSL connections. This simplifies the SSL configuration process since you can reuse many of these SSL configurations by simply specifying the alias in multiple places.

Steps for this task

1. From the SSL Configuration Repertoire window, click **New**. Type an Alias by which the configuration will be known. Click **OK**.
2. Select the new SSL configuration repertoire by clicking the link.
3. Now click **Secure Sockets Layer (SSL)** link in Additional Properties. The new configuration details can be entered in the window that appears.
4. Type the location of the key file name.
5. Type the password for the key file.
6. Repeat the above two steps for the trust file.
7. If Client Authentication is supported by this configuration, then check the Client Authentication box. This will only affect HTTP and LDAP request.

8. The appropriate security level must be set. Valid values are low, medium and high. Low specifies only digital signing ciphers (no encryption), medium specifies only 40 bit ciphers (including digital signing), high specifies only 128-bit ciphers (including digital signing).
9. If the preset security level does not define the required cipher, it can be manually added to the cipher suite option.
10. Select the **Cryptographic Token** box if hardware or software cryptographic support is available. Please refer to the InfoCenter for details regarding cryptographic support.
11. Additional properties can be added by selecting the Custom Properties link in the Additional Properties section.
12. Click **OK** to apply the changes.
13. If there are no errors, save the changes to the master configuration and restart WebSphere Application Server.

Results

When you have completed configuring this panel, you will have additional SSL Configuration repertoires besides DefaultSSLSettings.

Usage scenario

The appropriate repertoire is referenced during the configuration of a service that sends and receives requests encrypted using SSL, such as the Web and enterprise beans Containers, LDAP servers.

What to do next

For the changes to take effect, you'll need to restart the server after having saved the configuration.

Secure Socket Layer configuration repertoire settings: Use this page to define a new Secure Socket Layer (SSL) alias. The SSL configuration repertoire allows administrators to define any number of SSL settings which can be used to configure the HTTPS, IIOPS or LDAPS connections. You can pick one of the SSL settings defined here from any location within the administrative console that allows SSL connections. This simplifies the SSL configuration process since you can reuse many of these SSL configurations by simply specifying the alias in multiple places.

To view this administrative console page, click **Security** > **SSL**.

Click **New** to create a new SSL Configuration Repertoire alias.

Click **Delete** to remove a SSL Configuration Repertoire alias. Try not to delete aliases which are referenced elsewhere in the configuration.

Alias: Specifies the name of the specific SSL setting.

Before deleting SSL configurations from the repertoire, remember that if an SSL configuration alias is referenced somewhere, (and it is deleted here) then an SSL connection fails when the deleted alias is accessed.

Secure Socket Layer settings: Use this page to configure Secure Socket Layer settings for the server.

To view this administrative console page, click **Security** > **SSL Configuration Repertoires** > *alias_name* > **New**.

Alias: Specifies the name of the specific SSL setting

Data type	String

Key File Name: Specifies the fully qualified path to the SSL key file that contains public keys and private keys.

You can create an SSL key file with the IKeyMan key management utility, or it can correspond to a hardware device if one is available. In either case, this option indicates the source for personal certificates, as well as for signer certificates unless a trust file is specified. For default SSL key files, DummyClientKeyFile.jks or DummyServerKeyFile.jks, each contains a self-signed personal test certificate expiring on March 17, 2005. The test certificate is only intended for use in a test environment. The default SSL key files should never be used in a production environment because the private keys are the same on all the WebSphere Application Server installations. Refer to the Managing certificates article in the InfoCenter for information about creating and managing digital certificates for your WebSphere Application Server domain.

Data type	String

Key File Password: Specifies the password for accessing the SSL key file.

Data type	String

Key File Format: Specifies the format of the SSL key file.

Data type	String
Default	JKS
Range	JKS, JCEK, PKCS12

Trust File Name: Specifies the fully qualified path to a trust file containing the public keys.

You can create a trust file with the IKeyMan utility included in the WebSphere *bin* directory. Using the IKeyMan utility from GSKit (another SSL implementation) does not work with the WebSphere JSSE implementation.

Unlike the SSL key file, no personal certificates are referenced; only signer certificates are retrieved. For default SSL trust files, DummyClientTrustFile.jks and DummyServerTrustFile.jks, each contains multiple test public keys as signer certificates that can expire. The public key for the WebSphere Application Server Version 4.0 test certificates expires on January 15, 2004, and the public key for the WebSphere Application Server Version 5.0 test certificates and WebSphere Application Server CORBA C++ client expires on March 17, 2005. The test certificate is only intended for use in a test environment.

If a trust file is not specified but the SSL key file is specified, then the SSL key file is used for retrieval of signer certificates as well as personal certificates.

Data type	String

Trust File Password: Specifies the password for accessing the SSL trust file.

Data type	String

Trust File Format: Specifies the format of the SSL trust file.

Data type	String
Default	JKS
Range	JKS, JCEK, PKCS12

Client Authentication: Specifies whether to request a certificate from the client for authentication purposes when making a connection.

This attribute is only valid when used by the Web Container HTTP transport. When performing client authentication with the IIOP protocol (for EJB requests), you must click **Security** > **Authentication Protocol** > **CSIv2 Inbound** or **Outbound Authentication** from the left navigation pane of the administrative console. Select **SSL Client Certificate Authentication** to enable it for these requests.

Data type	Boolean
Default	Disabled
Range	Enabled or Disabled

Security Level: Specifies whether the server will select from a preconfigured set of security levels.

Data type	Valid values include **Low**, **Medium** or **High**. • LOW specifies only digital signing ciphers (no encryption) • MEDIUM specifies only 40-bit ciphers (including digital signing) • HIGH specifies only 128-bit ciphers (including digital signing). To specify all ciphers or any particular range, you can set the property **com.ibm.ssl.enabledCipherSuites**.See the SSL documentation in the InfoCenter.
Default	High
Range	Low, Medium or High

Cipher Suites: Specifies a list of supported cipher suites which can be selected for use during the SSL handshake. If you select cipher suites individually here, you will be overriding those cipher suites set in the Security Level field.

Enable Cryptographic Token Support: Specifies whether the server will enable or disable cryptographic hardware and software support.

Data type	Boolean
Default	Disabled
Range	Enabled or Disabled

Secure Socket Layer settings (additional): Use this page to configure additional Secure Socket Layer (SSL) settings for a defined alias.

To view this administrative console page, click **Security** > **SSL Configuration Repertoire** > *alias_name*.

Custom Properties: Specifies the name-value pairs that you can use to configure additional SSL settings beyond those available in the administrative interface `com.ibm.ssl.protocol`.

This is the SSL protocol used (including its version). The possible values are SSL, SSLv2, SSLv3, TLS, or TLSv1. The default value, SSL, is backward-compatible with the other SSL protocols.

com.ibm.ssl.keyStoreProvider. The name of the key store provider to use. Specify one of the security providers listed in your java.security file which has a key store implementation. The default value is IBMJCE.

com.ibm.ssl.keyManager. The name of the key management algorithm to use. Specify any key management algorithm that is implemented by one of the security providers listed in your java.security file. The default value is IbmX509.

com.ibm.ssl.trustStoreProvider. The name of the trust store provider to use. Specify one of the security providers listed in your java.security file which has a trust store implementation. The default value is IBMJCE.

com.ibm.ssl.trustManager. The name of the trust management algorithm to use. Specify any trust management algorithm that is implemented by one of the security providers listed in your java.security file. The default value is IbmX509.

com.ibm.ssl.trustStoreType. The type or format of the trust store. The possible values are JKS, PKCS12, JCEK. The default value is JKS.

com.ibm.ssl.enabledCipherSuites. The list of cipher suites to enable. By default, this is not set and the set of cipher suites used are determined by the value of the Security Level (High, Medium, or Low). A cipher suite is a combination of cryptographic algorithms used for an SSL connection. Enter a space-separated list of any of the following cipher suites:

- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_WITH_NULL_MD5

- SSL_RSA_WITH_NULL_SHA
- SSL_DH_anon_WITH_RC4_128_MD5
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA

Data type	String

Cryptographic Token: Specifies information about the cryptographic tokens related to SSL support.

A crypto token is a hardware or software device which has a built-in key store implementation. Document the exact values for the following fields in the found in the literature of your supported cryptographic device.

Digital certificates

Certificates provide a way of authenticating users (*authentication*, as referred to by trusted third parties). Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of certificates, electronic ID cards.

A digital certificate is equivalent to an electronic ID card. It serves two purposes:

- To establish the identity of the owner of the certificate
- To distribute the owner's public key

Certificates are issued by trusted parties, called *certificate authorities* (CAs). These authorities can be commercial ventures or they can be local entities, depending on the requirements of your application. Regardless, the CA is trusted to adequately authenticate users before issuing certificates to them. Also, when a CA issues certificates, it digitally signs them. When a user presents a certificate, the recipient of the certificate validates it by using the digital signature. If the digital signature validates the certificate, the certificate is known to be intact and authentic. Participants in an application need only to validate certificates; they do not need to authenticate users themselves. The fact that a user can present a valid certificate proves that the CA has authenticated the user. The descriptor *trusted third-party* indicates that the system relies on the trustworthiness of the CAs.

Contents of a digital certificate: A certificate contains several pieces of information, including information about the owner of the certificate and the issuing CA. Specifically, a certificate includes:

- The distinguished name (DN) of the owner. A DN is a unique identifier, a fully qualified name including not only the common name (CN) of the owner but the owner's organization and other distinguishing information.
- The public key of the owner.
- The date on which the certificate was issued.
- The date on which the certificate expires.
- The distinguished name of the issuing CA.
- The digital signature of the issuing CA. (The message-digest function is run over all the preceding fields.)

The core idea of a certificate is that a CA takes the owner's public key, signs the public key with the its own private key, and returns this to the owner as a certificate. When the owner distributes the certificate to another party, it signs the certificate with its private key. The receiver can extract the certificate (containing the CA's signature) with the owner's public key. By using the CA's public key and the CA's signature on the extracted certificate, the receiver can validate the CA's signature. If it is valid, the public key used to extract the certificate is known to be good. The owner's signature is then validated, and if the validation succeeds, the owner has successfully authenticated to the receiver.

The additional information in a certificate allows an application to decide if it should honor the certificate. With the expiration date, the application can determine if the certificate is still valid. With the name of the issuing CA, the application can check that the CA is considered trustworthy by the site.

A process that uses certificates must be able to provide its personal certificate, the one containing its public key, and the certificate of the CA that signed its certificate, called a signer certificate. In cases where chains of trust are established, several signer certificates may be involved.

Requesting certificates: To get a certificate, you must send a certificate request to the CA. The certificate request includes the following:

- The distinguished name of the owner (the user for whom the certificate is being requested).
- The public key of the owner.
- The digital signature of the owner.

The message-digest function is run over all these fields.

The CA verifies the signature with the public key in the request to ensure that the request is intact and authentic. The CA then authenticates the owner. Exactly what the authentication consists of depends on a prior agreement between the CA and the requesting organization. If the owner in the request is successfully authenticated, the CA issues a certificate for that owner.

Using certificates: Chain of trust and self-signed certificate: To verify the digital signature on a certificate, you must have the public key of the issuing CA. Since public keys are distributed in certificates, you must have a certificate for the issuing CA. That certificate will be signed by the issuer. One CA can certify other CAs, so there can be a chain of CAs issuing certificates for other CAs, all of whose public keys you need. Eventually, though, you reach a starting point. The starting point is a root CA that issues itself a self-signed certificate. In order to validate a user's certificate, you need certificates for all intervening participants, back to the root CA. Then you have the public keys you need to validate each certificate, including the user's.

A self-signed certificate contains the public key of the issuer and is signed with the private key. The digital signature is validated like any other, and if the certificate is valid, the public key it contains can be used to check the validity of other certificates issued by the CA. However, anyone can generate a self-signed certificate. In fact, you will probably generate self-signed certificates for testing purposes before installing production certificates. The fact that a self-signed certificate contains a valid public key does not mean that the issuer is really a trusted certificate authority. In order to ensure that self-signed certificates are generated by trusted CAs, such certificates must be distributed by secure means (hand-delivered on floppy disks, downloaded from secure sites, and so forth).

Applications that use certificates store those certificates in a keystore file. This file typically contains the necessary personal certificates, its signing certificates, and its private key. The private key is used by the application to create digital signatures. Servers will always have personal certificates in their keystore files. A client requires a personal certificate only if the client must authenticate to the server, that is, when mutual authentication is enabled.

To allow a client to authenticate to a server, a server's keystore file contains the server's private key and certificate and the certificates of its CA. A client's truststore must contain the signer certificates of the CAs of each server to which the client must authenticate.

If mutual authentication is needed, the client's keystore must contain the client's private key and certificate. The server's truststore needs a copy of the certificate of the client's CA.

Digital signatures: A digital signature is a number attached to a document. For example, in an authentication system that uses public-key encryption, digital signatures are used to sign certificates.

This signature establishes two different things for you:

- The integrity of the message: Is the message intact? That is, has the message been modified between the time it was digitally signed and now?
- The identity of the signer of the message: Is the message authentic? That is, was the message actually signed by the user who claims to have signed it?

A digital signature is created in two steps. The first consists of distilling the document down into a large number. This number is the digest code or fingerprint. The digest code itself is then encrypted, resulting in the digital signature. The digital signature is appended to the document from which the digest code was generated.

There are several ways of generating the digest code—WebSphere Application Server supports the MD5 message digest function and the SHA1 secure hash algorithm—but all of them reduce a message to a number. This process is not encryption; rather, it is a sophisticated checksum. The message cannot be regenerated from the resulting digest code. The crucial aspect of distilling the document down to a number is this: if the message is changed, even in trivial way, a different digest code results. This means that when the recipient gets a message and verifies the digest code by recomputing it, any changes in the document will result in a mismatch between the stated and the computed digest codes. If a message is changed, the resulting digest code changes as well.

So far, there is nothing to stop someone from intercepting a message, changing it, recomputing the digest code, and retransmitting the modified message and code. We need a way to verify the digest code as well. This is done by reversing the use of the public and private keys. For private communication, it makes no sense to encrypt messages with your private key; these can be decrypted by anyone with your public key. But this technique can be useful for proving that a message must have come from you. No one else could have created it, since no one else has your private key. If some meaningful message results from decrypting a document by using someone's public key, it verifies the fact that the holder of the corresponding private key did, in fact, encrypt the message.

The second step in creating a digital signature takes advantage of this reverse application of public and private keys. After a digest code has been computed for a document, the digest code itself is encrypted with the sender's private key. The result is the digital signature, which is simply attached to the end of the message.

When the message is received, the recipient follows these steps to verify the signature:

1. Recompute the digest code for the message.
2. Decrypt the signature by using the sender's public key. This yields the original digest code for the message.
3. Compare the original and recomputed digest codes. If they match, the message is both intact and authentic. If not, something has changed and the message is not to be trusted.

Public key cryptography: All encryption systems rely on the notion of a key. A key is the basis for a transformation, usually mathematical, of an ordinary message into a unreadable one. For centuries, most encryption systems have relied on what is called private-key encryption. Only within the last 30 years has a challenge to private-key encryption appeared: public-key encryption.

Private key encryption: Private-key encryption systems use a single key. This requires the sender and the receiver to share the key. Both must have the key; the sender encrypts the message by using the key, and the receiver decrypts the message with the same key. Both must keep the key private to keep their communication private. This kind of encryption has characteristics that make it unsuitable for widespread, general use:

- It requires a key for every pair of individuals who need to communicate privately. The necessary number of keys rises dramatically as the number of participants increases.
- The fact that keys must be shared between pairs of communicators means the keys must somehow be distributed to the participants. The need to transmit secret keys makes them vulnerable to theft.
- Participants can communicate only by prior arrangement. There is no way to send a usable encrypted message to someone spontaneously. You and the other participant must have made arrangements to communicate by sharing keys.

Private-key encryption is also called symmetric encryption, because the same key is used to encrypt and decrypt the message.

Public key encryption: Public-key encryption uses a pair of mathematically related keys. A message encrypted with the first key must be decrypted with the second, and a message encrypted with the second key must be decrypted with the first.

Each participant in a public-key system has a pair of keys. One of these keys is kept secret; this is the private key. The other is distributed to anyone who wants it; this is the public key.

To send an encrypted message to you, the sender encrypts the message by using your public key. When you receive it, you decrypt it by using your private key. When you wish to send a message to someone, you encrypt it by using the recipient's public key. The message can be decrypted only with the recipient's private key. This kind of encryption has characteristics that make it very attractive for general use:

- Public-key encryption requires only two keys per participant. The total number of keys rises much less dramatically as the number of participants increases than it does in private-key encryption.

- The need for secrecy is more easily met. The only thing that needs to be kept private is the private key, and since it does not need to be shared, it is less vulnerable to theft in transmission than the shared key in a private-key system.
- Public keys can be published. This eliminates the need for prior sharing of a secret key before communication. Anyone who knows your public key can use it to send you a message that only you can read.

Public-key encryption is also called asymmetric encryption, because the same key cannot be used to encrypt and decrypt the message. Instead, one key of a pair is used to undo the work of the other. WebSphere Application Server uses the RSA public/private key-encryption algorithm.

With private-key encryption, you have to be careful of stolen or intercepted keys. In public-key encryption, where anyone can create a key pair and publish the public key, the challenge is in verifying that the owner of the public key really is the person you think it is. There is nothing to stop a user from creating a key pair and publishing the public key under a false name. The person listed as the owner of the public key will not be able to read messages encrypted with that key because he or she will not have the private key. If the creator of the false public key can intercept these messages, that person can decrypt and read messages intended for someone else. To counteract the potential for forged keys, public-key systems provide mechanisms for validating public keys (and other information) with digital signatures and digital certificates.

Managing digital certificates

Secure Socket Layer (SSL) connections rely on the existence of *digital certificates*. A digital certificate reveals information about its owner, such as their identity. During the initialization of an SSL connection, the server must present its certificate to the client for the client to determine the server identity. The client can also present the server with its own certificate for the server to determine the client identity. SSL is therefore, a means for propagating identity between components. Refer to Configuring Secure Socket Layer and Configuring a SSL Repertoire Configuration Entry.

A client can trust the contents of a certificate if that certificate is digitally signed by a trusted third party. Certificate Authorities (CA) act as a trusted third party and sign certificates on the basis of their knowledge of the certificate requestor. Following are different approaches for generating certificates.

Steps for this task

1. Use the supplied key generation utility, *IKeyMan*. Refer to Starting the IKeyman tool. There are two options for creating a new certificate.
   a. Request that a CA generates the certificates on your behalf.

      Refer to Requesting a CA signed personal certificate, Creating a certificate signing request (CSR), Receiving a CA-signed personal certificate, and Extracting a Public Certificate for use in a Truststore. The CA creates a new certificate, digitally signs it, and delivers it to the requester. Popular Web browsers are pre-configured to trust certificates that are signed by certain CAs. So no further client configuration is necessary for a client to connect to the server through an SSL connection. Therefore, CA signed certificates are useful where configuration for each and every client that accesses the server is impractical.
   b. Generate a self-signed certificate.

      Refer to Create a Keystore file, Create a Trust store file, Adding Keystore files, Adding Truststore files, Create a self-signed personal Certificate, and

Importing a Signer Certificate. This might be the quickest option and require fewer details to create the certificate. However, the certificate is not signed by a CA. Any client that connects to this server over an SSL connection needs to be configured to trust the signer of this certificate. Therefore, self-signed certificates are only useful when you can configure each of the clients to trust the certificate. It is possible in some cases to present a self-signed certificate to an untrusting client. In some Web browsers, when the certificate is received and is found to not match any of those listed in the client trust file, a prompt appears asking if the certificate should be trusted for the connection and added to the trust file.

   c. For Java clients, refer to Configuring SSL for Java Client Authentication. In contrast, the WebSphere server stores the key store information in the repository and the key stores are referred to in the *security.xml* file. Therefore, do all server-side configuration through the administration tools, such as the administration console.

2. Use the command line Java utility called *keytool*. With keytool you can create a private and public self-signed certificate key pair.

   a. For this example, the first user is *cn=rocaj*. Specify **RSA** for the private key to ensure that the *MD5 with RSA* signature algorithm is used. Not all Web browsers support the *DSA* cryptograph algorithm (which is the default when RSA is not specified). Set a password of at least six characters to protect the private key. Finally, specify the keystore and keystore password (the option is storepass). The following example shows these steps.

      (Lines in the following example have been split for publication.)

      ```
      #keytool -genkey -keyalg RSA -dname "cn=rocaj, ou=users,
      ou=uk, dc=internetchaos,dc=com" -alias rocaj -keypass websphere
      -keystore testkeyring.jks -storepass websphere
      ```

   b. Create the second private and public self-signed certificate key pair in the same manner for the user *cn=amorv*.

      (Lines in the following example have been split for publication.)

      ```
      #keytool -genkey -keyalg RSA -dname "cn=amorv, ou=users,
      ou=uk, dc=internetchaos, dc=com" -alias amorv -keypass websphere
      -keystore testkeyring.jks -storepass websphere
      ```

   c. Now the keystore *testkeyring.jks* contains two self-signed certificates, with the owner being the same as the issuer for each certificate. Ensure the integrity and authenticity of the certificates by getting each certificate signed by the Certificate Authority.

      - Generate the Certificate Signing Request, CSR-1 (for the first user cn=rocaj).

        (Lines in the following example have been split for publication.)

        ```
        #keytool -v certreq -alias rocaj -file rocajReq.csr
        -keypass websphere -keystore testkeyring.jks -storepass websphere
        ```

      - Generate the CSR-2 (for the second user cn=amorv).

        (Lines in the following example have been split for publication.)

        ```
        #keytool -v -certreq -alias amorv  -file amorvReq.csr
        -keypass websphere -keystore testkeyring.jks -storepass websphere
        ```

   d. For this example, you can use the free Test SSL certificate program offered by Thawte Consulting to sign the Certificate Signing Requests (CSRs). In each case, the Custom Cert option is selected and the Certificate format is set to use the default for your kind of certificate. The example also selects the **Generate an X.509v3 Certificate** option and saves the two resulting files as *rocajRes.arm* and *amorvRes.arm*, respectively.

e. Before you can receive the signed certificate response back into the keystore, you must import the CA's trusted root certificate into the keystore. Copy and paste the Thawte test root certificate in BASE64-encoded ASCII data format to a file called *ThawteTestCA.arm*. Then add the test root CA certificate into the keystore with the following command:

   (Lines in the following example have been split for publication.)

   ```
   #keytool -import -alias "Thawte Test CA Root"
   -file ThawteTestCA.arm -keystore testkeyring.jks -storepass websphere
   ```

f. Import the two certificate responses from the CA into the keystore using the same alias name as first given to the self-signed certificates. In this example, these were *rocaj* and *amorv* respectively. Using an alternative alias name generates a new signer certificate and not a personal certificate chain.

   - Import the certificate response -1 (for the first user cn=rocaj).

     (Lines in the following example have been split for publication.)

     ```
     #keytool -import -trustcacerts -alias rocaj
     -file rocajRec.arm -keystore testkeyring.jks -storepass websphere.
     Certificate reply was installed in keystore.
     ```

   - Import the certificate response -2 (for the second user cn=amorv).

     (Lines in the following example have been split for publication.)

     ```
     #keytool -import -trustcacerts -alias amorv
     -file amorvRec.arm -keystore testkeyring.jks -storepass websphere.
     Certificate  reply was installed in keystore.
     ```

g. Launch the IBM JSSE Ikeyman utility, which supports the PKCS12 format and allows you to export the private key associated with any certificate (the public key is also exported).

h. Open the *testkeyring.jks* keystore and select the first certificate from the Personal Certificates drop-down menu.

i. Click **Export**, name the file *rocajprivate.p12*. Export the second Personal Certificate and name it *amorvprivate.p12*.

j. Before you can install a personal certificate signed by a third party CA into a given Web browser, you must make sure that the same root certificate of the authenticating CA is installed as a trusted authority in the browser.

k. To install either of the personal certificates into Netscape Communicator, click **Communicator > Tools > Security Info > Certificates > Yours**. Use the **Import a Certificate** option.

l. Netscape prompts you to enter a password or PIN for the Communicator Certificate DB, when you attempt to import the certificate. Enter the password as used when first initializing your Certificate DB. You are also prompted to enter the password protecting the PKCS#12 certificate file, as set when you exported the personal private and public certificate key pair in Ikeyman.

m. Once imported, select the **Verify** option to check integrity and validity of the certificate. If you did not install the root CA certificate, your certificate fails the verification.

n. Ensure that you have modified your Web server to support client side certificate requests.

o. When you connect to the following URL: *https://<server_name>/snoop*, the Web browser prompts you to select a personal certificate when accessing a resource protected by the *SSLClientAuth* directive.

p. If you select the HTTPS Information displayed by the snoop servlet, you now see the certificate SubjectDN matching the following: **Subject: CN=amorv, OU=users, OU=uk, DC=internetchaos, DC=com**.

3. Refer to Create a SSL Repertoire Configuration Entry to create a new SSL definition entry for WebSphere using the administrative console.

   Once a key store is configured, either by creating a self-signed certificate or by creating a certificate request and importing the reply, WebSphere Application Server can be configured to make use of the certificates. The product uses the certificates to establish a secure connection with a client through SSL.
4. Set up the appropriate components to use the newly-defined SSL configuration.

   To ensure a secure connection, configure some non-WebSphere components, such as a Web server. A digital certificate is created for each component. The WebSphere Application Server owns a certificate and the Web server owns another certificate. Refer to Configuring IHS for SSL Mutual Authentication.

Usage scenario

To set up SSL communication between the Web browser and WebSphere Application Server. Using digital signatures, you can communicate securely from the Web browser through Web server to WebSphere Application Server.

What to do next

Once finished configuring security, perform the following steps to save, synchronize, and restart the servers.
1. Click **Save** in the administrative console to save any modifications to the configuration.
2. Synchronize the configuration with all node agents (Network Deployment only).
3. Once synchronized, stop all servers and restart them.

Starting the IKeyman tool:

Before you begin

It is recommended to read the documentation located at `<production_installation_root>/web/docs/ikeyman/ikmuserguide.pdf` for further information.

WebSphere Application Server provides a graphical tool, the IBM Key Management tool (iKeyman), for managing keys and certificates. With iKeyman, you can do the followings:
- Create a new key database
- Create a self-signed digital certificate
- Add CA roots to the key database as a signer certificate
- Request and receive a digital certificate from a CA

To start the iKeyman tool complete the following steps.

Steps for this task
1. Move to the `<product_installation_root/bin directory`.
2. Issue one of the following commands:
   - On Windows systems, `ikeyman.bat`
   - On UNIX systems, `ikeyman.sh`

Results

A graphical user interface of the Ikeyman tool appears.

Usage scenario

Manage keys and digital certificates.

Creating a Keystore file: The keystore file is a a key database file that contains both the public keys and private keys. Public keys are stored as signer certificates while private keys are stored in the personal certificates. The keys are used for a variety of purposes, including authentication and data integrity. Both the IKeyman tool and the Keytool utility can be used to create the Keystore files.

Before you begin

It is recommended to read the documentation located at *<production_installation_root>*/web/docs/ikeyman/ikmuserguide.pdf. for further information.

Steps for this task

1. Start iKeyman, if it is not already running.
2. Open a new key database file by selecting Key Database File > **New** from the menu bar.
3. Select the Key Database Type: JKS (default), PKCS12, and JCEK. This is the *key file format* (or the value of `com.ibm.ssl.keyStoreType` property in the `sas.client.props` file) when you configure the SSL setting for your application.
4. Type in the file name and location. The full path of this key database file is used as the *key file name* (or the value of `com.ibm.ssl.keyStore` property in the `sas.client.props` file) when you configure the SSL setting for your application.
5. Click **OK** to continue.
6. Then, type in password to restrict access to the file. This password is used as the *key file password* (or the value of `com.ibm.ssl.keyStorePassword` property in the `sas.client.props`) when you configure the SSL setting for your application. Do not set an expiration date on the password or save the password to a file. You must then reset the password when it expires or protect the password file. This password is used only to release the information stored by iKeyman during runtime.
7. Click **OK** to continue. The tool displays all of the available default signer certificates. These are the public keys of the most common CAs. You can add, view or delete signer certificates from this screen.

Results

A new SSL keystore file is created.

Usage scenario

Prepare keystore files for a SSL connection.

What to do next

Specify the keystore in the configuration of WebSphere Application Server. A truststore should also be created if it has not been created yet.

Creating self-signed personal certificates: A self-signed personal certificate is a temporary digital certificate you issue to yourself, with yourself as the CA. Creating a self-signed certificate creates a private key and a public key within the key database file. The self-signed certificate is created in a keystore file and it is useful when you develop and test your application. You can also create a self-signed personal certificate from your cryptographic token device.

Before you begin

If you want to create a self-signed certificate for a keystore, you must have already created the keystore file. You can later extract the public key and add the key as a signer certificate to other truststore files.

Read the file <production_installation_root>/web/docs/ikeyman/ikmuserguide.pdf for further information about how to create a self-signed personal certificate within a key database file.

Steps for this task

1. Start iKeyman, if it is not already running.
2. Click **New Self-Signed** from the tool bar or select **Create** > **New Self-Signed Certificate** from the menu.
3. Select the X509 version and key size that suit your application.
4. Enter the appropriate information for your self-signed certificate.
   - **Key Label**. Give the certificate a key label, which is used to uniquely identify the certificate within the keystore. If you have only one certificate in each keystore, you can assign any value to the label. However, it is good practice to use a unique label related to the server name.
   - **Common Name**. Enter the common name. This is the primary, universal identity for the certificate; it should uniquely identify the principal that it represents. In a WebSphere environment, certificates frequently represent server principals, and the common convention is to use CNs of the form host_name/server_name. Note that the Common Name must be valid in the configured user registry for the secured WebSphere environment.
   - **Organization**. Enter the name of your organization.
   - **Optional fields**. Enter the organization unit (a department or division), location (city), state/province (if applicable), zip code (if applicable), and select the two-letter identifier of the country in which the server belongs. For a self-signed certificate, these fields are optional. However, commercial CAs may require them.
   - **Validity period**. Specify the lifetime of the certificate in days, or accept the default.
5. Click **OK**.

Results

Your key database file now contains a self-signed personal certificate.

Usage scenario

Create a self-signed test certificate for testing purposes.

What to do next

If you need a test certificate that has been signed by a Certificate Authority (CA), follow the procedure in Creating a certification request.

Requesting CA-signed personal certificates: In a production environment, a personal certificate signed by a Certificate Authority (CA) should be used. The principal or the owner of the CA-signed personal certificate is actually authenticated by a CA when the CA signs the principal certificate. Since the certificate authorities (CAs) keep their private keys secure, the signed certificate is more trustworthy than a self-signed certificate. Certificate Authorities are entities that are trusted to issue valid certificates for other entities. Well-known CAs include VeriSign, Entrust and GTE CyberTrust. *You can request a test certificate or a production certificate from some of the CAs like VeriSign.*

Before you begin

The authentication process by the CA can take a significant amount of time. Commercial CAs often require up to a week to complete their authentication process. Even on-site CAs can take several minutes, if not hours, or even days, to complete their authentication process. Therefore, you must plan for the certificates that you will need.

Considering the following points when you plan for the CA-signed certificate:

- On the certificate signing request that you send to the CA, you need to specify the common name for the certificate. This is the primary, universal identity for the certificate. It should uniquely identify the principal that it represents. Also, the common name should be valid in the configured user registry for the WebSphere Domain.
- Check the format of address fields that your CA requires when planning the address for a certificate request.

Steps for this task

1. Create and send a certificate signing request (CSR) to the CA.
2. Visit the CA web site, and follow the instructions to request a test or production certificate.

Results

Once the request is accepted, the Certificate Authority will verify your identity and finally issued a signed certificate to you. The certificate are usually be sent through e-mail.

Usage scenario

Request a production certificate from a trusted CA for the production WebSphere environment.

What to do next

Once you have received the E-mail from the CA, you need to follow the instructions sent to store your signed certificate as a file. Then, receive or store the certificate into the Keystore as a personal certificate.

Creating certificate signing requests (CSR): To obtain a certificate from a certificate authority, you must submit a certificate signing request (CSR) using the IKeyman tool. You can request either production or test certificates from a CA with a CSR.

With IKeyman, generating a certificate signing request also generates a private key for the application for which the certificate is being requested. The private key remains in the application keystore file, so it stays private. The public key is included in the certificate requested.

Before you begin

Read the file <production_installation_root>/web/docs/ikeyman/ikmuserguide.pdf for further information about how to create a certificate signing request from a key database file.

Steps for this task

1. Start IKeyman, if it is not already running.
2. Open the key database file from which you want to generate the request.
3. Type the password and click OK.
4. Click Create -> New Certificate Request. The Create New Key and Certificate Request window is displayed.
5. Type a Key Label, a Common Name and Organization, and select a Country. For the remaining fields, either accept the default values, or type or select new values. Note that the Common Name must be valid in the configured user registry for the secured WebSphere environment.
6. Type in a name for the file, such as certreq.arm.
7. Click OK to complete.
8. The file, certreq.arm, should then be sent to the CA following the instructions from the CA web site for requesting a new certificate.

Results

The Personal Certificate Requests list shows the key label of the new digital certificate request you just created. Send the file to a CA to request a new digital certificate, or cut and paste the request into the request forms of the CA Web site.

Usage scenario

You need to request a Certificate Authority signed digital certificate for your secure WebSphere domain.

What to do next

Once you have submitted the certificate signing request, you will then need to wait for the CA to accept the request. After the CA has verified your identity, it will send back the signed certificate back to you, usually via E-mail. You should then receive the signed certificate back to the keystore from which you generated the CSR.

Receiving CA-signed personal certificates: Once the CSR (Certificate Signing Request) request has been accepted, a Certificate Authority (CA) processes the request and will verify your identity. Once approved, the CA sends the signed certificate back through e-mail. The signed certificate should then be received or stored in a keystore database file. This procedure describes how to receive the CA-signed certificate into a keystore file using the iKeyman tool. It can be used for in the same way for both test certificates and production certificates. The primary difference between the two is the amount of time it takes for the CA to authenticate the principal your certificate represents. Test certificates are authenticated automatically based on some simple edit checks and should be returned to you in a matter of minutes or a few hours. Production certificates may take several days or a week to authenticate and return to you. If the CSR request is made for the cryptographic token, the certificate must be received into that token. If the request is made for the secondary key database of the token, the certificate must be received into that database.

Before you begin

You should receive the signed certificate from the CA via e-mail. Follow the instructions from the CA to store the certificate into a file. Read the file <production_installation_root>/web/docs/ikeyman/ikmuserguide.pdf for further information about how to receive a personal certificate into a key database file from the CA.

Steps for this task

1. Start IKeyman, if it is not already running.
2. Open the key database file from which you generated the request.
3. Type the password and click **OK**.
4. Select **Personal Certificates** from the pull-down list.
5. Click **Receive**.
6. Click **Data type** and select the data type of the new digital certificate, such as Base64-encoded ASCII data. You should select the data type that matches the CA-signed certificate. If the CA sends the certificate as part of an E-mail message, you may first need to cut and paste the certificate into a separate file.
7. Type the Certificate file name and Location for the new digital certificate, or click **Browse** to locate the CA-signed certificate.
8. Click **OK**.
9. Type a label for the new digital certificate and click **OK**.

Results

The Personal Certificate list should now display the label you just gave for the new CA-signed certificate.

Usage scenario

Needs digital certificate to support SSL for security over the WebSphere domain.

What to do next

Once the CA-signed certificate is successfully received, you can extract or export the public key of the certificate out to a file to prepare for distributing the public keys for the network.

Extracting public certificates for TrustStore files: Use this procedure to extract a public certificate (which includes its public key) from a keystore file. If target truststore already contains the signer certificate of the CA used to sign the certificate, you do not need to extract and add the certificate to the target truststore. However, in general, you need to complete this procedure for a self-signed certificate.

Before you begin

Extracting a certificate from one keystore and adding it to a truststore is not the same as exporting the certificate then importing it. Exporting a certificate copies all the certificate information, including its private key, and is normally only used if you want to copy a personal certificate into another keystore as a personal certificate.

If a certificate is self-signed, you need to extract the certificate (which includes its public key) from the keystore and add it into the target truststore.

If a certificate is CA-signed, ensure that the CA certificate used to sign the certificate is listed as a signer certificate in the target truststore file. The keystore file must already exist and contain the certificate to be extracted.

Read the file <production_installation_root>/web/docs/ikeyman/ikmuserguide.pdf for further information about how to extract a public certificate from a key database file.

Steps for this task

1. Start IKeyman, if it is not already running.
2. Open the keystore file from which the public certificate will be extracted.
3. Select **Personal Certificates** from the drop-down list.
4. Click **Extract Certificate**.
5. Select **Base64-encoded ASCII data** under Data type.
6. Enter the **Certificate File Name** and **Location**.
7. Click **OK** to export the public certificate into the specified file.

Results

A certificate file that contains the public key of the signed personal certificate, is now available to be added to the target truststore file.

Usage scenario

Prepare truststore files for distributing the public keys to support the secure WebSphere domain using SSL.

What to do next

Once the keystore and truststore files are ready, you need to make them accessible by specifying them into your client and server configurations.

Creating TrustStore files: Truststore file is a key database file that contains the public keys for target servers. The public key is stored as a signer certificate. If the target uses a self-signed certificate, you need to first extract the public certificate from the server keystore file. Then, add the extracted certificate into the truststore file as a signer certificate. For commercial CA, the CA root certificate is added. The truststore file can be a more publicly accessible key database file that contains all the trusted certificates.

Before you begin

It is recommended to read the documentation located at <production_installation_root>/web/docs/ikeyman/ikmuserguide.pdf. for further information.

Steps for this task

1. Start IKeyman, if it is not already running.
2. Open a new key database file by selecting **Key Database File** > **New** from the menu bar.
3. Select the Key Database Type: `JKS(Default)`, `PKCS12`, and `JCEK`.

   This is the *trust file format* (or the value of `com.ibm.ssl.trustStoreType` property in the `sas.client.props`) when you configure the SSL setting for your application.
4. Type in the file name and location. The full path of this key database file is used as the *trust file name* (or the value of `com.ibm.ssl.trustStore` property in the `sas.client.props`) when you configure the SSL setting for your application.
5. Click **OK** to continue.
6. Then, type in password to restrict access to the file. This password is used as the *trust file password* (or the value of `com.ibm.ssl.trustStorePassword` property in the **sas.client.props**) when you configure the SSL setting for your application.

   Do not set an expiration date on the password or save the password to a file. You must reset the password when it expires or protect the password file. This password is used only to release the information stored by iKeyman during run time.
7. Click **OK** to continue. The tool now displays all of the available default signer certificates. These are the public keys of the most common CAs. You can add, view or delete signer certificates from this screen.

Results

A new SSL truststore file is created.

Usage scenario

Prepare truststore files for a SSL connection.

What to do next

Specify the truststore in the configuration of WebSphere Application Server. A keystore file should also be created if it has not been done yet.

Importing signer certificates: A signer certificate is the trusted certificate entry usually in a truststore file. You can import a Certificate Authority (CA) root certificate from the CA or a public certificate from the self-signed personal certificate of the target into your truststore file, as a signer certificate.

Before you begin

It is recommended to read the documentation located at <production_installation_root>/web/docs/ikeyman/ikmuserguide.pdf. for further information.

Steps for this task

1. Start IKeyman, if it is not already running.
2. Open the truststore file.

   The Password Prompt window is displayed.
3. Type the password and click **OK**.

4. Select **Signer Certificates** from the pull-down list.
5. Click **Add**.
6. Click **Data type** and select a data type, such as Base64-encoded ASCII data. This data type must be the same as the data type of the importing certificate.
7. Type a Certificate file name and Location for the CA root digital certificate, or click **Browse** to select the name and location.
8. Click **OK**.
9. Type a label for the importing certificate.
10. Click **OK**.

Results

The Signer Certificates field now displays the label of the signer certificate you have just added.

Usage scenario

Receive a CA root certificate or the public key from your secure target.

Map certificates to users: Client-side certificates allow access to secured resources from Web or Java clients. A client presents an X.509-compliant digital certificate to perform mutual authentication with an single sockets layer-enabled server. The product security run time attempts to map the certificate to a known user in the associated LDAP directory. If the certificate is successfully mapped to a user, then the holder of the certificate is believed to be the user in the registry and is authorized as this user.

After the single sockets layer-enabled server gets the client certificate, there must be a way to map the certificate to a user. WebSphere Application Server supports two techniques for mapping certificates to entries in LDAP registries:

- By exact distinguished name
- By matching attributes in the certificate to attributes of LDAP entries

Steps for this task

1. Map by exact distinguished name (DN).

   This approach attempts to map the distinguished name (DN) associated with the Subject in the certificate to an entry in the LDAP directory. If the mapping is successful, the user is authenticated and is authorized according to the privileges granted to the identity in the LDAP directory.

   The mapping is case insensitive. For example, the following two DNs match on a case-insensitive comparison:

   ```
   "cn=Smith, ou=NewUnit, o=NewCompany, c=us"
   "cn=smith, ou=newunit, o=NewCompany, c=US"
   ```

   If a match is found, authentication succeeds, and if no match is found, authentication fails.
2. Map by filtering certificate attributes.

   This approach maps certificate attributes to attributes of entries in an LDAP directory. For example, you can specify that the common name (CN) attribute of the Subject field in the certificate is to be matched against the uid attribute of your LDAP entry. If the mapping is successful, the user is authenticated and is authorized according to the privileges granted to the identity in the LDAP directory.

If you are matching the Subject CN field in the certificate to the uid attribute of the LDAP entry, a certificate with the Subject DN ″`cn=Smith, ou=NewUnit, o=NewCompany, c=us`″ matches an LDAP user entry with uid=Smith.

To use this mapping technique, you must request Certificate Mapping and set up the certificate filter in the administrative console.

What to do next

This specification extracts the CN field from the Subject attribute in the certificate (`Smith`) and creates a filter (user ID = `Smith`) from it. The LDAP directory is searched for a user entry that matches the filter. If an entry matches the filter, authentication succeeds.

Note: The search and match of the LDAP directory are based in part on how your LDAP directory is configured.

Cryptographic token support

A cryptographic token is a hardware or software device with a built-in key store implementation. The cryptographic device is used to manage certificates stored on the cryptographic tokens (also know as smartcards).

Both cryptographic accelerators, where the cryptographic hardware device has no persistent storage of keys, and secure cryptographic hardware, where a cryptographic token generates and securely stores the private key used for SSL key exchange, are supported in the product.

The following token types are supported:

- PKCS#7
- PKCS#11
- PKCS#12
- MSCAPI (only on Windows platforms)

Cryptographic token support is limited to tested devices. These include support tested for SSL clients:

- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift
- IBM Security Kit Smartcard
- GemPlus Smartcards
- Rainbow iKey 1000/2000(USB ″Smartcard″ device)
- Eracom CSA800

Cryptographic token support has also been tested for the following SSL clients and servers:

- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift

WebSphere Application Server uses IBM JSSE to support cryptographic token device. Refer the document <*production_install_root*>\web\docs\jsse\readme.jsse.ibm.html for further information

Opening a cryptographic token using the IKeyman tool

Before you begin

You should verify that your cryptographic token device is installed and functions properly. A cryptographic token should have also been created following the instructions provided by the manual of the cryptographic device.

From your crypto device documentation, identify the token library. For example, IBM 4758 PCI Cryptographic Card uses CRYPTOKI.DLL as the PKCS#11-type token library (see `http://www.ibm.com/security/cryptocards/html/library.shtml` for details).

It is recommended that you read the documentation located at <*install_root*>/web/docs/ikeyman/ikmuserguide.pdf. for further information about using the IKeyman tool.

The IBM Key Management tool (IKeyman) can be used to open a cryptographic token. Once opened, you can manage your keys and certificates just like you do with KeyStore and TrustStore files:

- Create a self-signed digital certificate
- Add CA roots as a signer certificate
- Request and receive a digital certificate from a CA

Steps for this task

1. Start IKEYMAN, if it is not already running.
2. Select **Key DataBase File** > **Open**.
3. Choose **Cryptographic Token** from the Key database type list.
4. Fill in the information for **File Name** and **Location**, or browse for the cryptographic device library.
5. Click **OK** to open the library.
6. Type in the slot number in next panel. *This is the number of the slot in which you have previously created the cryptographic token.*
7. Enter the password. This is the password configured for the created cryptographic token.

Results

If any, you should see all the personal and signer certificates stored on the cryptographic token card. With the token opened, you can now create or request digital certificates and receive CA-signed certificates.

Usage scenario

Using a cryptographic token device as a key database to manage keys and certificates for SSL connection.

What to do next

Once the cryptographic token has been opened, keys and certificates can be added or deleted. After that, you are ready to configure the cryptographic token settings in WebSphere Application Server.

Configuring to use cryptographic tokens

You can configure cryptographic token support in both client and server configuration. To configure a Java client application, use the `sas.client.props` configuration file. By default, the `sas.client.props` is located in the properties directory under the <*install_root*> of your WebSphere Application Server installation. To configure a WebSphere Application Server, use the administrative console. To start the administrative console, specify URL: **`http://<`*`server_hostname`*`>:9090/admin`**.

Before you begin

To understand how to make WebSphere Application Server (both the runtime and the IKeyMan key management utility) work correctly with any cryptographic token device, you should become familiar with the JSSE documentation available from the Application Server product installation **<*install_root*>/web/docs/jsse/readme.jsse.ibm.html** and **<*install_root*>/web/docs/ikeyman/readme_jsse_ikeyman_ibm.htm**.

Be sure to unzip the file **<*install_root*>/web/docs/jsse/native-support.zip**, and copy the right libraries with respect to target operating system to the appropriate location; otherwise, link errors will occur at runtime, or the IBM IKeyman will not work properly with the crypto device library.

Follow the documentation that accompanies your device in order to install your crypto device. Installation instructions for IBM cryptographic hardware devices can be found in the Administration section of (Resources for learning).

Steps for this task

1. To configure a client to use cryptographic token, edit the sas.client.props and set the following properties. *You should leave the fields **KeyStore File Name** and **KeyStore File Password** in an SSL configuration blank, if you want to use only cryptographic token as your keystore.*
   - **com.ibm.ssl.tokenType**. Specifies the type of built-in key store that is implemented in the cryptographic token. (For example, com.ibm.ssl.tokenType=PKCS\#11). The valid values are: **PKCS\#7**, **PKCS\#11**, **PKCS\#12**, **MSCAPI**.
   - **com.ibm.ssl.tokenLibraryFile**. Specifies the token that is the file name for **PKCS#7**, **PKCS#12** tokens and the library name for **PKCS#11**, MSCAPI tokens. Make sure the cryptographic token device is installed and functions properly with a cryptographic token created. Also, you need to unzip the native-support.zip from `<`*`install_root`*`>/web/docs/jsse` directory to copy the required libraries with respect to the target operating system.
   - **com.ibm.ssl.tokenPassword**. Specifies the password to unlock the cryptographic token.
2. Configure your server to use cryptographic token. *You should leave the fields **KeyStore File Name** and **KeyStore File Password** in an SSL configuration blank, if you want to use only cryptographic token as your keystore.*
   a. Specify http://<server_hostname>:9090/admin to start the administrative console.
   b. Expand **Security** > **SSL** to open the SSL Configuration Repertoires panel. Create a new SSL setting alias if you have not done it yet. Otherwise, click on the alias that you want to configure for the cryptographic token.

c. Click **Cryptographic Token** in the section of Additional Properties to open the Cryptographic Token Settings panel. SSL Configuration Repertoires > (SSL alias) > Cryptographic Token.

d. Complete the information for **Token Type** to specify the type of built-in key store that is implemented in the cryptographic token. The valid values are: **PKCS#7**, **PKCS#11**, **PKCS#12**, **MSCAPI**.

e. Complete the information for Library File to specify the token that is the file name for PKCS#7, PKCS#12 tokens and the library name for PKCS#11, MSCAPI tokens. Make sure the cryptographic token device is installed and functions properly with a cryptographic token created. Also, you need to unzip the native-support.zip from `<`*`install_root`*`>/web/docs/jsse` directory to copy the required libraries with respect to the target operating system.

f. Complete the information for Password to specify the password to unlock the cryptographic token.

g. Click **Apply** and **OK** to go back to the SSL alias panel.

h. Check the box to enable **Cryptographic Token**.

i. Click **Apply**, **OK**, and **Save** to save the configuration.

Results

The configuration has been enabled to support the specified cryptographic token for SSL connection.

Usage scenario

WebSphere Application server uses the cryptographic token as a keystore for SSL connection

What to do next

If the server configuration has been changed, the configured server will need to be restarted.

Cryptographic token settings

Use this page to configure cryptographic token settings.

To view this administrative console page, click **Security** > **SSL** > *alias_name* > **Cryptographic Token**.

Token Type: Specifies the type of built-in key store that is implemented in the cryptographic token, such as PKCS#11.

The WebSphere Application Server uses IBM JSSE implementation to support cryptographic token with Secure Socket Layer (SSL). Different cryptographic devices are supported. For an SSL server, the following are supported:

- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift

For an SSL client, the the following are supported:

- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift

- IBM Security Kit Smartcard
- GemPlus Smartcards
- Rainbow iKey 1000/2000 (USB ″Smartcard″ device)
- Eracom CSA800

Follow the documentation that accompanies your device in order to install your cryptographic token.

Data type	String

Library File: Specifies the DLL or shared object that implements the interface to the cryptographic token device.

Data type	String

Password: Specifies the password for the cryptographic token device.

Data type	String

Using JSSE and JCE with Servlets and EJB components

Java Secure Socket Extension (JSSE)

Java Secure Socket Extension (JSSE) provides the transport security for WebSphere. It provides application programming interface (API) framework and the implementation of the APIs, for SSL/TLS protocols including functionality for data encryption, message integrity and authentication. The JSSE APIs were designed to allow other SSL/TLS protocol and Public Key Infrastructure (PKI) implementations to be plugged in.

IBMJSSE: A provider named IBMJSSE is used in WebSphere. The IBMJSSE provider comes pre-installed and pre-registered with the Java Cryptography Architecture (JCA) of the Java 2 platform, and supports the following cryptographic services:

- Rivest Shamir Adleman (RSA) public key cryptography support
- SSL and TLS security protocols and common cipher suites
- X.509-based key and trust managers
- PKCS12 as JCA keystore type

The IBMJSSE provider is pre-registered in the java.security security properties file located at <install_root>/java/jre/lib/security directory. It also supports cryptographic token of type: PKCS#7, PKCS#11, PKCS#12 and MSCAPI (only on Windows platforms) for cryptographic token support.

Customizing JSSE: It is possible to customize a number of aspects of JSSE by plugging in different implementations of Cryptography Package Provider, X509Certificate and HTTPS protocol, or specifying different default keystore files, key manager factories and trust manager factories. A table is provided to summarizes which aspects can be customized, what the defaults are, and which mechanisms are used to provide customization. You can find the table, by searching ″JSSE Customization″, at <install_root>/web/docs/jsse/JSSERefGuide.html.

Note: There is no guarantee that the properties shown in the table continue to have the same names and types or even that they will exist in the future releases.

Some of the key customizable aspects are listed below:

Customizable Item	Default	How To Customize
X509Certificate	X509Certificate implementation from IBM	cert.provider.x509v1 security property
HTTPS protocol	Implementation from IBM	java.protocol.handler.pkgs system property
Cryptography Package Provider	IBMJSSE	A security.provider.n= line in security properties file. See description.
Default keystore	None	* javax.net.ssl.keyStore system property
Default truststore	jssecacerts, if it exists. Otherwise, cacerts	* javax.net.ssl.trustStore system property
Default key manager factory	″IbmX509″	ssl.KeyManagerFactory.algorithm security property
Default trust manager factory	″IbmX509″	ssl.TrustManagerFactory.algorithm security property

For aspects that can be customized by setting a system property, you can statically set the system property by using the -D option of the Java command (Note, for WebSphere Application Server, you can set the system property using the admin console), or set the system property dynamically by calling the java.lang.System.setProperty method in your code: **System.setProperty(propertyName,″propertyValue″)**.

For aspects that can be customized by setting a Java security property, you can statically specify a security property value in the java.security security properties file located at <install_root>/java/jre/lib/security directory. The security property should be in a form of: propertyName=propertyValue. You can also dynamically set the Java security property by calling the java.security.Security.setProperty method in your code.

There are more information in JSSERefGuide.html for JSSE customization. Make sure you understand the implication to your application before you starts customizing.

Application Programming Interface (API): The JSSE provides standard API, available in packages of the javax.net, javax.net.ssl, and javax.security.cert. The APIs cover:

- Sockets and SSL sockets
- Factories to create the sockets and SSL sockets
- Secure socket context that acts as a factory for secure socket factories
- Key and trust manager interfaces
- Secure HTTP UTL connection class
- Public key certificate API

There are more information documented for the JSSE apis in jssedocs.jar located at <install_root>/web/docs/jsse directory. Unzip the jar file and open index.html with your browser.

Samples Using JSSE: JSSE also provides samples to demonstrate its functionality. The samples are included in <install_root>/web/docs/jsse/samplejsse.jar. Unzip the file and you will find the followings:

Files	Description
ClientJsse.java	To demonstrate a simple client/server interaction using JSSE. All enabled cipher suites will be used.
ClientJsseProvider.java	To demonstrate a simple client/server interaction using JSSE. All enabled cipher suites will be used.

`ServerJsse.java`
`ServerJsseProvider.java`
`OldClientJsse.java`
	To demonstrate a simple client/server interaction using JSSE. All enabled cipher suites will be used.

OldServerJsse.java	Back-level samples

File	Description
ServerPKCS12Jsse.java	To demonstrate a simple client/server interaction using JSSE with PKCS12 keystore type. All enabled cipher suites will be used.
ClientPKCS12Jsse.java	To demonstrate a simple client/server interaction using JSSE with PKCS12 keystore type. All enabled cipher suites will be used.
OldClientPKCS12Jsse.java	Back-level samples
OldServerPKCS12Jsse.java	Back-level samples

File	Description
UseHttps.java	To demonstrate accessing an SSL or non-SSL webserver using the Java protocol handler of the com.ibm.net.ssl.www.protocol class. The URL is specified with the http: or https: prefix. The HTML returned from this site is displayed.
HTTPTest.java	To demonstrate accessing an SSL or non-SSL webserver using the Java protocol handler of the com.ibm.net.ssl.www.protocol class. The URL is specified with the http: or https: prefix. The HTML returned from this site is displayed.
`HTTPSPanel.java` `OldHTTPTest.java`	Back-level sample

More instructions can be found in the source code. Make sure you follow the instuctions before you run the samples.

Permissions for Java 2 Security: The following permissions may be needed to run an application with JSSE (This is a list for reference only.):

- java.util.PropertyPermission ″java.protocol.handler.pkgs″, ″write″
- java.lang.RuntimePermission ″writeFileDescriptor″
- java.lang.RuntimePermission ″readFileDescriptor″
- java.lang.RuntimePermission ″accessClassInPackage.sun.security.x509″
- java.io.FilePermission ″${user.install.root}${/}etc${/}.keystore″, ″read″
- java.io.FilePermission ″${user.install.root}${/}etc${/}.truststore″, ″read″

For IBMJSSE provider:

- java.security.SecurityPermission ″putProviderProperty.IBMJSSE″
- java.security.SecurityPermission ″insertProvider.IBMJSSE″

For SUNJSSE provider:
- java.security.SecurityPermission ″putProviderProperty.SunJSSE″
- java.security.SecurityPermission ″insertProvider.SunJSSE″

Debugging: By configuring via the javax.net.debug system property, JSSE provides dynamic debug tracing: `-Djavax.net.debug=true`.

A value of **true** will turn on the trace facility provided that the debug version of JSSE has been installed (uses admin console to set the system property for debugging the application server.)

The debug version of JSSE, ibmjsse-debug.jar, is located at <install_root>/web/docs/jsse directory. Do the followings to collect the trace:
1. Stop your application.
2. Save the default version of JSSE, ibmjsse.jar, into a separate directory.
3. Locate ibmjsse.jar at <install_root>/java/jre/lib/ext directory.
4. Replace ibmjsse.jar with ibmjsse-debug.jar in /java/jre/lib/ext directory.
5. Specify the javax.net.debug system property.
6. Restart your application

The trace will be logged in the trace.log file for the application server or in the file specified by a system property -DtraceFileName for java client application.

Documentation: See the (Security: Resources for learning) article for documentation references to JSSE.

JCE: Java Cryptography Extension (JCE) provides cryptographic, key, and hash algorithms for WebSphere. It provides a framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block, and stream ciphers.

IBMJCE: IBMJCE is an implementation of JCE cryptographic service provider used in WebSphere. The IBMJCE is similar to SunJCE, except that the IBMJCE offers more algorithms in the following:
- Cipher algorithm
- Signature algorithm
- Message digest algorithm
- Message authentication code (MAC)
- Key agreement algorithm
- Random number generation algorithm
- Key Store

The IBMJCE also moved com.sun.crypto.provider.* packages to com.ibm.crypto.provider* packages. Please read <install_root>/web/docs/jce/readme.jce.ibm.html for further details.

Implementing a JCE Cryptographic Service Provider: A Cryptographic Service Provider, or ″provider″ for short, refers to a package (or a set of packages) that supply a concrete implementation of a subset of the cryptography aspects of the Java Security API. A provider could contain an implementation of one or more digital signature algorithms and one or more cipher algorithms. Steps to implement and integrate a JCE provider are listed as follows for reference:

Steps for this task

1. Write your service implementation code.
2. Give your provider a name.
3. Write your ″Master Class,″ a subclass of provider.
4. Compile your code.
5. Prepare for testing.
6. Write and compile test programs.
7. Run your test programs.
8. Document your provider and its supported services.
9. Prepare for production.
10. Run your test programs again.
11. Make Your provider software and documentation available to clients.

What to do next

Refer to <install_root>/web/docs/jce/HowToImplAProvider.html for further information.

Application Programming Interface (API): JCE has a provider-based architecture. Providers can be plugged into the JCE framework by implementing the APIs defined by JCE. The JCE APIs covers:

- Symmetric bulk encryption, such as DES, RC2, and IDEA
- Symmetric stream encryption, such as RC4
- Asymmetric encryption, such as RSA
- Password-based encryption (PBE)
- Key Agreement
- Message Authentication Codes (MAC)

Refer to <install_root>/web/docs/jce/api_users_guide.html and <install_root>/web/docs/jce/CryptoSpec.html for more information about Java Cryptography Extension technology. Javadoc for the JCE APIs can also be found by unzipping <install_root>/web/docs/jce/jcedocs.jar.

Samples Using JCE: There are samples provided in SampleJCE.jar located at <install_root>/web/docs/jce directory. Unzip the file and the following source code will be found:

File	Description
SampleDSASignature.java	To demonstrate how to generate a pair of DSA key (a public key and a private key) and use the key to digitally sign a message using the SHA1withDSA algorithm.
SampleMarsCrypto.java	To demonstrate how to generate a Mars secret key, and how to do Mars encryption and decryption.
SampleMessageDigests.java	To demonstrate how to use the message digest for MD2 and MD5 algorithms.

SampleRSACrypto.java	To demonstrate how to generate an RSA key pair, and how to do RSA encryption and decryption.
SampleRSASignatures.java	To demonstrate how to generate a pair of RSA key (a public key and a private key) and use the key to digitally sign a message using the SHA1withRSA algorithm.
SampleX509Verification.java	To demonstrate how to verify X509 Certificates

Documentation: Refer to the (Security: Resources for learning article) for documentation on JCE.

Java 2 Security

Java 2 Security is a supported feature in 5.0. Java 2 Security provides a policy-based, fine-grain access control mechanism that increases overall system integrity by checking for permissions before allowing access to certain protected system resources. Java 2 Security is orthogonal to J2EE role-based authorization. Java 2 Security guards access to system resources such as file I/O, sockets, properties whereas J2EE security guards access to web resources such as servlets/JSPs and EJB methods. WebSphere Global Security includes J2EE role-based authorization, the CSIv2 authentication protocol, and SSL configuration. Java 2 Security can be disabled and enabled independent of WebSphere Global Security (the Enable check box in Global Security Panel or Server Level Security Panel). However, when WebSphere Global Security is enabled, by default Java 2 Security is also enabled. Note that Java 2 security can be disabled even though WebSphere Global Security may be enabled.

Since Java 2 Security is relatively new to the industry and new to WebSphere, many existing or even new applications may not be prepared for the very fine grain access control programming model that Java 2 Security is capable of enforcing. Administrators should understand the possible consequences of enabling Java 2 Security if applications are not prepared for Java 2 Security. Java 2 Security places some new requirements on application developers and administrators.

Java 2 Security for Deployers and Administrators

There are implication if Java 2 Security is enabled, deployers or administrators are required to make sure that all the applications are granted the required permissions, otherwise, applications may fail to run. By default, applications are granted the permissions recommended in the J2EE 1.3 Specification. For details of default permissions granted to applications in WebSphere, please refer to the following policy files:

- WAS_HOME/java/jre/lib/security/java.policy
- WAS_HOME/properties/server.policy
- WAS_HOME/config/cells/<*cell name*>/nodes/<*node name*>/app.policy

Note: This policy embodied by these policy files cannot be made more restrictive because the WebSphere may not have the necessary Java 2 Security doPrivileged APIs in place. The result of attempting to make the default policy more restrictive may result in WebSphere throwing Java 2 security access control exceptions and causing applications or WebSphere itself to fail unexpectedly. WebSphere does not support a more restrictive policy than the default one embodied in the policy files mentioned above.

There are several policy files is used to define the security policy for Java process. These policy files are static (code base is defined in the policy file) and they are in the default policy format provided by the JDK. For enterprise applications resources and utility libraries, WebSphere provides ″dynamic policy″ support, the code base is dynamically calculated based on deployment information and permissions are granted based on templates policy files during runtime. Please refer to the section of Java 2 Security Policy Management.

Note: Syntax errors in the policy files will cause the application server process fail to start. Extreme care should be taken when editing these policy files. It is advised to use the policytool provided by the JDK for editing the policy files (WAS_HOME/java/jre/bin/policytool).

If an application is not prepared for Java 2 Security, if the application provider does not provide a was.policy file as part of the application, or if the application provider doesn't communicate the expected permissions that should be granted to one of the WebSphere policy files, the application is very likely to cause Java 2 security access control exceptions at runtime if Java 2 Security is enabled in WebSphere and probably will not run correctly. It may not be obvious that a application is or is not prepared for Java 2 security. WebSphere provides several runtime debugging aids to help determine troubleshoot applications that may be suffering Java 2 Security related access control exceptions. See the Java 2 Security Debugging Aids section below for more details. See the Handling Applications That Are Not Java 2 Security Ready section for information and strategies for dealing with such applications.

Java 2 Security for Application Developers

Application developers must understand the permission granted in the default WebSphere policy and the permission requirements of the Java SDK APIs that their application calls to know whether or not additional permissions need to be granted. The ″Permissions in the Java(TM) 2 SDK″ reference in the resources section describes which Java SDK APIs require which permission.

Application providers can assume that applications have the permissions granted in the default policy mentioned above. Applications that access resources not covered by the default WebSphere policy are required to grant the additional Java 2 security permissions to the application.

While it is possible to grant the application additional permissions in one of the other dynamic WebSphere policy files or in one of the more traditional static policy files, such as java.policy, the was.policy (which is embedded in the EAR file) ensures the additional permissions are scoped to the exact application that requires them. Scoping the permission beyond the application code that requires it may permit code that normally doesn't have permission to access resources that it would not otherwise.

If a component of an application is being developed, like a library, that may actually be included in more than one .ear file, then the library developer should document the required Java 2 permissions needed by the application assembler. There is no was.policy like file for library type components, the developer must communicate the required permissions via Java Doc or some other external documentation.

If the component library is shared by multiple enterprise applications, the permissions can be granted to all enterprise applications on the node in the app.policy.

If the permission is only used internal by the component library and the application should never be granted the access to resources protected by the permission, then it might be necessary to mark the code as ″privileged″ (inserting doPrivileged). Please refer to the article AccessControlException for more details. However, **improperly inserting a doPrivileged could open up security holes**, so great care and understanding the implication of doPrivileged is required to make a correct judgement whether a doPrivileged should be inserted or not.

The article, Java 2 security policy files describes how the permissions in was.policy files are granted at run time.

Developing an application with Java 2 Security in mind may be a new skill and impose a security awareness not previously required of application developers. Describing the Java 2 Security model and the implications on application development is beyond the scope of this section. The following URL can help you get started: (http://java.sun.com/j2se/1.3/docs/guide/security/index.html).

Debugging Aids

There are two primary aids, the WebSphere SystemOut.log file and the com.ibm.websphere.java2secman.norethrow property.

The WebSphere SystemOut.log File

The AccessControl exception logged in the SystemOut.log will contain the permission violation that cause the exception, the exception call stack, and the permissions granted to each stack frame. This information is usually enough to determine the missing permission and the code requiring the permission.

The com.ibm.websphere.java2secman.norethrow Property

When Java 2 Security is enabled in WebSphere, the Security Manager component will by default throw an java.security.AccessControl exception when a permission violation occurs. This exception, if not handled, will often cause a runtime failure. This exception is also logged in the SystemOut.log.

However, when the JVM com.ibm.websphere.java2secman.norethrow property is set and has a value of true, the Security Manager will not throw the AccessControl exception — it will only be logged.

Note: This property is intended for a sandbox or debug environment only since it instructs the Security Manager not to throw the AccessControl exception. By not rethrowing the exception, Java 2 security is not actually enforced. This property should not be used in a production environment where a relaxed Java 2 security environment will weaken the very integrity Java 2 security is intended to produce.

This property is valuable in a sandbox or test environment where the application can be thoroughly tested and the SystemOut.log can be inspected for AccessControl exceptions. Since this property causes the AccessControl exception not to be thrown, it will not be propagated up the call stack and will not cause a failure. Without this property, AccessControl exceptions would have to found and fixed one at a time.

Handling Applications That Are Not Java 2 Security Ready

If the increased system integrity that Java 2 Security provides is that important, then contact the application provider to have the application support Java 2 Security or at least communicate the required additional permissions beyond the default WebSphere policy that must be granted.

The easiest way to deal with such applications is to disable Java 2 Security in WebSphere. The downside is that this solution applies to the entire system and the integrity of the system is not as strong as it could be — this should be a consideration not taken lightly depending on the environment or policy of the organization. Depending on the organization though, disabling Java 2 Security is an acceptable risk.

Another approach would be to leave Java 2 Security enabled but grant either just enough additional permissions or all permissions to just the problematic application. Granting ″just enough″ permissions however, may not be a trivial thing to do. If the application provider has not communicated the required permissions in some way, then there is no easy way to determine what the require permissions are and granting all permissions may be the only choice. You minimize this risk by locating such an application on a different node which may help isolate it from certain resources. This can be done granting java.security.AllPermission in the was.policy embedded in application's .ear file, for example:

```
grant codeBase "file:${application}" {
            permission java.security.AllPermission;
        };
```

WAS_HOME/properties/server.policy

This policy defines the policy for the WebSphere classes. At present, all the server processes on the same install shared the same server.policy file. However, it can be configured that each server process can have their own server.policy file. This can be configured by defining the desired policy file as the value of the Java system properties java.security.policy. For details of how to define Java system properties, please refer to the Process Definition section of the Manage Application Servers.

The `server.policy` is not a configuration file managed the by repository and the file replication service. Changes to this file are local and do not get replicated to other machine. The `server.policy` file should be used to defined Java 2 security policy for server resources, please use app.policy (per node) or was.policy (per enterprise application) to define Java 2 security policy for enterprise applications resources.

WAS_HOME/java/jre/lib/security/java.policy

The file represents the default permissions granted to all classes. The policy of this file applies to all the processes launched by the WebSphere Application Server JVM.

Troubleshooting

Symptom:	`Error message SECJ0314E`: Current Java 2 Security policy reported a potential violation of Java 2 Security Permission. Please refer to Problem Determination Guide for further information.{0}Permission\:{1}Code\:{2}{3}Stack Trace\:{4}Code Base Location\:{5} Current Java 2 Security policy reported a potential violation of Java 2 Security Permission. Please refer to Problem Determination Guide for further information.{0}Permission\:{1}Code\:{2}{3}Stack Trace\:{4}Code Base Location\:{5}
Problem:	The Java Security Manager checkPermission() reported a SecurityException on the subject Permission with debugging information. The reported information can be different with respect to the system configuration. This report is enabled by either configuring RAS trace into debug mode or specifying a Java property. Please check the trace enabling section from WebSphere InfoCenter about how to configure RAS trace into debug mode. Specify the following property in the JVM Settings panel from the admin console: **java.security.debug**. Valid values include: • access — to print all debug information including: required permission, code, stack, and code base location • stack — to print debug information including: required permission, code, and stack • failure — to print debug information including: required permission and code

Recommended response:	The reported exception may be critical to the secure system. Turn on security trace to determine the potential code that may have violated the security policy. Once the violating code is determined, user should verify if the attempted operation is permitted with respect to Java 2 Security, by examining all applicable Java 2 security policy files and the application code itself.
	Note: If the application is running with Java Mail, this message may be benign. User can update was.policy to grant the following permissions to the application.
	(Lines in the following example have been split for publication.)
	`permission java.io.FilePermission`
	`"${user.home}${/}.mailcap", "read";`
	`permission java.io.FilePermission`
	`"${user.home}${/}.mime.types", "read";`
	`permission java.io.FilePermission`
	`"${java.home}${/}lib${/}mailcap", "read";`
	`permission java.io.FilePermission`
	`"${java.home}${/}lib${/}mime.types", "read";`

Messages

Message:	SECJ0313E: Java 2 Security Manager debug message flags are initialized\: TrDebug: {0}, Access: {1}, Stack: {2}, Failure: {3}
Problem:	Configured values of the valid debug message flags for Security Manager.
Recommended response:	None.

Message:	SECJ0307E: Unexpected exception is caught when trying to determine the code base location. Exception: {0}
Problem:	An unexpected exception is caught when the code base location is being determined.
Recommended response:	Please contact an IBM representative.

AccessControlException

The Java 2 Security behavior is specified by its security policy. The security policy is an access-control matrix that specifies which system resources certain code bases can access and who must sign them. The Java 2 Security policy is declarative and it is enforced by the *java.security.AccessController.checkPermission()* method.

The following is the algorithm for the *java.security.AccessController.checkPermission()* method. For the full algorithm, refer to the Java 2 Security check permission algorithm in (Resources for learning) for more details:

```
i = m;
while (i > 0) {
 if (caller i's domain does not have the permission)
  throw AccessControlException;
 else if (caller i is marked as privileged)
  return;
 i = i - 1;
};
```

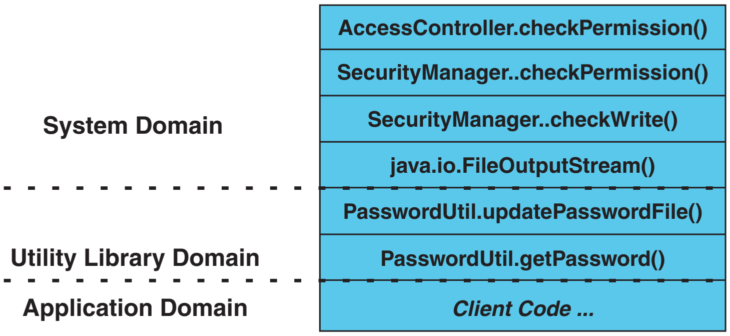
The algorithm requires that all the classes (callers) on the call stack be granted the said permissions when a *java.security.AccessController.checkPermission()* is performed, or the request is denied (a *java.security.AccessControlException* is thrown). However, if the caller is marked as *privileged* and the class (caller) is granted the said permissions, the algorithm returns at that point and does not walk the entire call stack. Subsequent classes (callers) do not need to be granted the required permission.

A *java.security.AccessControlException* exception is thrown as a result of certain classes on the call stack missing the required permissions during a *java.security.AccessController.checkPermission()* method. The following are the two possible resolutions to the *java.security.AccessControlException* exception:

- If the application is calling a Java 2 Security protected API, then grant the required permission to the application Java 2 Security policy. If the application is NOT calling a Java 2 Security protected API directly and the required permission is because of the side-effect of the third party APIs accessing Java 2 Security protected resources.
- If the application is granted the required permission, it gains more access than it should. In this case, it is likely that the third party code that accesses the Java 2 Security protected resource is not properly mark as *privileged*.

Example call stack

This is an example of a call stack where an application code is using a third party API utility library to update the password. *The following is only an example to illustrate the point. It is not the ultimate guide of where to mark the code as privileged.* The decision as to where is the appropriate place to mark the code as *privileged* is application specific and is unique in every situation, requiring great depth of domain knowledge and security expertise to make the correct judgement. There are a number of well written publications and books on this topic, it is highly recommended that you reference these materials for more detailed information.


AccessController.checkPermission()
SecurityManager..checkPermission()
System Domain
SecurityManager..checkWrite()
java.io.FileOutputStream()
PasswordUtil.updatePasswordFile()
Utility Library Domain
PasswordUtil.getPassword()
Application Domain
Client Code ...

You can use the `PasswordUtil` utility to change the password of a user. The types in the old password and the new password twice to ensure that the correct password is entered. If the old password matches the one stored in the password file, the new password is stored and updates the password file. Lets assume that none of the stack frame is marked as *privileged*. According to the `java.security.AccessController.checkPermission()` algorithm, the application fails unless all the classes on the call stack are granted *write* permission to the password file. The client application should not be granted the permission to write to the password file directly and update the password file at will.

However, if the `PasswordUtil.updatePasswordFile()` method marks the code that accesses the password file as *privileged*, then the check permission algorithm does not check for the required permission from classes that call the `PasswordUtil.updatePasswordFile()` method for the required permission as long as the `PasswordUtil` class is granted the permission. Then the client application can successfully update a password without granting the permission to write to the password file.

The ability to mark code *privileged* is very flexible and powerful, yet it is a double edged sword. If this ability is not used correctly, the overall security of the system can be compromised and security holes will be exposed. The ability to mark code `privileged` must be used with extreme care.

Note: Domain knowledge and security expertise is required to decide where to mark the code as *privileged*. A security exposure can result from code that is incorrectly marked.

Resolution to java.security.AccessControlException

As described previously, there are two possibilities to resolve a `java.security.AccessControlException` exception. Judge these case by case to decide which of the following is the best resolution to the problem:

1. Grant the missing permission to the application.

2. Mark some code as *privileged* (keeping in mind concerns and risk of doing so).

Configuring Java 2 Security

Before you begin

Java 2 Security is a new feature in WebSphere Application Server Version 5. It is a new programming model that is very pervasive and has a huge impact on application development. It is disabled by default, but is enabled automatically when Global Security is enabled. However, Java 2 Security is orthogonal to J2EE role-based security, you can disable or enable it independently of Global Security.

However, it does provide an extra level of access control protection on top of the J2EE role-based authorization. It particularly addresses the protection of system resources and APIs. The Java Development Kit used Java 2 Security to protect sensitive APIs (like the API to exit the Java Virtual Machine) and APIs that utilize resources (for example, open file for reading and writing) since the release of Java 2 Software Development Kit. The administrators should weigh the benefits against the risks of disabling Java 2 Security.

The following is a list of recommendations to ease the effort of enabling Java 2 Security in a test or production environment:

1. Make sure the application is developed with the Java 2 Security programming model in mind. Developers have to know whether or not the APIs used in the applications are protected by Java 2 Security. It is very important that the required permissions for the APIs used are declared in the policy file (*was.policy*), or the application fails to run when Java 2 Security is enabled. Developers can reference the Web site for Java Development Kit APIs that are protected by Java 2 Security. See the Programming model and decisions section of the (Security: Resources for Learning) article to visit this Web site.
2. Make sure that migrated applications from previous releases are given the required permissions. Since Java 2 Security is not supported or partially supported in previous WebSphere Application Server releases, applications developed prior to Version 5 most likely are not using the Java 2 Security programming model. There is no easy way to find out all the required permissions for the application. Following are activities you can perform to determine the extra permissions required by an application:
   a. Code review and code inspection
   b. Application documentation review
   c. Sandbox testing of migrated enterprise applications with Java 2 Security enabled in a pre-production environment. Enable tracing in WebSphere Java 2 Security Manager to help determine the missing permissions in the application policy file. The trace specification is `com.ibm.ws.security.core.SecurityManager=all=enabled`.
   d. Use the `com.ibm.websphere.java2secman.norethrow` system property to aid debuggging. This property should not be used in a production environment. Refer to (Java 2 Security).

Note: The default permission set for applications is the recommended permission set defined in the J2EE 1.3 Specification. The default is declared in the *config/cells/<cellname>/nodes/<nodename>/app.policy* policy file with permissions defined in the Java Development Kit (*${JAVA_HOME}/lib/security/java.policy*) policy file that grant permissions to everyone. However, applications are denied permissions declared in *config/cells/<cellname>/filter.policy* filter policy file. Permissions declared in the *filter.policy* file are filtered out for applications during the permission check.

Note: Define the required permissions for an application in a *was.policy* file and embed the *was.policy* file in the application EAR file as *YOURAPP.ear/META-INF/was.policy* (see Configuring dynamic policy for details).

Steps for this task

1. Click **Security** in the navigation tree, then click **Global Security**. The **Global Security** page appears.
2. Enable Java 2 Security by selecting the check box labeled as **Enforce Java 2 Security** (clear the check box for disabling Java 2 Security).
3. Click **OK** or **Apply** on the **Global Security** page.
4. Click **Save** to save the changes.
5. Restart the server for the changes to take effect.

Results

Java 2 Security is enabled and enforced for the servers. Java 2 Security permission is checked when a Java 2 Security protected API is called.

When to use Java 2 Security.

1. To enable protection on system resources. For example, when opening or listening to a socket connection, reading or writing to operating system file systems, reading or writing Java Virtual Machine system properties, and so on.
2. To prevent application code calling destructive APIs. For example, calling *System.exit()* brings down the application server.
3. To prevent application code obtaining privileged information (passwords) or gaining extra privileges (obtaining Server Credentials).

What to do next

The WebSphere Java 2 Security Manager is enhanced to dump the Java 2 Security permissions granted to all classes on the call stack when an application is denied access to a resource (the `java.security.AccessControlException` exception is thrown). However, this tracing capability is disabled by default. You can enable it by specifying the server trace service with the `com.ibm.ws.security.core.SecurityManager=all=enabled` trace specification. When the exception is thrown, the trace dump provides hints to determine whether the application is missing permissions or the product run time code or third party libraries used are not properly marked as *privileged* when accessing Java 2 protected resources. See the Security Problem Determination Guide for details.

Enable or disable Java 2 Security for the cell

1. Click **Security > Global Security** in the navigation tree. The Global Security page appears.
2. Enable Java 2 Security by selecting the check box labeled **Enforce Java 2 Security** (clear the check box to disable Java 2 Security). This enables Java 2 Security for the cell.
3. Click **OK** or **Apply** on the Global Security page.
4. Save the changes and make sure a file sync is performed before restarting the servers.

5. For the changes to take effect, restart all the servers, which include the Network Deployment Manager, all Node Agents, and all application servers.

Enable or disable Java 2 Security for an application server

1. Click **Server > Application Servers** in the navigation tree. The Application Servers page appears.
2. Click the **application server name** in the **Name** column of the Application Server collection table. The configuration panel of the application server selected appears.
3. Click **Server Security** in the Additional Properties section. The Server Security panel of the application server appears.
4. Click **Server Level Security** in the Additional Properties section. The Server Level Security panel of the application server appears.
5. Enable Java 2 Security by selecting the check box labeled **Enforce Java 2 Security** (clear the check box to disable Java 2 Security). This enables Java 2 Security for the selected application server.
6. Click **OK** or **Apply** on the Server Level Security page.
7. Save the changes and make sure a file sync is performed before restarting the application server.
8. Restart the application server for the changes to take effect.

Results

Java 2 Security is enabled and enforced for the servers. Java 2 Security permission is checked when a Java 2 Security protected API is called.

When to use Java 2 Security

1. To enable protection on system resources. For example, when opening or listening to a socket connection, reading or writing to operating system file systems, reading or writing Java Virtual Machine system properties, and so on.
2. To prevent application code calling destructive APIs. For example, calling *System.exit()* brings down the application server.
3. To prevent application code obtaining privileged information (passwords) or gaining extra privileges (obtaining Server Credentials).

What to do next

The WebSphere Java 2 Security Manager is enhanced to dump the Java 2 Security permissions granted to all classes on the call stack when an application is denied access to a resource (the `java.security.AccessControlException` exception is thrown). The trace information is dumped to the configured server log files. Check the server log files to access debugging information when an AccessControlException is thrown. In addition, the product Java 2 Security Manager trace can be enabled with the trace string, `com.ibm.ws.security.core.SecurityManager=all=enabled`. When the exception is thrown, the trace dump provides hints to determine whether the application is missing permissions or the product run time code or third party libraries used are not properly marked as *privileged* when accessing Java 2 protected resources. See the Security Problem Determination Guide for details.

Using PolicyTool to edit policy files

Java 2 Security uses several policy files to determine the granted permission for each Java program. See Dynamic Policy for the list of available policy files supported by Version 5 and their details. The Java Development Kit provides *policytool* to edit these policy files. It is recommended to always use this tool to edit any policy file, to guarantee the syntax of its contents. Syntax errors in the policy file cause an *AccessControlException* during application execution, including the server start. Identifying the cause of an AccessControlException is not an easy task, since the user might not be familiar with the resource that has an access violation. Extreme care should be taken when editing these policy files.

Steps for this task

1. Start policytool.
   a. Enter `%{`*`was.install.root`*`}/java/jre/bin/policytool` from a command prompt.

      The PolicyTool window opens. PolicyTool looks for the *.java.policy* file in your home directory. If it does not exist, an Error message displays. Click **OK**.
2. Click **File** > **Open**.
3. Navigate the directory tree in the **Open** window to pick up the policy file that you need to update. After selecting the policy file, click **Open**. The code base entries are listed in the window.
4. Create or modify the code base entry.
   a. Modify the existing code base entry by double clicking the code base, or click the code base and click **Edit Policy Entry**. The Policy Entry window opens with the permission list defined for the selected code base.

   b. Create a new code base entry by clicking **Add Policy Entry**. The Policy Entry window opens. At the code base column, enter the code base information as a URL format, for example, `/WebSphere/AppServer/InstalledApps/testcase.ear`.
5. Modify or add the permission specification
   a. Modify the permission specification by double clicking the entry you want to modify, or by selecting the permission and clicking **Edit Permission**. The Permissions window is opens with the selected permission information.

   b. Add a new Permission by clicking **Add Permission**. The Permissions window opens.
6. In the Permissions window there are four rows for *Permission*, *Target Name*, *Actions* and *Signed By*. Select the permission from the Permission list. The selected permission displays. After a permission is selected, the Target Name, Actions, and Signed By fields automatically show the valid choices, or they enable text input in the right text input area.
   a. Select **Target Name** from the list, or enter the target name in the right text input area.

   b. Select **Actions** from the list.

   c. Input **Signed By** if it is needed.
7. Click **OK** to close the Permissions window.

   Modified permission entries of the specified code base display.
8. Click **Done** to close the window. Modified code base entries are listed. Repeat steps 4 through 8 until you complete editing.
9. Click **File** > **Save** after you finish editing the file.

Results

A policy file is updated. If any policy files need editing, use the policytool. Do not edit the policy file manually. Syntax errors in the policy files could potentially cause application servers or enterprise applications to not start or function incorrectly. For the changes in the updated policy file to take effect, restart the Java processes.

Java 2 security policy files: The J2EE 1.3 specification has a well defined programming model of responsibilities between the container providers and the application code. Using Java 2 Security manager to help enforce this programming model is recommended. There are certain operations not allowed in the application code because such operations interfere with the behavior and operation of the containers. The Java 2 Security manager is used in the product to enforce responsibilities of the container and the application code.

This product provides support for policy file management. There are a number of policy files in the product, which are either static or dynamic. Dynamic policy is a template of permissions for a particular type of resource. There is no relative codeBase defined in the dynamic policy template. The codeBase is dynamically calculated from the deployment and runtime data.

Static policy files

Policy file	Location
java.policy	<WAS_HOME>/java/jre/lib/security/java.policy. Default permissions granted to all classes. The policy of this file applies to all the process launched by the WebSphere Application Server.
server.policy	<WAS_HOME>/properties/server.policy. Default permissions granted to all the product servers.
client.policy	<WAS_HOME>/properties/client.policy. This is the default permissions for all of the product client containers and applets on a node.

Each of the static policy files is not managed by configuration and file replication services. Changes made in these files are local and are not replicated to other nodes in the Network Deployment cell.

Dynamic policy files

spi.policy

(Lines in the following example have been split for publication.)

```
<WAS_HOME>/config/cells/
<cellname>/nodes/<nodename>/spi.policy
```

. This template is for the Service Provider Interface (SPIs) or third-party resources embedded in the product. Examples of SPI are Java Messaging Service (JMS) (MQSeries) and JDBC drivers. The codeBase for the embedded resources are dynamically worked out from the configuration **(resources.xml** file) and runtime data, and permissions defined in the `spi.policy` files are automatically applied to these resources. The default permission of `spi.policy` is `java.security.AllPermissions`.

library.policy

(Lines in the following example have been split for publication.)

```
<WAS_HOME>/config/cells/
<cellname>/nodes/<nodename>/library.policy
```

. This template is for the library (Java library classes). You can define a shared library for use in multiple product applications. The default permission of `library.policy` is empty.

app.policy

(Lines in the following example have been split for publication.)

```
<WAS_HOME>/config/cells/
<cellname>/nodes/<nodename>/app.policy
```

. The `app.policy` file defines the default permissions granted to all enterprise applications running on <nodename> or <cellname>.

was.policy

(Lines in the following example have been split for publication.)

```
<WAS_HOME>/config/cells
/<cellname>/applications
/<earfilename>/deployments
/<applicationname
>/META-INF/was.policy
```

. This template is for application-specific permissions. The `was.policy` is embedded in the EAR file.

ra.xml

<rarfilename>/META-INF/was.policy. RAR. This file can have a permission specification defined in the `ra.xml` file. The `ra.xml` file is embedded in the RAR file.

Syntax of the policy file

A policy file contains several policy entries. The format of each policy entry format follows:

```
grant [codebase <Codebase>] {
permission <Permission>;
 permission <Permission>;
permission <Permission>;
};

<CodeBase>:  A URL.
   For example,  "file:${java.home}/../lib/tools.jar"
      When [codebase <Codebase>] is not specified, listed permissions
are applied to everything.
      If URL ends with a JAR file name,  only the classes in the JAR file
belong to the codebase.
If URL ends with "/", only the class files in the specified directory
belong to the codebase.
If URL ends with "*", all JAR and class files in the specified directory
belong to the codebase.
If URL ends with "-", all JAR and class files in the specified directory and its
subdirectories belong to the codebase.
<Permissions>: Consists from
       Permission Type   : class name of the permission
          Target Name       : name specifying the target
          Actions           : actions allowed on target
   For example,
         java.io.FilePermission "/tmp/xxx", "read,write"
   Please refer JDK specification for the details of each permissions.
```

Syntax of dynamic policy

You can define permissions for specific types of resources in dynamic policy files for an enterprise application. This action is achieved by using product-reserved symbols. The reserved symbol scope depends on where it is defined. If you define the permissions in the `app.policy` file, the symbol applies to all the resources on all of the enterprise applications running on <*nodename*>. If you define the permissions in the `META-INF/was.policy` file, it only applies to the specific enterprise application. Valid symbols for codebase are listed in the following table:

Symbol	**Meaning**
file:${application}	Permissions apply to all resources within the application
file:${jars}	Permissions apply to all utility JAR file within the application
file:${ejbComponent}	Permissions apply to EJB resources within the application
file:${webComponent}	Permissions apply to Web resources within the application
file:${connectorComponent}	Permissions apply to connector resources within the application

Other than these entries specified by the codebase symbols, you can specify module name for a granular setting. For example:

```
"file:DefaultWebApplication.war" {
            permission java.security.SecurityPermission "printIdentity";
 };

grant codeBase "file:IncCMP11.jar" {
            permission java.io.FilePermission
"${user.install.root}${/}bin${/}DefaultDB${/}-", "read,write,delete";
};
Relative Code Base can be used only in META-INF/was.policy.
```

Several product-reserved symbols are defined to associate the permission lists to a specific type of resources.

Symbol	Meaning
file:${application}	Permissions apply to all resources within the application
file:${jars}	Permissions apply to all utility JAR files within the application
file:${ejbComponent}	Permissions apply to enterprise beans resources within the application
file:${webComponent}	Permissions apply to Web resources within the application
file:${connectorComponent}	Permissions apply to connector resources both within the application and stand alone connector resources.

There are five embedded symbols provided to specify the path and name for java.io.FilePermission. These symbols enables flexible permission specification. The absolute file path will be fixed after the installation of the application.

Symbol	Meaning
${app.installed.path}	Path where the application is installed
${was.module.path}	Path where the module is installed
${current.cell.name}	Current cell name

${current.node.name}	Current node name
${current.server.name}	Current server name

Note: You must not use the `${was.module.path}` in the `${application}` entry.

Carefully determine where to add a new permission. An incorrectly specified permission causes an AccessControlException exception. Since DynamicPolicy resolves the codebase at run time, determining which policy file has a problem is difficult. Add a permission only to the necessary resources. For example, use ${ejbcomponent}, and etc instead of ${application}, and update the `was.policy` file instead of the `app.policy` file, if possible.

Static Policy Filtering

There are some limited static policy filtering support. If the `app.policy` file and the `was.policy` file have permissions defined in `filter.policy` file, the runtime will remove the permissions from the applications and an audit message is logged. However, if the permissions defined in app.policy and was.policy are compound permissions, for example, `java.security.AllPermission`, the permission is not removed, rather an warning message is written to the log file. The policy filtering only supports Developer Kit permissions, (the permissions package name begins with `java` or `javax`).

If the **Issue Permission Warning** flag in the Global Security panel is enabled and if the `app.policy` filr and the `was.policy` file contain custom permissions (non-Developer Kit permissions, where the permissions package name begins with `java` or `javax`), a warning message logs. The permission is not removed. If the AllPermission permission is listed in the `app.policy` file and the `was.policy` file, a warning message logs.

Policy File Editing

Using the Policy tool provided by the Developer Kit (<WAS_HOME>/java/jre/bin/policytool), to edit the previous policy files is recommended. For Network Deployment, extract the policy files from the repository before editing. After the policy file is extracted, use the Policy tool to edit the file. The modified policy files must be checked into the repository and synchronized with other nodes.

Note: If there are syntax errors in the policy files, the enterprise application or server process may fail to start. Extreme care should be taken when editing these policy files. It is advised to use the policytool provided by the Developer Kit for editing the policy files (<WAS_HOME>/java/jre/bin/policytool).

Troubleshooting

To debug the dynamic policy, there are three ways to generate the detail report of the AccessControlException exception.

- **Trace** (Configured by RAS trace). Enables traces with the trace specification: (The line in the following example has been split for publication.)

  ```
  com.ibm.ws.security.policy.*=all=enabled:com.ibm.ws.security.core.SecurityManager=
  all=enabled
  ```

- **Trace** (Configured by property). Specifies a `java` property ″java.security.debug″. Valid values for property ″java.security.debug″ are:

  - Access. Print all debug information including, required permission, code, stack and code base location.
  - Stack. Print debug information including, required permission, code, and stack.
  - Failure. Print debug information including, required permission and code.
- **ffdc**. Enable `ffdc`, modify the `ffdcRun.properties` file by changing Level=4 and LAE=true. Look for a keyword `Access Violation` in the log file.

Configuring Java 2 security policy files:

Before you begin

Java 2 Security uses several policy files to determine the granted permission for each Java programs. See the Dynamic Policy article for the list of available policy files supported by WebSphere Application Server v5 and their details.

There are two types of policy files supported by WebSphere Application Server v5, dynamic policy files and static policy files. Static policy files provide the default permissions. Dynamic policy files provide application's permissions. There are six dynamic policy files.

Policy file	Description
app.policy file	Contains default permissions for all of the enterprise application in the cell.
was.policy file	Contains application specific permissions for an WebSphere Application Server enterprise application. This file is packaged in ear file.
ra.xml file	Contains connector application specific permissions for an WebSphere Application Server enterprise application. This file is packaged in RAR file.
spi.policy file	Contains permissions for SPI (Service Provider Interface) or third party resources embedded in WebSphere Application Server. The default contents grants everything. This file have to be updated when the cell requires more protection against SPI in the cell. Please be careful, since this file is applied to all of SPIs defined in resources.xml.
library.policy file	Contains permissions for shared library of enterprise application.
filter.policy	Contains the list of permission that has to be filtered out from was.policy file and app.policy file in the cell. This filtering mechanism only applies to was.policy and app.policy.

Steps for this task

1. Identify the policy file to be updated.
   - If the permission is required not only by application, update static policy file. Refer to Configuring static policy files.
   - If the permission is required by all of the WebSphere Application Server enterprise application in the node, Configuring spi.policy files.

- If the permission is required only by specific WebSphere Application Server enterprise application and the permission is required only by connector, ra.xml file has to be updated. Refer to Assembling Resource adapter modules. Otherwise was.policy file have to be updated. Refer to Configuring was.policy files and Adding the was.policy file to applications.
- If the permission is required by shared libraries. Refer to Configuring library.policy files.
- If the permission is required by shared libraries, Refer to Configuring spi.policy files.

Note: It is recommended to pick up the policy file of smaller scope. That way you can avoid giving an extra permission to the Java programs and protect the resources. Update ra.xml file or was.policy file than app.policy file. Use specific component symbols ($(ejbcomponent), ${webComponent},${connectorComponent} and ${jars}) than ${application} symbols. Update dynamic policy files than static policy files.

- If there is any permission that should never be granted to the WebSphere Application Server enterprise application in the cell, this permission have to be added to the filter.policy file. Refer to Configuring filter.policy files.

2. Restart the WebSphere Application Server enterprise application.

Results

The required permission is granted for the specified WebSphere Application Server enterprise application.

Usage scenario

If an WebSphere Application Server enterprise application in a cell requires permissions, some of the dynamic policy file have to be updated. The symptom of the problem of missing permission is the program getting an java.security.AccessControlException. The missing permission is listed in the exception data, for example,

(The lines in the following examples have been split for publication.)

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

When a Java program receives this exception and adding this permission is justified, add an permission to an adequate dynamic policy file, for example,

```
grant codeBase "file:<user client installed location>
" { permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read";
};
```

To decide whether to add a permission, refer to the article (AccessControlException).

Configuring app.policy files:

Before you begin

Java 2 Security uses several policy files to determine the granted permission for each Java programs. See the Dynamic Policy article for the list of available policy files supported by WebSphere Application Server v5 and their details. The app.policy file is a default policy file shared by all of the WebSphere Application Server enterprise application. The union of the permission contained in the app.policy file, server.policy file, the app.policy file, application's was.policy file and permission specification of ra.xml file are applied to the WebSphere Application Server enterprise application. The app.policy files are managed by configuration and file replication services, changes made in these files are replicated to other nodes in the Network Deployment cell.

If the default permissions for enterprise application (union of the permissions defined in the app.policy file, the server.policy file and the app.policy file) are enough, no action is required. The default app policy is picked up automatically. If a specific change is required to all of the enterprise application in the cell, the app.policy file must be updated. Syntax errors in the policy files will cause the application server fail to start. Extreme care should be taken when editing these policy files.

Steps for this task

1. Modify the app.policy file with policytool. The change of app.policy file is local for the node.

Results

The default Java 2 security policies have been changed for the enterprise application.

Usage scenario

Several product-reserved symbols are defined to associate the permission lists to a specific type of resources.

Symbol	Meaning
file:${application}	Permissions apply to all resources within the application
file:${jars}	Permissions apply to all utility JAR files within the application
file:${ejbComponent}	Permissions apply to enterprise beans resources within the application
file:${webComponent}	Permissions apply to Web resources within the application
file:${connectorComponent}	Permissions apply to connector resources both within the application and stand alone connector resources.

Also, there are five embedded symbols provided to specify the path and name for java.io.FilePermission. These symbols enables flexible permission specification. The absolute file path will be fixed after the installation of the application.

Symbol	Meaning
${app.installed.path}	Path where the application is installed
${was.module.path}	Path where the module is installed
${current.cell.name}	Current cell name
${current.node.name}	Current node name
${current.server.name}	Current server name

Note: The `${was.module.path}` can not be used in the `${application}` entry.

The app.policy file supplied by WebSphere Application Server resides at <WAS_HOME>/config/cells/<cellname>/nodes/<nodename>/app.policy, which contains the following default permissions: (Some lines in the following example have been split for publication.)

```
grant codeBase "file:${application}" {
  // The following are required by Java mail
  permission java.io.FilePermission
"${was.install.root}${/}java${/}jre${/}lib${/}ext${/}mail.jar", "read";
  permission java.io.FilePermission
"${was.install.root}${/}java${/}jre${/}lib${/}ext${/}activation.jar", "read";
};

grant codeBase "file:${jars}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

grant codeBase "file:${connectorComponent}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};
grant codeBase "file:${webComponent}" {
  permission java.io.FilePermission "${was.module.path}${/}-", "read, write";
  permission java.lang.RuntimePermission "loadLibrary.*";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

grant codeBase "file:${ejbComponent}" {
 permission java.lang.RuntimePermission "queuePrintJob";
 permission java.net.SocketPermission "*", "connect";
 permission java.util.PropertyPermission "*", "read";
};
```

If all of the WebSphere Application Server enterprise application in a cell requires permissions that is not defined as defaults in the app.policy file, the server.policy file and the app.policy, the app.policy file have to be updated. The symptom of the problem of missing permission is the program getting an java.security.AccessControlException. The missing permission is listed in the exception data, for example, `java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)`.

When a Java program receives this exception and adding this permission is justified, add an permission to server.policy, for example, grant codeBase ″file:<user client installed location>″ { permission java.io.FilePermission ″C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar″, ″read″; };.

To decide whether to add a permission, refer to the article (AccessControlException).

What to do next

Restart all WebSphere Application Server enterprise applications to ensure that the updated app.policy file takes an effect.

Configuring filter.policy files: Java 2 Security uses several policy files to determine the granted permission for each Java programs. Java 2 Security policy filtering is only in effect when Java 2 Security is enabled. Please refer to Configuring Java 2 Security. The filtering policy defined in the `filter.policy` file is cell wide. Refer to the article, Dynamic Policy, for the list of available policy files supported by WebSphere Application Server v5 and their details. The filter.policy file is the only policy file to be used to restrict the permission instead of granting. The permissions listed in the filter policy file will be filtered out from app.policy and was.policy. Permissions defined in the other policy files will not be affected by filter.policy file.

When a permission is filtered out, an audit message is logged. However, if the permissions defined in app.policy and was.policy are compound permissions, java.security.AllPermission, for example, the permission is not removed, rather an warning message is logged. If the Issue Permission Warning flag is enabled (default) is enabled, and if app.policy and was.policy contain custom permissions (non-JDK permission, the permission package name begins with characters other than `java` or `javax`), then a warning message is logged, and the permission is not removed. The value of the Issue Permission Warning flag can be changed from administrative console in the Global Security panel. If `java.security.AllPermission` is specified in the `app.policy` file or in the `was.policy` file, a warning message is logged. However, the permission is not removed. It is not recommended to use `AllPermission` for the enterprise application.

There are some default permissions defined in the `filter.policy` file. These are the minimal recommended by the product. If more permissions are added to the `filter.policy` file, certain operations can fail for enterprise applications. Be careful when adding permissions to the `filter.policy` file.

Note: If there are syntax errors in the `filter.policy` file, it will not be loaded by the product security run time, which implies that there is no filter in place. If there is no filter, then enterprise applications can access resources normally not allowed. Use extreme care editing the `filter.policy` file. In Version 5, there is no tool support for editing the `filter.policy` file.

Steps for this task

1. You cannot use the PolicyTool to edit the `filter.policy` file. Editing must be completed in a text editor. Be careful to ensure that there are no syntax errors in `filter.policy` file. If there are any syntax errors in `filter.policy` file, it will not be loaded by the product security run time, which implies that filtering is disabled.

Results

Updated filter.policy will be applied to all of the WebSphere Application Server enterprise application after the servers are restarted.

Usage scenario

The filter.policy file is managed by configuration and file replication services, changes made in the file is replicated to other nodes in the Network Deployment cell.

The filter.policy file supplied by WebSphere Application Server resides at <WAS_HOME>/config/cells/<cellname>/filter.policy.

It contains these permissions as defaults:

```
filterMask {
permission java.lang.RuntimePermission "exitVM";
permission java.lang.RuntimePermission "setSecurityManager";
permission java.security.SecurityPermission "setPolicy";
permission javax.security.auth.AuthPermission "setLoginConfiguration"; };
runtimeFilterMask {
permission java.lang.RuntimePermission "exitVM";
permission java.lang.RuntimePermission "setSecurityManager";
permission java.security.SecurityPermission "setPolicy";
permission javax.security.auth.AuthPermission "setLoginConfiguration"; };
```

The permissions defined in `filterMask` are for static policy filtering. The security run time tries to remove the permissions from applications during application startup. Compound permissions are not removed but issued with warning, and application deployment will be stopped if applications contain permissions defined in `filterMask`, and if scripting was used (wsadmin tool). The `runtimeFilterMask` defines permissions used by the security run time to deny access to those permissions to application thread. Do not add more permissions to the `runtimeFilterMask`. Doing so could cause application start failure or incorrect function. Be careful when adding more permissions to the `runtimeFilterMask`. Usually, you only need to add permissions to the `filterMask` stanza.

WebSphere Application Server itself relies on the filter policy file to restrict or disallow certain permissions that could compromise the integrity of the system itself. For instance, WebSphere Application Server considers the `exitVM` and `setSecurityManager` permissions as those permissions that most applications should never have. If these permissions were granted then the following scenarios are possible:

- **exitVM —** A servlet, JSP file, enterprise bean, or other library used by the aforementioned could call the System.exit() API and cause the entire WebSphere Application Server process to terminate.
- **setSecurityManager —** An application could install its own SecurityManager that could either grant more permissions or bypass the default policy the WebSphere Application Server SecurityManager enforces.

What to do next

For the updated `filter.policy` file to take effect, restart related Java processes.

Configuring was.policy:

Before you begin

Java 2 Security uses several policy files to determine the granted permission for each Java programs. Please see Dynamic Policy for the list of available policy files supported by WebSphere Application Server v5 and their details. The was.policy file is a application specific policy file for WebSphere Application Server enterprise application. It is embedded in the EAR file(META-INF/was.policy). It can be found at

```
WAS_HOME>/config/cells/<cellname>/applications/<earfilename>
/deployments/<applicationname>/META-INF/was.policy
```

The union of the permission contained in the java.policy file, server.policy file, the app.policy file, application's was.policy file and permission specification of ra.xml file are applied to the WebSphere Application Server enterprise application. was.policy file are managed by configuration and file replication services, changes made in these files are replicated to other nodes in the Network Deployment cell.

Several WebSphere reserved symbols are defined to associate the permission lists to a specific type of resources.

Symbol	Meaning
file:${application}	file:${application}
file:${jars}	Permissions apply to all utility jars within the application
file:${ejbComponent}	Permissions apply to enterprise beans resources within the application
file:${webComponent}	Permissions apply to WEB resources within the application
file:${connectorComponent}	Permissions apply to connector resources within the application

Other than these blocks, module name can be specified for granular setting. For example,

```
"file:DefaultWebApplication.war" {
          permission java.security.SecurityPermission "printIdentity";
 };

grant codeBase "file:IncCMP11.jar" {
          permission java.io.FilePermission
"${user.install.root}${/}bin${/}DefaultDB${/}-", "read,write,delete";
};
```

Also, there are five embedded symbols provided to specify the path and name for java.io.FilePermission. These symbols enables flexible permission specification. The absolute file path will be fixed after the installation of the application.

Symbol	Meaning
${app.installed.path}	Path where the application is installed
${was.module.path}	Path where the module is installed
${current.cell.name}	Current cell name
${current.node.name}	Current node name
${current.server.name}	Current server name

If the default permissions for enterprise application (union of the permissions defined in the java.policy file, the server.policy file and the app.policy file) are enough, no action is required. If an application has specific resource that has to access, the was.policy file have to be updated. The first two steps assume that you are creating a new policy file.

Note: Syntax errors in the policy files will cause the application server fail to start. Use extreme care when editing these policy files.

Steps for this task

1. Create a new was.policy using policytool.
2. Package it into the EAR file.
3. Start the Application Assembly Tool (AAT).
4. Select File->Open.

   a. Navigate the directory tree to pick up the application ear file you need to update.
   b. Click **Open**.

   The EAR file is loaded.

5. In left panel, click **Files**. List of the files are displayed in the right panel.
6. If was.policy already exists, right click was.policy in the right panel.
7. Select **Delete** to remove the existing was.policy.
8. Right click **Files** in the left panel.
   a. Select **AddFiles**.
9. Click **Browse**.
   a. Navigate the directory tree to pick up the directory where was.policy file is.
   b. Click **Select**.
      The directory contents is listed in **Add files** window.
10. Navigate the directory tree to pick up the was.policy file.
    a. Click **Add** button.
       Selected was.policy appears in the right panel.
11. Click **File** > **Verify**.
    Verification of EAR file is performed. Ensure that the was.policy is validated.
12. Select **File** > **Save** to save the updated EAR file.
13. **(Optional)** Update an existing installed application.
    a. Modify the was.policy file with policytool.
    a. Edit the extracted was.policy with policytool.
    b. Check in the policy file by entering the following from a command prompt:

```
wsadmin> $AdminConfig checkin cells/<cellname>/application/
<earfilename>/deployments/<applicationname>/META_INF/was.policy
c:/temp/test/was.policy $obj
```

       .

Results

Updated was.policy will be applied to the enterprise application after the application is restarted.

Usage scenario

If an enterprise application has to access a specific resource that is not defined as defaults in the java.policy file, the server.policy file and the app.policy, the was.policy file of that application have to be updated. The symptom of the problem of missing permission is the program getting an java.security.AccessControlException. The missing permission is listed in the exception data, for example, `java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)`.

When a Java program receives this exception and adding this permission is justified, add a permission to was.policy, for example, `grant codeBase "file:<user client installed location>" { permission java.io.FilePermission "C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };`.

To determine whether to add a permission, refer to (AccessControlException).

What to do next

In order for the updated app.policy file to take effect, all of the enterprise application has to be restarted.

Configuring spi.policy files: Java 2 Security uses several policy files to determine the granted permission for each Java programs. Please see Dynamic Policy for the list of available policy files supported by WebSphere 5.0 and their details.

Since the default permissions for SPI is AllPermission, usually there is no need to update spi.policy file, unless restricted spi.policy is required with some reason. In case some change in the spi.policy is required, complete the following steps.

Note: Syntax errors in the policy files will cause the application server fail to start. Extreme care should be taken when editing these policy files.

Steps for this task

1. Modify the spi.policy file with policytool.

Results

Updated spi.policy will be applied to SPI libraries after the Java process is restarted.

Usage scenario

The spi.policy file is the template for SPIs(Service Provider Interface) or third party resources embedded in the product. An example of SPIs are JMS (MQSeries) and JDBC drivers. They are specified in the resources.xml file. DynamicPolicy grants the permissions defined in the spi.policy file to the classpaths defined in resources.xml file. The union of the permission contained in the java.policy file and spi.policy file are applied to the SPI libraries. spi.policy file are managed by configuration and file replication services, changes made in these files are replicated to other nodes in the Network Deployment cell.

The spi.policy file supplied by WebSphere resides at <WAS_HOME>/config/cells/<cellname>/nodes/<nodename>/spi.policy. It contains the following default permission

```
grant {
  permission java.security.AllPermission;
};
```

Usually, no change is required to spi.policy. If there is any requirement to restrict SPI's permission, spi.policy file has to be updated.

What to do next

In order for the updated spi.policy file takes an effect, related Java processes have to be restarted.

Configuring library.policy files: Java 2 Security uses several policy files to determine the granted permission for each Java programs. See Dynamic Policy for the list of available policy files supported by WebSphere Application Server v5 and their details. The library.policy file is the template for library (Java library classes).

Shared library can be defined and be used by multiple enterprise application. Please refer Managing shared libraries how to define and manage the shared libraries.

If the default permissions for shared library (union of the permissions defined in the java.policy file, the app.policy file and the library.policy file) are enough, no action is required. The default library policy is picked up automatically. If a specific change is required to shared library in the cell, the library.policy file have to be updated.

Syntax errors in the policy files will cause the application server fail to start. Extreme care should be taken when editing these policy files.

Steps for this task

1. Modify the library.policy file with policytool.

Results

Updated library.policy will be applied to shared libraries after the servers are restarted.

Usage scenario

The union of the permission contained in the java.policy file, the app.policy file and the library.policy file are applied to the shared libraries. The library.policy file is managed by configuration and file replication services, changes made in the file is replicated to other nodes in the Network Deployment cell.

The library.policy file supplied by WebSphere Application Server resides at <WAS_HOME>/config/cells/<cellname>/nodes/<nodename>/library.policy, contains empty permission entry as a default. For example,

```
grant {
 };
```

If shared library in a cell requires permissions that is not defined as defaults in the java.policy file, app.policy file and the library.policy file, the library.policy file has to be updated. The symptom of the problem of missing permission is the program getting an java.security.AccessControlException. The missing permission is listed in the exception data, for example, `java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)`.

When a Java program receives this exception and adding this permission is justified, add an permission to library.policy, for example, grant codeBase ″file:<user client installed location>″ { permission java.io.FilePermission ″C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar″, ″read″; };

to decide to add a permission or not, refer to AccessControlException.

What to do next

In order for the updated library.policy file takes an effect, related Java processes have to be restarted.

Adding the was.policy file to applications: When Java 2 security is enabled for a WebSphere Application Server, all the applications that run on that WebSphere

Application Server undergo a security check before accessing any system resource. An application might need a was.policy if it access resources that require more permissions than have been granted in the default app.policy file. By default, the product security reads an app.policy file that is located in each node and grants the permissions in the app.policy file to all the applications. Any additional required permissions should be added in the was.policy file. The was.policy file is only required if an application requires additional permissions.

The default policy file for all applications is specified in the app.policy file. This file is provided by the product security and is common to all applications and should not be changed. Add any new permission required for each application in the was.policy file.

The app.policy file is located in <WAS_HOME>/config/cells/<cellname>/nodes/<nodename> directory. The contents of the app.policy file follow: (Some lines in the following example have been split for publication.)

```
// The following permissions apply to all the components under the application.
grant codeBase "file:${application}" {
  // The following are required by JavaMail
  permission java.io.FilePermission
"${was.install.root}${/}java${/}jre${/}lib${/}ext${/}mail.jar", "read";
  permission java.io.FilePermission
"${was.install.root}${/}java${/}jre${/}lib${/}ext${/}activation.jar", "read";
};

  // The following permissions apply to all utility .jar files
(other than enterprise beans jars ) under the  application.
grant codeBase "file:${jars}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

// The following permissions apply to connector resources within the application
grant codeBase "file:${connectorComponent}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

// The following permissions apply to all the Web modules (.war files)
within the application.
grant codeBase "file:${webComponent}" {
  permission java.io.FilePermission "${was.module.path}${/}-", "read, write";
       //  where "was.module.path" is the path where the web module is installed.
Refer to Dynamic Policy Concept for other symbols.
  permission java.lang.RuntimePermission "loadLibrary.*";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};


// The following permissions apply to all the EJB modules within the application
grant codeBase "file:${ejbComponent}" {
 permission java.lang.RuntimePermission "queuePrintJob";
 permission java.net.SocketPermission "*", "connect";
 permission java.util.PropertyPermission "*", "read";
};
```

If additional permissions are required for an application or one or more modules of an application, use the was.policy file for that application. For example, use codeBase of ${application} and add required permissions to grant additional permissions to the entire application. Similarly, use codeBase of ${webComponent} and ${ejbComponent} to grant additional permissions to all the Web modules and all the enterprise beans (EJB) modules in the application. You can assign additional permissions to each module (.war file or .jar file) as shown in the following example.

An example of the was.policy file to add extra permissions for an application follows:

```
// grant additional permissions to a WebModule
grant codeBase " file:aWebModule.war" {
 permission java.security.SecurityPermission "printIdentity";
};

// grant additional permission to a EJB Module
grant codeBase "file:aEJBModule.jar"  {
    permission java.io.FilePermission "${user.install.root}${/}bin${/}DefaultDB${/}-" .
"read.write,delete";
    // where, ${user.install.root} is the system property whose value is located in the
<install_root> directory.
 };
```

Steps for this task

1. Create a was.policy file using the policy tool located in the /java/jre/bin directory.
2. Add the required permissions in the was.policy file using the policy tool.
3. Place the was.policy file in the application EAR file. The was.policy file should be located under the META-INF directory of the application EAR file. Update the application EAR file with the was.policy file just created by using jar command.
4. Verify that the was.policy file is inserted, and start the Application Assembly Tool (AAT).
   a. Open the application EAR file.
   b. Click **File** > **Verify**. The verification process ensures that the was.policy file in the application is syntactically correct.

Results

An application EAR file is now ready to run when Java 2 security is enabled.

Usage scenario

This step is required for applications to run properly when Java2 security is enabled. If the was.policy file is not created and it does not contain required permissions, the application might not be able to access system resources.

The symptom of the missing permission problem is the application program getting the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example, java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read).

When an application program receives this exception and adding this permission is justified, add a permission to the was.policy file, for example, grant codeBase ″file:${application}″ { permission java.io.FilePermission ″C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar″, ″read″; };.

What to do next

Install the application.

Configuring static policy files:

Before you begin

Java 2 Security uses several policy files to determine the granted permission for each Java programs. See the Dynamic Policy article for the list of available policy files supported by WebSphere 5.0 and their details.

There are two types of policy files supported by WebSphere Application Server v5, dynamic policy files and static policy files. Static policy files provide the default permissions. Dynamic policy files provide application's permissions.

Policy file	Description
java.policy file	Contains default permissions for all of the Java program on the node. This file should not be touched most of the time.
server.policy file	Contains default permissions for all of the WebSphere Application Server program on the node. Most of the time this file does not have to be updated.
client.policy file	Contains default permissions for all of the applets and client containers on the node.

The static policy file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not get replicated to the other machine.

Steps for this task

1. Identify the policy file to be updated.
   - If the permission is required only by application, update dynamic policy file. Refer to Configuring Java 2 security policy files.
   - If the permission is required only by applets and client containers, update the client.policy file. Refer to Configuring client.policy files.
   - If the permission is required only by WebSphere Application Server servers (servers, agents, managers and application servers), the server.policy file have to be updated. Refer to Configuring server.policy files.
   - If the permission is required by all of the Java programs running on the Jave Virtual Machine (JVM), update the java.policy file. Refer to Configuring java.policy files.
2. Stop and restart the WebSphere Application Server server.

Results

The required permission is granted for all of Java programs running with the restarted JVM.

Usage scenario

If Java programs on a node requires permissions, some of the policy file have to be updated. If the Java program that required the permission is not a part of an enterprise application, static policy file have to be updated. The symptom of the problem of missing permission is the program getting an java.security.AccessControlException. The missing permission is listed in the exception data, for example,

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

When Java program receives this exception and adding this permission is justified, add an permission to an adequate policy file, for example,

```
grant codeBase "file:<user client installed location>" {
  permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read";
};
```

To decide whether to add a permission, refer to the article (AccessControlException).

Configuring java.policy files: Java 2 Security uses several policy files to determine the granted permission for each Java programs. See Dynamic Policy for the list of available policy files supported by WebSphere Application Server v5 and their details. The java.policy file is a global default policy file shared by all of the Java programs running in Jave Virtual Machine (JVM) on the node. It is not recommended to modify this file.

Steps for this task

1. If a specific change is required to some of the java programs on a node and the java.policy file must be updated, modify the java.policy file with policytool. The change of java.policy file is local for the node.

   The java.policy file is not supposed to be changed. The default java policy is picked up automatically. Syntax errors in the policy files will cause the application server fail to start. Extreme care should be taken when editing these policy files.

Results

Updated java.policy will be applied to all of the java programs running in all of the JVM on the local node. In order for the updated java.policy takes effect, the programs have to be restarted.

Usage scenario

The java.policy file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not get replicated to the other machine. The java.policy file supplied by WebSphere Application Server resides at <WAS_HOME>/java/jre/lib/security/java.policy. It contains these default permission.

```
// Standard extensions get all permissions by default
grant codeBase "file:${java.home}/lib/ext/*" {
        permission java.security.AllPermission;
};
// default permissions granted to all domains
grant {
        // Allows any thread to stop itself using the java.lang.Thread.stop()
        // method that takes no argument.
        // Note that this permission is granted by default only to remain
        // backwards compatible.
        // It is strongly recommended that you either remove this permission
        // from this policy file or further restrict it to code sources
        // that you specify, because Thread.stop() is potentially unsafe.
        // See "http://java.sun.com/notes" for more information.
        // permission java.lang.RuntimePermission "stopThread";

        // allows anyone to listen on un-privileged ports
        permission java.net.SocketPermission "localhost:1024-", "listen";

        // "standard" properties that can be read by anyone

        permission java.util.PropertyPermission "java.version", "read";
        permission java.util.PropertyPermission "java.vendor", "read";
        permission java.util.PropertyPermission "java.vendor.url", "read";
        permission java.util.PropertyPermission "java.class.version", "read";
        permission java.util.PropertyPermission "os.name", "read";
        permission java.util.PropertyPermission "os.version", "read";
        permission java.util.PropertyPermission "os.arch", "read";
        permission java.util.PropertyPermission "file.separator", "read";
        permission java.util.PropertyPermission "path.separator", "read";
        permission java.util.PropertyPermission "line.separator", "read";

        permission java.util.PropertyPermission "java.specification.version", "read";
        permission java.util.PropertyPermission "java.specification.vendor", "read";
        permission java.util.PropertyPermission "java.specification.name", "read";

        permission java.util.PropertyPermission "java.vm.specification.version","read";
        permission java.util.PropertyPermission "java.vm.specification.vendor","read";
        permission java.util.PropertyPermission "java.vm.specification.name", "read";
        permission java.util.PropertyPermission "java.vm.version", "read";
        permission java.util.PropertyPermission "java.vm.vendor", "read";
        permission java.util.PropertyPermission "java.vm.name", "read";
       };
```

If some Java programs on a node requires permissions that is not defined as defaults in the java.policy file and updating java.policy is justified, the java.policy file may be updated. Most of the time, other policy files should be updated instead of the java.policy file. The symptom of the problem of missing permission is the program getting an java.security.AccessControlException. The missing permission is listed in the exception data, for example, `java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)`.

When java program receives this exception and adding this permission is justified, add an permission to java.policy, for example, `grant codeBase ″file:<user client installed location>″ { permission java.io.FilePermission ″C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar″, ″read″; };`.

To decide whether to add a permission, refer to the article on (AccessControlException).

What to do next

Restart all of the Java processes for the updated java.policy file to take effect.

Configuring server.policy files: Java 2 Security uses several policy files to determine the granted permission for each Java programs. See Dynamic Policy for the list of available policy files supported by WebSphere 5.0 and their details. The server.policy file is a default policy file shared by all of the WebSphere servers on a node. The server.policy file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not get replicated to the other machine.

Steps for this task

1. If the default permissions for server (union of the permissions defined in the server.policy file and the server.policy file) are enough, no action is required. The default server policy is picked up automatically. If a specific change is required to some of the server programs on a node, the server.policy file have to be updated, modify the server.policy file with policytool.

   Refer to the Using the policy tool article to edit policy files. *The change of server.policy file is local for the node. Syntax errors in the policy files will cause the application server fail to start. Extreme care should be taken when editing these policy files.*

Results

Updated server.policy will be applied to all of the server programs on the local node. In order for the updated server.policy takes effect, the servers have to be restarted.

Usage scenario

If you want to add permissions to an application, then use app.policy and was.policy.

However, for those time when you do need to modify the server.policy file, the file supplied by this product resides at `<WAS_HOME>/properties/server.policy` It contains these default permission.

```
// Allow to use sun tools
grant codeBase "file:${java.home}/../lib/tools.jar" {
  permission java.security.AllPermission;
};

// WebSphere system classes
grant codeBase "file:${was.install.root}/lib/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/classes/-" {
  permission java.security.AllPermission;
};

// Allow the WebSphere deploy tool all permissions
grant codeBase "file:${was.install.root}/deploytool/-" {
  permission java.security.AllPermission;
};
```

If some server programs on a node requires permissions that is not defined as defaults in the server.policy file and the server.policy file, the server.policy file have to be updated. The symptom of the problem of missing permission is the program getting an java.security.AccessControlException. The missing permission is listed in the exception data, for example, `java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)`.

When Java program receives this exception and adding this permission is justified, add an permission to server.policy, for example, grant codeBase "file:<user client installed location>" { permission java.io.FilePermission "C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };.

To decide whether to add a permission, refer to (AccessControlException).

What to do next

Restarted all of the Java processes for the updated server.policy file to take effect.

Configuring client.policy files: Before you begin

Java 2 Security uses several policy files to determine the granted permission for each Java programs. Please see Dynamic Policy for the list of available policy files supported by WebSphere Application Server v5 and their details. The client.policy file is a default policy file shared by all of the WebSphere Application Server client container and applets on a node. The union of the permission contained in the client.policy file and the client.policy file are given to all of the WebSphere client container and applets running on the node. The client.policy file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not get replicated to the other machine. The client.policy file supplied by WebSphere resides at <WAS_HOME>/properties/client.policy. It contains these default permissions:

```
grant codeBase "file:${java.home}/lib/ext/*" {
  permission java.security.AllPermission;
};
// JDK classes
grant codeBase "file:${java.home}/lib/ext/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${java.home}/../lib/tools.jar" {
  permission java.security.AllPermission;
};
// WebSphere system classes
grant codeBase "file:${was.install.root}/lib/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/classes/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/installedConnectors/-" {
  permission java.security.AllPermission;
};
// J2EE 1.3 permissions for client container WAS applications in $WAS_HOME/installedApps
grant codeBase "file:${was.install.root}/installedApps/-" {
  //Application client permissions
  permission java.awt.AWTPermission "accessClipboard";
  permission java.awt.AWTPermission "accessEventQueue";
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  permission java.lang.RuntimePermission "exitVM";
  permission java.lang.RuntimePermission "loadLibrary";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.net.SocketPermission "localhost:1024-", "accept,listen";
  permission java.io.FilePermission "*", "read,write";
  permission java.util.PropertyPermission "*", "read";
};
// J2EE 1.3 permissions for client container - expanded ear file code base
grant codeBase "file:${com.ibm.websphere.client.applicationclient.archivedir}/-"
 {
  permission java.awt.AWTPermission "accessClipboard";
  permission java.awt.AWTPermission "accessEventQueue";
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  permission java.lang.RuntimePermission "exitVM";
  permission java.lang.RuntimePermission "loadLibrary";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.net.SocketPermission "localhost:1024-", "accept,listen";
  permission java.io.FilePermission "*", "read,write";
  permission java.util.PropertyPermission "*", "read";
};
```

```
// For MQ Series
grant codeBase "file:${mq.install.root}/java/*" {
  permission java.security.AllPermission;
};
```

Steps for this task

1. If the default permissions for client(union of the permission defined in the client.policy file and the client.policy file) are enough, no action is required. The default client policy is picked up automatically.
2. If a specific change is required to some of the client containers and applets on a node, modify the client.policy file with policytool. Please refer to the article, Using policytool, to edit policy files. The change of client.policy file is local for the node.

Results

All of the client containers and applets on the local node will be granted the updated permissions at the time of execution.

Usage scenario

If some client containers or applets on a node requires permissions that is not defined as defaults in the client.policy file and the default client.policy file, the client.policy file have to be updated. The symptom of the problem of missing permission is the program getting an java.security.AccessControlException. The missing permission is listed in the exception data, for example,`java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)`.

When client program receives this exception and adding this permission is justified, add an permission to client.policy, for example, grant codeBase ″file:<user client installed location>″ { permission java.io.FilePermission ″C:\WebSphere\AppServer\java\jre\lib\ext\mail.ja″, ″read″; };.

In order to decide to add a permission or not, please refer to the article on (AccessControlException).

What to do next

Close and restart the browser. You must also restart the client application if they have one.

Migrating Java 2 Security Policy

Background: Previous WebSphere Releases

Starting from Version 3.x, WebSphere Application Server installed a Java 2 Security Manager in the server run time to prevent Enterprise Applications from calling `System.exit()` and `System.setSecurityManager()`. These two Java APIs have undesired consequences if called by Enterprise Applications. The `System.exit()` API, for example, causes the Java Virtual Machine (Application Server process) to exit prematurely, which is an undesirable operation for an application server.

However, Java 2 Security was not a fully support feature prior to Version 5. To support Java 2 Security properly, all the server run time must be marked as privileged (with doPrivileged() API calls inserted in the correct places), and identify the default permission sets or policy. Application code is not privileged and subject to the permissions in defined in the policy files. The doPrivileged instrumentation is important and necessary to support Java 2 Security. Without it, the application code must be granted the permissions required by the server run time. This is due to the design and algorithm used by Java 2 Security to enforce permission check. Please refer to the Java 2 Security check permission algorithm.

The following two permissions are enforced by the WebSphere Java 2 Security Manager (hard coded):

- java.lang.RuntimePermission(exitVM)
- java.lang.RuntimePermission(setSecurityManager)

Application code will be denied access to these permissions regardless what is the Java 2 Security policy has been granted to the application. However, the server run time is granted these permissions. All the other permission checks is not enforced.

Partial support has been introduced since the 4.02 product release. Prior to Version 4.0.2, Java 2 Security was not support at any capacity. Starting from the 4.02 product release, only two permissions are supported (enforced):

- java.net.SocketPermission
- java.net.NetPermission

However, not all the product server run time is properly marked as privileged prior to this release. The application code has to be given all the other permissions beside the above two listed, else the enterprise application could potentially failed to run. This Java 2 Security policy for enterprise application is very forgiving or liberal.

What Changed

Java 2 Security is fully supported (but optional — it may be enabled or disabled) in Version 5, which means all permissions are enforced. The default Java 2 Security policy for enterprise application is the recommended permission set defined by the J2EE 1.3 Specification. Please refer to the ${*install_root*}/config/cells/<cellname>/nodes/<nodename>/app.policy file for the default Java 2 Security policy granted to enterprise application. This is a much more stringent policy compare to previous releases.

There is no hard coded policy, all policy is declarative. The product Security Manager honors all policy declared in policy files. There is an exception to this rule, enterprise applications are denied access to permissions declared in ${*install_root*}/config/cells/<cellname>filter.policy.

Note: Enterprise applications that run on Version 4.0.x with Java 2 Security enabled are not guarantee to run successfully when migrating to Version 5 (when Java 2 Security is enabled), even its Java 2 Security policy is migrated properly. This is because the default Java 2 Security policy for enterprise application is much more stringent and ALL permissions are enforced in Version 5. It may fail because the application code does not have the necessary permissions granted where system resources (such as file I/O for example) may be programmatically accessed and are now subject to the permission checking.

Migrating:

System Properties

The following system properties are used in previous releases in relation to Java 2 Security:

- **java.security.policy**. The absolute path of the policy file (action required). It has a mixed of both system permissions (permissions granted to the JVM and the product server run time) and enterprise application permissions. The Java 2 Security policy of the enterprise application needs to be migrated to Version 5. For Java 2 Security policy migration, see the steps for migrating Java 2 Security policy in this document.
- **enableJava2Security**. Used to enable Java 2 Security enforcement (no action required). This is deprecated, a flag in the WebSphere Common Configuration Model (WCCM) was used to control whether to enabled Java 2 Security or not. This options can be enabled or disabled through the administrative console.
- **was.home**. Expanded to the installation directory of the WebSphere Application Server (action might required). This is deprecated, superseded by `${user.install.root}` and `${was.install.root}`. If the directory contains instance specific data then `${user.install.root}` is used, else `${was.install.root}` is used. They can be used interchangeably for the WebSphere Application Server or the Network Deployment environment. See the steps below in Migrating Java 2 Security Policy for details.

Java 2 Security Policy

There is no easy way of migrating the Java policy file from Version 4.0.x automatically because there is a mixture of system permissions and application permissions in the same policy file. The Java 2 Security policy for enterprise applications would have to manually copy over to a `was.policy` or `app.policy` files. However, migrating the Java 2 Security policy over to a `was.policy` file is a preferable way (implies that symbols or relative codebase is used instead of absolute codebase). There are many advantages this process, rather than migrating the permissions to the `app.policy` file. The permissions defined in the `was.policy` file should only be granted to the specific enterprise application, while permissions in the `app.policy` file applies to all the enterprise applications running on that node where the `app.policy` file belongs. Refer to the Dynamic Policy (Policy Management) article for more details on policy management.

The following is an example of how to migrate Java 2 Security policy from previous releases. The contents include the Java 2 Security policy file (the default is `WAS_HOME/properties/java.policy`) for the `app1.ear` enterprise application, the system permissions (permissions granted to the Java Virtual Machine (JVM) and product server run time), and default permissions are omitted for clarity:

```
// For product Samples
   grant codeBase "file:${was.home}/installedApps/app1.ear/-" {
     permission java.security.SecurityPermission "printIdentity";
     permission java.io.FilePermission "${was.home}${/}temp${/}somefile.txt", "read";
   };
```

For clarity of illustration, all the permissions are migrated over as the application level permissions in this example. However, you can grant permissions at a more granular level, like component level (Web, enterprise beans, connector or utility JAR component level) or to a particular component.

Steps for this task

1. Ensure that Java 2 Security is disabled on the application server.

2. Create a new *was.policy* file (if one is not present) or update the *was.policy* for migrated applications in the configuration repository in (*config/cells/<cellname>/applications/app.ear/deployments/app/META-INF/was.policy*) with the following contents:

   (Lines in the following example have been split for publication.)

   ```
   grant codeBase "file:${application}" {
        permission java.security.SecurityP
   ermission "printIdentity";
        permission java.io.FilePermission
   "${user.install.root}${/}temp${/}somefile.txt", "read";
      };
   ```

3. Use the Application Assembly Tool (AAT) to attach the *was.policy* to the EAR file. You can use the AAT or the policytool provided by the Java Development Kit (JDK)to validate the contents of the *was.policy* file. This step is only required for re-deploying enterprise applications prior to J2EE 1.3.

4. It is important to validate that the enterprise application does not require additional permissions beside the migrated Java 2 Security permissions and the default permissions set declared in *${was.install.root}/config/cells/<cellname>/nodes/<nodename>/app.policy*. This requires code review, code inspection, application documentation review, and sandbox testing of migrated enterprise applications with Java 2 Security enabled in a pre-production environment. Refer to JDK APIs protected by Java 2 Security for information about which JDK APIs are protected by Java 2 Security. If you use third party libraries, you might want to consult the vendor documentation for APIs that are protected by Java 2 Security. Ensure the application is granted all the required permissions, or it might fail to run when Java 2 Security is enabled.

5. It is very important to perform pre-production testing of the migrated enterprise application with Java 2 Security enabled.

   Hint: Enable trace for the WebSphere Application Server Java 2 Security Manager in the pre-production testing environment (with trace string: *com.ibm.ws.security.core.SecurityManager=all=enabled*). This can be helpful in debugging the *AccessControlException* exception thrown when an application is not granted the required permission or some system code is not properly marked as *privileged*. The trace dumps the stack trace and permissions granted to the classes on the call stack when the exception is thrown.

   Note: Because the Java 2 Security policy is much more stringent compared with previous releases, it is strongly advised that the administrator or deployer review their enterprise applications to see if extra permissions are required before enabling Java 2 Security. If the enterprise applications are not granted the required permissions, they fail to run.

Troubleshooting security configurations

Refer to Security components troubleshooting tips for instructions on how to troubleshoot errors related to security.

The following topics explain how to troubleshoot specific problems related to configuring and enabling security configurations:

- Errors when configuring or enabling security
- Errors or access problems after enabling security
- Errors after enabling Secure Socket Layer (SSL) or SSL-related error messages

Tuning security configurations

Before you begin

Performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing involved, the slower the performance will be. Consider what type of security is necessary and what you can disable in your environment. For example, if your application servers are running in a Virtual Private Network (VPN), consider whether you must disable Single Sockets Layer (SSL). If you have a lot of users, can they be mapped to groups and associated the groups to your J2EE roles? These questions are things to consider when designing your security infrastructure.

The steps for general security tuning follow:

Steps for this task

1. Consider disabling Java 2 Security Manager, if you know exactly what code is put onto your server and you do not need to protect process resources. Remember that in doing so you are putting your local resources at some risk.
2. Disable security for the specific application server that does not require resource protection, as some application servers do not have any protected resources. If the application server needs to go downstream with credentials, however, this action may not be feasible.
3. Consider increasing the cache and token time-out if you feel your environment is secure enough. By doing so, you have to reauthenticate less often. This action will allow subsequent requests to reuse the credentials already created. The downside of increasing the token time-out is the exposure of having a token high-jacked and providing the highjacker more time to hack into the system before the token expires.
4. Consider changing your administrative connector from Simple Object Access Protocol (SOAP) to Remote Method Invocation (RMI) since RMI uses stateful connections while SOAP is completely stateless. Run a benchmark to determine if the performance has been improved in your environment.
5. Use the `wsdmin` script to complete the access IDs for all the users and or groups to speed up the application startup. Complete this action if applications contain many users and or groups or if applications are stopped and started frequently.
6. Consider tuning the ORB because it is a factor in enterprise beans performance with or without security enabled. Refer to the article, ORB tuning guidelines.

Tuning CSIv2

Steps for this task

1. Consider using SSL client certificates instead of a user ID and password to authenticate Java clients. Since you are already making the SSL connection, using mutual authentication adds little overhead while removing the service context containing the user ID/password completely.
2. If you send a large amount of data that is not very security sensitive, reduce the strength of your ciphers. The more data you have to bulk encrypt, and the stronger the cipher, the longer this action takes. If the data is not sensitive, do not waste your processing with 128-bit ciphers.
3. Consider putting just an asterisk in the trusted server ID list (meaning trust all servers) when you use Identity Assertion for downstream delegation. Use SSL mutual authentication between servers to provide this trust. Adding this extra step in the SSL handshake performs better than having to fully authenticate the

upstream server and check the trusted list. When an asterisk is used, we simply trust the identity token. The SSL connection trusts the server by way of client certificate authentication.

4. Ensure that stateful sessions are enabled for CSIv2. This is the default, but it will only require authentication on the first request and any subsequent token expirations.
5. If you are only communicating with WebSphere Application Server Version 5 servers, make the Active Authentication Protocol just CSI, instead of CSI and SAS. This action removes an interceptor invocation for every request on both the client and server sides.

Tuning LDAP Authentication

Steps for this task

1. Select the **Ignore Case** check box in the WebSphere Application Server LDAP User Registry configuration, when case-sensitivity is not important.
2. Select **Reuse Connections** in the WebSphere Application Server LDAP User Registry configuration.
3. Check to see what caches your LDAP server has and take advantage of them. This action is best with LDAP servers that do not change frequently.
4. Choose the directory type of either IBM_Directory_Server or SecureWay, if you are using an IBM Directory Server. The `IBM_Directory_Server` yields improved performance because it is programmed to use the new group membership attributes to improve group membership searches. However, it is required that authorization is case insensitive to use the IBM_Directory_Server type.
5. Choose either iPlanet or Netscape as directory if you are an iPlanet user. Using the iPlanet directory type has improved performance in group membership lookup. However, only use **Role** for group mechanisms.

Tuning Web Authentication

Steps for this task

1. Consider increasing the cache and token time-out if you feel your environment is secure enough. The Web authentication information is stored in these caches and as long as the authentication information is in the cache, the login module is not invoked to authenticate the user. This will allow subsequent requests to reuse the credentials already created. The downside of increasing the token time-out is the exposure of having a token high-jacked and providing the highjacker more time to hack into the system before the token expires.
2. Consider enabling Single Sing-On (SSO). SSO is only available when you select **LTPA** as the authentication mechanism in the global security panel. When you select SSO, a single authentication to one application server is enough to make requests to multiple application servers in the same SSO domain. There are some situations where SSO is not desirable and should not be used in those situations.

Tuning Authorization

Steps for this task

1. Consider mapping your users to groups in the user registry. Then associate the groups with your J2EE roles. This association greatly improves performance as the number of users increase.
2. Judiciously assign method-permissions for enterprise beans. For example, you can use an asterisk (*) to indicate all methods in the method-name element. When all the methods in enterprise beans require the same permission, an

asterisk (*) can be used for method-name to indicate all methods. This indication reduces the size of deployment descriptor and hence reduce the memory required to load the deployment descriptor. It also reduces the search time during method-permission match for the enterprise beans method.

3. Judiciously assign security-constraints for servlets. For example, you can use the URL pattern `*.jsp` to apply the same authentication data constraints to indicate all JSP files. For a given URL, the exact match in the deployment descriptor takes precedence over longest path match. Use the extension match (`*.jsp`, `*.do`, `*.html`) if there is no exact match and longest path match for a given URL in the security constraints.

Results

There is always a trade-off between performance, feature and security. Security typically adds more processing time to your requests, but for a good reason. Not all security features are required in your environment. When you decide on tuning security, you should create a benchmark before making any change to ensure the change is improving performance.

Usage scenario

In a large scale deployment, performance is very important. Running benchmark measurements with different combinations of features can help you to determine the best performance versus benefit configuration for your environment.

What to do next

Continue to run benchmarks if anything changes in your environment, to help determine the impact of these changes.

Security: Resources for learning

Use the following links to find relevant supplemental information about Securing applications and their environment. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:
- Planning, business scenarios and IT architecture
- Programming model and decisions
- Programming specifications
- Administration

Planning, business scenarios and IT architecture

- **WebSphere Application Server Library** (http://www-3.ibm.com/software/webservers/appserv/library.html)
- **WebSphere Application Server Support** (http://www-3.ibm.com/software/webservers/appserv/support.html)

Programming model and decisions

- JSSE Documentation.

  Look in {was_install_root}/web/docs/jsse/jssedocs.jar for the javadoc of the APIs. This document contains the following when unpacked: index.html (Developer’s Guide (JSSE JavaDoc))

  – For the JSSE Reference Guide look in {was_install_root}/web/docs/jsse/JSSERefGuide.html

  – For sample JSSE applications look in {was_install_root}/web/docs/jsse/samplejsse.jar
- iKeyman Documentation.

  Look in {was_install_root}/web/docs/ikeyman/ikmuserguide.pdf for the SSL Introduction and iKeyman.
- JCE Documentation.

  – For the JCA spec and JCE API usage refer to <install_root>/web/docs/jce/api_users_guide.html.

  – For JCE sample applications refer to <was_install_root>/web/docs/jce/SampleJCE.jar.

  – For Java Cryptography Architecture Reference refer to <install_root>/web/docs/jce/CryptoSpec.html.

  – For how to implement a JCE provider refer to <install_root>/web/docs/jce/HowToImplAProvider.html.

  – For the javadoc of JCE APIs refer to <install_root>/web/docs/jce/jcedocs.jar.

  – For overview of IBM JCE refer to <install_root>/web/docs/jce/readme.jce.ibm.html.
- Application Assembly Tool (AAT) Documentation.

  Refer to <was_install_root>/web/docs/aat/en/index.html for AAT documentation. This can help when securing J2EE enterprise applications.
- **Java 2 Platform Security for JDK 1.3 http://java.sun.com/j2se/1.3/docs/guide/security/index.html**

Programming specifications

- **J2EE Specifications http://java.sun.com/j2ee/download.html**
- **EJB Specifications http://java.sun.com/products/ejb/docs.html**
- **Servlet Specifications http://java.sun.com/products/servlet/download.html**
- **Common Secure Interoperability Version 2 (CSIv2) Specification http://www.omg.org/technology/documents/corba_spec_catalog.htm#CSIv2**
- JAAS Specification.

  For programming and usage in JAAS, refer to the specification located at {was_install_root}/web/docs/jaas/JaasDocs.jar. This document contains the following when unpacked:

  – login.html - LoginModule Developer’s Guide

  – api.html - Developer’s Guide (JAAS JavaDoc)

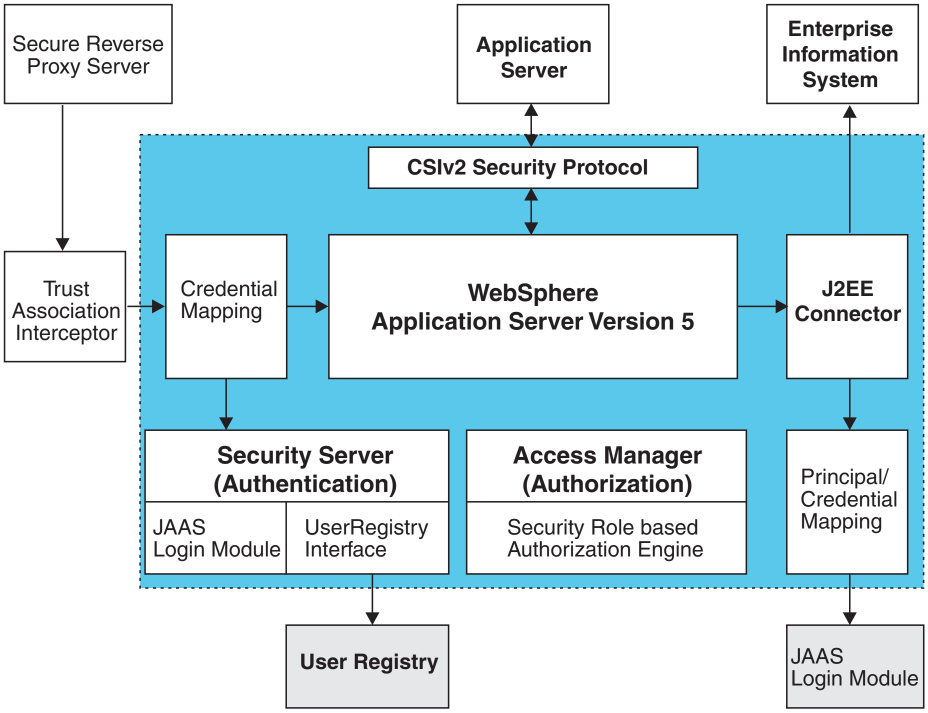
  – HelloWorld.tar - Sample JAAS Application
- **Java 2 Platform, Standard Edition, v1.3.1 API Specification http://java.sun.com/j2se/1.3/docs/api/index.html**

Administration

- **WebSphere Application Server Version 4.0 Security Redbook: WebSphere Security Model http://www.redbooks.ibm.com**
- **IBM HTTP Server Support and Documentation http://www-3.ibm.com/software/webservers/httpservers/support.html**
- **IBM Directory Server Support and Documentation http://www-3.ibm.com/software/network/directory/support/**
- (IBM JDK Readme)
  - For JDK Developer's Kit Readme refer to `{was_install_root}/java/docs/readme.devkit.ibm.html`
  - For JDK Installation and Configuration Readme refer to `{was_install_root}/java/docs/readme.install.ibm.html`
- **IBM cryptographic hardware devices http://www-3.ibm.com/security/cryptocards/html/library.shtml**

Chapter 3. Integrating IBM WebSphere Application Server security with existing security systems

WebSphere Application Server plays an integral part of the multiple-tier enterprise computing framework. WebSphere Application Server adopts the open architecture paradigm and provides many plug-in points to integrate with enterprise software components to provide end-to-end security. WebSphere Application Server plug-in points are based on standard J2EE specifications wherever applicable. WebSphere Application Server is actively involved in various standard bodies to externalize and to standardize plug-in interfaces. In the following several typical multiple-tier enterprise network configurations will be discussed. In each case, it will be shown how the various WebSphere Application Server plug-in points are used to integrate with other business components. We start with a basic multiple-tier enterprise network configuration as shown in the following picture.

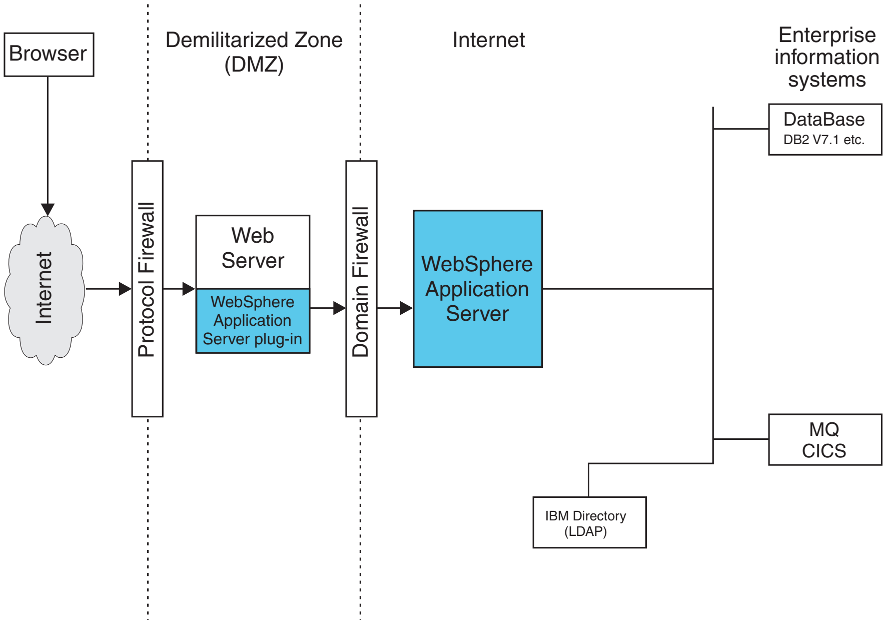

Secure Reverse Proxy Server
Application Server
Enterprise Information System
CSIv2 Security Protocol
Trust Association Interceptor
Credential Mapping
WebSphere Application Server Version 5
J2EE Connector
Security Server (Authentication)
JAAS Login Module
UserRegistry Interface
Access Manager (Authorization)
Security Role based Authorization Engine
Principal/ Credential Mapping
User Registry
JAAS Login Module

A list of terms used in this discussion follows:

- **Protocol firewall** - prevents unauthorized access from the Internet to the demilitarized zone. The role of this node is to allow the Internet traffic access only on certain ports and to block other IP ports.
- **WebSphere Application Server plug-in** - also referred to in WebSphere Application Server literature as *Web server redirector* was introduced in order to separate Web server from application server. Its job is to redirect to WebSphere Application Server all the request for servlets and JSP pages. Advantage of using Web server redirector is that you can move an application server and all the application business logic behind the domain firewall.
- **Domain firewall** - prevents unauthorized access from the demilitarized zone to an internal network. The role of this firewall is to allow the network traffic originated from the demilitarized zone and note from the Internet only.

- **Directory** - provides the information about the users and their rights in the Web application. The information may contain user IDs, passwords, certificates, access groups, and so forth. This node supplies the information to the security services like authentication and authorization service.
- **Enterprise information system** - these are existing enterprise applications and business data in back-end databases.

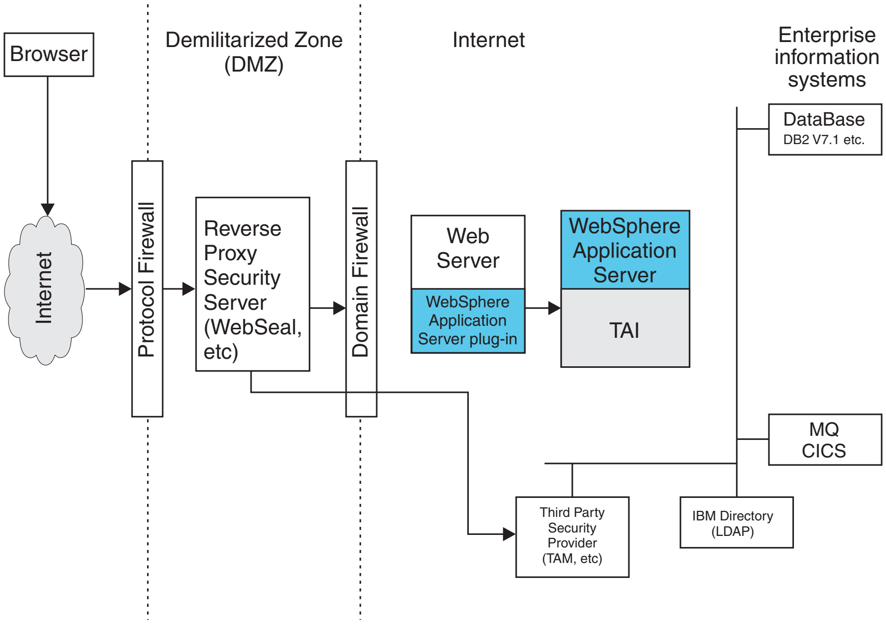
WebSphere Application Server provides the infrastructure to run application business logic and communicate with internal back-end systems and databases that can be accessed by Web applications and enterprise beans. WebSphere Application Server has a built in HTTP(S) server that may accepts client requests. A typical configuration, however, places WebSphere Application Server behind the domain firewall for better protection. WebSphere Application Server plug-in to Web server can be configured to redirect Web requests to WebSphere Application Server. WebSphere Application Server provides plug-ins for many popular Web servers. You can configure WebSphere Application Server and the Web server plug-in to communicate via secure SSL channels. You can configure a WebSphere Application Server HTTP server to allow communication channel to be opened only with restricted set of Web server plug-ins. You can configure the HTTP server to require client certificate authentication with self-signed certificates and to trust only the signer certificate. For instructions on how to generate self-signed certificate and to setup secure communications channels between HTTP server and the WebSphere Application Server plug-in, refer to (Configuring IHS Plug-in and the Internal Web Server for SSL) and (Configuring IHS for SSL Mutual Authentication).



The WebSphere Application Server plug-in routes HTTP requests according to the virtual host and port configuration and URL pattern matching. Client authentication and finer grained access control are handled by WebSphere Application Server behind the firewall.

In cases where the Web server may contain sensitive data and direct access to it is not desirable, the following configuration uses Tivoli WebSeal to shield a Web server from unauthorized requests. WebSeal is a Reverse Proxy Security Server (RPSS) that uses Tivoli Access Manager (TAM) to perform coarse-grained access control to filter out unauthorized requests before they reach the domain firewall.

WebSeal uses Tivoli Access Manager (TAM) to perform access control as illustrated in the picture. WebSphere Application Server supports various user registry implementations through the pluggable user registry interface. WebSphere Application Server ships LocalOS user registry implementation for Windows NT, AIX, AS400, and LDAP. WebSphere Application Server also allows users to develop their own custom registry and plug in via the pluggable user registry interface. When integrated with a third party security provider, WebSphere Application Server can share the user registry with the third party security provider. In the particular example of integrating with WebSeal, you can configure WebSphere Application Server to use the LDAP user registry, which can be shared with WebSeal and TAM. Moreover, you can configure WebSphere Application Server to use the Light Weight Third Party (LTPA) authentication mechanism, which supports the Trust Association Interceptor plug-in point. Basically the RPSS performs authentication and add proper authentication data into the request header and then redirect the request to Web server. A trust relationship is formed between a RPSS and WebSphere Application Server, and the RPSS can assert client identity to WebSphere Application Server to achieve Single Sign-on between RPSS and WebSphere Application Server. When the request is forward to WebSphere Application Server, WebSphere Application Server uses the TAI plug-in for the particular RPSS server to evaluate the trust relationship and to extract the authenticated client identity. WebSphere Application Server then maps the client identity to a WebSphere Application Server security credential. For instructions on setting up TAI interceptor, refer to (Trust Association), (Configuring Trust Association interceptor), and (Example:Sample Trust Association interceptor).


Browser
Demilitarized Zone (DMZ)
Internet
Enterprise information systems
DataBase
DB2 V7.1 etc.
Internet
Protocol Firewall
Reverse Proxy Security Server (WebSeal, etc)
Domain Firewall
Web Server
WebSphere Application Server plug-in
WebSphere Application Server
TAI
MQ
CICS
Third Party Security Provider (TAM, etc)
IBM Directory (LDAP)

When configured to use LDAP user registry, WebSphere Application Server uses LDAP to perform authentication. Client id and password will be passed from WebSphere Application Server to LDAP server. WebSphere Application Server can be configured to setup SSL connection to LDAP so that passwords will not be passed in plain text. To setup SSL connection from WebSphere Application Server to LDAP server, please refer to (Configuring SSL for LDAP client). WebSphere Application Server Version 5 supports the J2EE Connector Architecture (JCA). The connector architecture defines a standard interface for WebSphere Application Server to connect to heterogeneous Enterprise Information Systems (EIS). Examples of EIS includes database systems, transaction processing such as CICS, and messaging such as Message Queue (MQ). The EIS implementation may perform authentication and access control to protect business data and resources. Authentication with EISs is performed by Resource Adapters. The authentication data may be provided either by application code or by WebSphere Application Server. WebSphere Application Server provides a principal mapping plug-in point. A principal mapping module plug-in maps the authenticated client principal to a password credential, that is, user ID and password, for the EIS security domain. WebSphere Application Server ships a default principal mapping module which simply maps any authenticated client principal to a configured pair of user id and password.

Each connector may be configured to use a different set of id and password. For description on how to configure JCA principal mapping user id and password, please refer to (Managing J2C Authentication Data Entries). A principal mapping module is basically a special purpose Java Authentication and Authorization Service (JAAS) Login Module. You can develop your own principal mapping module to fit your particular business application environment. For detailed step by step instruction on developing and configuring a custom principal mapping module, please refer to (Developing your own Java 2 security mapping module underneath JAAS Programmatic Login), (Managing Java Authentication and Authorization Service (JAAS) Login Configuration).

Interoperability issues for security

To have interoperability of Security Authentication Service (SAS) between C++ and WebSphere Application Server, use the CSIv2 authentication protocol over RMI-IIOP. To have interoperability of SAS between WebSphere Application Server and WebSphere for z/OS use the SAS authentication protocol over RMI-IIOP.

For more information on these topics, refer to the articles that describe how to interoperate with C++ CORBA clients and how to interoperate with WebSphere product for z/OS.

Interoperability with C++ CORBA client support and limitations

In addition to the WebSphere base installation, you can choose from two types of C++ CORBA client supports, IBM WebSphere Application Server Enterprise V5 or WebSphere Application Server Client V5. If you plan to develop or rebuild your own C++ client applications, then the Enterprise version is required. It installs tools, libraries, and includes for the build environment in selecting C++ CORBA Client SDK. Otherwise, a client installation suffices to run your C++ client applications with security. In Version 5, WebSphere Application Server supports the C++ Corba client on the Windows 2000, Windows NT, Solaris, Linux, and AIX operating systems.

SSLv2 cipher suites are not supported. In V5, only most commonly used ciphers among JSSE and GSkit are supported.

Since the WebSphere Enterprise CORBA C++ Client has only implemented security on the transport layer, other authentication mechanisms such as userid and password (*Basic Authen*) are not supported.

Interoperating with C++ CORBA Client

Interoperability of Security Authentication Service between the C++ CORBA Client and WebSphere Application Server can be achieved by using CSIv2 authentication protocol over RMI-IIOP. The CSIv2 security service protocol has authentication, attribute and transport layers. Among the three layers, Transport Authentication is conceptually simple, however, cryptographically based transport authentication is strongest. WebSphere Application Server Version 5 Enterprise has implemented the transport authentication layer, so that C++ secure CORBA Client can use it effectively in making CORBA Clients and protected enterprise bean resources work together.

Security Authentication from non-java based C++ client to Enterprise JavaBeans. WebSphere Application Server 5 is adding security support in CORBA C++ Client to access protected enterprise beans. If configured, this support will allow C++ CORBA Client to access protected enterprise beans methods using client certificate to achieve mutual authentication on the WebSphere Application Server Enterprise applications.

To support the C++ CORBA Client to access the protected enterprise bean, perform the following steps:

Steps for this task

1. Obtain a valid certificate to represent the client and export its public key to Target enterprise bean server.

   First, a valid certificate is needed to represent the C++ client. This can be done by requesting certificate from the CA or creating a self signed certificate for testing purpose.

   Then, use IBM Key Management Utility from GSkit to extract the public key from the personal certificate and save in the .arm format. For details, see the related information about how to (extract personal certificate’s public key).
2. Prepare Truststore file for WebSphere Application Server.

   Add the extracted client public key in `.arm` from client to server’s Key Trust Store file. By doing so, the server can now be able to authenticate the client.

   Note: This is done by invoke IBM KEY Management Utility through ikeyman.bat or ikeyman.sh from WebSphere Application Server installation.For details, see the article on (Adding truststore files).
3. Configure WebSphere Application Server to support SSL as authentication mechanism.
   1. Start the administration console.
   2. Locate the application server has the Target enterprise bean deployed and configure it to use SSL client certificate authentication.

      If it is a base installation, go to the **Security** > **Authentication Protocol** > **CSIv2 Inbound Authentication** then check **Supported** for Basic Authenticate and Client Certificate Authentication and leave the rest as default. Also, go to the CSIv2 Inbound Transport and make sure **SSL-Supported** is checked.

      If it is a Network Deployment setting, go to the **Server** > **Application Server** > *server_name_where_EJB_resides* > **Server Security** > **CSI Authentication Inbound**. Then check Supported for ″Basic Authenticate″ and ″Client Certificate Authentication.″ Leave the rest as default. Also, go to the **CSI Transport** > **Inbound** to make sure ″SSL-Supported″ is checked.

For details, see the security InfoCenter articles (Configuring CSIv2 inbound authentication) and (Configuring CSIv2 inbound transport).

c. Restart Application Server.

By doing so, the WebSphere Application Server is ready to take C++ CORBA security Client and mutually authenticate server and client by using SSL in transport layer.

4. Configure C++ COBRA Client to use certificate to perform mutual authentication.

Client users are accustomed to using property files in their applications because they are handy in specifying configuration settings. The most important C++ security settings are as follows:

(Lines in the following **C++ security settings** have been split for publication.)

C++ security setting	Description
com.ibm.CORBA.bootstrapHostName= ricebella.austin.ibm.com	This will tell the target host name.
com.ibm.CORBA.securityEnabled=yes	This will enable security.
com.ibm.CSI.performTLClientAuthentication Supported=yes	This will ensure client is supporting mutual authentication by certificate
com.ibm.CSI.performTransportAssocSSLTL SSupported=yes	This will ensure SSL to be used not TCP/IP
com.ibm.ssl.keyFile=C:/ricebella/etc/DummyKey RingFile.KDB	This will tell which KDB key data base file to be used.
com.ibm.ssl.keyPassword=WebAS	Password for opening the KDB key file. WebSphere Application Server supports a utility called `PasswordEncode4cpp` to encode the plain password.
com.ibm.CORBA.translationEnabled=1	This will enable the valueType conversion.

To use the property files in running C++ client, a environment variable WASPROPS is used to indicate where is a property file or a list of property files.

For details of how to use CORBA client property file, see Topic under CORBA Client security properties Target server signer’s public key chain is also added onto client key data base file. See the article, Adding a certificate into a key database.

For the complete set of C++ client property, see sample property file ″scclient.props″ which is shipped with the product located in $WAS_HOME\etc

Interoperating with previous product versions

Before you begin

WebSphere Application Server Version 5 interoperates with the previous product versions (such as Version 4 and Version 3.5). Interoperability can be achieved only when LTPA authentication mechanism and LDAP user registry is used. This is because, credentials derived from SWAM authentication mechanism are not forwardable.

Steps for this task

1. Enable security with LTPA authentication mechanism and LDAP user registry. Make sure that the same LDAP user registry is shared by all the product versions.
2. Extract and add Version 5 server certificates into the server key ring file of the previous version.
   a. Open the Version 5 server keyring file using the IKEYMAN tool, and extract the server certificate to a file.
   b. Open server keyring of the previous product version, using IKEYMAN and add the certificate extracted from product Version 5.
3. Extract and add Version 5 trust certificates into the trust keyring file of the previous product version.
   a. Open the Version 5 trust keyring file using the IKEYMAN tool, and extract the trust certificate to a file.
   b. Open the trust keyring file of the previous product version, using the IKEYMAN tool, and add the certificate extracted from Version 5.
4. If single sign-on (SSO) is enabled, export keys from Version 5 product, and import them into the previous product version. The Version 4 product requires the fix, PQ61779, and the Version 3.5 product requires the fix, PQ59667, for SSO to function.
5. In Version 5, the enterprise beans are registered with long JNDI names like, `(top)/nodes/<node_name>/servers/<server_name>/HelloHome`. Whereas in previous releases, enterprise beans are registered under a root like, `(top)/HelloHome`. Therefore, EJB applications from previous versions perform a lookup on the Version 5 enterprise beans, ensure that the application uses the correct JNDI name. This problem can also be solved by creating EJB name bindings in Version 5 that are compatible with the previous version. To create an EJB name binding at the root Version 5, start the administrative console and click **Environment** > **Naming** > **Naming Space Bindings** > **New** > **EJB** > **Next**. Complete all the fields and enter a short name (for example, `-HelloHome`) as the JNDI Name. Click **Next** and **Finish**.
6. Stop and restart all the servers.
7. Also make sure that the correct naming bootstrap port is used to perform naming lookup. In previous product versions, the naming bootstrap port is **900**. In Version 5, the bootstrap port is **2809**.

Interoperating with WebSphere Application Server z/OS (Security Authentication Service)

Interoperability of Security Authentication Service between the WebSphere Application Server and WebSphere Application Server for z/OS can be achieved by using SAS authentication protocol over RMI-IIOP. The SAS protocol propagate the application server's security context over SSL for authentication taken place at z/OS before accessing the z/OS protected resources.

WebSphere Application Server v5 continues to support the interoperability of security authentication service with WebSphere Application Server z/OS. If configured, this support allows only secure WebSphere Application Servers to access protected z/OS targets using SAS authentication protocol with SSL. To configure an application server to log on and access resources on a secure z/OS server, perform the following steps:

Steps for this task

1. Create a login key file and add log-in information.

2. Prepare Truststore file for the WebSphere Application Server.
3. Configure the WebSphere Application Server.
4. Configure the WebSphere Application Server z/OS Application Server.

   For details, see WebSphere Application Server for z/OS and OS/390: Installation and Customization (GA22-7834).

Creating login key files

Steps for this task

1. First, a login key file must be created.

   The authenticating user IDs and passwords are specified in the login key file. Login information (target realm, user ID, and password) for each different z/OS target must be stored in the login key file, which is an ASCII file. When the security authentication service processes the login key file, the passwords in the file are encoded.
2. Add information to the login key file in the following format:

   ```
   RealmName   UserID   Password
   ```
3. Make sure that the data conforms to the following rules:
   - One realm name
   - One user ID, and one password defined in each entry
   - One entry per line
   - No blank lines between entries
   - Comments on separate lines only
   - Begin any comment with a pound sign (#) Example:

     ```
     Example:

     # Sample key file
     #
     # First target realm
     #
     TargetRealm serverID serverPassword
     #
     # Second target realm
     #
     TargetRealm2 serverID2 serverPassword2
     #
     # End of key file
     ```

   The realm name of a WebSphere Application Server z/OS target is the IP name of the daemon as specified in the configuration of the WebSphere Application Server z/OS product. The user ID and password are as defined for secured WebSphere Application Server z/OS servers.

   A sample file named wsserver.key also contains these instructions. After installation, this sample file can be found in install_root/properties directory. You may use or modify the sample file as needed for testing.

   Note: The login key file can be placed anywhere on a host machine running the application server. However, it is recommended that you place the login key file under a securable file system (for example, NTFS for Windows NT).

What to do next

Prepare truststore files for the WebSphere Application Server.

Preparing truststore files

Secure Socket Layer (SSL) protocol is used to protect the communication between WebSphere Application Server and WebSphere Application Server z/OS application servers. In order to complete the SSL connection, a valid truststore file for the WebSphere Application Server must be established. A truststore file is a key database file that contains the public keys (See related information about how to create a new keystore file.)

Steps for this task

1. Extract the public key of the z/OS server by using the key management tool from WebSphere Application Server z/OS.

   For details, see WebSphere Application Server for z/OS and OS/390: Installation and Customization (GA22-7834).
2. With the iKeyman tool from WAS, add the public key from the WebSphere Application Server z/OS server as a signer certificate into the requesting WebSphere Application Server's truststore file.

   For details, see the related information about how to ("Importing signer certificates").

Results

The WebSphere Application Server truststore file is now ready to use for SSL connection with the WebSphere Application Server z/OS servers.

What to do next

Configure the WebSphere Application Server for interoperability.

Configuring the application server for interoperability

After the truststore file has been ready, you must do the followings to configure the WebSphere Application Server.

Steps for this task

1. Configure your enterprise beans that access the WebSphere Application Server z/OS. Before deploying the enterprise beans, RunAs Identity must be configured. Since the security authentication service only supports WebSphere Application Servers to interoperate with WebSphere Application Server z/OS, set the RunAs Identity to `System Identity`.
2. Enable Security.
3. Enable outbound SAS authentication protocol.
4. Specify the truststore file in a SSL configuration alias and configure the WebSphere Application Server with that alias.
5. Set the Request timeout and Locate request timeout to 0 for the Object Request Broker service.

   When the WebSphere Application Server z/OS application server is first started, no server region is available for processing work. It is therefore recommended that you set these two properties to 0 in order to prevent potential timeouts.
6. Specify a security property named com.ibm.CORBA.keyFileName for the absolute path of the login key file created earlier. See related information about how to configure for a custom security property.
7. Restart the WebSphere application server.

Security: Resources for learning

Use the following links to find relevant supplemental information about Securing applications and their environment. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:
- Planning, business scenarios and IT architecture
- Programming model and decisions
- Programming specifications
- Administration

Planning, business scenarios and IT architecture

- **WebSphere Application Server Library** (http://www-3.ibm.com/software/webservers/appserv/library.html)
- **WebSphere Application Server Support** (http://www-3.ibm.com/software/webservers/appserv/support.html)

Programming model and decisions

- JSSE Documentation.

  Look in {was_install_root}/web/docs/jsse/jssedocs.jar for the javadoc of the APIs. This document contains the following when unpacked: index.html (Developer's Guide (JSSE JavaDoc))
  - For the JSSE Reference Guide look in {was_install_root}/web/docs/jsse/JSSERefGuide.html
  - For sample JSSE applications look in {was_install_root}/web/docs/jsse/samplejsse.jar
- iKeyman Documentation.

  Look in {was_install_root}/web/docs/ikeyman/ikmuserguide.pdf for the SSL Introduction and iKeyman.
- JCE Documentation.
  - For the JCA spec and JCE API usage refer to `<install_root>/web/docs/jce/api_users_guide.html`.
  - For JCE sample applications refer to `<was_install_root>/web/docs/jce/SampleJCE.jar`.
  - For Java Cryptography Architecture Reference refer to `<install_root>/web/docs/jce/CryptoSpec.html`.
  - For how to implement a JCE provider refer to `<install_root>/web/docs/jce/HowToImplAProvider.html`.
  - For the javadoc of JCE APIs refer to `<install_root>/web/docs/jce/jcedocs.jar`.
  - For overview of IBM JCE refer to `<install_root>/web/docs/jce/readme.jce.ibm.html`.
- Application Assembly Tool (AAT) Documentation.

Refer to `<was_install_root>/web/docs/aat/en/index.html` for AAT documentation. This can help when securing J2EE enterprise applications.

- **Java 2 Platform Security for JDK 1.3 http://java.sun.com/j2se/1.3/docs/guide/security/index.html**

Programming specifications

- **J2EE Specifications http://java.sun.com/j2ee/download.html**
- **EJB Specifications http://java.sun.com/products/ejb/docs.html**
- **Servlet Specifications http://java.sun.com/products/servlet/download.html**
- **Common Secure Interoperability Version 2 (CSIv2) Specification http://www.omg.org/technology/documents/corba_spec_catalog.htm#CSIv2**
- JAAS Specification.

  For programming and usage in JAAS, refer to the specification located at `{was_install_root}/web/docs/jaas/JaasDocs.jar`. This document contains the following when unpacked:
  – login.html - LoginModule Developer's Guide
  – api.html - Developer's Guide (JAAS JavaDoc)
  – HelloWorld.tar - Sample JAAS Application
- **Java 2 Platform, Standard Edition, v1.3.1 API Specification http://java.sun.com/j2se/1.3/docs/api/index.html**

Administration

- **WebSphere Application Server Version 4.0 Security Redbook: WebSphere Security Model http://www.redbooks.ibm.com**
- **IBM HTTP Server Support and Documentation http://www-3.ibm.com/software/webservers/httpservers/support.html**
- **IBM Directory Server Support and Documentation http://www-3.ibm.com/software/network/directory/support/**
- (IBM JDK Readme)
  – For JDK Developer's Kit Readme refer to `{was_install_root}/java/docs/readme.devkit.ibm.html`
  – For JDK Installation and Configuration Readme refer to `{was_install_root}/java/docs/readme.install.ibm.html`
- **IBM cryptographic hardware devices http://www-3.ibm.com/security/cryptocards/html/library.shtml**