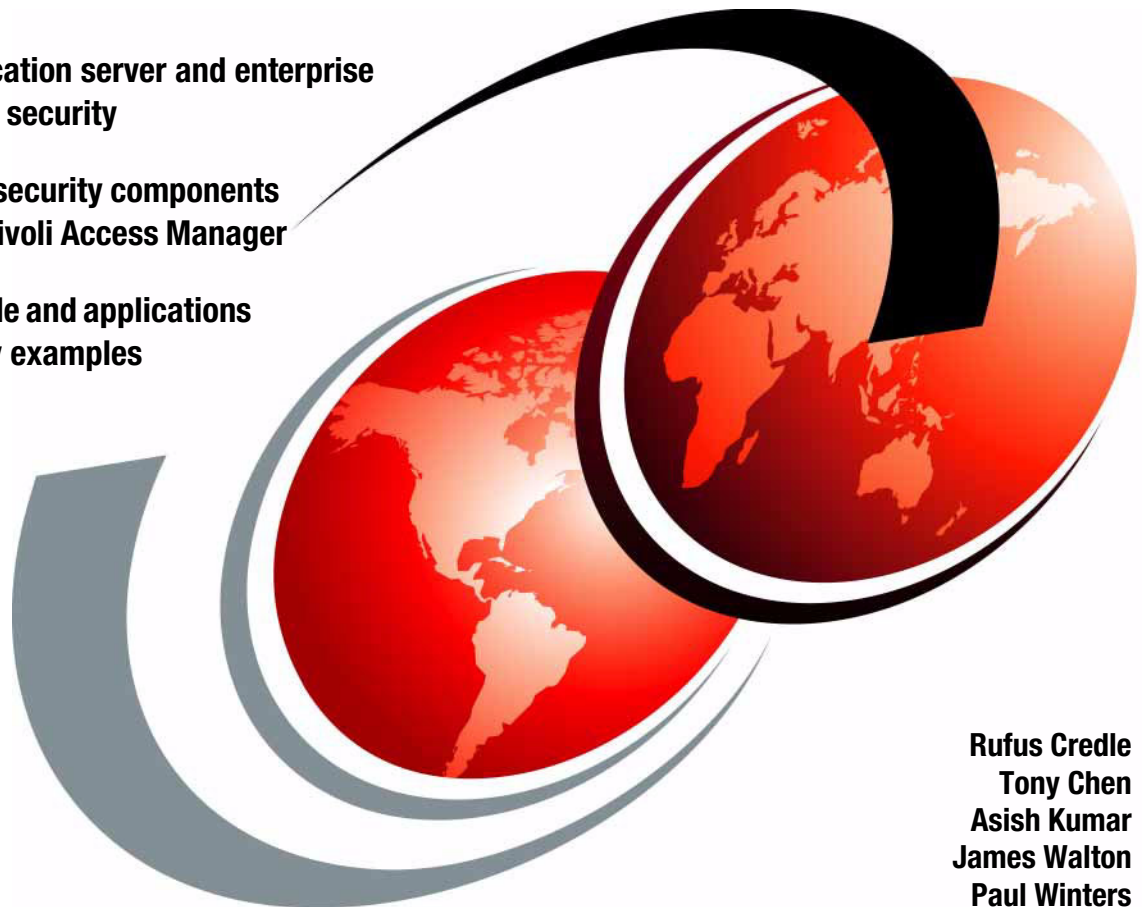


IBM WebSphere Application Server V6.1 Security Handbook

J2EE application server and enterprise application security

Additional security components including Tivoli Access Manager

Sample code and applications for security examples



Rufus Credle
Tony Chen
Asish Kumar
James Walton
Paul Winters



International Technical Support Organization

**WebSphere Application Server V6.1
Security Handbook**

December 2006

Note: Before using this information and the product it supports, read the information in “Notices” on page xiii.

Second Edition (December 2006)

This book was updated on June 15, 2009.

This edition applies to WebSphere Application Server V6.1 (base) on IBM AIX V5.2, Red Hat Enterprise Linux V3, Microsoft Windows 2000; WebSphere Application Server V6.1 Network Deployment on IBM AIX V5.2, Red Hat Enterprise Linux V3, Windows 2000; and Tivoli Access Manager V5.1 on IBM AIX V5.2, Red Hat Enterprise Linux V3, and Windows 2000.

© Copyright International Business Machines Corporation 2005, 2006. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	xiii
Trademarks	xiv
Preface	xv
The team that wrote this IBM Redbook	xvi
Become a published author	xix
Comments welcome	xix
Part 1. Application server security	1
Chapter 1. Introduction to this book	3
1.1 A focus on security	4
1.2 Scenario-based chapters	4
1.3 Sample applications	5
1.4 WebSphere Information Center	5
Chapter 2. Configuring the user registry	7
2.1 User registries and repositories	8
2.2 Stand-alone LDAP registry	10
2.2.1 Stand-alone LDAP registry for WebSphere Application Server V6.1	14
2.2.2 Configuring the advanced LDAP user registry	19
2.3 Local OS registry	23
2.3.1 Configuring WebSphere Application Server V6.1	24
2.3.2 Stand-alone custom registry	26
2.4 Federated repository	39
2.4.1 Connecting WebSphere Application Server to a federated repository	40
2.4.2 Configuring supported entity types in a federated repository	42
2.4.3 Configuring an entry mapping repository in a federated repository	43
2.4.4 Configuring a property extension repository in a federated repository	44
Chapter 3. Administrative security	49
3.1 Enabling administrative security	50
3.1.1 Main components of WebSphere security	51
3.1.2 Security Configuration Wizard	53
3.1.3 Other security properties	55
3.1.4 Stopping the application server	57
3.2 Disabling administrative security	58
3.3 Administrative roles	59
3.3.1 Mapping a user to an administrative role	61

3.3.2 Mapping a group to an administrative role	62
3.3.3 Fine-grained administrative security	63
3.4 Naming service security: CosNaming roles.	64
3.4.1 Mapping a user or a group to a CosNaming role	65
3.4.2 Applying CosNaming security: An example	65
Chapter 4. SSL administration and configuration management	69
4.1 Creating a new SSL key store entry	70
4.2 Managing SSL certificates.	74
4.2.1 Expiring certificates.	74
4.2.2 Managing Web server and plug-in certificates	74
4.3 Creating a new SSL configuration.	76
4.4 Additional SSL configuration attributes	78
4.4.1 Federal Information Processing Standard.	78
4.4.2 Dynamic SSL configuration updates.	78
4.5 Trust managers	79
4.5.1 Custom trust managers.	80
4.6 Key managers	83
4.6.1 Custom key managers	83
Chapter 5. JAAS for authentication in WebSphere Application Server	85
5.1 The importance of JAAS	86
5.2 JAAS in WebSphere	86
5.3 Custom JAAS login in WebSphere	88
5.3.1 Callback handler	88
5.3.2 Login module.	89
5.3.3 Principal	95
5.3.4 Configuration.	97
5.3.5 Viewing the sample JAAS module in action	99
5.3.6 Programming authentication	99
5.4 J2C authentication data.	99
Chapter 6. Application security	101
6.1 Application security	102
6.1.1 Enabling application security.	102
6.1.2 Testing application security	103
6.1.3 Application considerations	103
6.2 Deploying a secured enterprise application	105
6.2.1 Role mapping during application installation.	105
6.2.2 Role mapping after installation	106
Chapter 7. Securing a Web application	109
7.1 Transport channel	110
7.2 Securing the static content	110

7.2.1	Securing the transport channel between the Web browser and Web server	111
7.2.2	Authentication by using a Web server.	113
7.2.3	Authorization by using a Web server	116
7.3	Securing the Web server plug-in for WebSphere	117
7.3.1	Securing the transport channel between the Web server and WebSphere.	118
7.3.2	Testing the secure connection	124
7.4	Securing the Web container of the application server.	126
7.4.1	Securing the transport channel.	126
7.4.2	Authentication by using the Web container.	127
7.4.3	Authorization by using the Web container.	132
7.4.4	Programmatic security	141
7.5	Additional transport security, authentication, and authorization options	147
7.5.1	Configuring LDAP authentication with IBM HTTP Server	147
7.5.2	Configuring SSL certificate-based client authentication for the IBM HTTP Server.	152
7.5.3	Configuring SSL certificate-based client authentication for WebSphere Application Server.	156
Chapter 8. Securing an EJB application		171
8.1	Programmatic login (server-side) using JAAS.	173
8.2	Declarative J2EE security	174
8.2.1	Defining J2EE security roles for EJB modules	174
8.2.2	Security role references	175
8.2.3	Configuring method access control.	180
8.2.4	Enterprise JavaBeans Run-As delegation policy	186
8.2.5	Bean-level delegation	186
8.2.6	Method-level delegation	190
8.2.7	Run-as mapping	193
8.3	Programmatic J2EE security	197
8.4	EJB container access security	199
8.4.1	CSIV2 and Secure Authentication Service	199
8.4.2	Container authentication	200
8.4.3	RMI/IIOP transport channel protection	204
Chapter 9. Client security		207
9.1	Application clients in WebSphere	208
9.1.1	Developing and securing the J2EE application client	209
9.1.2	Deploying an application client by using the Java Web Start tool.	209
9.1.3	Thin application client	213
9.1.4	Itsosello client example.	214
9.2	Java client authentication protocol	216

9.2.1 CSIV2 Security Attribute Service	217
9.2.2 Authentication process	218
9.3 Java client configuration	220
9.4 J2EE application client	225
9.4.1 Itsohello unsecure J2EE client	225
9.4.2 Itsohello secure J2EE client	227
9.5 Thin application client	228
9.5.1 Running a thin application client	230
9.5.2 Itsohello unsecure thin client	231
9.5.3 Itsohello secure thin client	233
9.6 Programmatic login	233
9.6.1 JAAS login module in WebSphere	233
9.6.2 Programmatic login process	235
9.6.3 Client-side programmatic login using JAAS	236
9.7 Securing the connection	242
9.7.1 IIOP over SSL: A thin client example	242
Chapter 10. Securing the service integration bus	247
10.1 Messaging components of the service integration bus	248
10.1.1 Service integration bus	249
10.1.2 Messaging engine	249
10.1.3 Foreign bus	250
10.1.4 Bus destination	250
10.2 An overview of service integration bus security	250
10.2.1 Authentication	251
10.2.2 Authorization	251
10.2.3 Transport security: Confidentiality	253
10.3 Administering service integration bus security	253
10.3.1 Administering the Bus Connector role in the Administrative Console .	254
10.3.2 Administering the Bus Connector role by using the wsadmin tool	256
10.4 Administering destination security	257
10.4.1 Default roles for bus destinations	257
10.4.2 Destination specific roles	258
10.5 Administering topic space root roles and topic roles	259
Part 2. Extending security beyond the application server	263
Chapter 11. Security attribute propagation	265
11.1 Initial Login versus Propagation Login	267
11.2 Token framework	268
11.3 Custom implementation of tokens	270
11.3.1 Writing custom implementations of tokens	271
11.3.2 Common token functionality	272

11.3.3	Interaction of the login module and the token modules	275
11.3.4	Authorization token	276
11.3.5	Single Sign-On token	279
11.3.6	Propagation token	280
11.3.7	Authentication token	283
11.3.8	Changing the token factory associated with the default token . . .	283
11.4	Horizontal propagation	285
11.4.1	Horizontal propagation using Dynacache	285
11.4.2	Horizontal propagation using JMX	286
11.5	Downstream propagation	289
11.5.1	Downstream propagation scenario	290
11.6	Enabling security attribute propagation	292
11.6.1	Configuring security attribute propagation for horizontal propagation	292
11.6.2	Enabling downstream propagation	293
11.7	Advantages of security attribute propagation	295
 Chapter 12. Securing a WebSphere application using Tivoli Access Manager		
12.1	Introduction to Tivoli Access Manager	297
12.1.1	Benefits	298
12.1.2	When to use Tivoli Access Manager for e-Business in conjunction with WebSphere Application Server	299
12.1.3	Reverse proxies for authentication	301
12.1.4	Access Manager Secure Domain	301
12.1.5	Tivoli Access Manager auditing	305
12.1.6	Access Manager and WebSphere integration	306
12.1.7	Reverse proxy authenticators and the extended WebSphere trust domain	309
12.1.8	Challenges with reverse proxy authenticators	309
12.2	IBM Tivoli Access Manager security model	315
12.2.1	User registry	315
12.2.2	Master authorization (policy) database	316
12.3	Summary of Access Manager deployment for integration with WebSphere Application Server	320
12.4	Lab environment	320
12.5	The role of Tivoli Access Manager inside WebSphere Application Server V6.1	321
12.5.1	Embedded Tivoli Access Manager client architecture	323
12.5.2	High-level components of the integration	325
12.6	WebSEAL authentication	327
12.6.1	Basic authentication	327
12.6.2	Form-based authentication	328

12.6.3	Client certificate-based authentication	329
12.6.4	Token authentication	331
12.6.5	HTTP header authentication	332
12.6.6	Kerberos and SPNEGO authentication	332
12.6.7	External authentication interface	333
12.6.8	Combining authentication types using step-up authentication . . .	333
12.7	WebSEAL junctions	334
12.7.1	Simple junctions	335
12.7.2	Trust Association Interceptors and LTPA Junctions	338
12.7.3	Single sign-on junctions	339
12.8	Integrating IBM WebSphere Application Server and Tivoli Access Manager	344
12.8.1	aznAPI	345
12.8.2	Tivoli Access Manager and J2EE security	345
12.8.3	Embedded Tivoli Access Manager in WebSphere Application Server V6.1	346
Chapter 13. Trust Association Interceptors and third-party software integration		353
13.1	Trust Association Interceptor	354
13.1.1	The relatively new, enhanced TAI interface	355
13.2	Windows desktop single sign-on using SPNEGO	356
13.2.1	Lab scenario	358
13.2.2	Configuring the WebSphere Application Server environment to use SPNEGO	359
13.2.3	Troubleshooting SPNEGO environments	376
13.3	IBM WebSphere Application Server and WebSEAL integration	378
13.3.1	Integration options	378
13.3.2	Configuration for the Trust Association Interceptor approach . . .	380
13.3.3	Configuration for the LTPA approach	396
13.3.4	Security considerations	402
Chapter 14. Externalizing authorization with JACC		403
14.1	Deployment tools contract	405
14.2	Container contract	407
14.3	Provider contract	408
14.4	Why JACC	408
14.5	JACC in WebSphere Application Server V6.1	408
14.5.1	JACC access decisions in WebSphere Application Server V6.1 . .	410
14.5.2	JACC policy context identifiers in WebSphere Application Server V6.1	414
14.5.3	WebSphere extensions to the JACC specification	414
14.5.4	JACC policy propagation in WebSphere Application Server V6.1	415

14.5.5	Manual policy propagation	418
14.5.6	Dynamic module updates in WebSphere Application Server V6.1 for JACC	420
14.6	Integrating Tivoli Access Manager as an external JACC provider	420
14.6.1	Disabling the embedded Tivoli Access Manager	426
14.6.2	Reconfiguring the JACC provider by using wsadmin	427
14.7	Sample application for JACC	427
Chapter 15.	Web services security	429
15.1	Web services security exposures	430
15.2	WS-Security	432
15.2.1	WS-Security concepts	433
15.2.2	Evolution of the WS-Security specification	434
15.2.3	WS-Security roadmap	436
15.2.4	Example of WS-Security	437
15.2.5	Development of WS-Security	442
15.2.6	Hardware cryptographic device support for WS-Security	444
15.3	Transport-level security	447
15.3.1	SOAP over HTTP transport-level security	447
15.4	WS-I Basic Security Profile	448
15.5	Summary	449
15.6	More information	449
Chapter 16.	Securing access to WebSphere MQ	451
16.1	Application server and WebSphere MQ	452
16.1.1	WebSphere MQ messaging components	452
16.1.2	Authentication	454
16.1.3	Authorization	455
16.1.4	Transport security	456
16.1.5	Administering foreign service integration bus security	458
16.1.6	Administering WebSphere MQ security	459
16.2	Sample application	460
16.3	Additional information	461
Chapter 17.	J2EE Connector security	463
17.1	The J2EE Connector Architecture	464
17.1.1	Connector security architecture	465
17.2	Securing the J2EE Connector	466
17.2.1	Component-managed authentication	466
17.2.2	Container-managed authentication	468
17.3	JCA authentication mechanism	470
17.3.1	Role-based authorization	471
17.3.2	Topic security	471
17.3.3	Messaging security	471

17.3.4	Enable bus security	472
17.3.5	Inter-engine authentication alias	472
17.4	Mediations security	474
17.5	Transport security in service integration bus	474
17.5.1	Destination security	476
17.6	Securing Web services by using service integration technologies	476
17.7	Additional information	477
Chapter 18.	Securing the database connection	479
18.1	Securing the connection	480
18.1.1	JDBC type 2 driver	482
18.1.2	JDBC type 4 driver	483
18.2	Securing access to database data	483
Part 3.	Development environment	485
Chapter 19.	Development environment security	487
19.1	Rational Application Developer	488
19.1.1	Securing the workspace	488
19.2	WebSphere test environment	491
19.2.1	Creating a new test server	492
19.2.2	Enabling security for the WebSphere Test Server V6.1	494
19.3	Administering and configuring the WebSphere test servers	496
19.4	Enterprise application security	497
19.4.1	Configuring enterprise application security during the development phase	497
19.4.2	JAAS entries in the deployment descriptor	500
19.5	Creating a new profile for the WebSphere test server	501
19.5.1	Advantages of multiple profiles	501
19.5.2	Creating a new profile	502
19.6	Application Server Toolkit 6.1	506
Appendix A.	Additional configurations	509
	Sample application for client security	510
	Installing and testing Itsohello application	511
	Sample application for testing JACC	513
	Web module	514
	EJB module	514
	Deploying the sample application	514
	Verifying the installation	515
	Testing the application installation	516
	Configuring the service integration bus and default messaging provider	516
	Creating a service integration bus	517
	Adding an application server or server cluster to the bus	518

Defining a queue destination on the bus	520
Defining a JMS connection factory	522
Defining a JMS queue	525
Configuring WebSphere MQ as a foreign bus	526
Defining a foreign bus	527
Defining an MQ link	528
Defining a foreign destination	529
Defining a JMS queue for a foreign destination	530
Sample application for messaging	530
Configuring the application server	531
Optional: Configuring WebSphere MQ	536
Installing the sample application	537
Testing the sample application	538
Appendix B. Additional material	543
Locating the Web material	543
Using the Web material	543
System requirements for downloading the Web material	544
How to use the Web material	544
Abbreviations and acronyms	545
Related publications	547
IBM Redbooks	547
Other publications	547
Online resources	547
developerWorks	548
How to get IBM Redbooks	549
Help from IBM	549
Index	551

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

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

AIX®	DRDA®	Redbooks (logo)  ®
BladeCenter®	eServer™	System x®
ClearCase®	IBM®	System z®
DataPower®	Lotus®	Tivoli®
DB2 Universal Database™	OS/400®	WebSphere®
DB2®	RACF®	xSeries®
developerWorks®	Rational®	z/OS®
Distributed Relational Database Architecture™	RDN®	zSeries®
Domino®	Redbooks®	
	Redpapers™	

The following terms are trademarks of other companies:

SUSE, the Novell logo, and the N logo are registered trademarks of Novell, Inc. in the United States and other countries.

Interchange, and the Shadowman logo are trademarks or registered trademarks of Red Hat, Inc. in the U.S. and other countries.

EJB, Enterprise JavaBeans, J2EE, J2SE, Java, JavaBeans, JavaMail, JavaScript, JavaServer, JDBC, JDK, JMX, JNI, JRE, JSP, JVM, Prism, Solaris, Sun, Sun Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Active Directory, ActiveX, Expression, Internet Explorer, Microsoft, Visual Basic, Windows Server, Windows, 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.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

This IBM® Redbooks® publication is part of the IBM WebSphere® V6.1 series. It focuses on security and related topics, as well as provides technical details for designing and implementing secure solutions with WebSphere. Designed for IT architects, IT specialists, application designers, application developers, application assemblers, application deployers, and consultants, this book provides information about designing, developing, and deploying secure e-business applications using IBM WebSphere Application Server V6.1. It discusses theory and presents proven exercises performed in our lab by using sample applications.

Part 1 discusses security for the application server and its components, including enterprise applications. It focuses on administrative security and application security, which were previously known as *global security*. It includes essential information about how to secure Web and Enterprise JavaBeans™ (EJB™) applications and how to develop a Java™ client using security.

Part 2 introduces additional components from the enterprise environment and discusses security beyond the application server. External components include third-party security servers, messaging clients and servers, and database servers.

Part 3 provides a short introduction to development environment security. It includes guidelines and best practices that are applicable to a secure development environment.

This Redbooks publication provides enhancements to exercises performed in Version 6.0. In addition, this book discusses the following features in Version 6.1:

- ▶ Persistence with an authenticated identity for protected, unprotected resource
- ▶ Support for the Simple and Protected Negotiate (SPNEGO) protocol to flow Kerberos tokens from Microsoft® Internet Explorer®
- ▶ Ability to enable administrative security out-of-box (OOBE) by using the Virtual Member Manager (VMM) file registry
- ▶ Integration of VMM into WebSphere Application Server
- ▶ Simplified WebSphere Application Server key/certificate management
- ▶ Security performance through hardware crypto acceleration
- ▶ Web Services Interoperability Organization (WS-I) Basic Security Profile 1.0

The team that wrote this IBM Redbook

This IBM Redbook was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Raleigh Center.



The authors (from left): Rufus Credle, James Walton, Asish Kumar, Paul Winters, and Tony Chen

Rufus Credle is a Certified Consulting IT Specialist at the ITSO, Raleigh Center. In his role as project leader, he conducts residencies and develops Redbooks publications about network operating systems, ERP solutions, voice technology, high availability and clustering solutions, Web application servers, pervasive computing, and IBM and OEM e-business applications, all running IBM System x, IBM eServer™ xSeries®, and IBM BladeCenter®. Rufus' various positions during his IBM career have included assignments in administration and asset management, systems engineering, sales and marketing, and IT services. He holds a Bachelor of Science (BS) degree in business management from Saint Augustine's College. Rufus has been employed at IBM for 26 years.

Tony Chen is an Advisory IT Specialist at IBM Canada in Toronto. Tony has been working for IBM for over six years in IBM WebSphere technical support and financial industry application development. Prior to IBM, he worked in the software industry in Shanghai for two years. His areas of expertise include Java, Java 2 Platform Enterprise Edition (J2EE™), and WebSphere. He has several certifications from Sun™ and IBM in Java and WebSphere technologies. Tony received his bachelor degree in Computer Science from Sichuan University, China.

Asish Kumar is a Consulting IT Architect for Enterprise Architecture and Technology Group, ASEAN/SA. He has over 18 years of experience. He has been employed at IBM for four years. His expertise is on IT Architecture (J2EE, EAI, Portal), Project Management, Quality Management (SEI/CMM, ITSM/ITIL). His current focus area is high availability, security, and scalability. He holds a master degree in Mathematics from India Institute of Technology, Kharagpur, India.

James Walton is an Applications/Middleware specialist for the High Performance On Demand Services team in IBM Global Services US. He has over five years experience in Web application hosting and WebSphere Application Server administration. After joining IBM, his experience in support of production hosting environments has also included administration of WebSphere Portal, WebSphere Edge Server, and IBM HTTP Server. James holds a BS degree in Computer Science from Oklahoma Christian University. His key areas of expertise include application hosting architecture, high availability Web hosting, Web infrastructure security, and WebSphere Application Server.

Paul Winters is a software developer working with the IBM Tivoli® Security Development team on the Gold Coast, Australia. He has worked on many Tivoli Security products including Tivoli Access Manager for e-Business, IBM Tivoli Identity Manager for IBM z/OS®, and IBM Tivoli Federated Identity Manager. His areas of interest are Enterprise Security and Federated Identity Management. Paul has a Bachelor of Computer Systems Engineering degree from the University of Queensland in Australia.

Special thanks to the WebSphere Application Server V6.0 residency team, which included Peter Kovari, Emilio Bielsa, Saravana C. Chandran, Lucky Kartasasmita, Denis Masic, Sudhakar Nagarajan, Fumiko Satoh, Irina Singh, and Matthew Stokes.

Thanks to the following people for their contributions to this project:

Cecilia Bardy, Linda Robinson, Carolyn Sneed, Margaret Ticknor, and
Jeanne Tucker
ITSO, Raleigh Center

Keys Botzum, Senior Technical Staff Member, IBM Software Services for
WebSphere
IBM Bethesda

Sridhar Muppidi, IBM Software Group, Tivoli Directory and Security Architecture
IBM Austin

Peter Birk, Ching-Yun Chao, and Shengdong (Shendong) Chen, members of the WebSphere Application Server Security Development Team
IBM Austin

Kenneth Childers, Software Engineer
IBM Austin

Carlton Mason, WebSphere Application Server Development Manager
IBM Austin

Messaoud Benantar, Prism™ Project Security Lead
IBM Austin

Neil Readshaw, Chris Hockings, and Glen Gooding, members of the Tivoli Security Advanced Customer Engineering Team located on the Gold Coast
IBM Australia

Davin Holmes and Kerry Gunn, Tivoli Security Development Team
IBM Australia

Kenichiroh Ueno, WebSphere Performance
IBM Japan

Simon Chan, Senior I/T Specialist - Tech. Lead, WebSphere Application Server and Linux® on IBM System z® implementation
IBM Toronto, ON

Ajay Reddy, Technical Account Manager, Systems and Technology Group
IBM United Kingdom

Alasdair Nottingham, Service integration bus security, WebSphere Messaging Development
IBM United Kingdom

Become a published author

Join us for a two-week to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want IBM Redbooks to be as helpful as possible. Send us your comments about this or other IBM Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400



Part 1

Application server security

This part includes the following chapters:

- ▶ Chapter 1, “Introduction to this book” on page 3
- ▶ Chapter 2, “Configuring the user registry” on page 7
- ▶ Chapter 3, “Administrative security” on page 49
- ▶ Chapter 4, “SSL administration and configuration management” on page 69
- ▶ Chapter 5, “JAAS for authentication in WebSphere Application Server” on page 85
- ▶ Chapter 6, “Application security” on page 101
- ▶ Chapter 7, “Securing a Web application” on page 109
- ▶ Chapter 8, “Securing an EJB application” on page 171
- ▶ Chapter 9, “Client security” on page 207
- ▶ Chapter 10, “Securing the service integration bus” on page 247



Introduction to this book

This chapter provides a brief introduction to this book. It introduces the scenarios that are used in each chapter and gives a quick overview of how the security discussion is divided into multiple scenarios. This chapter also provides pointers that help you find your way around other WebSphere Application Server V6.1 IBM Redbooks publications.

1.1 A focus on security

The focus in this book is on security, mostly WebSphere Application Server V6.1 and Tivoli Access Manager security. This book covers the application server and other components, such as the directory server (for user registry), the reverse proxy security server, and so on.

1.2 Scenario-based chapters

In this book, the individual chapters focus on application scenarios. Instead of discussing bits and pieces or components, you can find descriptions of smaller scenarios, for example a scenario illustrating how to secure Web applications.

The diagram in Figure 1-1 gives you a general idea of the key components within WebSphere Application Server V6.1 as discussed in the following chapters.

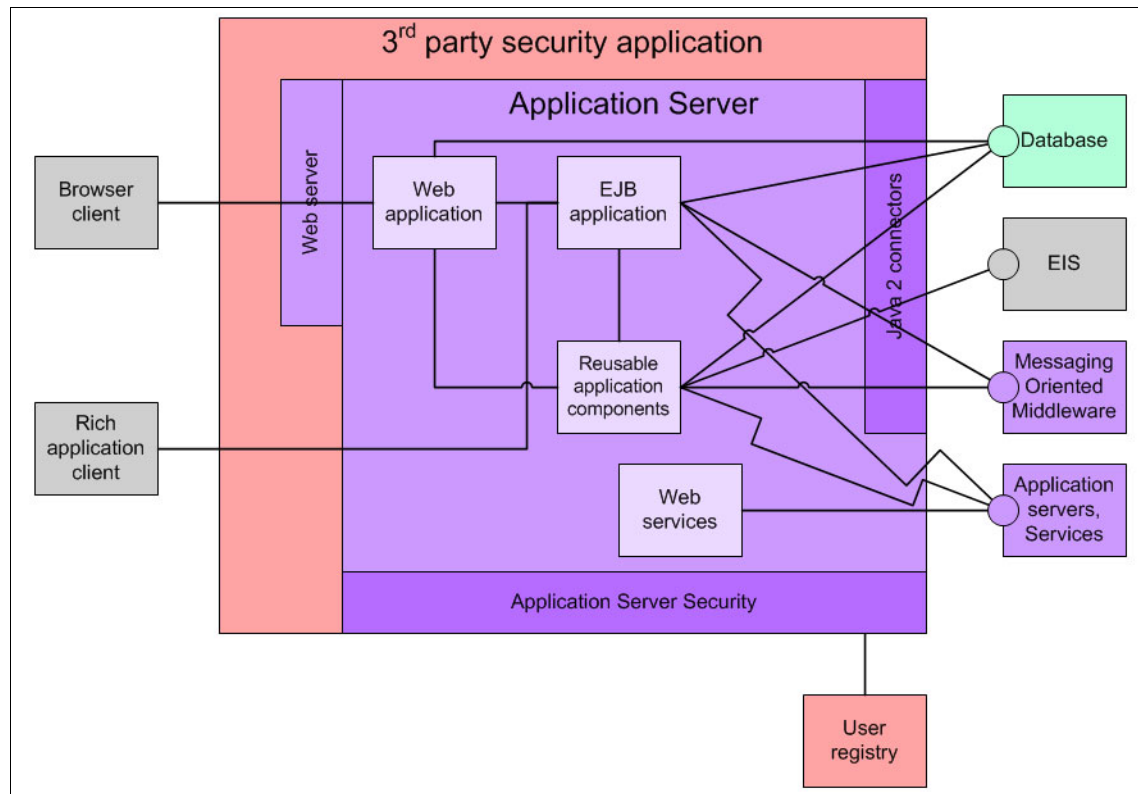


Figure 1-1 The big picture

1.3 Sample applications

Each scenario provides sample configurations and sample applications that you can try. The sample applications are available as additional material. For further information about additional materials, see Appendix B, “Additional material” on page 543.

The sample applications in this book are simple. Their purpose is to show, in practice, the theory described in the chapters. The samples are more like simple components running in small, self-contained applications.

The samples are different from previous Redbooks publications about WebSphere security. The samples in these chapters are not connected or related. Therefore, you can test each sample independently.

1.4 WebSphere Information Center

This Redbooks publication is not a replacement for the WebSphere Information Center, which is a great source of information for WebSphere Application Server V6.1. This book and the WebSphere Information Center work as complements to each other. However, note the following points:

- ▶ This book provides hands-on exercises and follows scenarios to explain the security-related tasks. The WebSphere Information Center is a tremendous reference guide for all the security-related tasks.
- ▶ This book follows a linear pattern, even though you can read only parts of the book and move back and forth. In contrast, the WebSphere Information Center contains hypertext documentation, which you can easily use to navigate between topics to find the piece of information that you need.

You can find the WebSphere Application Server V6.1 Information Center at the following address:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp>



Configuring the user registry

This chapter discusses the configuration of the user registry in WebSphere Application Server V6.1.

2.1 User registries and repositories

WebSphere Application Server V6.1 supports multiple types of registries and repositories:

- ▶ Local operating system registry
- ▶ Stand-alone Lightweight Directory Access Protocol (LDAP) registry
- ▶ Stand-alone custom registry
- ▶ Federated repositories

Information about users and groups reside in a user registry or repository. In WebSphere Application Server V6.1, a user registry or repository authenticates a user and retrieves information about users and groups to perform security-related functions, including authentication and authorization. Before configuring the user registry or repository, decide which user registry or repository to use. Although different types of registries and repositories are supported, all of the processes in WebSphere Application Server V6.1 can use one active registry.

When a user registry or repository is not configured, the local operating system registry is used by default. If your choice of user registry is not the local operating system registry, you must first configure the registry or repository, which is typically done as part of enabling security. Next, restart the servers, and then assign users and groups to roles for all your applications.

WebSphere Application Server V6.1 also provides a plug-in to support any registry by using the custom registry feature. With the custom registry feature, you can configure any user registry that is not made available through the security configuration panels of the WebSphere Application Server V6.1.

Custom registry: On occasion, although you might use supported registries, such as LDAP, you might want to implement your own custom registry for more control or when you have special situations. In general, you can use the custom registry if the default registry support is not enough.

The UserRegistry interface is used to implement both the custom registry and the federated repository options for the user account repository. The interface is helpful in situations where the current user and group information exists in some other format. The UserRegistry interface is also used for LocalOS and LDAP registries, for example, all our registries.

Figure 2-1 shows how the registry and repository fit in with Security Authentication components in WebSphere Application Server V6.1.

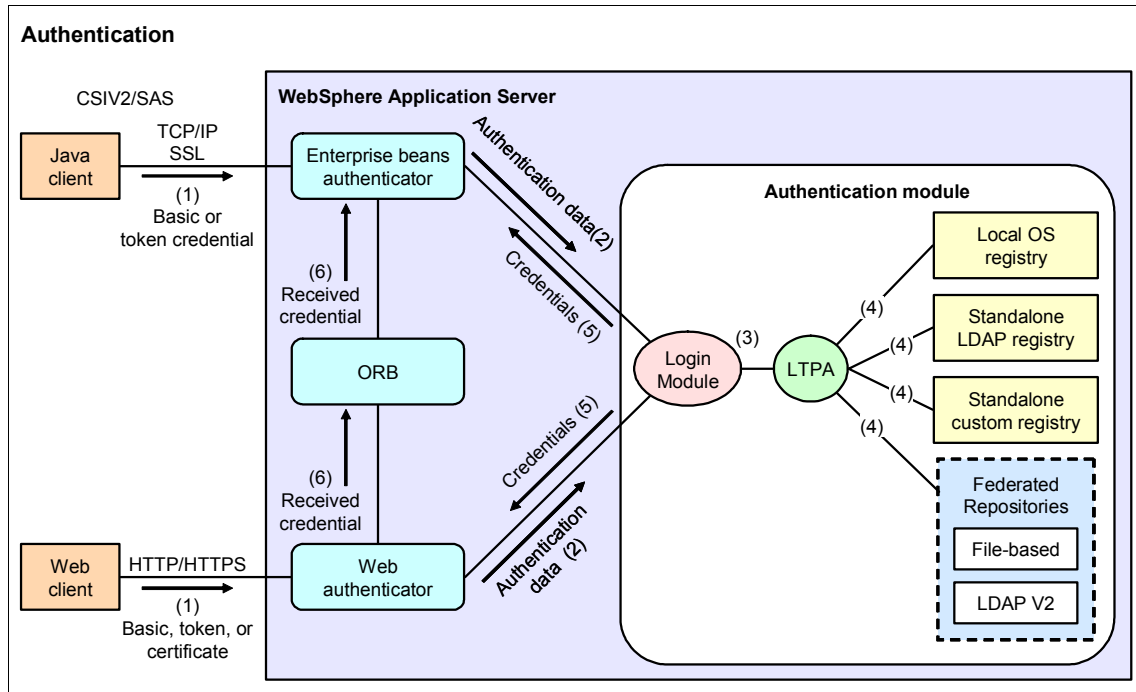


Figure 2-1 WebSphere Application Server V6.1 authentication mechanisms

Figure 2-1 shows the steps in the authentication process. Basically, authentication is required for enterprise bean clients and Web clients when they access protected resources. Enterprise bean clients (a servlet or other enterprise beans, or a pure client) send the authentication information to a Web application server by using one of the following protocols:

- ▶ Common Secure Interoperability Version 2 (CSIV2)
- ▶ Secure Authentication Service

Web clients use the HTTP or HTTPS to send the authentication information as shown in Figure 2-1. The authentication information can be basic authentication (user ID and password), a credential token, or a client certificate.

The Web authenticator and the Enterprise JavaBeans (EJB) authenticator pass the authentication data to the login module (2), which can use Lightweight Third Party Authentication (LTPA).

The Authentication module uses the registry that is configured on the system to perform the authentication (3). The following registries are supported:

- ▶ LocalOS
- ▶ Stand-alone LDAP
- ▶ Stand-alone custom registry
- ▶ Federated repositories

External registry implementation that follow the registry interface specified by IBM can replace either the LocalOS or the Stand-alone LDAP registry. The Login module creates a Java Authentication and Authorization Service (JAAS) subject after authentication and stores the credential derived from the authentication data in the public credentials list of the subject. The credential is returned to the Web authenticator or EJB authenticator (5).

The Web container Object Request Broker (ORB) is responsible for connecting Internet InterORB Protocol (IIOP) requests that contain the operation and any required parameter, and for sending the request in the network. The server receives the IIOP request, locates the target object, invokes the requested operation, and returns the result to the clients.

WebSphere Application Server V6.1 uses an ORB to manage communication between Java clients and server application and for communication among product components.

The Web authenticator and the EJB authenticator store the received credentials in the ORB for the authorization service to use in performing further access control checks.

2.2 Stand-alone LDAP registry

To use LDAP as the user registry in this chapter, we use the IBM Tivoli Directory Server V5.2 that ships with IBM Tivoli Access Manager for e-business V5.1. Figure 2-2 on page 11 shows the Directory Information Tree (DIT). IBM Directory Server V6.0 is used as the LDAP V3 server for the stand-alone LDAP user registry, which is authenticated by using an LDAP binding.

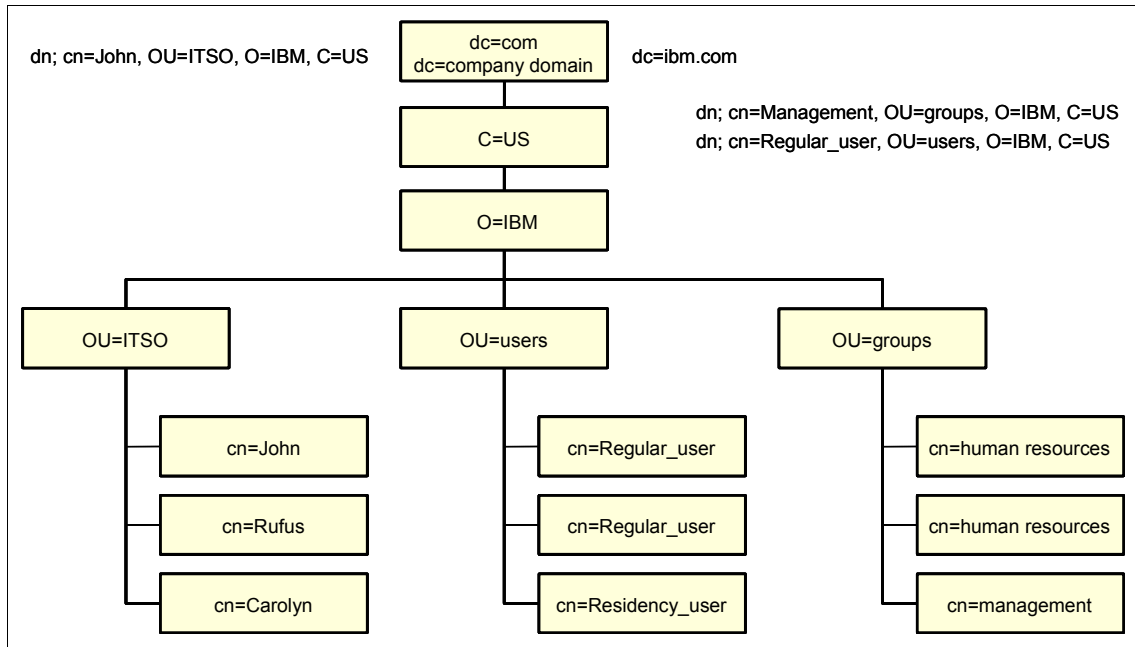


Figure 2-2 LDAP Directory Information Tree

LDAP registries naming model

The *naming model* defines how entries are identified and organized. Entries are organized in a tree-like structure called the *Directory Information Tree*. Entries are arranged within the DIT based on their *distinguished name* (DN). A DN is a unique name that unambiguously identifies a single entry. DNs are made up of a sequence of *relative distinguished names* (RDNs). Each RDN@ in a DN corresponds to a branch in the DIT that leads from the root of the DIT to the directory entry. Entries are named according to their position in the DIT.

DNs versus file system names: DNs read from leaf to root as opposed to file system names, which typically read from root to leaf.

WebSphere supports several other LDAP servers. For the latest information about the supported LDAP servers, see the following address:

<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>

Table 2-1 lists the LDAP V3 attributes that are similar to those used in Figure 2-2.

Table 2-1 LDAP V3 attributes

Attribute type or fields	String
CommonName	CN
LocalityName	L
StateOrProvinceName	ST
OrganizationName	O
OrganizationalUnitName	OU
CountryName	C
StreetAddress	STREET
DomainComponent	DC
UserID	UID
Relative DN	RDN
Specifies the top object class	objectClass : top
Specifies the organization object class	objectClass : organization
Specifies the organizational unit object class	objectClass: organizationalUnit

IBM Tivoli Directory Server supports the standards directory schema such as the following examples:

- ▶ IETF LDAP V3 RFCs-2252,2256
- ▶ The Directory Enabled Network (DEN)
- ▶ The Common Information Model (CIM) from the Distributed Management Task Force (DMTF)
- ▶ The Lightweight Intranet Person Schema (LIPS) from the Network Application Consortium

IBM Tivoli Directory Server also provides a set of extended common schema definitions, including the following objects, that other IBM products share when they exploit the LDAP directory server:

- ▶ Objects for white-page application such as ePerson, group, country, organization, organization unit and role, locality, state, and so forth
- ▶ Objects for other subsystem such as account services and access points, authorization, authentication, security policy, and so forth

The LDAP client usually requires read access to the user registry. Use replicas to increase security by separating the read function of the registry from the write function. You can do this if you create a registry replica that is used for read-only access, such as authentication, leaving the registry master only for making updates.

Figure 2-3 illustrates the architecture of LDAP security and positioning of the LDAP client.

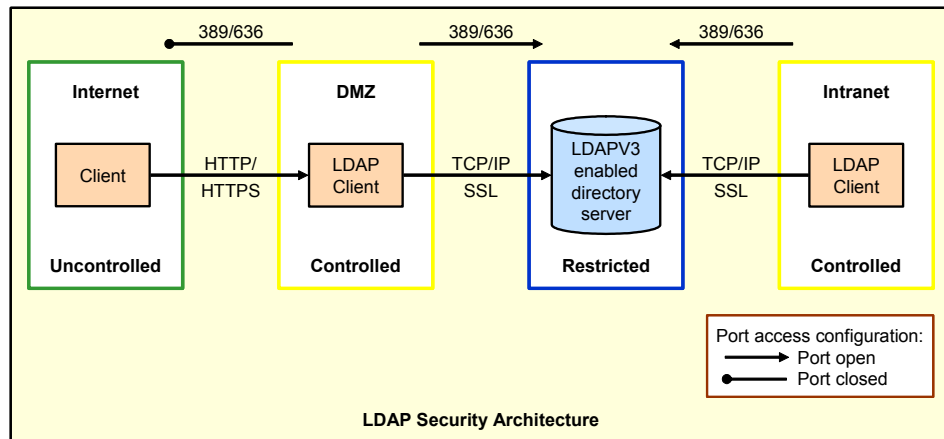


Figure 2-3 The LDAP security architecture

Security roles

IBM Tivoli Directory Server V5.2 supports the following security roles:

- ▶ Directory administrator

The directory administrator is associated with a specific user account. There is only one directory administrator account for the LDAP server. The directory administrator has complete rights to manage the LDAP server. This person also creates the user security role and defines the level of authorization that the user has over entries.

- ▶ Administrative group members

Administrative group members are users who have been assigned a subset of administrative privileges. All administrative group members have the same set of privileges. The administrative group is a way for the directory administrator to delegate a limited set of administrative tasks to one or more individual user accounts.

- ▶ Global administrative group members

The global administrative group is a way for the directory administrator to delegate administrative rights in a distributed environment to the database

back end. Global administrative group members are users who have been assigned the same set of privileges as the administrative group with regard to accessing entries in the database back end.

Global administrative group members do not have access to the audit log. The audit log can be used by local administrators to monitor the activity of global administrative group members. These members have activity or access rights related to any data or operations regarding the configuration settings of the directory server. This is commonly called the *configuration back end*. All global administrative group members have the same set of privileges.

► LDAP users

LDAP users are users whose privileges are determined by an access control list (ACL). Each LDAP user is identified with an LDAP entry that contains the authentication and authorization information for that user. The authentication and authorization information might also allow the user to query and update other entries depending on the type of authentication mechanism used. After the user ID and password are validated, the user can access any of the attributes of any entry to which that user has permission.

► Master server DN

The master server DN is a role that is used by replication that can update the entries under a replica's or a forwarding replica's replication context to which the DN is defined as a master server DN. The master server DN can create a replication context entry on a replica or forwarding replica if the DN is defined as the master server DN to that specific replication context or as a general master server DN.

2.2.1 Stand-alone LDAP registry for WebSphere Application Server V6.1

To use LDAP V3 as a user registry, you must have a valid user name (ID), user password, server host and port, base DN and, if necessary, bind DN and bind password.

Setting up security stand-alone LDAP registry

To set up security for the stand-alone LDAP registry:

1. In the administrative console, click **Security** → **Security administration, application, and infrastructure**.
2. In the User account Repository, click **Stand-alone LDAP registry** and then click **Configure**.

3. Under General properties, complete the following information as illustrated in Figure 2-4 on page 17:
 - a. In the Primary administrative user name field, enter a valid user name. You can either enter the complete DN of the user or the short name of the user as defined by the user filter in the Advanced LDAP settings panels. 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 local operating system registry to authenticate and obtain privilege information about users by calling the native application programming interface (API) in that particular registry.
 - b. Optional: If you want to use the server ID:
 - i. Know the differences between administrator name, internal server ID, and the serverID.
 - ii. Select **Automatically generated server identity** to enable the application server to generate the server identity that is used for internal process communication.
 - Alternatively, in the Server identity that is stored in the repository field, specify a user identity in the repository that is used for internal process communication.
 - Alternatively, in a Version 6.1.x node field, specify the user ID that is used to run the application server for security purposes for the server user ID or administrative user.
 - c. In the Advanced LDAP settings panel, click **Apply**.
 - d. From the Type list, select the type of LDAP server to use. The type of LDAP server determines the default filters that WebSphere Application Server uses. Select **Custom** and modify the user and group filters to use other LDAP servers, if required.
 - e. In the Host field, enter the fully qualified host name of the LDAP server. You can enter either the IP address or Domain Name Server (DNS) name.
 - f. In the Port field, enter the LDAP server port number. The host name and port number represent the realm for this LDAP server in the WebSphere Application Server cell. Therefore, if servers in different cells are communicating with each other by using LTPA tokens, these realms must match exactly in all the cells.

The default port number is 389. If multiple WebSphere Application Server profiles are installed and configured to run in the same single sign-on (SSO) domain, or if WebSphere Application Server interoperates with a previous version of WebSphere Application Server, the port number must match all configurations.

- g. In the Base Distinguished Name field, enter the base DN. The base DN indicates the starting point for searches in this LDAP directory server.

The Ignore Case option is always enabled. Ignore Case is required, and disabling it might cause authorization errors because of case sensitivity. This field is required for all LDAP directories except the Lotus® Domino® Directory. The Base Distinguished Name field is optional for the Domino server.
- h. Optional: In the Bind Distinguished Name field, enter the bind DN name. The bind DN is required if anonymous binds are not possible on the LDAP server to obtain user and group information. If the LDAP server is set up to use anonymous bind, leave this field blank. If a name is not specified, the application server binds anonymously.
- i. Optional: In the Bind password field, enter the password that corresponds to the bind DN.
- j. Optional: Modify the Search time-out value. This time-out value is the maximum amount of time that the client that sends a search request can wait for a response before timing out.
- k. Select the **Reuse connection** option. This option specifies that the server must reuse the LDAP connection. Clear this option only in rare situations where a router is used to send requests to multiple LDAP servers and when the router does not support affinity. Leave this option selected for all other situations.
- l. Optional: Verify that the **Ignore case for authorization** option is enabled. When you enable this option, J2EE authorization is case insensitive. Typically, an authorization check involves checking the complete DN of a user, which is unique on the LDAP server and is case sensitive. However, when you use either the IBM Directory or the Sun ONE Directory LDAP server, you must enable this option because the group information that is obtained from LDAP servers is not consistent in case. This inconsistency affects the authorization check only. Otherwise, this field is optional and can be enabled when a case-sensitive authorization check is required. You can also enable the Ignore case for authorization option when you are using SSO between the product and Lotus Domino. The default is enabled.
- m. Optional: Select the **Secure Sockets Layer (SSL) enabled** option if you want to use SSL communication with the LDAP server. If you select the SSL enabled option, you can select either the Centrally managed or the Use specific SSL alias option.

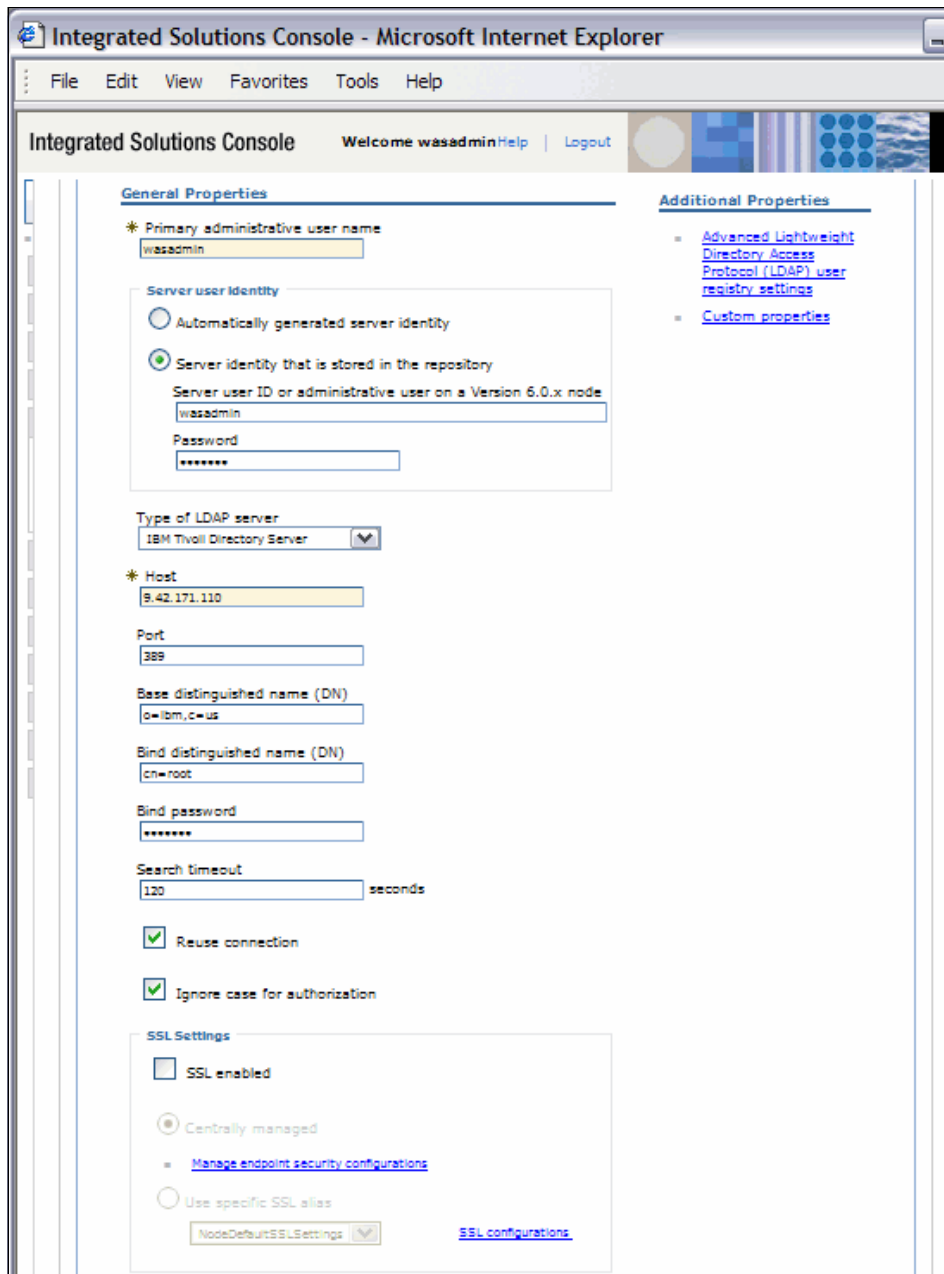


Figure 2-4 LDAP settings for WebSphere Application Server

Enabling the Centrally managed option to specify an SSL configuration for LDAP

Select the **Centrally managed** option (Figure 2-5) if you want to specify an SSL configuration for a particular scope such as the cell, node, server, or cluster in one location. To use the Centrally managed option, specify the SSL configuration for the particular set of endpoints. The Manage endpoint security configurations and trust zones panel shows all of the inbound and outbound endpoints that use the SSL protocol. For an LDAP registry, you can override the inherited SSL configuration by specifying an SSL configuration for LDAP.



Figure 2-5 Enabling SSL for the LDAP User Registry

To specify an SSL configuration for LDAP:

1. Click **Security** → **SSL certificate and key management** → **Manage endpoint security configurations and trust zones**.
2. Expand **Outbound** → *cell_name* → **Nodes** → *node_name* → **Servers** → *server_name* → **LDAP**.

Using a specific SSL alias

Select the **Use specific SSL alias** option if you intend to select one of the SSL configurations in the menu below the option. This configuration is used only when SSL is enabled for LDAP. The default is DefaultSSLSettings.

To modify or create a new SSL configuration:

1. Click **Security** → **SSL certificate and key management**.
2. Under Configuration settings, click **Manage endpoint security configurations**.
3. Select a Secure Sockets Layer configuration_name for selected scopes, such as a cell, node, server, or cluster.
4. Under Related items, click **SSL configurations**.
5. Click **New**.

6. Click **OK** and then click either **Apply** or **Save** until you return to the Secure administration, applications, and infrastructure panel.

Tip: Do run WebSphere Application Server as the root or administrator user. From a security point of view, for a production environment, configure WebSphere with an LDAP ID that is different from that of `cn=root`, with only read and search rights in the LDAP server.

Validation failure: If the validation fails for any reason, go back to the LDAP configuration panel and check your settings again.

Testing the LDAP user registry in WebSphere Application Server 6.1

To test the connection, follow the steps in 3.1, “Enabling administrative security” on page 50, to enable Administrative Security, and when the server starts, launch the administrative console. The administrative console must prompt you for your user ID and password for authentication because Administrative Security is enabled. Provide the user ID as `wsuser` and password as `test` if you load the directory with the data that accompanies this book. If you are able to log in successfully, your configuration is working properly.

Tips:

- ▶ Use the new `TestConnection` button in the LDAP panel to check your configuration.
- ▶ After you enable security to stop the server, provide the `-username` and `-password` parameters for the `stopserver` command script. For example, in a UNIX® environment, you can stop WebSphere Application Server with the following command:

```
/opt/IBM/WebSphere/AppServer/bin/stopServer.sh server1 -username  
wsuser -password test
```

2.2.2 Configuring the advanced LDAP user registry

To configure the advanced LDAP user registry settings when users and groups reside in an external LDAP directory:

1. Click **Security** → **Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Stand-alone LDAP registry** and click **Configure**.
3. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.

The default values for all the user and group related filters are already entered based on the type of LDAP server that is selected in the Standalone LDAP registry setting panel. When security is enabled and any of these properties change, go to the Secure administration, applications, and infrastructure panel and click **Apply** or **OK** to validate the changes.

Table 2-2 shows the default search settings for IBM Tivoli Directory Server.

Table 2-2 Advanced LDAP settings for Tivoli Directory Server

Property: Default value	Description
User Filter: (&(uid=%v)(objectclass=ePerson))	Specifies the LDAP user filter used to search the registry for users.
Group Filter: (&(cn=%v)(!(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))	Specifies the LDAP group filter used to search the registry for groups.
User ID map: *:uid	Specifies the LDAP filter that maps the short name of a user to an LDAP entry. This field takes multiple objectclass:property pairs delimited by a semicolon (;).
Group ID Map: *:cn	Specifies the LDAP filter that maps the short name of a group to an LDAP entry. This field takes multiple objectclass:property pairs delimited by a semicolon.
Group member ID map: ibm-allGroups:member;ibm-allGroups:uniqueMember	Specifies the LDAP filter that identifies user to group relationships.
Perform nested group search: un-checked	Select this option if the LDAP server does not support recursive server-side group member searches.
Certificate map mode: EXACT_DN	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.
Certificate filter: (& (uid=\${UniqueKey}))	The filter is used to map attributes in the client certificate to entries in the LDAP registry. The syntax or structure of this filter is: LDAP attribute=\${Client certificate attribute} (for example, uid=\${SubjectCN}).

Configuring the WebSphere Application Server key

Before you can configure WebSphere Application Server V6.1.1 to use SSL to communicate with the LDAP server, extract the LDAP server certificate from the LDAP key store and import it into the application server's key store that is used for LDAP connection.

Assuming a default installation of WebSphere Application Server V6.1.1, configure the WebSphere Application Server key as follows:

1. Open the IBM Tivoli Directory Server V6.0 key store with the iKeyman tool. For more information about how to use the tool, see the IBM Redpaper publication *WebSphere Security Fundamentals*, REDP-3944.
2. Find the location of the keystore that IBM Tivoli Directory Server V5.2 uses by looking up the `ibm-slapdSslKeyDatabase` parameter in the `ibmslapd.conf` configuration file. In our case, the location is `/etc/ldap_key.kdb`. See Chapter 4, “SSL administration and configuration management” on page 69, for detailed key configuration and management.
3. Export the LDAP signer certificate as `ldap_key.arm`.
4. Load the certificate into the keystore that is used by WebSphere Application Server V6.1 by opening the key store with the iKeyman tool. We used the default key store for LDAP security, which is in `{Websphere_root}/profiles/default/etc/DummyServerTrustFile.jks`.
Import the `ldap_key.arm` file into the key store.

Testing the LDAP SSL connection with WebSphere

See “Testing the LDAP user registry in WebSphere Application Server 6.1” on page 19 to test the connection. Follow the steps in 3.1, “Enabling administrative security” on page 50, to enable Administrative Security. The assumption here is that the LDAP connection has been tested and found to be working with Administrative Security enabled.

The purpose of this test is to make sure that WebSphere Application Server V6.1 is now communicating with LDAP by using SSL on the 636 port. The simplest way to verify this is to examine the network connections that opened after restarting it to execute the **netstat** command on the WebSphere Application Server V6.1.1 machine. This command must work in both Microsoft Windows® and UNIX systems.

If **netstat** reports that the `ldaps/636` port is being used, WebSphere Application Server V6.1.1 is using SSL to communicate with the LDAP server. When the server starts, launch the administrative console and provide the user name `wsuser` and password `test`. If you are able to log in successfully, your configuration was successful.

Dynamic groups and nested group support for Tivoli Directory Server

Dynamic and nested groups simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

WebSphere Application Server supports all LDAP dynamic and nested groups when using Tivoli Directory Server. This function is enabled by default, taking advantage of this new feature in Tivoli Directory Server.

Tivoli Directory Server uses the `ibm-allGroups` forward reference group attribute that automatically calculates all the group memberships including dynamic and recursive memberships for a user. Security directly locates a user group membership from a user object rather than indirectly search all the groups to match group members.

Configuring dynamic and nested group support for Tivoli Directory Server

To configure dynamic and nested group support for Tivoli Directory Server:

1. In the administrative console for WebSphere Application Server, click **Security** → **Secure administration, applications, and infrastructure**.
2. Under User account repository, click **Standalone LDAP registry**.
3. In the LDAP user registry configuration panel, select **IBM Tivoli Directory Server for the LDAP server**.
4. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.
5. On the next page:
 - a. Change the Group filter value as follows:

```
(& (cn=%v) (I
(objectclass=groupOfNames)
(objectclass=groupOfUniqueNames)
(objectclass=groupOfURLs)))
```
 - b. Change the Group member ID map value as follows:

```
ibm-allGroups:member;ibm-allGroups:uniqueMember
```
 - c. In the Auxiliary object class field in the Add an LDAP entry panel for your IBM Tivoli Directory Server, verify that the appropriate value is used:
 - When you create a nested group, the Auxiliary object class value is `ibm-nestedGroup`.
 - When you create a dynamic group, the Auxiliary object class value is `ibm-dynamicGroup`.

2.3 Local OS registry

With the local operating system (or local OS) user registry implementation, the WebSphere Application Server authentication mechanism can use the user accounts database of the local operating system.

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 or group information. Access to these APIs is restricted to users who have special privileges. These privileges depend on the operating system and are described in the following sections.

In WebSphere Application Server V6.1, you can use an internally-generated server ID because the Security WebSphere Common Configuration Model (WCCM) contains a new tag, `internalServerId`. You are not required to specify a server user ID and password during security configuration except in a mixed-cell environment.

See the “Administrative roles and naming service authorization” topic in the WebSphere Application Server - Express Information Center about the new internal server ID at the following address:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp>

Required privileges in Windows

The user running the WebSphere Application Server process requires proper operating system privileges to call the Windows systems API for authenticating and obtaining user and group information from the Windows operating system. (The user privilege is for starting the WebSphere Application Server and not `serverID` or `adminID`.) This user logs into the machine, or if the server is running as a service, is the *Log On As* user. Depending on the machine and whether it is a stand-alone machine or a machine that is part of a domain or is the domain controller, the access requirements vary:

- ▶ For a stand-alone machine, the user has the following characteristics:
 - Is a member of the *administrative* group
 - Has the *Act as part of the operating system* privilege
 - Has the *Log on as a service* privilege, if the server is run as a service
- ▶ For a machine that is a member of a domain, only a domain user can start the server process. This user has the following characteristics:
 - Is a member of the *domain administrative* groups in the domain controller
 - Has the *Act as part of the operating system* privilege in the Domain security policy on the domain controller

- Has the *Act as part of the operating system* privilege in the Local security policy on the local machine
- Has the *Log on as a service* privilege on the local machine, if the server is running as a service
- The user is a *domain user* and not a local user, which implies that when a machine is part of a domain, only a domain user can start the server
- ▶ For a domain controller machine, the user has the following characteristics:
 - Is a member of the *domain administrative* groups in the domain controller
 - Has the *Act as part of the operating system* privilege in the Domain security policy on the domain controller
 - Has the *Log on as a service* privilege on the domain controller, if the server is run as a service

More information: For more information about how to configure the required users for Windows, see the WebSphere Information Center (search for the *csec_locals* topic ID) or read the operating system's documentation.

Required privileges with UNIX

The user that is running the process ID that runs the WebSphere Application Server process requires *root* authority to call the local operating system APIs for authentication and for obtaining user or group information. With WebSphere Application Server in UNIX systems, you can only use the local machine registry, the Network Information Service (NIS) (Yellow Pages) is not supported.

2.3.1 Configuring WebSphere Application Server V6.1

To configure WebSphere to use the local operating system's registry:

1. Click **Security** → **Secure administration, application, and infrastructure**.
2. Under User account repository, select **Local operating system** and click **Configure**.
3. Enter a valid user name in the Primary administrative user name field. This value is the name of the user with administrative privileges that is defined in the registry and is used to access the administrative console or used by **wsadmin**. Click **Apply**.
4. Click **Specify user identity for interoperability**.
5. Select either the **Automatically generated server identity** or **Server identity that is stored in the repository** option. If you select the Server

identity that is stored in the repository option, as shown in Figure 2-6, enter the following information:

- a. For Server user ID or administrative user, specify the short name of the account that you chose in step 2 on page 19.
- b. For Server user password, enter the password of the account that you chose in step 2 on page 19.
- c. Click **OK**.

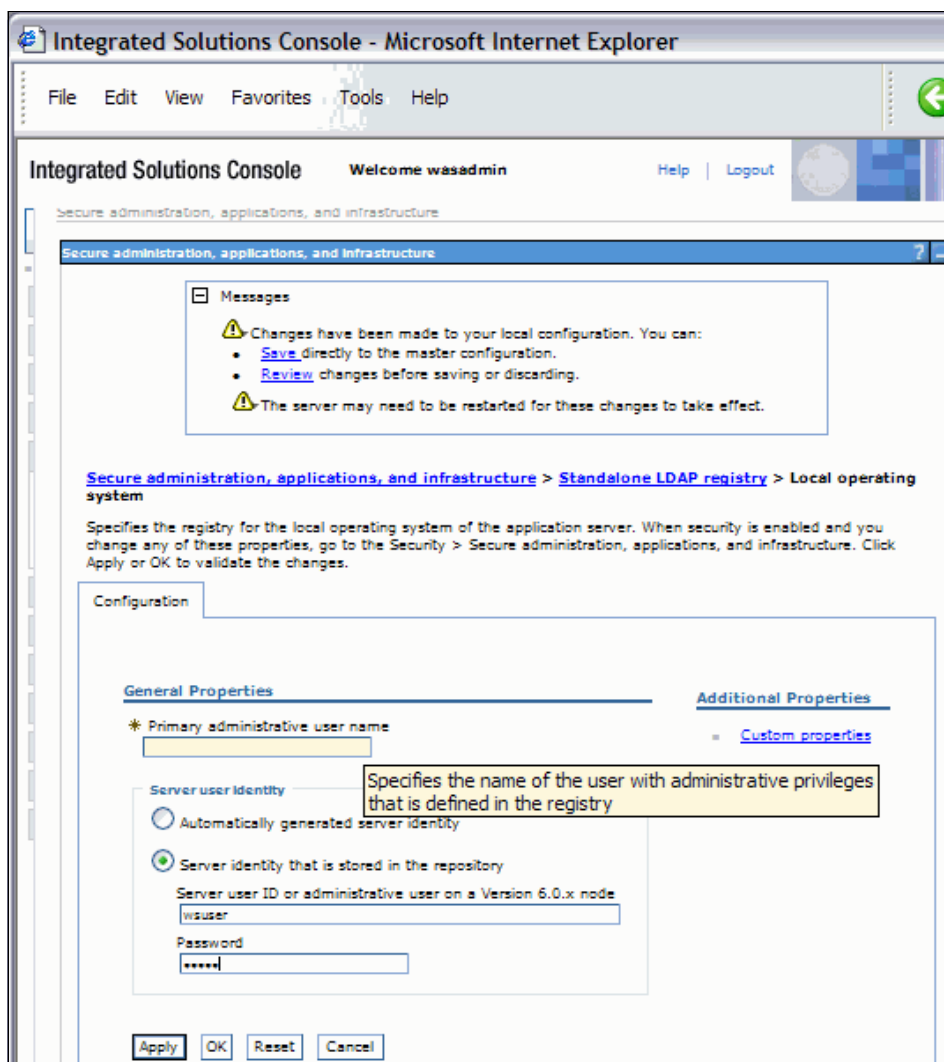


Figure 2-6 LocalOS registry user name and password

6. If there are no errors at this stage, select **Security** → **Secure administration, application, and infrastructure**.
7. Ensure that the Active User Registry option is set to **Local Operating System** and that **Security** is enabled. If this is not the case, make the necessary changes. Click **Apply** to validate the settings.
8. Save the configuration for WebSphere.
9. Restart your WebSphere Application Server V6.1.1.

Testing the Local OS user registry

To test the connection, click the **Test Connection** button. You receive the Success/Failure Message.

Tip: If WebSphere fails to start after enabling security, the failure might be caused by a problem with the user registry. If that is the case, you are unable to login to the administrative console. You require another solution to disable security. To disable administrative security manually for WebSphere, see 3.2, “Disabling administrative security” on page 58.

2.3.2 Stand-alone custom registry

WebSphere Application Server V6.1 security supports the use of a stand-alone custom registry, in addition to the local operating system registry, stand-alone LDAP registries, and federated repositories for authentication and authorization purpose. A Stand-alone custom-implemented registry uses the UserRegistry Java Interface as provided by WebSphere Application Server V6.1.

The UserRegistry interface

The UserRegistry interface is helpful in situations, for example, where the current user and group information exists in some other format (such as a database) and cannot be moved to a local OS or LDAP. In such a case, implement the UserRegistry interface so that WebSphere Application Server V6.1 can use the existing registry for all of the security-related operations. Using a custom registry is a software implementation effort. It is expected that the implementation does not depend on other WebSphere Application Server resources, such as data sources, for its operation.

WebSphere Application Server supports different types of user registries. Only one user registry can be active, and this active registry is shared by all of the product server processes.

To implement the UserRegistry interface, a Java class is required that provides WebSphere with a standard interface in order for WebSphere to communicate

with the registry in an appropriate fashion. The provision of this interface ensures that a variety of user registries can be used, such as relational databases and files stored directly on the file system. A combination of multiple registries can be used, such as LDAP and IBM RACF®.

The `UserRegistry` interface defines a general set of methods to allow the application server to obtain user and group information from the registry. The interface is also implemented by the two other available user registries in WebSphere Application Server V6.1.1, LDAP, and the local OS. The registry can operate as a process running remotely to the application server. Therefore, it is necessary for each registry to implement the `java.rmi.Remote` interface.

In regard to the initialization of a WebSphere Application Server V6.1.1 custom registry, with V4, it was possible to use other WebSphere Application Server components to initialize the custom registry. For example, a data source might have been used to connect to a database type custom registry or you might have used a deployed EJB. However, after V5, neither of these examples is possible because, unlike in V4, the security mechanism is initialized before other components such as containers. Therefore, these facilities are not available when the security component is started. Any implementation of the custom registry must not depend on any WebSphere Application Server component, such as data sources, enterprise beans, and so on.

The methods in the `UserRegistry` interface operates on the following information for users:

- | | |
|--------------------------|---|
| userSecurityName | The user name used to log in when prompted by an application. |
| uniqueUserID | Represents a unique identifier for the user. It is equivalent to the unique identifier (UID) in UNIX or the DN in LDAP. |
| userDisplayName | An optional string that describes a user. |
| groupSecurityName | Represents the security group. |
| groupUniqueld | Represents a unique identifier for the group. |
| groupDisplayName | An optional string that describes a group. |

Table 2-3 includes all the methods defined in the UserRegistry interface. Each method must be implemented by the custom registry.

Table 2-3 WebSphere UserRegistry interface

Method signature	Use
void initialize(java.util.Properties props) throws CustomRegistryException, RemoteException;	Initializes the registry. This method is called when creating the registry.
String checkPassword(String userSecurityName, String password) throws PasswordCheckFailedException, 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.
String mapCertificate(X509Certificate[] cert) throws CertificateMapNotSupportedException, CertificateMapFailedException, CustomRegistryException, RemoteException	Maps a certificate (of X.509 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.
String getRealm() throws CustomRegistryException, RemoteException;	The realm is a registry-specific string indicating the <i>realm</i> or <i>domain</i> for which this registry applies. For example, for IBM OS/400® or IBM AIX®, this is the host name of the system whose user registry this object represents. If null is returned by this, then method realm defaults to the value of "customRealm".
Result getUsers(String pattern, int limit) throws CustomRegistryException, RemoteException;	Gets a list of users that match a <i>pattern</i> in the registry. The maximum number of users returned is defined by the <i>limit</i> argument.
String getUserDisplayName(String userSecurityName) throws EntryNotFoundException, CustomRegistryException, RemoteException;	Returns the display name for the user specified by userSecurityName.
String getUniqueUserId(String userSecurityName) throws EntryNotFoundException, CustomRegistryException, RemoteException;	Returns the UniqueID for a userSecurityName. This method is called when creating a credential for a user.
String getUserSecurityName(String uniqueUserId) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the name for a user given its UniqueID.
boolean isValidUser(String userSecurityName) throws CustomRegistryException, RemoteException	Determines if the <i>userSecurityName</i> exists in the registry.
Result getGroups(String pattern, int limit) throws CustomRegistryException, RemoteException	Gets a list of groups that match a <i>pattern</i> in the registry. The maximum number of groups returned is defined by the <i>limit</i> argument.

Method signature	Use
String getGroupDisplayName(String groupSecurityName) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the display name for the group specified by groupSecurityName.
String getUniqueGroupId(String groupSecurityName) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the UniqueID for a group.
List getUniqueGroupIds(String uniqueUserId) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the UniqueIDs for all the groups that contain the UniqueID of a user. Called during creation of a user's credential.
String getGroupSecurityName(String uniqueGroupId) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the name for a group given its UniqueID.
boolean isValidGroup(String groupSecurityName) throws CustomRegistryException, RemoteException	Determines if the <i>groupSecurityName</i> exists in the registry.
Result getUsersForGroup(String groupSecurityName, int limit) throws NotImplementedException, EntryNotFoundException, CustomRegistryException, RemoteException	Gets a list of users in a group. The maximum number of users returned is defined by the <i>limit</i> argument.
public List getGroupsForUser(String userSecurityName) throws EntryNotFoundException, CustomRegistryException, RemoteException	Gets all the groups the given user is a member of.
Credential createCredential(String userSecurityName) throws NotImplementedException, EntryNotFoundException, CustomRegistryException, RemoteException	Throws the NotImplementedException for this method.

File-based user registry sample

A sample custom registry implementation is provided with the WebSphere Application Server. The user registry class is called `com.ibm.websphere.security.FileRegistrySample`.

The class is installed with WebSphere Application Server V6.1, and the source code is provided in the WebSphere Information Center for reference purposes. See the WebSphere Information Center for details regarding the format of these files and two sample files.

The files must be copied to the directories that are specified in the initialization properties (Table 2-4) for the custom registry before you can enable the registry.

Table 2-4 *FileRegistrySample* initialization properties

Name	Value
usersFile	File location and name, for example: \${USER_INSTALL_ROOT}/customer_sample/users.props
groupsFile	File location and name, for example: \${USER_INSTALL_ROOT}/customer_sample/groups.props

To configure the WebSphere Application Server to use the file user registry:

1. Launch the **Secure administration, application, infrastructure** console.
2. Under User account repository, select **Stand-alone custom registry** and click **Configure**.
3. On the Standalone customer registry page (Figure 2-7 on page 31):
 - a. In the Primary administrative user name field, enter a valid user name. 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 local operating system registry to authenticate and obtain privilege information about users by calling the native APIs in that particular registry.
 - b. Enter the complete location of the dot-separated class name that implements the `com.ibm.websphere.security`. In the Custom registry class name field, enter the UserRegistry interface. For the sample, this file name is `com.ibm.websphere.security.FileRegistrySample`.
 - c. Add your custom registry class name to the class path. Add the Java archive (JAR) file that contains your custom user registry implementation to the application server lib/ext directory.
 - d. Optional: Select the **Ignore case for authorization** option for the authorization to perform a case-insensitive check. Enabling this option is necessary only when your user registry is case insensitive and does not provide a consistent case when queried for users and groups.
 - e. Click **Apply** if you have any other additional properties to enter for the registry initialization.

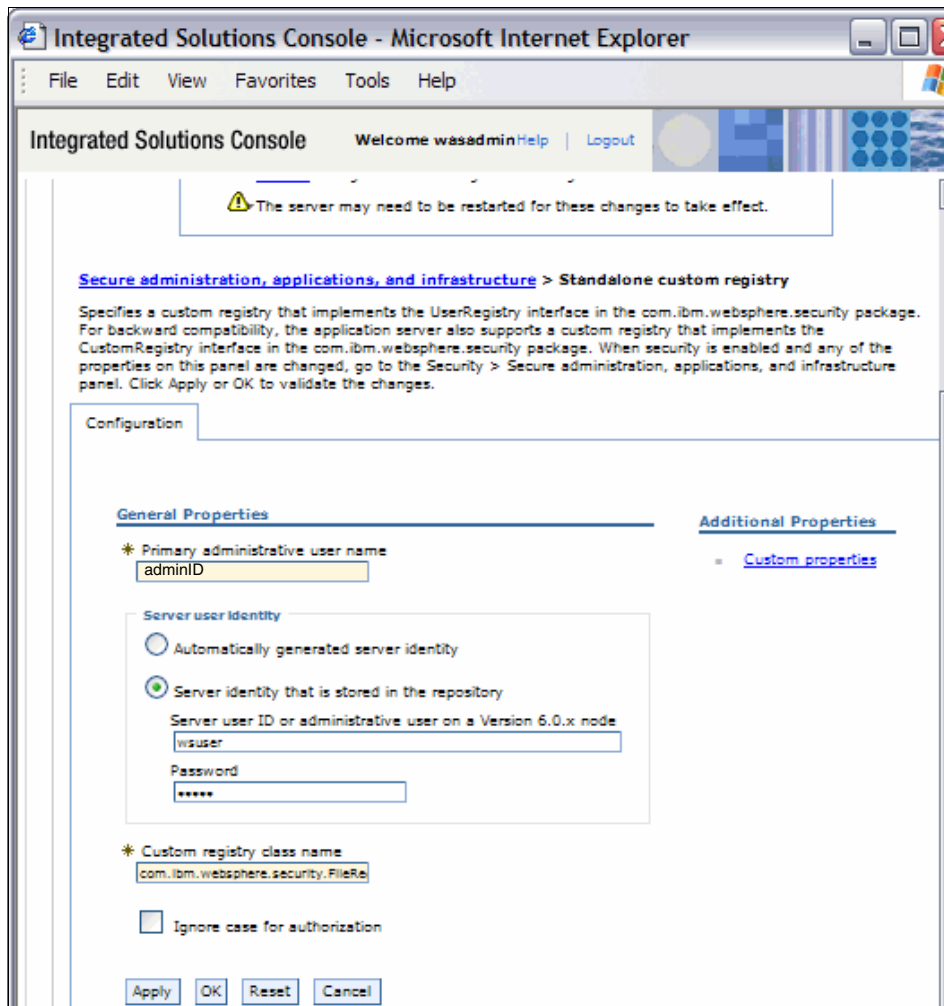


Figure 2-7 FileRegistry Sample Stand-alone Custom Registry user name and password

4. Click **Custom Properties**.

5. On the Customer properties page (Figure 2-8):
 - a. Add the properties necessary to initialize the registry. These properties are passed to the initialize method of the custom registry.
 - b. For the supplied FileRegistrySample code, enter the properties from Table 2-4 on page 30.

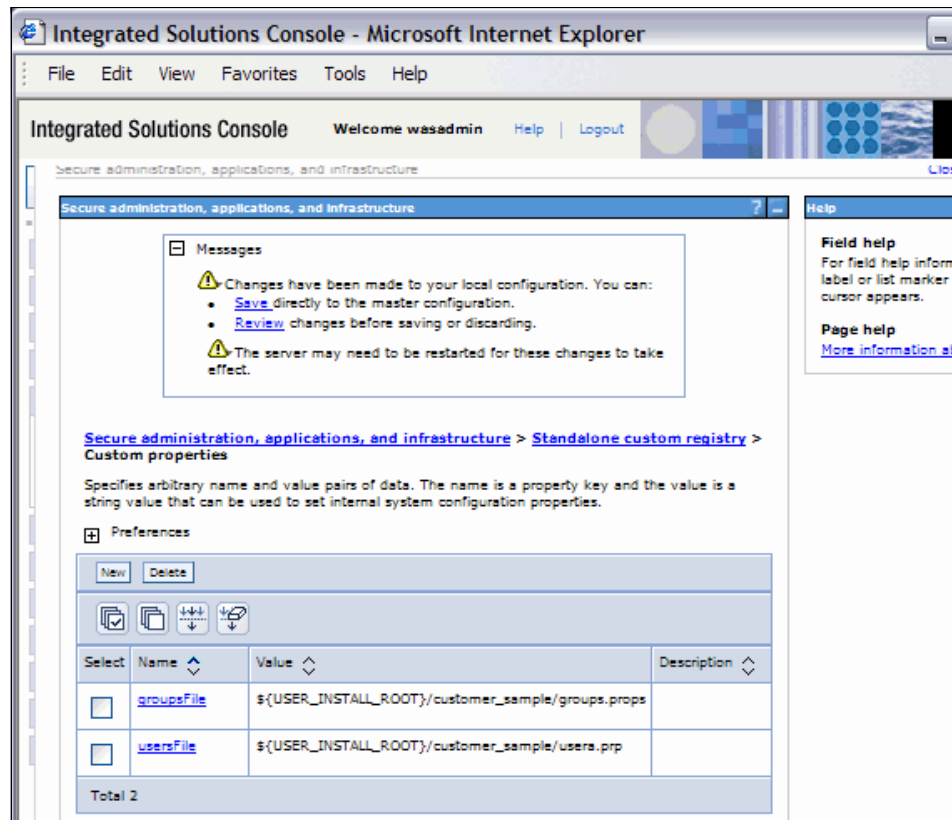


Figure 2-8 File registry sample custom properties

6. Click **Security** → **Secure administration, applications, and infrastructure**.
7. Under User account repository, select **Stand-alone custom registry** and click **Configure**.
8. Complete the following actions:
 - a. Under Additional properties, click **Specify user identity for interoperability**.
 - b. Select either the **Automatically generated server identity** or **Server identity that is stored in the repository** option. If you select the Server

Identity that is stored in the repository option, enter the following information:

- i. For Server user ID or administrative user on a Version 6.0.x node, specify the short name of the account (see Figure 2-7 on page 31).
 - ii. Server user password, specify the password of the account (see Figure 2-7 on page 31).
 - iii. Click **OK** and complete the required steps to turn on security.
- c. Click **Apply** to validate the settings.
9. Save the configuration for WebSphere.
 10. Restart the application server.

Testing the custom registry

To test the connection, click the **Test Connection** button. If the connection is correct, you see a message indicating success.

IBM DB2 custom user registry

The IBM DB2® registry uses Java Database Connectivity (JDBC™) to communicate with the database. Although this registry is tested with DB2, you must be able to modify it to work with other relational databases. The source code (DB2UserRegistrySample.java) is included in the files that are associated with this book along with the database structure that follows that of the LDAP registry.

Open the DB2UserRegistrySample.java source in IBM Rational® Application Developer V6.0 and check the comments in the source code. You will find that all required methods for the UserRegistry interface are implemented. Look for the SQL queries in the code and see what each method does with the database.

Although this can be modified, the sample instructions use the DB2 JDBC Universal Driver (Type 4) to successfully run and compile the application DB2 Type 4 driver libraries (db2jcc.jar, db2jcc_javax.jar, and db2jcc_license_cu.jar) and the WebSphere security libraries (sas.jar and wssec.jar) that you are required to add to the Java build path (Figure 2-9 on page 34).

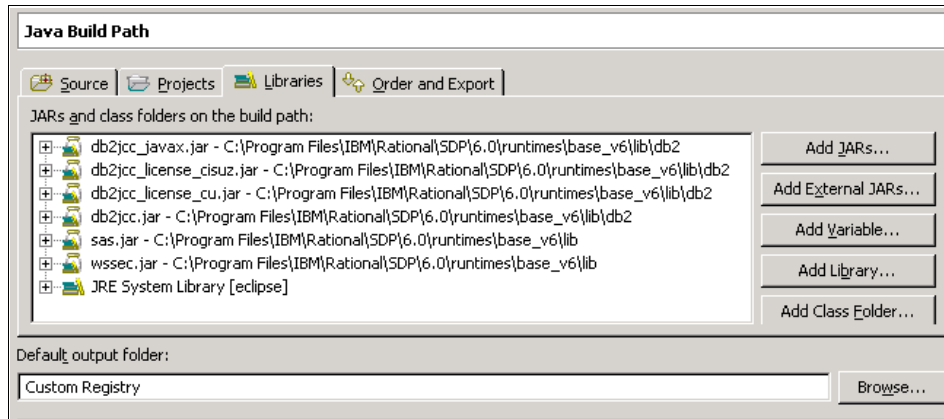


Figure 2-9 Rational Application Developer libraries required

The libraries (DB2 JAR files) shown previously and the compiled `DB2UserRegistrySample.class` file must be present in a directory that is accessible by the application server. This means the directory must be in the application server's class path (for example, `<WebSphere_root>/lib/ext`). Alternatively, update the application server's class path to refer to the directory that contains the class file and JAR files.

A simple custom registry test class is `DB2UserRegistrySampleTest`, which is shown in Example 2-1. This test class runs from the command line or is included from Rational Application Developer. It can be used to test whether the custom registry is working as required. The tool allows the developer to be sure that the custom registry is functioning before configuring the application server to use it.

Example 2-1 DB2UserRegistrySampleTest output

```

Initialized DB2UserRegistrySample
Enter a user name. wsuser
Enter a UID. 1
Enter a group name. admingrp
Enter a GID. 1
Enter a password. test
X509 certificate file.
Testing registry...
checkPassword: wsuser
getGroupDisplayName: group for administrators
getGroups: com.ibm.websphere.security.Result@1bb97283
getGroupSecurityName: admingrp
getRealm: customRealm
getUniqueGroupId: 1

```

```
getUniqueGroupIds: [1]
getGroupsForUser: [admingrp]
getUniqueUserId: 1
getUserDisplayName: WebSphere administrator
getUsers: com.ibm.websphere.security.Result@1eeab283
getUserSecurityName: wsuser
isValidGroup: true
isValidUser: true
mapCertificate: null
Test completed.
```

To run the DB2UserRegistrySampleTest tool, you must provide two arguments, the Custom Registry class, DB2UserRegistrySample, and the Custom Registry property file name. The property file contains the information shown in Table 2-5.

Table 2-5 DB2RegistrySample initialization properties

Name	Value
DBDRIVER	com.ibm.db2.jcc.DB2Driver
DBURL	jdbc:db2://9.42.171.75:50000/userreg
DBUSERNAME	webas
DBPASSWORD	test
DBSCHEMA	userreg

The tool prompts for user and group information and uses this information to query the custom registry. It also prompts for an X.509 certificate file, although the response can be empty (press Enter). In this case, the certificate check is not performed. The compiled classes are provided with this book as part of the additional material (see Appendix B, “Additional material” on page 543), and the DB2 libraries are available together with the DB2 product.

To configure the WebSphere Application Server to use the DB2 user registry:

1. Launch the **Secure administration, application, infrastructure** console.
2. Under User account repository, select **Stand-alone custom registry** and click **Configure**.
3. On the Standalone customer registry page (Figure 2-10 on page 36):
 - a. In the Primary administrative user name field, enter a valid user name. 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 local operating system registry to

authenticate and obtain privilege information about users by calling the native APIs in that particular registry.

- b. In the Custom registry class name field, enter the complete location of the dot-separated class name that implements the `com.ibm.websphere.security.UserRegistry` interface. We enter the file name `com.ibm.websphere.security.FileRegistrySample`.
- c. Add your custom registry class name to the class path. Add the JAR file that contains your custom user registry implementation to the application server `lib/ext` directory.

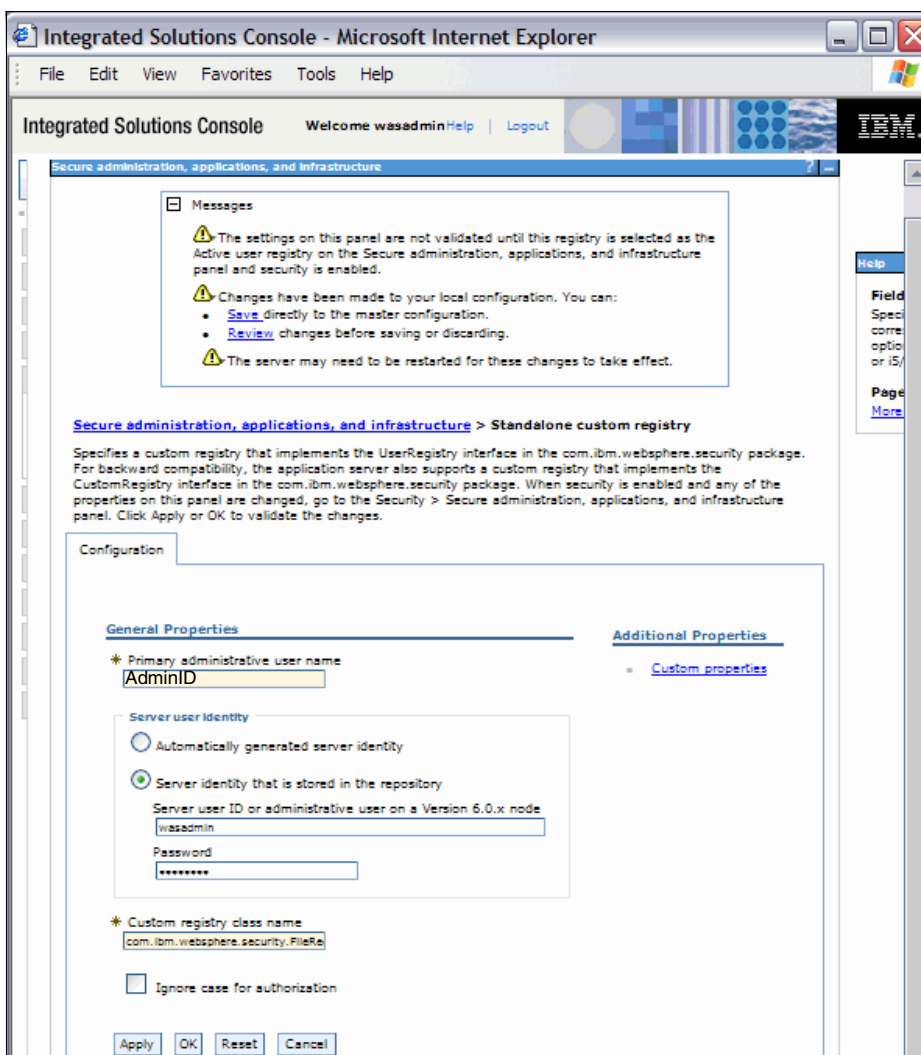


Figure 2-10 DB2Registry sample Standalone custom registry user name and password

- d. Optional: Select the **Ignore case for authorization** option for the authorization to perform a case insensitive check. Enabling this option is necessary only when your user registry is case insensitive and does not provide a consistent case when queried for users and groups.
 - e. Click **Apply** if you have other additional properties to enter for the registry initialization.
 4. Click **Custom Properties**.
 5. Add the properties necessary to initialize the registry. These properties are passed to the initialize method of the custom registry. For the supplied FileRegistrySample code, enter the properties as shown in Table 2-4 on page 30.
 6. Click **Security** → **Secure administration, applications, and infrastructure**.
 7. Under User account repository, select **Standalone custom registry** and click **Configure**.
 8. Complete the following actions:
 - a. Under Additional properties, click **Specify user identity for interoperability**.
 - b. Select either the **Automatically generated server identity** or **Server identity that is stored in the repository** option. If you select the Server Identity that is stored in the repository option, enter the following information:
 - i. For Server user ID or administrative user on a Version 6.0.x node, specify the short name of the account that you chose in step 1.
 - ii. Server user password, specify the password of the account that you chose in step 1.
 - iii. Click **OK** and complete the required steps to turn on security.
 9. Click **Custom Properties**.

10. Add the properties necessary to initialize the registry (Figure 2-11). These properties are passed to the initialize method of the custom registry.

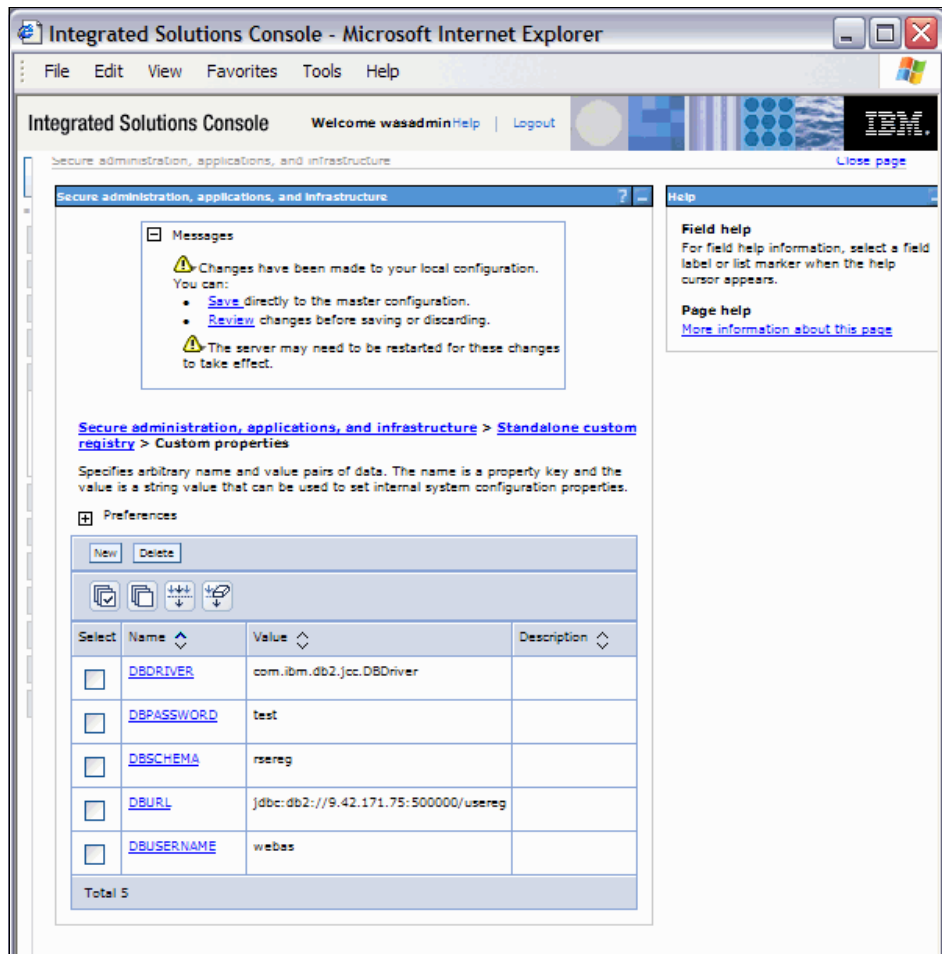


Figure 2-11 DB2 Registry Sample custom properties

11. If there are no errors at this stage, select **Security** → **Secure administration, application, and infrastructure**.
12. Ensure that the Active User Registry option is set to **Custom user registry** and that **Administrative Security** is enabled.
13. Click **Apply** to validate the settings.
14. Save the configuration for WebSphere.
15. Restart the application server.

Testing the custom registry

To test the connection, follow the steps in 3.1, “Enabling administrative security” on page 50, to enable Administrative Security. When the server starts, launch the administrative console. The administrative console prompts you for a user name and password for authentication. If you are able to log in successfully, your configuration is successful.

2.4 Federated repository

Presently, most WebSphere Application Server applications have their own models and components for mapping organizational entities, and they provide different levels of security. Most applications depend on a specific schema for the data in those repositories and are unable to use repositories with existing data. Virtual member manager helps these applications by providing them a common model, secure access to various brands and types of repositories, and the ability to use repositories with existing data. The single model includes a set of organizational entity types and their properties, a repository-independent API and a Service Provider Programming Interface (SPI) for plugging in repositories.

If you configure multiple repositories under the federated realm, you must also configure the supported entity type and specify a base entry for the default parent. The base entry for the default parent determines the repository location where entities of the specified type are placed on write operation by User and Group management.

A federated repository enables you to use multiple repositories with WebSphere Application Server V6.1. These repositories, which can be file-based repositories, LDAP repositories, or a sub-tree of an LDAP repository, are defined and theoretically combined under single realm. All of the user repositories that are configured under the federated repository functionality are transparent to WebSphere Application Server.

Tips for multiple user repositories: The user ID and the DN for an LDAP repository must be unique in multiple user repositories that are configured under the same federated repository configuration. In addition, the federated repositories functionality in WebSphere Application Server supports the logical joining of entries across multiple user repositories when the application server searches and retrieves entries from the repositories.

2.4.1 Connecting WebSphere Application Server to a federated repository

To connect WebSphere Application Server V6.1 to a federated repository:

1. Click **Security** → **Secure administration, applications, and infrastructure Security configuration wizard**.
2. Select your protection setting and click **Next**.
3. Select the **Federated repositories** option and click **Next**.

Then to modify the federated repository configuration:

1. Click **Security** → **Security administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** and click **Configure**.
3. In the Federated repositories pane (Figure 2-12 on page 41), enter the following values for the initial configuration of a built-in, file-based repository:

Tips:

- ▶ The user name and password do not have to be in the federated repository because they are created.
- ▶ If you previously configured federated repositories, do not use the Security configuration wizard to modify your configuration. Instead, modify your configuration by using the federated repositories selection under User account repository on the Secure administration, applications, and infrastructure panel.

- *Primary administrative user name* specifies the name of the user with administrative privileges that is defined in the repository, for example, adminUser.
- *Password* specifies the password of the administrative user who manages file product resources and user accounts.
- *Confirm password* confirms the password of the administrative user who manages the product resources and user accounts.

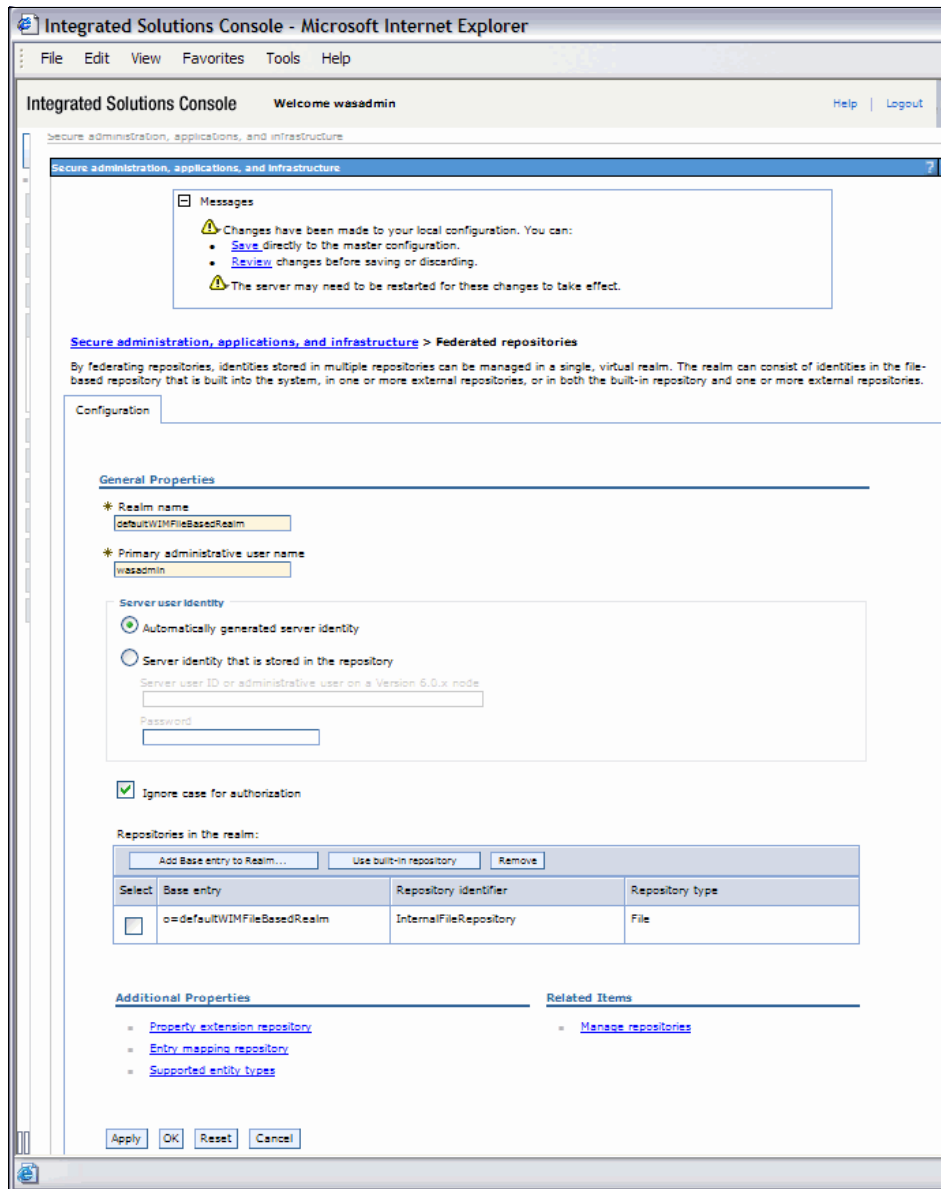


Figure 2-12 Configuring a federated repository

2.4.2 Configuring supported entity types in a federated repository

You must configure the supported entity types before you start managing the account with users and groups in the administrative console. You cannot add or delete the supported entity types, because these types are predefined.

To manage users and groups:

1. In the console navigation tree, click **Users and Groups**.
2. Click either **Manage Users** or **Manage Groups**.
3. In the administrative console, click **Security** → **Security administration, and infrastructure**.
4. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
5. Complete the following actions:
 - a. Click **Supported entity types** to view a list of predefined entity types.
 - b. Click the name of a predefined entity type to change its configuration.
 - c. In the Base entry for the default parent field, type the DN of the base entry in the repository. This entry determines the default location in the repository where entities of this type are placed on write operations by User and Group management.
 - d. In the Relative Distinguished Name properties field, enter the RDN properties for the specified entity type. Possible values are `cn` for group, `UID` or `cn` for `PersonAccount`, and `o`, `ou`, `dc`, and `cn` for `OrgContainer`. Delimit multiple properties for the `OrgContainer` entity with a semicolon.
 - e. Click **OK**.

Testing the configuration

To test the configuration:

1. In the Secure administration, application, and infrastructure pane:
 - a. Click **Apply** to verify the federated repositories configuration. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
 - b. Enable security for the realm. See 3.1, “Enabling administrative security” on page 50.
 - c. Click **Apply**.

2. Save, stop, and restart all the product servers (deployment managers, nodes, and application servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

2.4.3 Configuring an entry mapping repository in a federated repository

The federated repository configuration uses multiple repositories simultaneously and recognizes the entries in the different repositories as entries that represent distinct entities. By configuring an entry mapping repository, a federated repository configuration can use both LDAP and the database at the same time. The federated repository configuration hierarchy and constraints for identifiers provide the aggregated namespace for both of those repositories and prevent identifiers from colliding.

When you configure an entry mapping repository, you can supply a valid data source, a direct connection configuration, or both. The system first tries to connect by way of the data source. If the data source is not available, then the system uses the direct access configuration.

Restriction: You cannot configure an entry mapping repository in a mixed-version deployment manager cell.

To configure the entry mapping repository:

1. Configure the WebSphere Application Server data source.
2. Set up the entry mapping repository by using `wsadmin`.
3. Configure the entry mapping repository into the federated repository:
 - a. In the administrative console, click **Security** → **Secure administration, applications, and infrastructure**.
 - b. Under User account repository, select **Federated repositories from the available realm definitions** field and click **Configure**.
 - c. Click **Entry mapping repository**.
 - d. In the Data source name field, enter the name of the data source.
 - e. Select the type of database that issued for the property extension repository.
 - f. In the JDBC driver field, enter the name of the Java Database Connectivity (JDBC) driver.

- g. In the Database URL field, enter the database URL that is used to access the property extension repository with JDBC (Example 2-2). Use of an alphanumeric text string conforms to the standard JDBC URL syntax.

Example 2-2 DB2 database URL

```
COM.ibm.db2.jdbc.app.DB2Driver  
jdbc.db2.wim
```

- h. In the Database administrator user name field, enter the user name of the database administrator.
- i. In the Password field, enter the password of the database administrator.

Testing the configuration

To test the configuration:

1. Click **Security** → **Secure administration, application, and infrastructure** to return to the Secure administration, applications, and infrastructure panel.
2. Complete the following actions:
 - a. In the Current realm definition field, verify that Federated repositories is defined. If federated repositories is not identified, select **Federated repositories** from the available realm definitions field and click **Select as current**.
 - b. Click **Apply** to verify the federated repositories configuration. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. Enable security for the realm. See 3.1, “Enabling administrative security” on page 50.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and application servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

2.4.4 Configuring a property extension repository in a federated repository

A federated repository configuration provides a *property extension repository*, which is a database regardless of the type of main profile repositories for a property-level join configuration. When an application uses the federated repository configuration to retrieve an entry for a person, the federated repository configuration transparently joins the properties of the person that is retrieved from either the LDAP or the customer’s database with the properties of the

person that is retrieved from the property extension repository into a single logical person entry.

When you configure a property extension repository, you can supply a valid data source, a direct connection configuration, or both. The system first tries to connect by way of the data source. If the data source is not available, the system uses the direct access configuration.

Restriction: You cannot configure a property extension repository in a mixed-version deployment manager cell.

Property extension repository configuration

To configure the property extension repository:

1. Configure the WebSphere Application Server data source. See “Configuring the WebSphere Application Server data source” in the WebSphere Application Server V6.1 Information Center on the Web at the following address:
<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp>
2. Set up the property extension repository by using **wsadmin**. See 3.1, “Enabling administrative security” on page 50, to set up an entry mapping repository, a property extension repository, or a database repository by using **wsadmin** commands.
3. Click **Security** → **Secure administration, applications, and infrastructure**.
4. Under User account repository, select **Federated repository**, and click **Configure**.
5. Click **Property extension repository**.
6. In the Data source name field, enter the name of the data source.
7. Select the type of database that is used for the property extension repository.
8. In the JDBC driver field, enter the name of the JDBC driver.
9. In the Database URL field, enter the database URL that is used to access the property extension repository with JDBC. Use an alphanumeric text string that conforms to the standard JDBC URL syntax.
10. In the Database administrator user name field, enter the user name of the database administrator.
11. In the password field, enter the password of the database administrator.
12. In the Entity retrieval limit field, enter the entity retrieval limit. The entity retrieval limit is the maximum number of entities that the system can retrieve

from the property extension repository with a single database query. The default value is 200.

Testing the configuration

To test the configuration:

1. Enable security for the realm. See 3.1, “Enabling administrative security” on page 50.
2. Save, stop, and restart all the product servers (deployment managers, nodes, and application servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring LDAP in a federated repository

To configure secure access to an LDAP repository with failover servers option:

1. In the administrative console, click **Security** → **Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.

Configuration tab

Set the following items on the **Configuration** tab:

- ▶ Repository Identifier
Specifies a unique identifier for the LDAP repository. This identifier uniquely identifies the repository within the cell, for example, LDAP1.
- ▶ Directory Type
Specifies the type of LDAP server to which you connect. Expand the drop-down list to display a list of LDAP directory types.
- ▶ Primary host name
Specifies the host name of the primary LDAP server. This host name is either an IP address or a DNS name.
- ▶ Failover host name
Specifies the host name of the failover LDAP server. You can specify a secondary directory server to be used in the event that your primary directory server becomes unavailable. After switching to a secondary directory server, the LDAP repository attempts to reconnect to the primary directory server every 15 minutes.

- ▶ Port
Specifies the port of the failover LDAP server. The default value is 389, which is not an SSL connection. Use port 636 for an SSL connection. For some LDAP servers, you can specify a different port for non-SSL or SSL connections.
- ▶ Support referrals to other LDAP servers
Specifies how referrals that are encountered by the LDAP server are handled. A *referral* is an entity that is used to redirect a client request to another LDAP server. A referral contains the names and locations of other objects. It is sent by a server to indicate that the information that the client requested can be found at another location, possibly at another server. The default is ignored.
- ▶ Bind Distinguished Name
Specifies the distinguished name for the application server to use when binding to the LDAP repository.
- ▶ Bind Password
Specifies the password for the application server to use when binding to the LDAP repository.
- ▶ Login properties
Specifies the property names to use to log in to the application server.
- ▶ Certificate mapping
Specifies whether to map X.509 certificates to an LDAP directory by using EXACT_DN or CERTIFICATE_FILTER. Specify CERTIFICATE_FILTER to use the specified certificate filter for the mapping.
- ▶ Certificate Filter
Specifies the filter certificate mapping property for the LDAP filter. The filter is used to map attributes in the client certificate to entries in the LDAP repository.
- ▶ Require SSL communications
Specifies whether secure socket communication is enabled to the LDAP server.
- ▶ Centrally managed
Specifies that the selection of an SSL configuration is based on the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.
- ▶ Use specific SSL alias
Specifies the SSL configuration alias to use for LDAP outbound SSL communications. This option overrides the centrally managed configuration for the JNDI platform.

Limitation on federated repositories in a mixed-version environment

In a mixed-version deployment manager cell that contains both Version 6.1.x and Version 5.x or 6.0.x nodes, the following limitations apply for configuring federated repositories:

- ▶ You can configure only one LDAP repository under federated repositories, and the repository must be supported by Version 5.x or 6.0.x.
- ▶ You can specify a realm name that is compatible with prior versions only. The host name and the port number represent the realm for the LDAP server in a mixed-version nodes cell.
- ▶ You must configure a stand-alone LDAP registry. The LDAP information in both the stand-alone LDAP registry and the LDAP repository under the federated repositories configuration must match. During node synchronization, the LDAP information from the stand-alone LDAP registry propagates to the Version 5.x or 6.0.x nodes.

Important: Before node synchronization, verify that Federated repositories is identified in the current realm definition field. If Federated repositories is not identified, select **Federated repositories from the available realm definitions** field and click **Select as current**. Do not set the stand-alone LDAP registry as the current realm definition.

Federal Information Processing Standard support

Government agencies and financial institutions use Federal Information Processing Standards (FIPS) to ensure that the products conform to specified security requirements. For more information about these standards, see the National Institute of Standards and Technology Web site at:

<http://www.nist.gov/>

WebSphere Application Server integrates cryptographic modules including Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE), which have undergone FIPS 140-2 certification. In the WebSphere Application Server documentations, the IBM JSSE and JCE modules that have undergone FIPS certification are referred to as IBMJSSEFIPS and IBMJCEFIPS. When you enable FIPS, several components of the Application Server are affected including the cipher suites, the cryptographic providers, the load balancer, the caching proxy, the high availability manager, and the data replication service.

Note: IBM products with WebSphere Application Server V6.1 maintain a FIPS level of security compliance.



Administrative security

The term *administrative security* represents the security configuration which affects the entire *security domain*. The security domain consists of all the servers that are configured with the same user registry *realm* name. The basic requirement for a security domain is that the access ID returned by the registry from one server be the same access ID as that returned from the registry on any other servers within the same security domain.

Enabling administrative security activates a wide variety of security settings for WebSphere Application Server. While values for these settings can be specified, they take effect only when administrative security is activated. These settings include authentication of users, the use of Secure Sockets Layer (SSL), the choice of user account repository, and application security.

In previous releases of WebSphere Application Server, enabling global security activated security for both administration and applications. In WebSphere Application Server V6.1, global security has been split into administrative and application security, each of which can be enabled separately. However, as mentioned previously, in order for application security to take effect, administrative security must be enabled.

3.1 Enabling administrative security

In WebSphere Application Server V6.1, administrative security is enabled by default as part of the installation process. This out-of-box enabled security is made possible due to the inclusion of the built-in, file-based repository. The built-in repository is a new feature made possible through the integration of *Virtual Member Manager* (VMM) into WebSphere Application Server.

If WebSphere administrative security is disabled, you can re-enable it. From the administrative console, click **Security** → **Secure administration, applications, and infrastructure**. In the Secure administration, applications, and infrastructure window (Figure 3-1), select **Enable administrative security**.

The screenshot shows the 'Secure administration, applications, and infrastructure' configuration page. The page title is 'Secure administration, applications, and infrastructure'. Below the title is a description: 'The application serving environment is completely secured when administration is restricted. The applications and the infrastructure that supports the administration and applications also are secured.' The main content area is titled 'Configuration' and contains several sections: 'Administrative security' with a checked checkbox for 'Enable administrative security' and links for 'Administrative User Roles' and 'Administrative Group Roles'; 'Application security' with a checked checkbox for 'Enable application security'; 'Java 2 security' with a checked checkbox for 'Use Java 2 security to restrict application access to local resources' and sub-options for 'Warn if applications are granted custom permissions' (unchecked) and 'Restrict access to resource authentication data' (checked); 'User account repository' with a text field for 'Current realm definition' containing 'Federated repositories' and a dropdown for 'Available realm definitions' also containing 'Federated repositories', along with 'Configure' and 'Set as current' buttons. At the bottom are 'Apply' and 'Reset' buttons. There are also buttons for 'Security Configuration Wizard' and 'Security Configuration Report' at the top of the configuration area.

Figure 3-1 Administrative, application, and infrastructure security configuration page

3.1.1 Main components of WebSphere security

WebSphere security has three critical components:

- ▶ Authentication protocol

The authentication protocol is used for Remote Method Invocation (RMI) over the Internet InterORB Protocol (IIOP) requests when security is enabled. WebSphere Application Server is configured to use Common Secure Interoperability Version 2 (CSIV2) by default. IBM Secure Authentication Service is the authentication protocol that is used by all releases of WebSphere Application Server prior to Version 5.

Support for the Secure Authentication Service protocol is provided for backwards compatibility with previous product releases. However, the configuration pages for Secure Authentication Service are shown only in the administration console when WebSphere Application Server V6.0 and previous version servers are federated into the V6.1 cell. Secure Authentication Service has been deprecated and will be removed from future WebSphere releases. The CSIV2 is defined by the Object Management Group (OMG) as a standard authentication protocol for vendors to interoperate securely.

- ▶ Authentication mechanism

The WebSphere Application Server uses Lightweight Third Party Authentication (LTPA) as the default authentication mechanism. Previous releases supported the Simple WebSphere Authentication Mechanism (SWAM). However, SWAM was deprecated in WebSphere Application Server V6.1 and will be removed in future releases.

LTPA supports forwardable credentials and, for security reasons, a configurable expiration time is set on the credentials. The use of LTPA allows you to enable single sign-on (SSO) for your security domain. Additional configuration settings are available and are explained in the “Authentication mechanisms and expiration” topic in the WebSphere Application Server V6.1 Information Center at the following address:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp>

Important: If your infrastructure includes only WebSphere Application Server Version 5.1.1 and later, you must disable the *Interoperability Mode* on the Single Sign-On configuration page. WebSphere Version 5.1.1 and later use a new LTPA token format with stronger encryption. The interoperability mode provides backwards compatibility support for the older format tokens.

► User account repository

Four types of realm definitions can be used for the user account repository:

- Local operating system
- Stand-alone Lightweight Directory Access Protocol (LDAP) registry
- Stand-alone custom registry
- The Federated repositories

Details regarding the user account repository are in Chapter 2, “Configuring the user registry” on page 7. Make sure that the primary administrative user name and server user identity fields are correctly completed.

Whenever Local operating system is chosen for the user account repository, only special users can enable administrative security and later to start the secure WebSphere. See 2.3, “Local OS registry” on page 23:

- For UNIX-based platforms, the WebSphere Application Server process must be *owned* by a user with a root authority.
- For Windows-based platforms, WebSphere must be *started* by a user who has the “Act as part of the operating system” rights. Make sure that the system is rebooted if you must change the rights. Otherwise WebSphere might not make the changes.

Table 3-1 summarizes the differences in security authentication capabilities and user registries between WebSphere Application Server V6.0 and V6.1.

Table 3-1 Security capability comparison of WebSphere Application Server V6.0 and V6.1

	Authentication protocols	Authentication mechanisms	Local OS registry	LDAP registry	Custom registry	Federated repositories
V6.0	<ul style="list-style-type: none"> ► CSIV2 ► Secure Authentication Service ► CSIV2 and Secure Authentication Service 	LTPA SWAM	Yes	Yes	Yes	No
V6.1	<ul style="list-style-type: none"> ► CSIV2 ► CSIV2 and Secure Authentication Service^a 	LTPA	Yes	Yes	Yes	Yes

a. The IBM Secure Authentication Service authentication protocol has been deprecated. Support for it will be removed in future releases of WebSphere Application Server. Support for Secure Authentication Service in V6.1 is available for backwards compatibility with V5.x and V6.0 servers, which can be federated into a V6.1 cell.

3.1.2 Security Configuration Wizard

WebSphere Application Server V6.1 offers the Security Configuration wizard to help you enable security for your application serving environment. This wizard takes you through the basic components that you must configure to activate security for your realm. To enable security with the wizard:

1. Log in to the WebSphere Administration Console.
2. Select **Security** → **Secure administration, applications, and infrastructure**, and then click **Security Configuration Wizard**.
3. Select the extent of protection, as shown in Figure 3-2. At a minimum, the wizard enables administrative security based on the input you provide. Additional security features, such as application and Java 2 security, are optional settings that you can enable.

More information: For more information about enabling application security, see 6.1.1, “Enabling application security” on page 102.

Click **Next**.

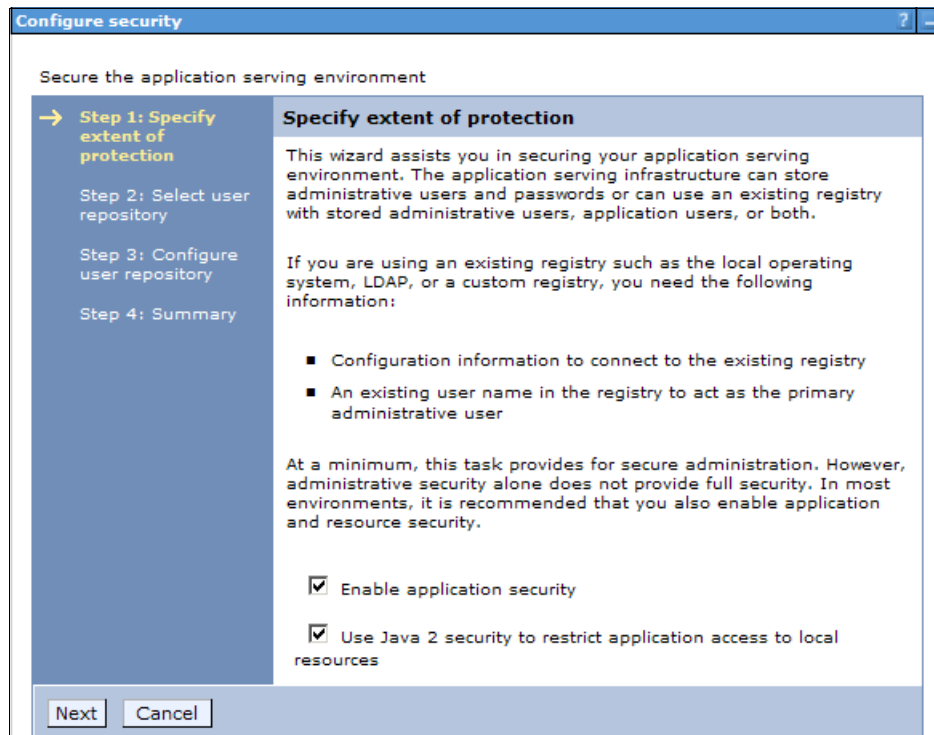


Figure 3-2 Configuring security with the Security Configuration Wizard

4. Select your User account repository:
 - Selecting Federated repositories walks you through configuration of the built-in, file-based user repository only. To configure a Federated repository with a non-file-based repository in the realm, you must use the User accounts repository section on the Secure administration, applications, and infrastructure panel.
 - Selecting any of the other repository options presents you with a page to provide the basic information required to configure the chosen registry.

Note: WebSphere Application Server V6.1 separates the server user identity from the primary administrative user. The *primary administrative user* is any valid user in the user account repository that you choose to give default administrative privileges. The *server user identity* is used for server to server communication. By selecting the automatically generated server identity, WebSphere creates an identity for internal communications that is not stored in the repository and does not have a password. LDAP directories still require a bind DN for successful communications.

Click **Next**.

5. Configure the repository and click **Next**.
6. On the Summary page, review the information. If everything is correct, click **Finish**.
7. When you return to the Secure administration, applications, and infrastructure panel, click **Apply** and save the WebSphere configuration.

Important: If the User account repository section does not list your new registry as the *Current realm definition*, select it from the list of Available realm definitions, click **Set as current**, and then click **Apply**.

8. Restart the application server in order for the changes to take effect.

After the server is restarted, to gain access to the Administrative Console, use the primary administrative user name and password defined in the user account repository. See 3.3, “Administrative roles” on page 59, about adding access to the Administrative console for other users or groups.

Verifying and testing administrative security

After your server is restarted in secure mode, you can test that security is properly enabled. There are basic tests that you can perform:

- ▶ Verify the form login. When using the Web-based Administrative Console, the Web-based form login page that is displayed forces you to enter a user ID and password. Only a user ID with administrative roles must be able to log in.
- ▶ Verify that the Java Client Basic Authentication works by executing the `<WebSphere_home>\bin\dumpNameSpace.bat` file.

A challenge login window must open. Although you might be able to click **Cancel**, you must type *any* correct user ID and password that is defined in the user account repository to test the security.

Be aware that the login panel for the Java client only opens if the `com.ibm.CORBA.loginSource` property is set to `prompt` in the `sas.client.props` file. Clicking **Cancel** works only if CosNaming security (see 3.4, “Naming service security: CosNaming roles” on page 64) allows *read access* to everyone. These values are the default values when you installed WebSphere.

Successfully running these basic tests indicates that the administrative security is working correctly.

More information: For information about how to test application security, see 6.1.1, “Enabling application security” on page 102.

3.1.3 Other security properties

Several other properties can be set from the Secure administration, applications, and infrastructure page (Figure 3-1 on page 50). Some of them are used only if administrative security is enabled, such as User account repository, Application security, Authentication, and authorization providers. Others, for example, Java 2 Security, are not related to the enabling of administrative security. This means that those properties can be activated *and used* even if administrative security is not enabled.

- ▶ Java 2 security

This property specifies whether to enable or disable Java 2 security permission checking. We recommend that you enable this security feature because it protects the WebSphere infrastructure from applications and applications from each other.

This protection is achieved by restricting access to the WebSphere internal APIs, administrative APIs, configuration files, and enforcing Java 2 Platform, Enterprise Edition (J2EE) recommended restrictions:

- Enterprise JavaBeans (EJB) are not allowed access to the file system.
- Servlets are only allowed file system access within the Web archive (WAR) file.
- Use of `getUserPrincipal()` is not allowed except by explicitly granted access.

When the Java 2 security option is enabled and if an application requires more Java 2 security permissions than are granted in the default policy, the application might fail to run properly until the required permissions are granted in either the node default `app.policy` file or the `was.policy` file of the application. For applications that were not developed with Java 2 security in mind, the simplest way is to grant full permission to all resources within the application to place the following entry in the `was.policy` file:

```
grant codeBase "file:${application}" {  
    permission java.security.AllPermission;  
}
```

Although this is a guaranteed way to get applications functioning in an environment with Java 2 security enabled, we do *not* recommend this practice. You must configure applications such that they are granted access to only those resources that they require and with only the appropriate permissions for the type of access required. Appropriate permissions can be determined by checking for failed permission error messages in the logs, or by using the Eclipse plug-in, Security Workbench Development Environment for Java (SWORD4J), which is available at the following address:

<http://alphaworks.ibm.com/tech/sword4j>

Determining the required permissions for an application can be a difficult task, but the end result is a much more secure application serving environment.

- ▶ Warn if applications are granted custom permissions

The `filter.policy` file contains a list of permissions that an application should *not* have according to the J2EE 1.4 specification. If an application is installed with a permission specified in this policy file and this option is enabled, a warning message is issued. Java 2 security must be enabled in order to enable this setting.

- ▶ Restrict access to resource authentication data

Enable this option to restrict application access to sensitive J2EE Connector Architecture (JCA) mapping authentication data. Consider doing this when both of the following conditions apply:

 - Java 2 security is enabled.
 - The application code is granted the `accessRuntimeClasses` `WebSphereRuntimePermission` in the `was.policy` file found within the application enterprise archive (EAR) file.
- ▶ Use domain-qualified user names

If this option is enabled, user names are displayed with their fully-qualified domain attribute when retrieved programmatically.

Note: Some of the properties in the Secure administration, applications, and infrastructure page, for example, *Java 2 Security*, can be enabled even if WebSphere administrative security is not enabled.

3.1.4 Stopping the application server

While the command to start the application server is still the same when administrative security is enabled, stopping the server requires extra information. You must specify a user ID with administrator role rights, or the primary administrative user name specified in the user account repository and its password, in the `stopServer` command:

```
<WebSphere_home>\bin\stopServer.bat <server_name> -username <userID>
-password <password>
```

For WebSphere Application Server running under a UNIX-based operating system (OS), the previously mentioned command (the UNIX equivalent) carries a serious security problem. Anyone who uses the `ps -ef` command while the `stopServer` process is running can see the user ID and the password.

To avoid this problem:

1. If you are using the SOAP connection type (default) to stop the server, edit the `<WebSphere_home>\profiles\<profilePath>\properties\soap.client.props` file. Then, change the values of the following properties:

```
com.ibm.SOAP.securityEnabled=true
com.ibm.SOAP.loginUserId=<user ID>
com.ibm.SOAP.loginPassword=<password>
```

Again, the user ID `<user ID>`, with its password `<password>`, is the user ID with administrator role rights or the primary administrative user name defined in the user account repository.

2. Encode the `com.ibm.SOAP.loginPassword` property value as follows:

```
<WebSphere_home>\bin\PropFilePasswordEncoder.bat soap.client.props  
com.ibm.SOAP.loginPassword
```

Examine the result and remove the `soap.client.props.bak` backup file, that was created by the previous command. This file contains the unencrypted password.

3. Make sure that proper file access rights for sensitive WebSphere Application Server files, such as properties files and executable files, are set. At a minimum, ensure that permissions prevent general users from accessing these files. WebSphere administrators must be the only users that are granted access to these files. For optimal security, access to the entire WebSphere directory tree must be removed for general users.

Whether administrative security is enabled or disabled, stop the WebSphere Application Server as follows:

```
<WebSphere_home>\bin\stopServer.bat <server_name>
```

3.2 Disabling administrative security

There are several ways to disable administrative security. The easiest method is to use the Administrative Console and select **Security** → **Secure administration, applications, and infrastructure**. However, this means that the application server must already have been started. For some reason, if the application server cannot be started, for example, because of a misconfigured user account repository, you can disable administrative security by using the command line.

To disable administrative security:

1. At the command prompt, type the following command:

```
<WebSphere_home>\bin\wsadmin.bat -conntype NONE
```
2. When the system command prompt redisplay, type the following command:

```
securityoff
```
3. Type `quit` and restart the application server.

This procedure works without any problems. However, if it fails, you can disable administrative security by directly editing the security.xml file in <WebSphere_home>\profiles\<profilePath>\config\cells\<cell_name>\.

In addition change the security attribute enabled="true" to enabled="false". Some other properties (Example 3-1), such as Java 2 security and application security, are also in this file. However, use care when modifying this file directly.

Example 3-1 Content snippet of the security.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<security:Security xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
...
  xmi:id="Security_1" useLocalSecurityServer="true"
  useDomainQualifiedUserNames="false" enabled="true"
cacheTimeout="600"
  issuePermissionWarning="false" activeProtocol="BOTH"
enforceJava2Security="true" enforceFineGrainedJCASecurity="false"
appEnabled="true" dynamicallyUpdateSSLConfig="true"
  activeAuthMechanism="LTPA_1" activeUserRegistry="WIMUserRegistry_1"
  defaultSSLSettings="SSLConfig_1">
<authMechanisms ...
...
...
</security:Security>
```

Administrative security: Enable administrative security for your infrastructure. Administrative security must only be disabled to fix a problem that stems from a situation in which WebSphere security is failing.

3.3 Administrative roles

As in WebSphere Application Server V5.0, the administration subsystem of WebSphere Application Server V6.1 uses the J2EE role-based authorization concept. Three new roles are introduced in V6.1 for a total of seven defined roles for performing administrative tasks (Table 3-2 on page 60).

Table 3-2 WebSphere administrative roles

Role	Description
Monitor	Least privileged. Allows a user to view the WebSphere configuration and current application server state.
Configurator	Monitor privilege in addition to the ability to change the WebSphere configuration.
Operator	Monitor privilege in addition to the ability to change runtime state, such as starting or stopping servers.
Administrator	Operator, configurator, and iscadmins privilege, in addition to additional privileges granted solely to the administrator role, such as: <ul style="list-style-type: none"> ▶ Modifying the primary administrative user and password ▶ Create, update, and delete users and groups ▶ Enabling or disabling administrative and Java 2 security Note: An administrator cannot map users/groups to administrative roles.
iscadmins	<i>Only available for administration console users.</i> Allows a user to manage users and groups in the Federated repositories.
Deployer	<i>Only available for wsadmin users (not administration console).</i> Allows a user to change configuration and runtime state on applications using wsadmin .
Admin Security Manager	Allows a user to map users and groups to administrative roles through the administrative console, or through wsadmin for fine-grained security. Also, when fine grained administrative security is used, users granted this role can manage authorization groups.

Effectiveness: The administrative roles are effective only when administrative security is enabled.

The primary administrative user specified when enabling administrative security is automatically mapped to the Administrator and AdminSecurityManager roles. Therefore, it is not necessary to manually add this identity to either of these administrative roles.

Users and groups, as defined by the user account repository, may be mapped to administrative roles. To enable a new mapping, it is necessary to save the changes to the master configuration and restart the server. For this reason, map groups to administrative roles so that users can be added to the groups appropriately (therefore, the users are mapped to administrative roles) without the requirement to restart the WebSphere server.

3.3.1 Mapping a user to an administrative role

In order for a user to perform an administrative action, its identity must be mapped to an administrative role. To map a user to an administrative role:

1. From the Administrative Console, select **Users and Groups** → **Administrative User Roles**.
2. Click **Add**.
3. Under General Properties:
 - a. In the User field, enter a user name. This user must be defined in the user account repository that is to be active when administrative security is enabled.
 - b. Select the appropriate administrative role. More than one role may be selected. See Figure 3-3.
 - c. Click **OK**. If the user cannot be found in the registry, an error occurs.
4. Ensure that the new mapping is in the Administrative User Roles list.
5. Click **Save** to save the change to the master configuration.

Active user: The recently added user is active only after the server is restarted.

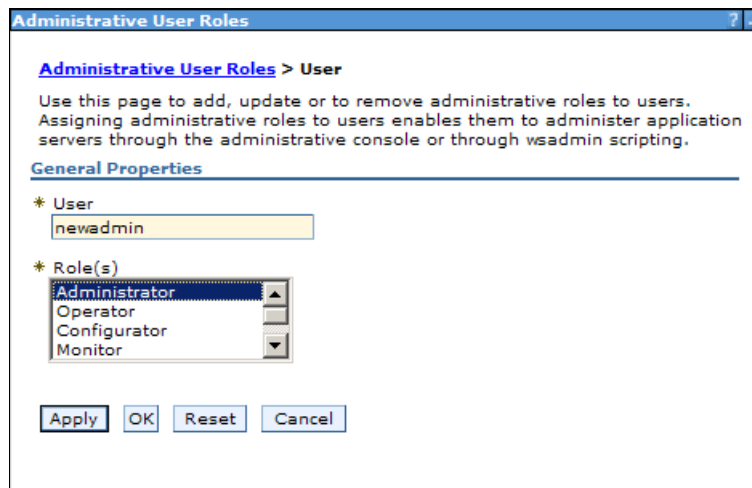


Figure 3-3 Mapping a user to an administrative role

3.3.2 Mapping a group to an administrative role

As mentioned earlier, map groups to roles rather than users. Mapping a group is fairly similar to mapping a user:

1. From the Administration Console, click **Users and Groups** → **Administrative Group Roles**.
2. Click **Add**.
3. Under General Properties:
 - a. Map either a specific group or a special subject (Figure 3-4):
 - To *map a specific group*, in the Specify group field, enter the group name. This group must be defined in the user account repository that becomes active when administrative security is enabled.
 - To *map a special subject*, select the **Special subjects** option and the appropriate subject from the drop-down list. A special subject is a generalization of a particular class of users. The *All Authenticated special subject* means that the access check of the administrator role ensures that the user making the request has at least been authenticated. The *Everyone special subject* means that anyone, regardless of whether they are authenticated, can perform the action, as though no security were enabled.

Administrative Group Roles ?

Administrative Group Roles > Group

Use this page to add, update or to remove administrative roles to groups. Assigning administrative roles to groups enables them to administer application servers through the administrative console or through wsadmin scripting.

General Properties

Group

Enter group name

Group name
operGroup

Select from special subjects

Special subjects
EVERYONE

* Role(s)

Administrator
Operator
Configurator
Monitor

Apply OK Reset Cancel

Figure 3-4 Mapping a group to an administrative role

- b. Select the appropriate administrative role. You can select more than one role.
- c. Click **OK**. If the group cannot be found in the registry, then an error occurs.
4. Ensure that the new mapping is in the Administrative Group Roles list.
5. Save the change to the master configuration by using the link at the top of the window. Then restart the server.

3.3.3 Fine-grained administrative security

WebSphere Application Server V6.1 offers new functionality for enforcing a fine-grained application of administrative security roles for **wsadmin** users. With this new functionality, user and group authorization can be granted within a specific scope, instead of the default cell-wide access that administration console users and groups receive.

Fine-grained access is granted by performing the following steps, which you must perform by using the **wsadmin** interface:

1. Connect to your application server with **wsadmin**. Specify a user that has the **AdminSecurityManager** role for the cell.
2. Create an *authorization group*:

```
$AdminTask createAuthorizationGroup {-authorizationGroupName
itsoAuthGroup}
```

3. Add resources to the authorization group:

```
$AdminTask addResourceToAuthorizationGroup {-authorizationGroupName
itsoAuthGroup -resourceName Server=server1}
$AdminTask addResourceToAuthorizationGroup {-authorizationGroupName
itsoAuthGroup -resourceName Application=DefaultApplication}
```

You can add the following resource instances to an authorization group:

- Cell
- Node
- ServerCluster
- Server
- Application
- NodeGroup

4. Add users or groups with security roles to the authorization group:

```
$AdminTask mapUsersToAdminRole {-authorizationGroupName  
itsoAuthGroup -roleName administrator -userids amy}  
$AdminTask mapGroupsToAdminRole {-authorizationGroupName  
itsoAuthGroup -roleName deployer -groupids itsodeploy}
```
5. Restart the application server for the changes to take effect, similar to console Administrative User/Group roles.

With these resource and role authorizations, Amy (in our example) now has administrator rights for the application server `server1` and the default application, through `wsadmin`. Also, members of the `itsodeploy` group can change the configuration and runtime states for the default application by using `wsadmin`.

Note: Fine-grained administrative security roles allow you to grant users and groups access to specific resource instances through `wsadmin`. This functionality is not available for administration console users.

Alternatively, you can run any of the previous commands by using the `{-interactive}` parameter to interactively walk through that configuration step by step. For more information about managing fine-grained administrative security roles, see the “Fine-grained administrative security” topic in the WebSphere Application Server V6.1 Information Center. In addition, see the “Commands for the AuthorizationGroupCommands group of the AdminTask object” topic, which contains more information about the available commands for managing the authorization group settings.

3.4 Naming service security: CosNaming roles

The J2EE role-based authorization concept has been extended to protect the WebSphere Common Object Request Broker Architecture (CORBA) naming service (CosNaming) to increase the granularity of its security control. In doing so, WebSphere can gain better control for a client program that accesses the content of the WebSphere Name space. There are generally two ways in which client programs make a CosNaming call:

- ▶ Through the Java Naming and Directory Interface (JNDI)
- ▶ If CORBA clients invoke CosNaming methods directly

In Chapter 9, “Client security” on page 207, several examples of J2EE and thin Java application clients that use the JNDI or CosNaming method call are explained. In order for some of these sample clients to work, at least a *CosNaming read role* to the CosNaming service must be granted to everyone.

This is the default setup for WebSphere. Table 3-3 shows all the four CosNaming roles.

Table 3-3 CosNaming roles

Role	Description
Cos Naming Read	Users are allowed to perform queries of the WebSphere Name Space, such as through the JNDI lookup method. The special subject Everyone is the default policy for this role.
Cos Naming Write	Users are allowed to perform write operations such as JNDI bind, rebind, or unbind, and also CosNamingRead operations. The special subject, AllAuthenticated, is the default policy for this role.
Cos Naming Create	Users are allowed to create new objects in the Name Space through such operations as JNDI create Subcontext, and perform CosNamingWrite operations. The special subject AllAuthenticated is the default policy for this role.
Cos Naming Delete	Users are able to destroy objects in the Name Space, for example, using the JNDI destroySubcontext method, as well as perform CosNamingCreate operations. The special subject AllAuthenticated is the default policy for this role.

Effectiveness: CosNaming roles are only effective when administrative security is enabled.

3.4.1 Mapping a user or a group to a CosNaming role

The process of mapping a user or group to a CosNaming role is similar to mapping a user or group to an administrative role. To map CosNaming roles, click **Environment** → **Naming** → **CORBA Naming Service Users** for user mappings and **Environment** → **Naming** → **CORBA Naming Service Groups** for group mappings.

3.4.2 Applying CosNaming security: An example

In this section, we show a simple, practical example of the use of CosNaming security. WebSphere Application Server provides a Java application client <WebSphere_home>\bin\dumpNameSpace.bat file, which is useful for listing all of the CORBA naming services available in the server.

When running dumpNameSpace.bat in a secure WebSphere environment, the Login at the Target Server window (Figure 3-5 on page 66) opens. This window

opens when the `com.ibm.CORBA.loginSource` property is set to “prompt” in the `sas.client.props` CORBA client configuration file. You can either enter *any* correct user ID and password defined in your user registry and click **OK** or simply click **Cancel**.



Figure 3-5 A window prompted by the `dumpNameSpace.bat` Java application client

With a default setup of the WebSphere Application Server, both actions must run without problems because the CosNaming read rights role is valid for everyone (Figure 3-6).

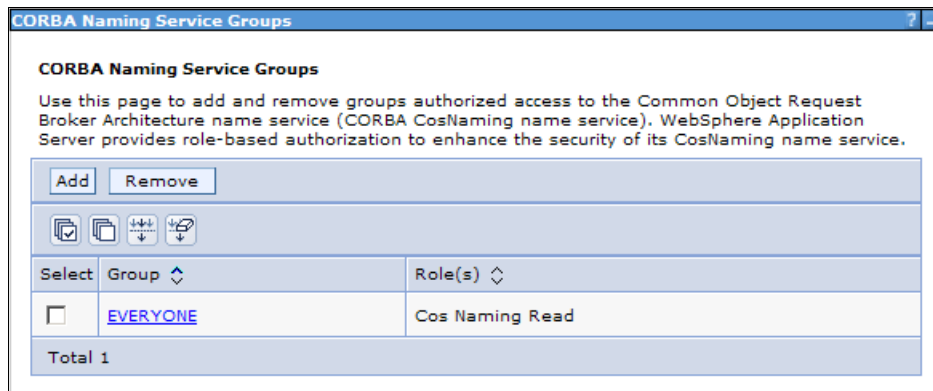


Figure 3-6 Default CosNaming security for WebSphere Application Server

To restrict the access to CORBA naming service by allowing read access only to authenticated users:

1. From the Administrative Console, click **Environment** → **Naming** → **CORBA Naming Service Groups**.
2. Click **Remove** to remove the entry for the special role group EVERYONE.
3. Click **Add** to add a new entry giving CosNaming read rights for the special group ALL_AUTHENTICATED.
4. Save the setup and restart the WebSphere Application Server.

Figure 3-7 shows the final setup for the CosNaming security.



Figure 3-7 Customized CosNaming security

After WebSphere has been started, running the `dumpNameSpace.bat` Java application client only gives good results if you enter a correct user ID and password during the authentication process. Otherwise, the WebSphere Application Server throws an exception as follows:

```
org.omg.CORBA.NO_PERMISSION.
```

Important: Granting read access to EVERYONE presents a small security risk. Therefore it is better to keep the CosNaming security settings as presented in this chapter (see Figure 3-7). If you experience unexpected results in applications that use the CORBA naming service that you cannot resolve with application security roles, add the default CosNaming security entry to the configuration as shown in Figure 3-6 on page 66. This security risk can be mitigated by ensuring that your WebSphere Application Server infrastructure is protected from other systems by firewalls.



SSL administration and configuration management

WebSphere Application Server uses the Secure Sockets Layer (SSL) protocol to provide Transport Layer Security (TLS), which allows for secure communication between a client and application server. The SSL configuration options in WebSphere offer full end-to-end management, including certificate management, individual endpoint SSL mappings, and scoped association of SSL configurations and key stores.

WebSphere Application Server V6.1 has a default SSL configuration that is set up during installation. This configuration is *CellDefaultSSLSettings* or *NodeDefaultSSLSettings*, depending on the type of profile installation. In previous releases, SSL settings were applied to transports on each individual server. In this release, SSL configurations are centrally managed by default, with changes able to be applied as widely as the cell-scope, or as narrowly as a particular endpoint on a specific application server. The SSL configuration associations are inherited. Therefore, the number of associations can be limited by only specifying unique configurations for the highest level management scope that require them. Additionally, separate SSL settings can be applied to the inbound or outbound communication topologies separately if required.

You can manage all SSL configurations and settings in the Administrative Console by selecting **Security** → **SSL certificate and key management** on the left side. From this page, by using the links under Related Items, you can

manage the cell-scoped SSL certificates, keys, and configurations. Alternatively, if you select **Manage endpoint security configurations**, you can manipulate the SSL settings for narrower scopes. As with previous releases, SSL mappings for the application server Web container transports can still be set within the application server configurations themselves by overriding the centrally managed associations.

Many new features are available in WebSphere Application Server V6.1 for SSL management. Some of the most obvious changes include the ability to manage key stores and certificates within the administrative console, certificate expiration management, and dynamic SSL configuration updates. This chapter covers most of these changes as it walks you through the creation of a new centrally managed SSL configuration.

4.1 Creating a new SSL key store entry

For more information about creating key stores, see *WebSphere Security Fundamentals*, REDP-3944. To create a new configuration for a previously created Java key store:

1. Configure a new SSL key store. Click **SSL certificate and key management** → **Manage endpoint security configurations** as shown in Figure 4-1 on page 71. Select the scope for your new key store as cell, nodegroup, cluster, node, server, or endpoint. For key store configuration, inbound or outbound topology does not matter. The new key store is available for both in that scope after it is created.

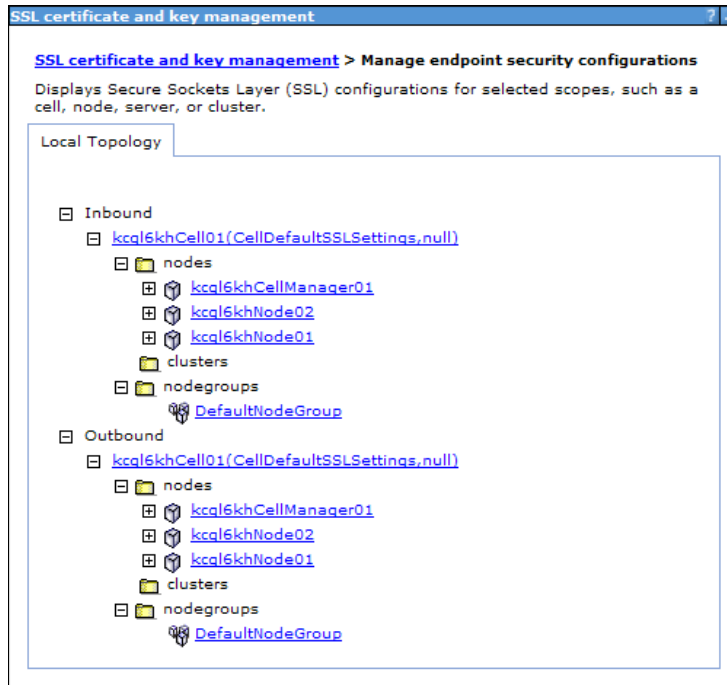


Figure 4-1 Endpoint security configuration management

- From the selected endpoint configuration page (Figure 4-2), under Related Items, click **Key stores and certificates**.

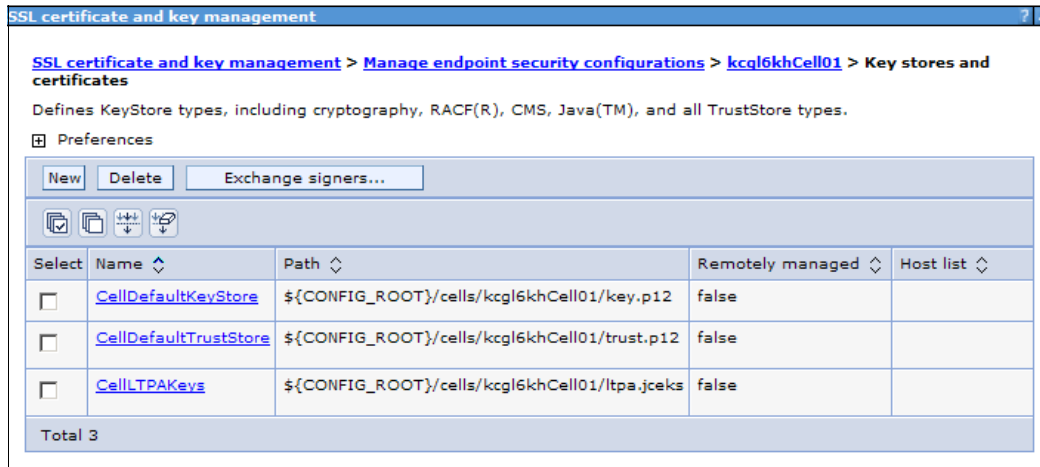


Figure 4-2 Key stores and certificates management

As shown in Figure 4-2, the default key stores are stored within the configuration repository directory tree. While keeping your custom key stores in these locations is entirely optional, it does make it easier for administration and propagation if you store them in the configuration repository at the appropriate scope level.

3. Click **New** to create your new key store.
4. Complete the form as shown in Figure 4-3 on page 73:
 - a. Set the name for your key store entry, for example, as Node01WebKeyStore.
 - b. Enter the path and file name for your key store, for example, `${CONFIG_ROOT}/cells/kcg16khCell01/nodes/kcg16khNode01/WebKey.jks`. WebSphere environment variables are valid. You can use them as shown in Figure 4-2 on page 71.
 - c. Enter and confirm the password for the key store, and select the type from the list.
 - d. Use the *Remotely managed* option for key stores that are not located on the same node as those the administration console is run from. The host list is then used in conjunction with a remote MBean call to manage the key store on each endpoint host.
 - e. If you have some other mechanism for managing your key store and its certificates, select the **Read only** option to ensure that WebSphere does not alter the key store. You can use the *Read only* item for two reasons:
 - If you do not want WebSphere Application Server to update certificates and have some other mechanism to handle the managing of your key store
 - If the particular key store type does not support writable key store
The key store type of JCERACFKS at JDK™ 5.0 does not support writable key store and must use *Read only*.
 - f. If the key store requires initialization before you can use it for cryptographic operation, select **Initialize at startup**.
 - g. Specify the use of cryptographic hardware by selecting **Enable cryptographic operations on hardware device**.

Note: If this is an WebSphere Application Server configuration for IBM z/OS, the “Enabled cryptographic operations on hardware device” check box (as shown in Figure 4-3 on page 73) must not be changed. In IBM z/OS, cryptography is dictated by the key store type.

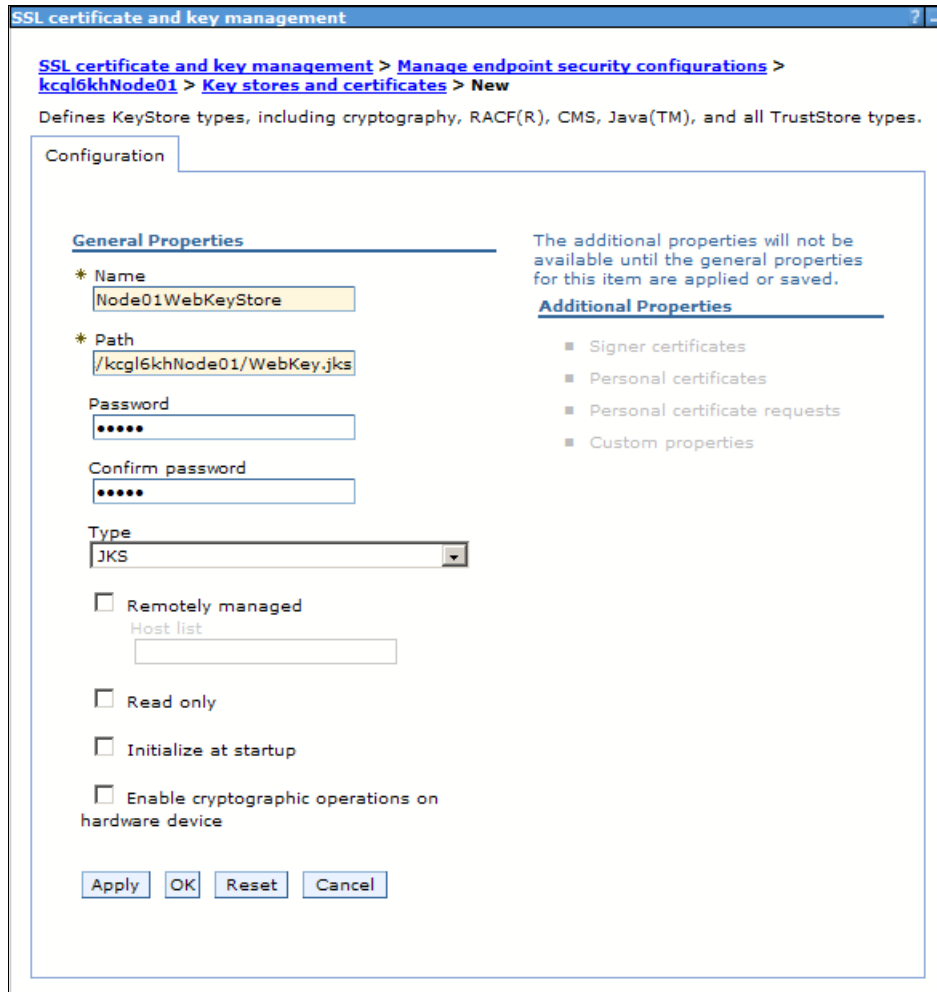


Figure 4-3 Creating a new SSL key store entry

- h. Click **OK**.
5. Save the WebSphere configuration.

You must use the same steps when adding a trust store to the WebSphere configuration.

4.2 Managing SSL certificates

After a key store is configured in WebSphere, most of the functions commonly found in iKeyman and the Java key tool can be accessed from the administration console. WebSphere certificate management provides the following functionality:

- ▶ Create personal certificate requests
- ▶ Import signer certificates
- ▶ Receive certificates from a certificate authority
- ▶ Create self-signed certificates
- ▶ View certificate properties
- ▶ Extract certificates
- ▶ Exchange signer certificates between key stores
- ▶ Delete certificates

Beyond the administration console, the command-line tools also have the ability to add unknown certificates to the default trust store. However, trusting unknown certificates must be done with care and only when you are certain of the connection you are making.

4.2.1 Expiring certificates

With the key store and certificate management interface, you can also manage your certificate expirations with the administration console. If you select **Manage certificate expirations** from the SSL certificate and key management page, you can configure the certificate expiration monitor. The monitor tracks all certificates based upon configured schedule and threshold. You can configure the monitor to log expiration notifications, or send e-mail expiration alerts to a list of addresses, or both.

In addition to certificate expiration monitoring, certificates can be updated and replaced, with self-signed certificates able to be replaced automatically, as they near expiration. Certificate updates can then be pushed out if a dynamic SSL configuration update is enabled. For more information about dynamic SSL configuration updating, see “Dynamic SSL configuration updates” on page 78.

4.2.2 Managing Web server and plug-in certificates

In addition to managing SSL certificates that are used by application servers, the WebSphere Administration Console can manage certificates and key stores that are used by your Web servers and WebSphere plug-ins.

For Web servers, complete end-to-end management of certificates and key stores is only possible for configurations that are managed through the administration console. For example, creating a new SSL-enabled virtual host on a federated Web server (with no previous SSL) accomplishes the following tasks:

- ▶ Creates a new key store and configuration entry in the WebSphere configuration
- ▶ Associates the key store with the Web server
- ▶ Assigns the specified certificate alias to the virtual host
- ▶ Propagates the new key store to the Web server node

See Figure 4-4.

Note: If your Web server already has a globally defined key store that is managed through the WebSphere Administration Console, and if you create a new SSL-enabled virtual host, you are prompted to choose a certificate alias from the existing key store. To view your global Web server security settings, click **Servers** → **Web servers** → **<Web server name>**. Then under the *Configuration settings* section, select **Global Directives**.

Create new security enabled virtual host

Create new security enabled virtual host

→ Step 1: Specify key store properties

Step 2: Specify virtual host properties

Step 3: Confirm new security enabled virtual host

Specify key store properties

A key store file with a self signed certificate will be created in the WebSphere repository and propagated to target Web server machine upon completion of this wizard. Web server key store management and propagation to target Web server machine can be done through WebSphere console at a later time.

* Key store file name
webserver1Certificates

* Target key store directory
\$(WEB_INSTALL_ROOT)/

* Key store password

* Verify key store password

* Certificate alias
webserver1

Next Cancel

Figure 4-4 Creating a new virtual host and key store in the administration console

In addition to managing certificates for the WebSphere plug-in key store, the plug-in key store configuration can also be managed from the console (Figure 4-5). To manage your Web server's plug-in configuration, click **Servers** → **Web servers** → **<Web server name>**. Then under Additional Properties, select **Plug-in properties**.

The screenshot displays two sections for configuring Web server plug-in files and key stores. The first section, titled "Repository copy of Web server plug-in files:", includes a text field for "Plug-in configuration file name" containing "plugin-cfg.xml" with a "View" button, two checked checkboxes for "Automatically generate the plug-in configuration file" and "Automatically propagate plug-in configuration file", a text field for "Plug-in key store file name" containing "WAS6PluginCertifica", and two buttons: "Manage keys and certificates" and "Copy to Web server key store directory". The second section, titled "Web server copy of Web server plug-in files:", includes a text field for "Plug-in configuration directory and file name" containing "C:\IBM\HTTPServer" and a text field for "Plug-in key store directory and file name" containing "C:\IBM\HTTPServer".

Figure 4-5 Configuring the Web server plug-in files and key stores

4.3 Creating a new SSL configuration

While central management makes it easy to use a single SSL configuration for securing an entire cell, create separate SSL configurations for the different transports in your application server. However, you can use a single key store to manage certificates that coincide with different SSL configurations to secure various transports. You can secure the following transports with an SSL configuration:

- ▶ Hypertext Transfer Protocol (HTTP)
- ▶ Lightweight Directory Access Protocol (LDAP)
- ▶ Internet InterORB Protocol (IIOP)
- ▶ Simple Object Access Protocol (SOAP)
- ▶ Session Initiation Protocol (SIP)
- ▶ Service Integration Bus (SIB)

In WebSphere Application Server V6.1, it is easier to configure SSL attributes. With the integration of key store and certificate management, specific certificates from a key store can be associated with an SSL configuration. Also, many features prepopulate lists based on input.

To create a new SSL configuration:

1. From the Manage endpoint security configurations page, choose your scope. In the Related Items section, select **SSL configurations** and click **New**.
2. On the SSL certificate and key management page (Figure 4-6):
 - a. For Name, type a name for the new configuration, for example, Node01WebSSL.
 - b. From the two lists, select your trust and key stores that you created.
 - c. Click **Get certificate aliases** to populate the list of available server and client certificates.
 - d. If the selected the Default client certificate alias and Default server certificate alias are not the aliases you want for this configuration, choose your aliases from the list of available ones.
 - e. Click **Apply** to create the basic configuration.

The screenshot shows a web browser window titled "SSL certificate and key management". The breadcrumb navigation is "SSL certificate and key management > Manage endpoint security configurations > kcgl6khNode01 > SSL configurations > New". Below the breadcrumb, there is a description: "Defines a list of Secure Sockets Layer (SSL) configurations." The main content area is titled "Configuration" and contains a form with the following fields and controls:

- General Properties** (Section Header)
- Name**: Text input field containing "Node01WebSSL".
- Trust store name**: Dropdown menu with "Node01WebTrustStore" selected.
- Keystore name**: Dropdown menu with "Node01WebKeyStore" selected. A "Get certificate aliases" button is positioned to the right of this dropdown.
- Default server certificate alias**: Dropdown menu with "webkey" selected.
- Default client certificate alias**: Dropdown menu with "webkey" selected.
- Management scope**: Text input field containing "(cell):kcgl6khCell01:(node):kcgl6khNode01".
- Buttons: "Apply", "OK", "Reset", and "Cancel".

On the right side of the form, there is a note: "The additional properties will not be available until the general properties for this item are applied or saved." Below this note is the **Additional Properties** section, which is currently collapsed and shows two items: "Quality of protection (QoP) settings" and "Custom properties".

Figure 4-6 Creating a new SSL configuration

3. After applying the changes, click **Quality of protection (QoP)** to perform further configuration settings:
 - a. Enable **Client authentication** to either support or require mutual certificate authentication between peers.
 - b. Under *Protocol*, define the protocol that you want to use to secure the transport. The options are SSL_TLS, SSL, SSLv2, SSLv3, TLS, and TLSV1.
 - c. Under the Provider, configure the Java Secure Socket Extension (JSSE) provider. This is the code that performs the cipher and decipher tasks.
 - d. Define the Cipher suite settings, which specify the encryption algorithms that are accepted by the server. By selecting a cipher suite group, only ciphers that meet your chosen group security level are made available. You can then choose which encryption algorithms to keep or remove. Make sure that matching ciphers are listed both on the server and client sides. Otherwise, the communication does not work.
4. Click **OK**. Then save the configuration for WebSphere.

You can create as many SSL configurations as you require. After the configurations are available, you can use them when configuring secured transports and other SSL enabled endpoints.

4.4 Additional SSL configuration attributes

There are a couple of other configuration options for SSL that are available from the main SSL certificate and key management page.

4.4.1 Federal Information Processing Standard

To enable FIPS support, select **Use the United States Federal Information Processing Standard (FIPS) algorithms** on the SSL certificate and key management page. When this option is selected, the LTPA implementation uses IBMJCEFIPS. IBMJCEFIPS supports the United States FIPS-approved cryptographic algorithms for Data Encryption Standard (DES), Triple DES, and Advanced Encryption Standard (AES).

4.4.2 Dynamic SSL configuration updates

If you select **Dynamically update the run time when SSL configuration changes occur**, all SSL-related attributes that change are read from the configuration dynamically after they have been saved. They are then

implemented for new connections. For outbound SSL endpoints, all outbound connections inherit the new configuration changes, because new connections are established for each request. For inbound SSL endpoints, only changes that are implemented by the SSL channel are affected by dynamic updates.

For more information about dynamic SSL configuration updates, see the “Dynamic configuration updates” topic in the WebSphere Application Server V6.1 Information Center.

Note: The Object Request Broker (ORB) and Admin SOAP inbound SSL socket factories are not affected by dynamic configuration changes. You must restart the server for SSL configuration changes to take effect on these protocols.

4.5 Trust managers

A *trust manager* is a class that is invoked during SSL handshakes to make trust decisions about remote endpoints requesting connections. The default trust manager, either the `IbmX509` or `IbmPKIX`, is used to validate the signature and expiration of certificates, while additional custom trust managers can be plugged in to perform extended certificate and host name checks.

The `IbmX509` trust manager provides basic peer certificate validation based on the trusted signer certificates present in the SSL configuration's trust store. Because of this, remove the unverified self-signed signer certificates and default root certificates, from certificate authorities, that you do not need.

The `IbmPKIX` trust manager can replace the `IbmX509` for trust decisions in an SSL configuration. Standard certificate validation is provided, similar to the `IbmX509` trust manager, but it also provides *extended certificate revocation list (CRL) checking*, where it checks that certificates contain CRL distribution points.

Note: Using the `IbmPKIX` trust manager further secures your application serving environment. It does this by checking that clients are presenting valid certificates and that those certificates have not been revoked by the certificate authority because of their compromised status. However, keep in mind that this introduces additional overhead that can affect performance.

4.5.1 Custom trust managers

If your environment requires that additional trust checks be implemented, then you can develop and configure a custom trust manager. When developing a custom trust manager, keep in mind that the trust manager class must implement the standard interface as follows:

```
javax.net.ssl.X509TrustManager
```

Implementing the following interface is optional:

```
com.ibm.wsspi.ssl.TrustManagerExtendedInfo
```

The custom trust manager must be packaged as a Java archive (JAR) file and be in the `<WebSphere_home>\lib\ext` directory in order for it to be configured properly and loaded by the application server. To configure a new trust manager in WebSphere:

1. Log in to the WebSphere Administration Console.
2. Select **Security** → **SSL certificate and key management** → **Manage endpoint security configurations**. Then select the scope for your new trust manager.
3. Under Related Items, click **Trust managers**.
4. Click **New**.
5. Complete the form as shown in Figure 4-7 on page 81 with the information about your new custom trust manager.

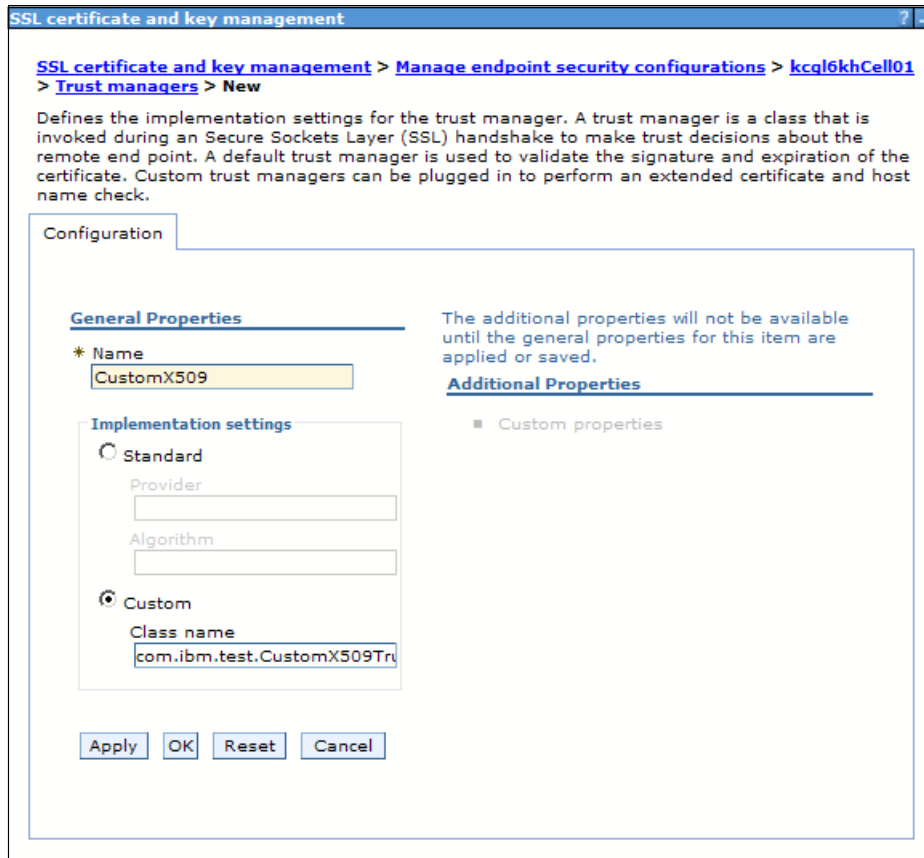


Figure 4-7 Configuring a new cell-wide custom trust manager

After you create your new custom trust manager configuration, you must associate it with an SSL configuration for it to take effect. To configure the trust managers for an SSL configuration:

1. From the SSL certificate and key management page, select **Manage endpoint security configurations**, then select the configuration scope.
2. Under Related Items, select **SSL configurations**, and click the SSL configuration you want to configure with your new trust manager. Then under Additional properties click **Trust and key managers**.

3. On the Trust and key managers configuration page (Figure 4-8), set the default trust manager and add custom trust managers.

Important: The Trust and key managers configuration option is not displayed in the initial release of WebSphere Application Server V6.1. This has been corrected in Fixpack 1 (V6.1.0.1), which is available at the following address:

<http://www.ibm.com/software/webservers/appserv/was/support/>

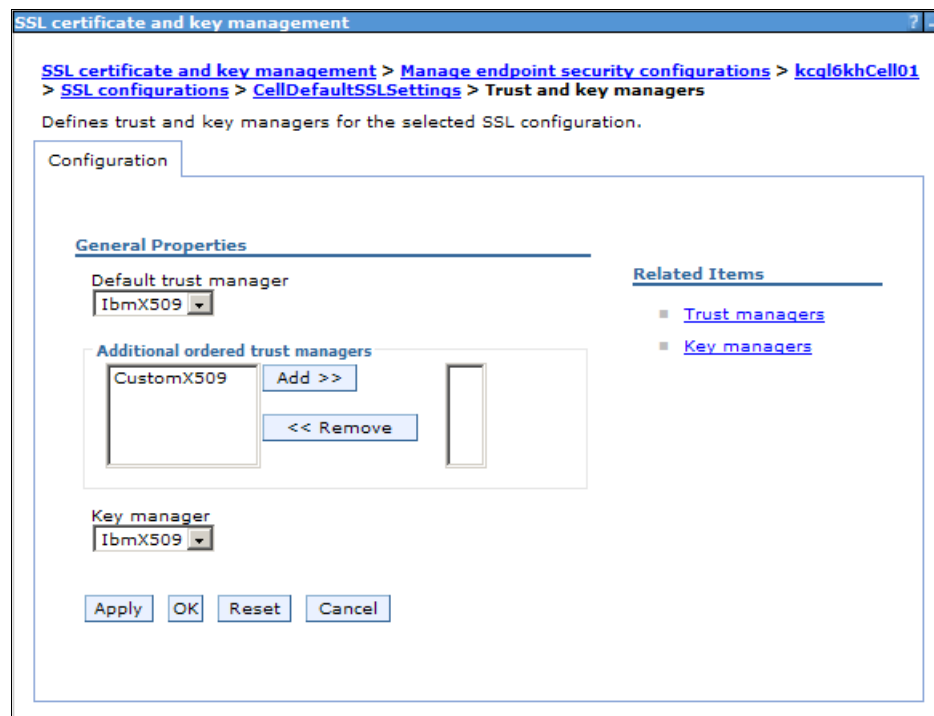


Figure 4-8 Configuring the trust and key managers for an SSL configuration

For more information about trust managers, see the “Trust manager control of X.509 certificate trust decisions” and “Creating a custom trust manager configuration” topics in the WebSphere Application Server V6.1 Information Center.

4.6 Key managers

A *key manager* is a class that is used during the SSL handshake to retrieve, by alias, the appropriate certificate from the key store. The default key manager in WebSphere is the `IbmX509`, or you can replace it with a custom key manager. Unlike the trust managers, only one key manager can be implemented for an SSL configuration at a time.

4.6.1 Custom key managers

When developing a custom key manager, keep in mind that the class must implement the following interface:

```
javax.net.ssl.X509KeyManager
```

This is because it is replacing the default `IbmX509` key manager and assuming sole responsibility for certificate alias selection.

Similar to a custom trust manager, the custom key manager must be packaged as a JAR file and be in the `<WebSphere_home>\lib\ext` directory in order for it to be configured properly and loaded by the application server. To configure a new key manager in WebSphere:

1. Log in to the WebSphere Administration Console.
2. Select **Security** → **SSL certificate and key management** → **Manage endpoint security configurations**. Then select the scope for your new key manager.
3. Under Related Items, click **Key managers**.
4. Click **New**.


5. Complete the form, as shown in Figure 4-9, with the information about your new custom key manager.

The screenshot shows a web browser window titled "SSL certificate and key management". The breadcrumb navigation is "SSL certificate and key management > Key managers > New". Below the breadcrumb is a descriptive paragraph: "Specifies the implementation settings for key managers. A key manager is invoked during a Secure Sockets Layer (SSL) handshake to determine which certificate alias is used. The default key manager (WSX509KeyManager) performs alias selection. If more advanced function is desired, define a custom key manager and select it on the Secure communications > Manage endpoint security configurations panel." Below this is a "Configuration" section with two tabs: "General Properties" and "Additional Properties". Under "General Properties", there is a field for "* Name" with the value "CustomX509". Under "Implementation settings", there are two radio buttons: "Standard" (unselected) and "Custom" (selected). Under "Standard", there are fields for "Provider" and "Algorithm". Under "Custom", there is a field for "Class name" with the value "com.ibm.test.CustomX509Ke". To the right of the "Additional Properties" tab, there is a note: "The additional properties will not be available until the general properties for this item are applied or saved." Below the "Additional Properties" tab, there is a section for "Custom properties" with a minus sign icon. At the bottom of the configuration area are four buttons: "Apply", "OK", "Reset", and "Cancel".

Figure 4-9 Configuring a new cell-wide custom key manager.

To configure a custom key manager on an SSL configuration, follow the same steps as outlined for associating a trust manager to an SSL configuration in “Custom trust managers” on page 80. From the Trust and key manager page, you can then select your custom key manager from the drop-down list (see Figure 4-8 on page 82).

For more information about key managers, see the “Key manager control of X.509 certificate identities” and “Creating a custom key manager” topics in the WebSphere Application Server V6.1 Information Center.



JAAS for authentication in WebSphere Application Server

This chapter provides a short overview of Java Authentication and Authorization Service (JAAS). JAAS is an integral part of Java 2 security, and WebSphere exploits it for authentication. By using JAAS, a user can implement and then chain together modules with the standard Pluggable Authentication Module (PAM) framework.

5.1 The importance of JAAS

JAAS is important because it gives application server users a plug-in point for authentication, allowing them to customize application and system login.

The JAAS standard specifies a set of interfaces that allow custom modules to be written. These modules can then be placed into a PAM chain that allows the container to carry out the custom authentication. A custom login module can perform principal and credential mapping, custom security token and custom credential-processing, and error-handling among other tasks.

5.2 JAAS in WebSphere

JAAS in WebSphere plays an important role. All system and application logins and Java 2 Platform Enterprise Edition (J2EE) Connector architecture (JCA) authentication aliases use JAAS. The only other plug-in point for Web authentication is the trust association interceptor (TAI) interface. TAI can be considered a little simpler to implement compared to JAAS, but JAAS provides greater flexibility.

A good example of how JAAS is used by WebSphere is when a user makes a Web request to a resource that is secured by the server. When this happens, the WEB_INBOUND JAAS login module chain is run so that the client can be authenticated. This chain, by default, contains a Lightweight Third Party Authentication (LTPA) login module and a default mapping module. The user can place other JAAS login modules in the chain.

As shown in Figure 5-1 on page 87, you can customize the login sequence by using JAAS login modules before and after the LTPA module or after the default mapping module.

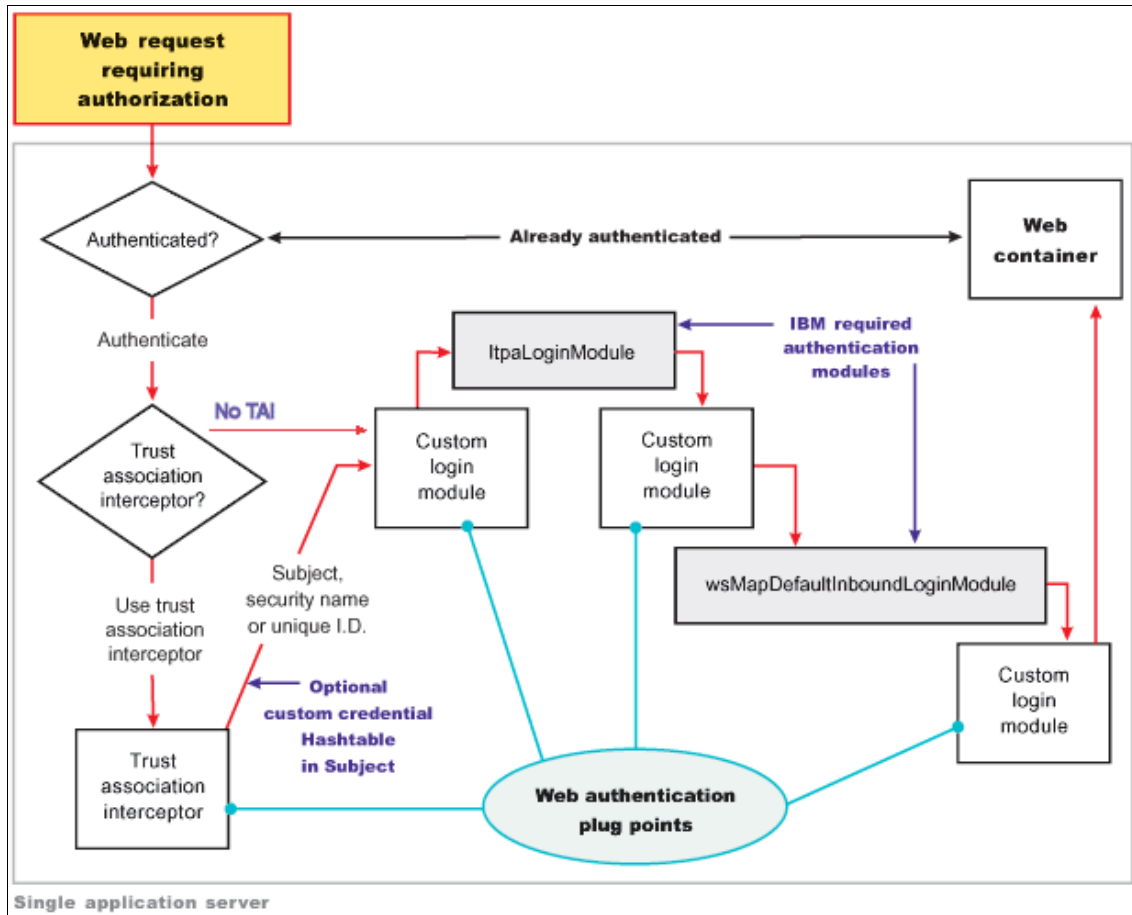


Figure 5-1 The login sequence for an inbound Web request

WebSphere extensions to JAAS

WebSphere provides the following extensions to JAAS:

- ▶ Traditionally JAAS module chains are specified in text files that require you to manually edit them. These files are similar to those in the Linux `/etc/pam.d` directory.

In WebSphere, you can perform the JAAS configuration from the Administrative Console or through `wsadmin` scripting. Although the plain file configuration is still available and supported in WebSphere, use the Administrative Console or `wsadmin` for configuration for the following reasons:

- Easy administration using the GUI (administration console only)
- Central management of configuration
- Distribution of configuration in Network Deployment environment

- ▶ WSSubject (com.ibm.websphere.security.auth.WSSubject) is an extension to the original Subject. The WSSubject implementation can return the subject in the running thread using the getSubject() method inside a doAs() method. This is not the case with the original JAAS V1.0 implementation.
- ▶ To allow the thread context class loader to load classes, a *proxy* LoginModule is responsible for loading the actual LoginModule. The reason for a proxy loader is to resolve class visibility. The proxy is an internal component. It is not going to effect application developers or administrators.

5.3 Custom JAAS login in WebSphere

JAAS provides a pluggable authentication framework for the application server. This section introduces the various components that you can plug in for JAAS and use it in WebSphere Application Server.

5.3.1 Callback handler

Callback handlers are responsible in JAAS for collecting the necessary information in the application to perform the authentication. The *callback handler*, as its name suggests, uses the callback programming model to collect information. Together with the callback handler, numerous different types of callbacks are defined that can be invoked. These callbacks, or just one, are invoked one after the other, for example, user ID callback to retrieve the user name, password callback to retrieve the user's password.

Callbacks can be implemented in different ways. They can be interactive or non-interactive (in other words, programmatic). The interactive callbacks can interact with the user (or device) in numerous ways, for example, asking for a user ID typed in from the system console. A non-interactive, programmatic callback collects information without prompting the user, for example, by reading the user ID from a properties file.

For more information about the available callback handlers in WebSphere V6, see “Built-in CallbackHandler in WebSphere” on page 237.

Custom callback handler

When writing your own callback handler, you can implement the different callbacks to collect information. For more information about writing your own callback handler, see “Custom CallbackHandler” on page 240.

5.3.2 Login module

Login modules are responsible for the actual login, including making the authentication check and creating the principal that is stored in the subject later.

WebSphere Application Server V6 comes with a few login modules implemented for various login situations. They come registered and configured for the application server. See 5.3.4, “Configuration” on page 97.

For details about how JAAS login modules work in WebSphere, see 9.6.1, “JAAS login module in WebSphere” on page 233. See also 9.6.2, “Programmatic login process” on page 235, which shows the interaction diagram for the whole login mechanism using JAAS.

Custom login module

You can also write your own login module for WebSphere Application Server. You must implement the LoginModule interface and code it by using the following methods:

- ▶ `public void initialize (Subject subject, CallbackHandler callbackHandler, Map sharedState, Map options)`
This method is responsible for initializing the login module after loading.
- ▶ `public boolean login() throws LoginException`
This method performs the actual login. This is the part where you can code the authentication check using callbacks.
- ▶ `public boolean commit() throws LoginException`
After a successful login, you are required to commit the login. You can add or insert the principal name and authentication data into the subject during the commit state.
- ▶ `public boolean abort() throws LoginException`
In case of any problem, this method aborts the login process.
- ▶ `public boolean logout() throws LoginException`
After a successful login and by the end of the session, the application requires you to log out and remove items. The `logout()` method is where you remove the items that you added or inserted during the commit state.

More information: A good resource JAAS login module requirements is the Sun Microsystems’ *Java Authentication and Authorization Service (JAAS) LoginModule Developer’s Guide* available at:

<http://java.sun.com/javase/6/docs/technotes/guides/security/>

Example 5-1 shows a simple custom login module. The code for this example is also in the additional materials ZIP file that is available for download. See Appendix B, “Additional material” on page 543.

Example 5-1 Custom login module: CustomLoginModule.java

```
/*
 * Created on 12/07/2006
 */
package com.itso.was61sec.loginmodule;

import java.io.IOException;
import java.util.Map;

import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;
import javax.security.auth.spi.*;

import com.ibm.wsspi.security.auth.callback.WSTokenHolderCallback;

/**
 * Custom JAAS login module.
 *
 * @author paulw
 */
public class CustomLoginModule implements LoginModule {

    // initial state
    private Subject subject;
    private CallbackHandler callbackHandler;
    private Map sharedState;
    private Map options;
    // the authentication status
    private boolean succeeded = false;
    private boolean commitSucceeded = false;
    // username and password
    private String username;
    private String password;
    // testUser's SamplePrincipal
    private SamplePrincipal userPrincipal;

    /* (non-Javadoc)
     * @see
     javax.security.auth.spi.LoginModule#initialize(javax.security.auth.Subj
```

```

ect, javax.security.auth.callback.CallbackHandler, java.util.Map,
java.util.Map)
    */
    public void initialize(Subject subject, CallbackHandler
callbackHandler, Map sharedState,
        Map options) {
        System.out.println("DEBUG: Initializing class " +
CustomLoginModule.class);
        this.subject = subject;
        this.callbackHandler = callbackHandler;
        this.sharedState = sharedState;
        this.options = options;
        System.out.println("DEBUG: CallbackHandler: " + callbackHandler);
    }

    /*
    * This is phase one of the login process.
    *
    * (non-Javadoc)
    * @see javax.security.auth.spi.LoginModule#login()
    */
    public boolean login() throws LoginException {
        System.out.println("DEBUG: Entering login()");
        // prompt for username and password
        if (callbackHandler == null) throw new LoginException("Error: No
CallbackHandler available!");

        Callback[] callbacks = new Callback[3];
        callbacks[0] = new WSTokenHolderCallback( "" );
        callbacks[1] = new NameCallback("user name: ");
        callbacks[2] = new PasswordCallback("password: ", false);

        try {
            callbackHandler.handle(callbacks);
        } catch (IOException ioe) {
            throw new LoginException(ioe.toString());
        } catch (UnsupportedCallbackException uce) {
            throw new LoginException("Error: " +
uce.getCallback().toString());
        }

        boolean requiresLogin = ( (WSTokenHolderCallback)callbacks[ 0 ]
).getRequiresLogin();

```

```

        if ( requiresLogin ) {
            username = ((NameCallback) callbacks[1]).getName();
            password = new String(((PasswordCallback)
callbacks[2]).getPassword());
            ((PasswordCallback) callbacks[2]).clearPassword();

            // verify the username/password
            // this code is using a hard-coded user name
            // and password for the sake of simplicity
            // if you would like to see an example with a
            // registry lookup and groups please refer to:
            //
http://www-128.ibm.com/developerworks/websphere/techjournal/0508\_benantar/0508\_benantar.html
            boolean usernameCorrect = false;
            boolean passwordCorrect = false;
            if (username.equals("alison"))
                usernameCorrect = true;
            if (password.equals("passw0rd"))
                passwordCorrect = true;
            if (usernameCorrect && passwordCorrect) {
                // authentication succeeded
                succeeded = true;
                System.out.println("DEBUG: Exiting login(), returning
TRUE");
                return true;
            } else {
                // authentication failed
                succeeded = false;
                username = null;
                password = null;
                System.out.println("DEBUG: Exiting login(), returning FALSE
and raising exception");
                throw new FailedLoginException("Authentication Failed!");
            }
        } else{
            System.out.println( "DEBUG: This is a propogation login,
nothing to do." );
            return true;
        }
    }

    /*
     * This is phase two of the login process when phase one
     * succeeded for all modules.
    */

```

```

*
* (non-Javadoc)
* @see javax.security.auth.spi.LoginModule#commit()
*/
public boolean commit() throws LoginException {
    System.out.println("DEBUG: Entering commit()");
    if (succeeded == false) {
        return false;
    } else {
        // Add Principal (authenticated identity) to the Subject.
        // This is a custom Principal: SamplePrincipal. In
        // WebSphere you may want to use the WSPPrincipalImpl class
        userPrincipal = new SamplePrincipal(username);
        try {
            if (!subject.getPrincipals().contains(userPrincipal))
                subject.getPrincipals().add(userPrincipal);
        } catch (Exception e) {
            username = null;
            password = null;
            commitSucceeded = false;
            throw new LoginException(e.getClass().toString() +
e.getMessage());
        }
        username = null;
        password = null;
        commitSucceeded = true;
        return true;
    }
}

/*
* This is phase two of the login process when phase one
* failed for one or more modules.
*
* (non-Javadoc)
* @see javax.security.auth.spi.LoginModule#abort()
*/
public boolean abort() throws LoginException {
    System.out.println("DEBUG: Entering abort()");
    if (succeeded == false) {
        // local variables are already clean
        return false;
    } else if (succeeded == true && commitSucceeded == false) {
        // login succeeded but overall authentication failed
        succeeded = false;
    }
}

```

```

        username = null;
        password = null;
        userPrincipal = null;
    } else {
        // overall authentication succeeded and commit
        // succeeded, but someone else's commit failed
        logout();
    }
    return true;
}

/* (non-Javadoc)
 * @see javax.security.auth.spi.LoginModule#logout()
 */
public boolean logout() throws LoginException {
    System.out.println("DEBUG: Entering logout()");
    succeeded = false;
    commitSucceeded = false;
    username = null;
    password = null;
    userPrincipal = null;

    // If the subject is read only then we need to destroy
    // the credentials associated with the subject during
    // the commit phase. We throw an exception if they do
    // not implement the Destroyable interface.
    if (!(subject.isReadOnly())) {
        throw new LoginException("Not able to destroy principal in
subject.");
    }
    subject.getPrincipals().remove(userPrincipal);
    return true;
}
}

```

The custom login module uses a custom principal (`SamplePrincipal`). For more information about principals, see 5.3.3, “Principal” on page 95.

The login module performs the following actions:

1. Initializes the login module and instantiates the necessary objects.
2. Sets up the callback handler and the callback methods in preparation for user input.

3. Gets user input by walking through the callbacks one after the other using the callback handler.
4. Authenticates the user by using the information that is returned from the callbacks:
 - If authentication is successful, it creates a principal based on the authentication data and inserts it into the subject setup during initialization.
 - If authentication fails, the module destroys the objects and returns with a failed flag.
5. If all login modules in the chain return successful authentication results from phase one, the commit function is called. Otherwise the abort function is called.
 - The commit function places the principal into the Java Subject.
 - The abort function cleans up and destroys any sensitive information.
6. After a successful login, the application can also log out. The logout method must remove the principal from the subject and destroy the objects in the login module.

5.3.3 Principal

In JAAS, *principals* are objects that store user credentials. Principals can then be added (stored) in subjects, which is another object in JAAS to store multiple (or just one) principals. The subject is then propagated with the security context and is available for the application server to check the logged in principals.

WebSphere has its own implementation of a principal, which is `WSPPrincipal`. This principal is used internally with the security context.

Custom principal

Principals can be customized, and new ones can be implemented based on the `java.security.Principal` interface. Example 5-2 on page 96 shows an implementation of a principal, called `SamplePrincipal`. You can customize the principals to store extra information about the user, other than just the user name.

Important: WebSphere does not handle serialization of principals. The principal and login module must handle serialization and deserialization itself. For more information and examples about this, see Chapter 11, “Security attribute propagation” on page 265.

Example 5-2 Custom principal: SamplePrincipal.java

```
package com.itso.was61sec.loginmodule;

import java.io.Serializable;
import java.security.Principal;

/**
 * Custom principal for JAAS login.
 *
 * @author paulw
 */
public class SamplePrincipal implements Principal, Serializable {

    private String name;

    public SamplePrincipal(String name) {
        if (name == null) throw new NullPointerException("Illegal null
input");
        this.name = name;
    }

    /* (non-Javadoc)
     * @see java.security.Principal#getName()
     */
    public String getName() {
        return name;
    }

    public String toString() {
        return ("SamplePrincipal: " + name);
    }

    public boolean equals(Object o) {
        if (o == null)
            return false;
        if (this == o)
            return true;
        if (!(o instanceof SamplePrincipal))
            return false;

        SamplePrincipal that = (SamplePrincipal) o;
        if (this.getName().equals(that.getName()))
            return true;
        return false;
    }
}
```

```
    }  
  
    public int hashCode() {  
        return name.hashCode();  
    }  
}
```

As shown in Example 5-2, the principal is a simple Java object with attributes, a collection of set and getter methods, and a few other supporting methods.

5.3.4 Configuration

Take the compiled code for the custom login module and the dependent classes and package them in a JAR file. You can also deploy the code unbundled, in directories and files, but package it in a JAR file.

You can place the custom login module code in the following places:

- ▶ Within a .ear file for a specific enterprise application, because then it is only accessible to the specific application
- ▶ In the WebSphere Application Server shared library, but remember to edit the server's security policy file
- ▶ In the Java extensions directory (WebSphere_root\jre\lib\ext), where it is available to all applications

In the WebSphere Administrative Console, you can configure JAAS login modules by selecting **Security** → **Secure administration, applications, and infrastructure**. Expand the **JAAS** menu and you see three items, where the first two are related to LoginModule configuration.

Application logins

Application logins are the ones that your enterprise applications can use. After installation, you find three items already defined:

- ▶ ClientContainer
com.ibm.ws.security.common.auth.module.WSClientLoginModuleImpl
- ▶ DefaultPrincipalMapping
com.ibm.ws.security.auth.j2c.WSPrincipalMappingLoginModule
- ▶ WSLLogin
com.ibm.ws.security.common.auth.module.WSLoginModuleImpl

Attention: You must *not* remove or modify these definitions. Some applications might use them, and those applications can break if you change any of the definitions.

To add a new application JAAS login module configuration to the list:

1. Under Application login configuration, click **New**.
2. Provide an alias name, for example MyLoginModule.
3. Click **Apply**.

Do *not* click OK yet, you are going to define the login module first before you save the configuration.

4. Click **JAAS login modules**.
5. Click **New**.
6. Provide the fully qualified name (including package name) for your custom LoginModule implementation in the Module class name field, for example:

```
com.ibm.itso.MyLoginModuleImpl
```

Select the **Use login module proxy** check box, to ensure the class visibility for applications. For more information about the login module proxy, see the WebSphere Information Center.

Select the authentication strategy, set as REQUIRED for now. The options include: REQUIRED, REQUISITE, SUFFICIENT, and OPTIONAL. For more information about the different strategies, see the WebSphere Information Center.

7. Click **OK**.
8. Save the configuration for WebSphere.

System logins

System login definitions are similar to the application login definitions, except that they are related to the application server itself, not the applications. These definitions are used for internal login purposes, for example, LTPA, Remote Method Invocation (RMI), Web. You must remember that WebSphere Application Server only authenticates a client when the resource being accessed has security enabled.

You can write your own login modules and use them internally, but we recommend that you do *not* remove or change the existing ones under the System login configuration. For more information, see Chapter 11, “Security attribute propagation” on page 265.

5.3.5 Viewing the sample JAAS module in action

The easiest way to see the sample module working is to follow the instructions in 5.3.4, “Configuration” on page 97, and specify the module as part of the system’s WEB_INBOUND module chain as OPTIONAL. After restart, when you attempt to access a resource that requires authentication (including the administration console, which is why you *do not* set it to REQUIRED), the callback handler presents a login prompt.

In the module’s current form, the only successful login is to use the user name `alison` and the password `passw0rd`. Therefore, it is beneficial to add this user to the server’s security repository. Looking at the server’s SystemOut.log shows you the debug output from the sample module.

Attention: This scenario is for *testing purposes* only. Changing the System login module chains requires careful planning and consideration of the environment’s security. A misconfigured System login chain *can stop any* administration from occurring because all requests can be rejected.

5.3.6 Programming authentication

You can customize the entire authentication process in WebSphere for various situations. You can customize and plug-in any or all of the components of the JAAS login. WebSphere comes with predefined and preconfigured components for every part of the login mechanism. If you require custom behavior in your application, you must plug in your own implementation.

For details about programmatic login using JAAS, see 9.6.2, “Programmatic login process” on page 235.

5.4 J2C authentication data

J2EE Connector security, known as J2C, allows secure connections to be made from J2EE applications. The J2C authentication data entries configure the access to external resources, for example, database, messaging oriented middleware, and other J2C adapters.

Why the J2C authentication data entries are required and how are they used

When the application accesses an external resource, it occurs through a J2C adapter. The application does not authenticate itself for the external resources by providing a user ID and password. The user ID and password for the J2C adapter is defined under the J2C authentication data. When the J2C adapter requests a login, it uses a JAAS login module in WebSphere. The login module simply looks up the J2C authentication data entry from the WebSphere configuration during the login process and uses the retrieved data with the callback methods.

JAAS authentication entries are in the Administrative Console when you select **Security → Secure administration, applications, and infrastructure → Java Authentication and Authorization Service → J2C Authentication data**. You can create new entries or remove entries here.

For more information about J2C authentication data, see Chapter 17, “J2EE Connector security” on page 463.



Application security

This chapter discusses application security for WebSphere Application Server V6.1.

6.1 Application security

In previous releases of WebSphere Application Server, when global security was enabled, both administrative and application security were enabled. In WebSphere Application Server V6.1, the concept of global security is split into administrative security and application security, of which each component can be enabled separately. Application security provides application isolation and requirements for authenticating users for the applications in your environment.

Application security must be enabled in case *declarative security* is used by any application that is deployed in the application server. However, if your application relies only on *programmatic security*, for example using the `HttpServletRequest` interface method `getRemoteUser()`, where authentication is already done on the Hypertext Transfer Protocol (HTTP) server side, you are not required to enable application security. Declarative security and programmatic security are discussed further in 8.2, “Declarative J2EE security” on page 174, and 8.3, “Programmatic J2EE security” on page 197.

6.1.1 Enabling application security

By default, administrative security is enabled during installation, and application security is disabled. For application security to be enabled and take effect, you must enable administrative security. For instructions on how to configure administrative security, see Chapter 3, “Administrative security” on page 49.

To enable application security:

1. Start the WebSphere Administration Console and login.
2. Select **Security** → **Secure administration, applications, and infrastructure**.
3. In the Application security section, select **Enable application security** (Figure 6-1).

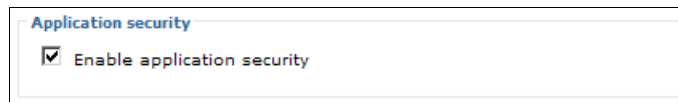


Figure 6-1 Enabling application security in WebSphere Application Server

4. Click **Apply**, and save the WebSphere configuration.
5. Restart the application server to make this change take effect.

6.1.2 Testing application security

After you restart your server in secure mode, test that security is properly enabled:

1. If *DefaultApplication* is installed, test the Web-based Basic Authentication by accessing the following URL:

```
http://<hostname>:<port>/snoop
```

The default *<port>* is 9080.

Important: The default application and other included sample applications must not be running in a production environment because they can reveal a wealth of information about your environment. They can be excellent diagnostic tools, but must be removed or stopped when not in use.

2. In the challenge login window that opens, type *any* user ID and password that is defined in the user account repository.

6.1.3 Application considerations

One of the most common problem that occurs when application security is enabled is that the `getRemoteUser()` method or `getUserPrincipal()` method of the `HttpServletRequest` interface returns a null value. This happens if, for example, authentication is done in the HTTP Server container before reaching the WebSphere Application Server. Whenever application security is enabled, WebSphere only passes the authentication token to *secure* resources within its container. To secure these resources, add a security constraint within the `web.xml` application descriptor file as shown in Example 6-1.

Example 6-1 Securing the resource /securedhello URI

```
<security-constraint>
  <display-name>Authenticated</display-name>
  <web-resource-collection>
    <web-resource-name>Authenticated Resources</web-resource-name>
    <url-pattern>/securedhello</url-pattern>
    <http-method>PUT</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <description>Authorized guest roles</description>
    <role-name>ServletGuest</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>INTEGRAL</transport-guarantee>
```

```

    </user-data-constraint>
</security-constraint>

<security-role>
  <description>Authenticated guest for servlet</description>
  <role-name>ServletGuest</role-name>
</security-role>

```

Remember to define a correct security role mapping for the role that you have added when deploying your application. In Example 6-1, the name of the role is *ServletGuest*. If the application is already installed, edit security role mapping by selecting **Applications** → **Enterprise Application** → **<your_application>** → **Security role to users/group mapping** (Figure 6-2).

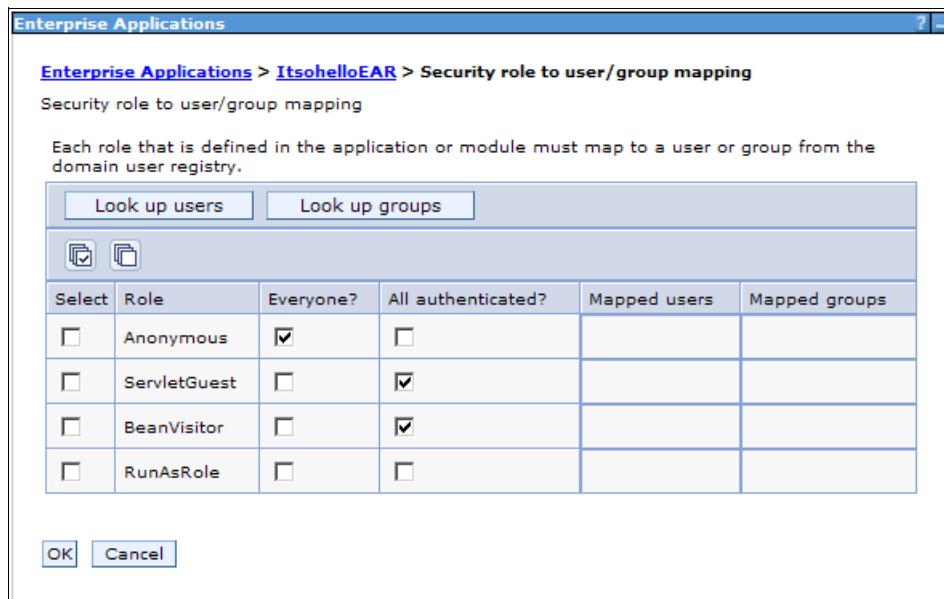


Figure 6-2 Security role mapping

For more detailed information, see Chapter 7, “Securing a Web application” on page 109. Also, consult the Java 2 Platform, Enterprise Edition (J2EE) servlet specification for further information about this subject.

Important: If application security is enabled, the `getRemoteUsers()` and `getUserPrincipal()` methods return a `null` value even if the user is logged in, unless the servlet of the JavaServer™ Pages (JSP™) is secured within the application server.

6.2 Deploying a secured enterprise application

Deploying a secured application is not much different than deploying any other (non-secured) enterprise application. The only difference is that during deployment, you can perform role mapping for users and groups, as well as run-as mapping.

6.2.1 Role mapping during application installation

During the process of running the application installation, you see the *Map security roles to users or groups* step. In this step, you have the option of selecting any of the roles and assigning a user or a group from the user registry using one of the lookups. You can also assign one of the special subjects (*Everyone* or *All authenticated*) to the role.

If you have Enterprise JavaBeans (EJB) or EJB methods with Run-As role mappings, you see the *Map RunAs roles to users* step. In this step, you can specifically assign a user name and password (an identity) to a Run-As (delegation) definition.

If you have EJB or EJB methods with Run-As system mappings, you see the *Correct use of system identity* step. In this step, you can override the default system identity with a specific user mapping. For more information about Run-As mapping, see 8.2.7, “Run-as mapping” on page 193.

If you have EJB methods without security assignments, you see the *Ensure all unprotected 2.x methods have the correct level of protection* step. In this step, you can assign a role to these methods, on a per EJB basis (not on a per method basis). You can also exclude the methods so that they cannot be accessed, or you can clear them so that they can be accessed by everyone. EJB method security is discussed further in 8.2.3, “Configuring method access control” on page 180.

These three types of mappings might be already defined in the enterprise archive. They can be defined during assembly time, just before deployment, for example in the Rational Application Developer or in the Application Server Toolkit. Even if the mappings were done previously, you can review and modify them during deployment or later, as discussed in 6.2.2, “Role mapping after installation” on page 106.

6.2.2 Role mapping after installation

After the application is installed, you can change the security settings for the application:

1. Launch Administrative Console and log in.
2. Click **Applications** → **Enterprise Applications**.
3. Select the application that you want to change.

You find the following items under the Detail Properties section (Figure 6-3):

- Security role to user/group mapping
- User Run-As roles

By selecting either of these options, you access the same configuration page as the one you saw during deployment.

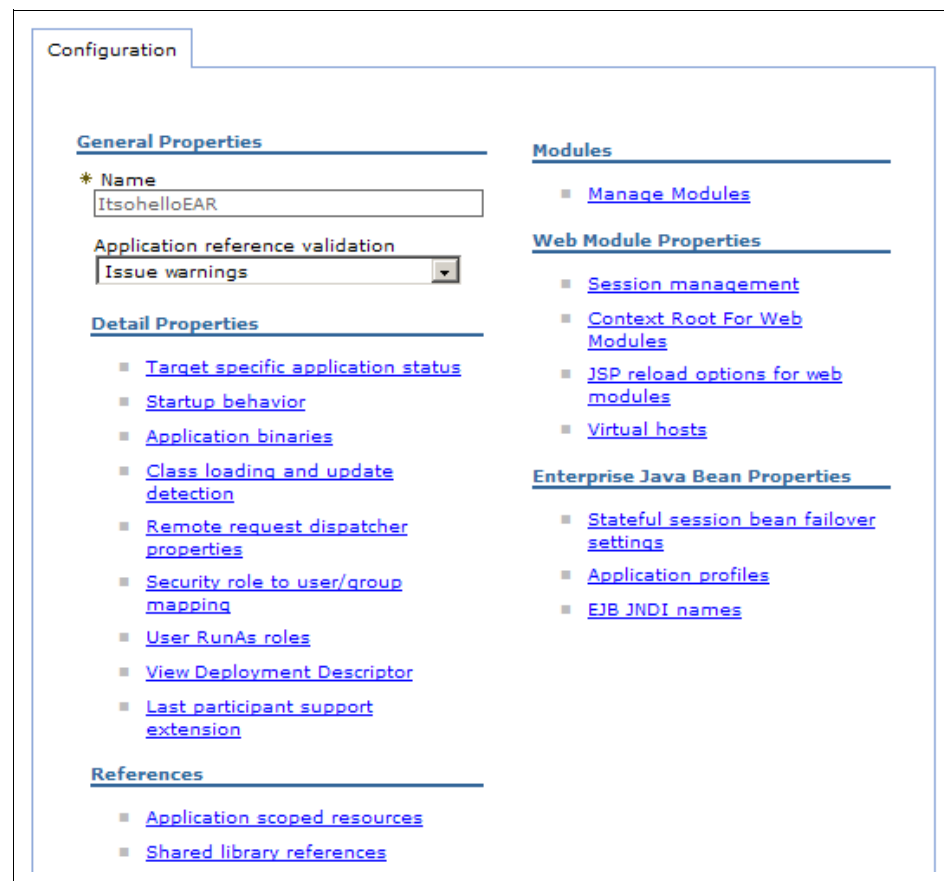


Figure 6-3 Enterprise Application configuration page

Note: The *User Run-As roles* configuration link only opens in the Administrative Console when your application uses Run-As delegation. WebSphere detects whether this configuration exists in the application and changes the interface accordingly.

The *Ensure all unprotected 2.x methods have the correct level of protection* configuration is not available after deployment. After the methods are defined as cleared, excluded or mapped to a role, this does not change.

4. Save the configuration for WebSphere after the changes.
5. Restart the Enterprise Application in order for the changes to take effect.



Securing a Web application

This chapter discusses the security aspects involved with securing Web applications. It explains how to secure the transport channels between all components and discusses the authentication and authorization options that are available for each processing component.

A Web application consists of different Web components, such as Hypertext Markup Language (HTML) pages, JavaServer Pages (JSP), and servlets. All of these components form Web pages on the server side. On the client side, a Web browser is typically used to issue a request for a Web resource. The request goes to the WebSphere Application Server. WebSphere Application Server then processes all the Web components that form the requested Web resource, creates a Web page, and sends it back as the response. The browser transforms and displays the response Web page in a more human readable format.

7.1 Transport channel

A *transport channel* refers to the communication channel between Web client and Web Application Server. The communication can be classified into different layers, each with its own functions and scope. This section focuses on the top layer of communication between Web clients and Web application servers.

Protocols define different communication types. HTTP is the protocol that is used for application communication between Web clients and Web application servers. By using only HTTP, the data flow is not encrypted. Therefore, anyone who can intercept it might understand the content.

To secure the transport channel, secured HTTP (HTTPS) is required. Typically, HTTP runs on top of the Transmission Control Protocol (TCP), which is a transport protocol, and to secure it, Secure Socket Layer (SSL) is required. In summary, for encrypted communication between Web clients (browsers) and Web application servers (WebSphere Application Server), you use HTTPS, which runs on top of an SSL secured TCP transport channel.

7.2 Securing the static content

Static Web resources are those whose content does not change over time regardless of which user accesses them or the user input data. For example, this can be a `static.html` page or a `.jpg` image file. Although WebSphere Application Server provides a mechanism to serve them, those resources must be served by the Web server. When the Web server is involved, WebSphere does not have security control over the resources served by the Web server. Therefore, transport security, authentication, and authorization must be configured for the Web server.

This section explains how to secure static content that is served by the Web server only. To secure static content served by WebSphere, see “Securing content served by WebSphere Application Server” on page 135.

In 7.2.2, “Authentication by using a Web server” on page 113, we provide an example of how to configure IBM HTTP Server to secure static content with HTTP basic authentication when the user registry is set to a Lightweight Directory Access Protocol (LDAP) directory. In 7.2.3, “Authorization by using a Web server” on page 116, we explain how to manage access to this static content by using the `.htaccess` configuration files.

Describing all the possible options for managing security in IBM HTTP Server is not within the scope of this book. For detailed information, see the product documentation for the appropriate release.

Additional products can also be used to provide the end-to-end security infrastructure. For information about how Tivoli Access Manager fits into this scenario, see Chapter 12, “Securing a WebSphere application using Tivoli Access Manager” on page 297.

7.2.1 Securing the transport channel between the Web browser and Web server

The Web browser and Web server communicate with each other over the HTTP protocol. By default, HTTP is not secured at all. To ensure the data integrity, you must use the SSL protocol with the HTTP protocol to secure the transport.

IBM HTTP Server uses the IBM proprietary SSL module and SSL configuration described in this section. If you use another Web server, consult the product documentation to see how to set up an SSL transport channel.

To begin, we created a key store of Certificate Management System (CMS) format that contains a self-signed digital certificate to secure the HTTP transport channel between the Web browser and the IBM HTTP Server. For more information about how to create a key database file that stores the necessary certificates, see the IBM Redpapers™ publication *WebSphere Security Fundamentals*, REDP-3944.

Configuring IBM HTTP Server for SSL

To configure IBM HTTP Server for SSL:

1. Open the `httpd.conf` file, which is the configuration file for IBM HTTP Server, in the `<IHS_root>\conf` directory.
2. Add the `ibm_ssl_module` definition to the end of the `LoadModule` list as shown in Example 7-1.

Example 7-1 Adding `ibm_ssl_module` definition to `httpd.conf`

```
LoadModule alias_module modules/mod_alias.so
#LoadModule rewrite_module modules/mod_rewrite.so
#LoadModule deflate_module modules/mod_deflate.so
LoadModule ibm_ssl_module modules/mod_ibm_ssl.so
```

3. Create a virtual host. Then enable and configure SSL just for this virtual host. On a global level, the IBM HTTP server still does not use SSL. Add the directives to the `httpd.conf` file as shown in Example 7-2.

Example 7-2 Adding a virtual host definition and configuring it for SSL

```
SSLDisable
Listen 0.0.0.0:443

<VirtualHost webserver01.redbook.net:443>
SSLEnable
KeyFile "C:/IBM/HTTPServer/conf/keys/IHS6Certificates.kdb"
</VirtualHost>
```

This is the most basic SSL setup, but you can use other SSL directives to set the SSL configuration more specifically to your requirements. Further explanation of these is not within the scope of this book. See the IBM HTTP Server documentation.

The directives that this book uses for configuration are explained here:

- The directive `Listen 0.0.0.0:443`, which is placed into global definition scope, makes the IBM HTTP Server listen on port 443 as well.
 - The directive `VirtualHost` starts the virtual host stanza. Make sure that you specify a TCP resolvable host name or IP address here.
 - The directive `SSLEnable` enables SSL for this virtual host only.
 - The directive `KeyFile` defines where the key database file is located.
4. Save the `httpd.conf` configuration file and restart IBM HTTP Server.

Testing SSL between Web browser and Web server

Open a Web browser and connect to the Web server using the following URL, where `<virtualhostname>` is the TCP domain resolvable name that you used with your virtual host definition:

```
https://<virtualhostname>
```

In our case, we type the following URL in the browser:

```
https://webserver01.redbook.net/
```

Because you did not specify any port with the request, the request goes to the default HTTPS server listening port, which is 443. The Web server recognizes the request, and because it comes to port 443, you can bind it to the configured virtual host.

Because SSL is enabled only for the virtual host, you can still access the Web server unsecured by HTTP on port 80, which is defined on the global scope. If you want an SSL-only configuration, specify SSL directives on the global scope without creating a virtual host, as shown in Example 7-3.

Example 7-3 Configuring SSL on global configuration scope

```
#Listen 0.0.0.0:80
Listen 0.0.0.0:443
SSLEnable
KeyFile "C:/IBM/HTTPServer/conf/keys/IHS6Certificates.kdb"
```

Also delete or comment out the default `Listen 0.0.0.0:80` directive to limit port 443 to be the only listening port when using an SSL-only configuration.

7.2.2 Authentication by using a Web server

Most Web servers can secure the files that they serve. For example, IBM HTTP Server can protect its own resources in the following ways:

- ▶ HTTP basic authentication

With IBM HTTP Server, different user registry modules are provided to be used for authentication, including simple text file, LDAP directory, or database.

In 7.5.1, “Configuring LDAP authentication with IBM HTTP Server” on page 147, we explain how to set IBM HTTP Server for using Basic authentication with the LDAP directory as the user registry. In “Configuring basic authentication for the Web server” on page 113, we explain how to set IBM HTTP Server to use Basic authentication with the text file user registry.

For more details about HTTP Basic Authentication, see the protocol definition document at the following address:

<http://www.ietf.org/rfc/rfc2617.txt>

- ▶ Digital client certificate authentication using SSL

Configuring basic authentication for the Web server

This section presents a simple scenario of how to implement basic authentication for the Web server when user registry is stored in a simple text file.

LDAP or client certificate authentication: Basic authentication is not a secure method of authorizing user access to Web server resources. If Web server level authentication is required, use LDAP or client certificate authentication.

For this example scenario, we enable security for all the static Web components in the C:\IBM\HTTPServer\htdocs\en_us directory.

Creating the user registry text file

First, create a simple text file that stores the user registry information. For this purpose, use the **htpasswd** utility that comes with IBM HTTP Server. Example 7-4 shows how to run this utility. In this case, we create a new user registry file named `users` and add the `ITS0user` user.

Example 7-4 Creating user registry text file

```
C:\IBM\HTTPServer\conf>..\bin\htpasswd -c users ITS0user
Automatically using MD5 format.
New password: *****
Re-type new password: *****
Adding password for user ITS0user

C:\IBM\HTTPServer\conf>
```

Configuring the Web server to use basic authentication

To configure the Web server to use basic authentication:

1. Open the `httpd.conf` file, which is the configuration file for IBM HTTP Server.
2. Make sure that the `auth_module` module definition in the `LoadModule` list is uncommented, as shown in Example 7-5.

Example 7-5 auth_module definition

```
LoadModule auth_module modules/mod_auth.so
#LoadModule auth_anon_module modules/mod_auth_anon.so
#LoadModule auth_dbm_module modules/mod_auth_dbm.so
```

3. Enable SSL on the global level and add Basic authentication. Previously you created a virtual host definition and defined SSL transport just for that scope.

Add the `Directory` directive to protect the C:\IBM\HTTPServer\htdocs\en_us directory. This directory is set as a global Web server root so that when you call the Web server using host name, the Web server searches for the `index.html` file under the Web server root.

Within Directory, specify additional security directives that are effective only on that scope (Example 7-6).

Example 7-6 Configuring HTTP basic authentication with text file user registry

```
#Listen 0.0.0.0:80
Listen 0.0.0.0:443
SSLEnable
KeyFile "C:/IBM/HTTPServer/conf/keys/IHS6Certificates.kdb"
<Directory "C:/IBM/HTTPServer/htdocs/en_US">
AuthType Basic
AuthName "Restricted Directory"
AuthUserFile "C:/IBM/HTTPServer/conf/users"
Require valid-user
Options None
AllowOverride None
</Directory>
```

4. Save the httpd.conf file and restart the Web server.

Testing the basic authentication configuration with the Web server

To test basic authentication:

1. Open a new browser window on the Web server machine.
2. In the address bar, type the following URL:

```
https://localhost
```

If you are using a browser on a separate machine, provide the proper server name in the URL.

3. After the SSL connection is established and the authentication window (Figure 7-1 on page 116) opens, for User Name, type ITS0user and the corresponding password. Click **OK**.

The content (index.html) is served.

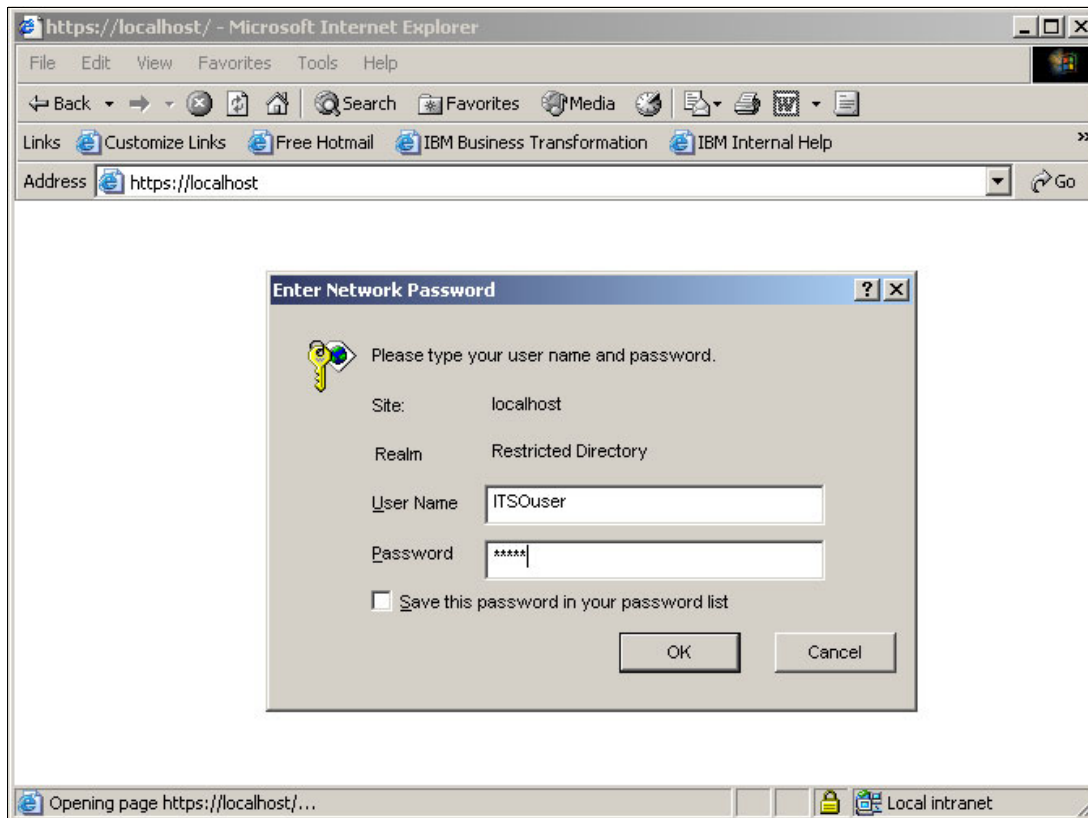


Figure 7-1 Testing HTTP Basic authentication with enabled SSL

7.2.3 Authorization by using aWeb server

By default, the Web server configuration and access control directives are handled by the Web server administrator by modifying the `httpd.conf` file. The appropriate section of the file enforces these settings as shown in Example 7-7.

Example 7-7 Enforcing access control management by settings in `httpd.conf` file

```
<Directory "C:/IBM/HTTPServer/htdocs/en_US">
    AllowOverride None
    Options None
</Directory>
```

The directive `AllowOverride None` tells the Web server not to look for any other access control definition files within the given directory scope. In a default `httpd.conf` configuration file shipped with IBM HTTP Server, this directive is included in every `<Directory>` container.

However, in many cases, this is a limiting factor and an administrator might need to intervene for simple changes to the file. You might want to give to an individual user or group of people the possibility to configure their own area of the Web site. This is not possible with the default `httpd.conf` settings.

If there is a requirement to set an access control on a per-directory basis, overriding the settings in the `httpd.conf` file, IBM HTTP Server uses `.htaccess` files for every directory over which the user wants such control. Changes done to any `.htaccess` file do not require restarting the Web server or any other administrator intervention because the file is read every time a resource is fetched from the directory.

A `.htaccess` file placed in one directory applies to all its subdirectories. If there is more than one access file in a directory tree, the directives set in a file for the subdirectory take precedence over the directives in the parent directory.

The drawback of using `.htaccess` files is the impact on the Web server's performance. Another problem with using `.htaccess` files is in regard to system management. System management is difficult to maintain, especially in a centralized security infrastructure.

For more information about how to use the `.htaccess` file, see the Apache tutorial at the following address:

<http://apache-server.com/tutorials/Atusing-htaccess.html>

7.3 Securing the Web server plug-in for WebSphere

This section focuses on securing the WebSphere Application Server HTTP plug-in. Although the plug-in runs as a part of the Web server's process, detached from WebSphere, it is an integral part of the WebSphere Application Server and security.

7.3.1 Securing the transport channel between the Web server and WebSphere

This section explains the configuration to instantiate a secure connection between the Web server plug-in and the embedded HTTP server in the WebSphere Web container. By default, this connection is not secure, even when global security is enabled. The documentation covers the configuration for IBM HTTP Server, but the Web server related configuration in this situation is not specific to any Web server.

Setting the authentication mechanism as a client certificate

You must complete the following steps to generate the certificates for SSL communication between the two peers:

1. Create a self-signed certificate for the Web server HTTP plug-in.
2. Create a self-signed certificate for the WebSphere Web Container.
3. Exchange the public keys between the two peers.
4. Create a new key store configuration.
5. Create a new SSL configuration (or modify an existing one).
6. Modify the WebSphere embedded HTTP Server (Web Container) to use SSL/HTTPS.
7. Regenerate and propagate the Web server `plug-in-cfg.xml` file for the Web Container SSL/HTTPS change to take effect. Figure 7-2 illustrates the exchange of the public certificate keys that are associated with each peer participating in the secure SSL communication.

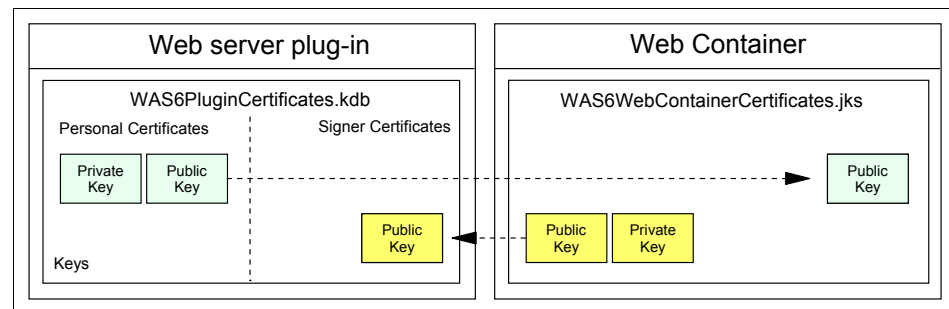


Figure 7-2 Certificates

Creating a self-signed certificate for the Web server HTTP plug-in

Create a CMS type key store, for example, `WAS6PluginCertificates.kdb` in the `C:\IBM\HTTPServer\conf\keys\` directory. Create a self-signed certificate.

More information: For more details about creating and extracting a self-signed certificate, see *WebSphere Security Fundamentals*, REDP-3944.

Creating a self-signed certificate for the WebSphere Web container

Create a Java Key Store (JKS) type key store, for example, `WAS6WebContainerCertificates.jks` in the `C:\WebSphere\Appserver\profiles\<profilename>\cells\<cellname>` directory. Create a self-signed certificate.

Exchanging the public keys between the two peers

Exchange the public certificates from the self-signed certificated between the two key stores:

- ▶ `WAS6PluginCertificates.kdb`
- ▶ `WAS6WebContainerCertificates.jks`

For information: For more details about exchanging certificates, see *WebSphere Security Fundamentals*, REDP-3944.

Creating a new SSL configuration

Within the WebSphere configuration, an SSL configuration represents a set of SSL key store entries and properties that you can use with different WebSphere resources.

To create a new cell-scoped key store entry and SSL configuration:

1. Start the WebSphere Administration Console and log in.
2. Select **Security** → **SSL certificate and key management** → **Manage endpoint security configurations** → **Inbound** → **<cellname>**.
3. In the Related Items section, click **Key stores and certificates**.
4. Click **New** to create a key store entry.
5. Enter the following values to complete the form:
 - a. For Name, type `WebContainerKeyStore`.
 - b. For Path, type `${CONFIG_ROOT}\cells\<cellname>\WAS6WebContainerCertificates.jks`.
 - c. For Password, type `passw0rd`.

- d. For Type, enter JKS.
- e. Leave other options as the default settings.
- f. Click **OK**.

Note: If you do not want WebSphere to make changes to the key store or the certificates it contains, select the **Read-only** check box.

6. Create an SSL configuration to contain the new key store. On the SSL certificate and key management page, select **SSL configurations**.
7. Click **New** to create a new SSL configuration.
8. Enter the following information to complete the form.
 - a. For Name, type WebContainerSSL.
 - b. For Trust store name, select **WebContainerKeyStore**.
 - c. For Key store name, select **WebContainerKeyStore**.
 - d. Before you can select an entry for the default server and client certificate aliases, click **Get certificate aliases**. WebSphere reads the key store configuration and populates the list with all the available server and client certificates that exist in the key store.
 - e. Select the desired server and client certificate aliases
 - f. Click **Apply**.
 - g. Optional: under Additional Properties, click **Quality of protection (QoP)** settings to set additional SSL settings, such as client authentication, cipher suites, providers, and protocols.

The client authentication option determines whether the resource that is using this SSL configuration also expects to get a client certificate either trusted by a well-known CA, or self-signed with the imported public key. If you enable this option for an SSL configuration that is used for HTTP plug-in to Web container transport protection, then a client certificate is required in the HTTP plug-in key database and the Web container must be able to recognize that certificate.

We used the default additional settings. If you require client authentication within this scope, see 7.5.2, “Configuring SSL certificate-based client authentication for the IBM HTTP Server” on page 152, to learn how to set a client certificate onto a CMS key database required for the HTTP plug-in. See also 7.5.3, “Configuring SSL certificate-based client authentication for WebSphere Application Server” on page 156, to set the SSL entry and Web container configuration for client authentication.

- h. Click **OK**.

Mutual SSL: If you want mutual SSL between the two parties, select either **Supported** or **Required** for Client authentication on the Quality of protection settings page.

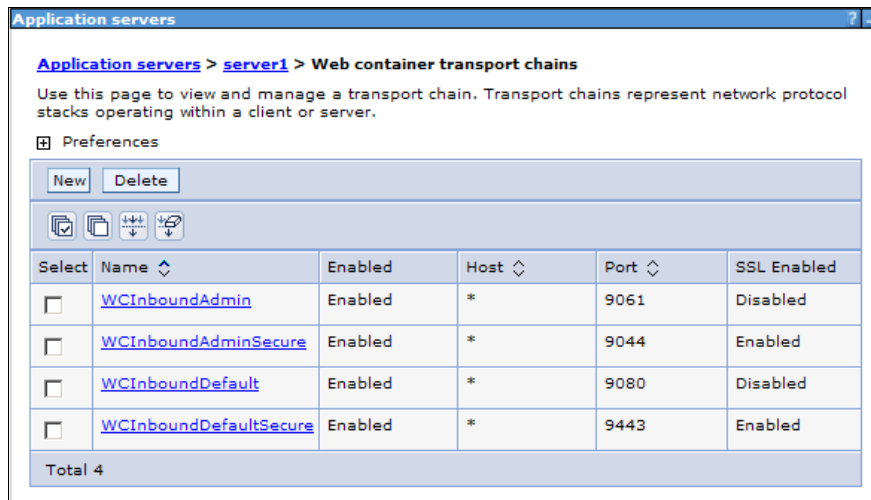
9. Save the configuration for WebSphere.

Modifying the Web Container configuration to support SSL

To complete the configuration between Web server HTTP plug-in and Web Container, the WebSphere Web Container must be modified to use the previously created self-signed certificates.

To modify the Web Container with the required changes:

1. Select **Servers** → **Application servers**. Click the server that you want to work with, which is **server1** in our case.
2. Under the Container Settings section, click **Web Container Settings** → **Web container transport chains**.
3. Modify the default secured transport chain called *WCInboundDefaultSecure* chain to use it in this section. Web Container listens on the TCP port 9443 for this chain. See Figure 7-3.



Select	Name	Enabled	Host	Port	SSL Enabled
<input type="checkbox"/>	WCInboundAdmin	Enabled	*	9061	Disabled
<input type="checkbox"/>	WCInboundAdminSecure	Enabled	*	9044	Enabled
<input type="checkbox"/>	WCInboundDefault	Enabled	*	9080	Disabled
<input type="checkbox"/>	WCInboundDefaultSecure	Enabled	*	9443	Enabled

Total 4

Figure 7-3 Default transport chains for server1

You can also create another transport chain and configure it as follows:

- a. Click **WCInboundDefaultSecure** (or on the new transport channel if you created one). Ensure that the **Enabled** check box is selected.

- b. Click **SSL Inbound Channel (SSL 2)**. Then, in the SSL configuration section, select **Specific to this endpoint**, and select **WebContainerSSL** from the list. Thus, you specify a previously created SSL configuration to be used with this transport channel (see Figure 7-4).
- c. Click **OK**, and then save the configuration for WebSphere.

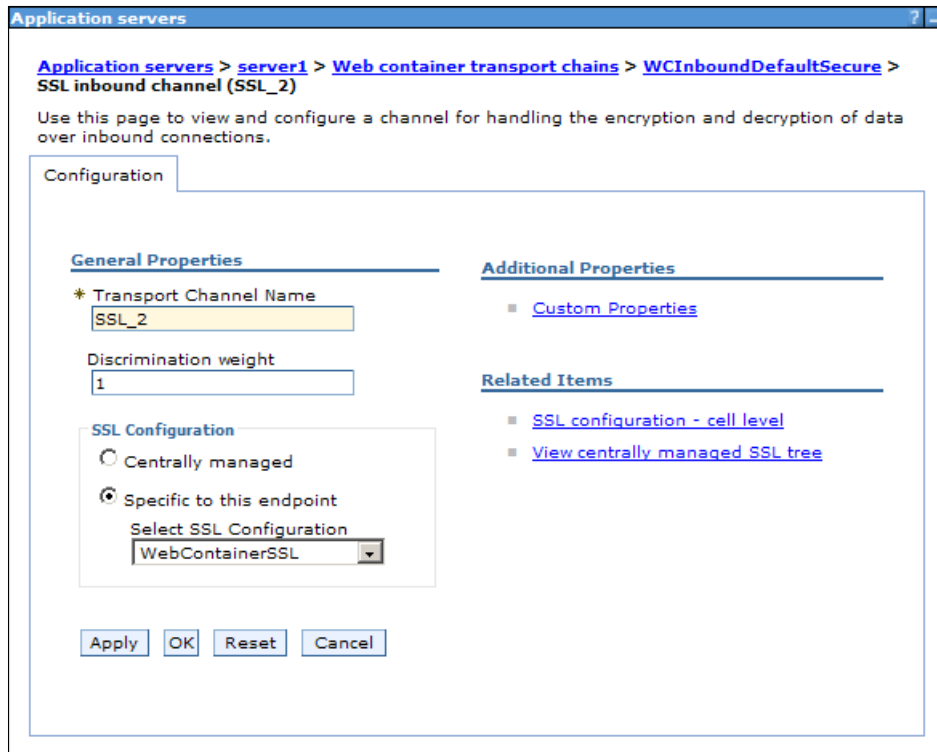


Figure 7-4 SSL Inbound Channel properties

4. Regenerate and propagate the Web server plug-in.
5. Restart the application server for the changes to the Web Container configuration to take effect.

Modifying the Web server plug-in file

You must modify the plug-in configuration file to reference the plug-in key ring and the password stash file. By doing this modification, you can change the transport protocol from HTTP to HTTPS with the certificates stored in the key ring.

With WebSphere Application Server V6.1, more than one Web server definition is possible, and each has its own set of HTTP plug-in properties, which are

configurable in the Administrative Console. Also by default, there is a separate installation directory of HTTP plug-in binaries. Within this directory, there are separate plug-in configuration directories for each Web server/plug-in definition. This section shows the use of the Web server definition, `webserver1`. In our case, its configuration is in `C:\WebSphere\Plugins\config\webserver1`.

Tip: If you are unsure about which HTTP plug-in configuration file is the right one, open the Web server's configuration file and check which file name is used when defining HTTP plug-in module configuration.

To modify the Web server plug-in file:

1. Copy your key database and stash files to the Web server repository directory for `webserver1`:

```
<config_root>\cells\<cellname>\nodes\<nodename>\servers\webserver1
```

Where `<config_root>` is the configuration directory of your WebSphere profile:

```
C:\WebSphere\AppServer\profiles\<profilename>\config
```

2. In the Administrative Console, select **Servers** → **Web servers**. Click the server you want to work with, which in this case is **webserver1**.
3. In the Additional Properties section, click **Plug-in properties**.
4. Change the following values on the Plug-in properties page
 - a. In *Repository copy of Web server plug-in files*, for Plug-in key store file name, type `WAS6PluginCertificates.kdb`.
 - b. In *Web server copy of Web server plug-in files*, for Plug-in key store directory and file name, type `C:\WebSphere\Plugins\config\webserver1\WAS6PluginCertificates.kdb`.
 - c. In *Plug-in logging*, for Log level, type `Trace`.
5. Click **Copy to Web server key store directory**.
6. Click **OK**.
7. Save the WebSphere configuration.
8. Open the `plugin-cfg.xml` HTTP plug-in configuration file for editing. A standard non-secure HTTP connection in the configuration looks similar to the following example:

```
<Transport Hostname="kcgl6kh.itso.ral.ibm.com" Port="9080"  
Protocol="http"/>
```

By default, there is also the secured connection entry (Example 7-8).

Example 7-8 Secured connection entry

```
<Transport Hostname="kcg16kh.itso.ra1.ibm.com" Port="9443"
Protocol="https">
  <Property name="keyring"
value="C:\WebSphere\Plugins\config\webserver1\WAS6PluginCertificates
.kdb"/>
  <Property name="stashfile"
value="C:\WebSphere\Plugins\config\webserver1\WAS6PluginCertificates
.sth"/>
</Transport>
```

Note that the `kcg16kh.itso.ra1.ibm.com` is the host name for the application server.

a. Comment out or delete the non-secure HTTP part as follows:

```
<!-- <Transport Hostname=kcg16kh.itso.ra1.ibm.com" Port="9080"
Protocol="http"/> -->
```

b. Save the `plugin-cfg.xml` file.

9. Restart the Web server.

Note: If the non-secure transport is not a requirement for your application, remove the `WCInboundDefault` transport chain from the application server. Then, after you regenerate and propagate the Web server plug-in, the non-secure HTTP transport channel can no longer be used.

In this scenario, you prevent the Web server from connecting to the Web Container on the non-secure transport channel. In later examples, this allows you to access the Web Container directly on the non-secure transport channel.

7.3.2 Testing the secure connection

In this section, we assume that your Web server is SSL-enabled and that the connection between the browser and Web server is secured HTTPS. Although we do not recommend this security practice, the behavior must not be different if you use a Web server that is not SSL-enabled.

After setting the HTTP plug-in and restarting the Web server, test the secure connection:

1. Open a new browser window. Make sure that the Web server, WebSphere (server1), and the default application are running.

2. Enter the following URL to access the Snoop servlet:

```
https://webserver01.redbook.net/snoop
```

Here webserver01.redbook.net is the host name for the Web server.

3. When application security is enabled in WebSphere, in the Basic authentication window, enter a valid user name and password. The Snoop servlet is output. For further information about enabling application security see 3.1, “Enabling administrative security” on page 50.

Because we commented out the HTTP transport in the plug-in configuration, leaving only the HTTPS option available, we left no choice to the plug-in. Therefore, it must use HTTPS transport to connect to WebSphere.

We called Snoop by using the Web server with HTTPS. The Web server passed the request to the HTTP plug-in, and the plug-in contacted WebSphere by using HTTPS.

Open the `http_plugin.log` plug-in trace file (Example 7-9). This file shows that the SSL connection has been used to connect to WebSphere and that WebSphere issued the basic authentication challenge before serving the Snoop servlet response.

Example 7-9 HTTP plug-in trace file showing the SSL connection

```
ws_common: websphereFindTransport: Finding the transport
ws_common: websphereFindTransport: Setting the transport(case 4):
kcgl6kh.itso.ral.ibm.com on port 9443
ws_common: websphereExecute: Executing the transaction with the app
server
ws_common: websphereGetStream: Getting the stream to the app server
ws_transport: transportStreamDequeue: Checking for existing stream from
the queue
ws_common: websphereGetStream: socket 7792 connected to
kcgl6kh.itso.ral.ibm.com:9443
lib_stream: openStream: Opening the stream
lib_stream: openStream: Stream is SSL
ws_common: websphereGetStream: Created a new stream; queue was empty,
socket = 7792
lib_htrequest: htrequestWrite: Writing the request:
  GET /snoop HTTP/1.1
  Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

```
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET
CLR 1.1.4322)
Host: webserver01.redbook.net
Connection: Keep-Alive
$WSIS: false
$WSSC: http
$WSPR: HTTP/1.1
$WSRA: 127.0.0.1
$WSRH: 127.0.0.1
$WSSN: localhost
$WSSP: 80
Surrogate-Capability: WS-ESI="ESI/1.0+"
lib_htrequest: htrequestWrite: Writing the request content
ws_common: websphereExecute: Wrote the request; reading the response
lib_htresponse: htresponseRead: Reading the response: c2976c
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Default Realm"
Content-Language: en-US
Content-Length: 0
lib_htresponse: htresponseSetContentLength: Setting the content length
|0|
```

7.4 Securing the Web container of the application server

This section discusses the authentication and authorization matters related to the application server Web container.

7.4.1 Securing the transport channel

In 7.3.1, “Securing the transport channel between the Web server and WebSphere” on page 118, you secured the transport channel for the Web container. For internal communication between Web and Enterprise JavaBeans (EJB) containers, WebSphere uses Internet Inter-ORB Protocol (IIOP), and the internal communication is protected, by default, with administrative security enabled.

For more information about Remote Method Invocation (RMI) over IIOP security settings, see 8.4.3, “RMI/IIOP transport channel protection” on page 204.

7.4.2 Authentication by using the Web container

The authentication method defines how the user is authenticated by the Web application. Before any authorization constraint is applied, the user must pass the authentication process by using a configured mechanism.

For Web container authentication, WebSphere provides full compliance of the Java 2 Platform, Enterprise Edition (J2EE) specification, which defines the following types of authentication methods:

- ▶ Basic authentication
- ▶ Form-based authentication
- ▶ Client certificate authentication

Note: Form-based or client certificate are the preferred methods of authentication. With the use of basic authentication, the client browser caches the user ID and password in memory, which are sent to the server whenever an authentication request is received. The cached authentication credentials never time out. Therefore, the user's session cannot be invalidated unless the browser is closed.

For more details about authentication mechanism, see *WebSphere Security Fundamentals*, REDP-3944.

If a security constraint is set but no authentication method for a Web module is configured, the default is to use basic authentication. For any type of authentication methods to work, at least one security constraint must be defined for the requested Web resources, and application security must be enabled for the application server.

For instructions on how to define security constraints for Web resources, see “J2EE Web module security fundamentals” on page 132. For instructions on how to enable application security, see 6.1, “Application security” on page 102.

This section shows basic scenarios for setting up authentication for the ITSObank application.

Configuring form-based authentication

One of the login challenges defined in the J2EE specification is the *form-based login*. It enables the application developer to customize the login process and presents an application-specific form by using the Form Login Authentication mechanism.

Form login works in the following manner:

1. An unauthenticated user requests a resource protected by the Form Login authentication type.
2. The application server redirects the request to the login form defined previously in the Web deployment descriptor.
3. On the HTML login form, the user enters the user ID and password and submits the form.
4. The action, triggered by the form submission, refers to a special servlet *j_security_check*. The Web container, after receiving a request for the *j_security_check* servlet, dispatches the information to the application server's security mechanism to perform the authentication.
5. If the servlet authenticates the user successfully, the originally requested resource is displayed.

Because Lightweight Third Party Authentication (LTPA) is the default authentication mechanism for administrative and application security, to use form login in any Web application, single sign-on (SSO) must be enabled. If SSO is not enabled, authentication during form login fails with a configuration error.

SSO is required because it generates an HTTP cookie that contains information representing the identity of the user to the Web browser. This information is required when using form login to authorize protected resources. In WebSphere Application Server V6.1, SSO is enabled by default. However, additional configuration might be necessary.

To configure SSO:

1. Log in to the WebSphere Administration console and select **Security** → **Secure administration, applications, and infrastructure**.
2. In the Authentication section, click **Web security** → **Single Sign-On**.

Figure 7-5 shows the default configuration for SSO. For the examples that follow, we use the default settings.

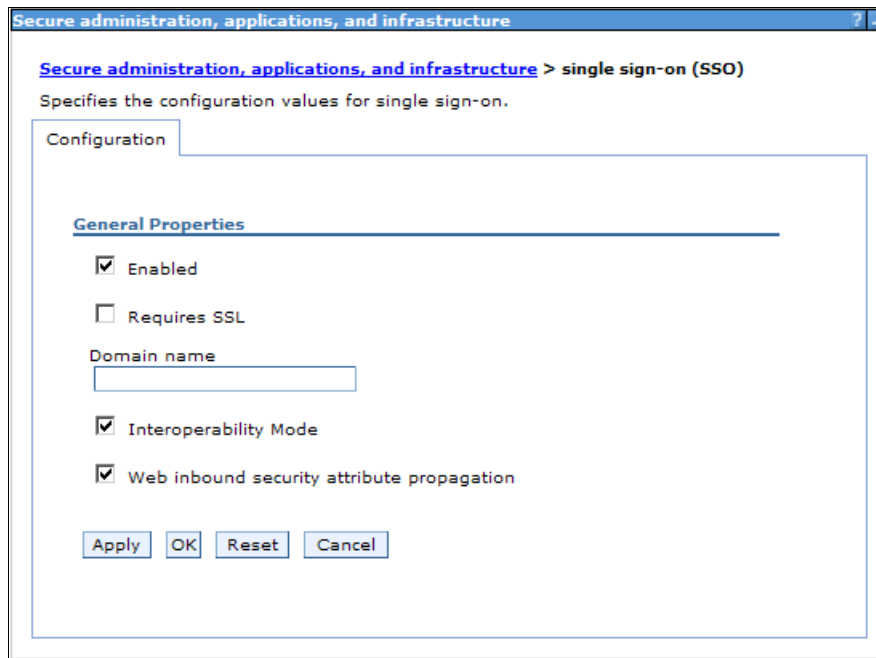


Figure 7-5 Configuring single sign-on

Configuring a form-based login

To configure a form-based login, as shown in Figure 7-6, by using the Rational Application Developer:

1. Load your Web application module into Rational Application Developer. In our example, this module is `itsobank.ear`.
2. Within J2EE perspective, expand **Dynamic Web Projects** → **itsobank**.
3. Double-click the **Deployment Descriptor** of itsobankWeb Module.

4. On the Web Deployment Descriptor page (Figure 7-6), click the **Pages** tab and scroll down.
 - a. In the Login section, select the **FORM** authentication method.
 - b. For Login Page, type /login/login.html.
 - c. For the Error Page, type /login/loginerror.html.

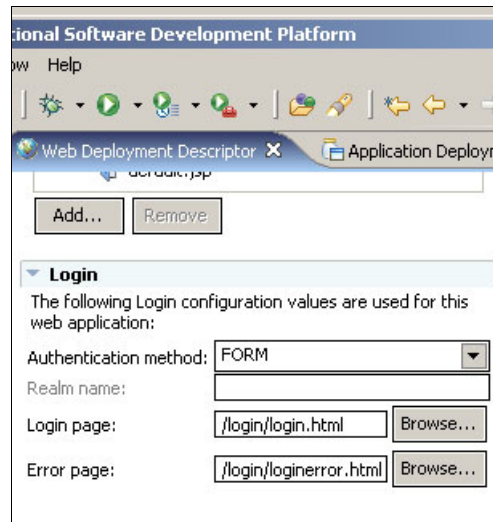


Figure 7-6 Form login configuration

5. Save the changes.

Setting the Authentication Method for the application Web module creates a <login-config> section in a Web deployment descriptor XML file, as shown in Example 7-10.

Example 7-10 Login-config section of the Web deployment descriptor

```

<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>ITSO Bank</realm-name>
  <form-login-config>
    <form-login-page>/login/login.html</form-login-page>
    <form-error-page>/login/loginerror.html</form-error-page>
  </form-login-config>
</login-config>

```

Simple form-based login does not require any extra code development on the server side. The `j_security_check` servlet that WebSphere Application Server

uses enforces only the name of the following input fields that the developer must put in the custom Login Form:

- ▶ `j_username` must be the input field in which a user types the user name.
- ▶ `j_password` must be the input field in which a user types the password.

The action required for the HTTP POST method is `j_security_check`. Example 7-11 shows simple HTML code for the custom login form.

Example 7-11 Sample custom login form from the ITSOBank application

```
<!-- ..... -->
<form method="post" action="/itsobank/j_security_check">
User name:<input type="text" name="j_username">
Password:<input type="password" name="j_password">
<input type="submit" name="action" value="Login">
</form>
<!-- ..... -->
```

Attention: The `j_security_check` servlet does not work when global security is disabled. The application server returns a Page Not Found error. This is also true for the `ibm_security_logout` servlet.

Form-based logout

One of the IBM extensions to the J2EE specification is the form-based logout as shown in Example 7-12. After logging out, the user is required to re-authenticate to have access to protected resources again. This logout form can be on any page that is calling a POST action on the `ibm_security_logout` servlet. This form must exist within the same Web application to which the user is redirected after logging out.

Example 7-12 Sample logout form from the ITSOBank application

```
<form method="post" action="ibm_security_logout" name="logout">
<input type="submit" name="logout" value="Logout">
<input type="hidden" name="logoutExitPage" value="/login/login.html">
</form>
```

Today's e-business Web applications require strict and well-designed security. Providing the logout function is one of the important functions. Obviously, closing the browser and destroying the session is always an option for the user, but it is not an appropriate solution to finish a session with an application.

By combining the logout function with programmatic security, you can implement step-up re-authentication, where the user can change credentials and get higher authority in the application.

Note: The logout only works together with form-based login. When the application is configured to use Basic Authentication, the credentials are stored in the client's browser, and the browser sends the user name and password to the server together with every request. The only way to log out is to break the session by closing the browser.

7.4.3 Authorization by using the Web container

For Web container authorization, WebSphere provides full compliance on the J2EE specification. This section discusses *declarative security*, which means that the application is not security aware. It does not contain any security related code. The protection on J2EE level is configured through deployment descriptors and enforced by WebSphere.

We describe a situation where the application is security-aware as *programmatic security*. See 7.4.4, "Programmatic security" on page 141, for more information.

J2EE Web module security fundamentals

In a J2EE application architecture, the Web module of the enterprise application is comprised of one or more related servlets, JSP files, XML files, and HTML files that can be managed as one integrated unit. The files in the Web module are related in the sense that they perform a common business logic function.

The Web modules of the enterprise application run within the Web container of the application server. The Web container, as a runtime environment for the Web application, is responsible for handling requests for servlets, JSP files, and other Web components running on the server-side or served from the server-side. The Web container creates servlet instances, loads and unloads servlets, creates and manages requests and response objects, and performs other servlet management tasks.

The following sections explain how to configure security for the Web module of an enterprise application.

Security roles

WebSphere implements roles-based security from the J2EE specification. The security role is a logical grouping of principals. Access to a specific part of the application is granted based on the role, which is then mapped during the development or deployment phase to specific user registry entries. It gives a certain level of transparency to the application development process.

The developer does not have to bother with the different user privileges that can be defined for the application.

To define a role for the Web module with a Rational Application Developer:

1. Within J2EE perspective, expand **Dynamic Web Projects** → **itsobank**. Double-click the **Deployment Descriptor** of the itsobankWeb Module.
2. On the Web Deployment Descriptor page, select the **Security** tab.
 - a. In the Security Roles section, click **Add** to add a new security role.
 - b. In the Add Security Role window (Figure 7-7), for the Name field, type **User** and click **Finish**.
3. Repeat steps a and b, and this time create a **Manager** security role.
4. Save and close the file.

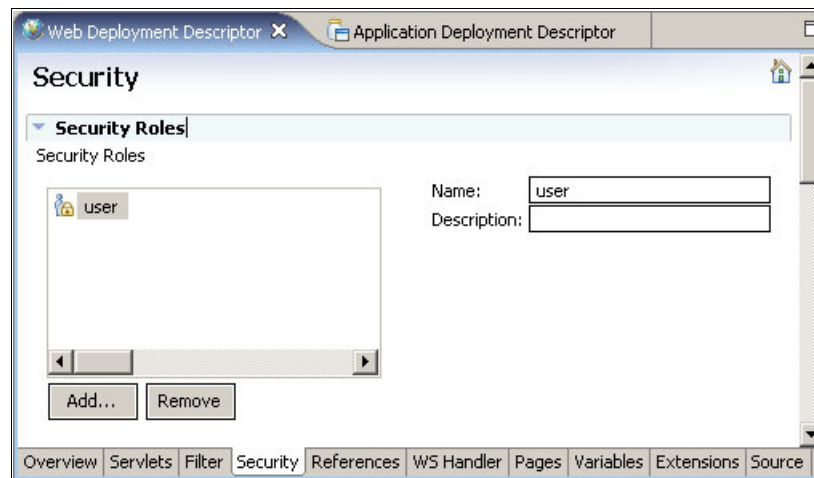


Figure 7-7 Creating a new security role for the Web module

Security constraints

Providing an authentication mechanism for global application security and creating security roles as you did in the previous two sections does not provide the mechanisms to control access to the Web resources.

Security constraints declare how the content of the application is protected. For a given security constraint, you must define the following items:

- ▶ One or more Web resources that define actual application components that are to be protected by the security constraint

A Web resource is a set of URL patterns and HTTP methods in those resources. All requests that are matched with the pattern defined for a given Web resource are subject to a security constraint.

- ▶ An authorization constraint that defines the roles that provide access to the Web resources existing within the security constraint

An *authorization constraint* is a set of roles that the user must be granted for access to a Web resource collection existing within a security constraint. To have access to the Web resource, the user must be granted at least one of the roles that are defined within the authorization constraint.

- ▶ The user data constraint, which indicates the transport layer setting for client/server communication to satisfy given security constraints

This setting must guarantee either content integrity (preventing tampering in transit) or confidentiality (preventing reading data during transfer). The user data constraint can override standard security settings for the application. For example, access to some functions of the application might require basic login by using a user ID and password. At the same time, some functions might require a higher level of protection. The user data constraint allows an application deployer to introduce such protection.

If WebSphere application security is enabled, and a security constraint is set for a particular resource, then the resource is secured.

J2EE security role reference

During the development phase of the application, the actual role names for security constraints might not be known to the groups of developers. The actual role names in a deployed runtime environment might not be known until the Web application and EJB modules are ready and assembled into the .ear file. Therefore, the role names used during development are considered to be *logical roles*. These logical roles are then mapped by the application deployer into the actual runtime roles during the application assembly and deployment phase.

Security role references provide a level of indirection to isolate roles that are used during development and actual runtime roles. They link the names of the roles that are used in the module to the corresponding name of the role in the encompassing application.

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.

To define security role references for the Web module in Rational Application Developer:

1. Load your Web application module into Rational Application Developer. In our example, this is `itsobank.ear`.
2. Within J2EE perspective, expand **Dynamic Web Projects** → **itsobank**.

3. Open the Web Deployment Descriptor for the **itsobankWeb** Module.
 - a. Select **Servlets** tab.
 - b. In the Servlets and JSPs section, select **TransferServlet**.
 - c. In the Add Security Role References window (Figure 7-8):
 - i. For Role link, select **User**.
 - ii. For Role reference name, enter The User role. The user role was created when you defined security constraints.
 - iii. Click **Finish**.



Figure 7-8 Setting security role reference for TransferServlet

4. Save and close the file.

Securing content served by WebSphere Application Server

On the J2EE security level, WebSphere Application Server can secure Web resources by using role-based declarative security mechanisms. This means that the logical application security structure is defined independently from the application itself. The logical security structure is stored in the deployment descriptors of the application.

You can divide Web resources into two categories:

▶ Static resources

Static resources refers to all the resources that do not change the response output over time. Static resources are, for example, static HTML pages and different image files.

You can secure static resources of the enterprise application only if WebSphere serves them. WebSphere cannot manage access to the static content that resides on the Web server. All the static content that must be protected by WebSphere Application Server must be packaged into the Web module (Web archive file (.war)). Static HTML pages can be served by the servlet that implements file serving behavior.

File serving: Consider disabling *File serving* on your application if you do not serve static content from your application.

▶ Dynamic resources

Dynamic resources refers to all the resources that change the response output over time, depending on different input parameters. Dynamic resources are, for example, servlets and JSPs.

Serve servlets by classname: Most applications have the option *Serve servlets by classname* enabled by default. Because this poses a security risk, you must disable it when preparing an application EAR file for deployment.

For all the Web resources, with WebSphere Application Server, you can protect them on the HTTP method level. For example, the POST method of a servlet can be part of a different security constraint than the GET method. The full list of predefined methods that can be secured is as follows:

- ▶ GET
- ▶ POST
- ▶ PUT
- ▶ DELETE
- ▶ HEAD
- ▶ OPTION
- ▶ TRACE

When using method-level security constraints for resources, you might want to separate the content that all the users can view from the administrative functions that only privileged users are allowed to access. In WebSphere Application

Server, you can do this by using different security constraints for the different methods.

Configuring security constraints

To set up constraint to protect content for the Web application module:

1. Load your Web application module into Rational Application Developer. In our example, this is `itsobank.ear`.
2. In the J2EE perspective, expand **Dynamic Web Projects** → **itsobank**.
3. Open the Web Deployment Descriptor for the **itsobankWeb** Module.
4. Click the **Security** tab:
 - a. In the Security Constraints section, click **Add** to add a new security constraint.
 - b. In the Add Security Constraints window, for the Name field, type `ITS0 Bank security constraint` and click **Next**.
 - c. In the Add Web Resource panel (Figure 7-9 on page 138):
 - i. Specify the resource name, HTTP methods that are allowed to be issued on the resource, and URL patterns on which this security constraint applies.

Important: Make sure that the URL patterns correctly match your resource names or URL bindings. There is no value syntax checking. Therefore if the URL pattern is entered incorrectly, you end up with an unprotected resource and a security constraint for a non-existent resource.

- ii. Complete the remaining values as shown in Figure 7-9.
 - iii. Click **Finish**.

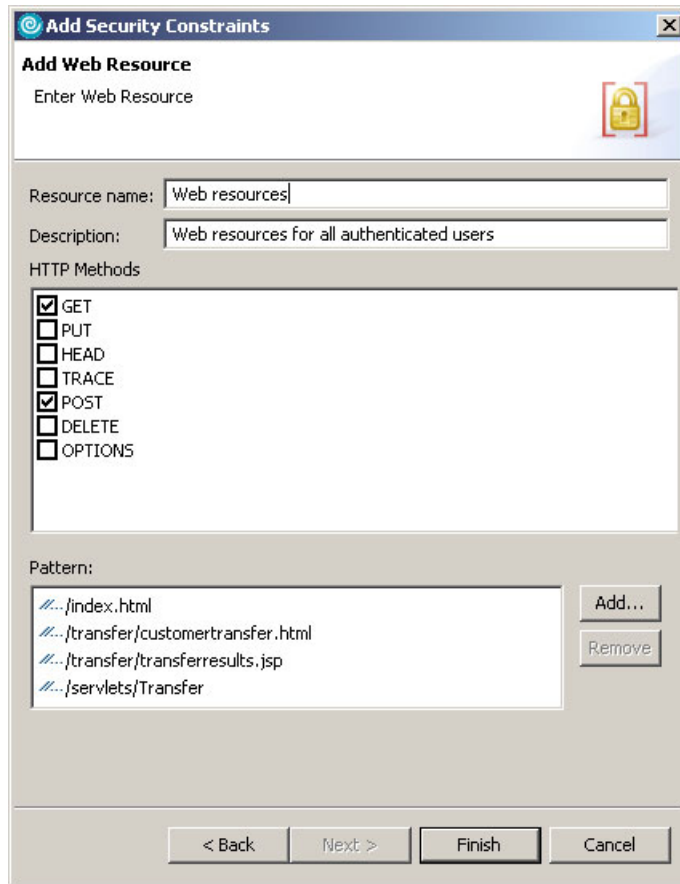


Figure 7-9 Configuring security constraints for Web resources

Configuring authorization constraints

The security constraint that you have created for static resources defines the HTTP methods that are allowed on certain URL definitions that represent the Web resources. More specifically, within your security constraint, you defined that only GET and POST methods can be run on your Web resources. However, you have not specified who is allowed to run the defined security constraint or specifically who is allowed to run GET and POST over your Web resources.

To define authorization constraint for the recently created security constraint:

1. On the Deployment Descriptor **Security** tab of the itsobankWeb Module, select **ITSO Bank security constraint**.
2. In the Authorized Roles section, click **Add**.

3. In the Define Authorization Constraint window:
 - a. For Description, type Web resources auth constraint.
 - b. In the Role Names list, ensure that **User** and **Manager** roles are selected.
 - c. Click **Finish**.
4. Save the changes. Figure 7-10 shows an overview of the security properties.

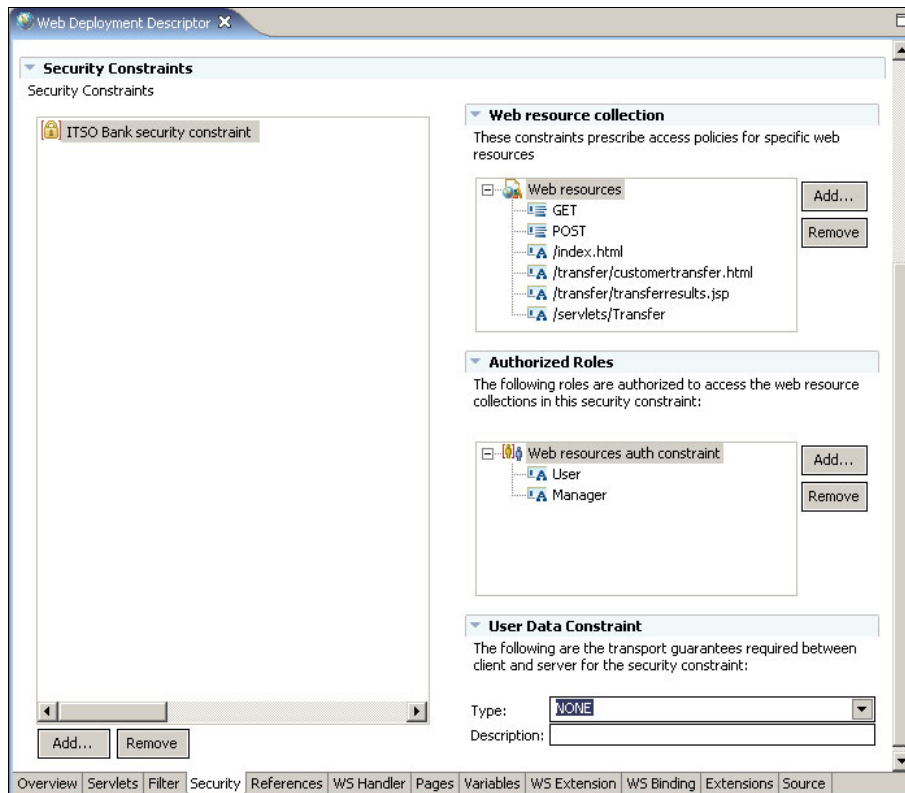


Figure 7-10 Overview of the configured security constraints for Web resources

User Data Constraint section: In the User Data Constraint section, you can choose a *transport guarantee*, which defines how to protect the communication between the client and server. Three options are available:

None	No constraint indicates that the application does not require any transport guarantee.
Integral	Data cannot be changed in transit. In practice, a request must be transmitted over an SSL encrypted channel.
Confidential	Data cannot be viewed in transit. In practice, the request must be transmitted over an SSL encrypted channel.

Adding security and authorization constraint for the AccountsView page

Up to this point, you have created a security constraint for the majority of your sample application Web resources and then given authorization on this constraint for previously created User and Manager security roles. The users that are mapped to these roles during application installation all have access to these resources.

However, your application contains a special static HTML page, called the *Accounts View* page, to which you do not want every user to have access. You only want that the Manager security role to have access to this page.

To add security and an authorization constraint for the AccountsView page:

1. Repeat steps 1 through 4 on page 137 to add another security constraint. Use the values shown in Figure 7-11 on page 141.

Add Security Constraints

Add Web Resource
Enter Web Resource

Resource name: Accounts View page

Description: Accounts View page security constraint

HTTP Methods

- GET
- PUT
- HEAD
- TRACE
- POST
- DELETE
- OPTIONS

Pattern: /*../transfer/accountsview.html

Add...
Remove

< Back Next > Finish Cancel

Figure 7-11 Creating a security constraint for the Accounts View page

2. Repeat steps 1 on page 138 through 3 on page 139 to create another authorization constraint for the just created Accounts View security constraint. This time, make sure that only the **Manager** security role is selected.

7.4.4 Programmatic security

Programmatic security means that the application is security aware and contains the code that handles security, providing any authorization and authentication capabilities that are beyond J2EE security. The opposite case when the protection is configured on the J2EE level and through application deployment descriptors is described as *declarative security*.

See 7.4.3, “Authorization by using the Web container” on page 132, for more details about security.

Programmatic security is divided into the following types:

- ▶ Java Authentication and Authorization Service (JAAS), where we use mechanisms available in JAAS API
- ▶ J2EE programmatic security, where we use a few extra Java methods available as part of Java Servlet 2.x specification

This section focuses only on the latter. For more details about JAAS security, see Chapter 5, “JAAS for authentication in WebSphere Application Server” on page 85.

J2EE servlet security methods

The Servlet 2.4 specification defines the following methods that allow programmatic access to the caller’s security information of `HttpServletRequest` interface:

Important: The `getRemoteUser()` and `getUserPrincipal()` methods return `null` as a result even if the user is logged in, unless the servlet or the JSP itself is secured.

- ▶ `String getRemoteUser()`

The `getRemoteUser` method returns the user name that the client has used to log in:

```
String user = request.getRemoteUser();
```

- ▶ `Boolean isUserInRole(String roleName)`

The `isUserInRole` method allows the developer to perform additional checks on the authorization rights of a user that are not possible, or are more difficult, to perform through the deployment descriptor of the servlet:

```
if (request.isUserInRole("Manager")) {  
    // the user is in the manager role  
    // ...  
}
```

- ▶ `java.security.Principal getUserPrincipal()`

The `getUserPrincipal` method allows the developer to get the name of the current caller. To do this, call `getName()` on the `java.security.Principal` object that is returned.

```
Principal principal=request.getUserPrincipal();  
String username=principal.getName();
```


Sample usage of security methods

Example 7-13 shows a modified code snippet from the ITSOBank sample application. Similar code is in the TransferServlet.java in the doPost() method. For more details, see the comments in Example 7-13.

Example 7-13 Sample code using the servlet security methods

```
// getting the environment variables for restricted role
// and for maximum transferable amount
restrictedRole=(String)environment.lookup("RestrictedRole");
maxWebTransferAmount=(Integer)environment.lookup("MaximumWebTransferAmount");
// checking if the user is restricted to a certain amount of transfer
if(request.isUserInRole(restrictedRole) &&
transferAmount>maxWebTransferAmount.intValue()) {
    // create an error message
    // the user cannot transfer the requested amount
    // forward the request to the response page with the message
}
// get the principal from the request
Principal principal=req.getUserPrincipal();
// print out the user information about the servlet invocation
System.out.println("Transfer Servlet was invoked by user:
"+req.getRemoteUser()+", principal: "+principal.getName());
```

With the security methods, the servlet does not let the user in a restricted role submit a transfer greater than the maximum amount that is allowed. Two environment variables are used in the code for this purpose:

- ▶ RestrictedRole, which defines the role that is restricted
- ▶ MaximumWebTransferAmount, which defines the upper limit of allowed transferable amount

Example 7-14 shows how these two environment variables are defined in the Web application deployment descriptor, which is the web.xml file.

Example 7-14 Environment variables for programmatic Web security sample code

```
<env-entry>
  <env-entry-name>MaximumWebTransferAmount</env-entry-name>
  <env-entry-type>java.lang.Integer</env-entry-type>
  <env-entry-value>5000</env-entry-value>
</env-entry>
<env-entry>
  <env-entry-name>RestrictedRole</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
```

```
<env-entry-value>User</env-entry-value>
</env-entry>
```

In this case, the RestrictedRole variable is mapped to the User security role, and that MaximumWebTransferAmount variable is set to 5000.

Testing the programmatic security sample

To test the programmatic security code in the ITSObank application, you must meet the following prerequisites:

- ▶ The ITSObank sample application must be installed into a WebSphere Application Server, and security must be enabled on that server.
- ▶ Two security roles must be defined for the application, User and Manager. Ensure that you map these two roles to different users (see Figure 7-12).

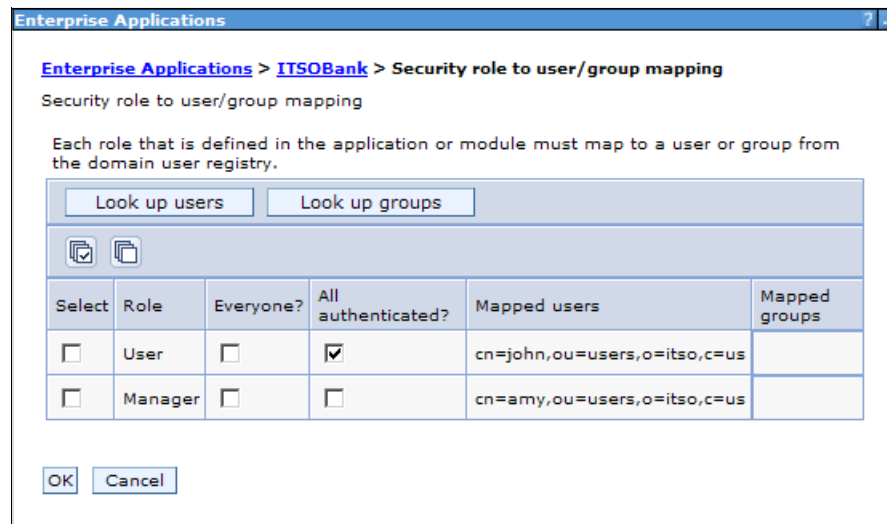


Figure 7-12 ITSObank security role to user/group mappings

To test the programmatic security code:

1. Make sure that the ITSObank application is started. Open a new browser window and enter the ITSObank index page URL as follows for our example:
`http://localhost:9080/itsobank/index.html`
Because this is a protected URL and you are not authenticated yet, you arrive at the login page.
2. Log in with a user mapped to the User security role. In our case, WebSphere used an LDAP for the user registry. During the application installation, we

mapped the user john to the User security role. Therefore, we enter the following information in the login page:

- Userid: john
- Password: test

Click the **Login** button.

3. On the Welcome page, click **Customer Transfer**.
4. In the Customer transfer page, for the Transferred amount field, type 10000. You can enter any number higher than 5000. Values for the other fields are not important. Click the **Transfer** button.

You receive a response similar to the page shown in Figure 7-13.

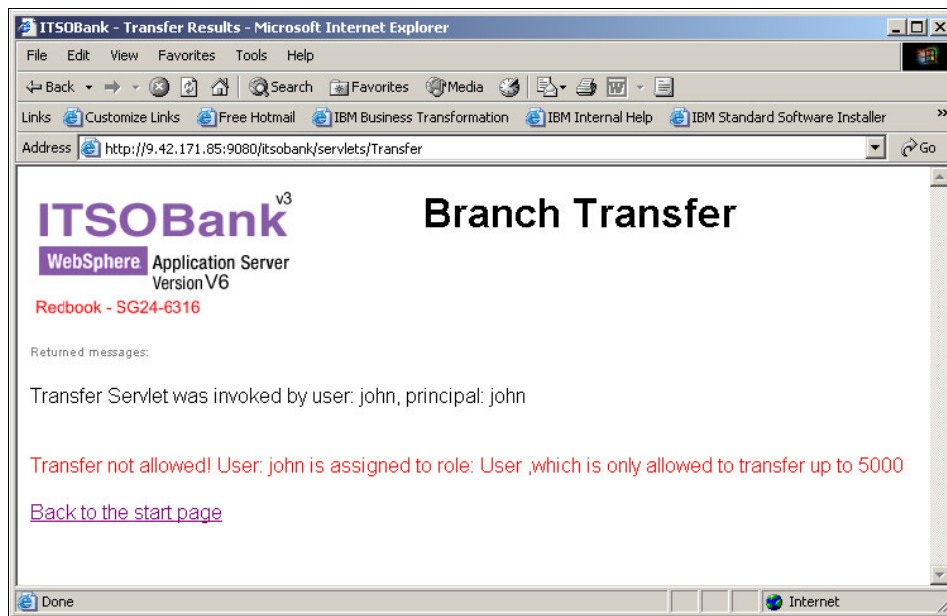


Figure 7-13 Programmatic login sample response show that transfer was not allowed

In Example 7-15, the first line shows which user, under which principal, called the TransferServlet servlet. The code in Example 7-15 is responsible for this line. You can see which servlet programmatic methods are used.

Example 7-15 Use of the getRemoteUser() and getUserPrincipal() methods

```
// check the transfer amount for restricted roles
Principal principal=req.getUserPrincipal();
if(principal!=null) {
    messagePrincipal="Transfer Servlet was invoked by user:
"+req.getRemoteUser()+", principal: "+principal.getName();
```

The second line in the Branch Transfer result page shows a negative response, which means that the transfer was not allowed. In the code, we check if the user is mapped to the User role and whether the wanted transfer amount exceeds the specified limit (5000) for which transfer is not allowed. Example 7-16 shows the code snippet.

Example 7-16 The use of isUserInRole() methods

```
if(message==null && req.isUserInRole(restrictedRole) &&
transferAmount>maxWebTransferAmount.intValue())
    message="Transfer not allowed! User: "+req.getRemoteUser()
+" is assigned to role: " + restrictedRole +" ,which is only allowed
to transfer up to "+ maxWebTransferAmount;
else
    message="Transfer initiated between customer:"+customerID+"
and branch:"+branchID+", the amount of:"+transferAmount;
```

5. Click **Back to the start page** link.
6. Click the **Log out** button to logout.
7. Log in again. This time log in with another user that must not be mapped to the User security role.

8. Repeat steps 3 and 4 on page 145. This time you see the response shown in Figure 7-14.

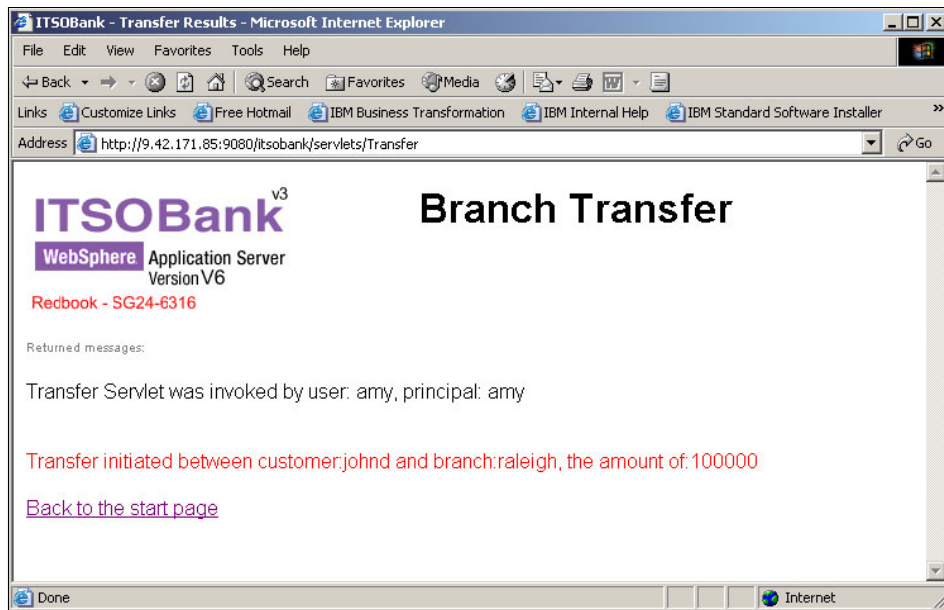


Figure 7-14 Programmatic login sample response show that transfer was fine

7.5 Additional transport security, authentication, and authorization options

In the following section, many additional transport security, authentication, and authorization options for Web servers and WebSphere are described.

7.5.1 Configuring LDAP authentication with IBM HTTP Server

This section presents a simple scenario of how to implement basic HTTP authentication for the Web server when the user registry is stored in an LDAP directory for the IBM HTTP Server. To test the scenario, we used IBM Tivoli Directory Server Version 6.0. In the following instructions, all the software has been installed and you already have an LDAP server populated with users. (See Figure 2-2 on page 11 for details about LDAP data structure.)

Enable security for all the static Web components in the C:\IBM\HTTPServer\htdocs\en_us directory.

Preparing the necessary configuration files

The following steps show which files must be defined for the Web server and how to use those files. The `ldap.prop` file is an LDAP configuration file (Example 7-17) for the Web server. It is stored in the `conf` directory of the server. In this case, it is `C:\IBM\HTTPServer\conf`. A sample LDAP configuration file with an explanation of each directive is supplied with Web server software. For basic authentication, the following entries are included.

Example 7-17 LDAP configuration for IBM HTTP Server

```
ldap.realm=LDAP Realm
ldap.url=ldap://kcgl6kh.itso.ra1.ibm.com/o=itso,c=us
ldap.transport=TCP
ldap.application.authType=Basic
ldap.application.DN=cn=root
ldap.application.password.stashFile=C:\IBM\HTTPServer\bin\ldap.sth
ldap.user.authType=Basic
ldap.user.name.filter=(&(objectclass=inetOrgPerson)(cn=%v1))
ldap.group.name.filter=(&(cn=%v1)(|(objectclass=groupofnames)(objectclass=groupofuniquenames)))
ldap.group.memberAttributes=member uniquemember
```

Tip: If you are using Windows, make sure that you use short file name types when specifying a fully qualified path name for the configuration files in `ldap.prop` as in the following example:

```
C:\Progra~1\IBM\HTTPServer\bin\ldap.sth
```

Define the following files for the Web server:

- ▶ The `ldap.url` file is of the form `ldap://<hostName>/<BaseDN>`.
- ▶ The `ldap.application.DN` file is the distinguished name (DN) by which the Web server authenticates itself to the LDAP server.
- ▶ The `ldap.sth` file is a stash file that contains an encrypted password for the Web server to authenticate with LDAP. If you are using Windows, ensure that you specify the fully qualified path name with a short Windows file name. The use of quotation marks does not work.

Decide with which user name and password the Web server connects to LDAP. To create the stash file, at the command prompt, enter:

```
C:\IBM\HTTPServer\bin\ldapstash <password>
C:\IBM\HTTPServer\bin\ldap.sth
```

Configuring your Web server to use LDAP for authentication

To configure the IBM HTTP Server to use LDAP for authentication:

1. Open the `httpd.conf` file, which is the configuration file for IBM HTTP Server.
2. Add the `ibm_ldap_module` definition to the end of the LoadModule list.

Note: There is a difference between whether you are specifying an LDAP module for UNIX, Linux, or Windows. See Example 7-18.

Example 7-18 Adding ibm_ldap_module definition to httpd.conf

```
#LoadModule deflate_module modules/mod_deflate.so
LoadModule ibm_ssl_module modules/mod_ibm_ssl.so

#For Windows only
LoadModule ibm_ldap_module modules/IBModuleLDAP.dll

#For UNIX/Linux only
LoadModule ibm_ldap_module modules/mod_ibm_ldap.so
```

3. Add the Directory directive to protect the `C:\IBM\HTTPServer\htdocs\en_us` directory. This directory is set as a global Web server root. Therefore, when you call the Web server by using only the host name, the Web server searches for the `index.html` file under the Web server root.

Within Directory, specify additional security directives that are effective only on that scope (Example 7-19).

Example 7-19 Configuring HTTP basic authentication with LDAP module

```
<Directory "C:/IBM/HTTPServer/htdocs/en_US">

LdapConfigFile "C:/IBM/HTTPServer/conf/ldap.prop"
AuthName "LDAP Realm"
AuthType Basic
Require valid-user
Options None
AllowOverride None

</Directory>
```

4. Save the `httpd.conf` file, and restart the Web server for the changes to take effect.

Testing LDAP authentication with Web server

To test how LDAP authentication works:

1. Open a new browser window on the machine where the Web server is running.
2. In the address bar, type `http://localhost/` or the address for the Web server.
3. In the authentication window, enter a valid user and password from your LDAP user registry scheme. According to the user registry scheme that we use in this book, we entered the information as shown in Figure 7-15. Click **OK**.

If the authentication process goes smoothly, you get the standard IBM HTTP index page output.

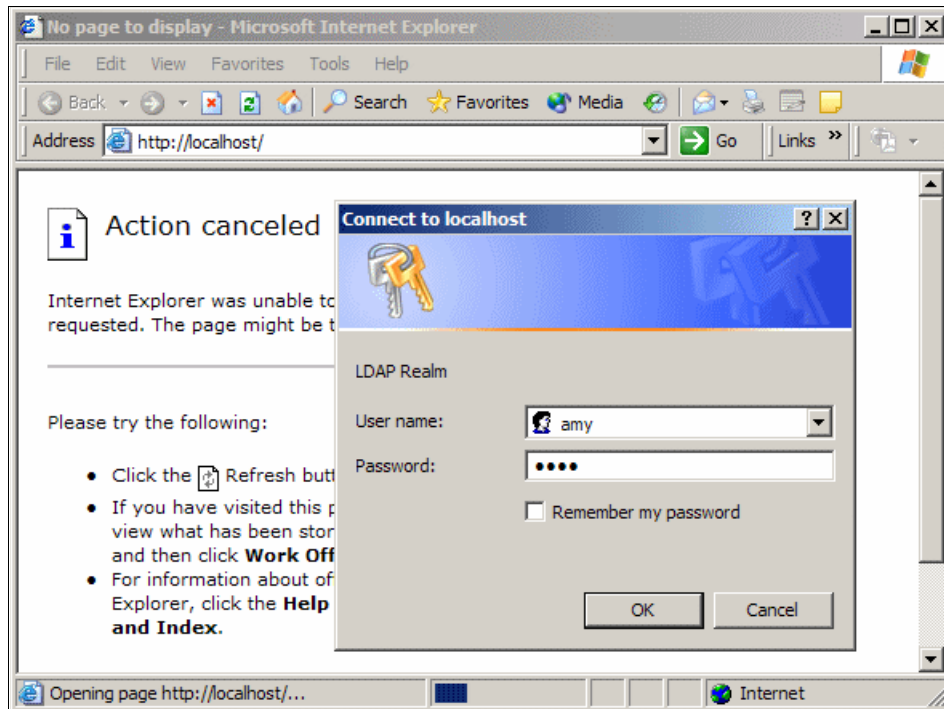


Figure 7-15 Testing the LDAP authentication module for IBM HTTP Server

4. Before starting IBM HTTP Server, enable traces for the LDAP module by exporting the following variables into the system:

```
LDAP_DEBUG=65535
LDAP_TRACE_FILE=C:\ldaptrace.txt
```


After successful authentication in the LDAP module trace, you receive output similar to what is shown in Example 7-20. The output shows a successful connection to the LDAP server and a successful authentication query.

Example 7-20 The LDAP module trace output

```
ldap_authenticate(): entered
LDAP_obtain_session()
LDAP_authenticate_user()
auth_type (BASIC)
calling LDAP_authenticate_user_using_basic
LDAP_authenticate_user_using_basic(): user_name (amy)
LDAP_user2DN(): user_name (amy)
LDAP_user2filter(): user_name (amy)
LDAP_prepare_filter(): template
((&(objectclass=inetorgperson)(cn=%v1)))
the resulting filter: ((&(objectclass=inetorgperson)(cn=amy)))
LDAP_prepare_filter(): returning 0
LDAP_user2filter(): returning 0
LDAP_perform_search(): base (o=itso,c=us), filter
((&(objectclass=inetorgperson)(cn=amy)))
looking in the cache for the base, scope and filter
LDAP_cache_find_ele(): entered
looking for base [o=itso,c=us], filter
[(&(objectclass=inetorgperson)(cn=amy))], scope [2]
no cached answer
reusing 'LDAP Realm' application connection.
Search start: 1100719467, end: 1100719467
converted '(&(objectclass=inetorgperson)(cn=amy))' to DN
'cn=amy,ou=users,o=itso,c=us' for realm 'LDAP Realm'
adding DN (cn=amy,ou=users,o=itso,c=us) to cache
LDAP_cache_find_ele(): entered
looking for base [o=itso,c=us], filter
[(&(objectclass=inetorgperson)(cn=amy))], scope [2]
LDAP_perform_search(): returning 0
LDAP_user2DN(): returning 0
using DN (cn=amy,ou=users,o=itso,c=us)
calling LDAP_obtain_connection
: LDAP_open_connection(): using LDAP V3 API, cp->Version (3)
: connecting to [9.42.171.77, 389]
: cp->Version (3); cp->Transport (TCP)
: LDAP_init(9.42.171.77, 389)
: connected
: setting deferrals
: setting timeout
```

```
: not an application connection
: opened new user connection for 'LDAP Realm'; expiration:
1100720067
calling LDAP_simple_bind_s() with DN (cn=amy,ou=users,o=itso,c=us)
successful authentication
updating the password cache
LDAP_cache_find_ele(): entered
looking for base [o=itso,c=us], filter
[(&(objectclass=inetorgperson)(cn=amy))], scope [2]
cache: [o=itso,c=us], [(&(objectclass=inetorgperson)(cn=amy))], [2]
setting correct password for 'cn=amy,ou=users,o=itso,c=us' cache
LDAP_authenticate_user_using_basic(): returning 0
LDAP_authenticate_user(): returning 0
LDAP_release_session()
```

7.5.2 Configuring SSL certificate-based client authentication for the IBM HTTP Server

The Web client can also provide a digital certificate to provide an identity during an SSL initialization.

This section explains how to use client-side certificates with your Web server and WebSphere Application Server. It also shows how to configure your servers to support client-side certificates and use them as a base for user authentication.

Creating a personal certificate

Typically, the creation of a client-side certificate involves a Certificate Authority. However, for purpose of this section, we create a self-signed personal certificate that we import into the key store databases of the Web server and WebSphere.

To request and install a personal client-side certificate on Windows:

1. Use the iKeyman tool to create a new key store and a self-signed personal certificate.

The PKCS12 format: When creating a new key store, use PKCS12 database format.

2. In the Create New Self-Signed Certificate window, enter the values shown in Figure 7-16. Then click **OK**.

The screenshot shows a dialog box titled "Create New Self-Signed Certificate". It contains the following fields and values:

Key Label	Client
Version	X509 V3
Key Size	1024
Common Name	John
Organization	ITSO
Organization Unit (optional)	users
Locality (optional)	
State/Province (optional)	
Zipcode (optional)	
Country or region	US
Validity Period	365 Days

Buttons: OK, Reset, Cancel

Figure 7-16 Creating a new self-signed certificate to use with a Web browser

3. After the certificate is successfully created, extract it to a .arm file and then import it to Web server's CMS type key datastore.

For more details about creating self-signed certificates, see *WebSphere Security Fundamentals*, REDP-3944.

Importing the certificate into your Web browser

Now, you must import the certificate into your Web browser. In our case, we use Windows Internet Explorer. The procedure is similar for other Web browsers.

1. From the Internet Explorer menu bar, select **Tools** → **Internet Options** → **Content** → **Certificates**.
2. From the Intended purpose list, select **Client Authentication**.
3. Click the **Personal** tab and then click **Import**.
4. In the Certificate Import Wizard, click **Next**.
5. Browse for the p12 file, the PKCS12 key datastore that was created for this purpose. Click **Next**.

6. Provide the password to open the p12 file. Click **Next**.
7. Select the **Enable strong private key protection** check box. A prompt pop-up window opens every time the certificate is accessed.
8. Click **Next**.
9. Click **Finish**.
10. In the Creating a new private exchange key! window, set the security level to medium and click **OK**.
11. On the **Personal** tab, on which you see the certificate that was just imported, select the certificate and click the **Advanced...** button.
12. Under the Advanced Options, select the **Client Authentication** check box.

Modifying the Web server to support client certificates

You must ensure that the selected Web server is configured to request client-side certificates. Use IBM HTTP Server to configure SSL for the Web Server, to force the clients to send their certificates.

1. Enable and configure SSL to use client-side certificates as the Web server requires. Follow the steps in 7.2.1, “Securing the transport channel between the Web browser and Web server” on page 111, to enable SSL for your IBM HTTP Server if you have not done so yet.
2. Open the httpd.conf file.
3. Within other SSL configuration directives, add the SSLClientAuth directive as shown in Example 7-21.

Example 7-21 Adding client authorization directive to SSL definition

```
SSLDisable
Listen 443

<VirtualHost kg16kh:443>
SSLEnable
KeyFile "C:/IBM/HTTPServer/conf/keys/IHS6Certificates.kdb"
SSLClientAuth required
</VirtualHost>
```

This is the most basic SSL setup for certificate client authentication. However, there are other SSL directives that you can use to set the SSL configuration more specifically to your requirements. For further explanation, see the IBM HTTP Server documentation.

4. Save the httpd.conf configuration file and restart IBM HTTP Server.

Testing the client-side certificate with IBM HTTP Server

To test client certificate authentication:

1. Open a new Internet Explorer window and enter the following URL:
`https://localhost/`
2. In the window that opens (Figure 7-17), click **OK** to allow Internet Explorer to access your personal certificate. This window opens because the certificate is protected.

The default IBM HTTP Server opens.

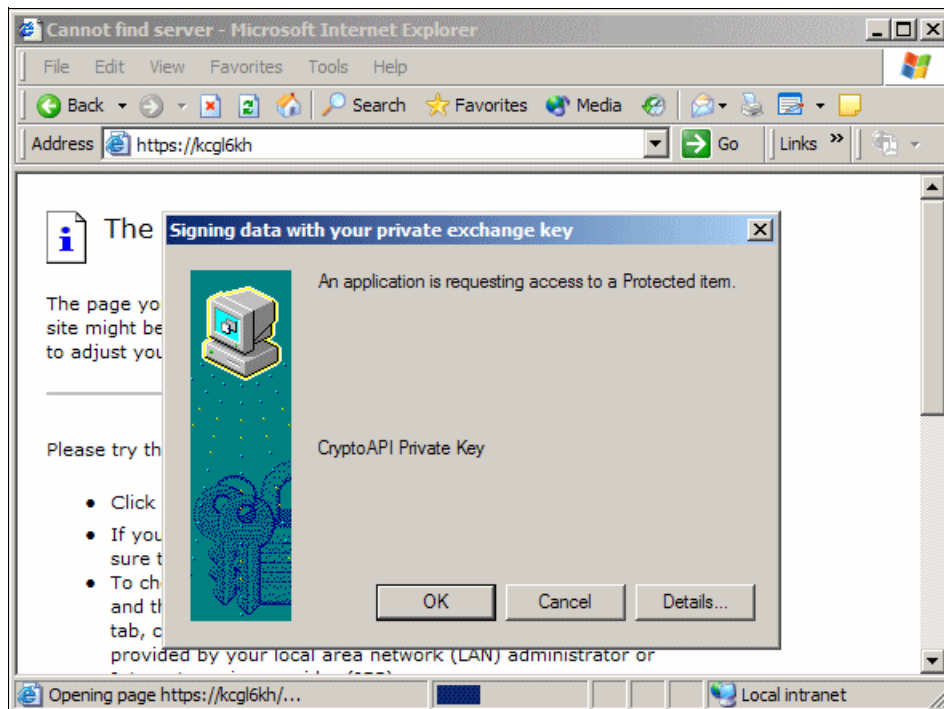


Figure 7-17 Testing client certificate authentication with IBM HTTP Server

3. To see what is happening in IBM HTTP Server, set the log level directive in `httpd.conf` configurational file as `LogLevel Debug`.

After successful client authentication, with logs set to debug, you can get similar entries in the IBM HTTP Server `error.log` file, as shown in Example 7-22 on page 156.

```
[info] [client 9.42.171.157] [9dec10] Session ID:
zAsAAK0jBuboHXV9qLYv707Ku+BYWFhY8Ze3RAEAAAA= (new)
[info] Cert Body Len: 664
[info] Serial Number: 44:b7:95:11
[info] Distinguished name CN=John,OU=users,O=ITSO,C=US
[info] Common Name: John
[info] Country: US
[info] Organization: ITSO
[info] Organization Unit: users
[info] Issuer's Distinguished Name: CN=John,OU=users,O=ITSO,C=US
[info] Issuer's Common Name: John
[info] Issuer's Country: US
[info] Issuer's Organization: ITSO
[info] Issuer's Organization Unit: users
```

7.5.3 Configuring SSL certificate-based client authentication for WebSphere Application Server

This section explains how to configure client certificate authentication for your applications. When you use client certificate authentication with your Web modules, WebSphere security service attempts to map the data from the digital certificate with the user data of the selected user account repository. The repository can be one of the following types:

- ▶ Local OS registry

Note: If a Local OS registry is used, the certificate DN is parsed and the name between the first equal sign (=) and the comma (,) is used as the mapped name. If the DN does not contain the = sign, the complete name is used. If there is no comma in the DN, everything after the = sign is used as the name.

Important: Only Java client certificate authentication is supported with a local OS user registry. Web client certificate authentication is not supported.

- ▶ Stand-alone LDAP registry

If you use an LDAP server for the user account repository, WebSphere provides two ways of matching client certificate information to LDAP: mapping by exact DN and mapping by filtering certificate attributes. Both options are described in the sections that follow.

► Federated repositories

If you use federated repositories for the user account repository, WebSphere provides the capability to match client certificate information to one or more LDAP registries.

Important: Client certificate login is not supported in a security realm that includes the built-in, file-based repository. Because of this, you cannot choose federated repositories as your user account repository unless the built-in, file-based repository is removed from the realm.

If the certificate successfully maps to a user, the holder of the certificate is regarded as the user in the registry and is authorized as this user.

Configuring J2EE Web application for client certificate authentication

By specification in the J2EE application Web module, the authentication method can be configured to be one of five available types, including unspecified. This is done in the Web deployment descriptor file.

To configure the ITSObank sample application for client certificate authentication:

1. Load the `itsobank.ear` application file into Rational Application Developer.
2. In the J2EE perspective, expand **Dynamic Web Projects** → **itsobank**.
3. Double-click the Deployment Descriptor of the **itsobankWeb** module.
4. On the Web Deployment Descriptor page, select the **Pages** tab:
 - a. Scroll down to the Login section.
 - b. In the Login section (Figure 7-18), select the **CLIENT_CERT** authentication method. You do not have to log in and log out of pages anymore. They can just be deleted.

▼ Login
The following Login configuration values are used for this web application:

Authentication method:

Realm name:

Login page:

Error page:

Figure 7-18 Configuring Web module for client certificate authentication

5. Save the changes. In your application, you only end up changing the Web module deployment descriptor, which is the `Web.xml` file. The changes are in the `login-config` tag, as shown in Example 7-23.

Example 7-23 Changes in the login-config tag

```
<login-config>
  <auth-method>CLIENT-CERT</auth-method>
  <realm-name>ITS0 Bank</realm-name>
</login-config>
```

6. Export the application EAR file. For testing purposes, we install it on a WebSphere Application Server.

Configuring WebSphere for the LDAP Certificate Filter option

In this section, the following prerequisites must be met:

- ▶ You successfully installed a personal certificate into a client Web browser. For details, see “Creating a personal certificate” on page 152 and “Importing the certificate into your Web browser” on page 153.
- ▶ The WebSphere Web container is configured to use the SSL configuration that uses your previously configured key store. For details, see “Creating a new SSL configuration” on page 119 and “Modifying the Web Container configuration to support SSL” on page 121.
- ▶ If using the federated repositories for your User account repository, the realm includes at least one LDAP registry and *does not* include the built-in, file-based repository.

It is anticipated that the personal certificate subject DN does not necessarily match, in any way, to your LDAP DN.

In the following sample, you use a self-signed personal certificate as described in “Creating a personal certificate” on page 152.

The SubjectDN value of the certificate in our case is:

```
CN = John
OU = users
O = ITS0
C = US
```

Next, modify the WebSphere LDAP filtering rules to map the certificate subject DN field to the IBM Tivoli Directory Server LDAP `uniqueIdentifier` field for a given user. You are not required to use the `uniqueIdentifier` field. However, you must ensure that the data type of the field selected is capable of handling the specific

value and that the certificate attribute selected for authentication is unique between certificates.

Also ensure that WebSphere has the right to search such a field when performing authentication.

To configure WebSphere Application Server to use the certificate filter as required:

1. Log in to the WebSphere Administration Console.
2. Select **Security** → **Secure administrative, applications, and infrastructure**.
3. In the User account repository section, select your realm definition from the list and click **Configure**.

Note: For this section, we assume that you are using the stand-alone LDAP registry. However, we used both the stand-alone LDAP registry and a single LDAP registry in the federated repositories. Both registries work, and where appropriate the alternate steps are noted.

4. In the Additional Properties section (Figure 7-19), click **Advanced LDAP user registry settings** link option.

The screenshot shows the 'Security' configuration page. It includes the following fields and options:

- Bind distinguished name:** cn=root
- Bind password:** Masked with dots
- Login properties:** uid
- Certificate mapping:** CERTIFICATE_FILTER (dropdown menu)
- Certificate filter:** uid=\${SubjectDN}
- Require SSL communications
- Centrally managed
 - [Manage endpoint security configurations](#)
- Use specific SSL alias
 - CellDefaultSSLSettings (dropdown menu) [SSL configurations](#)

Figure 7-19 Setting the CERTIFICATE_FILTER client certificate mapping for an LDAP registry in federated repositories

Note: If using federated repositories, in the Related items section, click **Manage repositories**. Then click the LDAP registry that you want to configure. The certificate mapping and filter settings are in the Security section of the LDAP configuration page (Figure 7-19).

5. In the Advanced LDAP user registry settings page (Figure 7-20):
 - a. For Certificate Map Mode, type CERTIFICATE_FILTER.
 - b. For Certificate Filter, type uid=\${SubjectDN}.
 - c. Click **OK**.

Secure administration, applications, and infrastructure

[Secure administration, applications, and infrastructure](#) > [Standalone LDAP registry](#) > **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**

Specify advanced Lightweight Directory Access Protocol (LDAP) user registry settings when users and groups reside in an external LDAP directory. When security is enabled and any of these advanced settings are changed, go to the Security > Secure administration, applications, and infrastructure panel. Click Apply or OK to validate the changes.

Configuration

General Properties

User filter
(&(uid=%v)(objectclass=inet)

Group Filter
(&(cn=%v)(objectclass=gro

User ID map
*:uid

Group ID map
*:cn

Group member ID map
ibm-allGroups:member;ibm-

Perform a nested group search

Certificate map mode
CERTIFICATE_FILTER

Certificate filter
uid=\${SubjectDN}

Apply OK Reset Cancel

Figure 7-20 Setting the CERTIFICATE_FILTER client certificate map mode for a stand-alone LDAP registry

6. Save the configuration for WebSphere.
7. Restart the application server for the changes to take effect.

Testing the client-side certificate

Test the client certificate authentication by using the default application that ships with WebSphere and use the snoop servlet by accessing it with your Web browser:

1. Make sure that your Web server and the default application are started.
2. Open a new browser window and access the following address from the client to determine whether your browser is correctly passing a client certificate:

`https://<your_webserver_name>/snoop`

Important: The default application is BASIC authentication enabled, which also means that client certificate authentication is not selected (the property in the Web deployment descriptor). Because of this, when accessing snoop, it still prompts for BASIC authentication login, as shown in Figure 7-21.

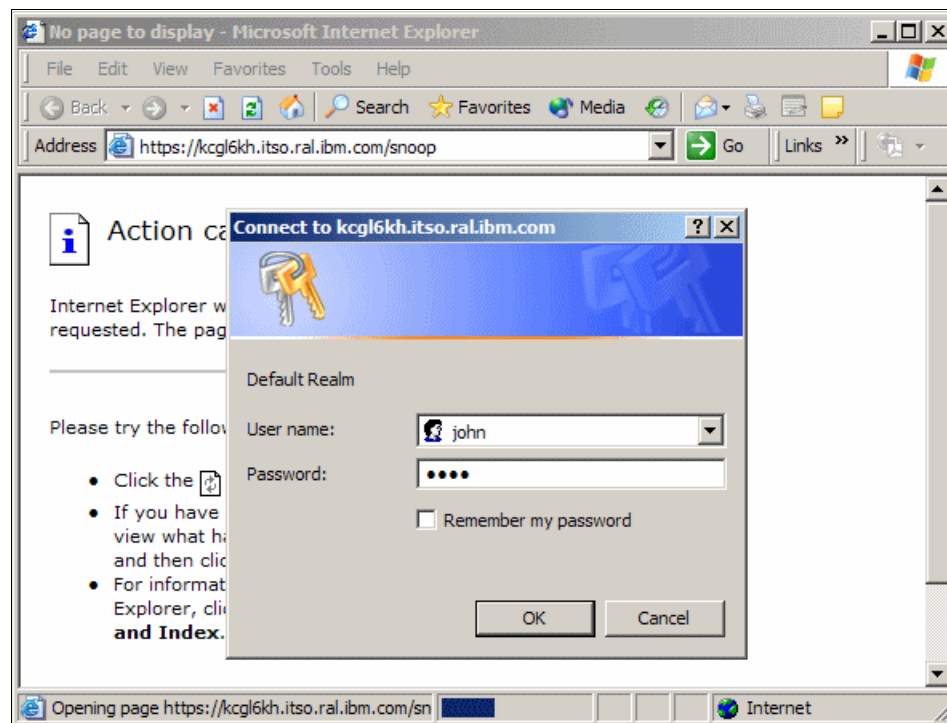


Figure 7-21 The snoop servlet application prompting for basic authentication login

3. Enter the login information. In our case, we enter the following information:
 - For User Name, type john.
 - For Password, type test.
4. When the snoop servlet shows various request related information, scroll down to the HTTPS Information section (Figure 7-22). You can see that our certificate data shows client certificate chain information. When a client certificate SSL is not used or if a client fails to pass a certificate, WebSphere only returns the Cipher suite specification as used in the HTTPS connections. No client certificate chain is displayed.

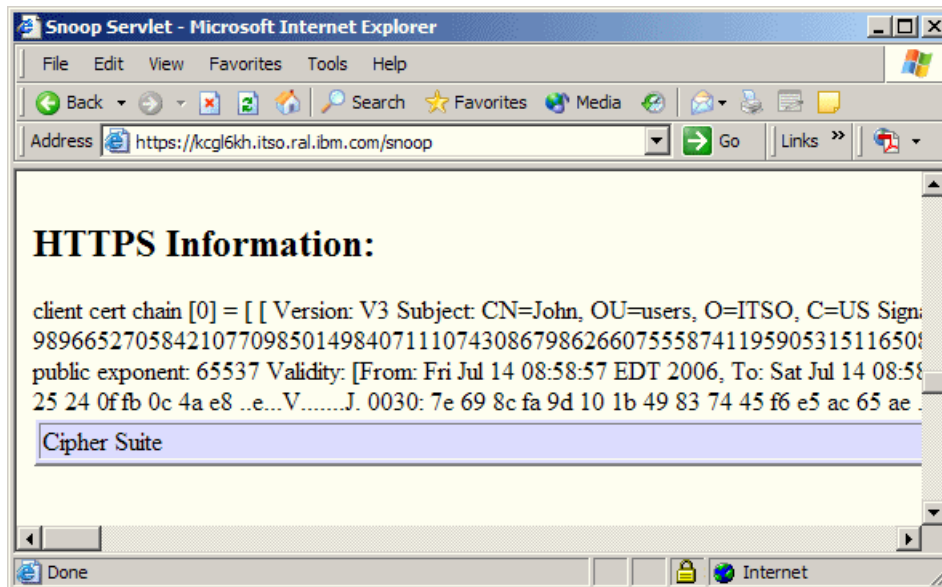


Figure 7-22 Response from the snoop servlet showing the client certificate

Now, create a test with your client certificate authentication enabled ITSObankWeb sample application:

1. Ensure that ITSObank application is installed and started.
2. Open a new browser window and access the following address from the client:

`https://<your_webserver_name>/itsobank/index.html`

Because client authentication is enabled, the server requests a client certificate during SSL handshake and consequently.

3. In the window that opens in your browser, when prompted, select a client certificate, as shown in Figure 7-23. Select the right client certificate. You might have more than one installed in your browser. Then click **OK**.

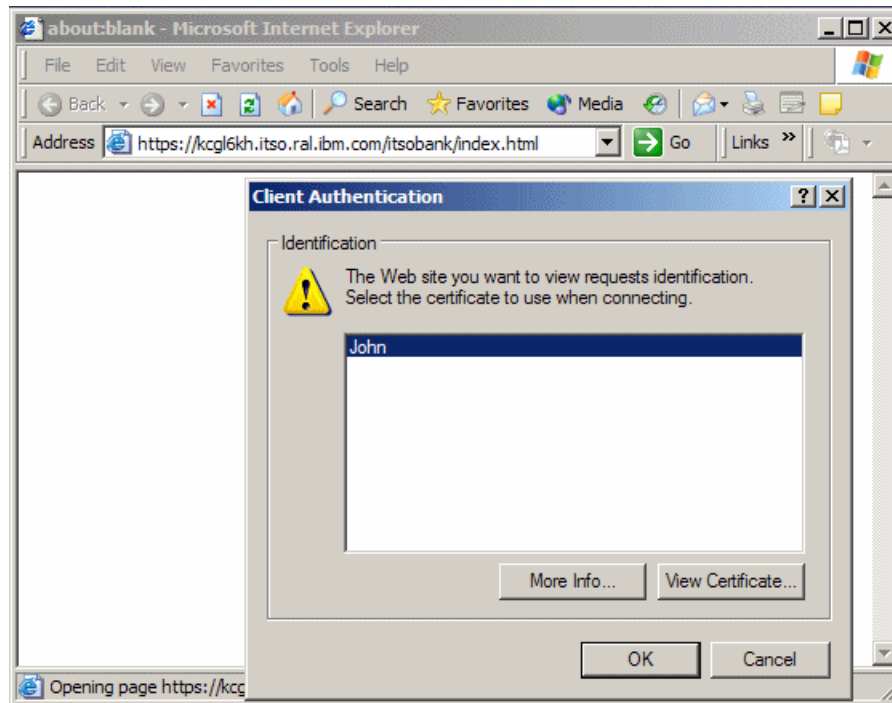


Figure 7-23 The browser prompts to select client certificate

4. Depending on the browser type and settings, if you see another window for extra confirmation to receive the server SSL certificate, click **OK**.

If the SSL handshake goes smoothly, WebSphere maps the data from the client certificate and authenticates the user. If the user defined in your certificate is authorized to have access to ITSObank application, you see the initial window similar to the one in Figure 7-24 on page 164.

No login form window is displayed. WebSphere authenticates the user with the data that is stored in the SSL client certificate. Also, you might still have remaining login and logout pages in your application. These pages are now dysfunctional.

Important: The Logout button no longer not functions. To log out, you must close the browser window. Otherwise, while it stays open, the browser sends the client certificate automatically when it requires access to the application.



Figure 7-24 Successful client certification login to ITSOBank application

You can follow the operation of the authentication if you have tracing enabled for security. You must be able to find, in your trace.log file, something similar to the code in Example 7-24.

Example 7-24 The trace.log file for the federated repositories LDAP registry authentication test

```

...
WebConstraint > getConstraints: Entry
    /index.html
    GET
...
WebConstraint 3   webConstraintsTable.length = 3
WebConstraint 3   webConstraintsTable.length = 3
WebConstraint > getRequiredRoles : /index.html GET Entry
WebConstraint 3   Required roles are
WebConstraint 3   User
WebConstraint 3   Manager
WebConstraint 3   .
WebConstraint < getRequiredRoles Exit
WebAuthentica > authenticate Entry

```

```

WebAuthentica > handleSSO Entry
WebAuthentica < handleSSO: no cookies present in the request. Exit
WebAuthentica > handleCertificates Entry
WebAuthentica 3 Challenge type used is CERT.
00000036 WebAuthentica 3 Map credential for this certificate.
00000036 UserRegistryI > mapCertificate Entry
[
[
  Version: V3
  Subject: CN=John, OU=users, O=ITSO, C=US
  Signature Algorithm: MD5withRSA, OID = 1.2.840.113549.1.1.4

  Key: IBMJCE RSA Public Key:
  modulus:
98966527058421077098501498407111074308679862660755587411959053151165089
68201190145559405820101428730334415495592381640547308186423677327644831
44116225187250555795206466929778340899790785112795760951597519247963907
37420062217125085382010987012738935408836882825330277013381463050332575
316231938006583657106291
  public exponent:
65537

  Validity: [From: Fri Jul 14 08:58:57 EDT 2006,
             To: Sat Jul 14 08:58:57 EDT 2007]
  Issuer: CN=John, OU=users, O=ITSO, C=US
  SerialNumber: [1152881937]

]
  Algorithm: [MD5withRSA]
  Signature:
0000: 53 b2 4f 24 d7 98 bd 76 02 3a 6a 68 36 5d 97 71
S.O....v..jh6..q
0010: 9d 19 7a a6 e9 02 77 49 03 b4 97 66 cb 7a 26 a8
..z...wI...f.z..
0020: 83 84 65 8a 40 dd 56 ed df 25 24 0f fb 0c 4a e8
..e...V.....J.
0030: 7e 69 8c fa 9d 10 1b 49 83 74 45 f6 e5 ac 65 ae
.i.....I.tE...e.
0040: d8 40 42 40 5a 79 7b 49 71 5b 52 18 dd 07 fd 8c
..B.Zy.Iq.R....
0050: 0b d0 f1 db 31 01 0f dd 34 8b d0 75 02 a6 12 e8
....1...4..u....
0060: 46 95 8a a6 71 48 8e b1 3f 00 d1 5c 3a e8 46 d7
F...qH.....F.

```

```

0070: ce 39 59 e3 c2 aa ce f5 b4 55 6d 1c 2a 03 89 27
.Y.....Um.....
]
UserRegistryI < mapCertificate Exit
    john
UserRegistryI > createCredential Entry
    john
UserRegistryI > getRealm Entry
UserRegistryI < getRealm Exit
    defaultWIMFileBasedRealm
RegistryUtil > appendRealm Entry
    user
    cn=john,ou=users,o=itso,c=us
    defaultWIMFileBasedRealm
RegistryUtil > getRealmWithSep Entry
    defaultWIMFileBasedRealm
RegistryUtil < getRealmWithSep Exit
    :defaultWIMFileBasedRealm/
RegistryUtil < appendRealm Exit
    user:defaultWIMFileBasedRealm/cn=john,ou=users,o=itso,c=us
UserRegistryI 3 securityName used in the credential is:
    john
UserRegistryI < createCredential Exit
    com.ibm.ws.security.auth.WSCredentialImpl@df40df4
WebAuthentica 3 Storing certificates in the credential
WebAuthentica < handleCertificates Exit
...
WebConstraint > getConstraints: Entry
    /transfer/accountsview.html
    GET
...
WebConstraint > getRequiredRoles : /transfer/accountsview.html GET
Entry
WebConstraint 3 Required roles are
WebConstraint 3 Manager
WebConstraint 3 .
WebConstraint < getRequiredRoles Exit
WebAuthentica > authenticate Entry
WebAuthentica > handleSSO Entry
WebAuthentica 3 Attempting primary cookie validation for: LtpaToken2
WebAuthentica > getCookieValues Entry
    LtpaToken2
...
WebAuthentica 3 The LTPA token was valid.
...

```


WebCollaborat A SECJ0129E: Authorization failed for john while invoking GET on default_host:itsobank/transfer/accountsview.html, Authorization failed, Not granted any of the required roles: Manager ...

To get the traces as shown in Example 7-24, set the trace string shown in Example 7-25 for the application server by selecting **Application Servers** → **server1** → **Logging and Tracing** → **Diagnostic Trace** → **Change Log Detail Levels**.

Example 7-25 Trace string for the application server

```
com.ibm.ws.security.web.WebAuthenticator=all:  
com.ibm.ws.security.web.WebConstraintsTable=all:  
com.ibm.ws.security.web.WebAccessContext=all:  
com.ibm.ws.security.registry.UserRegistryImpl=all:  
com.ibm.ws.security.registry.RegistryUtil=all
```

First we requested the `index.html` page for which WebSphere checked the security roles that are authorized to get the page. Furthermore, it extracted the user information from the SSL certificate, matched it with the data from the LDAP realm in the federated repositories, and checked if the user was in the required security role. Afterward, we also tried to get another resource, `/transfer/accountsview.html`, but were unable to achieve the authorization because the user was not in the required role.

Configuring WebSphere to use the exact DN mapping option

Using the DN from the certificate to look up the user means that the directory structure where the user is must match the DN exactly. For example, if the user DN is `cn=john,ou=users,o=ITSO,c=US`, John must be under the user's organizational unit (ou), ITSO, US country (c) in this order.

For this section, WebSphere must already be configured to use an LDAP registry, either as standalone or in the federated repositories, which contains some directory structure. Figure 2-2 on page 11 shows the directory structure that we used.

To configure WebSphere Application Server to use Exact DN mapping:

1. Log in to the WebSphere Administration Console.
2. Select **Security** → **Secure administrative, applications, and infrastructure**.
3. In the User account repository section, select your realm definition from the list and click **Configure**.

4. Depending on your User account repository, perform one of the following actions:
 - For a Stand-alone LDAP registry, in the Additional Properties section, click **Advanced LDAP user registry settings**.
 - For Federated repositories, in the Related items section, click **Manage repositories** and then select your LDAP registry.
5. On the Advanced LDAP user registry settings page (Figure 7-25), set the Certificate Map Mode as EXACT_DN.

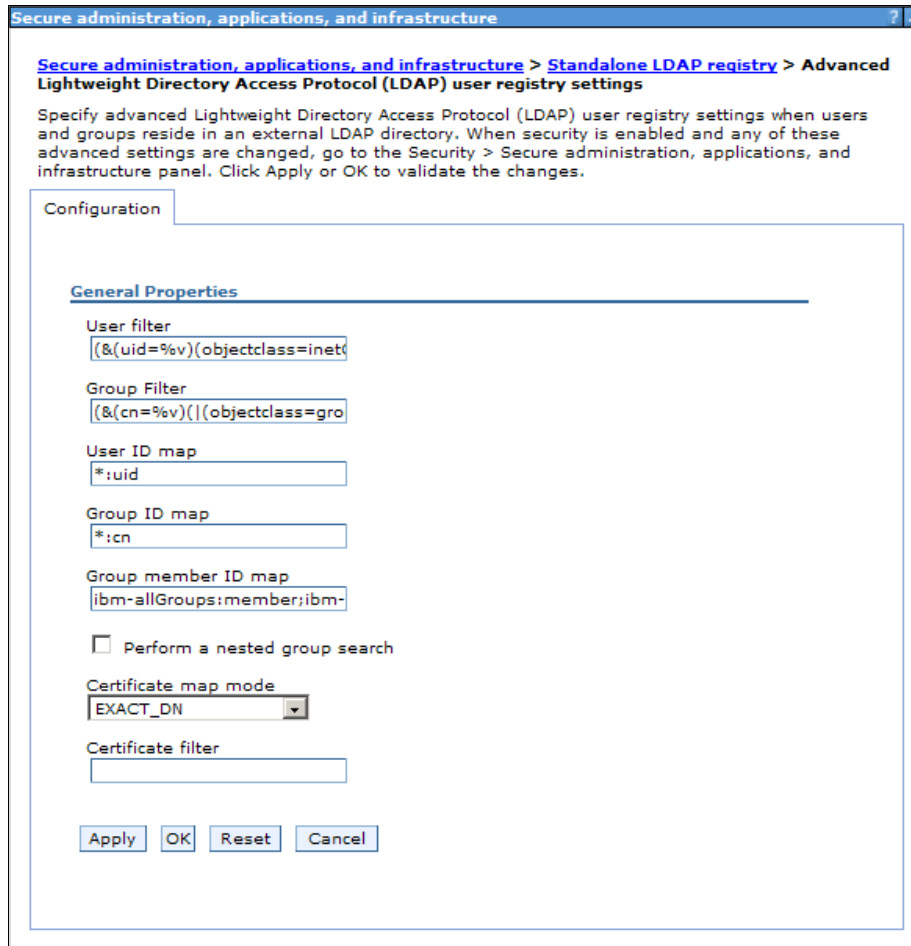


Figure 7-25 The EXACT_DN client certificate map mode for a standalone LDAP registry

For the LDAP repository configuration page in the Federated repositories, change the Certificate mapping selection to EXACT_DN.

6. Make sure that the Certificate Filter field is empty.
7. Click **OK**.
8. Save the configuration for WebSphere.
9. Restart the application server in order for the changes to take effect.

For testing, use the same steps from in “Testing the client-side certificate” on page 161 but with the certificate filter option.

You can follow the operation of the authentication if you tracing is enabled for security. Use the same trace string as shown in Example 7-24 on page 164.



Securing an EJB application

This chapter discusses the security aspects involved within the Enterprise JavaBeans (EJB) part of the Enterprise applications.

This chapter discusses which processing components are usually involved when using EJBs that are running in WebSphere Application Server. The EJB container in WebSphere Application Server that hosts the EJBs. The chapter also discusses how to secure the transport channels to the EJB container, what are the authentication and authorization options available, and how to configure EJB security on the Java 2 Platform, Enterprise Edition (J2EE) level.

EJBs are J2EE components that implement the business logic of an application. They typically have access to sensitive data, and it is important to understand how security is applied to these resources.

There are three types of EJBs:

- ▶ Session Beans, which represent clients inside the J2EE server. Clients call session bean methods to access an application.
- ▶ Entity Beans, which represent persistent business objects in an application's relational database. Typically, each entity bean has an underlying table in the database, and each instance of the bean corresponds to a row in that table.

- ▶ Message-driven Beans, which allow J2EE applications to process messages asynchronously. Message-driven bean methods are invoked by the Application Server run time as part of a message queue processing.

Important: Because queued messages generally do not have any authentication information associated with them, authentication information is unavailable to message-driven bean methods. As a result, securing message-driven beans from unauthorized access is really a matter of securing the message queue.

Security can be applied to EJBs in the following ways:

- ▶ Access control can be applied to individual session and entity bean methods so that only callers who are members of particular security roles can call those methods.
- ▶ Session and entity bean methods that require to be aware of the role or identity of the caller can programmatically call the J2EE application programming interface (API) methods known as *isCallerInRole()* and *getCallerPrincipal()* to determine a caller's role and principal, respectively. When using *isCallerInRole()*, the *security role references* are used, and these are later mapped to security roles.

Note: If WebSphere security is not enabled, or if the EJB is not a protected resource, *isCallerInRole()* returns `false` and *getCallerPrincipal()* returns UNKNOWN.

- ▶ You can delegate session, entity, and message-driven bean methods to execute under the identity of either the caller (default), the EJB server, or a specific security role. This is referred to as the *Delegation Policy* or *Run-As Mode Mapping*.

In the following sections, each of these methods of applying security to EJBs are discussed in detail.

Throughout this section, we use a simple EJB example as shown in Figure 8-1 on page 173. It consists of two stateless session EJBs, which are Hello and SecuredHello. Each has just one simple remote method which returns a hello message when invoked. Additionally, there is a servlet just to make the invoking of EJBs easier. We call the servlet from a browser.

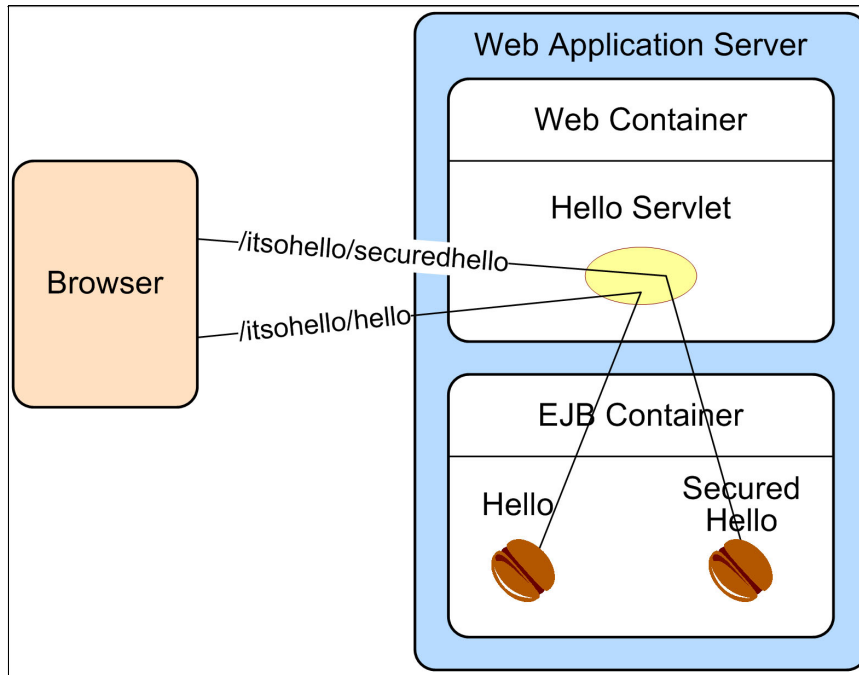


Figure 8-1 Sample application used throughout this section

8.1 Programmatic login (server-side) using JAAS

Programmatic login using Java Authentication and Authorization Service (JAAS) is explained in 9.6.2, “Programmatic login process” on page 235, and in more detail in 9.6.3, “Client-side programmatic login using JAAS” on page 236. Server-side login is similar to the client-side login except that the login occurs on the server side, for example, a servlet or an EJB performs the login.

From the programmer’s point of view, the two types of login are the same, except that on the server side, you cannot use login methods that require user interaction.

8.2 Declarative J2EE security

J2EE security can be applied declaratively or programmatically. WebSphere provides a security infrastructure for application security which is transparent to the application developer. This means that the developer is not required to code for security, because it is all handled at deployment and run time. This is called *declarative security*.

8.2.1 Defining J2EE security roles for EJB modules

The method for defining security roles for EJBs and Web components is similar. To add a role named *BeanVisitor* to the EJB component:

1. In the J2EE perspective, expand **EJB Projects** → **ItsohelloEJB**.
2. Open the Deployment Descriptor of the ItsohelloEJB module.
3. The EJB Deployment Descriptor page opens. Select **Assembly** tab.
4. In the Security Roles section, click **Add** to add a new security role.
5. In the Add Security Role window (Figure 8-2), for the Name field, type `BeanVisitor` and click **Finish**.



Figure 8-2 New Security Role dialog box

A new `<security-role>` tag with your definition is added to the assembly section of EJB deployment descriptor (Example 8-1).

Example 8-1 Security role definition in EJB deployment descriptor

```
<assembly-descriptor>
...
<security-role>
```



```
        <description>Authenticated guest for the EJB</description>
        <role-name>BeanVisitor</role-name>
    </security-role>
    ...
</assembly-descriptor>
```

8.2.2 Security role references

Security role references are used to provide a layer of indirection between security roles named in EJB Java code and security roles that are defined at application assembly time. This allows security roles names to be modified without requiring changes to the application code. See Figure 8-3. You can divide the security role reference usage process into three major parts:

- ▶ Use security role reference in the EJB code

When an EJB uses the `isCallerInRole(Java.lang.String roleName)` J2EE API method to determine whether the caller is a member of a particular role, `roleName` is a security role reference which is later linked to a defined security role in the EJB descriptor file, `ejb-jar.xml`. In Example 8-2, the Java code shows how you can use a security role referenced.

Example 8-2 Security role reference example

```
public String isInRole() {
    if (mySessionCtx.isCallerInRole("RoleReference")) {
        return "You are a member of the referenced role";
    } else {
        return "You are NOT a member of the referenced role";
    }
}
```

- ▶ Reference definition

Every security role reference that is coded must be defined in the assembly descriptor and we use the XML tag `<security-role-ref>` for this purpose. See Figure 8-3 on page 177 (step 2).

- ▶ Reference link

At the application assembly time, all the defined security role references must be linked to one of the existing security role definitions. The XML tag `<role-link>` specified within `<security-role-ref>` in `ejb-jar.xml` deployment descriptor defines the reference link.

In Example 8-3, the XML code shows how the security role reference `RoleReference` can be linked to the security role `BeanVisitor`.

Example 8-3 Security role reference in ejb-jar.xml

```
<enterprise-beans>
  <session id="SecuredHello">
    ...
    <security-role-ref>
      <description>
        The &quot;RoleReference&quot; string is mapped to
        BeanVisitor security role</description>
      <role-name>RoleReference</role-name>
      <role-link>BeanVisitor</role-link>
    </security-role-ref>
    ...
  </session></enterprise-beans>
```

For a security role reference to work as shown in Figure 8-3 on page 177, the security role to which it is linked must be a security role that is defined in the deployment descriptor and mapped to one or more users, groups, or special subjects.

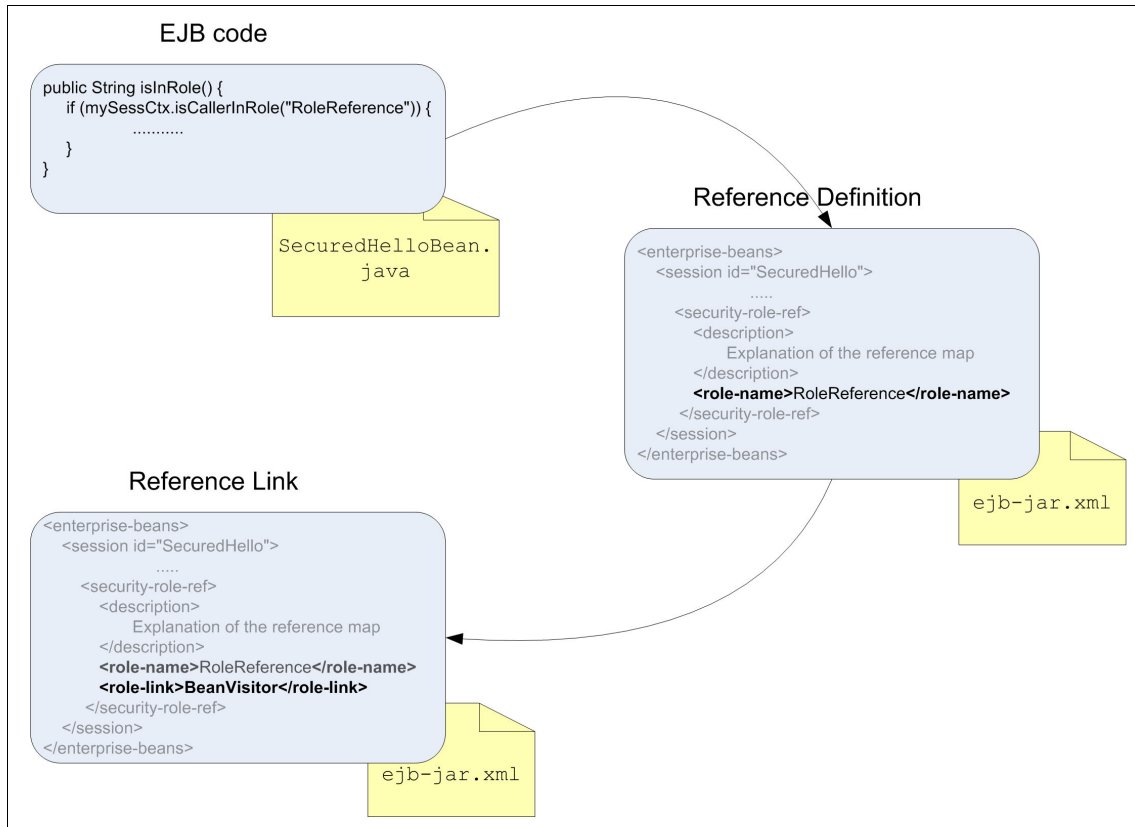


Figure 8-3 Security role references

Linking security role references

To define and link the RoleReference security role reference for the BeanVisitor security role using Rational Application Developer:

1. From the Resource Perspective, navigate to the EJB deployment descriptor file `ejb-jar.xml` and open it.
2. Select the **References** tab.
3. Select the bean that contains the method that calls the `isCallerInRole()` method and click **Add**.

4. In the Add Reference window (Figure 8-4), select **Security Role Reference** and click **Next**.

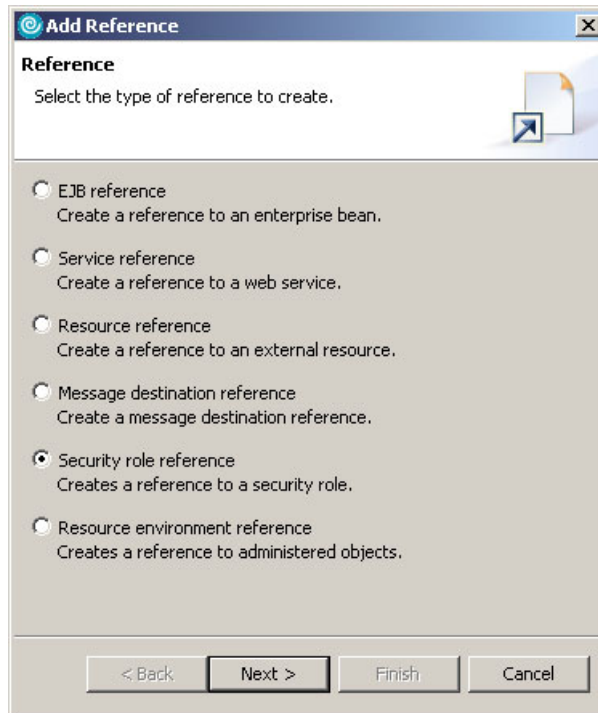


Figure 8-4 Adding security role reference

5. In the Add Security Role Reference window, fill in the values as shown in Figure 8-5. The reference's name is the string that is passed to the `isCallerInRole()` method in the Java code.

The desired security role is selected from the Link pull-down menu. Only security roles which have previously been defined in the EJB module are shown in this menu.

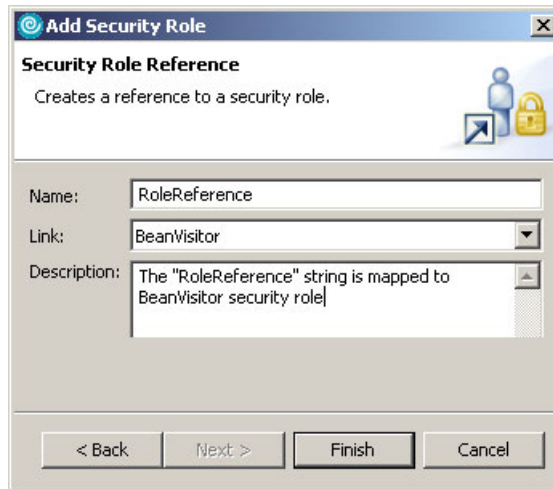


Figure 8-5 Linking security reference role

You can also optionally enter a Description for this security role reference.

Click **Finish** to apply the changes and close the window.

6. Save the deployment descriptor.

Figure 8-6 shows the Reference tab of the EJB deployment descriptor which shows added security reference role for the SecuredHello bean.

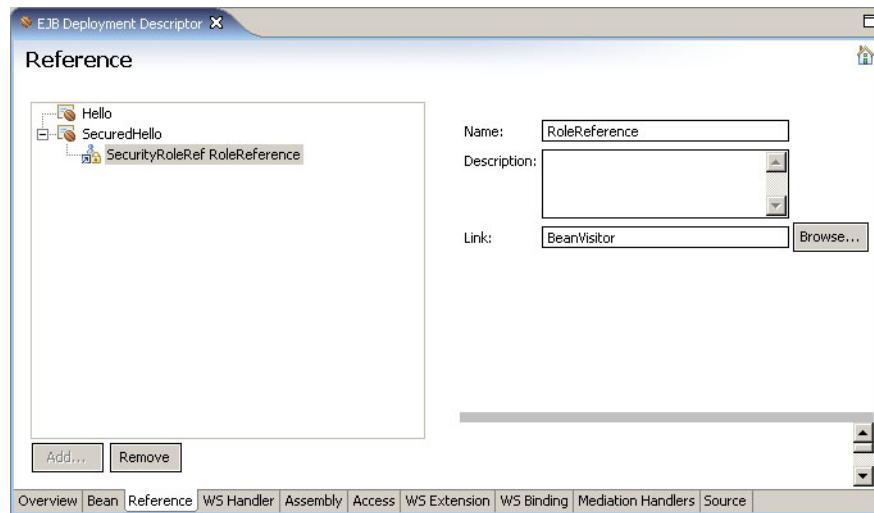


Figure 8-6 Reference tab of EJB deployment descriptor

8.2.3 Configuring method access control

You can secure session and entity bean methods if you prevent access to all except members of security roles that must access those methods. The assembly descriptor tag in the application deployment descriptor file `ejb-jar.xml` includes the method permissions. Example 8-4 shows the XML elements allowing members of the `BeanVisitor` role to call all methods in the `SecuredHello` EJB, and members of the `Anonymous` role to call all methods in the `Hello` EJB and Method permissions in the `ejb-jar.xml` file.

Example 8-4 Role and method permission definitions in the `ejb-jar.xml` file

```
<assembly-descriptor>
  <security-role>
    <description>Authenticated guest for the EJB</description>
    <role-name>BeanVisitor</role-name>
  </security-role>
  <security-role>
    <description>Anybody who access the bean</description>
    <role-name>Anonymous</role-name>
  </security-role>
  <method-permission>
    <role-name>Anonymous</role-name>
    <method>
```

```

        <ejb-name>Hello</ejb-name>
        <method-name>*</method-name>
    </method>
</method-permission>
<method-permission>
    <unchecked />
    <method>
        <ejb-name>SecuredHello</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>getMessageUnprotected</method-name>
        <method-params>
        </method-params>
    </method>
</method-permission>
<method-permission>
    <role-name>BeanVisitor</role-name>
    <method>
        <ejb-name>SecuredHello</ejb-name>
        <method-name>*</method-name>
    </method>
</method-permission>
</assembly-descriptor>

```

Assigning method permissions

To set up these method permissions by using Rational Application Developer:

1. Load the EJB project into the Rational Application Developer. In this example it is as follows:
ItsohelloEAR.ear
2. Within the J2EE perspective, expand **EJB Projects** → **ItsohelloEJB**.
3. Open the Deployment Descriptor for the itshelloEJB project. The EJB Deployment descriptor page opens. Switch to the Assembly tab.
4. In the Method Permissions section, click **Add**.

5. In the Method Permission window (Figure 8-7), select either one of the existing security roles or the **Unchecked** option. See “Assigning roles to unprotected methods” on page 184 for more details about the Unchecked option.

Choose the Security Roles option and select **BeanVisitor** role. Click **Next** to see the list of EJBs.

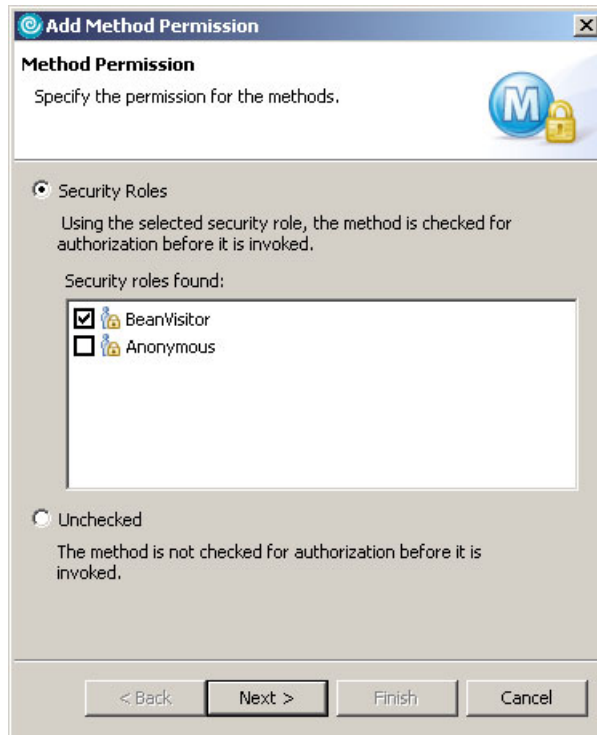


Figure 8-7 Adding method permission for defined security role

6. In the Enterprise Bean Selection window (Figure 8-8), select the EJBs on which you want to configure method permissions for the selected security role. Choose **SecuredHello** and click **Next** to see the list of methods.



Figure 8-8 Selecting EJBs for configuring method permissions

7. In the Method Elements window (Figure 8-9), select one or more methods that you want to be accessible by a selected security role. You can use the wildcards (*) if desired to include all methods of a given type or all methods for a given EJB.

In this example, we selected all methods. Thus, the *BeanVisitor* security role gets access to all SecuredHello EJB methods. For now, only the users which are mapped to the *BeanVisitor* security role have access to this EJB. Click **Finish**

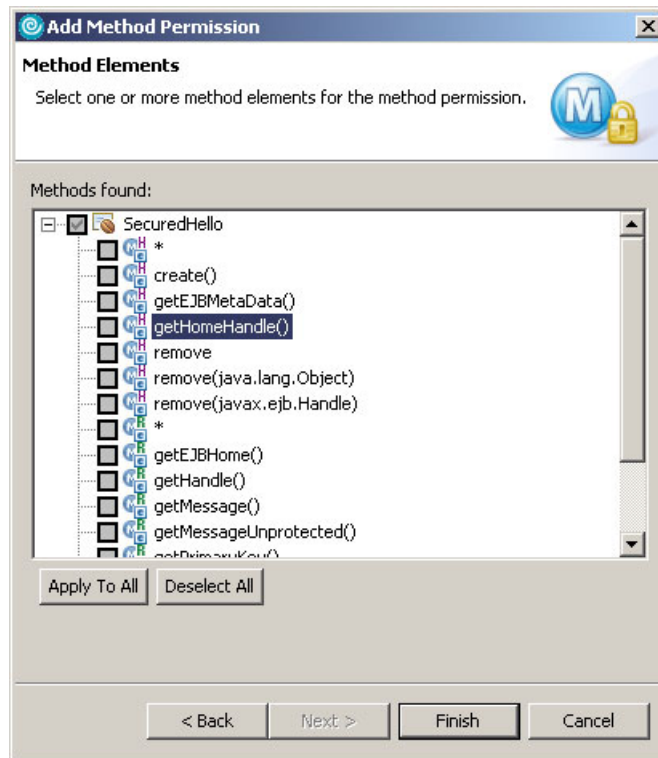


Figure 8-9 Selecting EJB methods

Assigning roles to unprotected methods

During application installation, the WebSphere Administrative Console allows you to specify what method permissions are applied to session and entity EJB methods that are not explicitly secured in the deployment descriptor. If all session and entity EJB methods are protected, this step is omitted.

Note: When assigning roles to EJB methods, methods can be specified using several types of wildcards to select all home methods, local methods, local home methods, remote methods, and so on. When installing an EJB containing methods that are protected using one method-type wildcard, for example, the home methods wildcard, but whose other methods are unprotected, the WebSphere Application Server does not prompt for how unprotected methods are to be secured. Instead, they are deselected.

One of the following permissions is applied to these unprotected methods (Figure 8-10):

- Uncheck** This is the default, and indicates that unprotected methods must be left unprotected. Anyone can call these methods.
- Exclude** Unprotected methods are unavailable to all callers.
- Role** Unprotected methods are available only to members of a specific security role.

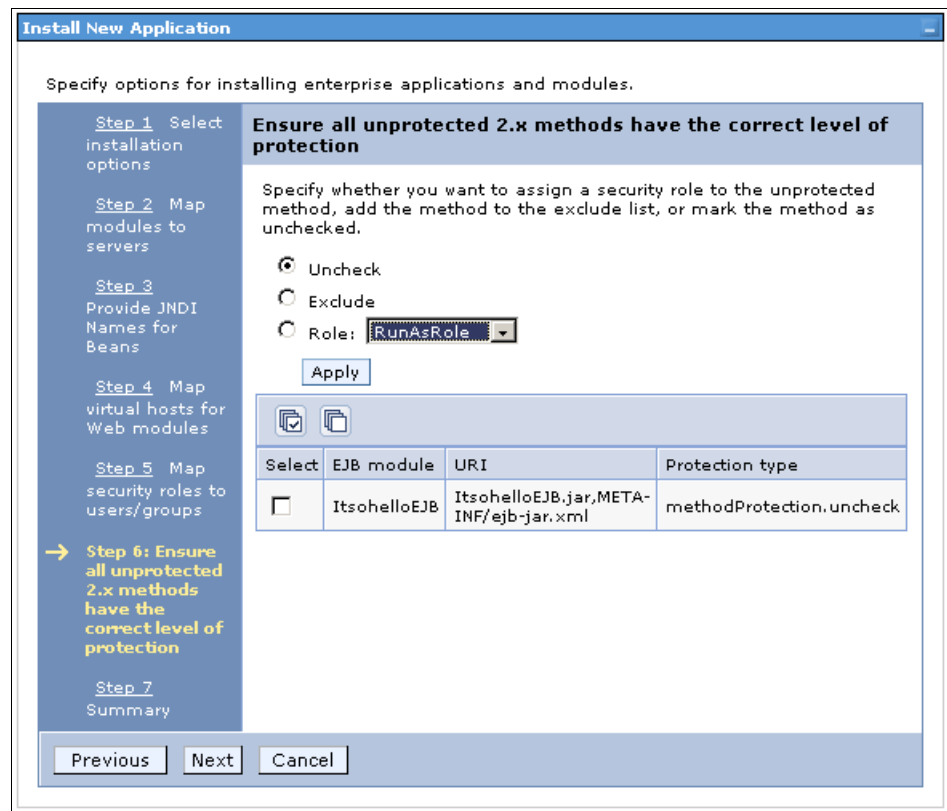


Figure 8-10 Assigning roles to unprotected methods

Note: The default behavior on EJB method protection is for methods that are not explicitly unprotected to be *unchecked*.

8.2.4 Enterprise JavaBeans Run-As delegation policy

When an EJB calls a method in another EJB, the identity of the caller of the first EJB is, by default, propagated to the next. In this way, all EJB methods in the calling chain would see the same principal if they were to call the `getCallerPrincipal()` method. Occasionally, however, it is desirable for one EJB to call another with a previously defined identity, for instance one that is a member of a specific role.

For example, consider the message-driven bean's `onMessage()` method which calls a protected method in an entity bean. Because message-driven beans' `onMessage()` methods are executed with no caller identity, this method cannot call the protected entity bean method. By delegating the `onMessage()` method to run as a specific role, and adding this role to the protected entity bean method's access permissions, the `onMessage()` method can successfully access the protected method.

Important: Although this feature is commonly referred to as the *Run-as Mode*, it does not have any noticeable effect on the bean to which it is applied. A bean configured to run as a member of a given security role actually executes using the identity of the caller. It is only when calling methods in other EJBs that the run-as mode applies. These methods are called using the delegated identity.

8.2.5 Bean-level delegation

The EJB 2.x specification defines delegation at the EJB level using the `<run-as>` element which allows the application assembler to delegate all methods of a given bean to run as a member of a specific security role. At deployment time, a real user that is a member of the specified role must be mapped to this role, through a process which is called *run-as role mapping*. All calls to other EJBs made by the delegated bean are called using the identity of this mapped user.

Figure 8-11 on page 187 shows EJB delegation in contrast to the default Run-As Caller mode. In the top scenario, the identity of the caller, *caller01*, is propagated from EJB1 to EJB2. In the bottom scenario, EJB1 is delegated to run as *role01*. During run-as mapping, another user, *caller02*, is mapped to *role01*, and therefore it is effectively *caller02* that calls EJB2. If, in the bottom scenario, EJB2 were to call EJB3, EJB3 would also appear to have been called by *caller02*.

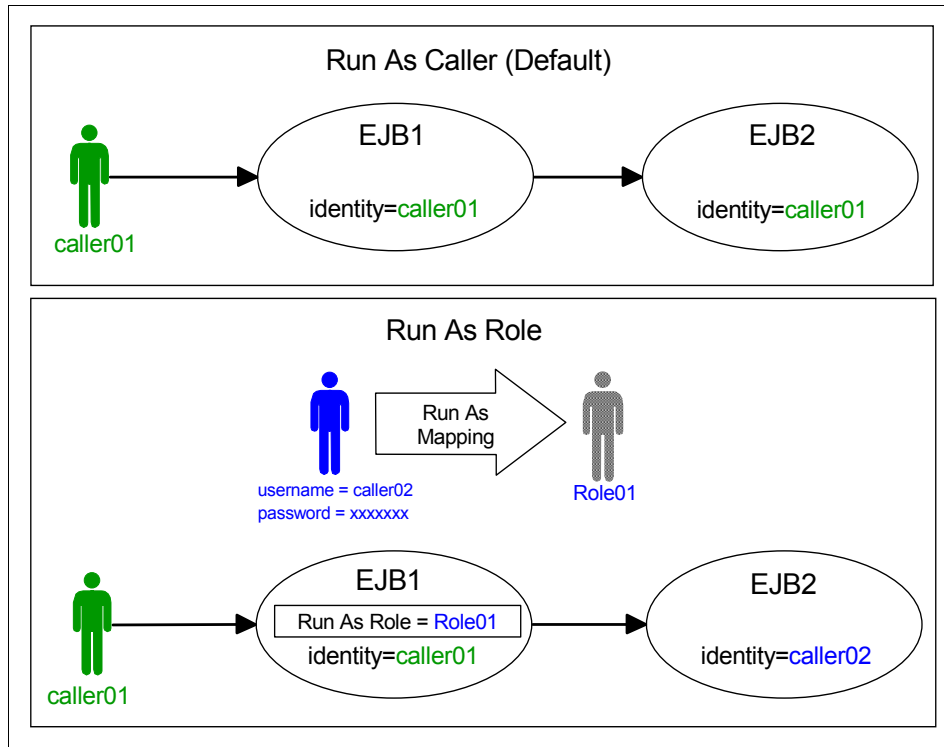


Figure 8-11 Run as Caller versus Run as Role

Example 8-5 shows the XML code in the `ejb-jar.xml` deployment descriptor file for the default mode (run as caller).

Example 8-5 `ejb-jar.xml` code for non-delegated EJB

```
<security-identity>
  <description>This bean requires no delegation</description>
  <use-caller-identity />
</security-identity>
```

Example 8-6 shows the XML code in the `ejb-jar.xml` file for a bean which has been delegated to run as a member of the `RunAsRole` security role.

Example 8-6 `ejb-jar.xml` code for EJB delegated to run as role `mdbuser`

```
<security-identity>
  <description>This EJB calls protected methods in other
EJBs.</description>
  <run-as>
```

```
<description>Methods of this EJB run as the RunAsRole
role</description>
  <role-name>RunAsRole</role-name>
</run-as>
</security-identity>
```

Assigning bean-level run-as delegation policies

To assign a bean-level run-as role to an EJB using Rational Application Developer:

1. Within the J2EE perspective, expand **EJB Projects** → **ItsohelloEJB**.
2. Open the Deployment Descriptor for the ItsohelloEJB module. The EJB Deployment descriptor page opens. Switch to the Access tab.
3. In the *Security Identity (Bean Level)* section, click **Add**.
4. In the Security Identity window (Figure 8-12 on page 189), select the desired run-as mode, you can select either of the following two:

- Use identity of caller

If you select this option, the called EJB which you are calling from your bean is called under your bean's identity. This applies to all the methods in the called bean.

- Use identity assigned to specific role (below)

If you select this option, the called EJB which you are calling from your bean is called under the specified role identity. This applies to all the methods in the called bean.

Select the desired role from the options list. This list contains all security roles which have been defined in the EJB module.

For example, select the specific role **Anonymous**. Enter an optional Role description and an optional Security identity description.

Click **Next**.

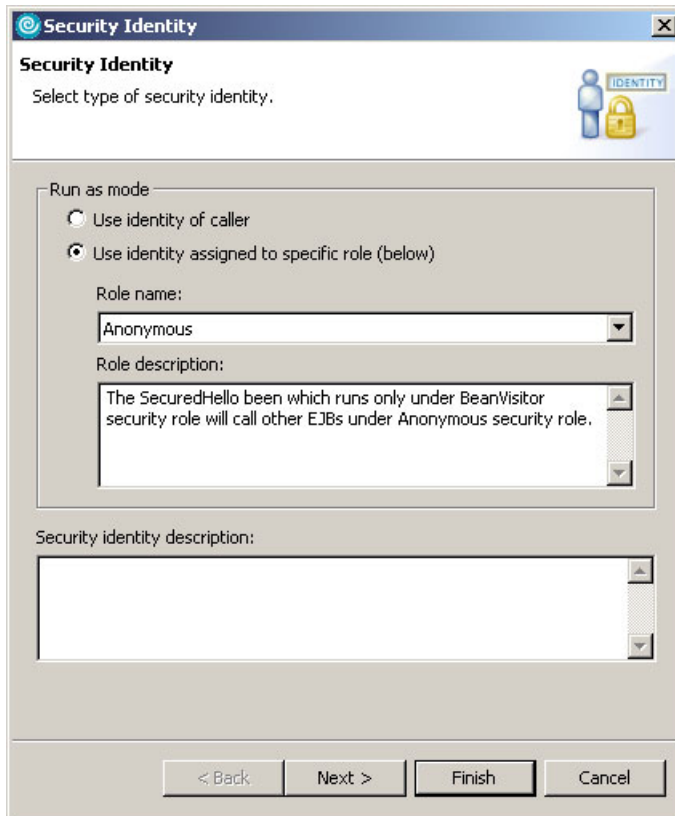


Figure 8-12 Assigning bean level run as delegation policy

5. Select one or more beans that must use this delegation policy. In this example, select **SecuredHello**.
6. Click **Finish**. Save and close the deployment descriptor.

8.2.6 Method-level delegation

In addition to the bean-level delegation policy defined by the EJB 2.x specification and described in the previous section, WebSphere Application Server provides additional capabilities to perform method-level EJB delegation as shown in Figure 8-13. This works in the same way as bean-level delegation, but can be applied to specific EJB methods, rather than to the bean as a whole. This finer degree of delegation granularity allows application assemblers to delegate different methods of the same EJB to different security roles.

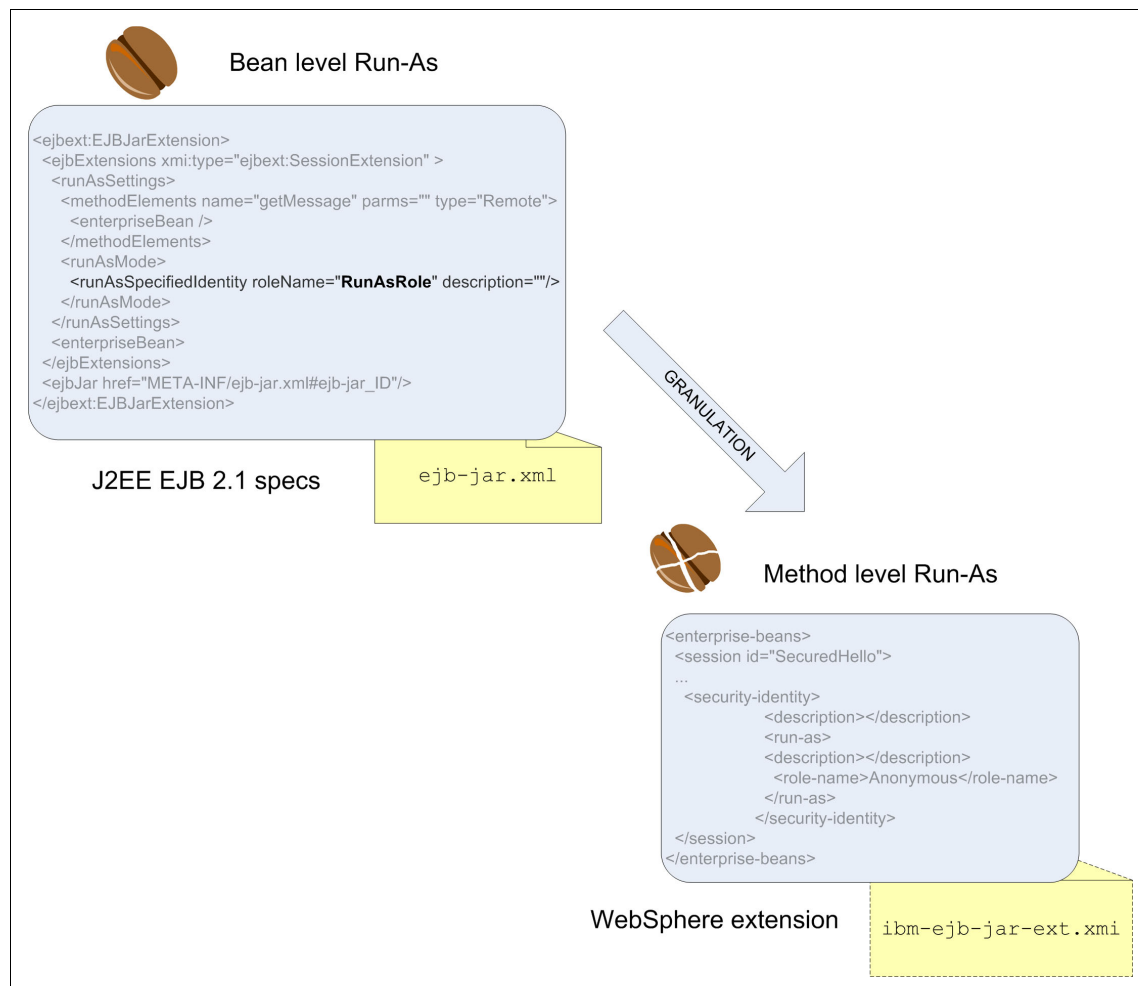


Figure 8-13 Method level Run-As delegation compared to Bean level Run-As delegation

In addition, method-level delegation provides an additional delegation option, which is called *run as server*. This option indicates that the method must make calls to other EJBs using the identity of the Application Server itself.

Method-level delegation policies are defined in the `ibm-ejb-jar-ext.xml` file. Example 8-7 shows the XML code for a `getMessage()` method which is delegated to run as the Application Server.

Example 8-7 Method-level run as server

```
<runAsSettings description="">
  <methodElements name="getMessage" parms="" type="Remote">
    <enterpriseBean xmi:type="ejb:Session"
href="META-INF/ejb-jar.xml#Hello"/>
  </methodElements>
  <runAsMode xmi:type="ejbext:UseSystemIdentity"/>
</runAsSettings>
```

Example 8-8 shows the XML code for a `getMessage()` method which is delegated to run as a member of the *RunAsRole* security role.

Example 8-8 Method-level run as role

```
<runAsSettings>
  <methodElements name="getMessage" parms="" type="Remote">
    <enterpriseBean xmi:type="ejb:Session"
href="META-INF/ejb-jar.xml#Hello"/>
  </methodElements>
  <runAsMode>
    <runAsSpecifiedIdentity roleName="RunAsRole" description=""/>
  </runAsMode>
</runAsSettings>
```

Assigning method-level run-as delegation policies

To assign a method-level run-as role to an EJB using Rational Application Developer:

1. Within the J2EE perspective, expand **EJB Projects** → **ItsohelloEJB**.
2. Open the Deployment Descriptor of the ItsohelloEJB module. The EJB Deployment descriptor page opens. Switch to the Access tab.
3. Scroll down to *Security Identity (Method Level)* section and click **Add**.
4. In the Add Security Identity window (Figure 8-14), select one of the following run-as modes and click **Next**:

- Use identity of caller

If you select this option, the called EJB methods which you are calling from your bean is called under your bean's identity. This applies just to the selected methods.

- Use identity of EJB server

If you select this option, the called EJB methods which you are calling from your bean is called under EJB server identity. This applies just to the selected methods.

- Use identity assigned to specific role (below)

If you select this option, the called EJB methods that you call from your bean are called under a specified role identity. This applies to the selected methods. Select the desired role from the option list. The specify role list contains all security roles that were defined in the EJB module. If you choose this option, enter an optional Role description and an optional Security identity description.

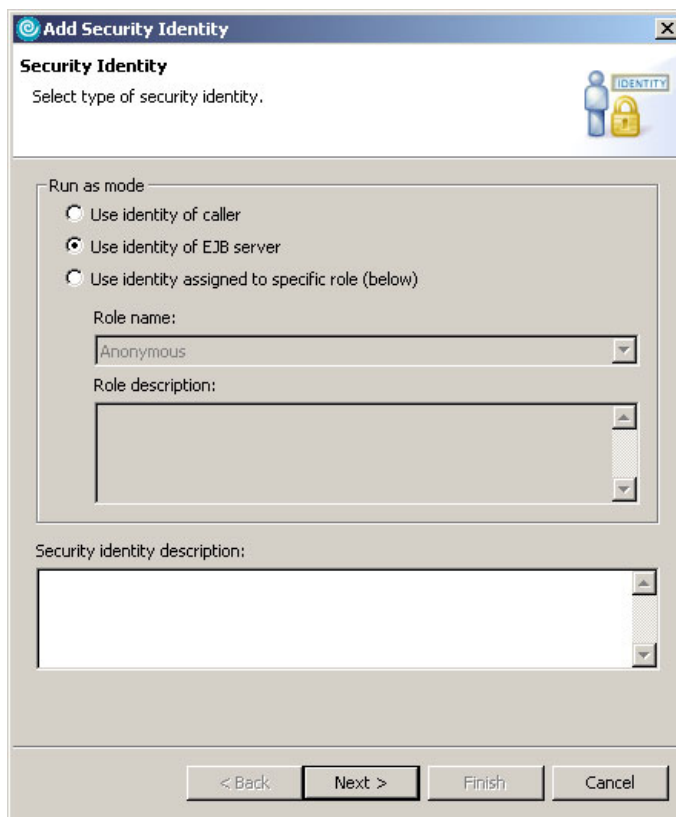


Figure 8-14 Method-level run-as role policy in Rational Application Developer

5. In the Enterprise Bean Selection dialog, select the EJBs that contain the methods to which you want to assign this delegation policy. Then click **Next**.
6. In the Method Elements window (Figure 8-15), select the EJB methods to which this delegation policy must be assigned. Click **Finish**.

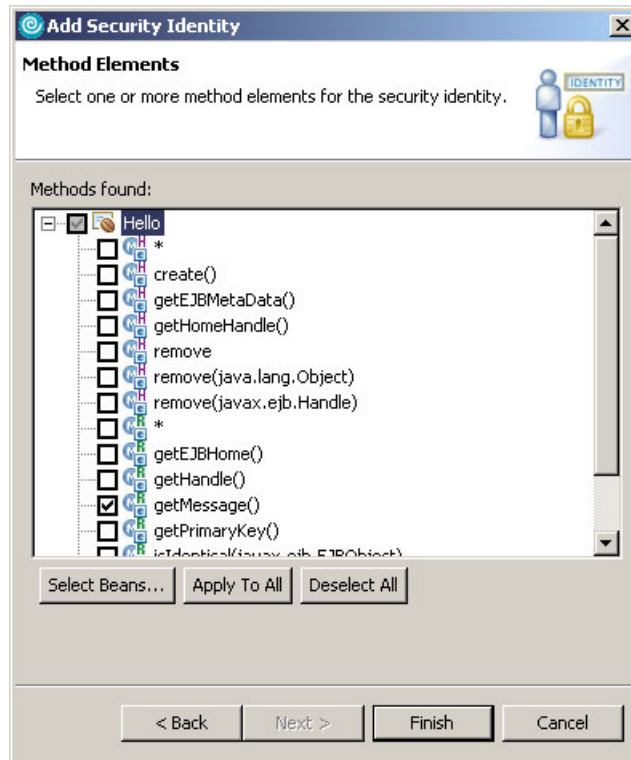


Figure 8-15 Method Elements selection window

7. Save the deployment descriptor changes.

8.2.7 Run-as mapping

Run-as mapping refers to the process of assigning a real user from the user registry that is a member of the specified security role to the bean-level or method-level delegation policies. Run-as mapping is different from, but easily confused with, security role mapping.

Table 8-1 compares run-as mapping and security role mapping.

Table 8-1 Run-as mapping versus security role mapping

Run-as mapping	Security role mapping
Run-as mapping is used to determine the principal from the user registry that is used as the caller identity when a delegated EJB makes calls.	Security role mapping is used to determine the users and groups from the user registry that are considered members of the security role.
Run-as mapping associates a single user that is a member of the specified security role with a delegation policy.	Security role mapping associates one or more users or groups with a security role.
A single user name and password for the mapped identity is stored in the deployment descriptor.	One or more user names and/or group names are stored in the deployment descriptor.
Authentication done at installation time.	Authentication done at run time.
Run-as mapping is performed using the WebSphere Administrative Console only.	Security role mapping is performed using the Application Server Toolkit, the WebSphere Studio, or the WebSphere Administrative Console.
Cannot be modified after application installation.	Can be modified after application installation using the WebSphere Administrative Console.

Important: The *Map RunAs roles to users* option opens in the WebSphere Administrative Console interface only when your application uses run-as delegation. During the enterprise application installation, WebSphere detects whether this configuration exists at all and changes the user interface accordingly. The same is true for the *Correct use of system identity* option and applications that delegate the EJB server identity for the method security role.

When installing an application that defines either a bean-level or method-level run-as role delegation policy, one of the steps is to map the run-as roles to a real user, as shown in Figure 8-16 on page 195:

1. Select the Role that you want to map.
2. Enter a valid user name and password of a user in the registry that is a member of the specified security role.
3. Click **Apply** to authenticate the user and associate that identity with the run-as role policy.

- After all run-as roles are mapped to real users, click **Next** to continue the installation.

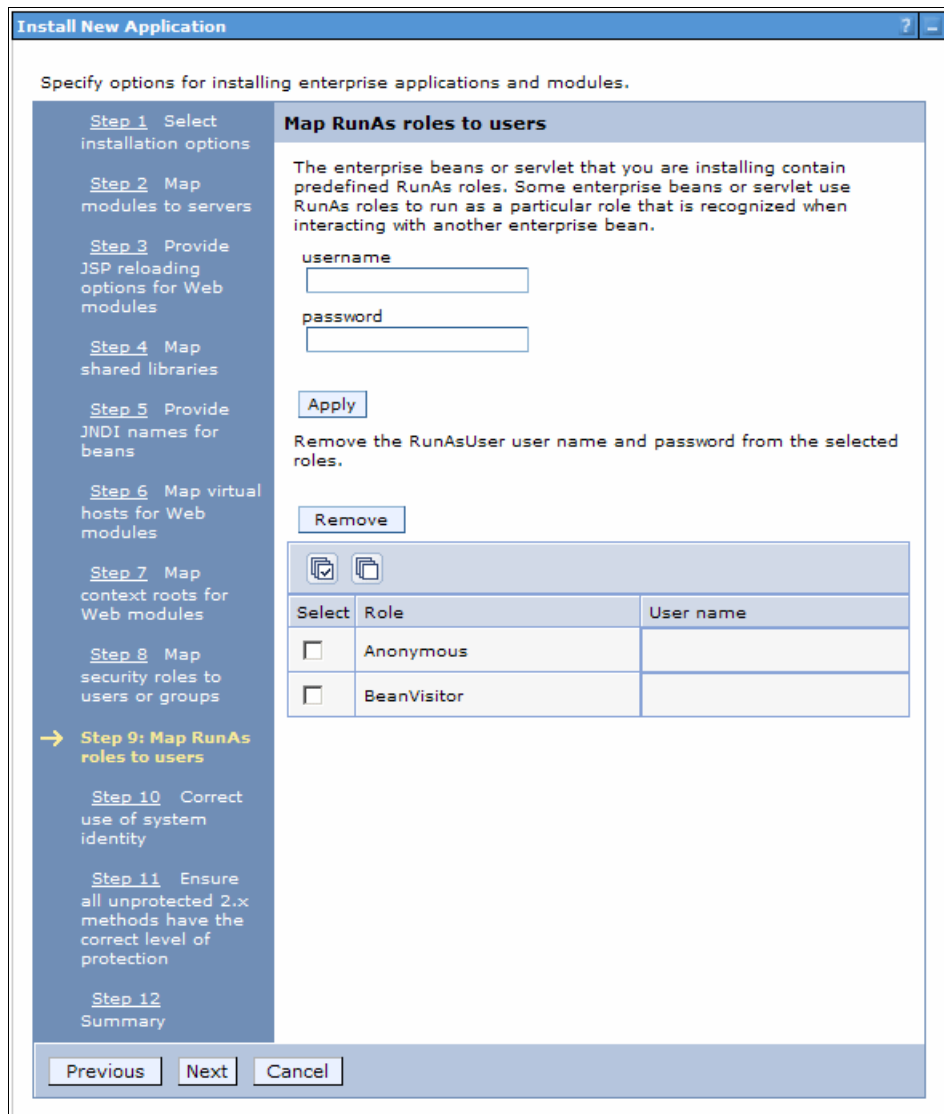


Figure 8-16 Run-as role mapping in WebSphere Application Server Version 6

If one or more method-level delegation policies specify the run-as system, one of the installation steps is going to be to verify this policy. The dialog opens as shown in Figure 8-16, and for each method that specifies the run-as system, the application deployer can do one of the following:

- ▶ Do nothing, and allow the method to make calls using the system identity.
- ▶ Assign the method a run-as role, and map the role to a user from the registry.

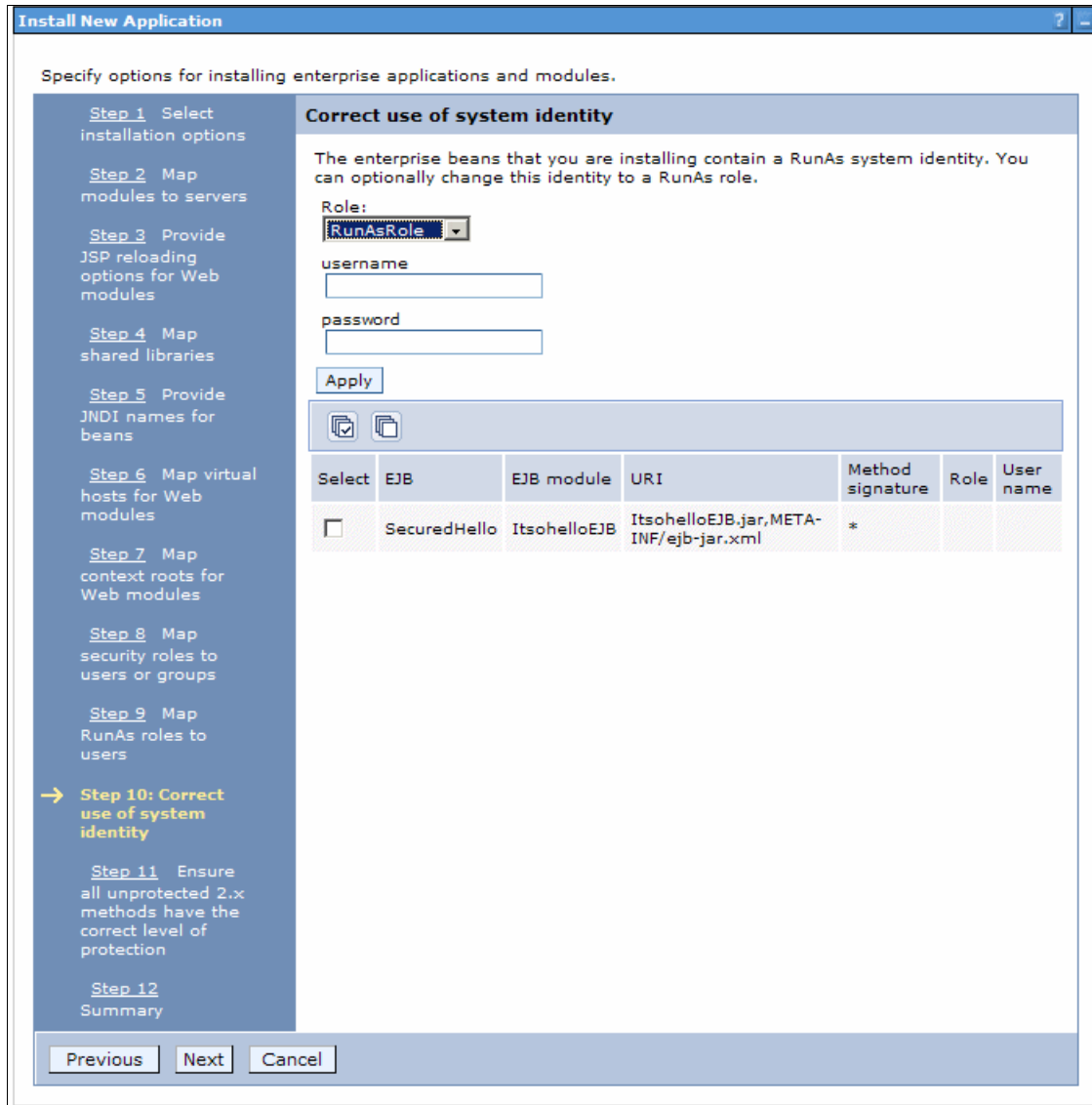


Figure 8-17 Verifying the run-as system

To override the run-as system mapping and assign a run-as role as shown in Figure 8-17 on page 196:

1. Select the methods to which you want to assign the run-as role.
2. Select the desired Role from the drop-down list of defined security roles. See Figure 8-17.
3. Enter the valid user name and password of a user in the registry that is a member of the specified security role.
4. Click **Apply** to authenticate the user and associate that identity with the run-as role policy.
5. Click **Next** to continue with the installation.

8.3 Programmatic J2EE security

Security-aware applications can use programmatic security when declarative security alone is not sufficient to express the security model of the application.

Programmatic security becomes useful when the Application Server provides a security infrastructure that cannot supply all the functionality required for the application. Using the Java APIs for security, developers can implement security for the whole application without using the Application Server security functions at all. Programmatic security also gives developers the option to implement dynamic security rules for your applications.

Having said that, when developing servlets and EJBs, there are a few security calls available if the developer wants greater control of what the end user is allowed to do than is provided by the infrastructure.

EJB security methods

The EJB 2.x specification defines two methods that allow programmatic access to the caller's security context, `javax.ejb.EJBContext`.

- ▶ `java.security.Principal` `getCallerPrincipal()`

The *getCallerPrincipal* method allows the developer to get the name of the current caller. To do this, call `getName()` on the `java.security.Principal` object returned. See Example 8-9.

Example 8-9 The getCallerPrincipal method

```
EJBContext ejbContext;  
...  
// get the caller principal
```

```
java.security.Principal callerPrincipal =
ejbContext.getCallerPrincipal();
// get the caller's name
String callerName = callerPrincipal.getName();
```

The `Principal.getName()` method returns the login name of the user.

► **Boolean `isCallerInRole(String roleName)`**

The *isCallerInRole* method allows the developer to make additional checks on the authorization rights of a user which are not possible, or more difficult, to perform through the deployment descriptor of the EJB. See Example 8-10.

Example 8-10 The isCallerInRole method

```
EJBContext ejbContext;
...
if (ejbContext.isCallerInRole(" "))
// Perform some function
else
// Throw a security exception
```

The `isCallerInRole(String role)` method returns `true` if the user is in the specified role, and `false` if it is not. The role name specified in the method is really a security role reference, not a role. If the security role reference is not defined for the EJB, the method returns `null`.

Sample usage of security methods

Example 8-11 is a code snippet from the `SecuredHelloBean` as part of the `ItsHelloEAR` application. For more details, check the original sample application.

Example 8-11 Sample code using the EJB security methods

```
public String getMessageUnprotected() {
    return "[Not protected] Hello to you " +
mySessionCtx.getCallerPrincipal();
}

public String isInRole() {
    if (mySessionCtx.isCallerInRole("RoleReference")) {
        return "You are a member of the referenced role";
    } else {
        return "You are NOT a member of the referenced role";
    }
}
}
```

With the security methods, the EJB does not let the user in a restricted role submit a transfer greater than the maximum transferable amount.

8.4 EJB container access security

The previous sections focused on EJB application security from the J2EE layer perspective. There are some more authentication and transport protection security mechanisms implemented by WebSphere working on the lower, Common Object Request Broker Architecture (CORBA), messaging layer.

8.4.1 CSIV2 and Secure Authentication Service

When a client component uses services from the WebSphere EJB container, all the communication go through the Remote Method Invocation over Internet InterORB Protocol (RMI/IIOP). The client component can either be a stand-alone Java client, a J2EE client container application, or another EJB container. See Figure 8-18 on page 200.

WebSphere provides a security service which is compliant with Common Security Interoperability Version 2, the CSIV2 protocol. There is another service called IBM Security Authentication Service which has been used in previous versions before CSIV2. Secure Authentication Service (IBM) is deprecated and it is only kept to provide interoperability with WebSphere versions later than V5.0 and is not displayed in the administration console unless a V6.0 or later server is federated into the cell.

In short, providing Common Security Interoperability, WebSphere basically provides two important services:

- ▶ *Authentication* capabilities on the CORBA level.
- ▶ *Transport channel encryption*. WebSphere provides IIOP transport channel protection using the Secure Sockets Layer (SSL) protocol.

For more details about CSIV2, see “CSIV2 Security Attribute Service” on page 217.

Figure 8-18 shows a simple scenario. A J2EE client application has to invoke some methods in an EJB which runs in Server A. Furthermore, Server A must run some methods in EJBs that run in Server B.

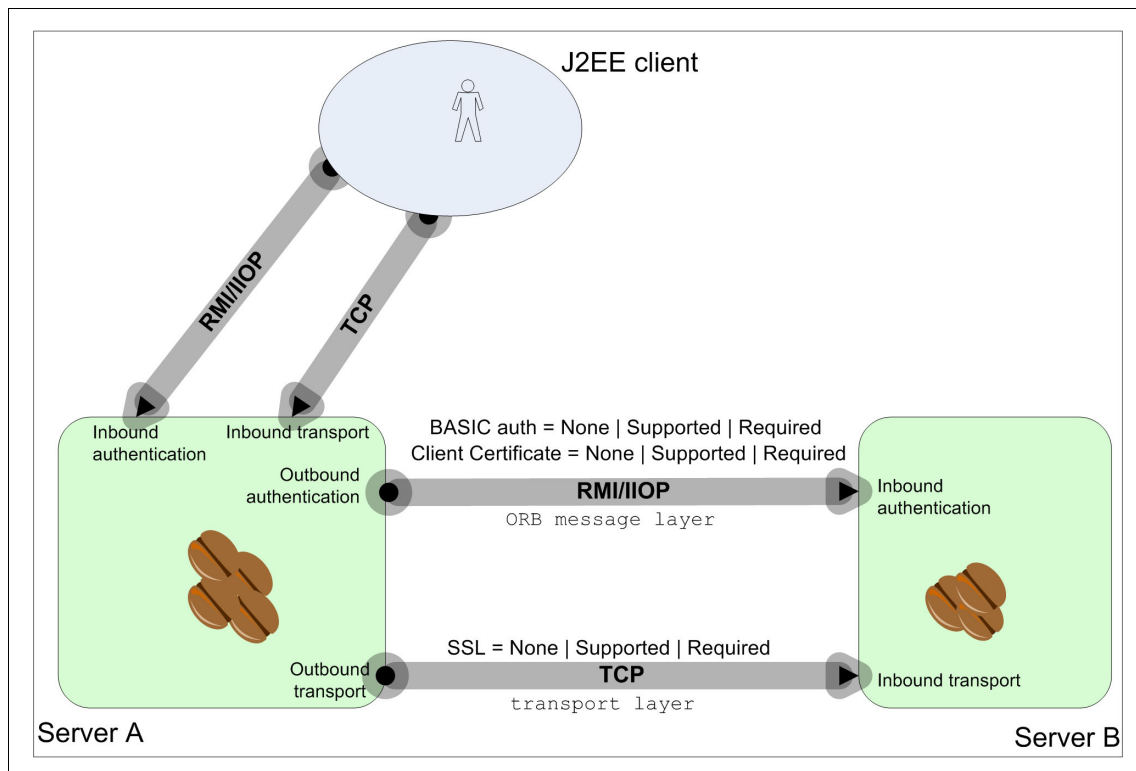


Figure 8-18 CSIV2 configurational options

8.4.2 Container authentication

When invoking EJB methods, the WebSphere Application Server environment determines the type of authentication required between the client and the server for each request. The following options are available:

- Basic authentication

In this case, plain user ID and password information is passed from client to server through the CORBA message layer.

For more details, see the *Message Layer authentication* document in the WebSphere Application Server 6.1 Information Center on the Web at:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp>

► Client certificate authentication

The client certificate authentication does not occur at the message layer as in the previous case, but occurs during the connection handshake using SSL certificates.

For more details, see *Scenario 3: Client certificate authentication and RunAs system* document in the WebSphere Application Server 6.1 Information Center on the Web at:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp>

Because basic authentication and client certificate authentication occur at a different level, they can be set independently. For example, you can have both authentications to be required or just basic authentication set and the other type supported.

Configuring container authentication

The configuration of EJB container authentication can be done through the WebSphere Administrative Console:

1. On the Secure administration, applications, and infrastructure page, in the Authentication section, click **RMI/IIOP Security** to display all the available options (Figure 8-19).
2. Set container authentication for either *inbound* or *outbound* requests independently. Inbound means all the incoming communication that comes from a client to the server, outbound means all the outgoing communication that goes from the server toward other servers. Click either the **CSIV2 inbound authentication** or **CSIV2 outbound authentication** page link.

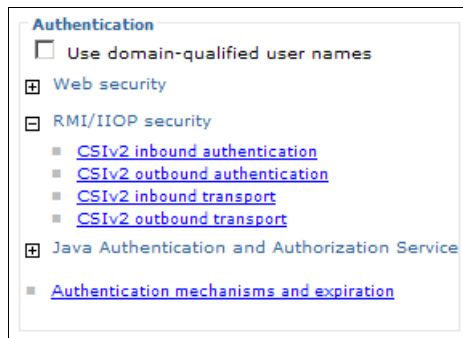


Figure 8-19 The Authentication section of the security administration page

3. On the Configuration page (Figure 8-20 on page 203):
 - a. Set Basic Authentication to one of the following options:

Never	Indicates that the server is not configured to accept message layer authentication from any client.
Supported	Indicates that this server accepts basic authentication. However, other methods of authentication can occur (if configured) and anonymous requests are accepted.
Required	Indicates that only clients configured to authenticate to this server through the message layer can invoke requests on the server.
 - b. Set *Client certificate authentication* to one of the following options:

Never	Indicates that the server is not configured to accept client certificate authentication from any client.
Supported	Indicates that the server accepts SSL client certificate authentication. However, other methods of authentication can occur (if configured) and anonymous requests are accepted.
Required	Indicate that only clients that are configured to authenticate to the server through SSL client certificates can invoke requests on the server.
 - c. To enable client certificate authentication for the IIOP transport layer, set the SSL to be **required** or **supported** (required is the more secure option). The prerequisite on the client side is that the client must have set a key database with a client certificate. As always, the certificate can be signed by a known Certificate Authority. Using an imported self-signed public key from the client is also an option, although we do *not* recommend it.

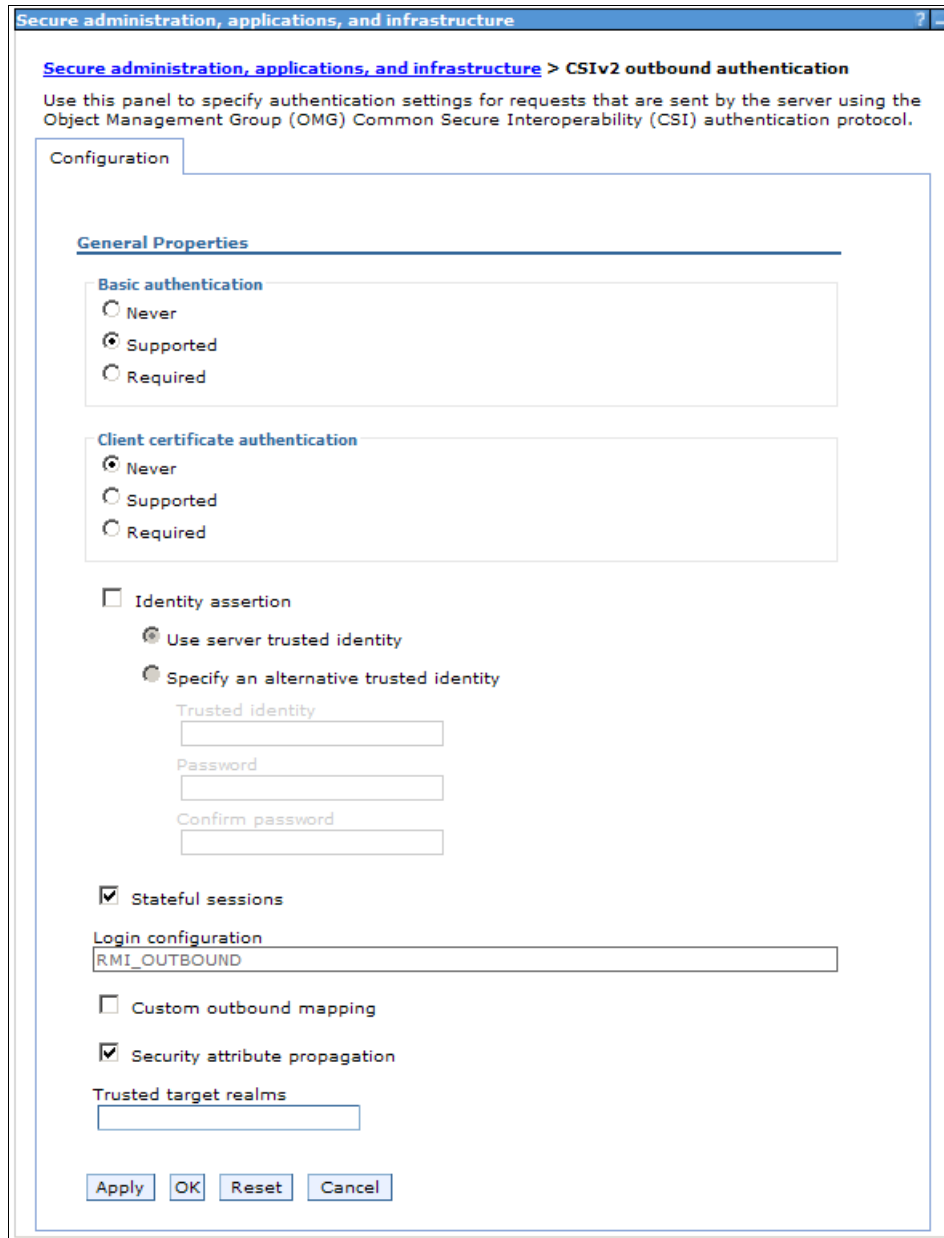


Figure 8-20 Setting CSIV2 outbound authentication properties

8.4.3 RMI/IOP transport channel protection

When accessing EJB services, the client and server communicate through the Object Request Broker (ORB) service, using the IOP protocol. Prior to any request flowing, a connection is established between the client ORB and the server ORB over the Transmission Control Protocol (TCP) transport. WebSphere provides the option of encrypting the connection using SSL.

According to the connection encryption policies of both the client and the server, they negotiate the level of security for the connection used for the IOP communication.

Configuring IOP transport channel protection

The configuration of IOP transport channel protection can be done using the WebSphere Administrative Console:

1. In the Secure administration, applications, and infrastructure page, go to the Authentication section.
2. Select **RMI/IOP Security** to display all the available options as shown in Figure 8-19 on page 201.
3. Set transport channel protection for either *inbound* or *outbound* transport independently. Select the **CSIV2 inbound transport** or **CSIV2 outbound transport** page link.
4. On the page that opens (Figure 8-21 on page 205):
 - a. Set *Transport* to one the following options:

Transmission Control Protocol/Internet Protocol (TCP/IP)

Server only supports TCP/IP and cannot accept SSL connections.

SSL supported

Server can support either TCP/IP or SSL connections.

SSL required

Any client communicating with this server must use SSL.

Note: Configure both the inbound and outbound CSIV2 transports with *SSL required* in a secured environment. By default, WebSphere negotiates a mutually acceptable level of transport security. However, if a client requests a non-SSL connect, unless SSL required is configured, a non-secure connection is established.

- b. Configure *SSL Settings* by selecting either **Centrally managed** or **Use specific SSL alias**. If you select **Use specific SSL alias**, select one of the

defined SSL configurations from the drop-down list. For more information about SSL configuration, see , “WebSphere Application Server uses the Secure Sockets Layer (SSL) protocol to provide Transport Layer Security (TLS), which allows for secure communication between a client and application server. The SSL configuration options in WebSphere offer full end-to-end management, including certificate management, individual endpoint SSL mappings, and scoped association of SSL configurations and key stores.” on page 69.

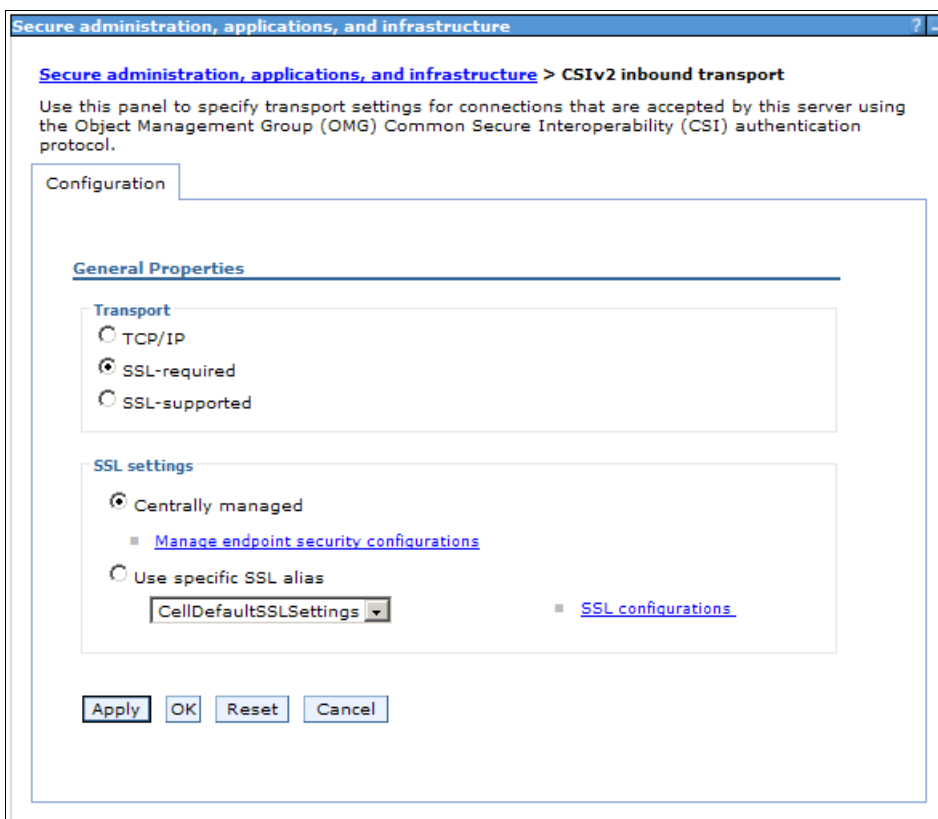


Figure 8-21 Setting CSIV2 Inbound transport properties

By default, the ORB transport listener ports are dynamically allocated during run time. You might consider fixing the listener ports used for CSIV2. Because each Application Server runs its own ORB, they all have their own set of listening ports. The listener ports are managed by changing the Application Server's endpoints. In this case, specify the following endpoints to fix the port numbers:

```
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS  
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS  
ORB_LISTENER_PORT
```

Check the WebSphere Application Server V6.1 Information Center for further details about how to configure the endpoints.



Client security

This chapter discusses client security in WebSphere Application Server V6.1.

9.1 Application clients in WebSphere

A *client* is a generic term that refers to the process typically responsible for requesting a service. The service is provided by the *server*. This chapter discusses Java-based specific application clients that access a remote enterprise bean server.

WebSphere Application Server V6.1 supports several important models of Java application clients, such as the following examples:

- ▶ Java 2 Platform, Enterprise Edition (J2EE) application client
This client uses the Java Remote Method Invocation over Internet InterORB Protocol (RMI/IIOP) to access enterprise bean references and to use the Common Object Request Broker Architecture (CORBA) service provided by the J2EE platform implementation. This client also provides initialization of the runtime environment for the client application. The J2EE application client run time also provides support for security authentication to the enterprise beans and local resources.
- ▶ Thin application client
This client is a lightweight, downloadable Java application run time that is capable of interacting with enterprise beans. WebSphere Application Server V6.1 supports the pluggable client. The thin application client uses the RMI/IIOP to access enterprise bean references and CORBA references and allows the client application to use any supported CORBA services. WebSphere Application Server V6.1 supports the thin client in a pluggable client.
- ▶ Pluggable application client
This client is a kind of thin application client that uses a Sun Java™ Runtime Environment (JRE™) instead of the JRE that IBM provides.
- ▶ Applet application client
This client is the applet that a client provides a browser-based Java run time access to the enterprise bean directly. The client accesses the enterprise beans by considering the enterprise bean object reference as CORBA object references.
- ▶ ActiveX® application client
- ▶ This client refers to when WebSphere Application Server V6.1 provides an Active X to Enterprise JavaBeans (EJB) bridge that enables Active X programs to access enterprise beans through a set of Active X automation objects. The bridge accomplishes this access by loading the Java virtual machine (JVM™) into an Active X automation container such as Visual Basic®, VBScript, and Active Server Pages. The Active X to EJB bridge uses

the Java Native Interface (JNI™) architecture to programmatically access the JVM.

You can find further extensive information regarding the application clients and their capabilities in the WebSphere Application Server Information Center. This chapter shows how the J2EE and thin clients access enterprise beans resources.

9.1.1 Developing and securing the J2EE application client

To develop and secure the J2EE application client:

1. Create an instance of the object that you want to access from the remote server.
2. Specify the user ID and password on the connection method when you create a connection to the server. You must enable security.
3. Assemble the application client enterprise archive (EAR) file by using an assembly tool, such as the Application Server Toolkit, Rational Application Developer. Assemble the application client .ear file on any development machine where the assembly tool is installed.
4. Add the resources to the client deployment descriptor by completing the binding Java Naming and Directory Interface (JNDI) name for the resources object on the server.
5. Distribute the configured .ear file to the client machines.
6. Deploy the application client.
7. Configure the application client resources.
8. Run the application client.

The J2EE application client supports the client container that runs stand-alone Java applications and provides J2EE services to the applications. J2EE services include naming, security, and resource connection.

To launch J2EE application clients, use the launchClient tool. See the WebSphere Application Server V6.1 Reference Document for more information.

9.1.2 Deploying an application client by using the Java Web Start tool

Java Web Start is an application deployment technology that includes the portability of the applets, the maintainability of servlets, and JavaServer Pages (JSP) file technology, Extensible Markup Language (XML), and Hypertext Markup Language (HTML). The Java Web Start client is used with a platform that supports a Web browser.

Java Web Start is built from the J2EE infrastructure. The technology inherits the complete security architecture of the J2EE platform. Java Web Start uses the Java Network Launching Protocols (JNLP) and API. The JNLP client reads and parses a JNLP descriptor file (JNLP file). Based on the JNLP descriptor, it downloads appropriate pieces of a client application and any of its dependencies. After downloading and caching the client application, Java Web Start launches it natively on the client machine.

Java Web Start on the Java 2 Standard Edition (J2SE™) developer kits that IBM provides is packaged in Application Client for WebSphere Application Server V6.1.

To deploy an application client by using the Java Web Start tool:

1. In an assembly tool, import your Web archive (WAR) file or an EAR file that contains one or more Web modules:
 - a. In the Project Explorer folder, locate your application.
 - b. Right-click the deployment descriptor and select **Open with** → **Deployment Descriptor Editor**.
 - c. In the Deployment Descriptor window, to see online information about the editor, press F1 and click the editor name. If you select a WAR file, a Web deployment descriptor editor opens. If you select an EAR file, an application deployment descriptor editor opens.
2. Create security roles either at the application level or at the Web module level. If a security role is created at the Web Module level, the role also is displayed in the application level. If a security role is created at the application level, the role is not displayed in all of the Web modules. You can copy and paste a security role at the application level to one or more Web module security roles.
 - To create a role at the Web-module level:
 - i. In a Web deployment descriptor editor, click the **Security** tab.
 - ii. Under Security Roles, click **Add**.
 - iii. Enter the Security role name, describe the security role, and click **Finish**.
 - To create a role at the application level:
 - i. In an application deployment descriptor editor, click the **Security** tab.
 - ii. Under the list of security roles, click **Add**.
 - iii. In the Add Security Role wizard, name and describe the security role and then click **Finish**.

3. Create *security constraints*, which are a mapping of one or more Web resources to a set of roles.

On the **Security** tab of a Web deployment descriptor editor, click **Security Constraints**. On the **Security Constraints** page, you can perform the following tasks:

- Add or remove security constraints for specific security roles.
- Add or remove Web resources and their HTTP methods.
- Define which security roles are authorized to access the Web resources.
- On the user data, specify one of the following types of constraints:

None	The application does not require transport guarantees.
Integral	Data cannot be changed in transit between the client and the server.
Confidential	Data content cannot be observed while it is in transit.

Integral and Confidential constraints usually require the use of Secure Sockets Layer (SSL).

- a. Under Security Constraints, click **Add**.
- b. Under Constraints name, specify a display name for the security constraint and click **Next**.
- c. Type a name and description for the Web resource collection.
- d. Select one or more of the following HTTP methods:
 - GET
 - PUT
 - HEAD
 - TRACE
 - POST
 - DELETE
 - OPTIONS
- e. Next to the Pattern field, click **Add**.
- f. Specify a URL pattern. The 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. Wild card (*.*,*.jsp) URL pattern matching is used last.
- g. Click **Finish**.
- h. Repeat steps a through g to create multiple security constraints.

4. Map security-role-ref and role-name elements to the role-link element. During the development of a Web application, you can create the security-role-ref element. The security-role-ref element contains only the role-name field. The role-name field contains the name of the role that is referenced in the servlet of the JSP code to determine if the caller is in a specified role.

Because security roles are created during the assembly stage, the developer uses a logical role name in the role-name field and provides enough description in the Description field for the assembler to map the actual role. 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. Click the **References** tab of a Web deployment descriptor editor. On the References tab, you can add or remove the name of an enterprise bean reference to the deployment descriptor:
 - EJB reference
 - Service reference
 - Resource reference
 - Message destination reference
 - Security role reference
 - Resource environment reference
 - b. Under the list of Enterprise JavaBeans references, click **Add**.
 - c. In the Name and Ref Type fields, specify a name and a type for the reference.
 - d. Select **Enterprise Beans in the workplace** or **Enterprise Beans not in the workPlace**.
 - e. Optional: If you select Enterprise Beans not in the workplace, in the Type field, select the type of the enterprise bean. You can specify either an entity bean or a session bean.
 - f. Optional: Click **Browse** to specify values for the local home and local interface in the Local home and Local fields. Then click **Next**.
 - g. Map every role name that is used during development to the role by using the previous steps. Every role name that is used during development maps to the actual role.
5. Specify the RunAs identity for Servlets and JSP files. The *RunAs identity* of a servlet is used to invoke enterprise bean from within the servlet code. When enterprise beans are invoked, the RunAs identity is passed to the enterprise bean 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. On the **Servlets** tab of the Web Deployment descriptor editor, under Servlets and JSP, click **Add**.
 - b. In the Add Servlet of the JSP wizard, specify the servlet or JSP file settings, including the name, initialization parameters, and URL mappings. Click **Next**.
 - c. Specify the class file destination.
 - d. Back on the **Servlets** tab, click **RunAs**, select the security role, and describe the role.
 - e. Specify a RunAs identity for each servlet and JSP file that is used for your Web application.
6. Configure the login mechanism for the Web module. This configured login mechanism applies to all the servlets, JSP files, and HTML resources in the Web module.
 - a. Click the **Pages** tab of a Web deployment descriptor editor and click **Login**. Select the required authentication method. The following methods are available:
 - Unspecified
 - Basic
 - Digest
 - Form
 - Client-Cert
 - b. Specify a realm name.
 - c. If you select the Form authentication method, select a login page and an error page Web address. For example, you might use `/login.jsp` or `/error.jsp`. The specified login and error pages are present in the `.war` file.
 - d. Install the client certificate on a browser or Web client. If the **ClientCert** is selected, place the client certificate in the server trust keying file.
 7. Close the deployment descriptor editor, and when prompted, click **Yes** to save the changes.

9.1.3 Thin application client

The thin application client run time provides the necessary support to the client application for object resolution, security, reliability, availability, and serviceability. However, this client does not support a container that provides easy access to these services. For example, no support exists for nicknames of enterprise bean or local resource resolution. When resolving to an enterprise bean (using either JNDI or CosNaming) sources, the client application must know the location of the name server and fully qualified name used when the

reference was bound to the name space. The client does not perform initialization of any of the services that the client application might require.

The WebSphere thin application client provides the implementation for various services (Security, Workload Management, CORBA objects, and CORBA-based services).

9.1.4 Itsohello client example

This chapter uses the Itsohello application as an example. Figure 9-1 shows two enterprise beans, Hello and SecuredHello, as the core of the Itsohello application. These enterprise beans are installed in a WebSphere Application Server. They are accessible from different remote clients, such as the user's browser (by using the HelloServlet servlet), four J2EE Java application clients, and four thin Java application clients.

Figure 9-1 shows an interaction diagram of the Itsohello client applications that are used for this chapter. Four J2EE application clients and four thin application clients access secure and unsecure Hello beans in the EJB container.

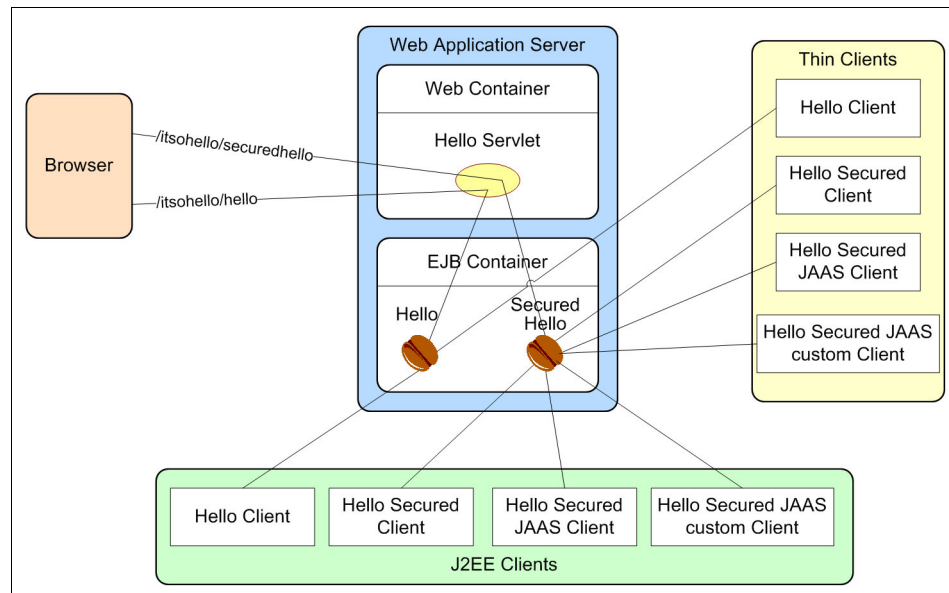


Figure 9-1 Accessing secure and unsecure Hello beans in the EJB container

The components are described as follows:

- ▶ An enterprise application `ItsHelloEAR.ear` is installed in WebSphere Application Server. This `.ear` file contains a servlet called *HelloServlet*, which accesses two simple session beans: `ejb/itshello/hello` (unsecure) and `ejb/itshello/securedhello` (secure). The latter implies that only an authenticated user allows you to access the bean. To verify the installation, access the beans by using your browser with the following HTTP addresses:

- For the unsecure session bean:

`http://<hostname>:<port>/itshello/hello`

- For the secure session bean:

`http://<hostname>:<port>/itshello/securedhello`

The default `<port>` number is 9080. If you installed the application correctly, you see Hello replies.

- ▶ J2EE application clients are marked as J2EE clients in Figure 9-1 on page 214.

The following J2EE application clients are also wrapped in the `ItsHelloEAR.ear` file mentioned previously:

- *HelloClient*, which is a J2EE client that accesses the unsecure hello bean in the EJB container directly. See “ItsHello unsecure J2EE client” on page 225
- *HelloSecuredClient*, which is a J2EE client that accesses the secure hello bean in the EJB container. See “ItsHello secure J2EE client” on page 227.
- *HelloSecuredJAASClient*, which behaves similar to the *HelloSecuredClient*, but the authentication process is controlled programmatically within the client. See “J2EE Java application client” on page 237.
- *HelloSecureJAASClientC*, which is similar to *HelloSecuredJAASClient* but uses a custom *CallbackHandler* for collecting authentication information. See “Custom CallbackHandler” on page 240.

- ▶ Thin application clients are marked as “Thin Clients” in Figure 9-1 on page 214. They contain clients similar to the four J2EE application clients, but are written as thin application clients. The application contains two `.jar` files called `ItsHelloTHINCLIENT.jar` and `ItsHelloEJB.jar` with additional configuration and key files.

You can find the installation process for this ItsHello application in Appendix A, “Additional configurations” on page 509.

9.2 Java client authentication protocol

Accessing secure EJB resources in a secure WebSphere Application Server V6.1 environment requires an authentication protocol to determine the level of security and the type of authentication between the client and the server. The authentication protocol merges the server and client authentication requirements and comes up with an authentication policy specific for them. This authentication policy, among others, determines the following items:

- ▶ The kind of connection used, either SSL or TCP/IP
- ▶ If SSL is used, the strength of the encryption
- ▶ The way to authenticate the client, whether by using, for example, a user ID and password combination or client certificate

In WebSphere Application Server V6.1, two authentication protocols are available:

- ▶ IBM Secure Authentication Service
- ▶ Common Secure Interoperability Version 2 (CSIV2)

IBM Secure Authentication Service is the only authentication protocol that all WebSphere Application Servers used prior to Version 5. The CSIV2, defined by the Object Management Group (OMG), is a standard protocol defined for vendors to interoperate securely. CSIV2 is considered the strategic protocol and is implemented with more features than IBM Secure Authentication Service within the WebSphere Application Server V6.1.

In preparation for a request to flow from client to server, two client and server side Object Request Brokers (ORBs) must establish a connection over a TCP/IP (or SSL) transport layer. The IIOp is used for handling the communication between these two ORB objects. The IBM Secure Authentication Service and CSIV2 authentication protocols are add-on services for the IIOp.

Note: The IBM Secure Authentication Service and CSIV2 authentication protocols used in WebSphere Application Server are add-on services to the standard IIOp protocol for handling communication between two ORBs. Within WebSphere Application Server V6.1, the authentication protocol IBM Secure Authentication Service is deprecated, but is still included for backward compatibility.

9.2.1 CSIV2 Security Attribute Service

The Common Security Interoperability Version 2 specification is defined by the OMG at the following Web address:

<http://www.omg.org>

This specification defines the CSIV2 Security Attribute Service protocol to address the requirements of CORBA security for interoperable authentication, delegation, and privileges.

The CSIV2 Security Attribute Service protocol is designed to exchange its protocol elements in the *service context* of a General Inter-ORB Protocol (GIOP) request and to reply to messages that are communicated over a connection-based transport. The protocol provides client authentication, delegation, and privilege functionality that can be applied to overcome corresponding deficiencies in an underlying transport.

The CSIV2 Security Attribute Service protocol facilitates interoperability by serving as the higher-level protocol under which secure transports can be unified. The CSIV2 Security Attribute Service protocol is divided into two layers:

- ▶ The *authentication layer* is used to perform client authentication where sufficient authentication cannot be accomplished in the transport.
- ▶ The *attribute layer* can be used by a client to deliver security attributes, such as identity and privilege, to a target where they can be applied in access control decisions.

The attribute layer also provides the means for a client to assert identity attributes that differ from the client's authentication identity (as established in the transport or CSIV2 Security Attribute Service authentication layers). This identity assertion capability is the basis of a general-purpose impersonation mechanism that makes it possible for an intermediate to act on behalf of an identity other than itself. This can improve the performance of a system because the authentication of a client is relatively expensive. The server can validate the request by checking its trust rules.

9.2.2 Authentication process

Authentication is a process of establishing whether a client, which can be a user, a machine, or an application, is valid. Figure 9-2 shows the authentication process between client and server ORBs.

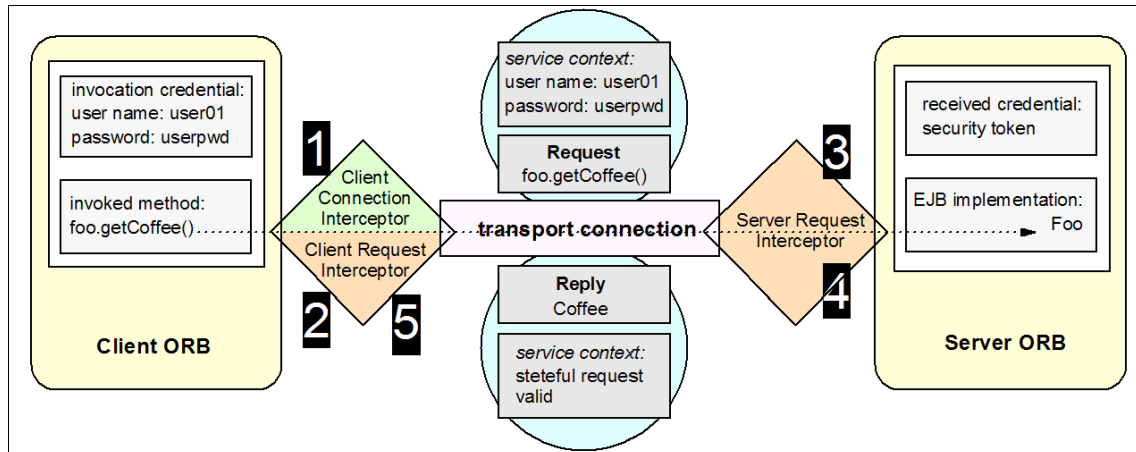


Figure 9-2 Authentication process

The process is summarized as follows:

1. The client ORB calls the connection interceptor to create the connection. The client ORB invokes the authentication protocol's client connection interceptor. It is used to read the tagged components in the interoperable Object Reference (IOR) of the server-based object that is being requested. This is how the authentication policy is established. After the policy is established, the ORB makes the connection, with the optional addition of the SSL cipher.
2. The client ORB calls the request interceptor to get client security information. The client ORB invokes the client request interceptor after the connection is established and sends security information other than what was established by the transport. This might include one of the following tokens:
 - A user ID and password token (authenticated by the server)
 - An authentication mechanism-specific token (validated by the server)
 - An identity assertion token (allows an intermediate to act on behalf of some identity other than itself)

This additional security information is sent with the message in a GIOP's *service context*. After the client request interceptor finishes adding the service context to the message, the message is sent to the server ORB.

3. The server ORB calls the request interceptor to receive the security information, authenticate, and set the received credential. Upon receiving the message, the server ORB invokes the authentication protocol's server request interceptor, which looks for the *service context*.
 - If the *service context* is found, a method is invoked to the security server to validate the client identity. When the client identity is valid, a *credential* is returned. This credential contains additional information about the client, retrieved from the used user registry, and is used for authorization process. The authorization process determines whether the user is allowed to access an EJB resource.
 - If the *service context* is not found, the server request interceptor looks at the transport connection to see if a client certificate chain is supplied. This is the case when SSL client authentication is configured between the client and server. If such a certificate is found, the distinguished name (DN) is extracted and is mapped to an identity in the selected user registry.
 - If the certificate does not map, no credential is created, and the request is rejected.
 - If the certificate maps, but the presented security information is invalid, the method request is rejected, and an exception is sent back with the reply.
 - If the certificate maps, but no security information is presented, an *unauthenticated credential* is created. Only EJB methods with no security roles or EJB methods with a special *Everyone* role can be accessed using this unauthenticated credential.
4. The server ORB calls the request interceptor so that security can send information back to the client with the reply.

When the method invocation is completed, the server request interceptor is invoked again to complete the server authentication, and a new reply service is created to inform the client request interceptor of the outcome.
5. The client ORB calls the request interceptor so that the client can clean up and set the session status as good or bad.

The client request interceptor receives a reply from the server. The CSIV2 Security Attribute Service supports both *stateless* and *stateful* security contexts. A *stateless context* exists only for the duration of the GIOP request that was used to establish the context. A *stateful context* endures until they are discarded. If a stateful context is used, only the first request between a client and server requires that the security information is sent. All subsequent method requests must send a unique context ID only, and the server can look up the credential stored in its session table.

9.3 Java client configuration

As explained in 9.2, “Java client authentication protocol” on page 216, accessing secure EJB resources in a secure WebSphere Application Server environment requires an authentication protocol. This protocol is required to determine the level of security and the type of authentication between the client and the server, such as the kind of connection used (for example SSL or TCP/IP), the strength of the encryption used, the type of authentication used (for example user ID/password or certificate), and so on.

A Java client application that is accessing a secure EJB resource within WebSphere Application Server must specify these properties. These configuration properties are specified in a file, as shown in Example 9-1, defined by the `com.ibm.CORBA.ConfigURL` system property of the client's JVM and are in the `sas.client.props` sample file. The application server must also be configured to communicate with a client in the required fashion. If a Java client requires that client certificates be transmitted, for example, by using SSL, then the server must be set to expect this.

Example 9-1 shows the starting of the Java client application `com.ibm.Foo` using the CORBA configuration file `properties/sas.client.props` and using the Java Authentication and Authorization Service (JAAS) login configuration file `properties/wsjaas_client.conf`.

Example 9-1 Starting the Java client application `com.ibm.Foo`

```
java -Dcom.ibm.CORBA.ConfigURL=file:properties/sas.client.props
-Djava.security.auth.login.config=file:properties/wsjaas_client.conf
com.ibm.Foo
```

The `sas.client.props` file

The `sas.client.props` configuration file contains several sets of properties that are explained as follows. The default values are marked by an asterisk (*).

► Client Security Enablement

`com.ibm.CORBA.securityEnabled (true*, false)`

This property determines if client security has been enabled. If the server's global security is enabled, the value of this property must be set to `true`. Otherwise all the secured remote EJB resources cannot be accessed by the client.

► RMI/IIOP Authentication Protocol

`com.ibm.CSI.protocol` (`sas`, `csiv2`, `both*`)

This property determines which add-on authentication protocol is used.

- `both` is used when communicating with all kind of WebSphere Application Server V6.1
- `csiv2` is used when communicating with only servers Versions 5.x or 6.x.
- `sas` is used when communicating with only servers prior to Version 5.x.

► Authentication Configuration:

- `com.ibm.CORBA.authenticationTarget` (`BasicAuth*`)

`BasicAuth` is the only supported option for a pure Java client. The user name and password are sent to the server for message layer authentication only. The SSL client certificate authentication must be configured under CSIV2 configuration.

- `com.ibm.CORBA.validateBasicAuth` (`true*`, `false`)

This property determines if the user details are authenticated immediately or deferred until the first method request is communicated to the server, when the `com.ibm.CORBA.authenticationTarget` property is set to `BasicAuth`.

- `com.ibm.CORBA.authenticationRetryEnabled` (`true*`, `false`)

This property determines whether a failed login must be retried. It also applies to stateful CSIV2 sessions and validations that have failed due to an expired credential. Only those failures that are known to be correctable are retried. This option is valid when `com.ibm.CORBA.validateBasicAuth` is set to `true`.

- `com.ibm.CORBA.authenticationRetryCount` (an integer value, 3*)

This property determines how many retrieves are to be attempted for failed login when `com.ibm.CORBA.authenticationRetryEnabled` is set to `true`.

- `com.ibm.CORBA.securityServerHost`

This property indicates the name (or IP address) of the security server to validate the user ID and password.

- `com.ibm.CORBA.securityServerPort`

This property indicates the port number of the security server.

- `com.ibm.CORBA.loginSource` (`prompt*`, `keyfile`, `stdin`, `none`, `properties`)

This property determines how the authentication request interceptor logs in if it does not find an invocation credential set.

- `prompt` displays a window requesting a user name and password.
 - `keyfile` extracts the user details from the file specified by `com.ibm.CORBA.keyFileName`.
 - `stdin` displays a command line prompt requesting user details.
 - `none` must be selected if the client uses programmatic login.
 - `properties` retrieves the user details from the `com.ibm.CORBA.loginUserId` and `com.ibm.CORBA.loginPassword` properties.
- `com.ibm.CORBA.loginUserId`
This property indicates the user ID that is used when the `com.ibm.CORBA.loginSource` property is set to `properties`.
 - `com.ibm.CORBA.loginPassword`
This property indicates the user password that is used when the `com.ibm.CORBA.loginSource` property is set to `properties`.
 - `com.ibm.CORBA.keyFileName`
This property indicates the location of the key file that contains a list of realm, user ID, and password combinations. See the file `<WebSphere_home>/profile/default/properties/wsserver.key`. This file is used when the `com.ibm.CORBA.loginSource` property is set to `keyfile`.
 - `com.ibm.CORBA.loginTimeout`
This property is an integer within the range of 0 and 600. The default is 300. It is the amount of time, in seconds, that the login prompt is available before the login is considered invalid.
- **SSL Configuration:**
- `com.ibm.security.useFIPS` (`false*`, `true`)
This property indicates that the client wants to be in Federal Information Processing Standard (FIPS)-approved cryptographic algorithm mode.
 - `com.ibm.ssl.contextProvider` (`IBMJSSE2*`, `IBMJSSE`, `IBMJSSEFIPS`)
This property indicates is the Java Secure Socket Extension (JSSE) provider that is used. Specifying `IBMJSSEFIPS` means that the client wants to be in FIPS-approved cryptographic algorithms mode. It also means that the run time uses the `IBMJSSE2` provider in combination with the `IBMJCEFIPS`.
 - `com.ibm.ssl.protocol` (`SSL*`, `SSLv2`, `SSLv3`, `TLS`, `TLSv1`)
This property determines which variety of the SSL and Transport Layer Security (TLS) protocols are used to perform transport-layer encryption.

- `com.ibm.ssl.keyStoreType` (JKS*, JCEK, PKCS12)
This property indicates the format of the SSL key store file.
- `com.ibm.ssl.keyStore`
As an example, consider the `keys/DummyClientKeyFile.jks` file. This property indicates the location of the SSL key store file, which has personal certificates and private keys.
- `com.ibm.ssl.keyStorePassword`
This property indicates the password with which the key store file is protected.
- `com.ibm.ssl.trustStoreType` (JKS*, JCEK, PKCS12)
This property indicates the format of the SSL key trust file.
- `com.ibm.ssl.trustStore`
As an example, consider the `keys/DummyClientTrustFile.jks` file. This property indicates the location of SSL key trust file.
- `com.ibm.ssl.trustStorePassword`
This property indicates the password with which the key trust file is protected.
- ▶ IBM Secure Authentication Service add-on authentication protocol:
`com.ibm.CORBA.standardClaimQOPModels` (low, medium, high*)
This property determines the quality of protection (QOP), or rather the security level. If the server and client values differ, the highest value is chosen, and the connection is initialized with this QOP property.
- ▶ CSIV2 add-on authentication protocol
Certain security properties have supported or required property pairs. The required properties take precedence over the supported properties pair. Therefore, if the required property is enabled, communication with the server must satisfy this property.
 - `com.ibm.CSI.performStateful` (true*, false)
This property determines whether the client supports the stateful or stateless session.
 - `com.ibm.CSI.performClientAuthenticationRequired` (true*, false)
`com.ibm.CSI.performClientAuthenticationSupported` (true*, false)
When supported, message layer client authentication is performed when communicating with any server that supports or requires authentication. Message layer client authentication transmits a user ID and password if the `authenticationTarget` property is `BasicAuth`, or it transmits a

credential token if the `authenticationTarget` property is one of the token-based mechanism, for example, Lightweight Third Party Authentication (LTPA), Kerberos.

When required, message layer client authentication must occur when communicating with any server. If the transport layer authentication property is also enabled (see the following property), both authentications are performed. However, the message layer client authentication takes precedence at the server side.

- `com.ibm.CSI.performTLClientAuthenticationRequired (true*, false)`
`com.ibm.CSI.performTLClientAuthenticationSupported (true*, false)`

When supported, transport layer client authentication can be performed, and the client sends digital certificate to the server during the authentication stage.

When required, the client only authenticates with servers that support transport-layer client authentication.

- `com.ibm.CSI.performTransportAssocSSLTLSRequired (true*, false)`
`com.ibm.CSI.performTransportAssocSSLTLSSupported (true*, false)`

When supported, the client can use either TCP/IP or SSL to communicate with the server.

When required, the client only communicates with servers that support SSL.

- `com.ibm.CSI.performMessageIntegrityRequired (true*, false)`
`com.ibm.CSI.performMessageIntegritySupported (true*, false)`

These properties are only valid when SSL is enabled.

When supported, it can make an SSL connection with either 40-bit ciphers or digital-signing ciphers.

When required, the connection fails if the server does not support 40-bit ciphers.

- `com.ibm.CSI.performMessageConfidentialityRequired (true*, false)`
`com.ibm.CSI.performMessageConfidentialitySupported (true*, false)`

These properties are only valid when SSL is enabled.

When supported, it can make SSL connection with either 128-bit ciphers or a lower encryption strength.

When required, the connection fails if the server does not support 128-bit ciphers.

► Additional CORBA configuration

`com.ibm.CORBA.requestTimeout` (integer value, 180*)

This property specifies the timeout period, in seconds, for responding to requests sent from the client. Use care when specifying this property, and set it only if the application is experiencing problems with timeouts.

For a more complete list of directives, see the WebSphere Application Server Information Center.

9.4 J2EE application client

A J2EE application client operates in a similar fashion to a J2EE server-based application. It uses the RMI/IIOP protocol and CORBA services that are provided by the J2EE platform. This usage enables the J2EE application client to access both EJB and CORBA object references. The J2EE platform allows the J2EE application client to use the JNDI names, defined in the deployment descriptor, to access the EJBs or other resources such as Java Database Connectivity (JDBC), Java Messaging Service (JMS), JavaMail™, and so on.

The `ItsohelloJ2EEClient.jar` file (wrapped in the `ItsohelloEAR.ear` application) provided with this IBM Redbooks publication has four J2EE application clients. These clients request services to enterprise beans operating in a remote EJB container. The first two clients are discussed in this section.

The J2EE client application depends on the *application client run time* to configure its execution run time. You can use the **launchClient** command (`<WebSphere_home>\bin\launchClient.bat`) to start and configure the J2EE application client environment by examining the application client's descriptor (`application-client.xml`). Access to the EJB JAR file that contains the EJB home interfaces and access to the client's class file must be referenced in the client's MANIFEST.MF file as shown in Example 9-2.

Example 9-2 Client's MANIFEST.MF file

```
Manifest-Version: 1.0
Class-Path: ItsohelloEJB.jar
Main-Class: com.ibm.itsohello.j2eeclient.J2EEClient
```

9.4.1 Itsohello unsecure J2EE client

The unsecure Itsohello J2EE client application is a text-based Java application that accesses the unsecure bean `HelloBean` in a remote EJB container. Upon successful execution, it shows a hello message created by the bean. The JNDI

name for this bean is `ejb/itsohello/hello` (Figure 9-1 on page 214). Example 9-3 shows the part of the client for connecting to the bean and getting the message.

Example 9-3 J2EE client to unsecured Hello bean

```
InitialContext ic = new InitialContext();
Object homeObject = ic.lookup("ejb/itsohello/hello");
HelloHome helloHome = (HelloHome)
PortableRemoteObject.narrow(homeObject,
    HelloHome.class);
msg = helloHome.create().getMessage();
```

In Example 9-3, note that there is no reference that indicates the server in which the remote enterprise bean is located. As mentioned already, the *Application Client run time* is responsible for this configuration. This is one of the features of the J2EE application client.

To start the application to access the unsecured bean:

1. Make sure that the `ItsohelloEAR.ear` application is already installed in the destination WebSphere Application Server. Among others, two EJB with appropriate access privilege are automatically installed by using JNDI names, which are `ejb/itsohello/hello` and `ejb/itsohello/securedhello`. These two EJB resources are the remote resources for the example application clients. See Figure 9-1 on page 214.

2. From a command prompt, launch the J2EE client as follows:

```
<WebSphere_home>\bin\launchClient ItsohelloEAR.ear
-CCBootstrapHost=<Server_hostname>
-CCBootstrapPort=<RMICconnector_port>
```

The default value for `<Server_hostname>` is *localhost* and `<RMICconnector_port>` is 2809. You can also use the included `runJ2EEClient.bat` batch file for running the application. Remember to modify the parameters according to the setup of your system.

Example 9-4 shows the messages from the application.

Example 9-4 Results from the runJ2EEClient.bat file

J2EE Itsohello clients:

- a. UNSECURED CLIENT.
...
- b. SECURED CLIENT.
...
- c. SECURED CLIENT with JAAS.
...

d. SECURED CLIENT with JAAS using custom callback handler.

...

Please enter your choice (a/b/c/d):

3. Press option **a** and ENTER.

The client application uses the `com.ibm.itsohello.j2eeclient.HelloClient` class to connect to the unsecured HelloBean. When it finishes, you see the following message:

```
Accessing unsecured Hello bean
Message from Hello bean: Hello to you UNAUTHENTICATED (role:
Anonymous)
```

Note: If the global security is enabled, and the value of the property `com.ibm.CORBA.loginSource` is set to `prompt` in the file `sas.client.props`, the client shows a window that requests a user identity and password, even if the bean is not secured. To disable the window, set the `com.ibm.CORBA.loginSource` property to `none`.

9.4.2 Itsohello secure J2EE client

The Itsohello secure J2EE client application is similar to the unsecured J2EE client, except that it shows a message created by the secure `SecuredHelloBean` in a remote EJB container. The JNDI name for this bean is `ejb/itsohello/securedhello`. In Example 9-5, the code snippet from `com.ibm.itsohello.j2eeclient.SecuredHelloClient` class shows that there is no difference in client code for accessing secure or unsecured enterprise beans.

Example 9-5 J2EE client to secure Hello bean

```
InitialContext ic = new InitialContext();
Object homeObject = ic.lookup("ejb/itsohello/securedhello");
SecuredHelloHome helloHome = (SecuredHelloHome)
PortableRemoteObject.narrow
(homeObject, SecuredHelloHome.class);
msg = helloHome.create().getMessage();
```

The procedure to start the included secure Itsohello example application is similar to the Itsohello unsecured J2EE client. However, in step 2 in 9.4.1, "Itsohello unsecured J2EE client" on page 225, you choose SECURED CLIENT (option **b**). This option starts the secure client `com.ibm.itsohello.j2eeclient.SecuredHelloClient`.

When global security is enabled and the `com.ibm.CORBA.loginSource` property in the CORBA client configuration file (for example `sas.client.props`) is set to `prompt`, a window (Figure 9-3) opens that prompts for a user ID and password.



Figure 9-3 Challenge window

After the client is authenticated, the appropriate remote method in `com.ibm.itsohello.bean.SecuredHelloHome` is invoked. When the access is successful, it shows a message similar to the following example:

```
Accessing Secured Hello bean
Message from Hello bean: [Secured] Hello to you viking (role:
BeanGuest)
```

Note: There is no difference in the J2EE client code for accessing secure or unsecure EJB resources, unless JAAS APIs are used. The behavior of the authentication process is controlled by the client configuration file (for example `sas.client.props`).

9.5 Thin application client

The phrase *thin application client* refers to a Java client that is not running within the J2EE client container. It is a stand-alone Java application that implements EJB clients that connect to a remote EJB container of WebSphere Application Server. Because it is not running under a J2EE client container, when resolving to an enterprise bean, the client must know the location of the name server and the fully qualified name that is used for the remote resource. The thin application client must initialize and code explicitly access to any of the services that the client might require.

To develop a thin application client:

1. Initialize the `org.omg.CORBA.ORB` object.
2. Optional: Initialize the `org.omg.Cosnaming.NamingContextExt` file if `CosNaming` is used.

3. Use the ORB object (or the derived NamingContextExt object) to get a reference to the enterprise bean by using the fully qualified physical location of the enterprise bean in the name space. WebSphere Application Server provides a script <WebSphere_home>\bin\dumpNameSpace.bat, which is useful for determining the fully qualified physical location names. Example 9-6 shows such an output.

Example 9-6 Results of the dumpNameSpace.bat file

```
...
=====
Name Space Dump
  Provider URL: corbaloc:iiop:localhost:2809
  Context factory: com.ibm.websphere.naming.WsnInitialContextFactory
  Requested root context: cell
  Starting context: (top)=mka0k1myNode01Cell
  Formatting rules: jndi
  Time of dump: Mon Nov 08 11:45:20 EST 2004
=====

=====
Beginning of Name Space Dump
=====

1 (top)
2 (top)/persistent          javax.naming.Context
...
38 (top)/nodes/mka0k1myNode01/servers/server1/ejb/itsohello
38                          javax.naming.Context
39 (top)/nodes/mka0k1myNode01/servers/server1/ejb/itsohello/hello
39                          com.ibm.itsohello.bean.HelloHome
40 (top)/nodes/mka0k1myNode01/servers/server1/ejb/itsohello/secured
hello
40                          com.ibm.itsohello.bean.SecuredHelloHome
...
=====
```

The rest of the code is similar to the J2EE client.

Example: Consider the following fully qualified name that is used in Example 9-6:

```
cell/nodes/mka0klmyNode01/servers/server1/ejb/itsohello/hello
```

This can also be reached by using an *unqualified name*, with the help of the ORB method:

```
string_to_object(corbaname:iop:<host>:<port>/NameServiceServerRoot#ejb/itsohello/hello).
```

9.5.1 Running a thin application client

You must set certain configurations to run a thin application client to operate in a secure environment. If the WebSphere Application Clients product is installed, you can use the provided *buildClientRuntime* tool (in the <WebSphereClient_home>\bin\buildClientRuntime.bat file) to build the required components for the client.

However, if the WebSphere Application Clients product is not installed, obtain the components by using the installed WebSphere Application Server, as follows:

1. Get the Java 2 Runtime Environment (J2RE) that is provided by WebSphere, including the libraries in the <WebSphere_home>\java\jre\lib and <WebSphere_home>\java\jre\lib\ext directories.
2. Collect all application client runtime, properties, and configuration files. For a secure environment, the JVM must point to a CORBA configuration file (for example, the sas.client.props file) by using the JVM system property of com.ibm.CORBA.ConfigURL. Optionally, if the application client uses the JAAS APIs for login, the JVM must probably include the JAAS login configuration file indicated by the JVM system property java.security.auth.login.config.
3. Collect the keystore and truststore files if SSL is used. These files are referred to in the CORBA configuration file mentioned previously.
4. Collect libraries from the WebSphere runtime library in the <WebSphere_home>\lib directory. Not all of them are required. However, for the Itsohello thin application client (see the script in Example 9-7 on page 231), the following files are required:

activity.jar	admin.jar	bootstrap.jar	cluster.jar
ecutils.jar	emf.jar	idl.jar	iwsorb.jar
j2ee.jar	lmpoxy.jar	management.jar	naming.jar
namingclient.jar	ras.jar	runtime.jar	runtimefw.jar
sas.jar	securityimpl.jar	txClient.jar	txClientPrivate.jar
utils.jar	wccm_client.jar	wsexception.jar	wssec.jar

Example 9-7 Script for running ItsohelloTHINCLIENT application

```
set WAS_HOME=C:\WebSphere\AppServer
set SERVER_HOST=mka0k1my.itso.ral.ibm.com
set SERVER_PORT=2809

set CLASSPATH=.\;. \prop\;itsohelloEJB.jar;itsohelloTHINCLIENT.jar
set JAVA_LIB=-Djava.ext.dirs=%WAS_HOME%\java\jre\lib;
    %WAS_HOME%\java\jre\lib\ext;%WAS_HOME%\lib

set CORBA_CONFIG=-Dcom.ibm.CORBA.ConfigURL=file:prop/sas.client.props
set LOGIN_CONFIG=-Djava.security.auth.login.config=file:prop/wsjaas_client.conf
set CLIENT_TRACE=-Dcom.ibm.CORBA.CommTrace=false

%WAS_HOME%\java\bin\java -cp %CLASSPATH% %CORBA_CONFIG%
%LOGIN_CONFIG% %CLIENT_TRACE% %JAVA_LIB%
com.ibm.itsohello.thinclient.ThinClient %SERVER_HOST% %SERVER_PORT%
```

Attention: Do *not* run the application client by using the different JVM versions than what is used by the destination server. The use of a different class implementation might give unexpected results.

9.5.2 Itsohello insecure thin client

You can find all the thin application clients used in the `ItsohelloTHINCLIENT.jar` file. To run the code, the `ItsohelloEJB.jar` EJB JAR file is required. The setup of the clients is the same as for the J2EE application clients. See 9.4, “J2EE application client” on page 225. However, the batch `runThinClient.bat` is used to start the application client. Remember to modify the parameters according to the setup of your system.

There are two ways to program the thin application client to access a remote enterprise bean. You can use either `CosNaming` with a fully qualified resource name or the ORB method `string_to_object` with an unqualified resource name. Both approaches are shown as follows:

- Example 9-8 shows the use of `CosNaming` with a qualified name. The fully qualified name is used in this code snippet.

Example 9-8 Code snippet of thin client to an insecure Hello bean using CosNaming

```
// initialize ORB object
java.util.Properties props = new java.util.Properties();
props.put("org.omg.CORBA.ORBClass", "com.ibm.CORBA.iiop.ORB");
```

```

props.put("com.ibm.CORBA.ORBInitRef.NameService","corbaloc:iiop:" +
    serverHostname + ":" + serverPort + "/NameService");
props.put("com.ibm.CORBA.ORBInitRef.NameServiceServerRoot","corbaloc:iiop:" +
    serverHostname + ":" + serverPort + "/NameServiceServerRoot");
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init((String[]) null,
props);

// get the home object
Object obj = orb.resolve_initial_references("NameService");
org.omg.CosNaming.NamingContextExt initCtx =
    org.omg.CosNaming.NamingContextExtHelper.narrow(obj);
Object homeObject = initCtx.resolve_str(

"ce11/nodes/mka0k1myNode01/servers/server1/ejb/itsohello/hello"
    );

HelloHome helloHome = (HelloHome)
PortableRemoteObject.narrow(homeObject,
    HelloHome.class);
msg = helloHome.create().getMessage();

```

- ▶ Example 9-9 shows use of the *ORB string_to_object* method with an unqualified name. With the help of the name server, the Hello bean can also be reached by using an unqualified name, which is shown in the code snippet.

Example 9-9 Code for thin client to unsecure Hello bean using ORB string_to_object

```

java.util.Properties props = new java.util.Properties();
org.omg.CORBA.ORB orb=org.omg.CORBA.ORB.init((String[])null, props);

// get the home object
String resourceName = "corbaname:iiop:" + serverHostname + ":" +
serverPort +
    "/NameServiceServerRoot#ejb/itsohello/hello";
Object homeObject = orb.string_to_object(resourceName);

HelloHome helloHome = (HelloHome)
PortableRemoteObject.narrow(homeObject,
    HelloHome.class);
msg = helloHome.create().getMessage();

```

9.5.3 Itsohello secure thin client

The process to program the Itsohello secure thin client is similar to the process for a J2EE application client. See 9.4.2, “Itsohello secure J2EE client” on page 227. There is no difference in coding for secure or for unsecure client. The behavior of the application is determined by the client configuration file, for example `sas.client.props`, as explained in 9.3, “Java client configuration” on page 220.

9.6 Programmatic login

If you are required to implement a custom login mechanism for the application client, for example, because the provided security infrastructure cannot supply all the functionality that is required, you can use a programmatic login by using JAAS. JAAS contains a collection of strategic authentication APIs that enable developers to create their own login module.

9.6.1 JAAS login module in WebSphere

The authentication process between a Java application client and a remote EJB is explained in 9.2.2, “Authentication process” on page 218. In Figure 9-4 on page 234 shows the simplified authentication process within the WebSphere Application Server to indicate the role of JAAS. The process flows as follows:

1. Java clients send the authentication information to the EJB authenticator module. The authentication information can be a basic authentication (only a user ID and password pair) or a credential token (for LTPA).
2. The EJB authenticator module pass the authentication information to the JAAS login module.
3. The login module uses the specified authentication mechanism, which is LTPA.
4. To validate the authentication information, the authentication module uses LocalOS, Lightweight Directory Access Protocol (LDAP), or custom registry.
5. After authentication, the login module creates a JAAS *Subject* (`javax.security.auth.Subject`). This subject, in addition to having the user's realm (`getPrincipals()`), contains a CORBA credential in its public credential list attribute (`getPublicCredentials()`). This credential is used by the authorization service to perform further access to any resources.

Note the importance of the JAAS login module in Figure 9-4.

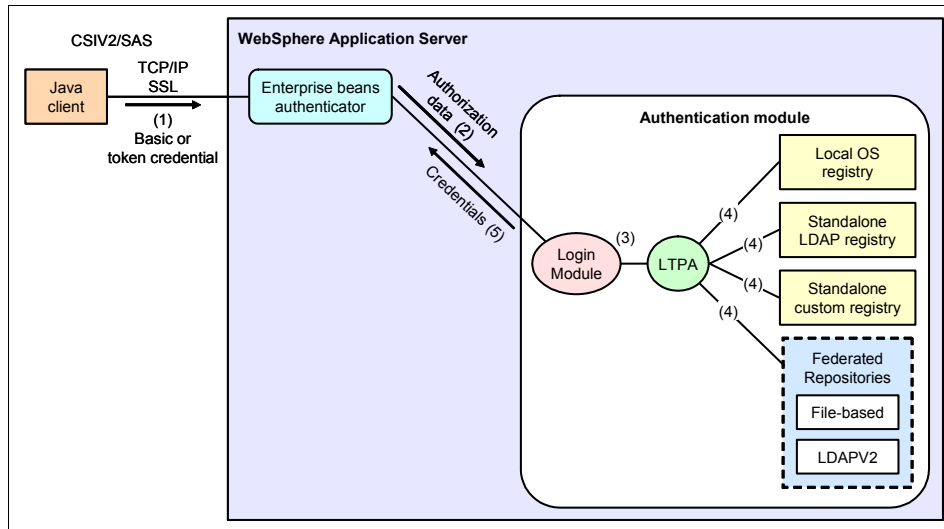


Figure 9-4 Authentication process within WebSphere Application Server

WebSphere Application Server allows the JAAS login module (Figure 9-4) to perform programmatic authentication to the WebSphere Application Server security run time. It has already several built-in JAAS login configurations that programmers can use directly, such as the following examples:

- ▶ WLogin

This configuration is a generic JAAS login configuration that you can use with almost any application, including the Java application client, to perform authentication based on a user ID and password or a token.

- ▶ ClientContainer

Similar to WLogin, this JAAS login configuration acknowledges the *CallbackHandler* that is 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 specifies one *CallbackHandler* in the *LoginContext*.

In WebSphere, you can find the information of the supported built-in JAAS login configurations in the `wsjaas_client.conf` file. This file must be referred by the JVM runtime system property `java.security.auth.login.config` of the application client. See Example 9-1 on page 220.

9.6.2 Programmatic login process

Programmatically, the interaction diagram in Figure 9-5 explains the login process and access to secure resource by using the JAAS APIs.

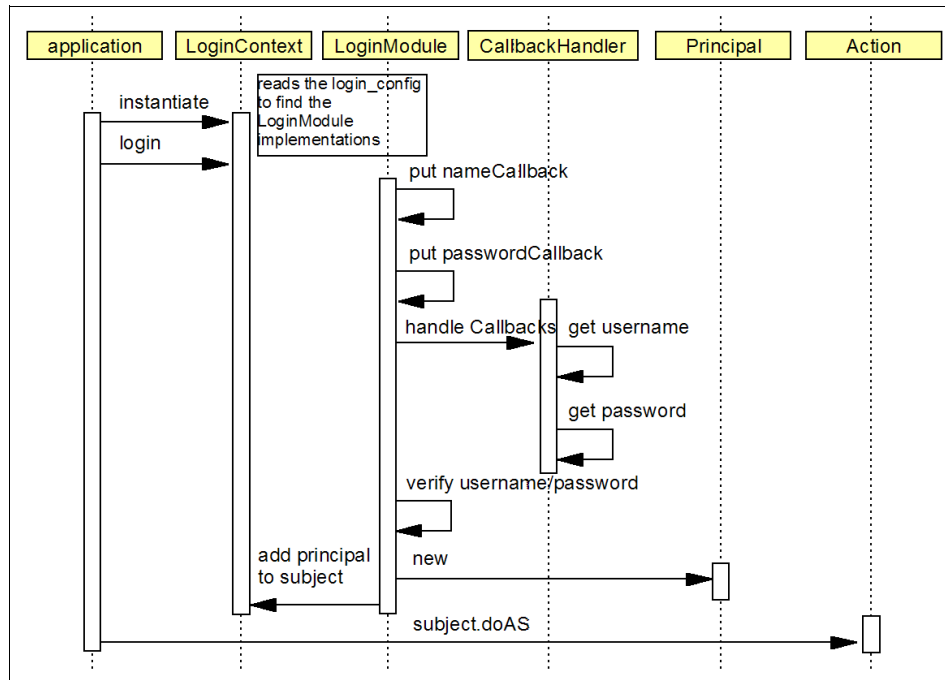


Figure 9-5 Interaction diagram for login process using JAAS APIs

The illustration in Figure 9-5 is described as follows:

- ▶ Application starts the login process.
 - LoginContext is initialized.
 - *LoginModule* is invoked. Depending on the design of the code, a user ID or password combination can be provided with the help of a created *CallbackHandler* object.
 - Upon successful verification of the supplied user ID and password, the *LoginModule* creates a *Subject* that contains the user's realm and a credential. The *LoginModule* must provide a validated credential and some context for where that credential is valid, which is the authentication realm (or user registry) where the user ID was validated.
- ▶ Application retrieves the created *Subject* from *LoginContext*.
- ▶ Using the *doAs* method, the application invokes an *Action* under the acquired *Subject*.

Having read the description, the Java code can be as simple as Example 9-10.

Example 9-10 Java code snippet for interaction diagram shown in Figure 9-5

```
// login block
CallbackHandler loginHandler = new WSCallbackHandlerImpl("uid", "pwd");
LoginContext lc = new LoginContext("WSLogin", loginHandler);
lc.login();
Subject subject = lc.getSubject();

// create Action for accessing the protected bean method
java.security.PrivilegedAction getHelloMessage = new
    java.security.PrivilegedAction() {
    public Object run() {
        try {
            Object obj = ic.lookup("ejb/itsohello/securedhello");
            SecBeanHome hello = (SecBeanHome)
                PortableRemoteObject.narrow(obj, SecBeanHome.class);
            return hello.create().getMessage();
        }
        catch (Exception e) {
            ...
        }
    }
}

// run the created Action with the acquired subject
msg = (String) com.ibm.websphere.security.auth.WSSubject.doAs
    (subject, getHelloMessage);
```

9.6.3 Client-side programmatic login using JAAS

A client-side login is useful when the user has to login to a security domain on a remote system. However, this type of login requires that both the client and server use the same process to authenticate and to collect the login information for authentication purposes. The JAAS interface `javax.security.callback.CallbackHandler` defines how the security services might interact with the application to retrieve the authentication data.

Built-in CallbackHandler in WebSphere

WebSphere Application Server provides several class implementations of the `javax.security.auth.callback.CallbackHandler`. The following `CallbackHandlers` are the most useful for a *client-side programmatic* login:

- ▶ `com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl`
This implementation presents a GUI login panel to prompt users for authentication data.
- ▶ `com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl`
This callback prompts a user for authentication data, which is useful for text-based client applications.

Whenever the `CallbackHandler` implementations do not fulfill the user's requirement, you can make a custom `CallbackHandler` implementation, which is further discussed in “Custom `CallbackHandler`” on page 240.

J2EE Java application client

Example 9-11 shows the code for you to perform a programmatic login by using a GUI `CallbackHandler` `WSGUICallbackHandlerImpl`. If you prefer the text-based login, you must change the callback handler in the code to `WSStdinCallbackHandlerImpl`.

Notice that the code in Example 9-11 is non-functional because it has nothing that the authentication calls against. That is, there is no reference to which server this authentication must be validated nor is there a reference for the enterprise bean.

Example 9-11 J2EE client to secure Hello bean, using JAAS APIs

```
try
{
    ic = new InitialContext();

    // Invoke the JAAS Login module
    CallbackHandler loginCallbackHandler = new WSGUICallbackHandlerImpl();
    LoginContext lc = new LoginContext("WSLogin", loginCallbackHandler);
    lc.login();
    Subject subject = lc.getSubject();

    // create action to access the protected bean method
    java.security.PrivilegedAction getHelloMessage = new
    java.security.PrivilegedAction() {
        public Object run() {
            try {
```

```

        Object homeObject = ic.lookup("ejb/itsohello/securedhello");
        SecuredHelloHome helloHome = (SecuredHelloHome)
            PortableRemoteObject.narrow(homeObject, SecuredHelloHome.class);
        return helloHome.create().getMessage();
    }
    catch (CreateException ce) {
        ...
    }
}
};

// invoke the secure action using the created subject
msg = (String) com.ibm.websphere.security.auth.WSSubject.doAs(subject,
getHelloMessage);
}
catch (NamingException ne) {
    ...
}

```

The procedure to start the included secure Itsohello application by using the JAAS APIs client example is similar to the Itsohello unsecure J2EE client. However, in step 2 in 9.4.1, “Itsohello unsecure J2EE client” on page 225, you can now choose SECURED CLIENT with JAAS (option c). This starts the client `com.ibm.itsohello.j2eeclient.SecuredHelloJAASClient`. Because the login process is now controlled programmatically, the value of the `com.ibm.CORBA.loginSource` property in the CORBA client configuration file (for example `sas.client.props`) no longer has any influence.

Thin Java application client

Similar to the J2EE application client, the JAAS programmatic login can be implemented for a thin Java application client. Notice how the security realm is established. Example 9-12 shows the difference in the implementation.

Example 9-12 Thin application client to secure Hello bean

```

try {
    // initialize the ORB object
    orb = ORB.init((String[]) null, new Properties());

    // IMPORTANT: this is a dummy call to server to establish security realm for JAAS.
    // it should be done before the JAAS login
    orb.string_to_object("corbaname:iop:" + serverHostname + ":" + serverPort);

    // Invoke the JAAS login module
    CallbackHandler loginCallbackHandler = new WSGUICallbackHandlerImpl();
}

```



```

LoginContext lc = new LoginContext("WSLogin", loginCallbackHandler);
lc.login();
Subject subject = lc.getSubject();

final String resourceName = "corbaname:iiop:" + serverHostname + ":" + serverPort
    + "/NameServiceServerRoot#ejb/itsohello/securedhello";

// create action to access the protected bean method
java.security.PrivilegedAction getHelloMessage = new
java.security.PrivilegedAction() {
    public Object run() {
        try {
            Object homeObject = orb.string_to_object(resourceName);
            SecuredHelloHome helloHome = (SecuredHelloHome)
                PortableRemoteObject.narrow(homeObject, SecuredHelloHome.class);
            return helloHome.create().getMessage();
        }
        catch (CreateException ce) {
            ...
        }
    }
};
msg = (String) com.ibm.websphere.security.auth.WSSubject.doAs(subject,
getHelloMessage);
}
catch (LoginException le) {
    ...
}

```

Example 9-12 shows the usual differences in programming between J2EE and thin application clients. However, there is another difference that required only when JAAS APIs programmatic login is used, which is the ORB method call:

```
orb.string_to_object("corbaname:iiop:<serverHostname>:<serverPort>");
```

This call is required to establish connection to the security realm server. This call is required because the JAAS programmatic login must know where the security realm server is to validate the user ID and password. Therefore, the call must be done *before* the `LoginContext.login()` method is invoked.

Custom CallbackHandler

When required, a custom `CallbackHandler` that implements the `CallbackHandler` interface can also be created. The interface has only one method that must be implemented:

```
public void handle(javax.security.auth.callback.Callback[] callbacks)
```

Different types of `Callback` objects can be used in this method. This feature gives a programmer the ability to interact with a calling application, to retrieve specific authentication data, such as user name and password, or to display certain information, such as an error or a warning message.

For a complete list of `CallbackHandler` implementations, see the WebSphere Information Center and the JAAS `javax.security.auth.callback` API documentation. For convenience, implementation examples are listed as follows:

- ▶ `javax.security.auth.callback.TextOutputCallback`
This implementation is used to display information messages as well as warning and error messages.
- ▶ `javax.security.auth.callback.NameCallback`
This implementation is used to retrieve the name information (login name).
- ▶ `javax.security.auth.callback.PasswordCallback`
This implementation is used to retrieve the password information.

A simple example for a custom callback handler is also included in the `Itsohello` client. Change the `WSGUICallbackHandlerImpl` to `HelloCallbackHandlerImpl` as shown in Example 9-11 on page 237 or as shown in Example 9-12 on page 238 if you want to use this custom `CallbackHandler`. The code snippet for the custom `CallbackHandler` is shown in Example 9-13.

Example 9-13 Code from custom `CallbackHandler` `HelloCallbackHandlerImpl` class.

```
public void handle(Callback[] callbacks) throws IOException,
    UnsupportedCallbackException {

    System.out.println("Custom CallbackHandler");
    System.out.println("Realm:" +
        WSLoginHelperImpl.getDefaultRealmName());

    for(int i = 0; i < callbacks.length; i++)
        if (callbacks[i] instanceof TextOutputCallback)
        {
            TextOutputCallback toc = (TextOutputCallback)callbacks[i];
            switch(toc.getMessageType())
            {
```

```

        case 0: // '\0'
            System.out.println(toc.getMessage());
            break;
        ...
        default:
            throw new IOException("Unsupported message type: " +
                toc.getMessageType());
    }
}
else if (callbacks[i] instanceof NameCallback)
{
    NameCallback nc = (NameCallback)callbacks[i];
    System.out.print(nc.getPrompt());
    System.out.flush();
    nc.setName((new BufferedReader(new
        InputStreamReader(System.in))).readLine());
}
else if (callbacks[i] instanceof PasswordCallback)
{
    PasswordCallback pc = (PasswordCallback)callbacks[i];
    System.out.print(pc.getPrompt());
    System.out.flush();
    String pwd = (new BufferedReader(new
        InputStreamReader(System.in))).readLine();
    pc.setPassword(pwd.toCharArray());
}
else if (!(callbacks[i] instanceof WSCredTokenCallbackImpl))
    throw new UnsupportedCallbackException(callbacks[i],
        "Unsupported callback");
}
}

```

Running the client with the CallbackHandler in Example 9-13 shows a challenge text-based prompt such as the following example:

```

Custom CallbackHandler
Realm : <default>
Username: viking
Password: thepwd

```

9.7 Securing the connection

As explained in 9.2, “Java client authentication protocol” on page 216, the IIOP is used when an application client accesses an EJB service by using ORB objects. However, in preparation for a request to flow between these two ORB objects, the client and server, a connection over TCP/IP transport layer must be established (IIOP over TCP/IP). When a secure connection between the client and server is required, WebSphere provides the option to encrypt the connection by using SSL (IIOP over SSL).

Securing EJB in WebSphere is discussed in 8.4.3, “RMI/IIOP transport channel protection” on page 204. For the application client, enabling IIOP over SSL involves several configuration properties in the CORBA client configuration file, for example, `sas.client.props`

- ▶ All properties under the *SSL Configuration* block. Make sure the values are synchronized with the ones specified in the server side.
- ▶ Some properties under the *CSIV2 add-on authentication protocol* block include the following properties:
 - `com.ibm.CSI.performTransportAssocSSLTLSRequired` is set to `true`. This property ensures that the client only communicates with servers that support SSL.
 - `com.ibm.CSI.performMessageIntegritySupported` and `com.ibm.CSI.performMessageConfidentialitySupported` properties are set to `true`. This setting ensures that the client can operate with different SSL encryption levels. If required, the *required* version of these properties can also be set to `true`.

See “The `sas.client.props` file” on page 220 for more information.

9.7.1 IIOP over SSL: A thin client example

We use a simple thin application client (see 9.5, “Thin application client” on page 228) to illustrate the IIOP over TCP/IP and IIOP over SSL connections between a Java application client and an enterprise bean resource.

To begin, modify or verify both the thin application client and the WebSphere Application Server where the enterprise bean resource is installed:

1. Run the `runThinClient.bat` script that is provided in this book. Make sure that you obtain a correct result. For example, when you access the `UnsecuredClient` example (choice **a**), you must see the following output:

```
Accessing unsecured Hello bean
Message from Hello bean: Hello you you UNAUNTHENTICATED (roles:
Anonymous)
```

2. Open WebSphere Administrative Console and verify that CSIV2 Inbound Transport is set to **SSL-supported** (Figure 9-6). This setting indicates that the server accepts both SSL and non-SSL connections. Note that SSL is supported but not required.

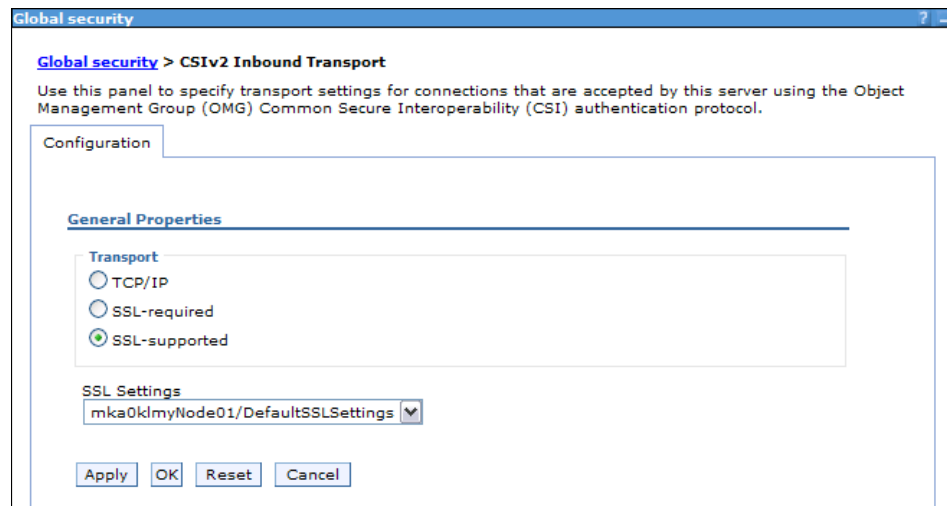


Figure 9-6 CSIV2 Inbound Transport default setup

3. Modify the `runThinClient.bat` script by changing the following line:

```
set CLIENT_TRACE=-Dcom.ibm.CORBA.CommTrace=false
```

Change this line as follows:

```
set CLIENT_TRACE=-Dcom.ibm.CORBA.CommTrace=true
```

This change enables the tracing for the thin application client example, where the trace output is in the `orbtrc.<timestamp>.txt` file.

IIOP over TCP/IP

The following example shows a thin client that does not support SSL connection, connecting to an enterprise bean resource in a server that supports (but does not require) SSL connection:

1. Edit the CORBA configuration client file in `thinClient\properties\sas.client.props` (not the one on the server). Set SSL connection properties as follows:

```
com.ibm.CSI.performTransportAssocSSLTLSRequired=false
com.ibm.CSI.performTransportAssocSSLTLSSupported=false
```

This means that the client does not support the SSL connection. With this setup, although the server supports SSL connection (not required), the connection between this client and the server is performed by using TCP/IP. Verify this by examining the trace output file.

2. Run the `runThinClient.bat` script and choose option **a**. When done, examine the `orbtrc.<timestamp>.txt` trace output file. See Example 9-14.

Example 9-14 Trace output file, a client connecting to a server using TCP/IP

```
12:02:52.303 com.ibm.rmi.ras.Trace dump:80 P=968498:0=0:CT
ORBRas[default]
...
Date:      November 17, 2004 12:02:52 PM EST
Thread Info:
RT=0:...:WSTCPTransportConnection[addr=9.42.171.128,port=2809,...
...
Date:      November 17, 2004 12:02:53 PM EST
Thread Info:
RT=1:...:WSTCPTransportConnection[addr=9.42.171.128,port=9100,...
...
Date:      November 17, 2004 12:02:55 PM EST
Thread Info:
RT=1:...:WSTCPTransportConnection[addr=9.42.171.128,port=9100,...
...

```

By looking at the trace output file in Example 9-14, you can see that the connection is kept in the TCP/IP level, as compared with the trace output shown in Example 9-15 on page 245 for an SSL connection.

IIOP over SSL

The following example shows a thin client that supports an SSL connection, connecting to an enterprise bean resource in a server that supports (but does not require) an SSL connection:

1. Similar to the steps in the previous section, edit the CORBA configuration client file in the `thinClient\properties\sas.client.props` folder.

Set the SSL connection properties as follows:

```
com.ibm.CSI.performTransportAssocSSLTLSRequired=false
com.ibm.CSI.performTransportAssocSSLTLSSupported=true
```

This setting means that the client supports the SSL connection (but is not required). Because now both the client and server support the SSL connection, whenever this client connects to the server, the connection is completed in SSL mode.

2. Run the `runThinClient.bat` script and choose option **a**. When completed, examine the `orbtrc.<timestamp>.txt` trace output file. See Example 9-15.

Example 9-15 Trace output file, a client connecting to a server by using SSL

```
11:28:54.323 com.ibm.rmi.ras.Trace dump:80 P=930648:0=0:CT
ORBRas[default]
...
Date:      November 17, 2004 11:28:54 AM EST
Thread Info:
RT=0:...:WSTCPTransportConnection[addr=9.42.171.128,port=2809,...
...
Date:      November 17, 2004 11:28:57 AM EST
Thread Info:
RT=1:...:WSSLTransportConnection[addr=9.42.171.128,port=9100,...
...
Date:      November 17, 2004 11:28:59 AM EST
Thread Info:
RT=1:...:WSSLTransportConnection[addr=9.42.171.128,port=9100,...
...

```

By looking at the trace output file in Example 9-15, you can see that the connection is switched from TCP/IP to SSL.



Securing the service integration bus

This chapter discusses securing the service integration bus during a WebSphere Application Server V6.1 configuration.

The default messaging provider is part of WebSphere Application Server V6.1. It is based on the service integration bus and supports the Java Messaging Service (JMS) 1.1 domain-independent interfaces. Communicating with the service integration bus using the default messaging provider is discussed in this section.

10.1 Messaging components of the service integration bus

The following sections discuss the various pieces of the service integration bus that work together to provide applications with messaging services, such as JMS. Figure 10-1 shows a simple messaging infrastructure using the default messaging provider in a single server.

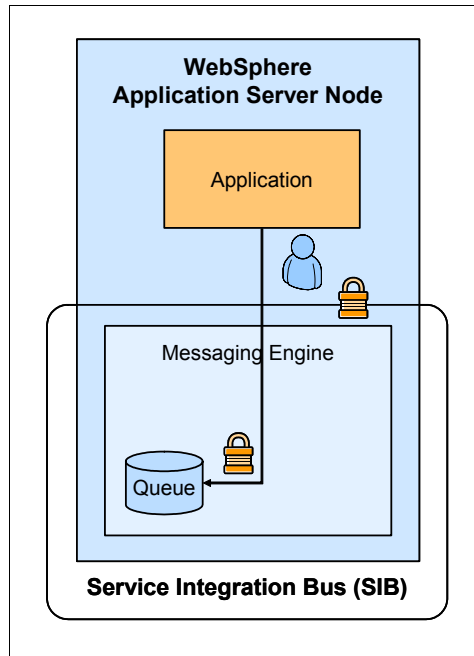


Figure 10-1 Single server messaging

Figure 10-2 illustrates a messaging infrastructure in a multi-node WebSphere Application Server Network Deployment installation.

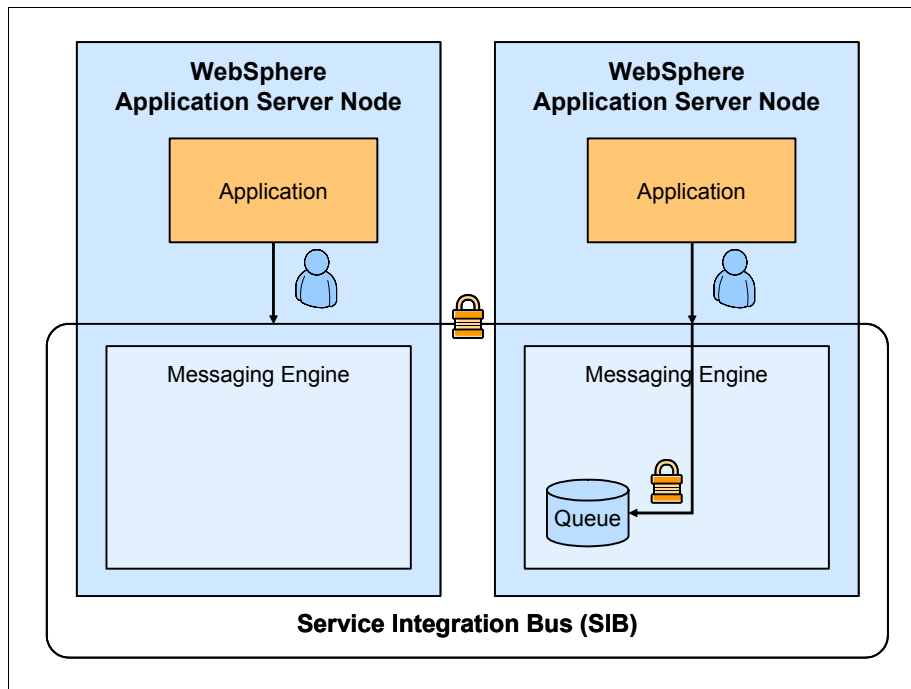


Figure 10-2 Multi-node messaging

10.1.1 Service integration bus

The service integration bus provides the basic framework for the Application Server to provide JMSs to applications. Using this framework, it is possible to connect multiple application servers into a messaging fabric.

Applications connect to the bus at specific points to send messages which are then routed among the servers and clusters connected to the bus.

10.1.2 Messaging engine

A *messaging engine* is the server component running in an Application Server that provides the messaging functionality of a service integration bus. When a server or server cluster is added to the bus as a bus member, a messaging engine is automatically created for it. Messaging engines host the bus destinations that applications send messages to and receive messages from.

10.1.3 Foreign bus

A *foreign bus* is another service integration bus or a WebSphere MQ queue manager that the local bus can communicate with. Messages can be routed to the foreign bus directly through a link between the buses or indirectly through one or more intermediary buses. For communication with WebSphere MQ, see 16.1, “Application server and WebSphere MQ” on page 452.

10.1.4 Bus destination

A *bus destination* is a virtual location within the service integration bus that applications send messages to or receive messages from. Destinations can be either permanent or temporary. Temporary destinations are used by an application during one *connection* with the service integration bus only. The destinations are the following types:

Queue	Used for point-to-point messaging.
Topic Space	Used for publish/subscribe based messaging.
Alias	An alternate name that you can use in place of the name of another destination in either the local bus or a foreign bus.
Foreign	Used to identify a destination on another bus. This allows the application to access the destination directly on a foreign bus.

Note: An application cannot receive messages from any foreign destination. An application that subscribes to a local topic space can receive messages published to a foreign topic space if the topic space names have been mapped between the local bus and the foreign bus.

10.2 An overview of service integration bus security

This section discusses the three main topics of security as related to the default messaging provider of WebSphere Application Server V6.1. Security can be enabled on the bus if administrative security has been enabled for the Application Server. Access to the bus and resources on the bus is role-based and administered through the WebSphere Application Server `wsadmin` tool.

Comparison of JMS application authentication and two messaging engines authentication: *JMS application authenticating* is used for the server to authenticate the client/user. *Two messaging engines authenticating* is for the two messaging engines to mutually authenticate each other.

10.2.1 Authentication

To access a secured bus and the resources on the bus, a set of credentials has to be supplied. After the credentials are verified, authorization to access the bus is checked. Authentication is checked between client and messaging engine, and also between messaging servers on the bus. The credentials are checked against the user registry defined during the administrative security setup for the Application Server. For administrative security information, see Chapter 3, “Administrative security” on page 49.

10.2.2 Authorization

After it is connected to the bus, the messaging engine checks roles for the destination being accessed. To access a destination on the bus, the user must first be authorized to access the bus. Membership in the Bus Connector role determines access to the bus. See 10.3, “Administering service integration bus security” on page 253, for details about changing role membership. If the user or its group does not have the Bus Connector role, connection is denied. After it is connected to the bus, the messaging engine checks roles for the destination being accessed.

Access to bus destinations are based on role membership. The bus destinations each have a set of roles which are checked based on the type of actions available for the destination. The service integration bus also has a set of default roles that apply to all local destinations on the bus. The default roles and the roles defined on the destination work together to define who can perform what action on the bus destination. For example, to send a message to a queue endpoint, the user must be a member of the Sender role for the queue endpoint or a member of the default Sender role for the bus.

Note: When a bus is initially created, a set of default permissions are granted to all authenticated users with full access to all local destinations. However, only the Server user is given the Bus Connector role. The administrator must grant the Bus Connector role to users to give them full access to the bus and its destinations.

Table 10-1, taken from the WebSphere Application Server V6.1 Information Center, lists the bus destination types and the available roles for that destination.

Table 10-1 Destination role types

Destination type	Role types
queue	Sender, Receiver, Browser, Creator
port	Sender, Receiver, Browser, Creator
webService	Sender, Receiver, Browser, Creator
topicSpace	Sender, Receiver
foreignDestination	Sender
alias	Sender, Receiver, Browser

On the bus destinations, the following default roles are defined:

- ▶ Sender
- ▶ Receiver
- ▶ Browser
- ▶ Creator

If `Topic access check required` is specified on a Topic Space destination, topic level security is enabled. After topic level security is enabled, additional authorization checks are performed when users perform actions on topics. By default, everyone has full access to topics within a topic space. Authorization roles for topics are inherited from parent topics in the topic tree. The following roles are defined for topics:

- ▶ Sender
- ▶ Receiver

For the commands to administer bus destination authorization, see 10.4, “Administering destination security” on page 257. For the commands to the administer topic level roles, see 10.5, “Administering topic space root roles and topic roles” on page 259.

Table 10-2 gives a brief description for each of the roles available.

Table 10-2 Service integration bus roles

Role type	Capability
Bus connector	Connect to the local bus.
Sender	Send a message to the destination.

Role type	Capability
Receiver	Receive (consume) a message from the destination.
Browser	Browse (view) messages on the destination.
Creator	Create a temporary destination based on the temporary destination prefix.

10.2.3 Transport security: Confidentiality

Providing credentials to the messaging engine and gaining access to the bus destination is only part of the security battle. To secure the connection between the client and the messaging engine, or between messaging engines, it is important to enforce transport encryption. This is done via SSL.

When a bus is created with the security enabled, only the transport channel chains protected by SSL are used by the bus. You can also choose to use secure transport channel chains without enabling bus security. This is achieved if you select **Restrict the use of defined transport channel chains to those protected by SSL** for the Permitted transports.

10.3 Administering service integration bus security

Access to the service integration bus is determined by user or group membership in the Bus Connector role. When both administrative security and the bus security are enabled, access to the bus is checked when a user tries to connect to a bus. By default, only Server group is assigned with this role.

The AllAuthenticated special group can be added into this role to allow all logged in users to access the service integration bus. Another special group, called Everyone, can be added into this role to allow unauthenticated users to connect to the bus. Any user or group can also be assigned to this role as required. Changing membership in the Bus Connector role can be accomplished using either the Administrative Console or the **wsadmin** tool.

10.3.1 Administering the Bus Connector role in the Administrative Console

To administer Bus Connector role:

1. In the Administrative Console, select **Service integration** → **Buses**.
2. Click the name of the service integration bus, and under Additional Properties, click **Security**.

Alternatively, instead of clicking the name of the bus, under Security, click the **Enabled** or **Disabled** link of the bus.

3. On the Bus security settings page (Figure 10-3), under Additional Properties, click **Users and groups in the bus connector role**.

The screenshot shows the 'Security for bus TEST_LOCAL_SIBUS' configuration page. It features a 'Configuration' tab and two main sections: 'General Properties' and 'Additional Properties'. In the 'General Properties' section, the 'Security' sub-section is active, showing a checked 'Enable bus security' checkbox, an 'Inter-engine authentication alias' dropdown set to '(none)', a 'Permitted transports' section with three radio button options (the second is selected), and a 'Mediations authentication alias' dropdown set to '(none)'. The 'Additional Properties' section contains two links: 'Users and groups in the bus connector role' and 'Permitted transports'. The 'Related Items' section at the bottom right lists 'JAAS - J2C authentication data' and 'Secure Administration and Applications'. At the bottom of the configuration area are buttons for 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 10-3 Security settings for service integration bus

4. On the Users and groups in the bus connector role page (Figure 10-4), perform either of the following options:
 - Grant a user or group the Bus Connector role:
 - i. Click **New**.

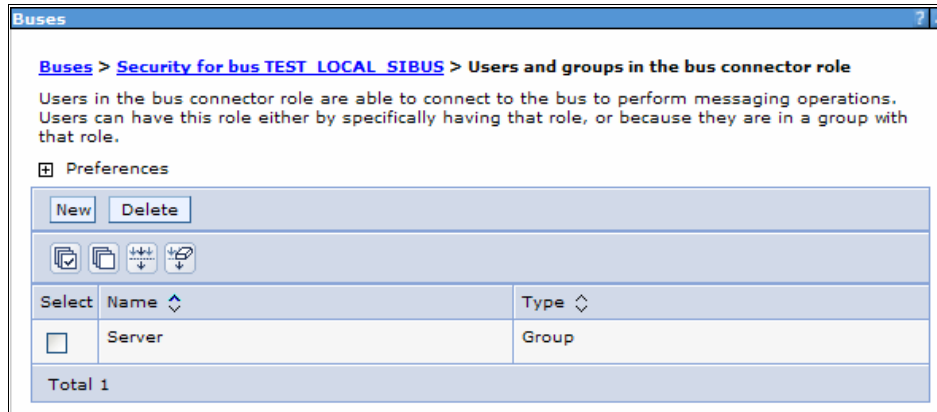


Figure 10-4 Users and groups in the bus connector role

- ii. On the New page (Figure 10-5), grant a user or group the Bus Connector role.

Buses > Security for bus TEST_LOCAL_SIBUS > Users and groups in the bus connector role > New

Create a user or group in the bus connector role.

Configuration

General Properties

Bus Connector Role

Group name

User name

Server - Allow servers to connect to the bus

All Authenticated - Allow all authenticated users to connect to the bus

Everyone - Allow unauthenticated users to connect to the bus

Figure 10-5 Create a user or group in the bus connector role

- Remove a user or a group from the Bus Connector role. Select the user or group from the list (Figure 10-4 on page 255) and click **Delete**.

10.3.2 Administering the Bus Connector role by using the wsadmin tool

Alternatively, use the following **wsadmin** commands to view and modify membership in the Bus Connector role.

- ▶ List users in Bus Connector role:
`$AdminTask listUsersInBusConnectorRole {-bus busName}`
- ▶ List groups in Bus Connector role:
`$AdminTask listGroupsInBusConnectorRole {-bus busName}`
- ▶ Add a user to Bus Connector role:
`$AdminTask addUserToBusConnectorRole {-bus busName -user username}`

- ▶ Add a group to Bus Connector role:

```
$AdminTask addGroupToBusConnectorRole {-bus busName -group groupname}
```
- ▶ Remove a user from Bus Connector role:

```
$AdminTask removeUserFromBusConnectorRole {-bus busName -user username}
```
- ▶ Remove a group from Bus Connector role:

```
$AdminTask removeGroupFromBusConnectorRole {-bus busName -group groupname}
```

10.4 Administering destination security

Access to a bus destination is based on the user or group membership in the default roles defined on the service integration bus and the various roles defined on the specific destination. These two sets of roles are combined to determine if the action is authorized for the user.

10.4.1 Default roles for bus destinations

The following default role names for bus destinations are available:

- ▶ Sender
- ▶ Receiver
- ▶ Browser
- ▶ Creator

The following commands work with the default roles for destinations on the service integration bus.

- ▶ List users in default role:

```
$AdminTask listUsersInDefaultRole {-bus busName -role roleName}
```
- ▶ List groups in default role:

```
$AdminTask listGroupsInDefaultRole {-bus busName -role roleName}
```
- ▶ Add user to default role:

```
$AdminTask addUserToDefaultRole {-bus busName -role roleName -user userName}
```
- ▶ Add group to default role:

```
$AdminTask addGroupToDefaultRole {-bus busName -role roleName -group groupName}
```

- ▶ Remove user from default role:

```
$AdminTask removeUserFromDefaultRole {-bus busName -role roleName
-user userName}
```

- ▶ Remove group from default role:

```
$AdminTask removeGroupFromDefaultRole {-bus busName -role roleName
-group groupName}
```

10.4.2 Destination specific roles

The following role names for bus destinations are available:

- ▶ Sender
- ▶ Receiver
- ▶ Browser
- ▶ Creator

The following commands set the roles for a specific local bus destination. For information about working with foreign destinations, see 16.1.5, “Administering foreign service integration bus security” on page 458 and the WebSphere Application Server V6.1 Information Center.

- ▶ List users in role:

```
$AdminTask listUsersInDestinationRole {-type destinationType -bus
busName -destination destinationName -role roleName}
```

- ▶ List groups in role:

```
$AdminTask listGroupsInDestinationRole {-type destinationType -bus
busName -destination destinationName -role roleName}
```

- ▶ Add a user to role:

```
$AdminTask addUserToDestinationRole {-type destinationType -bus
busName -destination destinationName -role roleName -user userName}
```

- ▶ Add a group to role:

```
$AdminTask addGroupToDestinationRole {-type destinationType -bus
busName -destination destinationName -role roleName -group
groupName}
```

- ▶ Remove a user from role:

```
$AdminTask removeUserFromDestinationRole {-type destinationType -bus
busName -destination destinationName -role roleName -user userName}
```

- ▶ Remove a group from role:


```
$AdminTask removeGroupFromDestinationRole {-type destinationType
-bus busName -destination destinationName -role roleName -user
userName}
```
- ▶ Override or restore the inheritance of default permissions for a destination:


```
$AdminTask setInheritDefaultsForDestination {-type destinationType
-bus busName -destination destinationName -inherit <true|false>}
```
- ▶ Override or restore the inheritance of default permissions for a destination:


```
$AdminTask setInheritDefaultsForDestination {-type destinationType
-bus busName -destination destinationName -inherit <true|false>}
```
- ▶ Determine whether a specified destination inherits default destination user roles:


```
$AdminTask isInheritDefaultsForDestination {-type destinationType
-bus busName -destination destinationName}
```

10.5 Administering topic space root roles and topic roles

The following top-level topics are within a topic space:

- ▶ Sender
- ▶ Receiver

The following commands are for topic space root roles:

- ▶ List users in a topic space root role:


```
$AdminTask listUsersInTopicSpaceRootRole {-bus busName -topicSpace
topicSpaceName -role roleName}
```
- ▶ List groups in a topic space root role:


```
$AdminTask listGroupsInTopicSpaceRootRole {-bus busName -topicSpace
topicSpaceName -role roleName}
```
- ▶ Add a user to a topic space root role:


```
$AdminTask addUserToTopicSpaceRootRole {-bus busName -topicSpace
topicSpaceName -role roleName -user userName}
```
- ▶ Add a group to a topic space root role:


```
$AdminTask addGroupToTopicSpaceRootRole {-bus busName -topicSpace
topicSpaceName -role roleName -group groupName}
```

- ▶ Remove a user from a topic space root role:
`$AdminTask removeUserFromTopicSpaceRootRole {-bus busName
 -topicSpace topicSpaceName -role roleName -user userName}`
- ▶ Remove a group from a topic space root role:
`$AdminTask removeGroupFromTopicSpaceRootRole {-bus busName
 -topicSpace topicSpaceName -role roleName -group groupName}`

The following roles are available for the topic:

- ▶ Sender
- ▶ Receiver

The following commands are for topic roles:

- ▶ List users in a topic role:
`$AdminTask listUsersInTopicRole {-bus busName -topicSpace
topicSpaceName -topic topicName -role roleName}`
- ▶ List groups in a topic role:
`$AdminTask listGroupsInTopicRole {-bus busName -topicSpace
topicSpaceName -topic topicName -role roleName}`
- ▶ Add a user to a topic role:
`$AdminTask addUserToTopicRole {-bus busName -topicSpace
topicSpaceName -topic topicName -role roleName -user userName}`
- ▶ Add a group to a topic role:
`$AdminTask addGroupToTopicRole {-bus busName -topicSpace
topicSpaceName -topic topicName -role roleName -group groupName}`
- ▶ Remove a user from a topic role:
`$AdminTask removeUserFromTopicRole {-bus busName -topicSpace
topicSpaceName -topic topicName -role roleName -user userName}`
- ▶ Remove a group from a topic role:
`$AdminTask removeGroupFromTopicRole {-bus busName -topicSpace
topicSpaceName -topic topicName -role roleName -group groupName}`
- ▶ Set or disable Sender role inheritance for a topic:
`$AdminTask setInheritSenderForTopic {-bus busName -topicSpace
topicSpaceName -topic topicName -inherit <true|false>}`
- ▶ Set or disable Receiver role inheritance for a topic:
`$AdminTask setInheritReceiverForTopic {-bus busName -topicSpace
topicSpaceName -topic topicName -inherit <true|false>}`

- ▶ Determine whether a role is inheritance Receiver for a topic:
`$AdminTask isInheritReceiverForTopic {-bus busName -topicSpace
topicSpaceName -topic topicName}`
- ▶ Determine whether a role is inheritance Sender for a topic:
`$AdminTask isInheritSenderForTopic {-bus busName -topicSpace
topicSpaceName -topic topicName}`



Part 2

Extending security beyond the application server

This part includes the following chapters:

- ▶ Chapter 11, “Security attribute propagation” on page 265
- ▶ Chapter 12, “Securing a WebSphere application using Tivoli Access Manager” on page 297
- ▶ Chapter 13, “Trust Association Interceptors and third-party software integration” on page 353
- ▶ Chapter 14, “Externalizing authorization with JACC” on page 403
- ▶ Chapter 15, “Web services security” on page 429

- ▶ Chapter 16, “Securing access to WebSphere MQ” on page 451
- ▶ Chapter 17, “J2EE Connector security” on page 463
- ▶ Chapter 18, “Securing the database connection” on page 479



Security attribute propagation

Java Authorization and Authentication Service (JAAS) provides a standard application programming interface (API) for defining pluggable authentication and Java 2 authorization extensions. Many LoginModules can be chained together using JAAS configuration files. User authentication is done by LoginModules and the authenticated user is represented by a Subject. A Subject may also own security-related attributes, which are referred to as credentials. Sensitive credentials that require special protection, such as private cryptographic keys, are stored within a private credential Set. Credentials intended to be shared, such as public key certificates, are stored within a public credential Set. WebSphere Application Server V5.1.1 and later uses JAAS for authentication. In WebSphere Application Server, LoginModules authenticate the user, create the subject, and populate it with security attributes information.

The security attribute propagation feature enables WebSphere Application Server to send security attribute information regarding the original login from one server to another server. Prior to V5.1.1, WebSphere Application Server authenticated the user and got the group information during login but passed only the identity of the user downstream. This has been significantly enhanced in Version 5.1.1 and later. This enhanced feature is called security attribute propagation using which WebSphere Application Server can now pass security attribute information, including authenticated Subject contents and other custom security attributes downstream.

These security attributes that can be transported to other Application Servers may be obtained during the initial login in the following ways:

- ▶ When WebSphere Application Server does the authentication, it can query the user registry for static security attributes such as users language preference or e-mail, and so on, and the subject is populated with these attributes.
- ▶ The security attributes may also be populated by using a custom login module in WebSphere Application Server. You can use the custom login module for populating the dynamic attributes such as users login time, location of the login, and Internet Protocol (IP) address of the original user. The custom LoginModule can insert custom security attributes in the Subject which contains the static and also the dynamic information.
- ▶ If there is an external security server, such as an IBM Tivoli Access Manager involved, the security attributes may be propagated using the appropriate Trust Association Interceptor for that reverse proxy server. The enhanced TAI++ interface is able to assert a fully populated subject which can be propagated to other servers.

Note: The custom attributes or tokens in Subject are not used by WebSphere Application Server for authentication or authorization. However, WebSphere Application Server still handles propagation of these customized tokens. WebSphere does not do serialization or deserialization of the custom tokens. The Java programming language specifies the rules for how Java code can serialize and deserialize an object. The serialization and deserialization of the custom tokens must be carried out by the implementation, and handled in the custom login module.

The importance of security attribute propagation

Security attribute propagation is useful when you want to retain and distribute some or all security attributes of the authenticated user, especially dynamic attributes such as login time and logon location. Security attribute propagation makes the JAAS Subject-based run time more useful, which can be highlighted through the use of the Reverse Proxy Server. The originating attributes are important because they define the access control list (ACL) of the originating caller throughout the down stream system. When you want to maintain the information about the originating caller identity, authenticated user strength, location, and so on, you can use the security attribute propagation feature and add these attributes to the Subject, which is then propagated downstream.

Important: Give careful consideration to the security of attribute propagation especially if you are implementing a custom token module. During planning consider the following aspects:

- ▶ The trust domain: Who sends and receives the information, can they be trusted, and how.
- ▶ Confidentiality: Making sure that only the correct parties receive the information and no one else. This can include encrypting tokens or using a secure transport. For example, enforcing Secure Sockets Layer (SSL) for Common Secure Interoperability Version 2 (CSIV2) communication.
- ▶ Integrity: Making sure that the information shared is correct and has not been tampered with by third parties.

This is not a complete list, but it does give a few points of consideration to start thinking about.

11.1 Initial Login versus Propagation Login

Before the discussion of Initial Login and Propagation Login, we define identity propagation and identity assertion.

Identity propagation refers to the low level capability of passing the users identity to another server or system. For example, if there are two systems A and B. A knows who the user is. The system A passes the identity of the user to system B. This is known as *identity propagation*. In the context of WebSphere Application Server, this means that the WebSphere Application Server A does the initial authentication, authenticates the user and creates a Subject, and then propagates the users identity to another WebSphere Application Server, which is server B, in its trust domain.

Identity assertion is the manner in which identity of the user or system is projected (or asserted) from one system to another. With respect to Identity assertion and propagation, the following considerations among others are important:

- ▶ The basis for identity propagation and assertion is the establishment of the trust relationship between systems A and B. The systems can authenticate to each other by using SSL based client certificates or by using a system password which represents a “shared secret” shared only by system A and system B.
- ▶ Strong network protection is a must while doing identity assertion. It is important that intruders are unable to attack the system from within the

network and then take advantage of the identity assertion trust relationship. Thus, for example, if a password is used for authenticating, network protection must be in place to protect that password.

- ▶ When asserting identities, the identities must be the same in the registry of system A or system B. If the registries are not the same, some sort of identity mapping has to be done which complicates things more. We do not discuss the scenario when the identities are different in this book.

When WebSphere Application Server authenticates a request, it first checks to see if the authentication must occur using initial login or a propagation login. An *initial login* is the process of WebSphere Application Server authenticating the user information. Typically the user proves his identity through a credential which may be a user ID and a password, or a certificate, and WebSphere Application Server then validates the user against the user registry and looks up secure attributes that represent the user access rights.

Propagation login is the process of validating the user information, typically an Lightweight Third Party Authentication (LTPA) token, and then deserializing a set of tokens that constitute both custom objects and token objects known to the WebSphere Application Server. For example, when the user identity is propagated from WebSphere Application Server on system A to WebSphere Application Server on system B, WebSphere Application Server B does a propagation login to validate the tokens typically the LTPA token it received from the WebSphere Application Server A to ensure that its a valid LTPA token.

11.2 Token framework

WebSphere Application Server provides a token framework to enable populating the JAAS Subject with Java objects and to provide the serialization functionality for those objects. The token framework is able to identify the uniqueness of the token contained in the Authenticated Subject. This uniqueness of the token determines how the Subject gets cached and the purpose of the token. This uniqueness of the token also determines how the token gets recreated when the Subject is lost.

The Token framework is useful in propagating custom security attributes downstream. WebSphere Application Server Token framework defines four token interfaces that enable the WebSphere Application Server run time to determine how to propagate the token. All of the token types defined by the propagation framework have similar interfaces. Basically, the token types are *marker interfaces* (marker interface is a Java interface that does not actually define any fields and is just used to “mark” Java classes) that implement the

com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods.

The following tokens are provided by the WebSphere Application Server Token framework:

Authorization token This token is user specific and it contains the authorization related security attributes for the authenticated Subject. It is used by WebSphere Application Server to make Java 2 Platform, Enterprise Edition (J2EE) authorization decisions

Single sign-on (SSO) token

A Single Sign-On token is also a user specific token that is added to the JAAS Subject. It enables WebSphere Application Server to do Single Sign-On to other WebSphere Application Servers. It is added to the response as a Hypertext Transfer Protocol (HTTP) cookie and sent to the browser and represents unique authentication. The default value of this token is the LTPA Token Version 2. The LTPA Token Version 2 is significantly enhanced compared to the previous LTPA Token version.

Propagation token

The propagation token is not a user specific token and, therefore, it is not stored in the Subject. Instead, the propagation token is stored on the thread context. The default propagation token records all user switches and host switches.

Authentication token

The authentication token contains the identity of the user. This token is equivalent to the LTPA token in previous versions. This token type is typically reserved for internal WebSphere Application Server purposes. The Authentication Token is added to the HTTP Response as an LTPA Token cookie to maintain backward compatibility with previous versions. Table 11-1 illustrates the Token framework.

Table 11-1 Token framework

Token Name	Interface	Subject based or Thread based	Notes
Authorization token	com.ibm.wsspi.security.token.* com.ibm.wsspi.security.token. .AuthorizationToken	Based on authenticated Subject	Propagated downstream

Token Name	Interface <code>com.ibm.wsspi.websphere. security.token.*</code>	Subject based or Thread based	Notes
Single Sign-On token	<code>com.ibm.wsspi.security.token .SingleSignonToken</code>	Based on authenticated Subject	Sent to the browser as a cookie named <code>LtpaToken2</code> by default. Propagated downstream
Authentication token	<code>com.ibm.wsspi.security.token .AuthenticationToken</code>	Based on authenticated Subject	Exists for backward compatibility. Has the old <code>LtpaToken</code> for backward compatibility. Propagated downstream.
Propagation token	<code>com.ibm.wsspi.security.token .PropagationToken</code>	Based on the thread and not based on Subject.	Propagated downstream

Important: Any custom tokens that are used in this framework are not used by WebSphere Application Server for authorization or authentication. The framework serves as a way to notify WebSphere Application Server that you want these tokens propagated in a particular way.

WebSphere Application Security run time uses the tokens in the following situations only:

- ▶ Call the `getBytes` method for serialization.
- ▶ Call the `getForwardable` method to determine whether to serialize the authentication token.
- ▶ Call the `getUniqueId` method for uniqueness.
- ▶ Call the `getName` and the `getVersion` methods for adding serialized bytes to the token holder that is sent downstream.

11.3 Custom implementation of tokens

Each of the WebSphere Application Server tokens discussed previously can be customized by implementing the appropriate interface. You can perform the customization in the following two ways:

- ▶ You can add custom attributes to the default token.
- ▶ You can create your own implementation of the token by extending the specific Token Interface.

First, carefully consider the requirement that you must implement a custom token. In most cases, you can add custom attributes to the default token and retrieve them in your application code. You must also carefully consider writing your own implementation if you want to accomplish one of the following tasks:

1. Isolate your attributes within your own implementation.
2. Serialize the information using custom serialization, which means, your java code must be able to serialize and deserialize the token. If you are using the default token that WebSphere Application Server provides, then WebSphere Application Server takes care of this for you. Make custom decisions based on the information in the customized token at the appropriate time.
3. You might need to use custom encryption and decryption for tokens.

Adding custom attributes to the default token is usually sufficient for propagating the user or non-user specific attributes. Writing custom implementations is usually for Service Providers to enable them to provide custom services.

11.3.1 Writing custom implementations of tokens

To modify the default implementation of the tokens:

1. If you plan to implement more than one token type, consider creating an abstract class that implements the *com.ibm.wsspi.security.token.Token* interface. All of your token implementations might extend the abstract class and then most of the work is completed. However, if there are considerable differences between how you handle the various token implementations, you can implement the interface directly.
2. If you must implement a custom token interface, ensure that the methods required by the specific token that you are trying to implement are implemented. When the custom token object is added to the Subject, it does affect the cache lookup of the Subject if you return something in the *getUniqueID()* method. Therefore, when you are implementing a custom token, you must ensure that the *getUniqueID* method returns either null or a unique token.
3. After you implement the specific token interface, place your compiled code in the *WebSphere_Root/classes* directory. Alternatively, you can place the class in any private directory. If you place it in any private directory, add the Java archive (JAR) file or the directory that contains your code into the *server.policy* file so that it has the necessary permissions that are required by the server code. The preferred directory to place any custom JAR files is *WebSphere_Root/lib/ext*.
4. Write a JAAS login module so that the customized tokens are added and received and processed properly during WebSphere Application Server logins. You may make your specific token a read-only in the commit phase of

the login module. If you make the token a read-only, you cannot add additional attributes to the token within your applications. For further information about implementing JAAS login modules, look at Chapter 5, “JAAS for authentication in WebSphere Application Server” on page 85.

5. Add the JAAS login module to the specific application and system login configurations. You can also add the implementation from an application. However, to deserialize the information, you must still plug in a custom login module, so that when the token is propagated, the WebSphere Application Server logins receive the serialized version of the custom token.
6. In most cases while implementing a custom JAAS LoginModule, you might add your custom login module, after the *com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule* for receiving serialized versions of your custom token. The *com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule* is in the following JAAS aliases:

WEB_INBOUND, RMI_INBOUND, DEFAULT

Add this login module to any of the application logins where you might want to generate your custom token.

11.3.2 Common token functionality

If you are implementing more than one token type, it can be beneficial to implement an Abstract class that all the tokens extend. This method is used in the sample code in the additional material. To access additional material, see Appendix B, “Additional material” on page 543. While there is functionality that is shared, there are also certain parts of functionality that you must implement within the subclasses, especially with regards to encryption, signatures, and cloning. The following is a list indicates what you can share and cannot share:

- ▶ Shared functionality:
 - Basic validation techniques such as expiration
 - The fundamental data structure such as using a hash table to store the user data
 - Basic construction
 - Most administration functions such as `addAttribute()`, `getAttributes()`, `getAttributeNames()`, `isForwardable()`, `getExpiration()`, `getPrincipal()`, `setReadOnly()`, `getVersion()`, and perhaps `getUniqueID()`.

- ▶ Functionality that must be token specific:
 - The encryption, signing, and serialization of tokens for the `getBytes()` method
 - The decryption, signature validation and deserialization of tokens when the `byte[] token_bytes` constructor is called
 - Any advanced construction

Important: Do not start your Java package names for JAAS modules with `com.ibm.ws.security.server`, because this causes problems when WebSphere is running them.

Note: During development of these tokens, place the `WebSphere_Root/plugins/com.ibm.ws.runtime_6.1.0.jar` file into the class path. This has changed since WebSphere Application Server 6.0.x because of WebSphere's further adoption of OSGi.

Example 11-1 provides snippets of the token class for your review. For readability reasons, most of the methods have their code removed (denoted by an ellipses (...)).

Example 11-1 Snippets of the abstract token class

```
package com.itso.was61sec.customtokens;

import com.ibm.wsspi.security.token.Token;

public abstract class AbstractCustomToken implements Token {

    java.util.Hashtable hashtable = new java.util.Hashtable();
    private byte[] tokenBytes = null;
    // 2 hours in millis, by default
    private static long expire_period_in_millis = 2 * 60 * 60 * 1000;
    private static short tokenVersion = 1;

    /**
     * Constructor used to create initial AuthorizationToken instance
     */
    public AbstractCustomToken(String principal) {
        // Sets the token version
        addAttribute("version", new Short(tokenVersion).toString());
        // Sets the token expiration
        addAttribute("expiration", new Long(System.currentTimeMillis())
```

```

        + expire_period_in_millis).toString());
    }

    /**
     * Constructor used to deserialize the token bytes received during a
     * propagation login.
     */
    public AbstractCustomToken(byte[] token_bytes) {
        // Since the first step may or may not be to decrypt it
        // we can't do much here
    }

    public boolean isValid() {
        long expiration = getExpiration();

        // if you set the expiration to 0, it does not expire
        if (expiration != 0) {
            // return if this token is still valid
            long current_time = System.currentTimeMillis();

            boolean valid = ((current_time < expiration) ? true : false);
            System.out.println("isValid: returning " + valid);
            return valid;
        } else {
            System.out.println("isValid: returning true by default");
            return true;
        }
    }

    public long getExpiration() { ... }

    public boolean isForwardable() { ... }

    public String getPrincipal() { ... }

    abstract public byte[] getBytes();

    abstract public String getName();

    public short getVersion() { ... }

    public String getUniqueID() {
        // if you don't want to affect the cache lookup, just return NULL here.
        // return null;

        String cacheKeyForThisToken = "dynamic attributes";
    }

```

```
// if you do want to affect the cache lookup, return a string of
// attributes that you want factored into the lookup.
return cacheKeyForThisToken;
}

public void setReadOnly() { ... }

public String[] getAttributes(String key) { ... }

public String[] addAttribute(String key, String value) { ... }

public Enumeration getAttributeNames() { ... }

abstract public Object clone();
```

11.3.3 Interaction of the login module and the token modules

The following generic workflow indicates how a custom token module is handled in a WebSphere Application Server:

1. When a user connects to a WebSphere Application Server, the corresponding JAAS login chain is run and the custom login module is run.
2. The login module must go through all the tokens that the client has and look for any of the corresponding token type. If a token is present then the user has already logged in previously and it is considered a propagation login.
3. The token object is now constructed. Depending on the login the construction of the token is different. In a case where a propagation login is occurring, the `byte[] token_bytes` constructor of that token type is called. In the initial login case, a token object is constructed using the `String principal` constructor and information is gathered from the default authentication token.
4. The token is added to the Subject associated with the request and you can now use it. It can be especially useful in conjunction with custom JACC providers which can extract information from the tokens.
5. When WebSphere Application Server wants to serialize the token to send it somewhere, the `getBytes()` method is called. This method must serialize itself and perhaps sign and encrypt the serialized token.

11.3.4 Authorization token

The authorization token contains most of the user's information. It contains the authorization-related security attributes that are propagated through the Subject. The WebSphere Application Server authorization engine uses the default authorization token to make J2EE authorization decisions.

Default authorization token

You can use the default authorization token when you want to add security attributes that get propagated downstream. These security attributes must be specific to the user associated with the authenticated Subject. If they are not specific to the user, consider adding them in the propagation token that we discuss later.

Adding custom attributes to the default authorization token

To add attributes into the default AuthorizationToken, use a custom JAAS login module. This custom login module must be configured in the WEB_INBOUND JAAS login module configuration (system login). Two login modules are defined for the WEB_INBOUND JAAS alias:

- ▶ `com.ibm.ws.security.server.lm.ltpaLoginModule`
- ▶ `com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule`

Figure 11-1 shows the two configurations.

The screenshot displays the configuration page for JAAS login modules. The breadcrumb path is: [Secure administration, applications, and infrastructure](#) > [JAAS - System logins](#) > [WEB_INBOUND](#) > [JAAS login modules](#). Below the breadcrumb, there is a note: "Each entry in the login configuration must contain at least one login module. However, you can define more than one login module for a login configuration. If you define more than one login module for a login configuration, they are processed in the order that they are defined." There is a "Preferences" section with a plus icon. Below that are buttons for "New", "Delete", and "Set Order". A toolbar contains icons for selection, copy, paste, and refresh. The main table lists the configured login modules:

Select	Module class name	Authentication strategy	Module order
<input type="checkbox"/>	com.ibm.ws.security.server.lm.ltpaLoginModule	REQUIRED	1
<input type="checkbox"/>	com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule	REQUIRED	2
<input type="checkbox"/>	com.itso.was61sec.loginmodule.CustomLoginModule	REQUIRED	3

Total 3

Figure 11-1 JAAS login configurations

You can insert the custom login module after the `com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule`, by using a higher module order number in the definition.

First login occurs in `ltpaLoginModule` and after that a default `AuthorizationToken` is created in the `wsMapDefaultInboundLoginModule`. As a third step, your custom login module can add custom attributes to the authorization token.

Custom authorization token

When must you implement a custom authorization token?

The default `AuthorizationToken` is sufficient for propagating attributes that are user-specific. However, you can write a custom authorization token implementation if you want to accomplish one of the following:

- ▶ Isolate your attributes within your own implementation.
- ▶ Serialize the information using custom serialization. You must deserialize the bytes at the target and add that information back on the thread. This task also might include encryption and decryption.
- ▶ Affect the overall uniqueness of the Subject using the `getUniqueID()` method.
- ▶ If you want to make custom authorization decisions using the information in the token at the appropriate time.

Implementing a custom authorization token

To implement a custom authorization token:

1. Implement the `com.ibm.wsspi.security.token.AuthorizationToken` interface. The sample code shown in Example 11-2 extends from an abstract class. See Appendix B, “Additional material” on page 543 to obtain the sample code. If you are only implementing an authorization token, it might be easier to implement the interface directly.

Example 11-2 Snippets of the sample implementation of an authorization token

```
package com.itso.was61sec.customtokens;

import com.ibm.wsspi.security.token.AuthorizationToken;

public class CustomAuthorizationTokenImpl extends
AbstractCustomToken implements AuthorizationToken {

    public CustomAuthorizationTokenImpl(byte[] token_bytes) {
        super(token_bytes);
        // The reverse of what we did in getBytes must be done here.
        try {
```

```

        // If you encrypted the token then you would decrypt it
        // here. We didn't encrypt it, so we can just deserialize.
        hashtable = (java.util.Hashtable)
com.ibm.wsspi.security.token.WSOpaqueTokenHelper
        .deserialize(token_bytes);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public CustomAuthorizationTokenImpl(String principal) {
    super(principal);
    // Sets the principal in the token
    addAttribute("principal", principal);
}

public byte[] getBytes() {
    // get bytes goes through the following flow, depending on how
    // secure and trusted you want the token.
    // 1. Serialize user data
    // 2. Sign serialized data
    // 3. Encrypt data
    // The four token types often have different levels of
    // security during transport so getBytes should probably
    // be a subclass specific function.
    if (hashtable != null) {
        try {
            // Do this if the object is set to read-only during login
            // commit,
            // because this makes sure that no new data gets set.
            // You can deserialize this in the downstream login module using
            // WSOpaqueTokenHelper.deserialize()
            if (isReadOnly() && getTokenBytes() == null)
                setTokenBytes(com.ibm.wsspi.security.token.WSOpaqueTokenHelper
                    .serialize(hashtable));

            // You could encrypt the token's bytes here, but
            // we will just pass them back unencrypted
            return getTokenBytes();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}

```



```
        System.out.println("getBytes: returning null");
        return null;
    }
}
```

2. Write a custom JAAS login module that adds and receives the custom `AuthorizationToken` during WebSphere Application Server login. You can see an example of a custom JAAS login module in Example 5-1 on page 90.
3. Add the custom login module to the application and system login configurations that already contain the `com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule` for receiving serialized versions of your custom authorization token. Add your custom login module after `com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule`, because the custom `AuthorizationToken` implementation relies on the information `wsMapDefaultInboundLoginModule` adds.

11.3.5 Single Sign-On token

The default Single Sign-On token is used by the WebSphere Application Server runtime code only. It is added to the authenticated Subject and also added to the HTTP response as an HTTP cookie.

Default Single Sign-On token

WebSphere Application Server defines a default `SingleSignonToken` with the name of `LtpaToken` and the Version 2. The name and version together form the cookie name and therefore the cookie name added is `LtpaToken2`. There are size limitations for this token when it is added as an HTTP cookie and therefore be careful about adding extra attributes to this token.

We recommend that any time you use cookies, use the Secure Sockets Layer protocol to protect the request. Web users can use an SSO token to authenticate once when they are accessing Web applications across multiple WebSphere Application Servers.

Custom Single Sign-On token

You can implement your own custom SSO token which adds an HTTP response as an HTTP cookie. Consider writing your own implementation of the Single Sign-On token if you want to accomplish one of the following tasks:

- ▶ Separate your attributes within your custom implementation.
- ▶ Use custom serialization, or custom encryption/decryption.
- ▶ Check the uniqueness of the subject using the `getUniqueID()` method.

Keep in mind the following guidelines while implementing your custom Single Sign-On token:

- ▶ HTTP cookies have a size limitation, therefore do not add too much data to this token.
- ▶ This cookie is not used and nor is handled by the WebSphere Application Server run time.

The following steps explain the process for developing a custom Single Sign-On token:

1. Write your custom token properly. The sample code in additional properties extends a base token class. You can implement the *com.ibm.wsspi.security.token.SingleSignonToken* interface directly.
2. Add the class to WebSphere_Root/classes or place it into a JAR file and then into WebSphere_Root/lib/ext. Make sure that you add this directory or the JAR file to the server.policy file so that WebSphere Application Server can load your classes.
3. Write the JAAS login module that creates and adds your tokens properly during WebSphere Application Server logins. You can see an example of a custom JAAS login module in Example 5-1 on page 90.
4. Add your JAAS login module to WebSphere Application Server system login configurations that contain the *com.ibm.ws.security.server.Im.wsMapDefaultInboundLoginModule* for receiving serialized versions of your custom propagation token.

11.3.6 Propagation token

The propagation token is not user specific and thus not part of the Subject. A default PropagationToken is stored on the thread of execution for applications. WebSphere Application Server propagates this PropagationToken downstream and the token stays on the thread context. When a request is sent outbound to another server, the propagation token on that thread is sent with the request and the token is executed by the target server.

Default propagation token

The default propagation token does the following tasks:

1. It monitors and logs all user switches and host switches. The token data must be available from within the container of any resource where the PropagationToken lands. Remember that you must enable the propagation feature at each server where a request is sent in order for propagation to work.

2. There is a `WSSecurityHelper` class that has APIs for accessing the `PropagationToken` attributes and for adding custom attributes to the propagation token in your application code.
3. After you add attributes to the `PropagationToken`, you cannot change these attributes. This enables the WebSphere Application Server security run time to add auditable information and have that information remain there for the life of the invocation. Any time that you add an attribute to a specific key, an `ArrayList` is stored to hold that attribute. The order of the attributes added is preserved. The first element in the `String Array` returned is the first attribute added for that specific key.
4. In the default `PropagationToken`, any data changes to the token is recorded using a change flag. These changes are tracked to enable WebSphere Application Server to know when to re-send the authentication information downstream so that the downstream server has those changes. A CSIV2 session is maintained between servers for an authenticated client. Whenever the `PropagationToken` changes, a new CSIV2 session is generated and a new authentication occurs. Therefore, if there are frequent changes to the `PropagationToken` during a method, it causes frequent downstream calls which may impact performance.
5. Whenever the `PropagationToken` is propagated either horizontally or downstream, the name of the receiving Application Server is logged into the `PropagationToken`. The format for each server in the list is “Cell:Node:Server”, which provides you access to the cell name, node name, and server name of each Application Server that receives the invocation.
6. You can also get the caller list from the `PropagationToken`. Anytime an authenticated `Subject` is generated, it is logged in the token. Basically, whenever an authenticated user is set on the thread, the user is logged in the default `PropagationToken`. At times, the same user might be logged in multiple times if the `RunAs` user is different from the caller.

Adding custom attributes to the default propagation token

You can add custom attributes to the default `PropagationToken` for application usage. This token is transported along with the request to downstream servers so that the attributes are available in your downstream Enterprise JavaBeans (EJB) or in your Application Servers when they are required.

Keep in mind the following considerations to add attributes when you use the default `PropagationToken`:

- ▶ When you add information to the `PropagationToken`, it affects CSIV2 session caching. Add information sparingly between remote requests.
- ▶ After you add information with a specific key, the information cannot be removed.

- ▶ You can add as many values to a specific key as you require. However, all of the values are returned as a string array in the order they were added. Therefore, keep track of values added and their sequence.
- ▶ The `PropagationToken` is available only on servers where security attribute propagation is enabled and WebSphere application security is enabled.
- ▶ An application cannot use keys that begin with either `com.ibm.websphere.security` or `com.ibm.wsspi.security`. These prefixes are reserved.

Implementing a custom propagation token

The default `PropagationToken` is typically sufficient for propagating attributes that are not user-specific. Consider writing your own implementation if you want to do the following tasks:

- ▶ Isolate your attributes within your own implementation.
- ▶ Use custom serialization.
- ▶ Use custom encryption and decryption for your tokens.

To implement a custom propagation token:

1. Code your implementation of the `PropagationToken` interface. The sample code implements `com.ibm.wsspi.security.token.PropagationToken`. You can download the sample from the additional materials.
2. Add the class to `<WebSphere_root>/classes`, then add the JAR file to the `server.policy` file so that WebSphere Application Server can load your classes.
3. Write the JAAS login module that creates and adds your tokens properly during WebSphere Application Server logins.
4. Add your JAAS login module to WebSphere Application Server system login configurations that contain the `com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule` for receiving serialized versions of your custom propagation token.

The `com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule` is used in the following JAAS login module configurations, which are `WEB_INBOUND`, `RMI_INBOUND`, `DEFAULT`. You can also add this login module to any of the application logins where you might want to generate your custom `PropagationToken` and store it on the thread context during the login.

11.3.7 Authentication token

Because the name indicates the authentication token, it contains the authentication information of the user. The authentication token serves the same function as the old LTPA token in earlier versions of WebSphere Application Server (prior to V5.1.1). The default authentication token is reserved for WebSphere Application Server run time and is authentication-mechanism specific. The Single Sign-On token is the new token format and the authentication token just serves the purpose of backward compatibility. Any modifications to this token by custom code can potentially cause interoperability problems.

11.3.8 Changing the token factory associated with the default token

When WebSphere Application Server generates the default tokens, it uses an appropriate TokenFactory class for creating the default tokens. This token factory class is specified using a custom property in the WebSphere Application Server. To view the token class that is used by default for the tokens:

1. Launch the WebSphere Application Server Administrative Console and log in.
2. Select **Security** → **Secure administration, applications, and infrastructure**.
3. On the right side of the page, in the Authentication section, select **Custom properties**. Among the various properties that you can set, you see a list of token factories. You can use the filter function of this view to narrow the list to only token properties.

You can plug in your own custom TokenFactory class implementation. Locate the specific token factory you want to modify. Associate your custom token factory implementation class with the TokenFactory property value. Also, verify that your implementation classes are available for the WebSphere Application Server classloader.

If you must perform your own signing and encryption of the default token, implement the following classes:

- ▶ `com.ibm.wsspi.security.ltpa.Token`
- ▶ `com.ibm.wsspi.security.ltpa.TokenFactory`

You can use the LTPA keys or you can use your own keys for instantiating and validating your token implementation. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using these keys.

► Authorization token, propagation token

For both these tokens, the default TokenFactory used is called *com.ibm.ws.security.ltpa.AuthzPropTokenFactory*. This token factory encodes the data, but does not encrypt the data in the AuthorizationToken. This is because the AuthorizationToken is transmitted over CSIV2 using SSL and therefore there is no requirement to encrypt the token. If you require additional security for the AuthorizationToken, you can associate a different TokenFactory implementation with this property to get encryption.

For example, if you associate *com.ibm.ws.security.ltpa.LTPAToken2Factory* with this property, the token uses an encryption called the Advanced Encryption Standard (AES). However, there may be a performance impact with the encryption.

► Single Sign-On token

Single Sign-On token, by default, uses the *com.ibm.ws.security.ltpa.LTPAToken2Factory* class, which creates the token LtpaToken2. This TokenFactory uses the AES/CBC/PKCS5 Padding cipher for encoding.

Note: If you change this TokenFactory, you lose the interoperability with any servers running a version, prior to V5.1.1, of WebSphere Application Server that use the default TokenFactory. Only servers running WebSphere Application Server V5.1.1 or later with propagation enabled are aware of the LtpaToken2 cookie.

► Authentication token

The default TokenFactory for authentication token is called *com.ibm.ws.security.ltpa.LTPATokenFactory*. The LTPATokenFactory uses the DESede/ECB/PKCS5Padding cipher. This token factory creates an interoperable LTPA token.

Note: If you modify this TokenFactory, you lose the interoperability with any servers running a version of WebSphere Application Server prior to Version 5.1.1 and any other servers that do not support the new TokenFactory implementation.

If you associate `com.ibm.ws.security.ltpa.LTPAToken2Factory` with the `com.ibm.wsspi.security.token.authenticationTokenFactory` property, the token is encrypted using AES. However, you must weigh the performance against your security requirements.

11.4 Horizontal propagation

In horizontal propagation, the Subject containing the security attributes are propagated amongst the front-end WebSphere Application Servers. The default Single Sign-On token is LTPAToken Version 2. You can create your own custom token and add that to the Subject in a custom login module. The token contains the following information:

- ▶ The users unique ID
- ▶ Timestamp
- ▶ The key to lookup the serialized security attributes
- ▶ The originating servers' Java Management Extensions (JMX™) administration endpoint which tells the receiving server how to communicate with it

During the WebSphere Application Server initial login process, the Single Sign-On token is added to the Subject and the token is added to the HTTP response as a cookie. This login process can also be customized to add custom information to the Single Sign-On token or to the Subject by using JAAS LoginModules. If you have horizontal propagation enabled, it enables the front-end receiving servers to retrieve the Subject information and extract the security attributes information from the Subject. In this case, initial login occurs at the originating server and propagation login occurs at the receiving servers.

11.4.1 Horizontal propagation using Dynacache

When WebSphere Application Servers are configured in a cluster and in the same Distributed Replication Service (DRS) domain, the Application Server propagates the serialized information to all the servers within the same domain. The following actions happen during horizontal propagation by using Dynacache (Figure 11-2):

1. Server 1 and Server 2 are members of the same DRS domain. Application1 is deployed on Server 1 and Server 2. Assume the user is logged in on Server 1. During the initial login process on Server 1, a fully populated JAAS Subject containing the tokens is created and placed in Dynacache. The Single Sign-On token is created and placed on the HTTP response as a cookie.
2. Dynacache is replicated in the DRS domain.

3. An HTTP request from application1 on Server 1 makes another call to application1 on Server 2. The original login attributes are found on Server 2 without additional remote requests. This is because the Single Sign-On token is passed to Server 2 by using a cookie.
4. WebSphere Application Server security searches for authentication information, using the Single Sign-On token as the key. It first searches in the local security cache for the Subject. Because this login is done on Server 1, the subject is instantiated on Server 1. Hence, the local security cache on Server 2 does not have the instantiated subject. Then WebSphere Application Server security searches in Dynacache for tokens. Because Server 2 is in the same DRS domain, the tokens are found in the Dynacache.

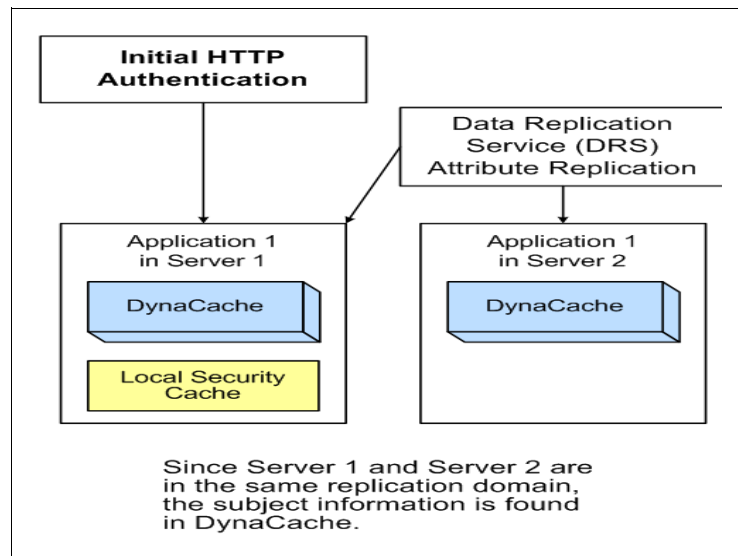


Figure 11-2 Horizontal propagation sample

The Lifetime of Dynacache entry is the same as the LtpaToken lifetime (120 minutes by default).

11.4.2 Horizontal propagation using JMX

Horizontal propagation can be accomplished by using the JMX infrastructure. In Figure 11-3 on page 288, Server 1 and Server 2 are configured in the same Data Replication Service Domain. Server 3 and Server 4 are configured in a separate Data Replication Service Domain.

The process is explained as follows:

1. The request originates from application1 on Server 1 (or Server 2). During the initial login, a fully populated Subject is created and put in DynaCache which gets replicated by DRS to all the servers within the DRS domain.
2. The request is redirected to application2 on either Server 3 or Server 4. Server3 gets the Single Sign-On token from Server 1. It uses the Single Sign-On token as a key, and checks the DynaCache for the serialized information. The serialized information is not found in the DynaCache because the Server 3 and Server 4 are not configured in the same DRS domain. As a result, a secure remote JMX request is sent back to the originating server (Server 1), that hosts application1 to obtain the Subject information. Server 1 sends the serialized information to Server 3.
3. Server3 is able to deserialize the Subject and decrypt the Tokens to get the security attribute information. This results in a propagation login by Server 3.

Use of JMX across cells: If you are using JMX *across cells*, significant trust is implied between the cells. In addition to the requirement for shared LTPA encryption keys, the cell level server identities end up with substantial authority across the cell boundaries. This is because as with any administrative calls, the JMX call requires authentication and authorization. Looking at Figure 11-3 on page 288 and assuming that Servers 1 and 2 make one cell and Servers 3 and 4 make another, when Server 3 makes a call, it must send its server user ID and password to server 1. Server A, then, validates this password and ensures that the user ID has administrative authority to its cell, which has significant implications. This means that for *cross cell* Web layer (called horizontal) subject propagation to work, it must have the following:

- ▶ The receiving server (Server 3) must send its administrative secret password to server 1. Server 1, therefore now knows the server user ID and password for Server 3's cell, and that ID has full administrative authority.
- ▶ Server 1's cell must grant administrative authority to Server B's server ID. Server B, thus, has administrative authority over server 1's cell.

Ultimately both cells now completely trust each other. Each has administrative authority over the other. The same behavior holds with propagation within a cell, but in that case there is no issue because servers within a cell already trust each other and share a common administrative identity.

Note that this does not apply when downstream propagation occurs using Internet Inter-ORB Protocol (IIOP). In that case the upstream server simply sends the subject to the downstream server. No JMX callbacks are required.

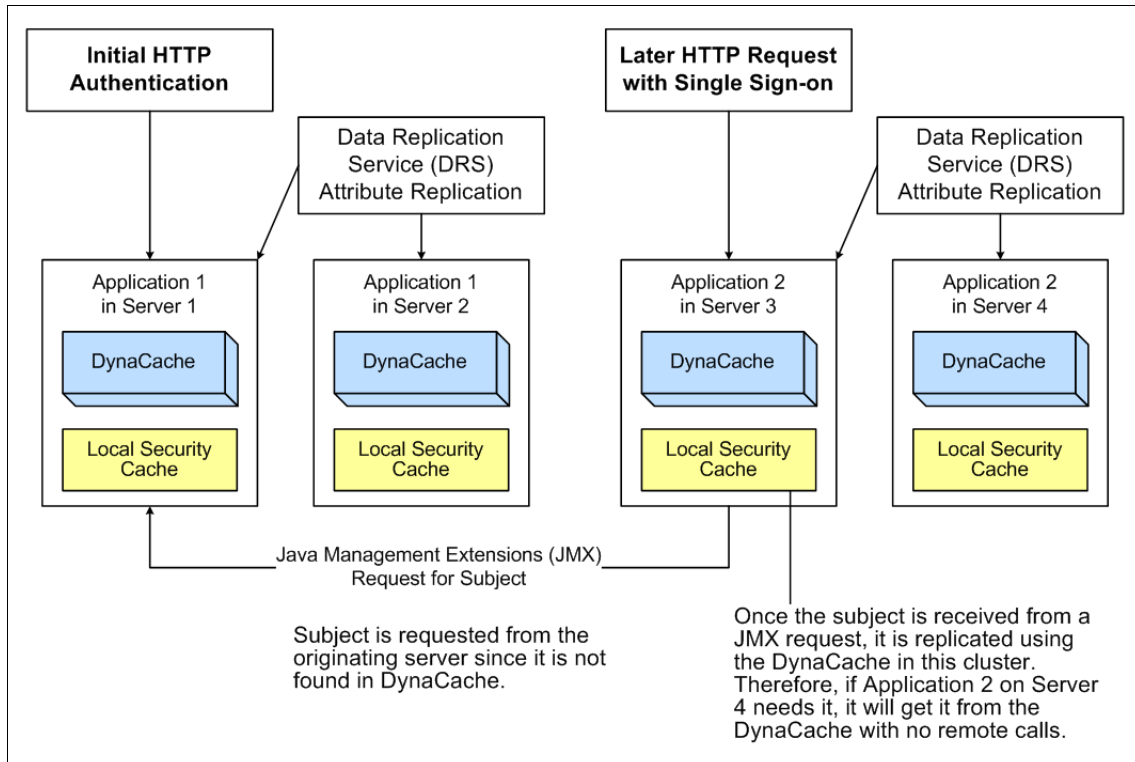


Figure 11-3 Horizontal propagation using JMX

By using a single JMX remote call back to the originating server, the following benefits are realized:

- ▶ You get the login information from the original server.
- ▶ You are not required to perform any User registry calls because the Application Server can regenerate the subject from the serialized information.
- ▶ After the Server 3 gets the serialized information, it regenerates the Subject and also puts that in the DynaCache for subsequent horizontal propagation using dynacache for its DRS domain.
- ▶ If the JMX Call fails for some reason, WebSphere Application Server falls back to an initial login. In this scenario, the login modules are called and the Subject is recreated.

Note: For the remote JMX Administration call across the cell to succeed, the two servers must share common security infrastructure, registries, SSL Keys, and so on. Also the cell's security server ID has administrator access to the remote cell.

Earlier versions of WebSphere Application Server supported Single Sign-On from Server 1 to Server 2 or Server 3 using the LTPA token. Ever since WebSphere Application Server V6.0, this has been supported using the Single Sign-On token. This means that if you do not enable Web propagation, the Single Sign-On still works using the Single Sign-On token which gets sent as a cookie to the browser and to the servers of WebSphere Application Server. The information contained in the Single Sign-On token enables the servers of WebSphere Application Server to perform SSO. By enabling horizontal propagation, you can pass the complete Subject to other front-end WebSphere Application Server profiles.

There are some performance implications of enabling horizontal propagation. Enabling Web inbound propagation eliminates some user registry calls. However, the deserialization and the decryption of tokens are processing intensive tasks and may impact performance. We recommend that you run performance tests in your environment with typical number of users, with the propagation enabled and with propagation disabled to determine the implications.

11.5 Downstream propagation

Previously there were two authentication protocols supported by IBM Secure Association Service (SAS) is the authentication protocol used by all previous releases of the WebSphere product. SAS is deprecated and it is maintained for backwards compatibility. The Object Management Group (OMG) has defined an authentication protocol called CSIV2 so that vendors can interoperate securely. CSIV2 is implemented in WebSphere Application Server with more features than SAS and is considered the strategic protocol. In fact, the only time the SAS panels are shown in the administration console is when there is an older server federated into the domain.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Downstream propagation uses Remote Method Invocation (RMI) over IIOP to access enterprise beans running on a back-end, which means, a downstream server. The security attributes are passed to the enterprise beans running on the downstream server by using the CSIV2 protocol that is established between the WebSphere Application Servers. Basically downstream propagation enables a downstream server to accept the client identity established on an upstream server, without having to reauthenticate.

Types of downstream propagation are possible by using RMI:

► RMI_INBOUND

When you enable security attribute propagation for RMI_INBOUND, then this indicates that the server can receive propagated security attributes from other servers in the same realm over CSIV2 protocol.

► RMI_OUTBOUND

When you enable security attribute propagation for RMI_OUTBOUND, this indicates that the server can send (propagate) security attributes from itself to other servers in the same realm over CSIV2 protocol. For example, consider a scenario described in Figure 11-4 on page 291 where Server 1 makes an RMI call to Server 5. The following actions occur:

- a. Subject contents and the PropagationToken contents are serialized at Server 1.
- b. Server 1 makes an RMI call to Server 5.
- c. Serialized content sent over CSIV2 protocol to the target server (Server 5) that has RMI_INBOUND propagation enabled.
- d. If the receiving server does not support security attribute tokens, WebSphere Application Server sends the LTPA token only.

11.5.1 Downstream propagation scenario

In Figure 11-4 on page 291, Server 1 and Server 2 are in the same DRS domain. Server 5 is a downstream server to which an RMI call is made from Server 1.

1. User authenticates to Server 1. Subject is created during the login process at the front-end server (Server 1) in this case, either by a propagation login or a user registry login.
2. Server 1 and Server 5 have downstream propagation enabled.
3. The user makes an RMI request to an EJB running on application3 on Server 5.

4. WebSphere Application Server propagates the tokens from Subject including custom tokens from Server 1 to the downstream WebSphere Application Server, Server 5. Thus, the security information is available for Server 5 for enterprise bean invocations.
5. If tokens are available, a propagation login is performed otherwise an initial login is performed. The subject is generated at the downstream server (Server 5) and is added to the CSIV2 session.

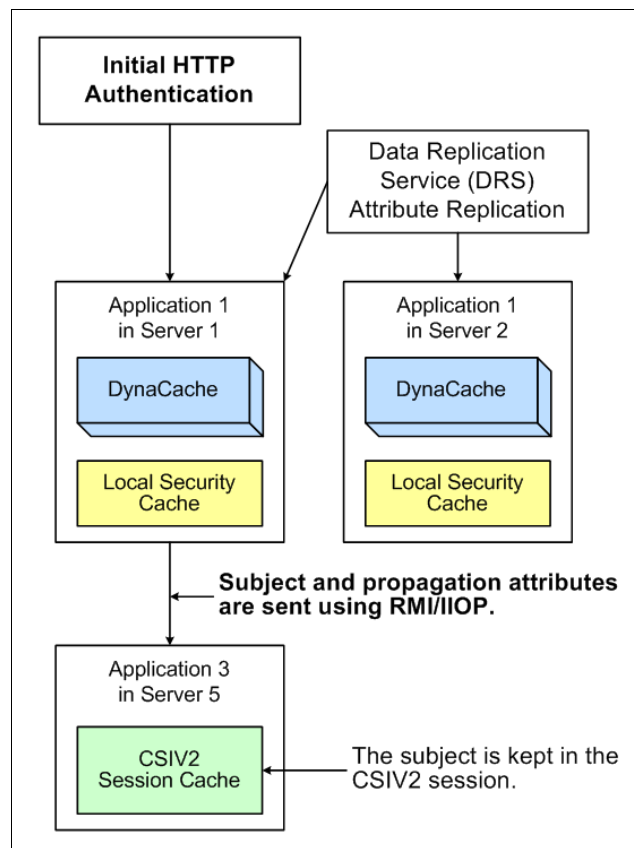


Figure 11-4 Downstream propagation scenario

The downstream server trusts the upstream server, both RMI outbound and inbound propagation must be enabled.

11.6 Enabling security attribute propagation

CSIV2 defines the Security Attribute Service that enables interoperable authentication, delegation, and privileges. You can selectively enable parts of security attribute propagation depending on your server configuration and requirements. For example, you can choose to enable horizontal propagation among front-end WebSphere Application Servers using DynaCache or JMX. You can also choose to enable downstream propagation. Typically, both types are enabled for any given cell.

11.6.1 Configuring security attribute propagation for horizontal propagation

To configure WebSphere Application Server for horizontal propagation:

1. Launch the Administrative Console and log in.
2. Select **Security** → **Secure administration, applications, and infrastructure**. Select **Web security** and then **Single Sign-On**.
3. Optional: Earlier versions of WebSphere Application Server, prior to V5.1.1, did not support security attribute propagation. It used an LTPA token for SSO purposes.

If you must interoperate with such servers, select the **Interoperability Mode** option. A WebSphere Application Server does not support security attribute propagation receive the LTPA token and the propagation token, instead it ignores the security attribute information that it does not understand.

4. Select the **Web inbound security attribute propagation** option, which enables horizontal propagation (Figure 11-5).
See Figure 11-5.

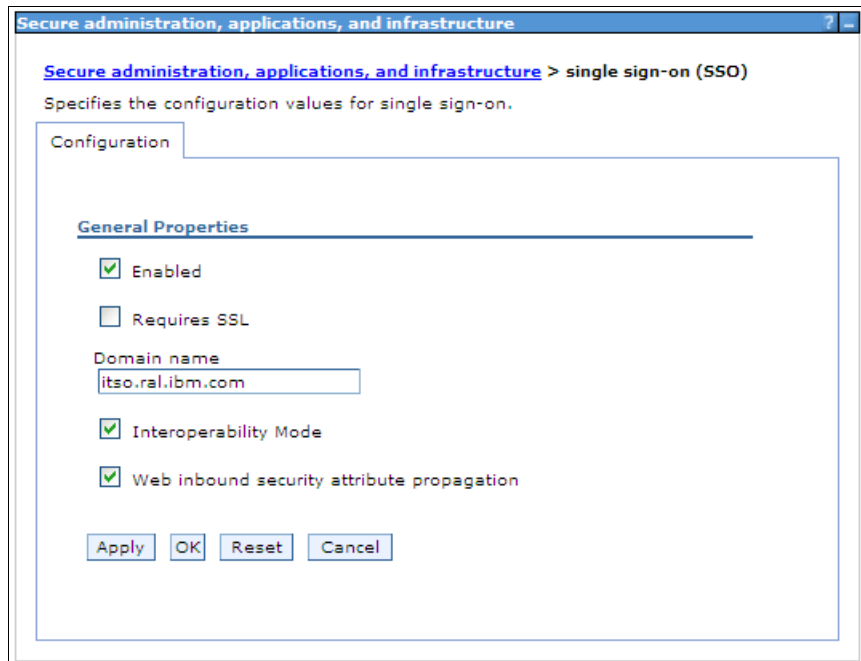


Figure 11-5 Horizontal propagation

With the Web inbound security attribute propagation enabled, the security attributes of the originating server where the initial login occurred, gets propagated to the receiving server. These security attributes include any custom attributes or token that are set in the custom login modules in the login server.

11.6.2 Enabling downstream propagation

For downstream propagation, configure CSIV2 inbound and CSIV2 outbound:

1. Select **Security** → **Secure administration, applications, and infrastructure**.
2. Under RMI/IIOP security, click **CSIV2 inbound authentication**. The login configuration field specifies RMI_INBOUND as the system login configuration used for inbound requests. This cannot be changed. However, you can chain custom login modules to the login configuration.

On this panel, ensure that **Security Attribute Propagation** is checked. Click **Apply**.

3. Select **Security** → **Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSIV2 Outbound authentication**. Note that the login configuration says RMI_OUTBOUND. You cannot change the Login Configuration but you can add additional custom login modules to the configuration.
4. Ensure that **Security Attribute Propagation** is checked in this panel whenever outbound security attribute propagation is selected.

Important: WebSphere Application Server propagates only the objects within the Subject that it can serialize. For the custom objects within the Subject, take care of serialization for it to be propagated properly.

5. Click **Apply**.
6. Save the configuration for WebSphere.

Optional: Select the **Custom Outbound Mapping** option if you deselect the Security Attribute Propagation option and you want to use the RMI_OUTBOUND login configuration. If neither of the options, Custom Outbound Mapping option or Security Attribute Propagation option, is selected, then WebSphere Application Server does not call the RMI_OUTBOUND login configuration. If you require to plug in a credential mapping login module, you must select the Custom Outbound Mapping option.

Note: If you want to propagate security attributes to a different realm then you must specify the target realms in the Trusted target realms field. You must specify each trusted target realm and separate them by a pipe (|) character. For example, specify *server_name.domain:port_number* for a Lightweight Directory Access Protocol (LDAP) server or the machine name for local operating system.

11.7 Advantages of security attribute propagation

The propagation of security attributes in WebSphere Application has significant benefits. It eliminates the requirement to perform registry look-ups at each hop along an invocation.

In your environment, you might use a Web proxy server (for example, WebSEAL) to perform user authentication and gather group information and other security attributes. Previous to 6.0.x, WebSphere Application Server can only use the identity of the user and disregard all the other security attributes. Since then, information that is obtained from the Web proxy server can be used by WebSphere Application Server and propagated downstream to other server resources without additional calls to the user registry.

Another significant benefit of the security attribute propagation is that the user switches that occur because of J2EE Run-As configurations do not cause the Application Server to lose the original caller information. This information is stored in the propagation token that is located on the running thread.

This also enables third-party providers to plug in custom tokens which can then be propagated via custom login modules. The token interface contains a `getBytes()` method that enables the token implementation to define custom serialization, or encryption methods, or both.

Security attribute propagation provides the ability to have a unique ID for each token type. This unique ID is used to formulate a more unique subject identifier than just the user name in cases where dynamic attributes might change the context of a user login. The token interface has a `getUniqueId()` method that is used for returning a unique string for caching purposes.

For example, you might have to propagate the time of the day when the user logs into the system. You can generate this time of the day during the login using either a Web proxy server or by configuring a custom login module in the `WEB_INBOUND` login configuration. This information can then be added to the subject prior to serialization. Other attributes might be added to the subject and use a unique ID. All of the unique IDs must be considered for the uniqueness of the entire Subject. WebSphere Application Server has the ability to specify what is unique about the information in the Subject, which might affect how the user accesses the subject later.



Securing a WebSphere application using Tivoli Access Manager

This chapter discusses IBM Tivoli Access Manager for e-business (Access Manager) use in securing WebSphere Application Server V6.1 applications.

12.1 Introduction to Tivoli Access Manager

IBM Tivoli Access Manager for e-business (Access Manager) is a policy-based access control solution for e-business and enterprise applications. Access Manager is a collected suite of security management services with a variety of distributed blades and plug-ins for the infrastructure components of e-business applications.

12.1.1 Benefits

Tivoli Access Manager allows you to control access across your entire e-business infrastructure, without multiple and possibly conflicting security policies, for any enterprise with multiple Web-based applications.

There is a business-wide change in focus from implementing application-specific security to prevent inappropriate users from accessing resources, towards attempting to develop a common and consistent security policy and base its implementation on common reusable security services and infrastructure.

This is about controlling network identity, correctly identifying a user after authentication and passing that identity together with credentials through to the other components of the e-business infrastructure, applications included. Then the permissions for that identity can be tested locally and access can be given, depending on the security policy for those resources through authorization.

The externalized security provided by Tivoli Access Manager includes strategies to include legacy applications in single sign-on (SSO) solutions through integration with pre-existing user registries and authorization databases.

If a user, regardless of which application a user accesses within an enterprise, always logs in with the same ID and password, although there may be a requirement for stronger authentication or re-authentication, perhaps token or certificate-based around particularly sensitive information or high value transactions, then this consistent user experience is displayed, from the user's viewpoint at least, as SSO. Attempting to ensure users have only a single identity within your network increases the likelihood of leveraging existing infrastructure to actually provide it.

The central definition and management/administration of security policies provides a number of benefits, such as:

- ▶ Reduced security risk through ensured consistency from a services-based security architecture.
- ▶ Lower administration costs due to centralized administration of a reduced number of security systems. This also allows for the “de-skilling” of support

staff because the security policies are based on a single application suite rather than, as in many current examples, the multiple and different operating systems of chained infrastructure platforms.

- ▶ Faster development and deployment with a common services-based architecture.
- ▶ Reduced application development and maintenance costs from increased efficiency and productivity by saving on isolated system and/or application specific security development efforts
- ▶ For those industries where legislative compliance impacts security, for example privacy requirements, centralized architecture provides a more responsive environment and also a single point to apply policy.
- ▶ Tivoli Access Manager's auditing can also help prove compliance to Sarbanes-Oxley (SOX) Act, Health Insurance Portability and Accountability Act (HIPAA), or the Basel II international banking accord.

All of these benefits contribute to enabling an enterprise to be faster to market with new applications and features.

The down side of having a single security solution based on a single technology or product is that any product-specific security exploitation results in enterprise-wide vulnerability. It does, however, let you concentrate your defenses rather than be forced to dissipate your efforts across multiple platforms and products.

12.1.2 When to use Tivoli Access Manager for e-Business in conjunction with WebSphere Application Server

Both Tivoli Access Manager and WebSphere Application Server are good products that are leaders in their areas. They also compliment each other well when appropriate. The decision whether to use Tivoli Access Manager is dependant on whether the features that it provides are required. WebSphere Application Server is a high quality secure product in its own right but Tivoli Access Manager can increase the number of security features available to the user. When deciding whether to use Tivoli Access Manager the user must identify specific requirements and then determine what additional value Tivoli Access Manager provides.

Some extra features that Tivoli Access Manager provides are:

- ▶ Web SSO across multiple products such as WebSphere Application Server and WebLogic. Note that if the environment contains only WebSphere Application Server and Portal then this feature is already provided by

WebSphere. This is because multiple WebSphere Application Server servers and cells (along with Portal) can form a single SSO domain

- ▶ Cross Domain Name Server (DNS) domain SSO.
- ▶ Advanced authentication, which includes SecurID, password strength testing, password expiration, login rules, Resource Access Control Facility (RACF) authentication, multi factor authentication, step up authentication, and so on.
- ▶ Defense-in-depth
 - WebSEAL can be placed in a demilitarized zone (DMZ) in front of a Web server.
 - WebSEAL can ensure that only authenticated traffic enters enterprise intranet.
 - This allows multiple authorization enforcement points throughout the environment that use the same policies.

Note: Using WebSEAL for defense in depth raises the following issues:

- ▶ More infrastructure is required in the DMZ.
- ▶ Proxy server configuration raises URL issues that must be addressed early in development.
- ▶ If any application does not require authentication then the DMZ authentication requirement becomes invalid.

Attention: Except for Tivoli Access Manager for WebSphere Application Server Embedded, Tivoli Access Manager 6.0 is under different licensing arrangements to WebSphere Application Server. While the Tivoli Access Manager Base components are shipped as part of WebSphere Application Server ND, products that are part of the Web Security suite, such as WebSEAL, is required to be purchased under a Tivoli license agreement.

12.1.3 Reverse proxies for authentication

One of the more well known products in the Tivoli Access Manager range is WebSEAL. WebSEAL is a reverse proxy that has the ability to authenticate and perform coarse grained authorization. Using reverse proxies for authentication in the DMZ as a point-of-contact is a popular and common Web security architecture. Even though Figure 12-1 is Tivoli specific, it is a good example of the standard architecture for a reverse proxy deployment. Reverse proxies are designed to work with firewalls to greatly reduce the exposure of Web servers and Application Servers to external attacks.

All the user's contact with the Web site is through reverse proxy. When the user first makes a request to the Web site the reverse proxy has the ability to authenticate the user and also carry out coarse grained authorization. The reverse proxy server then retrieves the Web resources through the firewall from the actual Web server. Some reverse proxies, including WebSEAL, allow caching of Web resources at the proxy level, helping speed up retrieval of static pages and images.

There are some situations where a reverse proxy is useful, such as acting as a point-of-contact in a federated SSO environment. But as with every technology, reverse proxies are appropriate in many situations but are not the answer in every case.

12.1.4 Access Manager Secure Domain

The Access Manager Secure Domain provides a secure computing environment in which Access Manager enforces security policies for authentication, authorization, and access control. It ignores performance, redundancy, and availability considerations which must be addressed in production systems. Figure 12-1 on page 302 shows the essential components.

IBM Tivoli Access Manager V6.0 requires a user registry and can be configured to use different products, including Microsoft Active Directory® and iPlanet, but the product itself ships with IBM Tivoli Directory Server V6.0, underpinned by the IBM DB2 Universal Database™.

The Access Manager Policy Server maintains the master authorization policy database, which contains the security policy information for all resources and all credential information of all participants within the secure domain, both users and servers. A secure domain contains of physical resources requiring protection. These resources include programs, files, and directories. A virtual representation of these resources, protected by attaching access control list (ACL) and protected object policy (POP) entries, is stored by the Policy Server.

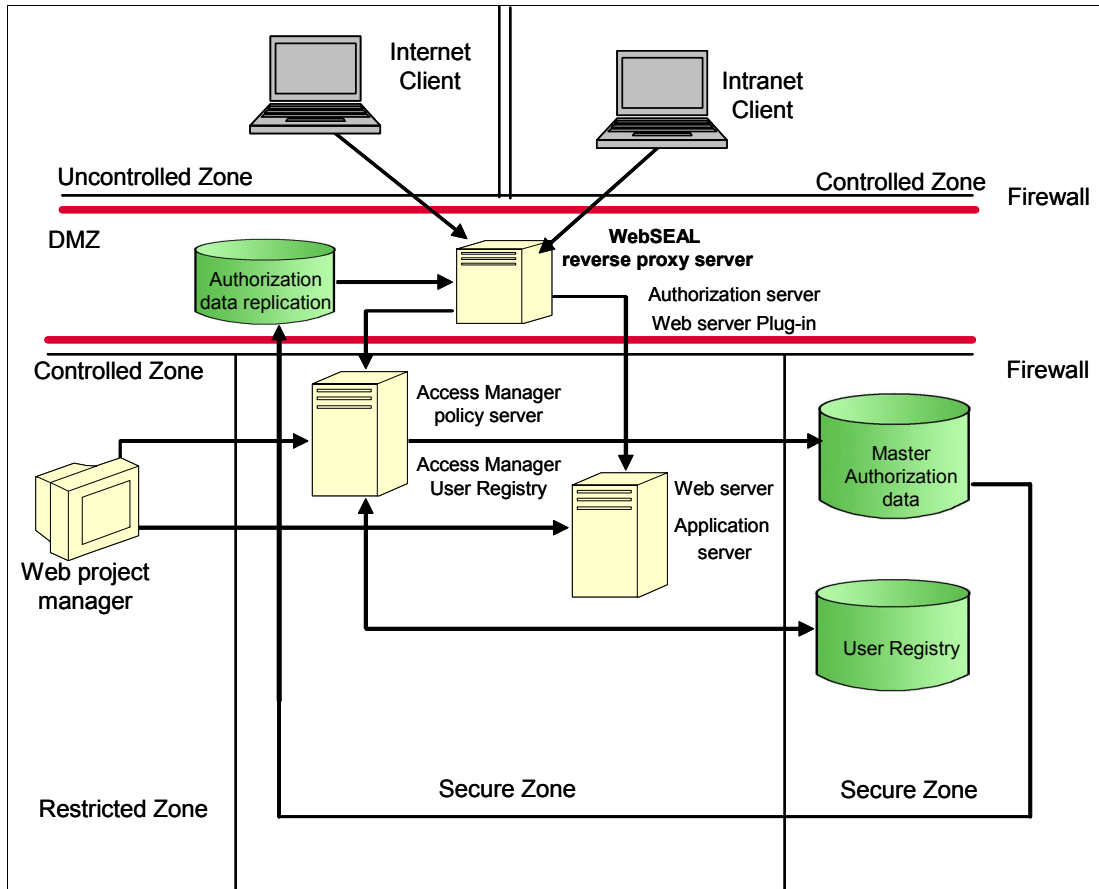


Figure 12-1 Typical three-tier infrastructure supporting e-business applications

The policy server replicates this database to all the local authorization servers, including WebSEAL, throughout the domain, publishing updates as required. The policy server also maintains location information about the other Access Manager and non-Access Manager servers operating in the secure domain. There can be only one policy server active within a domain.

Access Manager provides C and Java authentication and authorization application programming interfaces (APIs), which can be used programmatically within other applications and clients. Client calls for authorization decisions, through the Access Manager runtime service which must be on every server participating in the secure domain, are always referred to an authorization server. Programmatically made calls can be local or remote. They are passed to an authorization server.

Authorization servers are the decision-making servers that determine a client's ability to access a protected resource based on the security policy. Each server has a local replica of the policy database. There must be at least one within a Secure Domain.

Web Portal Manager, a WebSphere-hosted application is provided to enter and modify the contents of the policy store and the user registry. There is also a command line utility, `pdadmin`, which extends the commands available to include the creation and registration of authentication blades such as WebSEAL which is described in a subsequent section.

A new feature added to Tivoli Access Manager for e-Business 6.0 is the Session Management Server (SMS). This optional component which runs on WebSphere is able to manage and monitor sessions across dispersed or clustered Access Manager-protected Web servers or Access Manager proxies. Using the session management server allows the Access Manager WebSEAL and Access Manager plug-in for Web servers components to share a unified view of all current sessions. Session management server permits any authorized user to monitor and administer user sessions. It records a variety of session information, including session inactivity and lifetime timeout information, login activity, and concurrent login information. Because it runs on WebSphere, this gives the environment a logical single point of administration while still enjoying replication and high-availability.

You can configure Access Manager to integrate with many of the WebSphere branded products and ships with explicit plug-ins for the following products:

- ▶ WebSphere Application Server
- ▶ WebSphere Edge Server
- ▶ Web server plug-in which supports:
 - Apache Web server on AIX, Linux on zSeries®, and Solaris™
 - IBM Hypertext Transfer Protocol (HTTP) Server on AIX, Linux on x86, Linux on zSeries, and Solaris
 - Internet Information Services on Windows 2003
 - Sun Java System Web server on AIX and Solaris

For details of the supported operating systems for every component consult the Tivoli Information Center at the following address:

<http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame.doc/welcome.htm>

Table 12-1 on page 304 shows the components that are installed for the sample configurations in this book. The components are installed in a mixture of Linux and Windows servers.

For installation instructions, see the original product documentation that comes with the package or read the documentation in the Information Center.

Note: We do not use the Session Management Server (SMS) or Common Audit and Reporting Service (CARS) for this scenario.

Table 12-1 IBM Tivoli Access Manager V6.0 components used

Server	Required component
Tivoli Directory Server V6.0	Directory server
	Directory client
	DB2 Universal Database edition
	Global Security Toolkit (gskit)
Tivoli Access Manager Policy Server V6.0	Access Manager run time
	Access Manager policy server
	Tivoli security utilities
	Global Security Toolkit (gskit)
	Access Manager license
	Tivoli directory client
Tivoli Access Manager Authorization Server V6.0	Access Manager authorization server
	Access Manager run time
	Tivoli security utilities
	Global Security Toolkit (gskit)
	Access Manager license
	Tivoli directory client

Server	Required component
Tivoli Access Manager Web Portal Manager V6.0	Web Portal Manager (WebSphere enterprise application)
	WebSphere Application Server
	Tivoli security utilities
	Directory client
	Global Security Toolkit (gskit)
	Access Manager license
	Access Manager run time
Tivoli Access Manager WebSEAL Server V6.0	Access Manager WebSEAL server
	Access Manager Web run time
	Access Manager run time
	Tivoli security utilities
	Global Security Toolkit (gskit)
	Access Manager license
	Tivoli directory client

12.1.5 Tivoli Access Manager auditing

Auditing is the process of maintaining detailed, secure logs of critical activities in a business environment. These activities can be related to security, content management, business transactions, or other such activities. The following events are common events that are audited:

- ▶ Login failures
- ▶ Unauthorized access to protected resources
- ▶ Modification to security policy

Tivoli Access Manager provides two methods for managing audit events. One method uses the native Tivoli Access Manager approach, and the other method uses the CARS.

Enterprises can use information contained in audit trails to help them show compliance with government regulations such as the SOX Act, the HIPAA, and the Basel II international banking accord. Audit trails are also useful to check enforcement and effectiveness of IT controls, for accountability, and vulnerability, and risk analysis.

For more information about the auditing, see *Auditing Guide, IBM Tivoli Access Manager for e-Business V6.0, SC32-2202*.

Common Auditing and Reporting Service

The Common Auditing and Reporting Service is a new feature provided with Tivoli Access Manager 6.0. It provides a mechanism by which the enterprise can centrally audit and report on their environment. The service's clients, for example WebSEAL, use Web service requests in the form of Common Base Events (CBE) to send auditing information to the IBM Common Event Infrastructure (CEI) server.

The audit events are stored as Extensible Markup Language (XML) in a data store. This is good from a reporting standpoint as it means that reporting software, such as Crystal Reports, is able to create reports with greater ease and more flexibility. CARS comes with some out-of-box reports including:

- ▶ Audit event history
- ▶ Authentication event history
- ▶ Authorization event history
- ▶ Event details
- ▶ Password change activity
- ▶ Resource access
- ▶ Server availability reports

The Common Auditing and Reporting Service comes with the Common Auditing Service server, which includes the event server and the operational reports feature and also the Common Auditing Service C and Java clients.

12.1.6 Access Manager and WebSphere integration

To provide a standard-based authorization framework for WebSphere applications, Tivoli Access Manager supports the Java 2 security model and also the Java Authentication and Authorization Service (JAAS) and Java 2 Enterprise Extensions (J2EE).

Integrating WebSphere and Access Manager adds WebSphere resources to the significant list of elements that you can manage via Tivoli Access Manager's consistent authorization policy, and it also adds to WebSphere applications the benefits that accrue in an Access Manager protected environment. Some examples of this include URI-based access control, availability, and scalability characteristics inherent in Access Manager implementations, and the ability to support many authentication mechanisms without any impact to the target application and Web SSO, which are fully applicable for WebSphere Application Server.

what component initiates the connection. Figure 12-2 shows boxes, at all the different points where the Web client can authenticate, listing the HTTP and Hypertext Transfer Protocol Secure (HTTPS) authentication methods available. Not all the communications in Figure 12-2 must happen at the same time, but they are shown to give an idea of all the possibilities.

Some components can be taken out the picture and the architecture is still valid in some scenarios. For example, without WebSEAL, WebSphere can still use the policy server for its authentication decisions. The Web server is also an optional component, because WebSEAL can junction directly to the WebSphere Application Server. However, generally it is considered good practice to use the Web server to serve static content. This chapter concentrates on the Tivoli Access Manager components. See previous chapters in this book for security regarding the IBM HTTP Server, WebSphere Application Server, and the Tivoli Directory Server.

Communications to the Access Manager policy server are always encrypted for security reasons but this is transparent to the other components as it is handled internally by the Access Manager run times. WebSEAL provides different authentication mechanisms from the client and integrates them by using different authentication mechanisms with the back-end servers to provide a true SSO even to Application Servers not aware of it.

The integration of WebSphere Application Server and Access Manager offers the following additional options or possibilities:

- ▶ Shared user registry
- ▶ Web SSO using:
 - Tivoli global sign-on (GSO) junctions
 - Web Trust Association Interceptor (TAI)
 - WebSEAL Lightweight Third Party Authentication (LTPA) cookie support
- ▶ Application integration using:
 - Authorization application programming interface (aznAPI)
 - JAAS
 - JACC
 - PDPermission
 - J2EE security

12.1.7 Reverse proxy authenticators and the extended WebSphere trust domain

In order for reverse proxy authenticators to carry out their tasks successfully they must terminate any Secure Sockets Layer (SSL) connections, authenticate the user, and then access the Web resource. One by-product of this situation is that WebSphere Application Server must completely trust the proxy. For example, the Application Server does not get to see the clients certificate directly, thus any client certificate authentication must be carried out by the proxy, therefore the proxy must be trusted by the Application Server. Also, when the proxy server vouches for an authenticated identity the Application Server must trust proxy and be absolutely certain of its correctness. Another by-product is when the proxy tries to access a Web resource on behalf of a user, the application server or Web server must be absolutely certain that it is the proxy that is making the request.

To fulfill these trust requirements the proxy must become part of the Application Server's trust domain. There are a number of ways to do this but the most common are:

- ▶ The proxy server authenticates itself to the Application Server either using a user name and password or token.
- ▶ The proxy server uses a mutually authenticated SSL during communication with the Application Server or Web server.

By using these methods the Application Server can trust who the proxy is, that it is the proxy making the connection, and that the information received from the proxy is secure.

This trust association is carried out in WebSphere Application Server using Trust Association Interceptors, otherwise known as TAIs. For further information, see 13.1, "Trust Association Interceptor" on page 354.

12.1.8 Challenges with reverse proxy authenticators

This section covers the challenges you may face with reverse proxy authenticators.

Session management

While the proxy is being used to authenticate for WebSphere Application Server and perhaps other systems as well, there are multiple applications keeping user state information. This session information must be kept consistent or else there is a potential for security vulnerabilities. The problems occur because not all user state implementations follow the same rules. For example, WebSEAL's authentication lifetime is based on absolute time and also idleness. This is

different to WebSphere Application Server's authentication lifetime which is based on absolute time. And different yet again is WebSphere Application Server's HTTP session lifetime which is based on idleness. These inconsistencies require careful planning of security attribute lifetimes.

The following situations indicate where the state information can become inconsistent:

- ▶ Log out of proxy but not WebSphere
- ▶ Proxy Crash
- ▶ Authentication to proxy times out but WebSphere cookies are still valid
- ▶ Log out of WebSphere but not proxy
- ▶ HTTP Session in WebSphere times out but WebSphere cookies still valid

The first three situations result in a situation where the user still has a valid back-end state while the front-end state has been invalidated. This is potentially bad.

Logout considerations

In an SSO environment, logging out can have different scopes. These logout considerations must be handled carefully. The best way to approach this problem is from the users' perspective. What might a user expect when they click logout? This is partly determined by how integrated the SSO domain is. Two major cases lead to three logout semantics:

- ▶ Loosely related SSO

For example, a user logging in to an operating system and then starting an application that requires authentication. The operating system automatically logs the user in, resulting in a loosely related SSO environment. When the user logs out of the application they do not expect to be logged out of the entire operating system.

- ▶ Seamlessly related

For example, a portlet where after the user authenticates, they have access to all resources offered without having to re-authenticate. When the user logs out they expect to be logged out of all the resources simultaneously.

The logout semantics or scopes derived from the previous cases are as follows:

- ▶ Logout is only the application
- ▶ Logout is SSO wide
- ▶ The type of logout is dynamically determined by the user, where the user is asked what they wish to do

Implementation of these logouts is quite complicated.

Step-up authentication

Step-up authentication is the act of forcing the user to authenticate again in a stronger manner to reach a protected resource. For example, when a bank manager changes from a user name/password protected intranet site to a highly sensitive site, this manager can be forced to enter a SecurID one time password. Traditionally reverse proxies enforce step-up authentication based on the URL of the request. Some products today, such as WebSphere Portal Server, do not use URLs. This can mean that step-up authentication may not have been enforced. For resources that require step-up authentication, the programmer must ensure that step-up authentication has occurred every time, redirecting to the proxy to enforce it.

Standard configuration practices

The following set of standard practices helps to alleviate these problems:

- ▶ Ensure that the only path from the user to WebSphere is through the proxy.
- ▶ Configure the LTPA lifetime to be shorter than that of the proxy session lifetime.
- ▶ SSO is carried out (from the Application Server's point of view) *outside in*.
- ▶ Single sign-off is carried out (from the Application Server's point of view) *inside out*, or for use cases not addressed by a global logout, leverage JavaScript™ to clean the clients browser.
- ▶ Explicitly enforce step-up authentication if you are using a product that does not use URLs. For example, Portal Server.

Ensuring that the only path from the clients browser to the Application Server is through the reverse proxy reduces the risks associated with a user having a valid back-end system session but invalid proxy session. Some methods to achieve this include setting up SSL keys so that they only trust a small set of clients, only accepting connections from certain hosts, and of course, the use of firewalls.

Configuring the LTPA lifetime to be shorter than the proxy session lifetime must help prevent the user having a valid back-end session and invalid front-end session. If the LTPA cookie times out during use, the application server issues a new one transparently from the user's point of view (as long as the request is part of a valid proxy session). If the proxy server's timeout is dependent on idleness then make the LTPA lifetime less than that. Otherwise make the LTPA lifetime a small fraction of proxy absolute session lifetime.

Changing the token time-outs does not help if another user gets to the browser before the LTPA token times out. Ensuring that the proxy and application server session state are the same is the only way to ensure that stray back-end tokens are not misused.

For SSO, the best practice is to first create a session at the proxy server and then have the proxy server pass the request to the back end where another session is created. This way the back-end is only creating sessions that have a corresponding session in the proxy.

For single sign-off, the best practice is to have the environment perform it from the inside out. For a global logout, then, it can be beneficial to make each application to have a logout URL that destroys its session for a user. A manager must be running in the environment that ideally knows which applications that the user has sessions with. When the logoff request reaches the manager, the manager sends notifications to all the applications that have sessions for the user, telling them to destroy it. After that the manager then destroys its own session, finally calling the proxy and telling it to destroy its session. You can use this approach for quite large deployments, especially federated single-sign on environments using Tivoli Federated Identity Manager (TFIM).

Example 12-1 is a simple inside out logout for a WebSphere Portal Server environment. By changing the Portal's ConfigService.properties file so that the post-logout URL points to the proxies logout URL (in WebSEAL's case, pkmslogout), when the session is terminated at the Portal it cleans itself up and then calls the proxy, telling it to clean itself up. This results in a clean environment where the proxy, Application Server, and portal server have destroyed the session and removed the user's tokens. If there were more than one application, then either the proxy must notify the other applications, or the manager approach earlier must be employed.

A single application logout can also be tricky. One of the most common and well documented solutions is to have JavaScript on the proxy's login and logout pages that destroys any relevant tokens. This approach can also be useful in both the global and single application logout cases, where it stops tokens from a previous session contaminating the users session (Example 12-1).

Example 12-1 An example of a logout page that clears any cookies that are in the user's browser

```
<!-- DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN" -->

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>PKMS Administration: User Log Out</title>
```

```

<Script language="JavaScript">

// The cookie_list array stores the names of all the session cookies that
// will be clear upon the load of this page.
//
// Each entries in this array should be formatted as
//   cookieName[__cookiePath[__domain]].
//
// For example:
//   cookie_list = new Array("JSESSIONID__/", "ASPSESSION__/html_ibm.com");

cookie_list = new Array();

// Clears all the session cookies by iterating through the array cookie_list
function session_clean() {
    // initialise variables
    var cookieData;
    var endName;
    var endPath;
    var cookiePath;
    var cookieName;
    var cookieDomain;

    // iterate through cookie_list to clear each cookie
    for (var i in cookie_list) {
        cookiePath="";
        cookieDomain="";
        cookieData = cookie_list[i];
        endName = cookie_list[i].indexOf("__");

        // Check to see if a path was entered
        // if no path was entered then assign the cookie array element to 'cookieName'
        if (endName==--1) {
            cookieName = cookieData;
        }
        // if a path was entered, extract the cookieName and cookiePath details from
the provided array element
        else {
            cookieName = cookieData.substring(0,endName);

            // check to see if domain was entered
            cookieData = cookieData.substring(endName+2, cookieData.length);
            endPath = cookieData.indexOf("__");

```

```

        // if no domain was entered, extract the cookiePath details from the
provided array element
        if (endPath == -1) {
            cookiePath = cookieData;
        }
        // if a domain was entered, extract the cookiePath and cookieDomain details
from the provided array element
        else {
            cookiePath = cookieData.substring(0,endPath);
            cookieDomain = cookieData.substring(endPath+2,cookieData.length);
        }
    }
    // if from the root server directory then create cookie with current directory
as domain (default)

    if ((cookiePath=="")&&(cookieDomain=="")) {
        document.cookie = cookie_list[i] + "=" + "" +
            ";expires=Monday, 01-Jan-80 00:00:00 GMT";
    }
    else if ((cookiePath != "")&&(cookieDomain=="")) {
        document.cookie = cookieName + "=" + "" + ";path=" + cookiePath +
            ";expires=Monday, 01-Jan-80 00:00:00 GMT";
    }
    else if ((cookiePath != "")&&(cookieDomain!="")) {
        document.cookie = cookieName + "=" + "" + ";domain=" + cookieDomain +
";path=" + cookiePath +
            ";expires=Monday, 01-Jan-80 00:00:00 GMT"
    }
}
}
}

</Script>
</head>
<!--Call session_clean() to clear cookies when the html page loads-->
<body bgcolor="#FFFFFF" text="#000000" onLoad=session_clean()>
<font size="+2"><b>User %USERNAME% has logged out.</b></font>
</body>
</html>

```

12.2 IBM Tivoli Access Manager security model

The security policy for a Tivoli Access Manager secure domain is maintained and governed by two key security structures:

- ▶ User registry
- ▶ Policy database

12.2.1 User registry

The user registry (such as Lightweight Directory Access Protocol (LDAP), Lotus Domino, or Microsoft Active Directory) contains all users and groups who are allowed to participate in the Tivoli Access Manager secure domain. In the example used in this book, the IBM Tivoli Directory Server LDAP directory contains the user registry shared by Tivoli Access Manager and WebSphere Application Server.

The ability of Tivoli Access Manager to coexist with federated repositories has the following limitations:

- ▶ You can configure only one LDAP repository under Federated repositories, that LDAP repository configuration must match the LDAP server configuration under Tivoli Access Manager.
- ▶ The Distinguished Name (DN) for the realm base entry must match the LDAP DN of the base entry within the repository. In WebSphere Application Server, Tivoli Access Manager recognizes the LDAP user ID and LDAP DN for both authentication and authorization. The federated repositories configuration does not include additional mappings for the LDAP user ID and DN.
- ▶ The federated repositories functionality does not recognize the metadata that is specified by Tivoli Access Manager. When users and groups are created under user and group management, they are not formatted using the Tivoli Access Manager metadata. The users and groups must be manually imported into Tivoli Access Manager before you use them for authentication and authorization.

Attention: A new minimal data format was introduced with Tivoli Access Manager for e-Business 6.0. If you want to interact with previous versions of Access Manager clients, do *not* select this option during configuration of the policy server.

12.2.2 Master authorization (policy) database

The authorization database contains a representation of all resources in the domain (the protected object space). The security administrator can dictate any level of security by applying rules, known as ACL policies, POP, and authorization rules, to those resources requiring protection

The Tivoli Access Manager authorization service enforces security policies by comparing a user's authentication credentials with the policy permissions assigned to the requested resource. The resulting recommendation is passed to the resource manager, for example, WebSEAL or WebSphere Application Server, which completes the response to the original request. The user credential is essential for full participation in the secure domain.

The protected object space

The protected object space is a hierarchical portrayal of resources belonging to an Access Manager secure domain. The virtual objects that are displayed in the hierarchical object space represent the actual network resources in the domain. They can be *system resources*, which are the actual file or application, and *protected objects*, which are the logical representation of an actual system resource used by the authorization service and other Access Manager management components.

You can attach policy templates to objects in the object space to provide protection of the resources. The authorization service makes authorization decisions based on these templates.

These rules can be explicitly attached or inherited. The Access Manager protected object space shown in Figure 12-3 on page 317 supports inheritance of the security policies or rules. This is an important consideration for the security administrator who manages the object space. The administrator has to apply explicit policies only at points in the hierarchy where the rules must change.

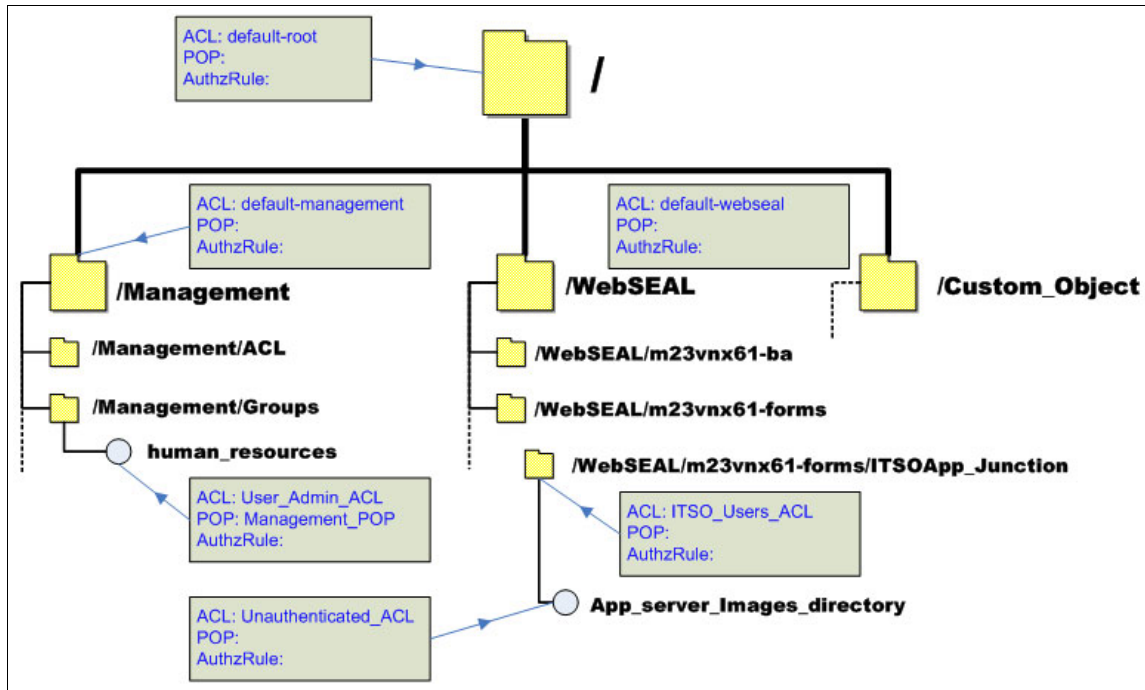


Figure 12-3 Tivoli Access Manager object space

The following object space categories are used by Access Manager:

- ▶ Web objects

Web objects represent any resource that a URL can address, including static and dynamic contents. The WebSEAL server is a component of Access Manager, responsible for protecting Web resources.

- ▶ Access Manager management objects

Management objects represent the management activities that you can perform through the Web Portal Manager. The objects represent the tasks necessary to define users and set security policy. Access Manager supports delegation of management activities and can restrict an administrator's ability to set security policy to a subset of the object space. An example of an Access Manager management object is a defined group, for example, **/Management/Groups/boardmembers**. ACLs can be attached to the object to restrict who can add members to that group.

- ▶ User-defined objects

User-defined objects represent customized tasks or network resources protected by applications that access the authorization service through the Access Manager authorization API. For instance, in library application you

can map actions to objects and allow *everyone* access to the object /library/book/summary but only allow *authenticated* access users to the object /library/book/reservation.

Access Manager authorization engine

The Access Manager authorization service performs authorization decisions, as shown in Figure 12-4, based on the policies applied to the objects as explained earlier. For each request, for access to an object inside the protected object space, the request is evaluated against the ACL, the POP, and the authorization rule attached to the object or inherited by it, in the order described. A single object can have none, one, two, or all three types of rule attached to it but only one of each type.

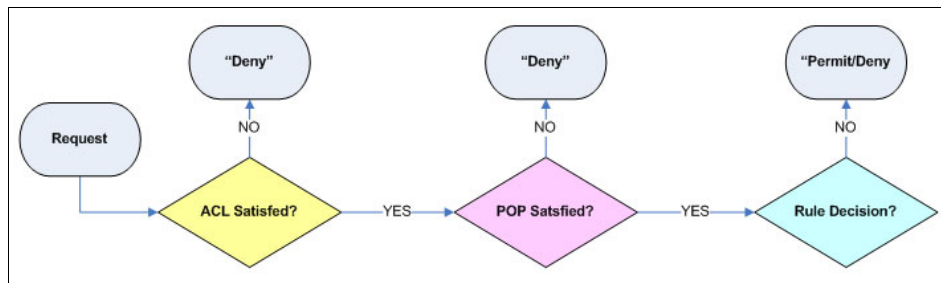


Figure 12-4 Access Manager authorization flow

Access control list

An access control list policy, or ACL policy, is the set of rules (permissions) that specifies the conditions necessary to perform certain operations on a resource. ACL policy definitions are important components of the security policy established for the secure domain. ACL policies, similar to all other policies, are used to stamp an organization's security requirements onto the resources represented in the protected object space. An ACL policy specifically controls:

- ▶ What operations can be performed on the resource.
- ▶ Who can perform these operations.

An ACL policy is made up of one or more entries that include user and group designations and their specific permissions or rights. An ACL can also contain rules that apply to unauthenticated users.

Protected object policy

ACL policies provide the authorization service with information to make a *yes* or *no* answer on a request to access a protected object and perform some operations on that object. POPs contain additional conditions on the request that are passed back to the Access Manager Base and the resource manager (such

as WebSEAL) along with the *yes* ACL policy decision from the authorization service. An example of a POP is *time-of-day* access privileges. It is the responsibility of Access Manager and the resource manager to enforce the POP conditions.

Authorization rules

An Access Manager authorization rule is a policy type similar to an access control list or a protected object policy. Authorization rules provide the flexibility required to extend an ACL or POP by tailoring security policy to your requirements. The rule is stored as a text rule within a rule policy object and is attached to a protected object in the same way and with similar constraints as ACLs and POPs.

Rules allow you to make decisions based on the attributes of a person or object and the context and environment surrounding the access decision. For example, you can use a rule to implement a time-of-day policy that depends on the user or group. You also can use a rule to extend the access control capabilities, which ACLs provide, by implementing a more advanced policy, such as one based on quotas. While an ACL can grant a group permission to write to a resource, a rule can go a step further by allowing you to determine if a group has exceeded a specific quota for a given week before permitting that group to write to a resource.

More information: For more detailed information about Tivoli Access Manager security and administration, see the following documents:

- ▶ *Administration Guide, IBM Tivoli Access Manager for e-Business V6.0*, SC32-1686
- ▶ *Installation Guide, IBM Tivoli Access Manager for e-Business V6.0*, SC32-1684
- ▶ *WebSEAL Administration Guide, IBM Tivoli Access Manager for e-Business V6.0*, SC32-1687
- ▶ *Auditing Guide, IBM Tivoli Access Manager for e-Business V6.0*, SC32-2202
- ▶ *Enterprise Security Architecture Using IBM Tivoli Security Solutions*, SG24-6014
- ▶ *IBM Tivoli Access Manager for e-business*, REDP-3677

12.3 Summary of Access Manager deployment for integration with WebSphere Application Server

Note that the deployment assumes that you have already planned the environment. It also assumes that during deployment and configuration, application hardening guidelines are followed. The basic steps involved in Access Manager deployment are as follows:

1. Deploy and configure the registry. In Figure 12-5 we deployed DB2 and LDAP.
2. Deploy WebSphere Application Server and configure it to use the registry from the previous step.
3. Deploy, configure and secure IBM HTTP Server (IHS) and plug-in for WebSphere Application Server.
4. Deploy Access Manager policy manager.
5. Deploy Access Manager authorization server (if required).
6. Deploy WebSEAL.
7. Create or obtain certificates for WebSEAL, IHS, and WebSphere. Make sure that they are trusted for each communication link, for example, Client <-> WebSEAL, WebSEAL <-> IHS, IHS <-> WebSphere Application Server.

Internal communication security: Security of all internal communication between Tivoli Access Manager components are automatically handled by Tivoli Access Manager certificates.

8. Configure WebSphere Application Server to allow secure connections from WebSEAL. To do this use TAI (recommend), LTPA junctions, or connect to legacy applications using basic authentication (BA) or forms junctions.
9. Configure the front-end authentication for WebSEAL.
10. Create junctions in WebSEAL that are able to connect to WebSphere Application Server.

12.4 Lab environment

To test some different Access Manager - WebSphere integration scenarios described in this chapter, we use a lab environment with all the elements as shown in Figure 12-2 on page 307.

Use Figure 12-5 to understand the lab environment used for these examples.

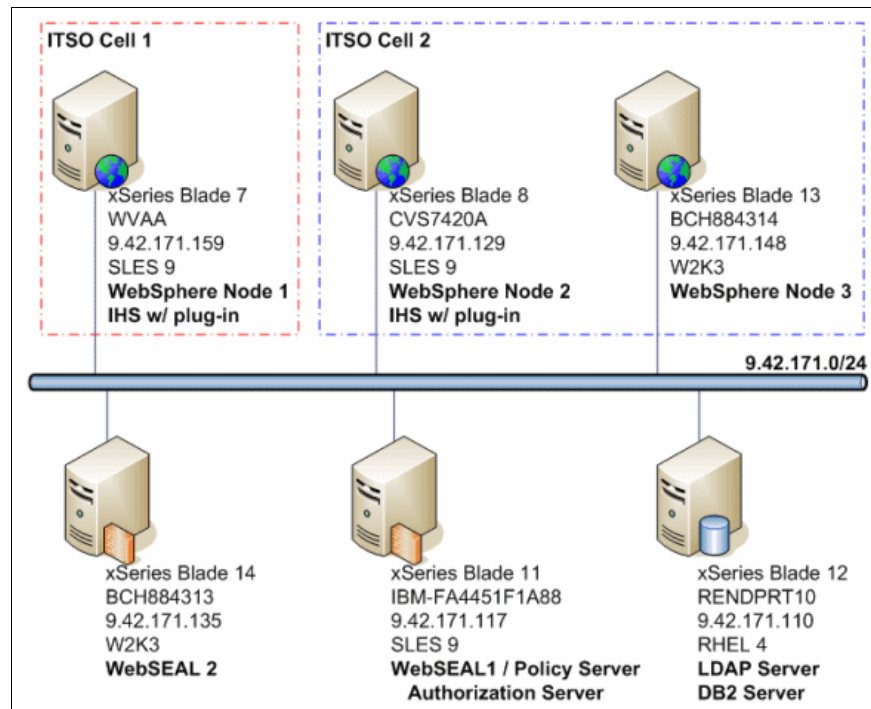


Figure 12-5 Lab environment

All the WebSphere Application Servers have the ITSOBank and ITSOHello installed. For details, see Appendix A, “Additional configurations” on page 509.

They also have J2EE, administrative and application security enabled with LDAP as the user registry, for details see Chapter 2, “Configuring the user registry” on page 7. The LDAP server is also the registry configured with Tivoli Access Manager.

12.5 The role of Tivoli Access Manager inside WebSphere Application Server V6.1

The JACC provider in WebSphere Application Server is implemented by both the client and the server pieces of the Tivoli Access Manager server. The client piece of Tivoli Access Manager is embedded in WebSphere Application Server. The server piece is located on a separate installable CD that is shipped as part of the WebSphere Network Deployment (ND) package. The JACC provider is not an

out-of-box solution. Figure 12-6 represents the high level components of the WebSphere embedded Tivoli Access Manager (Tivoli Access Manager Client) and Tivoli Access Manager Server.

In Figure 12-6, conceptually three layers of integration points are allowed in the embedded Tivoli Access Manager. It is provided in a hierarchical fashion so that the function in each layer can become an integration point to fit into different application-specific requirements. The bottom layer, which is the lowest building block for the other two layers, includes a set of client-server components, which are the Access Manager Java Runtime (AMJRTE) component and the Tivoli Access Manager Server component. While the AMJRTE component serves the client-side integration point, the Tivoli Access Manager Server component provides the infrastructure for both runtime and management operations.

For more information about JACC, see Chapter 14, “Externalizing authorization with JACC” on page 403.

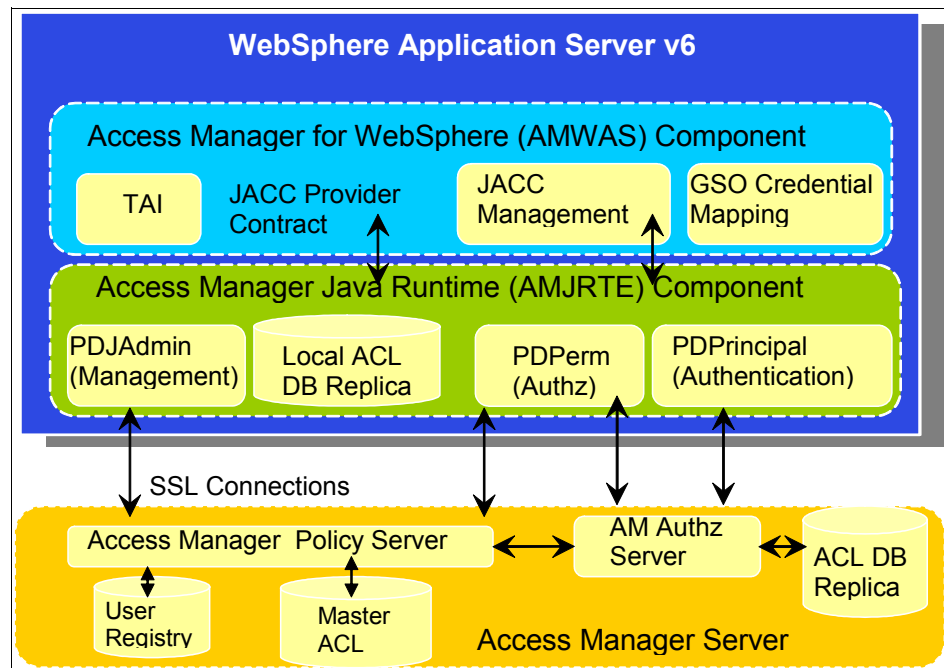


Figure 12-6 WebSphere and Access Manager relations

12.5.1 Embedded Tivoli Access Manager client architecture

Figure 12-7 shows the Tivoli Access Manager client architecture in WebSphere Application Server V6.1.

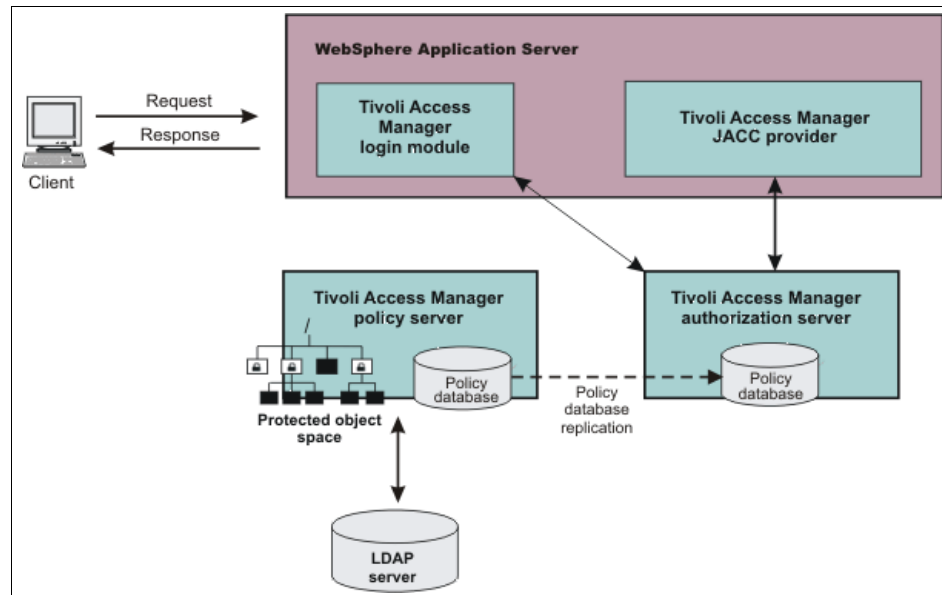


Figure 12-7 Embedded Tivoli Access Manager client architecture

The process is performed as follows:

1. Users that access protected resources are authenticated using the Tivoli Access Manager login module that is configured for use when the embedded Tivoli Access Manager client is enabled.
2. The WebSphere Application Server container uses information from the J2EE application deployment descriptor to determine the required role membership.
3. WebSphere Application Server uses the embedded Tivoli Access Manager client to request an authorization decision (granted or denied) from the Tivoli Access Manager authorization server. Additional context information, when present, is also passed to the authorization server. This context information is comprised of the cell name, J2EE application name, and J2EE module name. If the Tivoli Access Manager policy database has policies that are specified for any of the context information, the authorization server uses this information to make the authorization decision.
4. The authorization server consults the permissions that are defined for the specified user within the Tivoli Access Manager - protected object space. The protected object space is part of the policy database.

5. The Tivoli Access Manager authorization server returns the access decision to the embedded Tivoli Access Manager client.
6. WebSphere Application Server either grants or denies access to the protected method or resource, based on the decision returned from the Tivoli Access Manager authorization server.

See Figure 12-8.

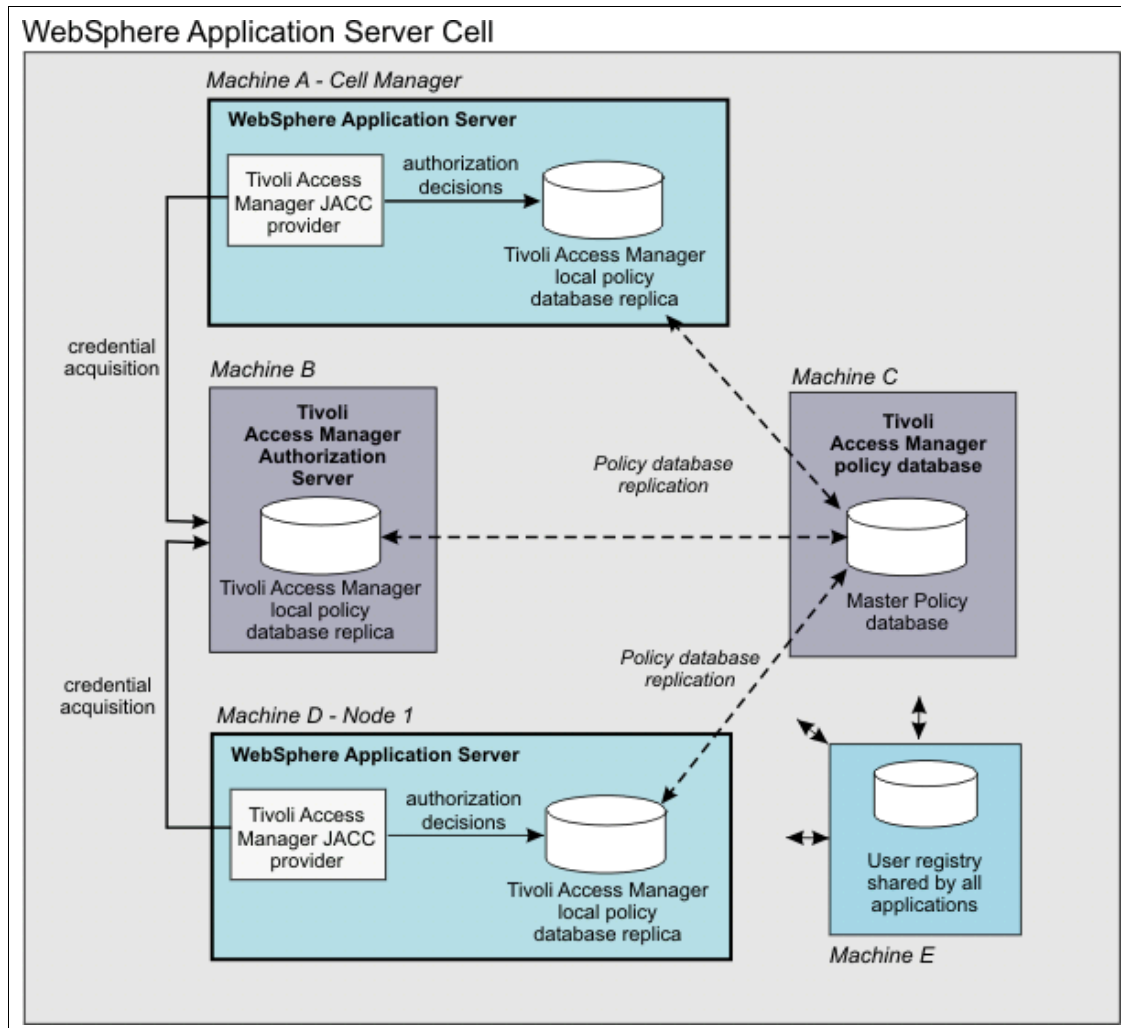


Figure 12-8 WebSphere V6.x and Tivoli Access Manager in a sample architecture

The participating WebSphere Application Servers use a local replica of the Tivoli Access Manager policy database to make authorization decisions for incoming requests. The local policy databases are replicas of the master policy database that are installed as part of the Tivoli Access Manager installation. Having policy database replicas on each participating WebSphere Application Server optimizes performance when making authorization decisions and provides failover capability.

The authorization server can also be installed on the same system as WebSphere Application Server, although this configuration is not illustrated in the diagram.

All instances of Tivoli Access Manager and WebSphere Application Server in the example architecture share the LDAP user registry on Machine E.

12.5.2 High-level components of the integration

The integration of Tivoli Access Manager in WebSphere (Tivoli Access Manager client) using the JACC model to perform access checks can be divided into the following high level components:

- ▶ Runtime
- ▶ Client configuration
- ▶ Authorization table support
- ▶ Access check
- ▶ Authentication using the PDLoginModule

Tivoli Access Manager runtime support of JACC

Tivoli Access Manager implements the PolicyConfigurationFactory and the PolicyConfiguration interfaces, as required by JACC. During the application installation, the security policy information in the deployment descriptor and the authorization table information in the binding files are propagated to the Tivoli JACC provider using these interfaces. The Tivoli JACC provider stores the policy and the authorization table information in the Tivoli Access Manager policy server by calling the respective Tivoli Access Manager APIs. The information is stored in the security policy database in the Tivoli Access Manager policy server.

Tivoli Access Manager client configuration

The Tivoli Access Manager client can be configured using either the administrative console or `wsadmin` scripting. The administrative console panels for the Tivoli Access Manager client configuration are located under the Security center panel. The Tivoli client must be set up to use the Tivoli JACC provider. You can perform the setup using `wsadmin`, either before or during the time of WebSphere Application Server configuration.

Authorization table support

Tivoli Access Manager uses the RoleConfiguration interface to ensure that the authorization table information is passed to the Tivoli Access Manager provider when the application is installed or deployed. When an application is deployed or edited, the set of users and groups for the user or group-to-role mapping are obtained from the Tivoli Access Manager server, which shares the same LDAP server as WebSphere Application Server. This sharing is accomplished by plugging in to the application management users or groups-to-role administrative console panels. The management APIs are called to obtain users and groups rather than relying on the WebSphere Application Server-configured LDAP registry.

Access check

When WebSphere Application Server is configured to use the JACC provider for Tivoli Access Manager, it passes the information to Tivoli Access Manager to make the access decision. The Tivoli Access Manager policy implementation queries the local replica of the ACL database for the access decision.

Authentication using the PDLoginModule module

The custom login module in WebSphere Application Server can perform the authentication. This login module is plugged in before the WebSphere Application Server-provided login modules. The custom login modules can provide information that can be stored in the Subject. If the required information is stored, no additional registry calls are made to obtain that information.

As part of the JACC integration, the Tivoli Access Manager-provided PDLoginModule module is also used to plug-in to WebSphere Application Server for both LTPA and Simple WebSphere Authentication Mechanism (SWAM) authentication.

Attention: Note that Simple WebSphere Authentication Mechanism is deprecated in WebSphere Application Server V6.1 and is to be removed from future versions.

The PDLoginModule module is modified to authenticate with the user ID or password. The module is also used to fill in the required attributes in the Subject, so that no registry calls are made by the login modules in WebSphere Application Server. The information that is placed in the Subject is available for the Tivoli Access Manager policy object to use for access checking.

12.6 WebSEAL authentication

This section focuses on the authentication from a client to the Access Manager WebSEAL reverse proxy. It describes the configurations available and provides instructions for the most common authentication scenarios.

12.6.1 Basic authentication

By default, WebSEAL is configured for authentication over HTTPS using basic authentication. If you want to enable basic authentication over HTTP, which we do *not* recommend, the default shown in Example 12-2 must change accordingly to either *http* or *both*. Edit the `webseald.conf` file and locate the `[ba]` stanza.

Example 12-2 Basic authentication

```
# Enable authentication using the Basic Authentication mechanism
# One of <http, https, both, none>
ba-auth = https
```

If you decide to use basic authentication in your configuration you may want to consider changing the security realm displayed in the dialogue window by changing the `basic-auth-realm` setting.

Figure 12-9 on page 328 shows the result of changing `basic-auth-realm` to a parameter which is performed in Example 12-3.

Example 12-3 WebSEAL authentication realm

```
# Realm name. This is the text that is displayed in the
# browser's dialog box when prompting the user for login data.
# By default, the string 'Access Manager' is used.
basic-auth-realm = ITS0 Applications
```

Restart the WebSEAL instance for the changes to take effect. To test the settings point your browser to the root of your WebSEAL server.

After the user is logged in, as shown in Figure 12-9 on page 328, the only way to close the WebSEAL session is that the user has to close the browser. The browser caches the credentials and automatically authenticates the user again even if WebSEAL closed the session.



Figure 12-9 Modified basic authentication dialogue

Tip: Basic authentication is often considered less secure than form authentication. This is due to the fact that the basic authentication header is sent on every request, whereas form authentication only sends login data once during the POST and then keeps session state via session cookie or SSL session-ID.

Form authentication has flaws as well. For example, if cookies are being used for the session, someone can steal a user's session cookie. With this cookie, unless they managed to get the initial POST request which contained the user's identity, they can only steal the user's session, not their identity.

No matter what authentication mechanism you use, a secure transport protocol such as an SSL must always be used.

12.6.2 Form-based authentication

To configure form-based authentication in WebSEAL, edit the `webseald.conf` file and then restart WebSEAL. Open the `webseald.conf` file and locate the `[ba]` stanza and set `ba-auth = none`, then locate the `[forms]` stanza and change it as shown in Example 12-4.

Example 12-4 Forms authentication

```
# Enable authentication using the forms authentication mechanism
# One of <http, https, both, none>
forms-auth = https
```

Restart your WebSEAL instance for the changes to take effect. Then test the configurations by accessing a protected page.

After the user is logged in and you want to close the WebSEAL session, the user must close the browser or preferably the application can redirect the user to the pkmslogout page. After the user reaches this page, WebSEAL destroys the session and displays the logout message.

Tip: If you are going to use form-based authentication, you can tailor your login and logout pages to match your application design by modifying the `login.html` and `logout.html` in the `<webseal_instance_root>/lib/html/C/directory`.

12.6.3 Client certificate-based authentication

To configure certificate-based authentication in WebSEAL, edit the `webseald.conf` file and then restart WebSEAL. The use of certificates to authenticate clients requires server and client configuration on the WebSEAL side.

1. Open the `webseald.conf` file.
2. Locate the `[ba]` stanza and set the `ba-auth=none` entry.
3. Locate the `[certificate]` stanza and change it as shown in Example 12-5.

Example 12-5 Certificate authentication

```
# When to accept a certificate from HTTPS clients. Options are:
# never          Never request a client certificate.
# required       Always request a client certificate. Do not accept
the             connection if the client does not present a
#               certificate.
# optional       Always request a client certificate. If presented,
use it.
# prompt_as_needed Certificates will only be prompted for and processed
when
#               certificate authentication is necessary (due to an
ACL or
#               POP check failure).
accept-client-certs = required
```

4. Find the [authentication-mechanisms] stanza. Uncomment the line and change <cert-ssl-library> for your cert-ssl library.

In Example 12-6, you see the change for AIX. See *WebSEAL Administration Guide, IBM Tivoli Access Manager for e-Business V6.0*, SC32-1687, for information about the specific libraries for the operating system used.

Example 12-6 Certificate authentication library

```
# Certificates
cert-ssl          = libsslauthn.a
```

5. Create a client certificate in the client browser:
 - If you are using self-signed certificates, load the certificate into the WebSEAL keystore as a signer certificate.
 - If you are using your own Certificate Authority (CA), the CA public key certificate is loaded in the WebSEAL keystore as a signer certificate. WebSEAL does a one-to-one DN matching of the certificate with LDAP.

In this sample, we use a self-signed certificate:

- a. Create the user for the sample user01, with the **user create** command in **pdadmin**, the Tivoli Access Manager Administration Command Line Interface (CLI). Ensure that the user is valid by using the **user modify** command and viewing the information about the new user with the **user show** command in **pdadmin**. See Example 12-7.

Example 12-7 Access Manager user show command

```
# pdadmin -a sec_master
Enter Password:
pdadmin sec_master> user create -no-password-policy user01
cn=user01,ou=users,o=itso,c=US user01 " " test
pdadmin sec_master> user modify user01 account-valid yes
pdadmin sec_master> user show user01
Login ID: user01
LDAP DN: cn=user01,ou=users,o=itso,c=US
LDAP CN: user01
LDAP SN:
Description:
Is SecUser: Yes
Is GSO user: No
Account valid: Yes
Password valid: Yes
```

- b. Create a self-signed certificate that matches the following DN:

```
LDAP DN: cn=user01,ou=users,o=itso,c=US
```

You can also create the self-signed certificate by using the iKeyman tool. For more information, see *WebSphere Security Fundamentals*, REDP-3944.

6. Extract the certificate as Base64-encoded American Standard Code for Information Interchange™ (ASCII) data (*.arm file) to import it into the WebSEAL keystore later.
7. Export the certificate as PKCS12 (*.p12 file) to import it into the browser.
8. Use the iKeyman utility and open the WebSEAL keystore in <webseal_instance_root>\certs\pdsrv.kdb. The default password is pdsrv.
For example, depending on your operating system:
 - For Linux, the keystore is in
/var/pdweb/www-<instance>/certs/pdsrv.kdb.
 - For Microsoft Windows, it is in C:\Program
Files\Tivoli\PDWeb\www-<instance>\certs\pdsrv.kdb.
9. Import the certificate that you exported with the .arm extension.
10. Restart the WebSEAL instance to make the changes take effect.
11. Load the certificate into your browser, and use the PKCS12 certificate (*.p12 file).
12. Test the configuration by accessing a secured resource with your browser.
You must be able to login without entering the user name or password.
Depending on your security settings and browser, you might see a certificate request that allows you to choose the certificate to use.

12.6.4 Token authentication

Token authentication is used in a two-factor authentication and is used when users must provide two forms of identification, for example, a single factor of identification, such as a password, in addition to a second factor in the form of an authentication token. The two-factor method is based on information that the user knows in addition to something the user possesses. It provides a more reliable level of user authentication than reusable passwords.

Tivoli Access Manager provides a built-in two-factor authentication library, xtokenauth. It is a client implementation for the RSA SecurID token authentication server (ACE/Server) and is written against the RSA authorization API. WebSEAL provides RSA token authentication client (ACE/Agent) functions, and is certified as SecurID Ready.

By default, this built-in shared library for token authentication is hard-coded to map SecurID (RSA) token passcode data. This default token authentication

mechanism expects the user name used by the client to map to an existing user account in the Access Manager LDAP registry.

For information about configuring token authentication, see *WebSEAL Administration Guide, IBM Tivoli Access Manager for e-Business V6.0*, SC32-1687.

12.6.5 HTTP header authentication

Tivoli Access Manager WebSEAL provides an authentication module that authenticates users based on information obtained from custom HTTP headers supplied by the client or a proxy agent. This module consists of a mapping function that maps header data to an Access Manager identity.

WebSEAL trusts that this custom HTTP header data is the result of a previous authentication. The WebSEAL authentication module is built specifically to map data obtained from Entrust Proxy headers. When you enable HTTP header authentication using the built-in authentication module, you must disable all other authentication methods. You must accept connections only from the Entrust Proxy. Disabling other authentication methods eliminates methods that can be used to impersonate custom HTTP header data.

For further information about configuring HTTP header authentication, see *WebSEAL Administration Guide, IBM Tivoli Access Manager for e-Business V6.0*, SC32-1687.

12.6.6 Kerberos and SPNEGO authentication

WebSEAL supports the SPNEGO protocol and Kerberos authentication for use with Windows clients to achieve Windows desktop SSO. The SPNEGO protocol allows for a negotiation between the client (browser) and the server regarding the authentication mechanism to use. The client identity presented by the browser can be verified by WebSEAL using Kerberos authentication mechanisms.

WebSEAL's support for Kerberos authentication has been implemented specifically to support a Windows desktop SSO solution. This solution requires that you configure the WebSEAL server into an Active Directory domain, and that WebSEAL access a Kerberos Key Distribution Center (KDC). In addition, the Internet Explorer client must be configured to use the SPNEGO protocol and Kerberos authentication when contacting WebSEAL.

For further information about configuring Kerberos SPNEGO authentication, see *WebSEAL Administration Guide, IBM Tivoli Access Manager for e-Business V6.0*, SC32-1687.

12.6.7 External authentication interface

The external authentication interface is a plug-in point for custom login modules. With this interface, third-party systems can supply an authenticated identity to WebSEAL and Web-server plug-ins. The identity information is then used to generate a credential.

This extended authentication functionality is similar to the existing custom authentication module capability provided by the Web security external authentication C API. However, the external authentication interface allows the user identity to be supplied in HTTP response headers rather than through the authentication module interface.

12.6.8 Combining authentication types using step-up authentication

One advantage of using WebSEAL to protect WebSphere Application Server profiles is that WebSEAL can enforce step-up authentication. Step-up authentication is not its own authentication type like BA. Instead it is designed to enforce a higher level of authentication for highly confidential resources. For example, a user might be logged into the SSO domain with a user name and password. When the user tries to access a highly confidential resource, they must also present a certificate.

One of the main concepts of step-up authentication is the idea of *authentication strength*. It is a relative measure that is determined by the administrator. The only rule is that *unauthenticated* is always lower than all other authenticated levels. Therefore, for example, an administrator might decide that their environment has the following authentication strengths (with unauthenticated always at 0):

- ▶ Password authentication (only form authentication is a support for password authentication, not BA)
- ▶ SSL (client certificate) authentication
- ▶ Token card (SecurID, and so on) authentication
- ▶ External authentication using EAI

These authentication strengths are specified in the WebSEAL configuration file. The authentication level is then applied to a sensitive resource object by using a POP. The full process of deploying a step-up enabled environment is beyond the scope of this document.

The following outline briefly indicates the tasks that must be done:

- ▶ Establishing an authentication strength policy
- ▶ Specifying authentication levels
- ▶ Specifying the authentication strength login form

- ▶ Creating a protected object policy
- ▶ Specifying network-based access restrictions
- ▶ Attaching a protected object policy to a protected resource
- ▶ Enforcing user identity match across authentication levels
- ▶ Controlling the login response for unauthenticated users

This process is not trivial. You must take great care during planning and implementation. For more information about step-up authentication and how to use it, see the “Authentication strength policy (step-up)” topic in the Tivoli Access Information Center, on the Web at the following address:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame.doc/am61_webseal_admin270.htm

Important: Step-up authentication is typically enforced by looking at the request URL and mapping it to a resource object. When using products that do not use URLs, such as portal server, use extra care to ensure that step-up authentication has occurred.

12.7 WebSEAL junctions

The purpose of authenticating to WebSEAL is to access its protected resources, although WebSEAL provides minimum Web server functionality. Most commonly the resources protected are on back-end servers. WebSEAL's connections with the back-end Web servers have constantly been referred to as *junctions*. All WebSEAL junctions are connections between a front-end WebSEAL server and a back-end Web server that might be another WebSEAL server and might go by another proxy server. Only the HTTP and HTTPS protocols are supported, and WebSEAL-to-WebSEAL connections must have the SSL enabled.

A *junction* is where the back-end server Web space is connected to the WebSEAL server at a specially designated mount point in the Access Manager Web space created in the policy server database by appropriate use of the `pdadmin` command.

The junction is a logical Web object space that is typically on another Web server, rather than the physical file and directory structure of the proxied Web server. Junctions define an object space that reflects organizational structure rather than the physical machine and directory structure commonly encountered on standard Web servers. A browser client never knows the physical location of a Web resource as WebSEAL translates the requested URL addresses into the addresses that a back-end server expects without exposing them to the client. Web objects can be moved from server to server without affecting the way the client accesses those objects.

WebSEAL attempts to pass the request to the back-end server by referencing the object in Access Manager's protected object space. If it encounters an ACL or POP on that object that requires authentication before the request can be authorized, the client is questioned. WebSEAL is configurable for several different challenge mechanisms including the default of basic authentication and form-based logon from a junctioned application. It comes with an Application Developers Kit with to build customized Cross Domain Authentication Services. Another option is to use the External Authentication Interface to write custom authentication modules.

WebSEAL junctions can also be configured to enable the creation of SSO solutions. This way, users can access resources, regardless of the security domain controls for these resources, following their initial authentication log in through WebSEAL. With the global sign-on junction option, a third-party user registry to be referenced to supply that junction with the appropriate user ID and password.

Other options involve manipulation and additions to the underlying Access Manager schema of *inetOrgPerson*, because each junction can be configured to supply any and all attributes from the schema through to the back-end server. If the login identity and passwords from the user registries of several legacy applications can be migrated into extra attributes, those applications can be accessed through WebSEAL by using only one initial login. Any further login requirements from back-end servers are handled as transparent to the user.

In addition, the Cross Domain SSO and e-Community SSO solutions allow for the transfer of Access Manager user credentials across multiple security domains. For more information, see the Tivoli documentation at the following addresses:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame.doc/am60_webseal_admin213.htm

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame.doc/am60_webseal_admin218.htm

12.7.1 Simple junctions

`pdadmin` is a simple and easy-to-use command line utility for administration. You can also use the Tivoli Access Manager Web Portal Manager, which provides a graphical interface.

Before creating junctions, log in to the secure domain by using `sec_master` user ID as follows:

```
pdadmin -a sec_master
password: Enter your password for sec_master
pdadmin sec_master>
```

You can get a list of configured WebSEAL servers by using the following server list command:

```
pdadmin sec_master> server list
default-webseald-ibm-fa4431f1a88
  default-webseald-bch884313
  ivacld-ibm-fa4431f1a88
```

From this server list output, you can choose the server that is required, for example, `default-webseald-ibm-fa4431f1a88`, for junction creation. Three options are required for creating basic WebSEAL junctions:

- ▶ `-h`: The host name of the back-end junctioned server
- ▶ `-t`: The junction transport type with the options `tcp`, `ssl`, `tcpproxy`, `sslproxy`, and `local`
- ▶ junction point name

The creation of a basic junction has the following syntax:

```
server task webseal-instance_name create -t transport_type -h host_name
jct_point_name
```

Consider the following example:

```
server task default-webseald-ibm-fa4431f1a88 create -t tcp -h
wvaa.itso.ral.ibm.com /test
```

Tip: For the `pdadmin` command line, see the following address:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame.doc/am60_webseal_admin338.htm

You can configure a junction to insert Tivoli Access Manager specific client identity and group information into the HTTP header by using the `-c` option. Then, this information can be passed to the back-end servers which can use it from the HTTP header. There are four options that you can use with `-c`:

- | | |
|------------------|---|
| iv-user | Passes the short name or the long name. Defaults to unauthenticated if the client is unauthenticated. |
| iv-user_l | Passes the complete DN of the user. |

iv-groups	Passes a list of comma-separated groups to which the client belongs.
iv-creds	An encoded opaque data structure representing an Access Manager credential. It is used by the new TAI to create a PDPrincipal object and inserts that object into the Subject.

The -c all option: The -c all option adds all these options. You can also list the options by using a comma as a separator.

A junction can be configured to supply client identity in the BA header by using the -b option when creating the junctions. This is different from the -c option discussed earlier. When configuring a junction for use with the TAI of WebSphere Application Server, configure your junction with the -b supply option. This option inserts the *dummy* password configured in the webseald.conf file in a BA header. This *dummy* password is used in the WebSEAL TAI to establish trust between the participating WebSEAL servers and WebSphere Application Servers. You can use the -f option to force a new junction to overwrite an existing junction mount point. The following example uses these options:

```
server task default-webseald-ibm-fa4431f1a88 create -t ssl -h
wvaa.itso.ral.ibm.com -f -c iv_user,iv_creds /test
```

Important: In order for a successful SSL junction to be created, WebSEAL must be able to create a valid SSL connection to the Web server or Application Server. This means that WebSEAL must be able to trust the certificate presented. If the Web server also has SSL client authentication required (which is considered good practice), the Web server must be able to trust WebSEAL's personal certificate in pdsrv.kdb.

It can also be considered beneficial to remove all other CA certificates from the Web server, except those required to trust the proxy and Application Server, so that it trusts requests from valid servers.

This creates a new junction test that overwrites the existing junction. To view the details of this junction use the **server task <webseal server> show <junction-name>** command (Example 12-8).

Example 12-8 Access Manager shows junction command

```
pdadmin sec_master> server task default-webseald-ibm-fa4431f1a88 show
/test
  Junction point: /test
  Type: SSL
  Junction hard limit: 0 - using global value
```

```
Junction soft limit: 0 - using global value
Active worker threads: 0
Basic authentication mode: filter
Forms based SSO: disabled
Authentication HTTP header: insert - iv_user iv_creds
Remote Address HTTP header: do not insert
Stateful junction: no
Boolean Rule Header: no
Scripting support: no
Preserve cookie names: no
Cookie names include path: no
Transparent Path junction: no
Delegation support: no
Mutually authenticated: no
Insert WebSphere LTPA cookies: no
Insert WebSEAL session cookies: no
Request Encoding: UTF-8, URI Encoded
Server 1:
  ID: bec450e6-20bb-11db-8b52-00145e3ee66e
  Server State: running
  Operational State: Online
  Hostname: wvaa.itso.ral.ibm.com
  Port: 443
  Virtual hostname: wvaa.itso.ral.ibm.com
  Server DN:
  Query_contents URL: /cgi-bin/query_contents
  Query-contents: unknown
  Case insensitive URLs: no
  Allow Windows-style URLs: yes
  Current requests : 0
  Total requests : 1
```

Note: If the communications channel between WebSEAL and the junctioned back-end server is not secure, you can use SSL junctions to ensure security. However, you should use SSL junctions in all situations.

12.7.2 Trust Association Interceptors and LTPA Junctions

When using WebSEAL as a perimeter authentication device for WebSphere Application Server, there must be a certain level of trust required by WebSphere of WebSEAL. The best way to ensure this trust is through TAIs or LTPA junctions.

12.7.3 Single sign-on junctions

For most cases, applications in a secured production environment behind several security measures, such as protected environments and firewalls, can safely be configured to trust WebSEAL and get the user identity by using the headers provided. However, sometimes there is a requirement to integrate back-end servers that require authentication and cannot or will not be modified to support better methods such as TAI or LTPA as mentioned previously and further explained in Chapter 13, “Trust Association Interceptors and third-party software integration” on page 353.

In these cases, WebSEAL provides mechanisms to authenticate to Web servers or application servers transparently, on behalf of the users, without them being aware that Access Manager is handling the authentication.

Basic authentication

You can configure WebSEAL junctions to supply the back-end server with original or modified client identity information. By using the set of `-b` options, you can supply specific client identity information in the HTTP BA headers. After the initial authentication between the client and WebSEAL, WebSEAL can build a new basic authentication header. The request uses this new header as it continues across the junction to the back-end server. You use the `-b` options to dictate the specific authentication information that is supplied in this new header. The following options are available:

supply	The authenticated Tivoli Access Manager user name with a static, generic dummy password. The original client password is not used in this scenario. This option is used for TAI junctions as explained previously.
ignore	Passes the original client BA header to the back-end server without interference. You can configure WebSEAL to authenticate this BA client information or ignore the BA header supplied by the client and forward the header, without modification, to the back-end server.
filter	Causes WebSEAL to remove the BA header from any client requests, ensuring that WebSEAL is the single security provider.
GSO	Is used when the back-end server requires different user names and/or passwords to authenticate. To use GSO, a GSO resource credential database must be configured. The user registry contains extra data for each user beyond the Access Manager required data. The GSO data is a list of gso resource-username-password entries.

See the Tivoli Access Manager documentation for more information about GSO.

Configuring a junction to authenticate to a server using basic authentication

In the following example, the client authenticates to WebSphere Application Server by using basic authentication. WebSphere must be configured with application security and be able to accept basic authentication.

We use the snoop application (WebSphere default application sample) to test a junction that requires basic authentication and shares the same user registry as the policy server. To create the junction:

1. From the Access Manager server, use the **pdadmin** command line tool and log on as the `sec_master`.
2. List the WebSEAL servers to find the one to configure the junction:

```
pdadmin sec_master> server list
default-webseald-bch884313
ba-webseald-bch884313
forms-webseald-bch884313
```

For this example, use the server `ba-webseald-bch884313`, which listens on IP:port `9.42.171.135:444` on HTTPS and is configured to use basic authentication.

3. Create a junction to the WebSphere Server that listens on IP:port `9.42.171.159:443` on HTTPS. The junction uses the `-b ignore` option to pass the BA header from WebSEAL to WebSphere transparently. Enter the following command in the **pdadmin** command line:

```
pdadmin sec_master> server task ba-webseald-bch884313 create -t ssl
-h 9.42.171.159 -p 443 -b ignore /SnoopApp
```

If you access the snoop application directly by using the following address, the basic authentication challenge comes from WebSphere:

```
https://9.42.171.159:443/snoop
```

To test the junction, access the same application through the WebSEAL server. Enter the following address in your browser:

```
https://9.42.171.135:444/SnoopApp/snoop
```

You must be presented with the WebSEAL basic authentication header as shown in Figure 12-9 on page 328.

Enter a valid user name and password from the LDAP registry, for example, `user01/test`. WebSEAL authenticates to WebSphere and presents the snoop

servlet output, as shown in Figure 12-10. The address is for WebSEAL, and the address that was requested to WebSphere is in the response.

Snoop Servlet - Request/Client Information

Requested URL:

https://9.42.171.159/snoop/ping

Servlet Name:

Snoop Servlet

Request Information:

Request method	GET
----------------	-----

Figure 12-10 WebSEAL basic authentication SSO

Form-based authentication

Enabling WebSEAL to complete a form-based challenge from a back-end application can be a powerful tool for integration with legacy systems. However, WebSEAL does not keep that user's password because it cannot authenticate to the back-end resource as a confirmation that it is the user. You can correct this problem if you have a trusted user and password for each application and then pass the user's credentials in as headers.

To enable SSO forms authentication to a back-end application, the Access Manager administrator must perform two tasks. Firstly, a configuration file must be created defining to WebSEAL how to identify a login form when it is received from the back-end application and which fields in the back-end server form are relevant for the authentication. Secondly, a junction must be created to the back-end Web server using the `-s` option, which specifies the location of the configuration file. After this is completed, WebSEAL provides login support for Access Manager users to the back-end WebSphere application.

For further information about enabling single-sign on forms authentication, see the *Access Manager for e-business WebSEAL Administrators Guide*.

Tip: For this example, an `ITSOBank_latest_with_all_security_defined.ear` file is required from the Web chapter code that is deployed and configured in *IBM WebSphere Application Server V6.1 Security Handbook*, SG24-6316. You must also specify users or groups for security roles.

Creating the form-based authentication configuration file

The purpose of the configuration file for SSO forms authentication is to define the following patterns and fields to WebSEAL:

- ▶ A pattern which WebSEAL can use to identify the URI which indicates a request to the back-end application for a login form.
- ▶ A pattern which WebSEAL can use to identify the login form with a page returned from the back-end application.
- ▶ A list of fields within the login form which WebSEAL is to provide the values for, and where these values are to be obtained.

Important: For security reasons WebSEAL never knows the password of the user because it is immediately forwarded to the authentication provider (for example, LDAP) which sends back user information and no password. This means that you cannot forward the user's password to the back-end application.

Therefore, it is beneficial to create one user for each application and then use the GSO functionality for requests. For more information about GSO, see the following Web site:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame.doc/am60_webseal_admin205.htm

Example 12-9 is the source for a sample login page for the ITSOBank sample application. It is a summary of the version from the `ITSOBank_latest_with_all_security_defined.ear` files in Web chapter code.

Example 12-9 ITSOBank - login.html

```
<form method="post" action="/itsobank/j_security_check">
<table width="80%">
<tr>
<td width="20%" align="right">Userid:</td><td><input size="20"
type="text" name="j_username" maxlength="25"></td>
</tr>
```



```

<tr>
<td align="right">Password:</td><td><input size="20" type="password"
name="j_password" maxlength="25"></td>
</tr>
<tr>
<td></td>
<td>
<input type="submit" name="action" value="Login">&nbsp;<input
type="reset" name="reset" value="Clear">
</td>
</tr>
</table>
</form>

```

In our form, there are two input fields, `j_username` and `j_password`. These are the two fields that WebSEAL must complete. Example 12-10 shows the SSO forms configuration file.

Example 12-10 SSO forms authentication configuration file

```

[forms-sso-login-pages]
login-page-stanza = login-itsobank
[login-itsobank]
login-page = /itsobank/login/login.html
login-form-action = *
gso-resource =
argument-stanza = args-for-login-itsobank
[args-for-login-itsobank]
j_username = cred:azn_cred_authzn_id
# passwords for all users must be the same on back end
j_password = string:static-passw0rd

```

In Example 12-10, we have configured one login form page, which is `login-itsobank`. The URI for the login form is `/itsobank/login/login.html`. This entry defines the URI that must be intercepted by WebSEAL. When a request is received for this URI, WebSEAL intercepts the form, and returns to the ITSOBank application that the user ID defined for this Access Manager user and the fixed password `test`.

The forms configuration also allows you to use GSO resources, although we did not use it in this example. The users can easily be created in the back-end systems with the same password or no-password if the infrastructure and WebSEAL provide a secure environment. This sample is similar to the `-b supply` option discussed earlier.

To create the junction:

1. Create the file `itsobank.fsso.conf` in the `WebSEAL_install_directory/etc` directory. Ensure that the file is readable by the `ivmgr` user.
2. On the Access Manager server launch `pdadmin` and log on as `sec_master`.
3. Find the WebSEAL server that you are going to use:

```
pdadmin sec_master> server list
default-webseald-bch884313
ba-webseald-bch884313
forms-webseald-bch884313
```

In this case, we use the server `ba-webseald-bch884313`, which listens on IP:port `9.42.171.135:444` on HTTPS and is configured to use basic authentication.

4. Create a junction to the WebSphere Application Server that listens on IP:port `9.42.171.159:443` on HTTPS. The junction uses the `-s` option to indicate the forms SSO file.

Enter the following command:

```
pdadmin sec_master> server task ba-webseald-bch884313 create -t ssl
-h 9.42.171.159 -p 443 -f -S "C:\Program
Files\Tivoli\PDWeb\etc\itsobank.fsso.conf" /ITSOBank
```

5. To test the junction, access the WebSEAL server, the ITSOBank junction and the protected resource `/itsobank/transfer/customertransfer.html`.

12.8 Integrating IBM WebSphere Application Server and Tivoli Access Manager

To integrate WebSphere Application Server applications with Tivoli Access Manager, you must distinguish between the following types of integration:

- ▶ Integration of new applications that are to be developed or existing applications that are to be changed.
- ▶ Integration of existing applications without any changes.

For Java applications Access Manager provides a pure Java version of the Authorization API (aznAPI) providing classes, which are `PDPermission`, `PDPrincipal`, and `PDLoginModule`.

`PDPermission` is usable in both a JAAS and non-JAAS environment. You can use these methods to secure new applications or to adjust existing applications. Often, there are already existing J2EE applications secured by WebSphere

declarative security also using J2EE security methods alternatively. When the embedded Tivoli Access Manager is enabled in WebSphere Application Server, it imports WebSphere security definitions into the Access Manager's object space. The function that determines whether a user is granted any permitted roles is then handled by Tivoli Access Manager.

12.8.1 aznAPI

aznAPI is an API specifically designed for Access Manager. It has been approved by the OpenGroup as the standard implementation of the Authorization Model. Access Manager provides a C and a Java version of the API. The aznAPI Java classes are basically Java wrappers for the original C API. WebSphere applications may use the aznAPI to retrieve fine-grained authorization information about a user. The authorization API consists of a set of classes and methods that provide Java applications with the ability to interact with Access Manager to make authentication and authorization decisions.

The aznAPI classes are installed as part of the Tivoli Access Manager Java runtime component which comes with WebSphere Application Server V6.1. These classes communicate directly with the Tivoli Access Manager authorization server by establishing an authenticated SSL session with the authorization server process.

More information: For more information about Java development with Tivoli Access Manager security and administration, see the following documents:

- ▶ *IBM Tivoli Access Manager: Authorization Java Classes Developer Reference, Tivoli Access Manager V6.0, SC32-1695*
- ▶ *IBM Tivoli Access Manager: Administration Java Classes Developer Reference, Tivoli Access Manager V6.0, SC32-1693*

12.8.2 Tivoli Access Manager and J2EE security

The Java security is policy based, which means that authorization to perform an action is not hard coded into the Java run time or executables. Instead, the Java environment consults policy external to the code to make security decisions. In the simplest case, this policy is implemented in a flat file, which somewhat limits its scalability and also adds administrative overhead.

To overcome the flat file implementation of Java 2 policy, and to converge to a single security model, the authorization framework provided by Access Manager can be leveraged from inside a normal Java security check. As mentioned earlier, the most natural and architecturally pleasing implementation of this support is the JAAS framework.

Support for this standard provides the flexibility for Java developers to leverage fine-grained usage of security and authorization services as an integral component of their application and platform software. The Tivoli Access Manager provides the *PDLoginModule* login module which is enabled when the embedded Tivoli Access Manager is enabled in WebSphere V6.

With the Java 2 and JAAS support delivered with the embedded Tivoli Access Manager, Java applications take advantage of the following benefits:

- ▶ Use the Tivoli Access Manager to acquire authentication and authorization credentials from Access Manager.
- ▶ Use the *PDPermission* class to request authorization decisions.

This offers Java application developers the following advantages:

- ▶ The security of Java applications that use *PDPermission* is managed using the same, consistent model as the rest of the enterprise.
- ▶ Java developers are not required to learn anything beyond Java 2 and JAAS.
- ▶ Updates to security policy involve Tivoli Access Manager-based administrator actions, rather than any code updates.

Today, JavaServer Pages (JSP), servlets, and Enterprise JavaBeans (EJB) can take direct advantage of these services. When WebSphere containers support Java 2 security, EJB developers can avoid the requirement to make security calls by having the containers handle security while they focus on business logic.

There are two options for implementing fine-grained authorization (at the level of actions on objects) within servlets and EJBs:

- ▶ Given the Access Manager credential information (EPAC) passed in the HTTP header, the servlet or the EJB must use the *PDPermission* class extensions directly to query Access Manager for access decisions. The access enforcement is still the responsibility of the application, servlet, or EJB.
- ▶ Develop a proxy bean (a session bean) within an EJB. This proxy bean intercepts all method invocations and communicates with Access Manager (using the *PDPermission* class) to obtain the access decision and enforce it.

12.8.3 Embedded Tivoli Access Manager in WebSphere Application Server V6.1

If the application is designed as a J2EE application, it relies on the J2EE security methods to get a user ID and role. Tivoli Access Manager for WebSphere Application Server provides container-based authorization and centralized policy

management for WebSphere Application Server V6.1. Tivoli Access Manager for WebSphere Application Server is implemented as an Access Manager aznAPI application which runs on the WebSphere Application Server instance.

Access Manager for WebSphere Application Server supports applications that use the J2EE Security Classes without requiring any coding or deployment changes to the applications.

Tivoli Access Manager for WebSphere Application Server is used to evaluate access requests from a user to protected resources based on the following tasks:

- ▶ Authentication of the user.
- ▶ Determination of whether the user has been granted the required role by examining the WebSphere deployment descriptor.
- ▶ The WebSphere container using Tivoli Access Manager to perform role membership checks for security code added directly into an application (programmatic security).

Enabling the Embedded Tivoli Access Manager

To configure the WebSphere to use the Tivoli Access Manager APIs and the Access Manager JACC implementation, enable the embedded Tivoli Access Manager in WebSphere Application Server V6.1. For this scenario, Tivoli Access Manager and WebSphere Application Server must share the same user registry.

To enable embedded Tivoli Access Manager:

1. Create a Tivoli Access Manager user, for example, wstam. Enter the following command in **pdadmin**:

```
pdadmin sec_master> user create wstam -no-password-policy  
cn=wstam,ou=users,o=itso,c=us wstam wstam test
```

```
user modify wstam account-valid yes
```

```
user modify wstam description "Access Manager user ID for WebSphere"
```

2. Enable Tivoli Access Manager JACC provider. See 14.6, "Integrating Tivoli Access Manager as an external JACC provider" on page 420 for further details. Make sure that Administrative and Application Security is enabled, save your settings, and restart WebSphere. After enabling Tivoli Access Manager, you must be able to see the following servers. For example, our environment show three new servers in the **pdadmin** utility:

```
pdadmin sec_master> server list  
Authn_418532961-wvaa  
Authz_293977456-wvaa  
JACC_293977456-wvaa
```

Migration of roles and principals to groups

When the embedded Tivoli Access Manager included in WebSphere Application Server is installed, only the Administration Console roles and resources are migrated to Tivoli Access Manager.

In WebSphere Application Server V5.0 there was a migration utility to migrate the application roles to the Tivoli Access Manager object space. In WebSphere Application Server V6.1, if JACC is enabled, every time a new application is deployed all the required objects and mappings are created. To create the resources required for the applications already installed in your Application Server, you can either reinstall the applications or manually propagate the policy information using `wsadmin`. If you choose reinstall, WebSphere creates the required objects in the Tivoli Access Manager object space during deployment, you also have the option to assign Application Roles to Tivoli Access Manager users or groups. For information about manual policy propagation see 14.5.5, “Manual policy propagation” on page 418.

Deployment descriptor mapping in Tivoli Access Manager

WebSphere maps the application deployment descriptor in a peculiar way that might not be obvious at first sight. To show how this works, we use some of the objects created after installing the `Itsohel10` application. Figure 12-11 on page 349 represents a part of the Tivoli Access Manager object space.

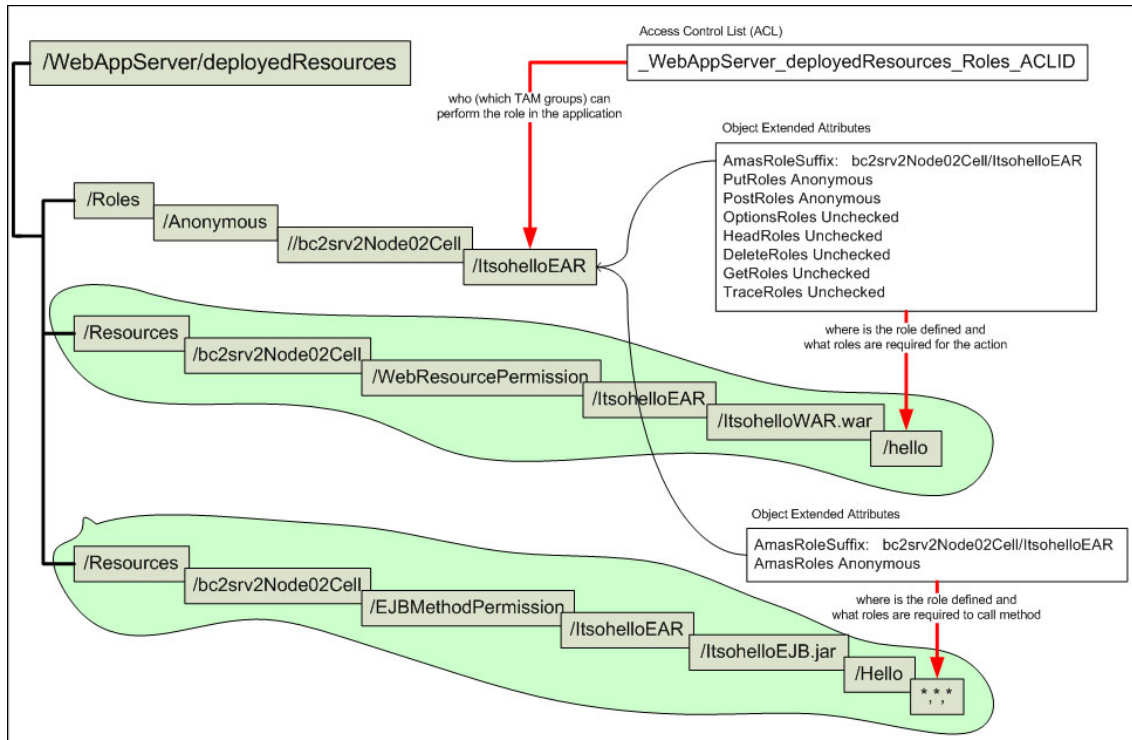


Figure 12-11 Deployment descriptor and J2EE roles in Tivoli Access Manager object space

When the application is deployed all the application roles are created under /webSppServer/deployedResources/Role, the mapping is of the form /<role_name>/<server_node_cell>/<application_name>.

Although there are some extra objects under this, such as the Access Manager ACL that governs the role, in other words, the Tivoli Access Manager users and groups assigned to the role, is attached at the <application_name> level. In the example in this book, the ACL attached to the following changes to reflect the Tivoli Access Manager users and groups mapped to the role in the Administration Console:

```
/WebAppServer/deployedResources/Roles/Anonymous/wvaaNode02Cell/ItsohelloEAR
```

Also the Administration console reads the mappings by looking at the ACLs attached to that object.

Note: If you update the ACLs using the Tivoli tools, the WebSphere server has to be restarted to re-read the ACLs attached to the J2EE roles.

During deployment WebSphere translates each Web resource in the web.xml descriptor to an object under the /webSppServer/deployedResources/Resources directory. The mapping is of the following form:

```
<server_node_cell>/<application_name>/<???.>/<application_name>/<war_file_name>/<resource>
```

In the sample, the ItsHello application receives the information from the web.xml file. The relevant information in the web.xml file (Figure 12-12) translates to the following object:

```
/WebAppServer/deployedResources/Resources/wvaNode02Cell/WebResourcePermission/ItsHelloEAR/ItsHelloWAR.war/hello
```

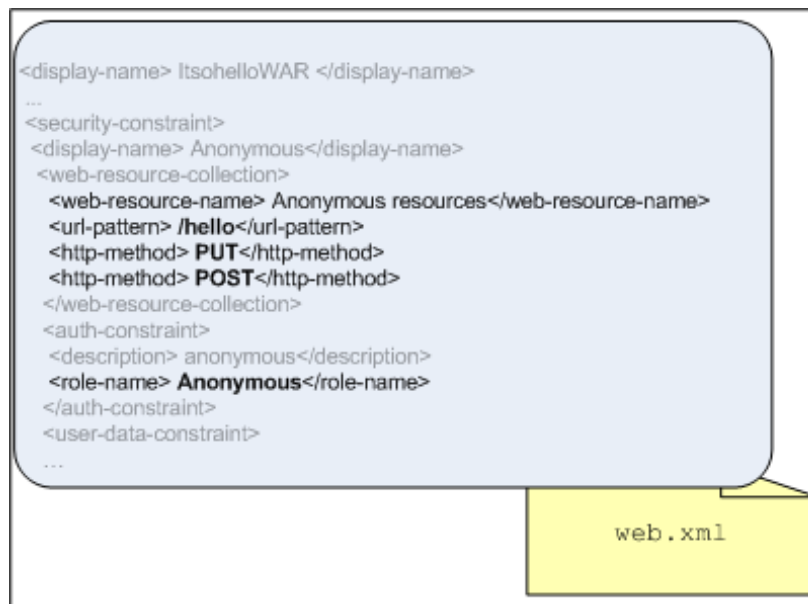


Figure 12-12 The web.xml deployment descriptor

The Web deployment descriptor lists each action allowed in the Web objects according to the HTTP Action. In this case the descriptor allows the Anonymous role PUT and POST to the /hello Web resource and in Tivoli Access Manager the object is created with extended attributes to represent this action as described in Figure 12-11 on page 349.

Also, for the EJB methods, WebSphere translates each method signature in the EJB as described in the `ejb-jar.xml` descriptor to an object under `/webSppServer/deployedResources/Resources`, the mapping is of the form:

```
<server_node_cell>/<EJBMethodPermission>/<application_name>/<ejb_filena  
me>/<ejb_name>/<bean_name>/<method_signature>
```

In the sample, the `ItsHello` application receives the information from the `ejb-jar.xml` file. The relevant information in the `ejb-jar.xml` file the can be seen in Figure 12-12. As you can see in Figure 12-11 on page 349 translates to:

```
/WebAppServer/deployedResources/Resources/wvaanode02Cell/EJBMethodPermi  
ssion/ItsHelloEAR/ItsHelloEJB.jar/Hello/*,*,*
```

The EJB deployment descriptor lists the roles that are allowed to invoke the methods in the bean, in our case the descriptor allows the `Anonymous` role to invoke any method in the `Hello` bean.

This is shown in both the descriptor in Figure 12-13 and the Tivoli Access Manager object space in Figure 12-11 on page 349.



Figure 12-13 The `ejb-jar` deployment descriptor



Trust Association Interceptors and third-party software integration

This chapter discusses the use of Trust Association Interceptors (TAIs) with WebSphere Application Server V6.1.

13.1 Trust Association Interceptor

In many enterprises, you can use third-party applications, such as Web proxy authentication servers like WebSEAL, to perform authentication. When WebSphere is deployed in such scenarios, it is essential to establish a trust relationship between the Application Server and the third-party security software such as WebSEAL or Microsoft Windows Active Directory. This *trusted relationship* between WebSphere Application Server and the third-party software is established by using a TAI that specifically built for the product. Thus, TAI enables the integration of third-party security servers with WebSphere Application Server.

TAI is an interface that is provided by WebSphere Application Server. This interface must be implemented for the specific Web proxy servers by the vendor alone or in conjunction with IBM. The implementation of this interface determines the contract used between WebSphere Application Server and the proxy server to establish trust.

The perimeter authentication service can be any one of the following types:

- ▶ A reverse proxy such as WebSEAL
- ▶ A Web server security plug-in such as Access Manager plug-in for Web servers
- ▶ A Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) protocol enabled client, as in the case of WebSphere Application Server V6.1

This perimeter authentication service is expected to achieve the following tasks:

- ▶ Establish *trust* with WebSphere Application Server.
- ▶ Perform user authentication.
- ▶ Insert user credential information into Hypertext Transfer Protocol (HTTP) requests.

The TAI module in WebSphere Application Server is expected to perform the following tasks:

- ▶ Validate the trust of the perimeter authentication service.
- ▶ Extract the user's credential information from the request.

Introduced as part of WebSphere Application Server V5.1.1 was a new TAI interface that had significantly enhanced features. This interface introduced performance benefits that allowed the TAI module to return credential information to the application server. This means that no additional user registry searches are required by the login modules, thus reducing authentication overhead. This can be combined with WebSphere Application Server's downstream security attribute propagation services to allow information propagation.

The IBM developerWorks® article at the following address discusses the Tivoli implementation of the newer interface called *Tivoli Access Manager Trust Association Interceptor (TAI++)*:

<http://www-128.ibm.com/developerworks/tivoli/library/t-tamtai/>

com.ibm.websphere.security.TrustAssociationInterceptor: You can still use the old TAI interface `com.ibm.websphere.security.TrustAssociationInterceptor`. For more information about this version of TAI, see *IBM WebSphere V5.0 Security WebSphere Handbook Series*, SG24-6573.

13.1.1 The relatively new, enhanced TAI interface

The TAI interface introduced in WebSphere Application Server V5.1.1, *com.ibm.wsspi.security.tai.TrustAssociationInterceptor*, enhanced the original interface with the following new features:

- ▶ Support for a multi-phase negotiation during the authentication process
- ▶ *TAIResult* returned by the TAI and it indicates if more negotiation is required or the negotiation process is completed
- ▶ Trust Association Interceptor capable of asserting the userID and group information to WebSphere Application Server
- ▶ The ability to add custom information to the subject during TAI processing; can be returned as a Java Authentication and Authorization Service (JAAS) subject and can be used in application code.

There are two key methods in the new interface:

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

This method determines if the request originated from one of the proxy servers associated with the Trust Association Interceptor. The code in this method must determine whether the incoming request originated from one of the configured Proxy Servers by examining the request object. The result of this method may be `True` or `False`. `True` value tells WebSphere Application Server to continue the processing of TAI. In case of `false`, the TAI is ignored.

- ▶ `public TAIResult negotiateValidateandEstablishTrust (HttpServletRequest req, HttpServletResponse res) throws WebTrustAssociationFailedException`

The code in this method must determine whether to trust the proxy server from which the request originated. This code is proxy server specific and must authenticate the proxy server in some meaningful way.

Also this method enables the TAI to use a trust negotiation protocol, such as SPNEGO, to provide challenge and responses back to the client.

The return value of `negotiateValidateandEstablishTrust` is *TAIResult*. This object indicates the status of negotiation or the final result of negotiation. The *TAIResult* class has three static methods for creating a *TAIResult*:

▶ `create(int status)`

You can set the `int` to something other than `HttpServletResponse.SC_OK`, and the `HttpServletResponse` is sent back to the client to make another request to the TAI.

▶ `create(int status, String principal)`

You can set the `status` to `HttpServletResponse.SC_OK` and then provide the user ID or the unique ID for this user. WebSphere Application Server then queries the registry with this ID for additional information to create the credentials.

▶ `create(int status, String principal, Subject subject)`

You can set the `status` to `HttpServletResponse.SC_OK`, thus indicating that no further negotiation is required. WebSphere Application Server creates the subject by using the information provided in `principal` and `subject`.

A few additional methods on the `TrustAssociationInterceptor` interface are used for `TrustAssociationInterceptor` initialization, shut down, and identification. For more information about these methods, see the WebSphere Application Server V6.1 Information Center or the WebSphere Application Server Java API documentation.

13.2 Windows desktop single sign-on using SPNEGO

A new feature provided in WebSphere Application Server V6.1 is the ability to use single sign-on (SSO) for WebSphere applications from a Microsoft Windows desktop by using SPNEGO. The SSO is invisible to the user. Previously this function was only achieved through third-party software such as Tivoli Access Manager. Now this function is achieved by having a SPNEGO protocol enabled client, such as a .NET application or SPNEGO-enabled browser (for example, Microsoft Internet Explorer 5.5 and later or Mozilla Firefox 1.0). These establish trust and pass credentials to the application server using Kerberos tokens issued from a Microsoft Windows 2000 or 2003 Active Directory domain controller.

Figure 13-1 show the challenge-response handshake.

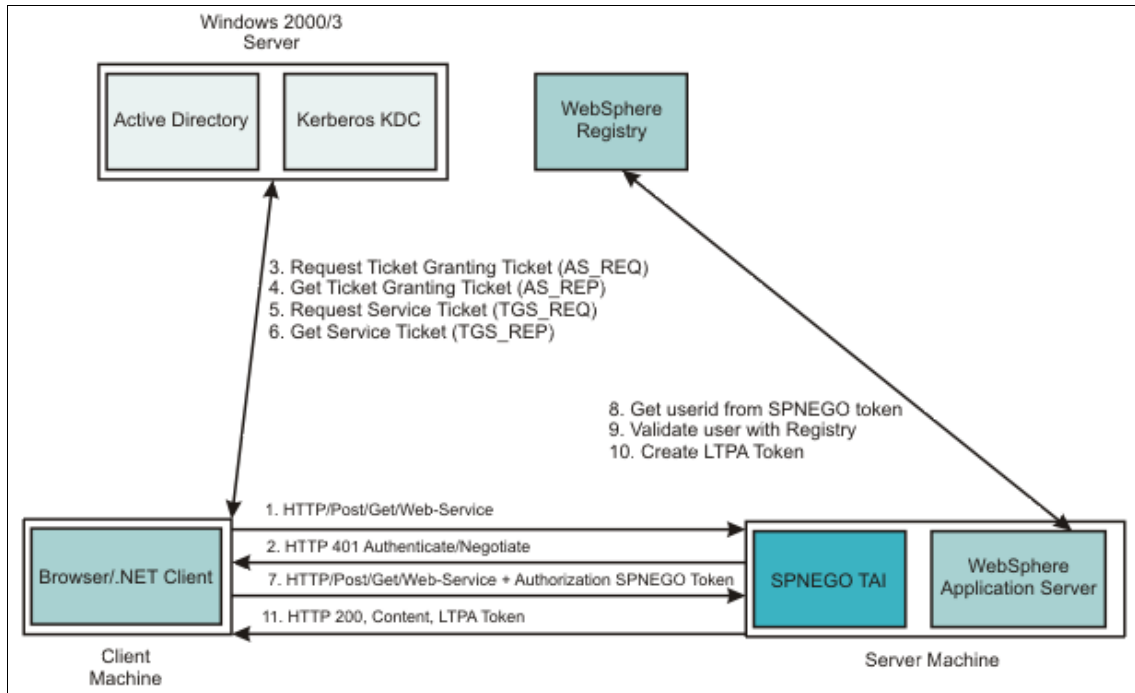


Figure 13-1 HTTP request flow when using SPNEGO TAI

This environment has a clear separation of responsibilities between the four main components:

- ▶ SPNEGO protocol enabled client
- ▶ Microsoft Windows Active Directory Domain Controller
- ▶ WebSphere Application Server with SPNEGO TAI
- ▶ The registry that WebSphere Application Server is using

The SPNEGO protocol-enabled client is responsible for creating the request. The request creation is an important concept because it forces any client developers to ensure that they implement the required functionality. The client creates the request with the help of the Kerberos Key Distribution Center (KDC) located at the Active Directory Domain Controller. This request contains the SPNEGO token, which allows the TAI to authenticate the user.

Tip: To understand the basics of Kerberos and its use within Microsoft Windows environments it, see the following Web sites:

- ▶ Kerberos: The Network Authentication Protocol
<http://web.mit.edu/Kerberos/>
- ▶ Kerberos Protocol Transition and Constrained Delegation
<http://technet2.microsoft.com/windowsserver/en/technologies/featured/kerberos/default.aspx>
- ▶ Authentication for Administrative Authority
<http://www.microsoft.com/technet/Security/bestprac/authent.aspx>

After WebSphere Application Server receives the request that contains trust information and user credentials, it validates the user against the registry. If a different registry is used for WebSphere to the Windows Domain Controller this validation can be tricky due to user name mappings. After successful validation, a Lightweight Third Party Authentication (LTPA) token is created and a session started for the user. For user mapping, see the WebSphere Application Server Information Center at the following address:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tsec_SPNEGO_tai_umapper.html

13.2.1 Lab scenario

Figure 13-2 on page 359 illustrates an environment with the following machines:

- ▶ Microsoft Windows Server® 2003 SP1 (W2K3) Active Directory Domain Controller
- ▶ Microsoft Windows Server 2003 SP1 (W2K3) domain member (client)
- ▶ SUSE® Linux Enterprise Server 9 with WebSphere Application Server V6.1
- ▶ Microsoft Windows Server 2003 SP1 (W2K3) domain member hosting WebSphere Application Server V6.1

The W2K3 Domain Controller, bchhs409.paul.itso.ral.ibm.com, is also the Domain Name System (DNS) and Kerberos Key Distribution Center. It has a W2K3 domain member, bch884314.paul.itso.ral.ibm.com, with two users, which are *alison* and *emily* who can log in to the domain.

A SUSE Linux Enterprise Server 9 workstation, cvs7240a.paul.itso.ral.ibm.com, is hosting WebSphere Application Server V6.1. WebSphere Application Server V6.1 is installed on W2K3 domain member bch884313.paul.itso.ral.ibm.com. The Active Directory Domain repository is federated into the WebSphere Application

Servers' security repository. These two WebSphere Application Server profiles are *not* part of the same cell and *do not* have any WebSphere Application Server cross-cell SSO configured.

The goal of this lab is to allow users *alison* and *emily* to successfully access WebSphere Application Server resources on either of the WebSphere Application Server profiles without re-authenticating, for example, to achieve Microsoft Windows desktop SSO.

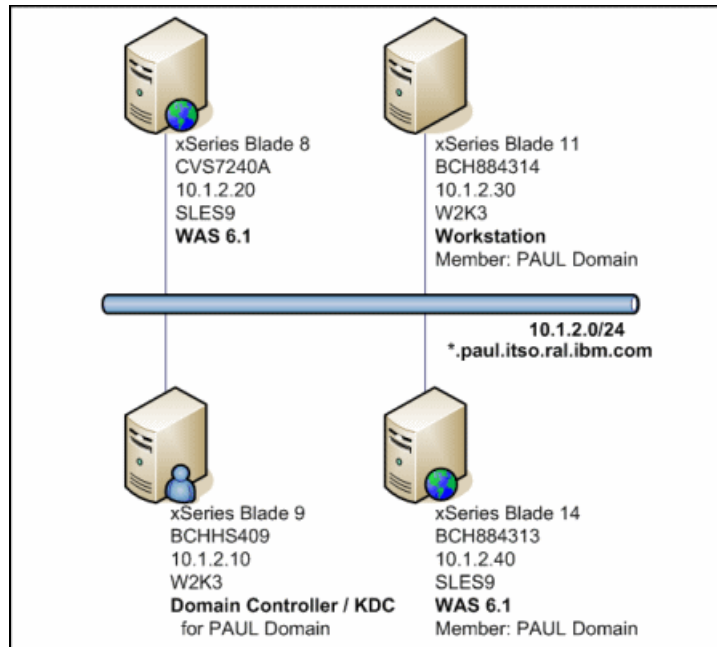


Figure 13-2 Lab environment for Microsoft Windows SSO to WebSphere Application Server using SPNEGO TAI

13.2.2 Configuring the WebSphere Application Server environment to use SPNEGO

In this section, we explain how to configure an environment for Microsoft Windows users to use SSO to WebSphere Application Server resources. The configuration of WebSphere Application Server to use SPNEGO can be complex.

Prerequisites: Prior to reading this section, you must understand how Microsoft Windows Active Directory and Kerberos work. You must also have a working Active Directory Domain that allows users to successfully log in to desktops in the domain and know how to use native Windows SSO capabilities.

The WebSphere Application Servers' repositories must also be considered before configuration. It is easiest to use the domain's Active Directory registry as either a stand-alone Lightweight Directory Access Protocol (LDAP) or federated into the federated repositories (as is done in the example). If you use another registry, depending on your environment, you might have to do name mapping for the users that are presented. Also, a process must be implemented to ensure that user mapping exists between registries. This mapping might be one-to-one or many-to-one depending upon the environments architecture.

Important: A working domain controller and at least one client computer in that domain are required, because trying to use SPNEGO from the domain controller does *not* work.

From a technical perspective, these steps have the following goals:

- ▶ Enable the SPNEGO TAIs of the WebSphere Application Server profiles to trust requests.
- ▶ Validate user credentials that come from SPNEGO-enabled clients that are part of a Microsoft Windows Active Directory Domain.

Note: Before you start, install the Windows Support Tools that come on the Microsoft Windows installation CD. The following packages are also helpful for managing Microsoft Windows Active Directory domains:

- ▶ Windows Server 2003 Resource Kit Tools (contains the `kerbtray.exe` file, which is helpful for viewing Kerberos tickets)
<http://www.microsoft.com/downloads/details.aspx?familyid=9d467a69-57ff-4ae7-96ee-b18c4790cffd&displaylang=en>
- ▶ Windows Server 2003 Administration Tools Pack
<http://www.microsoft.com/downloads/details.aspx?familyid=C16AE515-C8F4-47EF-A1E4-A8DCBACFF8E3&displaylang=en>
- ▶ Windows Server 2003 Service Pack 1 Administration Tools Pack
<http://www.microsoft.com/downloads/details.aspx?familyid=E487F885-F0C7-436A-A392-25793A25BAD7&displaylang=en>

Tip: Ensure that you complete the following checklist before continuing:

- ▶ You must have a functioning Microsoft Windows 2000/2003 Active Directory Domain including the following requirements:
 - Domain controller
 - Client workstation
 - Users that can login to the client workstation
- ▶ You must have a functioning WebSphere Application Server with Application Security enabled.
- ▶ The users from the Active Directory must be able to successfully access WebSphere Application Server's protected resources using a native WebSphere Application Server authentication mechanism such as basic authentication (BA) or forms authentication.
- ▶ The domain controller and host of WebSphere Application Server must have the same local time.
- ▶ Make sure that the clocks on the clients and WebSphere Application Server are in sync with less than five minutes.
- ▶ Ensure that the client's browsers are SPNEGO-enabled. For instructions regarding this, see the following address:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tsec_SPNEGO_config_web.html

The Firefox configuration steps do not mention that you must also set the value for the *network.negotiate-auth.trusted-uris* variable.

To configure the WebSphere Application Server environment to use SPNEGO:

1. Create a user account in the Microsoft Active Directory for the WebSphere Application Server. Depending on whether the WebSphere Application Server is hosted on a Linux, UNIX, or Microsoft Windows system, it determines the type of user created.

For Linux, the user must be created on the General tab as shown in Figure 13-3.

For Windows, the computer must already have an associated user in the *Computers* view (created when it joined the domain).

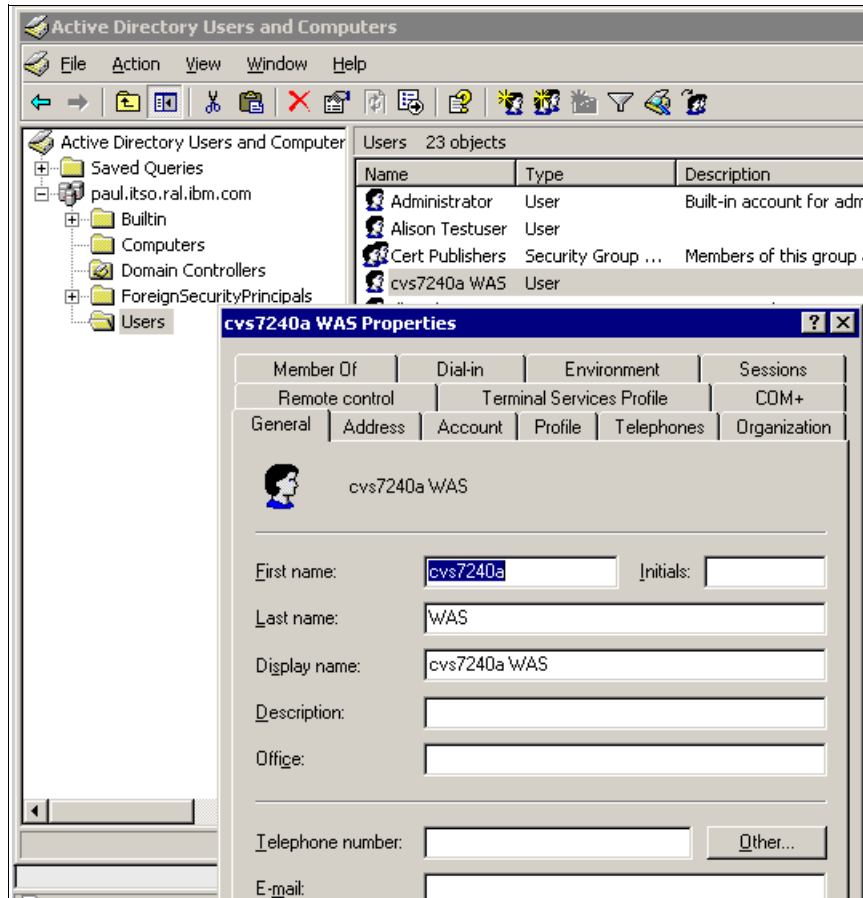


Figure 13-3 The Active Directory user corresponding to the WebSphere Application Server, CVS7420A

2. Map the user account to the Kerberos service principal name (SPN). This user account represents the WebSphere Application Server as being a Kerberized service with the Kerberos KDC. Use the `setspn` tool to establish WebSphere Application Server as the user. You can skip this step if you use the `-mapuser` switch in the following step.

Note some SPNs might already be related to the Microsoft Windows hosts that were added to the domain. You can see this by using the **setspn -L host name** command. You must still add an HTTP SPN for that host.

The usage for the **setspn** tool is as follows:

```
setspn.exe [switches data] computer_name
```

The switches are:

- R** Resets the computers ServicePrincipalName
- A** Adds an arbitrary SPN
- D** Deletes an arbitrary SPN
- L** Lists registered SPNs

You are required to specify the long host name for the principal. Example 13-1 shows the command used for cvs7240a in the example in this section.

Example 13-1 Command used for cvs7240a

```
C:\Program Files\Support Tools>setspn.exe -A  
HTTP/cvs7240a.paul.itso.ra1.ibm.com cvs7240a
```

```
Registering ServicePrincipalNames for  
CN=cvs7240a,CN=Computers,DC=paul,DC=itso,DC=ra1,DC=ibm,DC=com  
HTTP/cvs7240a
```

```
Updated object
```

Check which SPNs have been associated by using the **-L** flag as shown in Example 13-2.

Example 13-2 Checking SPNs using -L flag

```
C:\Program Files\Support Tools>setspn.exe -L cvs7240a
```

```
Registered ServicePrincipalNames for  
CN=cvs7240a,CN=Computers,DC=paul,DC=itso,DC=ra1,DC=ibm,DC=com:
```

```
HTTP/cvs7240a.paul.itso.ra1.ibm.com
```

More information: For more information about the **setspn** tool, see the following Web site:

<http://technet2.microsoft.com/WindowsServer/en/library/b3a029a1-7ff0-4f6f-87d2-f2e70294a5761033.aspx?mfr=true>

3. Create the Kerberos keytab file and make it available to WebSphere Application Server. Use the **ktpass** tool to create the Kerberos keytab file (krb5.keytab).

The following switches are the most common (use **--help** switch for a full list of switches):

-out	File name specifies the keytab to produce.
-princ	The principal_name specifies the principal name.
-pass	The password specifies the one to use. Use an asterisk (*) for a password prompt.
-mapuser	The user name maps princ to the user.
-crypto {DES-CBC-CRCIDES-CBC-MD5IRC4-HMAC-NT}	Specifies the type of cryptographic system to use.

For Microsoft Windows, this switch might look similar to the following example:

```
ktpass.exe -princ HTTP/<host>@<domain> -out C:\<new-file-name> -pass  
passw0rd -ptype KRB5_NT_SRV_HST
```

Attention: Do *not* use the **-pass** switch to reset a password for a Windows server account as this produces problems.

There are warnings regarding the *account type* and the *ptype*, but you can safely ignore them.

Example 13-3 shows the output for getting the key for bch884313.

Example 13-3 Output from the ktpass utility for Windows host

```
C:\Program Files\Support Tools>setspn.exe -L bch884313  
Registered ServicePrincipalNames for  
CN=BCH884313,CN=Computers,DC=paul,DC=itso,DC=ral,DC=ibm,DC=com:  
HTTP/bch884313.paul.itso.ral.ibm.com  
HOST/bch884313.paul.itso.ral.ibm.com  
HOST/BCH884313
```

```
C:\Program Files\Support Tools>ktpass.exe -princ  
HTTP/bch884313.paul.itso.ral.ibm.com@PAUL.ITSO.RAL.IBM.COM -out  
C:\wdir\spnego\bch884313.HTTP.key -pass passw0rd -ptype  
KRB5_NT_SRV_HST
```

NOTE: creating a keytab but not mapping principal to any user.

For the account to work within a Windows domain, the principal must be mapped to an account, either at the domain level (with /mapuser) or locally (using ksetup)

```
    If you intend to map
HTTP/bch884313.paul.itso.ral.ibm.com@PAUL.ITSO.RAL.IB
M.COM to an account through other means
    or don't need to map the user, this message can safely be ignored.
WARNING: pType and account type do not match. This might cause
problems.
Key created.
Output keytab to C:\wdir\spnego\bch884313.HTTP.key:
Keytab version: 0x502
keysize 93
HTTP/bch884313.paul.itso.ral.ibm.com@PAUL.ITSO.RAL.IBM.COM ptype 3
(KRB5_NT_SRV_HST) vno 1 etype 0x17 (RC4-HMAC) keylength 16
(0xb9f917853e3dbf6e6831ecce60725930)
```

For an application server hosted on a UNIX or Linux server, use the following command on the domain controller:

```
ktpass -princ HTTP/<WAS_Host>@PAUL.ITSO.RAL.IBM.COM -pass <password>
-out was_host.HTTP.keytab -mapuser was_host -mapOp set -princ
KRB5_NT_PRINCIPAL
```

You must see the output, which is shown in Example 13-4, such as Key created. Output keytab to was_host.HTTP.keytab, and some other information about the key. Remember the password that you used to create the keytab because it is required later.

Example 13-4 Output from the ktpass utility for a Linux host

```
C:\Program Files\Support Tools>ktpass.exe -out
c:\wdir\spnego\cvs7240a_long.HTTP.key -princ
HTTP/cvs7240a.paul.itso.ral.ibm.com@PAUL.ITSO.RAL.IBM.COM -pass
passw0rd -mapuser cvs7240a -target paul.itso.ral.ibm.com -ptype
KRB5_NT_PRINCIPAL
Using legacy password setting method
Successfully mapped HTTP/cvs7240a.paul.itso.ral.ibm.com to cvs7240a.
Key created.
Output keytab to c:\wdir\spnego\cvs7240a_long.HTTP.key:
Keytab version: 0x502
keysize 92 HTTP/cvs7240a.paul.itso.ral.ibm.com@PAUL.ITSO.RAL.IBM.COM
ptype 1 (KRB5_NT_PRINCIPAL) vno 5 etype 0x17 (RC4-HMAC) keylength 16
(0xb9f917853e3dbf6e6831ecce60725930)
C:\Program Files\Support Tools>setspn.exe -L cvs7240a
Registered ServicePrincipalNames for CN=cvs7240a
WAS,CN=Users,DC=paul,DC=itso,DC
=ral,DC=ibm,DC=com:
    HTTP/cvs7240a.paul.itso.ral.ibm.com
```

More information: For more information about the keytab files and the **ktpass** command, see the following Web address:

<http://technet2.microsoft.com/WindowsServer/en/library/64042138-9a5a-4981-84e9-d576a8db0d051033.aspx?mfr=true>

4. Configure and enable the application server and the associated SPNEGO TAI by using the administrative console or the **wsadmin** command to perform command tasks. See “Configuring SPNEGO TAI in WebSphere Application Server” on page 369.
5. Ensure that LTPA is the authentication mechanism. It is also beneficial to set the SSO domain for the LTPA cookies.
6. On the WebSphere Application Server host, install the Kerberos keytab file. Get the key file that was created in step 3 on page 364 to the host machine, and then reference the key file in the `krb5.conf` file (step 7). This file must be secured so that only the correct users can read it (the example, use **chmods** on the file to 600).

You can manipulate key files by using the `ktutil` or `ktab` utilities. For example, to ensure that it is the correct key file, use the `list` command as shown in Example 13-5.

Example 13-5 Using list command to confirm the correct key files

```
cvs7240a:/wdir/spnego # ktutil -k cvs7240a_long.HTTP.key list
cvs7240a_long.HTTP.key:
```

```
Vno  Type                Principal
   5  arcfour-hmac-md5
HTTP/cvs7240a.paul.itso.ral.ibm.com@PAUL.ITSO.RAL.IBM.COM
```

7. Update the associated Kerberos configuration (`krb5.conf`) by using the following hierarchy to find this file:
 - a. File referred to by the `java.security.krb5.conf` property
 - b. `<java.home>/lib/security/krb5.conf`
 - c. Depending on your platform, choose one of the following files:
 - The `c:\winnt\krb5.ini` file on Microsoft Windows platforms
 - The `/etc/krb5/krb5.conf` file on UNIX platforms
 - The `/etc/krb5.conf` file on Linux platforms

Edit the file to correspond to your environment. For further information about the Kerberos configuration file, see the Linux man page or the MIT Kerberos documentation. Kerberos has many Linux man pages for Kerberos utilities

such as, **ktutil**, **kinit**, **kdestroy**, **klist**, and **krb5.conf**, which you type as follows:

```
man krb5.conf
```

Note: By now you must be able to make a valid Kerberos session from the WebSphere Application Server host by using the **kinit** command (Linux and UNIX only). See 13.2.3, “Troubleshooting SPNEGO environments” on page 376, for more details.

The stanzas of interest are *libdefaults*, *realms*, and *domain_realm*. Example 13-6 shows the file that is used for the example in this section.

Example 13-6 Editing the file to correspond to the environment

```
[libdefaults]
  clockskew = 300
  default_realm = PAUL.ITSO.RAL.IBM.COM
  default_keytab_name = FILE:/wdir/spnego/cvs7240a_long.HTTP.key
  default_tkt_enctypes = des-cbc-md5 rc4-hmac
  default_tgs_enctypes = des-cbc-md5 rc4-hmac
[realms]
  PAUL.ITSO.RAL.IBM.COM = {
    kdc = bchhs409.paul.itso.ral.ibm.com:88
    default_domain = paul.itso.ral.ibm.com
  }
[domain_realm]
  .paul.itso.ral.ibm.com = PAUL.ITSO.RAL.IBM.COM
```

Another option is to use the **wsadmin** command **createKrbConfigFile**. After using this utility to create the program, you might want to edit the created file manually.

Example 13-7 shows the command for the example scenario.

Example 13-7 Command for our scenario

```
wsadmin>$AdminTask createKrbConfigFile {-krbPath /etc/krb5.conf
-realm PAUL.ITSO.RAL.IBM.COM -kdcHost bchhs409.paul.itso.ibm.com
-dns bchhs409.paul.itso.ral.ibm.com -keytabPath
/wdir/spnego/cvs7240a_long.HTTP.key}
```

Example 13-8 shows the command for the Windows host.

Example 13-8 Command for the Windows host

```
wsadmin>$AdminTask createKrbConfigFile {-krbPath "C:\Program
Files\IBM\WebSphere\AppServer\java\jre\lib\security\krb5.conf"
-realm PAUL.ITSO.RAL.IBM.COM -kdcHost bchhs409.paul.itso.ral.ibm.com
-dns bchhs409.paul.itso.ral.ibm.com -keytabPath
C:\wdir\spnego\bch884313.HOST.key}
```

Important: During the setup of this example, bch884313 threw Checksum errors when it validated the client's ticket. We resolved this by changing the encryption types as follows:

```
[libdefaults]
default_realm = PAUL.ITSO.RAL.IBM.COM
default_keytab_name = FILE:C:\wdir\spnego\bch884313.HTTP.key
default_tkt_enctypes = des-cbc-md5
default_tgs_enctypes = des-cbc-md5
kdc_default_options = 0x5480000
```

Avoid using the single data encryption standard (DES), because it is a less secure encryption.

If you have Windows 2000 or a mix of Windows 2000 and 2003, use DES-CBC-MD5. If you have a pure set of Windows 2003 servers, use RC4-HMAC. If you use RC4-HMAC, when you use the **createKrbConfigFile** command, specify the encryption type of RC4-HMAC to avoid the checksum error.

8. Configure the Java virtual machine (JVM) properties and enable SPNEGO TAI. See "Configuring JVM properties and enabling SPNEGO TAI in WebSphere Application Server" on page 374.

Important: Make sure that the machines that host the Kerberos ticket granting service, Windows Active Directory, and the WebSphere Application Server are synchronized with less than five minutes. This is essential because a machine's local time with respect to Kerberos tokens' validity is considered important.

Configuring SPNEGO TAI in WebSphere Application Server

Make sure that you have followed all the preceding steps in the previous section. Then configure the SPNEGO TAI in WebSphere Application Server:

1. Log in to the WebSphere Application Server administrative console.
2. Click **Security** → **Secure administration, applications, and infrastructure**.
3. Expand **Web security** and click **Trust association**.
4. Under the General Properties heading, select the **Enable trust association** check box and then click **Interceptors**.
5. Select **SPNEGO TAI** in the list of interceptors, *com.ibm.ws.security.spnego.TrustAssociationInterceptorImpl*. Then click **Custom properties**.

Note: Use the SPNEGO TAI admin command tasks to create custom properties so that you can avoid missing a property or a typing error.

6. Click **New**.
7. Complete the Name and Value for the text boxes. Click **OK**.
8. Repeat steps 6 and 7 for each custom property that you want to apply to the SPNEGO TAI.

The properties are entered so that each SPN can be assigned individual values. The SPN ID has to match with those specified for the JVM. See “Configuring JVM properties and enabling SPNEGO TAI in WebSphere Application Server” on page 374. Table 13-1 on page 370 shows the list of properties.

Tip: The Kerberos SPN is a string of the form HTTP/<hostname>@realm. The complete SPN is used with the Java Generic Security Service (JGSS) by the SPNEGO provider to obtain the security credential and security context that are used in the authentication process. It is set for the Active Directory user when running **setspn** or **ktpass**.

Use the **wsadmin addSpnegoTAIProperties** tool so that property names are correct. You can use the interactive mode by using the following command:

```
wsadmin>$AdminTask addSpnegoTAIProperties -interactive
```

For further information about this command, see the following address:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tsec_SPNEGO_add_wsadmin.html

Table 13-1 Attributes required for the SPNEGO TAI configuration

Attribute name	Required	Default value
com.ibm.ws.security.spnego.SPN<id>.host Name	Yes	None
com.ibm.ws.security.spnego.SPN<id>.filter Class	No	See the description that follows this table.
com.ibm.ws.security.spnego.SPN<id>.filter	No	See the description that follows this table.
com.ibm.ws.security.spnego.SPN<id>.enableCredDelegate	No	false
com.ibm.ws.security.spnego.SPN<id>.spnegoNotSupportedPage	No	See the description that follows this table.
com.ibm.ws.security.spnego.SPN<id>.NTLMTokenReceivedPage	No	See the description that follows this table.
com.ibm.ws.security.spnego.SPN<id>.trim UserName	No	true

The list in Table 13-1 is explained as follows:

- com.ibm.ws.security.spnego.SPN<id>.hostName

This attribute is required. It specifies the host name in the SPN used by the SPNEGO TAI to establish a Kerberos secure context.

Note: The host name is the long form of host name, for example, cvs7240a.paul.itso.ral.ibm.com.

- com.ibm.ws.security.spnego.SPN<id>.filterClass

This attribute is optional. It specifies the name of the Java class that is used by the SPNEGO TAI to select which HTTP requests are subject to SPNEGO authentication. If no class is specified, the default *com.ibm.ws.security.spnego.HTTPHeaderFilter* implementation class is used. The Java class that is specified must implement the *com.ibm.wsspi.security.spnego.SpnegoFilter* interface. A default implementation of this interface is provided. Specify the *com.ibm.ws.security.spnego.HTTPHeaderFilter* class to use the default implementation. This class uses the selection rules specified with the com.ibm.ws.security.spnego.SPN<id>.filter property.

– `com.ibm.ws.security.spnego.SPN<id>.filter`

This attribute is optional. It defines the filtering criteria that is used by the specified class with the previous attribute. It defines arbitrary criteria that is meaningful to the implementation class used. The `com.ibm.ws.security.spnego.HTTPHeaderFilter` default implementation class uses this attribute to define a list of selection rules that represent conditions that are matched against the HTTP request headers to determine whether or not the HTTP request is selected for SPNEGO authentication.

Each condition is specified with a key-value pair, separated from each other by a semicolon. The conditions are evaluated from left to right, because they are displayed in the specified attribute. If all conditions are met, the HTTP request is selected for SPNEGO authentication.

The key and value in the key-value pair are separated by an operator that defines which condition is checked. The key identifies an HTTP request header to extract from the request and its value is compared with the value that is specified in the key-value pair according to the operator specification. If the header that is identified by the key is not present in the HTTP request, the condition is treated as not being met.

You can use any of the standard HTTP request headers as the key in the key-value pairs. See the HTTP specification for the list of valid headers. In addition, two keys are defined to extract information from the request, which is useful as a selection criterion and not available through standard HTTP request headers.

The *remote-address* key is used as a pseudo header to retrieve the remote Transmission Control Protocol/Internet Protocol (TCP/IP) address of the client application that sent the HTTP request. The *request-URL* key is used as a pseudo header to retrieve the URL that is used by the client application to make the request. The interceptor uses the result of the `getRequestURL` operation in the `javax.servlet.http.HttpServletRequest` interface to construct the Web address. If a query string is present, the result of the `getQueryString` operation in the same interface is also used. In this case, the complete URL is constructed as follows:

```
String url = request.getRequestURL() + '?' +
request.getQueryString();
```

Table 13-2 defines the operators and conditions.

Table 13-2 Conditions and operators

Condition	Operator	Example
Match exactly	= = Arguments are compared as equal.	host=host.my.company.com
Match partially (includes)	%= Arguments are compared with a partial match being valid.	user-agent%=IE 6
Match partially (includes one of many)	^= Arguments are compared with a partial match being valid for one of many arguments specified.	user-agent%=IE 6
Does not match	!= Arguments are compared as not equal.	request-url!=noSPNEGO
Greater than	> Arguments are compared lexicographically as greater than.	remote-address>192.168.255.130
Less than	< Arguments are compared lexicographically as less than	remote-address<192.168.255.135

– com.ibm.ws.security.spnego.SPN<id>.enableCredDelegate

This attribute is optional. It indicates whether the Kerberos GSS delegated credentials are stored by the SPNEGO TAI. This attribute enables the capability for an application to retrieve the stored credentials and propagate them to other applications downstream for additional SPNEGO authentication.

This attribute requires use of the advanced Kerberos credential delegation feature and requires development of custom logic by the application developer. The developer must interact directly with the Kerberos Ticket Granting Service (TGS) to obtain a Ticket Granting Ticket (TGT) using the delegated Kerberos credentials on behalf of the end-user who originated the request. The developer must also construct the appropriate Kerberos SPNEGO token and include it in the HTTP request to continue the downstream SPNEGO authentication process, including handling additional SPNEGO challenge-response exchanges, if necessary.

- `com.ibm.ws.security.spnego.SPN<id>.spnegoNotSupportedPage`
This attribute is optional. It specifies the Web address of a resource that contains the content that the SPNEGO TAI includes in the HTTP response that the (browser) client application displays if it does not support SPNEGO authentication. It can specify a Web (`http://`) or a file (`file://`) resource. WebSphere has a default page that is returned.
- `com.ibm.ws.security.spnego.SPN<id>.NTLMTokenReceivedPage`
This attribute is optional. It specifies the Web address of a resource that contains the content that the SPNEGO TAI includes in the HTTP response that the (browser) client application displays when the SPNEGO token is received by the interceptor when the challenge-response handshake contains an NT LAN Manager (NTLM) token instead of the expected SPNEGO token. It can specify a Web (`http://`) or a file (`file://`) resource. WebSphere has a default page that is returned.
- `com.ibm.ws.security.spnego.SPN<id>.trimUserName`
This attribute is optional as shown in Figure 13-4. It specifies whether (true) or not (false) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the “@” that precedes the Kerberos realm name. If this attribute is set to true, the suffix of the principal user name is removed. If this attribute is set to false, the suffix of the principal name is retained. The default value used is true.

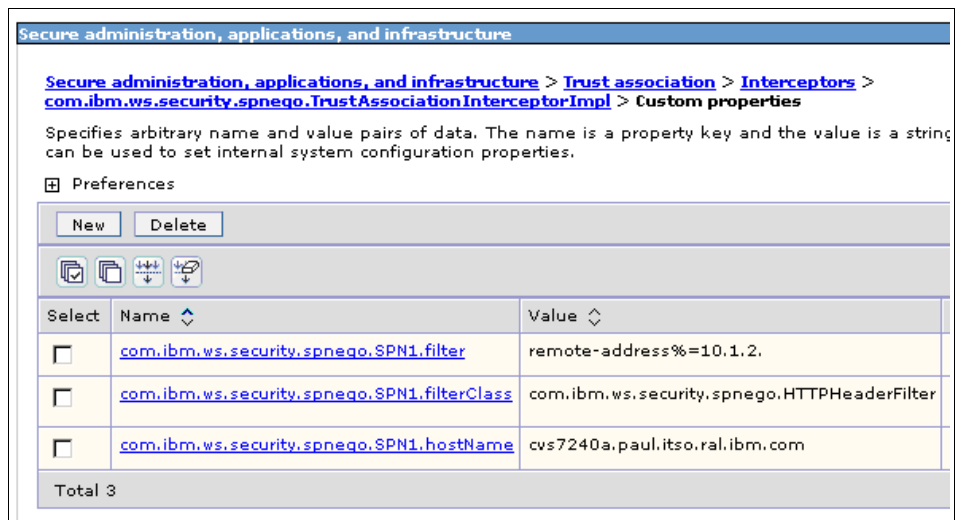


Figure 13-4 The custom properties specified for the example scenario

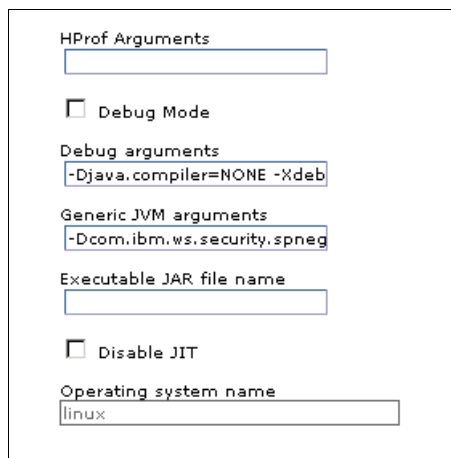
9. After you finish defining your custom properties, click **Save** to store the updated SPNEGO TAI configuration.

Configuring JVM properties and enabling SPNEGO TAI in WebSphere Application Server

To configure the JVM for use with the SPNEGO TAI:

1. Log in to WebSphere Application Server administrative console.
2. Click **Servers** → **Application servers**.
3. Select the appropriate servers and click **Java and process management**. Then click **Process Definition**.
4. Click **Java virtual machine**.
5. In the Generic JVM arguments text box (Figure 13-5), add the following line:

```
-Dcom.ibm.ws.security.spnego.isEnabled=true
```



The screenshot shows the configuration page for a Java virtual machine. It includes several fields and checkboxes:

- HProf Arguments:** An empty text box.
- Debug Mode**
- Debug arguments:** A text box containing `-Djava.compiler=NONE -Xdeb`.
- Generic JVM arguments:** A text box containing `-Dcom.ibm.ws.security.spnego`.
- Executable JAR file name:** An empty text box.
- Disable JIT**
- Operating system name:** A text box containing `linux`.

Figure 13-5 Enabling the JVM for SPNEGO TAI authentication

6. Optional: Complete the following JVM options, which are specified similar to step 5 (space separated):

Tip: These two JVM options enable debugging and can be set to off or all. They can be set as Custom Properties for the JVM using the Administration Console:

```
com.ibm.security.jgss.debug  
com.ibm.security.krb5.Krb5Debug
```

- `com.ibm.ws.security.spnego.propertyReloadFile`

Use this attribute to identify the file that contains configuration properties for the SPNEGO TAI, when it is not convenient to stop and restart the

Application Server. The properties in this file can be reloaded to configure the SPNEGO TAI.

Note: The properties that are defined in the specified file override any properties defined using the administrative console.

Example 13-9 shows the template for this file.

Example 13-9 The template for the SPNEGO property file

```
#####  
# Template properties files for SPNEGO TAI  
#  
# Where possible defaults have been provided.  
#  
#####  
  
#-----  
# Hostname  
#-----  
#com.ibm.ws.spnego.SP1.HostName=wsecurity.austin.ibm.com  
  
#-----  
# (Optional) SpnegoNotSupportedPage  
#-----  
#com.ibm.ws.spnego.SP1.SpnegoNotSupportedPage=  
  
#-----  
# (Optional) NTLMTokenReceivedPage  
#-----  
#com.ibm.ws.spnego.SP1.NTLMTokenReceivedPage=  
  
#-----  
# (Optional) FilterClass  
#-----  
#com.ibm.ws.spnego.SP1.FilterClass=com.ibm.ws.spnego.HTTPHeaderF  
ilter  
  
#-----  
# (Optional) Filter  
#-----  
#com.ibm.ws.spnego.SP1.Filter=  
#-----
```

- `com.ibm.ws.security.spnego.propertyReloadTimeout`

Use this attribute to specify a time interval in seconds that elapses after which the SPNEGO TAI reloads the configuration properties. Also, the SPNEGO TAI reloads the configuration properties if the file that is identified by the `com.ibm.ws.security.spnego.propertyReloadFile` attribute has changed since the last time the configuration attributes were retrieved. This time interval in seconds must be specified as a positive integer.

Important: If the `com.ibm.ws.security.spnego.propertyReloadFile` attribute and the `com.ibm.ws.security.spnego.propertyReloadTimeout` attribute are not set, then the SPNEGO TAI properties are only loaded once from the SPNEGO TAI custom properties defined in the WebSphere Application Server configuration data. This one-time loading occurs when the JVM is initialized.

If the `com.ibm.ws.security.spnego.propertyReloadTimeout` attribute is set, but the `com.ibm.ws.security.spnego.propertyReloadFile` attribute is not, then the SPNEGO TAI is not initialized.

7. Click **Apply** and then click **OK** to save the configuration.

13.2.3 Troubleshooting SPNEGO environments

Use the following tips to help you troubleshoot an SPNEGO environment:

- ▶ Turn on logging. For details, see the following address:
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/tsec_SPNEGO_trouble_shoot.html
- ▶ Make sure that you are trying to access WebSphere Application Server with a user that is logged into the domain.
- ▶ Make sure that you are *not* trying to access the WebSphere Application Server from the domain controller.
- ▶ Make sure that the TAI initialized successfully as shown in Example 13-10.

Example 13-10 Successful TAI initialization

```
[8/8/06 17:56:56:247 EDT] 0000000a TrustAssociat I
com.ibm.ws.security.spnego.TrustAssociationInterceptorImpl
initialize CWSPN0006I: SPNEGO Trust Association Interceptor
initialization is complete. Configuration follows:
    TAI configuration (JVM) properties:
    com.ibm.ws.security.spnego.isEnabled=true
```

```

Server configuration:
Kerberos
ServicePrincipalName=HTTP/cvs7240a.paul.itso.ral.ibm.com@PAUL.ITSO.R
AL.IBM.COM

com.ibm.ws.security.spnego.SPN.filter=remote-address%=10.1.2.

com.ibm.ws.security.spnego.SPN.filterClass=com.ibm.ws.security.spneg
o.HTTPHeaderFilter@4fe24fe2
com.ibm.ws.security.spnego.SPN.NTLMTokenReceivedPage=null
com.ibm.ws.security.spnego.SPN.spnegoNotSupportedPage=null

```

- ▶ Make sure that you have the correct encryption types specified in your Kerberos configuration file.
- ▶ Make sure that your machine can resolve an IP address from the host names specified in DNS.
- ▶ Make sure that you have the same time on the WebSphere Application Server host and the domain controller.
- ▶ For Linux and UNIX, try to start a Kerberos session from the WebSphere Application Server host using `kinit`. This tip allows a user to debug the Kerberos environment itself, before having to think about WebSphere configuration.

Example 13-11 shows sample output where a successful session is started.

Example 13-11 Output from a successful start of a session

```

cvs7240a:/opt/IBM/WebSphere/AppServer/profiles/SPNEGOAppSrv/bin #
kinit cvs7240a@PAUL.ITSO.RAL.IBM.COM
cvs7240a@PAUL.ITSO.RAL.IBM.COM's Password:
kinit: NOTICE: ticket renewable lifetime is 1 week
cvs7240a:/opt/IBM/WebSphere/AppServer/profiles/SPNEGOAppSrv/bin #
klist
Credentials cache: FILE:/tmp/krb5cc_0
Principal: cvs7240a@PAUL.ITSO.RAL.IBM.COM

```

```

Issued          Expires          Principal
Aug  8 19:10:25 Aug  9 05:10:25
krbtgt/PAUL.ITSO.RAL.IBM.COM@PAUL.ITSO.RAL.IBM.COM

```

- ▶ For Windows, use `kerbtray.exe` to see which tickets have been granted to the user logged in.
- ▶ Restart the client machine. Sometimes this can have adverse affects if the client is brought up before the domain controller.

- ▶ Remove any filters that you specified for the TAI. Try to get a connection working before you start filtering requests.
- ▶ Make sure that you have SPNEGO authentication enabled on the client's browser. For instructions, see the following address:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tsec_SPNEGO_config_web.html

In the Firefox configuration steps, you must also set the value for `network.negotiate-auth.trusted-uris` variable.

Tip: For SPNEGO troubleshooting, see the WebSphere Information Center at the following Web address:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/tsec_SPNEGO_trouble_shoot.html

13.3 IBM WebSphere Application Server and WebSEAL integration

This section discusses the different integration scenarios between WebSphere Application Server and WebSEAL.

13.3.1 Integration options

Various options are available to set up SSO between WebSphere Application Server and Access Manager's WebSEAL.

Using the Trust Association Interceptor

You can set up the TAI in two ways:

- ▶ With a trusted user
- ▶ With a trusted connection

Trusted user

In this configuration, the TAI identifies the WebSEAL server using the Basic Authentication header. A trusted user is created in LDAP and the TAI is configured with that userID. Only the password (not the userID) is placed on the basic authentication header by WebSEAL. This represents a "shared secret", which only the TAI and the WebSEAL server know.

During run time, the TAI examines the password and validates it with the user registry to confirm that the password belongs to the trusted user. This procedure enables the TAI to trust that it really is the WebSEAL server asserting the end user's identity, and the TAI can therefore trust it. To set up the WebSEAL junction to use the basic authentication header to identify the WebSEAL server, you can use the *-b supply* option with the junction creation command. WebSEAL builds the Basic Authentication header using the password, which is specified in the `Webseald.conf` file (`basicauth-dummy-passwd` property). This set up is described in detail in consequent sections.

Trusted connection using mutual Secure Sockets Layer

If you are using the old TAI implementation and using this configuration, then the WebSEAL server identifies and authenticates itself to the Web server using its own client-side certificates. In this case, the TAI performs no further validation of the WebSEAL server hosts. This configuration is set in TAI using the `com.ibm.websphere.security.WebSEAL.mutualSSL=true` setting. With these settings the TAI validates the WebSEAL host using the *hostname* property, and does no further validation. It assumes that the connection from WebSEAL to Application Server is completely trusted. This setup requires a Secure Sockets Layer (SSL) junction. You set up an encrypted junction using SSL with client certificates.

Mutual SSL authentication: Mutual SSL authentication is no longer supported by the newer WebSEAL TAI included with WebSphere Application Server V5.1.1 and later. You can still configure mutual SSL in WebSphere Application Server (and doing so is often useful), but the TAI does not acknowledge it. You must use password-based authentication from WebSEAL with the newer TAI. The older TAI (*WebSealTrustAssociationInterceptor*) continues to be included if you want to use it.

Using Lightweight Third Party Authentication

With LTPA, you do not have to configure a TAI for the Application Server. Instead, you can configure an LTPA junction.

Important: An LTPA junction is considered non-strategic. Administrators must try to use TAI junctions instead.

13.3.2 Configuration for the Trust Association Interceptor approach

In this section, we describe the configuration for using TAIs.

Flow of tokens from WebSEAL to WebSphere Application Server

WebSEAL provides authentication and authorization to all requests before passing them to the junctioned Web server. In Figure 13-6, WebSEAL is in the demilitarized zone.

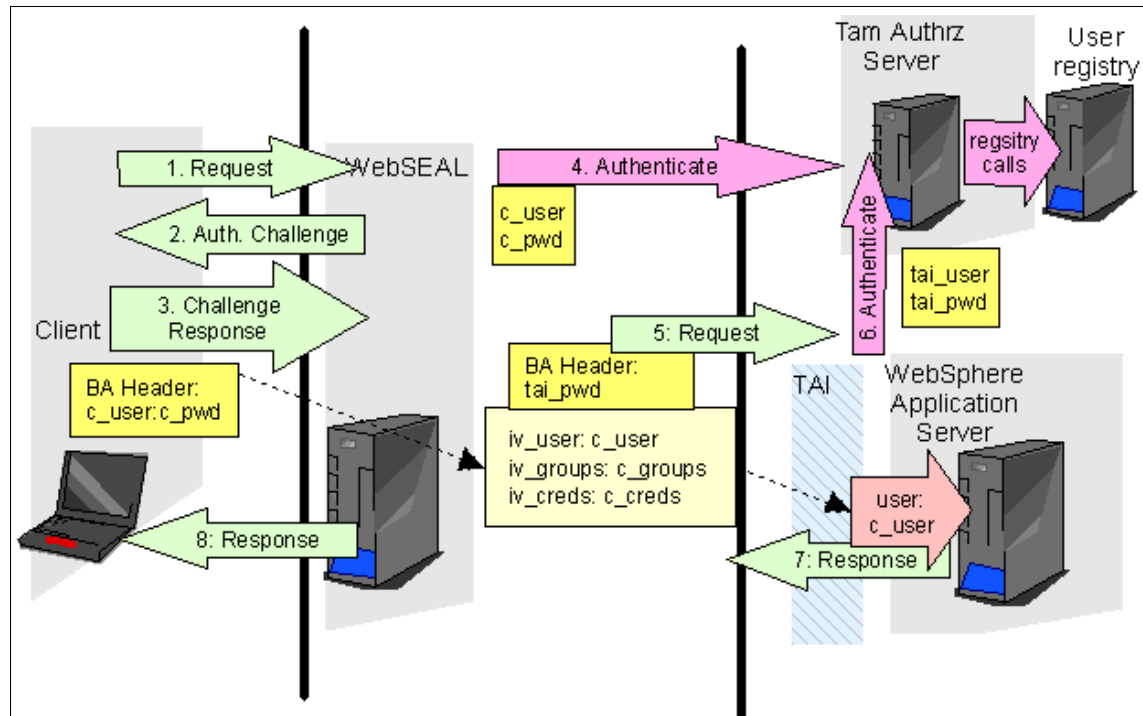


Figure 13-6 Request flow when using the TAI

The flow in Figure 13-6 is as follows:

1. An unauthenticated client issues a request for a secure resource which is intercepted by the reverse proxy (WebSEAL).
2. WebSEAL issues an HTTP authentication challenge to the client. Note that WebSEAL can be configured to use a different authentication mechanism, such as forms authentication or token authentication, but the overall flow of information remains the same.

3. The client responds to the authentication challenge with a new request containing the client's userID (c_user) and password (c_pwd) in the HTTP basic authentication header.
4. WebSEAL authenticates the user against the user registry using the c_user and c_pwd values. WebSEAL also authorizes the request based on access control list (ACL) that is configured for the junction.
5. WebSEAL modifies the BA header so that it includes the TAI password configured in the WebSEAL configuration file. WebSEAL also attaches the client's userID and group membership and credentials into an additional HTTP headers (iv_user, iv_groups, and iv_creds) that are sent along with the request to the Application Server.
6. The request goes to the WebSphere Application Server, where the TAI intercepts the request for further security processing. The TAI performs the authentication for the configured tai_user (using the configured tai_pwd). This authentication ensures that the TAI, together with the WebSphere Application Server, is trusted. The TAI then extracts the credential information from the incoming request from WebSEAL.
7. The user credentials are extracted from the request by the TAI and are used to construct a PDPrincipal object in the Application Server. A credential object that contains user information is constructed from information contained in the PDPrincipal. The Principal and the Credential objects are inserted into a JAAS Subject, which is returned from the call.
8. WebSphere sends the output to WebSEAL.
9. WebSEAL dispatches the output to the client.

Note the following additional comments for the TAI:

- ▶ WebSphere Application Server does not query the registry directly for Trust Association Interceptor processing. The new Interceptor class TAMTrustAssociationInterceptorplus contacts the Tivoli Access Manager Authorization Server which does the check with the user registry. This also indicates that additional configuration is required to ensure that WebSphere Application Server can contact the authorization server.
- ▶ It is possible to negotiate with the client in a multiphase handshake. Keep in mind that the Tivoli Access Manager TAI does not use the multiphase negotiation and it does not have to. There are security protocols that may require negotiation, for example, SPNEGO. TAI that can handle negotiation can be developed using the new, extended interface.
- ▶ With the new TAI, you can add custom attributes to the Subject in the form of Java sets.
- ▶ You can continue to use the old WebSEAL TAI class called *WebsealTrustAssociationInterceptor* in WebSphere Application Server V6.1.

Cookie configuration options

WebSEAL forwards cookies created for the user by the back-end system to the user's browser. The decision as to which cookies are to be created by the back-end system and subsequently sent to the user's browser is something that can be further considered when using WebSEAL and TAI. Because TAI can create a valid JAAS subject that gives the Application Server the authenticated user's credentials, it is possible for it not to use WebSphere Application Server SSO capabilities. Disabling WebSphere Application Server SSO means that no LTPA tokens are going to be created to track the user's session. Some customers consider this beneficial.

Therefore, there are two options for which cookies are sent to the user's browser:

- ▶ Keeping WebSphere Application Server SSO enabled, which creates LTPA tokens for the user
- ▶ Disabling WebSphere Application Server SSO, which *does not* create LTPA tokens for the user

Depending upon your environment and personal opinion on what might be considered more secure, this decision must be made on a case-by-case basis. Figure 13-7 and Figure 13-8 give examples of the cookies that are sent to the user's browser depending on the configuration.

Figure 13-7 shows the cookies that are sent to the user's browser when WebSphere Application Server SSO is enabled.

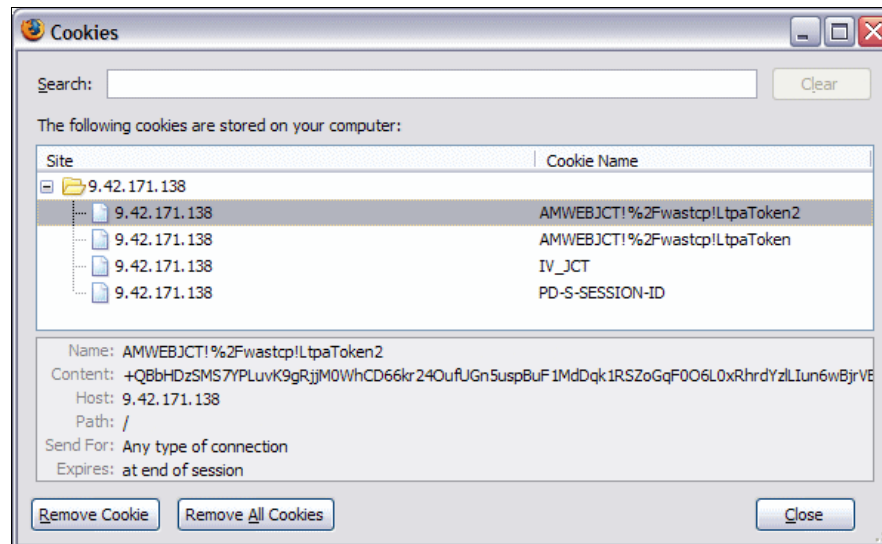


Figure 13-7 Cookies sent to the user's browser when WebSphere Application Server SSO is enabled

Figure 13-8 shows the cookies that are sent to the user's browser when WebSphere Application Server SSO disabled.

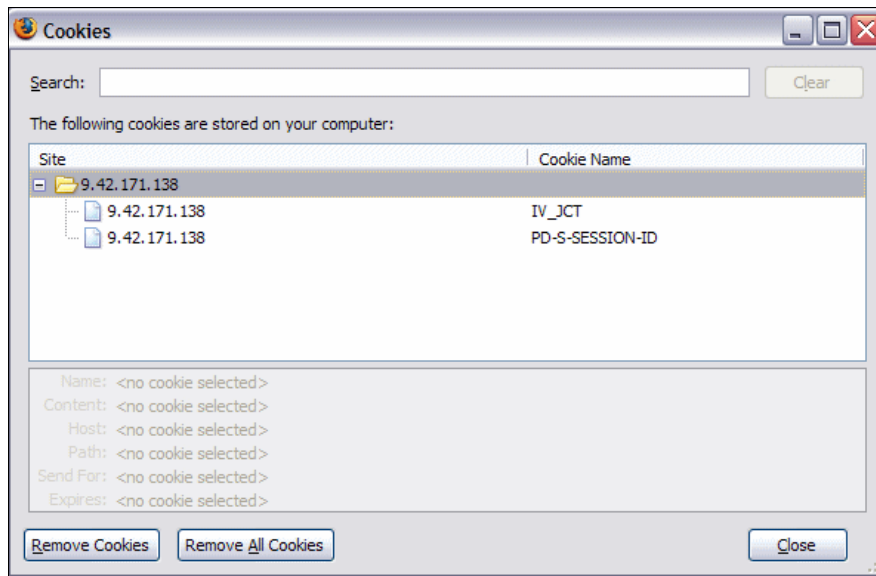


Figure 13-8 Cookies sent to the user's browser when WebSphere Application Server SSO disabled

Attention: Disabling SSO in WebSphere Application Server disables all forms-based authentication. This includes the WebSphere administration console and also application logins. The reason why WebSEAL can still allow access to resources is that TAI creates a JAAS subject without the requirement of a form.

Therefore, if SSO is disabled while login through WebSEAL to applications works, then all administration must be carried out by using **wsadmin**, unless a different, non-publicly accessible, WebSEAL junction is set up to protect the administration console (we do *not* recommend). Although, forcing all administration through **wsadmin** can also be considered beneficial, especially with the use of the new **wsadmin** *fine grained administrative security*.

Configuring SSO between WebSphere Application Server and WebSEAL

Prior to configuring SSO between WebSphere Application Server and WebSEAL, you must have the following prerequisites in place:

- ▶ Ensure that the IBM Tivoli Directory Server V6.1 is installed and configured for both Tivoli Access Manager for e-business and for WebSphere Application Server registry. If you are following the scenario in this book, you can import the `import.ldif` file that is provided as part of the additional material. See Appendix B, “Additional material” on page 543.
- ▶ Ensure that Administrative and Application Security is enabled with LTPA and LDAP.
- ▶ We use a WebSphere Application Server sample application called *Technology Samples*. Ensure that you install this application and can run it through the Web server.
- ▶ You must install and configure Tivoli Access Manager for e-business V6.0 correctly. You must be able to access the WebSEAL form login page. To configure WebSEAL for form-based authentication, make the following changes in the `webseald.conf` file:

```
forms-auth=https
basic-auth=none
```

Figure 13-9 shows the simple environment to test this configuration.

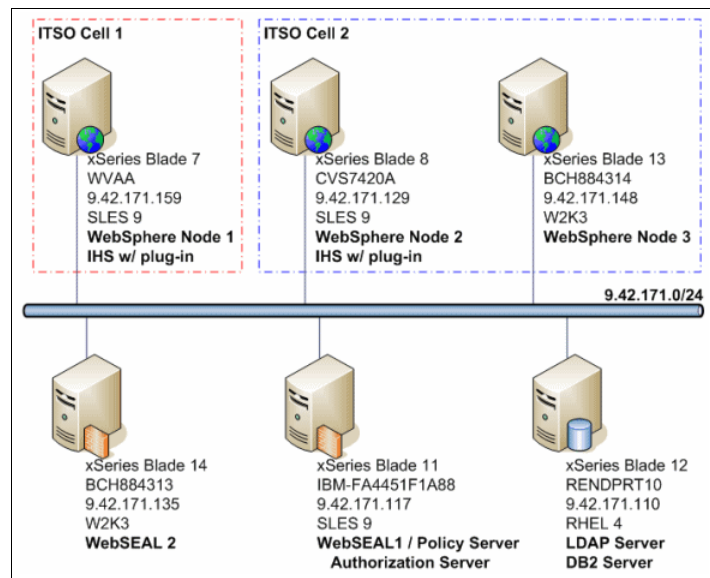


Figure 13-9 Test environment for the configuration

Installing and configuring the base products

To begin, install and configure WebSphere Application Server, IBM Directory Server with DB2, IBM HTTP Server, and Tivoli Access Manager for e-business (including WebSEAL). See the WebSphere Application Server product documentation, and the Tivoli product documentation for installation and configuration. For user registry configuration for WebSphere Application Server, see Chapter 2, “Configuring the user registry” on page 7.

Creating test users for Tivoli Access Manager

To create the test users;

1. Create two user accounts in LDAP by importing the text shown in Example 13-12 as a .ldif file.

Example 13-12 tai-config.ldif

```
dn: uid=taiuser,o=ibm,c=us
objectclass: organizationalPerson
objectclass: ePerson
objectclass: top
objectclass: inetOrgPerson
uid: taiuser
userpassword: taiuser1
sn: taiuser
givenname: taiuser
cn: tai
preferredlanguage: en
```

```
dn: uid=amy,o=ibm,c=us
objectclass: organizationalPerson
objectclass: ePerson
objectclass: top
objectclass: inetOrgPerson
uid: amy
userpassword: test
sn: amy
givenname: amy
cn: amy
preferredlanguage: en
```

```
dn: uid=john,o=ibm,c=us
objectclass: organizationalPerson
objectclass: ePerson
objectclass: top
objectclass: inetOrgPerson
uid: john
```

```
userpassword: test
sn: john
givenname: john
cn: john
preferredlanguage: en

dn: cn=managers,o=ibm,c=us
objectclass: top
objectclass: groupOfUniqueNames
cn: managers
uniquemember: uid=amy,o=ibm,c=us

dn: cn=human_resources,o=ibm,c=us
objectclass: top
objectclass: groupOfUniqueNames
cn: hr
uniquemember: uid=john,o=ibm,c=us
```

2. Use the following command to import the file:

```
ldif2db -i tai-config.ldif
```

This command creates three users in the directory, which are taiuser, amy, and john.

3. Start the pdadmin tool and import the users into Tivoli Access Manager by using the following commands:

```
user import taiuser uid=taiuser,o=ibm,c=us
user modify taiuser account-valid yes
user modify taiuser password-valid yes
```

Similarly, run all three commands by replacing taiuser with amy and john.

4. Import the managers group by using the following command:

```
group import managers cn=managers,o=ibm,c=us
```

Configuring SSL for the Web server

If you are setting up WebSEAL junction to use SSL, configure SSL for the Web server so that the HTTPS traffic uses a self-signed certificate. If you are using TCP, instead of SSL for your WebSEAL junction, skip this configuration step and proceed to the next step. The Web server must have a port defined for SSL (usually 443). You can use the IBM Key Management Utility, iKeyman, to generate a self-signed certificate.

To configure SSL for the Web server:

1. Set up the IBM HTTP server using SSL.
2. Use the keystore name of `ihskeys.kdb`.
3. Extract the self-signed certificate into the `IHSCertificate.arm` file.
4. Restart the HTTP Server.
5. Verify the configuration by accessing a Web page using SSL (HTTPS).

Importing the Web server certificate to WebSEAL to establish trust

Load the certificate that you created in the previous step into the key database of WebSEAL:

1. If your HTTP Server is on a different machine than WebSEAL, copy your certificate (the `IHSCertificate.arm` file from the previous step) from the HTTP Server machine to the WebSEAL machine.
2. Start iKeyman on the WebSEAL machine and open the WebSEAL keystore in the `<Access_Manager_Install_root>\PDWeb\www-<profile_name>\certs\pdsrv.kdb` file. This is the key-ring used by WebSEAL to store acceptable CA certificates for SSL junctions. The password for this keystore is `pdsrv`.
3. Add the IBM HTTP Server certificate named `IHSCertificate.arm` to the WebSEAL keystore.

Ensuring the SSL port of the virtual hosts in WebSphere Application Server is specified

In order for the Web server plug-in to forward the HTTPS traffic to the Application Server, the host alias port for the virtual host must be specified. In WebSphere Application Server V6.1, this is enabled by default, but it is good practice to check. If it is not present, update the virtual host list for WebSphere Application Server to include the correct host name and port numbers and then regenerate the plug-in configuration:

1. Launch the Administrative Console and log in on the WebSphere Application Server machine.
2. Select **Environment** → **Virtual Hosts** → **default_host** → **Host Aliases** → **New**. Add a host alias for the host name and an SSL port. The host name can be a single asterisk (*) or a fully-qualified host name. Usually this is the host name of the Web server. The port number for SSL is usually 443.
3. Click **OK**.
4. Save the configuration for WebSphere, and regenerate the plug-in configuration.

Configuring WebSEAL

Configure a WebSEAL junction from the WebSEAL Server to the Web server. This task is performed on the WebSEAL machine.

1. On the WebSEAL machine, use the **pdadmin** command line to create a WebSEAL junction. Enter the following command:

```
server task default-webSEAL-<hostname> create -t ssl -h  
<webserver_host> -p <SSL_port> -j -b supply -c all -f /ssl1
```

For TCP junctions, use **tcp** instead of **ssl**.

In our scenario, we entered the command as follows:

```
pdadmin sec_master> server task forms-webseald-bch884313 create -t  
ssl -h wvaa -p 443 -j -b supply -c all -f /wastcp
```

2. Edit the `webseald.conf` file to configure the dummy password that is passed in the HTTP header and for forms authentication. Open the file in `<Access_Manager_install_root>/PDWeb/etc/webseald-default.conf`.

3. In the `[junction]` stanza, change the `basic-auth-dummy-password` to the user password of the `taiuser` as follows:

```
basicauth-dummy-passwd = taiuser1
```

4. In the `[forms]` stanza, enable WebSEAL authentication by using forms. If you want to use only SSL junction, then set the `forms-auth` to `https`:

```
forms-auth = https
```

5. Because you are using form-based authentication and not basic authentication, change the `ba-auth` from `https` to `none`:

```
ba-auth = none
```

6. Restart the WebSEAL server, policy server, and the authorization server.

Configuring the Access Manager Java Runtime

In order for Tivoli Access Manager Trust Association Interceptor to run correctly, first configure the Access Manager Java Runtime by using the `PDJrteCfg` utility:

```
java com.tivoli.pd.jcfg.PDJrteCfg -action {config | unconfig}  
-cfgfiles_path configuration_file_path -host policy_server_host [-was]
```

Here, the `configuration_file_path` is the path to the JRE that you want to configure or unconfigure and `policy_server_host` is the host name of the policy server. The `-was` flag indicates that this is a WebSphere Application Server installation. Run the WebSphere **setupCmdLine** command before running this utility.

Example 13-13 shows the result from our scenario.

Example 13-13 Result from our scenario

```
cd <WAS_HOME>/bin
. ./setupCmdLine.sh
java -cp ${CLASSPATH} -Dpd.home={WAS_HOME}/java/jre/PolicyDirector
com.tivoli.pd.jcfg.PDJrteCfg -action config -cfgfiles_path
${WAS_HOME}/java/jre -host ibm-fa4451f1a88.itso.ral.ibm.com -was
```

Running the SvrSslCfg utility

Run the **SvrSslCfg** command to configure an SSL connection between Tivoli Access Manager and WebSphere Application Server. This command creates a <WebSphere_root>/java/jre.PdPerm.properties configuration file and a Java key store file, which securely stores a client certificate. These two files enable WebSphere Application Server to contact the Tivoli Access Manager server.

Note: The **SvrSslCfg** command must be run for each WebSphere Application Server machine, Java environment. For example, in a Network Deployment setup, if the deployment manager and a node are installed on the same machine, you must run the **SvrSslCfg** twice: once for the deployment manager developer kit and once to configure the node developer kit.

To run the SvrSslCfg utility:

1. Run the **setupCmdLine.bat** (or **setupCmdLine.sh**) script, which is in <WebSphere_root>\bin to set up the environment.
2. Make sure the WAS_HOME environment variable reflects the WebSphere Application Server installation root.
3. Run the **SvrSslCfg** utility as one continuous command line, or enter keywords on several lines by using a trailing continuation character (\) as shown in Example 13-14.

Example 13-14 Entering keywords using a trailing continuation character

```
CLASSPATH=${WAS_HOME}/java/jre/lib/ext/PD.jar:${WAS_CLASSPATH}
java \
-cp ${CLASSPATH} \
com.tivoli.pd.jcfg.SvrSslCfg \
-action config \
-admin_id sec_master \
-admin_pwd password \
-appsvr_id wasuser \
-policysvr tam_policy_server_host:7135:1 \
```

```
-authzsvr tam_authorization_server_host:7136:1 \  
-mode remote \  
-cfg_file configuration_file \  
-key_file key_file \  
-cfg_action create
```

The explanation of the different switches follows:

action	Action to be taken, it can be config or unconfig.
admin_id	Administrator ID for Tivoli Access Manager, use <i>sec_master</i> .
admin_pwd	Password for the Tivoli Access Manager administrator.
appsvr_id	The name that is specified here is combined with the host name to create unique names for Tivoli Access Manager objects created for your application. The following names are reserved for Tivoli Access Manager applications: <ul style="list-style-type: none">– ivaclid– secmgrd– ivnet– ivweb This ID is created in the registry and is used by WebSphere Application Server to communicate with Tivoli Access Manager.
appsvr_pwd	The password for the Application Server ID (appsvr_id).
authzsvr	Access to the authorization server in the following format: authorization_server_name:port_number:rank
polycysvr	Access to the policy server in the following format: policy_server_host_name:port_number:rank
cfg_action	<i>Create</i> specifies to create the configuration and key store files during server configuration. Configuration fails if either of these files already exists. <i>Replace</i> specifies to replace the configuration and key store files during server configuration. Configuration deletes any existing files and replaces them with new ones.
cfg_file	Specifies the fully-qualified file name.
key_file	Specifies the directory that contains the key files for the server. Make sure that server user (for example, ivmgr) or all users have permission to access the .kdb file.

host	The host name for the Application Server.
mode	Specifies the mode in which the application operates. This value must be either <i>local</i> or <i>remote</i> .

Example 13-15 shows the scenario in this book.

Example 13-15 Example for the scenario in this book

```
java -cp ${CLASSPATH} -Dpd.cfg.home=${WAS_HOME}/java/jre
com.tivoli.pd.jcfg.SvrSslCfg -action config -admin_id sec_master
-admin_pwd its0ral -appsvr_id wvaa -appsvr_pwd its0ral -port 7135 -mode
remote -host wvaa -policysvr ibm-fa4451f1a88.itso.ral.ibm.com:7135:1
-authzsvr 9.42.171.117:7136:1 -cfg_file
/opt/IBM/WebSphere/AppServer/java/jre/PdPerm.properties -domain Default
-key_file /opt/IBM/WebSphere/AppServer/java/jre/lib/security/PdPerm.sh
-cfg_action create
```

Enabling and configuring Tivoli Access Manager Trust Association Interceptor++

The Tivoli Access Manager TAI++ module must be told where it can find information and what it must do with the information that comes in. This is done by setting custom properties for the TAI module.

Note: If you plan to use the old TAI, follow the procedures in the WebSphere Application Server V6.1 Information Center at the following address:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tsec_sso_ws_step4_sso_using_TAI_for_WAS.html

To enable and configure Tivoli Access Manager Trust Association Interceptor++:

1. From the administrative console for WebSphere Application Server, click **Security** → **Secure administration, applications, and infrastructure**.
2. Under Web security, click **Trust association**.
3. Click **Enable Trust Association** and then click **Apply**.
4. Click **Interceptors**.
5. Select **com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus** to edit the WebSEAL interceptor.
6. Click **Custom Properties**.

- Click **New** to enter the property name and value pairs. Verify that the parameters shown in Table 13-3 are set.

Table 13-3 The parameters that must be entered for Tivoli Access Manager TAI++ configuration

Parameter	Description
com.ibm.websphere.security.webseal.checkViaHeader	You can configure TAI so that you can ignore the via header route when you are validating trust for a request. Set this property to false if none of the hosts in the via header require to be trusted. When set to false you do not require to set the trusted host names and host ports properties. The only mandatory property to check when via header is false is com.ibm.websphere.security.webseal.loginId. The default value of the check via header property is false. When using Tivoli Access Manager plug-in for Web servers, set this property to false. ViaHeader: The via header is part of the standard HTTP header that records the server names of the request that passed through.
com.ibm.websphere.security.webseal.loginId	The WebSEAL trusted user as created earlier. The format of the user name is the short name representation. This property is mandatory. If it is not set in WebSphere Application Server, the TAI initialization fails.
com.ibm.websphere.security.webseal.id	A comma-separated list of headers that exists in the request. If all of the configured headers do not exist in the request, trust cannot be established. The default value for the ID property is iv-creds. Any other values set in WebSphere Application Server are added to the list along with iv-creds, separated by commas.
com.ibm.websphere.security.webseal.hostnames	Do not set this property if using Tivoli Access Manager plug-in for Web Servers. The property specifies the host names (case sensitive) that are trusted and expected in the request header. Requests arriving from un-listed hosts might not be trusted. If the checkViaHeader property is not set or is set to false then the trusted host names property has no influence. If the checkViaHeader property is set to true, and the trusted host names property is not set, TAI initialization fails.
com.ibm.websphere.security.webseal.ports	Do not set this property if using Tivoli Access Manager plug-in for Web servers. This property is a comma-separated list of trusted host ports. Requests that arrive from unlisted ports might not be trusted. If the checkViaHeader property is not set, or is set to false this property has no influence. If the checkViaHeader property is set to true, and the trusted host ports property is not set in WebSphere Application Server, the TAI initialization fails.

Parameter	Description
<p>com.ibm.websphere.security.webseal.viaDepth</p>	<p>A positive integer that specifies the number of source hosts in the via header to check for trust. By default, every host in the via header is checked, and if any host is not trusted, trust cannot be established. The via depth property is used when only some of the hosts in the via header must be trusted. The setting indicates the number of hosts that are required to be trusted.</p> <p>As an example, consider the following header: Via: HTTP/1.1 webseal1:7002, 1.1 webseal2:7001</p> <p>If the viaDepth property is not set, or set to 2, or set to 0, and a request with the previous via header is received, then both webseal1:7002 and webseal2:7001 must be trusted. The following configuration applies: com.ibm.websphere.security.webseal.hostnames = webseal1,webseal2 com.ibm.websphere.security.webseal.ports = 7002,7001</p> <p>If the via depth property is set to 1, and the previous request is received, then only the last host in the via header has to be trusted. The following configuration applies: com.ibm.websphere.security.webseal.hostnames = webseal2 com.ibm.websphere.security.webseal.ports =7001</p> <p>The viaDepth property is set to 0 by default, which means all of the hosts in the via header are checked for trust.</p>
<p>com.ibm.websphere.security.webseal.ssoPwdExpiry</p>	<p>After trust is established for a request, the SSO user password is cached, eliminating the requirement to have the TAI re-authenticate the SSO user with Tivoli Access Manager for every request. You can modify the cache timeout period by setting the SSO password expiry property to the required time in seconds. If the password expiry property is set to 0, the cached password never expires. The default value for the password expiry property is 600.</p>
<p>com.ibm.websphere.security.webseal.ignoreProxy</p>	<p>This property can be used to tell the TAI to ignore proxies as trusted hosts. If set to true, the comments field of the hosts entry in the via header is checked to determine if a host is a proxy. Remember that not all proxies insert comments in the via header indicating that they are proxies. The default value of the ignoreProxy property is false. If the checkViaHeader property is set to false then the ignoreProxy property has no influence in establishing trust.</p>

Parameter	Description
com.ibm.websphere.security.webseal.configURL	For the TAI to establish trust for a request, it requires that the <code>SvrSslCfg</code> run for the Java virtual machine on the Application Server and result in the creation of a properties file. If this properties file is not at the default URL, which is <code>file:///java.home/PdPerm.properties</code> , then the correct URL of the properties file must be set in the configuration URL property. If this property is not set, and the <code>SvrSslCfg</code> -generated properties file is not in the default location, then the TAI initialization fails. The default value for the config URL property is: <code>file:///\${WAS_INSTALL_ROOT}/java/jre/PdPerm.properties.</code>

8. Click **OK**.
9. Save the configuration and log out.
10. Restart WebSphere Application Server.

Figure 13-10 shows the options specified for the example scenario in this book.

Secure administration, applications, and infrastructure

[Secure administration, applications, and infrastructure](#) > [Trust association](#) > [Interceptors](#) > [com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus](#) > **Custom properties**

Specifies arbitrary name and value pairs of data. The name is a property key and the value is a string value that can be used to set internal system configuration properties.

Preferences

New Delete

Select	Name	Value	Description
<input type="checkbox"/>	com.ibm.websphere.security.webseal.checkViaHeader	true	
<input type="checkbox"/>	com.ibm.websphere.security.webseal.configURL	file:/// \${WAS_INSTALL_ROOT}/java/jre/PdPerm.properties	
<input type="checkbox"/>	com.ibm.websphere.security.webseal.hostnames	bch884313	
<input type="checkbox"/>	com.ibm.websphere.security.webseal.id	iv-creds	
<input type="checkbox"/>	com.ibm.websphere.security.webseal.ignoreProxy	false	
<input type="checkbox"/>	com.ibm.websphere.security.webseal.loginId	taiuser	
<input type="checkbox"/>	com.ibm.websphere.security.webseal.ports	443,444,446	
<input type="checkbox"/>	com.ibm.websphere.security.webseal.ssoPwdExpiry	300	
<input type="checkbox"/>	com.ibm.websphere.security.webseal.viaDepth	0	

Total 9

Figure 13-10 The custom properties specified for the TAI++ module in the example scenario

Testing the configuration

After restarting the Application Server check the server's `SystemOut.log` file. There must be an entry in the log from each of the TAI modules initializing. The successful initialization of the module can be seen in the log output in Example 13-16.

Example 13-16 SystemOut.log showing a successful TAI module initialization

```
[8/3/06 6:20:06:134 PDT] 0000000a TrustAssociat A SECJ0122I: Trust
Association Init Interceptor signature: WebSeal Interceptor Version 1.1
[8/3/06 6:20:06:222 PDT] 0000000a TrustAssociat A SECJ0121I: Trust
Association Init class
com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus loaded
successfully
[8/3/06 6:20:08:622 PDT] 0000000a TAMTrustAssoc I
com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlusinitialize(Pr
operties) The Trust Association Interceptor component of embedded
Tivoli Access Manager has been initialized.
[8/3/06 6:20:08:623 PDT] 0000000a TrustAssociat A SECJ0122I: Trust
Association Init Interceptor signature: $Id: @(#)64 1.6
src/pdwas/com/ibm/ws5/security/web/TAMTrustAssociationInterceptorPlus.j
ava, amemb.jacc.was, amemb600, 051118a 05/10/05 09:50:42 @(#) $
```

Next, access the WebSEAL junction that points to the Application Server. It is successful if you can access protected WebSphere applications. Access the snoop application. Notice the User Principal value, which must be the user ID of the user logged into Access Manager.

If there are any errors at this stage, try the following actions:

- ▶ Check the WebSEAL and WebSphere application logs and look for any errors. See if TAI initialization was successful.
- ▶ Make sure that `PDJrteCfg` and `SvrSslCfg` completed successfully.
- ▶ Make sure that you can log in to the WebSEAL instance by accessing the WebSEAL root.
- ▶ Make sure that the page on the Web server routing to WebSphere Application Server is available.
- ▶ Make sure that, if you are using an SSL junction, the certificates are trusted.
- ▶ Try turning on a trace for TAI in WebSphere by using the following setting:
`com.ibm.ws.security.*=all=enabled`
- ▶ Make sure that configuration values specified in the custom properties are correct, especially the trusted host names.

More information: For a full list of troubleshooting tips, see the following address:

<http://www-128.ibm.com/developerworks/tivoli/library/t-tamtai/>

13.3.3 Configuration for the LTPA approach

The LTPA approach type of trust association is considered far inferior to the TAI approach. That said, sometimes it might be unavoidable to use this approach. If you decide to use this approach, familiarize yourself with all the LTPA key management facilities available in WebSphere Application Server V6.1, which you can see at the following address:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tsec_ltpa.html

LTPA is an IBM proprietary technology used in IBM products like WebSphere Application Server, Tivoli Access Manager, and Lotus Domino. The LTPA token is an encrypted string that contains a user ID, expiration time, and a digital signature.

Look at a scenario in which WebSphere issues the LTPA token. In this case, WebSphere Application Server authenticates the user and issues an LTPA token. This LTPA token can be passed to another WebSphere Application Server instance which can read this LTPA token and determine the authenticated user ID. The basis for reading and trusting the LTPA token is that the two WebSphere Application Server instances share the same LTPA keys for token generation and they must also share the same user registry.

LTPA key management: LTPA key management has changed significantly in WebSphere Application Server V6.1. The keys are stored in a JCEKS key store and the key store, but not the keys themselves, which can be viewed in the WebSphere administration console.

The password used to encrypt exported ltpa keys has been separated from the storage of the keys. This means that, in previous versions of WebSphere Application Server, whenever a new password was set in the LTPA configuration panel, new keys were generated. This is no longer the case. See the WebSphere Application Server Information Center for more information:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tsec_ltpa.html

However, in most real-world scenarios, the authentication is done by a third-party security server such as Tivoli Access Manager, which is also capable of issuing LTPA tokens. Both Tivoli Access Manager and WebSphere Application Server are configured with the same LTPA encryption key. WebSphere Application Server receives the LTPA Token, decrypts it, and determines the authenticated user ID. It does not challenge the user again thus providing SSO from Tivoli Access Manager to WebSphere Application Server. Figure 13-11 illustrates this scenario.

Attention: WebSEAL LTPA support for trust association to WebSphere Application Server is *not strategic*. The preferred option is to use the Tivoli Access Manager Trust Association Interceptor.

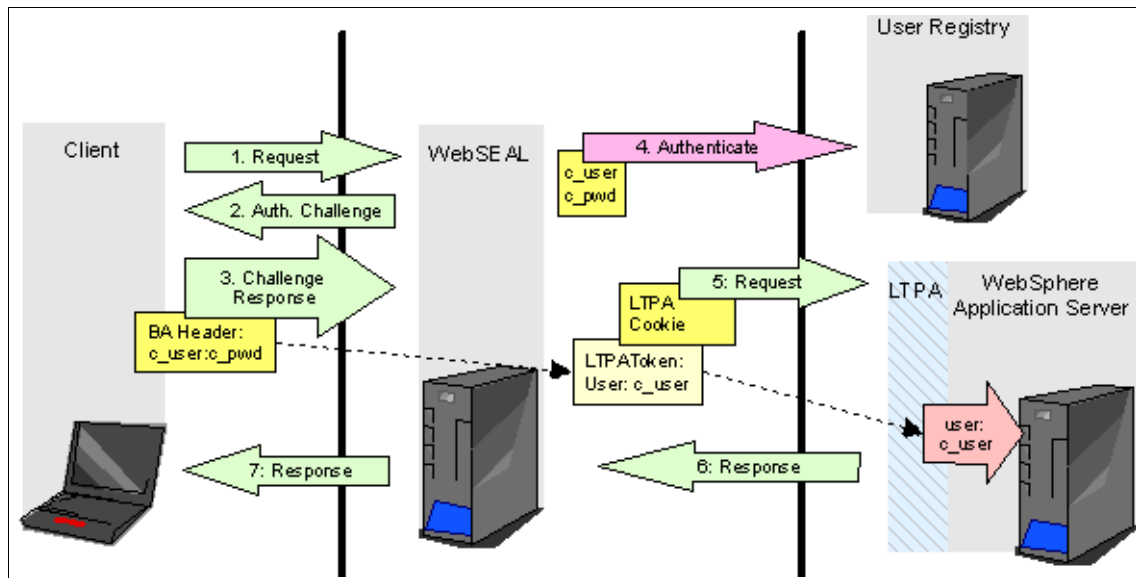


Figure 13-11 Request flow when using LTPA

The following steps describe the flow when using LTPA:

1. A client requests a secured resource.
2. WebSEAL is the Web proxy that intercepts the request and challenges the client.
3. The client supplies the credentials in a new request.
4. WebSEAL authenticates the user against the user registry and constructs an LTPA token and attaches it to an LTPA cookie.

5. A request is passed to the back-end junctioned Web server with the WebSphere Application Server plug-in.
6. WebSphere Application Server receives the request. WebSphere Application Server looks for the LTPA token and finds it in the cookie. WebSphere Application Server decrypts the LTPA token and verifies that the signature is correct. From here on, WebSphere Application Server trusts the identity of the user as specified in the LTPA token.
7. WebSphere Application Server sends output to WebSEAL, and WebSEAL sends the output to the client.

Cookie in LTPA cache of WebSEAL: WebSEAL does not send the LTPA cookie to the client, but rather the cookie is stored in WebSEAL's LTPA cache. WebSEAL uses a different cookie to identify the session with the client, then the actual LTPA token is mapped to the session. This approach provides higher security because the LTPA token can be captured on the network.

Configuring LTPA

If you have already set up security on your WebSphere Application Server, you can skip to “Exporting the LTPA Keys from WebSphere” on page 399. To set up LTPA in WebSphere:

1. Launch the Administrative Console for WebSphere Application Server and login.
2. Click **Security** → **Secure administration, applications, and infrastructure** → **Authentication mechanisms and expiration**.
3. Select the appropriate group from the Key set group field that contains your public, private, and shared LTPA keys. These keys are used to encrypt and decrypt data that is sent between servers. You can access these key set group configurations using the Key set group link. In the Key set group configuration, you can indicate whether to automatically generate new keys and when to generate them.
4. Enter a positive integer value in the authentication cache timeout field. This timeout value refers to how long an LTPA token is valid in minutes. The token contains this expiration time so that any server that receives the token can verify that the token is valid before proceeding further. This value must be smaller than your WebSEAL session timeout. See 12.1.8, “Challenges with reverse proxy authenticators” on page 309, for further information.
5. Enter a positive integer in the Timeout value for forwarded credentials between servers field. This value refers to how long the server credentials from another server are valid before they expire. The default value is 120 minutes. The value in the Timeout value for forwarded credentials between

servers field must be greater than the value in the authentication cache timeout field.

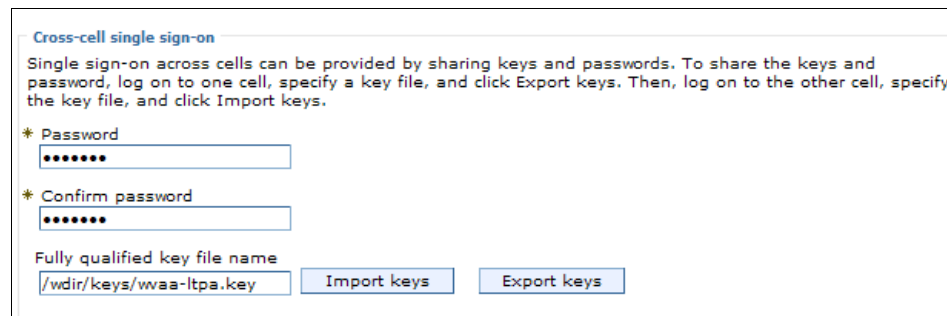
6. Click **Apply** or **OK**. The LTPA configuration is now set. Do not generate the LTPA keys in this step because they are automatically generated later. Proceed with the rest of the steps that are required to enable security, and start with SSO, if it is required.
7. Complete the information in the Security → Secure administration, applications, and infrastructure panel and click **OK**. Make sure that both Administrative Security and Application Security are enabled. The LTPA keys are generated automatically the first time. Do not generate the keys manually.
8. Click **Web security** → **Single Sign-On** and select the **Enabled** box. Also enter the SSO domain name that corresponds to your environment.
9. Click **OK** and then restart the server.

Exporting the LTPA Keys from WebSphere

Export password encrypted LTPA keys into a file that you can use with WebSEAL:

1. Launch the Administrative Console for WebSphere Application Server and log in.
2. Select **Security** → **Secure administration, applications, and infrastructure** → **Authentication mechanisms and expiration**.
3. Toward the bottom of the page, in the Cross-cell single sign-on section, set or change the password if required. In the Key File name field, enter the full path of a file on the WebSphere Application Server machine where the key file must be placed.

Click **Export Keys** as shown in Figure 13-12 to create the exported key file.



Cross-cell single sign-on

Single sign-on across cells can be provided by sharing keys and passwords. To share the keys and password, log on to one cell, specify a key file, and click Export keys. Then, log on to the other cell, specify the key file, and click Import keys.

* Password
.....

* Confirm password
.....

Fully qualified key file name
/wdir/keys/waa-ltpa.key

Import keys Export keys

Figure 13-12 Exporting LTPA keys

The key file must look similar to Example 13-17.

Example 13-17 Password encrypted LTPA key file

```
#IBM WebSphere Application Server key file
#Thu Aug 03 12:23:14 PDT 2006
com.ibm.websphere.CreationDate=Thu Aug 03 12\:23\:14 PDT 2006
com.ibm.websphere.ltpa.version=1.0
com.ibm.websphere.ltpa.3DESKey=c6UsDMRoQ/CaarcTAPFgANfDqP0gKYaLWFNxYMEpC10\=
com.ibm.websphere.CreationHost=wvaa
com.ibm.websphere.ltpa.PrivateKey=44gdF8RbgmyNvE/7kETT+BqlxLdvxLVyZpIvY9AYia/aIpJav7j
M3yTMH3C+kt08G8lI080tPCZaby/G+HdBbc5cbFcpwGyCltiy+NQt1KSwMnpXTn/Lok0KqSN01crHwzW/Nio
gL1mE10ux4EqAAwrRG9qtXu4guBm8UWPzBAGYwih4wcY01URg2Z09rhDbrYvLC98LVJJ9wDV3uLa/zEcChpS+
fk1dHvTDi+pxvaQIQCNuSR5FE96EusysfdzF0pV3iKqdB3JQgsKh75D3y4wpD1IFLnXPUzhzZY0zckQBMor4x
fp6q2yZH2y/R0axt8wY6yZo5mT+I21I+WWhiKr6CeuvG1iCxXqLvmafBg\=
com.ibm.websphere.ltpa.Realm=9.42.171.110\:389
com.ibm.websphere.ltpa.PublicKey=AMrUkQ5ZqJX0r8zfTzRsms2gBgv3t3f/V59ntHA55fGhHt8vpQSH
yLLldzNn0UgY+b/Q++ZxctbUSC4KoM69kzY10pysq1EIAKgj/Ij1/KjvAN25j01T4HNFQ2Zr8wF+2grHI1R41
4XZBQkEiykL11kJiddxEh1GSZfZS0jbBA5zAQAB
```

Configuring WebSEAL to use an LTPA junction

In this stage, set up WebSEAL to use LTPA tokens to establish trust with the application server:

1. Copy the LTPA key file to the WebSEAL server. Note that this file must be kept secure. Otherwise, the LTPA trust relationship might be compromised.
2. To set up an SSL junction, enable the Web server to use SSL and exchange certificates between the Web server and WebSEAL.
3. Create the junction on the WebSEAL server. For junction creation, specify the following three options:

A Enables LTPA cookies.

F <full_path_to_ltpa_keys_file>

Specifies the full path name and location (on the webseal host machine) of ltpa key file exported from the WebSphere Application Server machine. This shared ltpakeys.txt file was originally created on the WebSphere Application Server host and copied to the WebSEAL machine.

Z <keyfile_password> Specifies the password required to open the keyfile for LTPA, which is defined in the WebSphere Application Server Administrative Console.

By using **pdadmin** on the WebSEAL server, execute the following commands:

```
pdadmin sec_master> server task default-webseald-ibm-fa4431f1a88  
create -t ssl -A -F "/wdir/keys/wvaa-ltpa.key" -Z "passw0rd" -h  
wvaa -p 443 /ltpa
```

4. Test the junction by accessing the snoop servlet. In our example, we enter the following URL:

```
https://ibm-fa4431f1a88/ltpa/snoop
```

Configuring LTPA cache for WebSEAL

Because LTPA creation, encryption, and decryption introduce processing overhead, with LTPA cache processing, you can improve the performance of LTPA junctions in a high-load environment. The LTPA cache is enabled by default. To configure the ltpa cache settings, open the `webseald.conf` file and locate the `[ltpa-cache]` stanza. The following settings are available:

- ▶ `ltpa-cache-enabled`, where the default value is “yes”. It enables and disables the LTPA cache.
- ▶ `ltpa-cache-size`, where the default value is “1096”. It defines the maximum number of entries allowed in the cache hashtable. Higher value sets more memory and results in faster information access.
- ▶ `ltpa-cache-entry-lifetime`, where the default is 3600 seconds. It is the lifetime of a cache entry.
- ▶ `ltpa-cache-entry-idle-timeout`, where the default value is 600 seconds. It defines the maximum time an inactive cache entry can remain in cache.

For more information about tuning these values, see the WebSEAL administration guide on the Web at the following address:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame.doc/am60_webseal_admin.htm

13.3.4 Security considerations

Tivoli Access Manager Trust Association is a powerful mechanism that has to be tightly secured. Because the environment is only as secure as the weakest point, use care for every installation.

The first possible security vulnerability is that an attacker can obtain the trusted user name and password. If this happens, the attacker can vouch for false users and be trusted.

To help prevent this:

- ▶ Ensure that all traffic between WebSEAL, IBM HTTP Server, and WebSphere Application Server is secured at the transport layer by using SSL. This helps prevent eavesdropping.
- ▶ Use transport level security so that only requests from trusted WebSEAL hosts are allowed to reach the IBM HTTP Server. This helps to prevent spoofing of WebSEAL requests by using a stolen TAI trusted password.

Another security consideration is the synchronization of WebSEAL and WebSphere Application Server sessions. See 12.1.8, “Challenges with reverse proxy authenticators” on page 309, for further information.



Externalizing authorization with JACC

The Java Authorization Container Contract (JACC) is a specification that was introduced in Java 2 Platform, Enterprise Edition (J2EE) 1.4 through the Java Specification Request (JSR) 115 process. This specification defines a contract between J2EE containers and authorization providers. This enables any third-party authorization providers to plug into any J2EE 1.4 Application Servers such as WebSphere to make authorization decisions when a J2EE resource is being accessed. The access decisions is made through the standard `java.security.Policy` object.

You can find more information about JACC under JSR 115 at the following address:

<http://www.jcp.org/en/jsr/detail?id=115>

The specification defines new `java.security` as the permission classes to satisfy the J2EE authorization model. It also defines the binding of container access decisions to operations on instances of these permission classes. In addition, it defines the semantics of policy providers that employ the new permission classes to address the authorization requirements of J2EE, including the following information:

- ▶ The definition of roles as named collections of permissions
- ▶ The granting to principals of permissions corresponding to roles

- ▶ The determination of whether a principal has been granted the permissions of a role (for example, isCallerInRole)
- ▶ The definition of an identifier to role mappings that bind application-embedded identifiers to application scoped role names

The specification defines the installation and configuration of authorization providers for use by containers. The specification defines the interfaces that a provider must make available to allow container deployment tools to create and manage permission collections corresponding to roles.

The JACC specification defines three primary components as shown in Figure 14-1:

- ▶ Deployment tools contract
- ▶ Container contract
- ▶ Provider contract

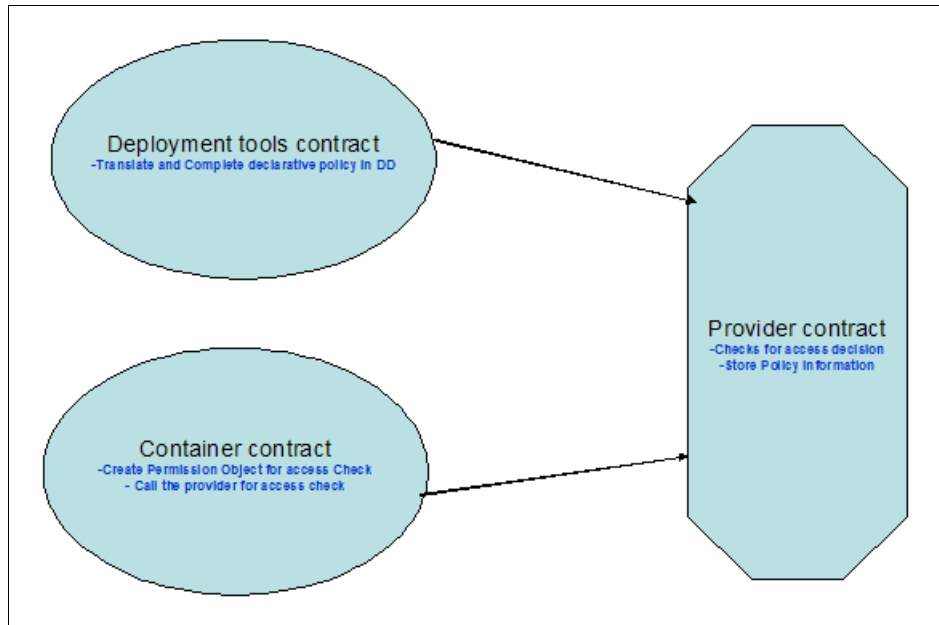


Figure 14-1 Security contracts in JACC

14.1 Deployment tools contract

J2EE deployment tools must translate and complete the declarative policy statements appearing in deployment descriptors into a form suitable for securing applications on the platform. The resulting policy statements may differ in form from the policy statements appearing in the deployment descriptors. The specification requires that the policy information in the deployment descriptor be propagated to the container during the application install time. The policy information contains the security-related information in the deployment descriptor.

Specifically, the security-constraint information in the `web.xml` and the method-permission information in the `ejb-jar.xml` along with `security-role-ref` information in both these files must be propagated to the provider in the format specified by the contract. The format is different for Web and Enterprise JavaBeans (EJB) modules and is governed by the rules specified in the contract. The deployment tools contract defines the following key components:

- ▶ Policy contexts and policy context identifiers

The JACC specification states that deployment tools of contract must define the separate authorization policy contexts corresponding to each deployed instance of a J2EE module. Deployment tools of contract must provide the per module scoping of policy context that is necessary to provide for the independent administration of policy contexts corresponding to individual application modules (perhaps deployed multiple times) within a common policy provider.

- ▶ Servlet policy context identifiers

The JACC specification states that deployment tools of contract must define servlet policy context identifiers sufficient to differentiate all instances of a Web application deployed on the logical host or on any other logical host that may share the same policy statement repository. One way to satisfy this requirement is to compose policy context identifiers by concatenating the host name with the context path (as defined in the Servlet specification) identifying the Web application at the host.

- ▶ Translating servlet deployment descriptors

The JACC specification states that deployment tools of contract must translate the `security-constraint` and `security-role-ref` elements in the deployment descriptor into permissions and add them to the `PolicyConfiguration` object.

- ▶ EJB policy context identifiers

The JACC specification states that the EJB policy context identifiers are sufficient to differentiate all instances of the deployed EJB Java archive (JAR) files on every Application Server.
- ▶ Translating EJB deployment descriptors

If the method-permission element contains the unchecked element, then the deployment tools must call the `addToUncheckedPolicy` method to add the permissions resulting from the translation to the `PolicyConfiguration` object. Alternatively, if the method-permission element contains one or more role-name elements, then the deployment tools must call the `addToRole` method to add the permissions resulting from the translation to the corresponding roles of the `PolicyConfiguration` object.
- ▶ Deploying an application or module

The Application Server's deployment tools must translate the declarative authorization policy appearing in the application or module deployment descriptors into policy statements within the policy providers used by the containers to which the components of the application or module are being deployed.
- ▶ Undeploying an application or module

To ensure that there is not a period during undeployment when the removal of policy statements on application components renders previously protected components unprotected, the Application Server must stop dispatching requests for the application's components before undeploying an application or module.
- ▶ Deploying to an existing policy configuration

To associate an application or module with an existing set of linked policy contexts, the identifiers of the existing policy contexts must be applied by the relevant containers in fulfilling their obligations as defined in the Policy Decision and Enforcement Subcontract. The policy contexts must be verified for existence, by calling the `inService` method of the `PolicyConfigurationFactory` of the relevant containers' Policy providers. The deployment tools must call `Policy.refresh` on the Policy provider of each of the relevant containers, and the containers must not perform predispatch decisions or dispatch requests for the deployed resources until these calls have completed.
- ▶ Redeploying a module

Containers are not required to implement redeployment functionality.

14.2 Container contract

The container contract of the JACC specification specifies how the container creates the permission objects during access checks and calls the provider with appropriate information to help make the access decision. When a resource is being accessed, the container is expected to create the appropriate permission object and call the provider's `Policy.implies` method. The container is also expected to register what are called the policy context handler objects that contain additional information to make the access decision. The following handlers are required to be registered by the containers. The container contract defines the following components:

- ▶ Policy Enforcement by Servlet Containers which includes the Evaluation of Transport Guarantees, Predispatch Decision, and Application Embedded Privilege Test.
- ▶ Provider Support for Servlet Policy Enforcement which includes Servlet Policy Decision Semantics, Matching Qualified URL Pattern Names, `WebResourcePermission` Matching Rules, `WebRoleRefPermission` Matching Rules, and `WebUserDataPermission` Matching Rules.
- ▶ Policy Enforcement by EJB Containers which includes the EJB Predispatch Decision and EJB Application Embedded Privilege Test.
- ▶ Provider of Support for EJB Policy Enforcement which includes EJB Policy Decision Semantics, `EJBMethodPermission` Matching Rules, and `EJBRoleRefPermission` Matching Rules.
- ▶ Component `runAs` Identity
- ▶ Setting the Policy Context
- ▶ Checking `AccessControlContext` Independent Grants
- ▶ Checking the Caller for a Permission
- ▶ Missing Policy Context
- ▶ Default Policy Context
- ▶ Policy Compatibility Requirements
- ▶ Optimization of Permission Evaluations

14.3 Provider contract

The provider contract in the JACC specification specifies that each JRE of an Application Server must be provided with classes that implement the `PolicyConfigurationFactory` class and the `PolicyConfiguration` interface. The classes are used by the container to propagate the security information to the provider. The provider is also expected to provide the implementation for the `java.security.Policy` object. This `Policy` object must assume responsibility for performing all access decisions within the JRE in which it is installed. The `Policy` object can delegate the non-javax.security.jacc access decisions to the corresponding default system `Policy` implementation class.

The Provider contract defines the following components:

- ▶ Policy Implementation Class
- ▶ Policy Configuration Interface
- ▶ PolicyContext Class and Context Handlers
- ▶ What a Provider Must Do
- ▶ Optional Provider Support for JAAS Policy Object
- ▶ What the Application Server Must Do

14.4 Why JACC

In the J2EE 1.3 Specification, there is no specification address. The Application Server vendor implementations make the access decisions and proprietary interfaces are used for third party vendor product integration. There is no standard way for third party authorization providers such as Tivoli Access Manager to plug in to Application Servers to make the decisions. There is no standard way for the third party providers to collect the security policy information from the application or from the Application Servers. To address these issues, the JACC was introduced in the J2EE 1.4 specification.

14.5 JACC in WebSphere Application Server V6.1

WebSphere Application Server V6.1 supports JACC and provides several key components to support the provider contract, container contract, and deployment tool contract. The JACC specification only specifies a contract to propagate the policy information to the provider using the `PolicyConfiguration` interface and `PolicyConfigurationFactory` abstract class. There is no contract specified to propagate the authorization table information to the provider. It is the responsibility of the provider to present some kind of management interface to handle principals (users/groups) to roles.

To propagate the authorization table information, WebSphere Application Server provides interfaces RoleConfigurationFactory and RoleConfiguration. The implementation of these interfaces is optional.

Figure 14-2 shows the WebSphere support for the Deployment tools Contract, Provider Contract, and Container Contract as specified by the JACC specification.

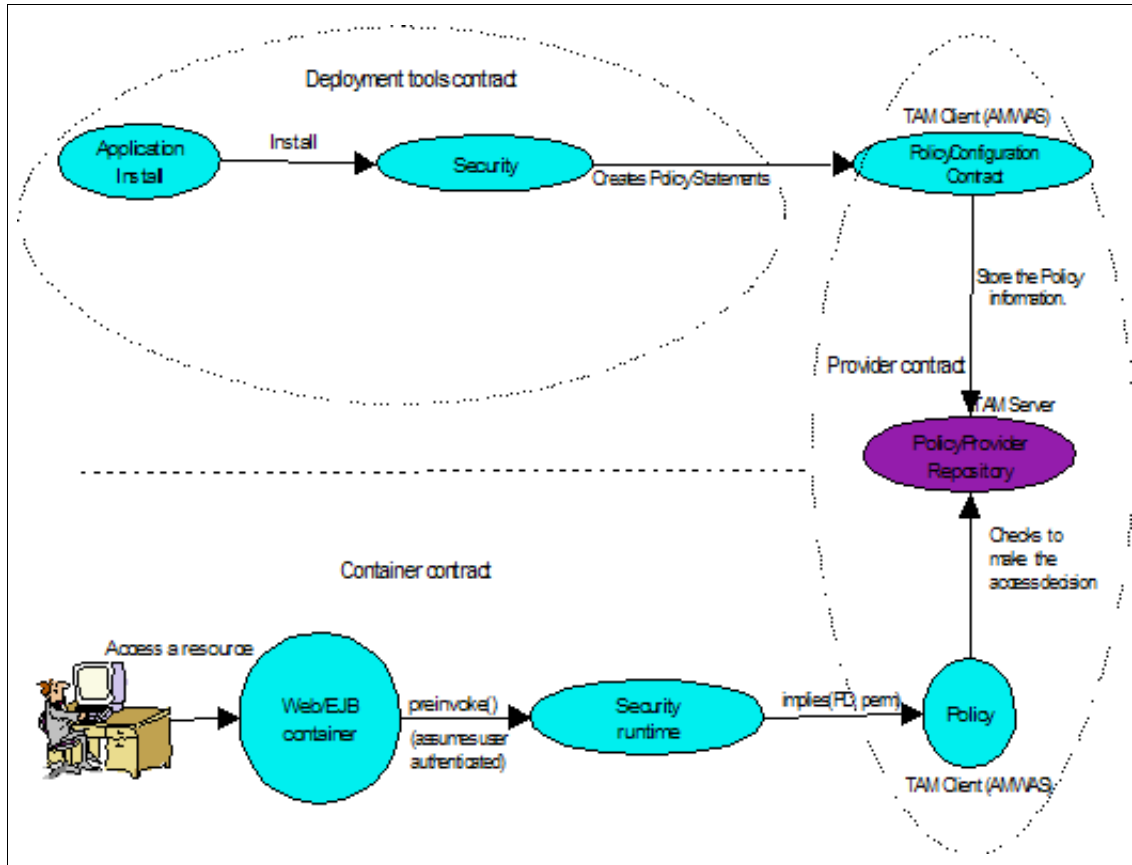


Figure 14-2 JACC support in WebSphere Application Server V6.1

Deployment tools contract

The deployment tools contract components must employ the following steps:

1. Create a PolicyContext identifier (contextID) for the module.
2. Get the PolicyConfiguration for the contextID.
3. Translate the declarative policy in DD into appropriate permission classes.
4. Create Policy Statements in the PolicyConfiguration objects using the permission classes.
5. Commit the changes and refresh the Policy.

Container contract

The container contract components must employ the following steps:

1. Create the PolicyContext identifier for the module.
2. Register the various PolicyContextHandlers.
3. Create the Protection Domain (PD) and the appropriate Permission object (perm).

Provider contract

The provider makes the access decision based on the permission object.

14.5.1 JACC access decisions in WebSphere Application Server V6.1

The authenticated user makes a request to the Web or the EJB resource. The security run time makes the decision of whether to allow the access. This is called an access decision. Figure 14-3 on page 411 shows the generic flow of the access decision for protected resources under the WebSphere environment, where external authorization is enabled through JACC.

Based on JACC, the appropriate permission object is created, the appropriate policy context handlers are registered, and the appropriate policy context identifier (contextID) is set. A call is made to the java.security.

Figure 14-3 shows how the provider implements the policy object method to make the access decision.

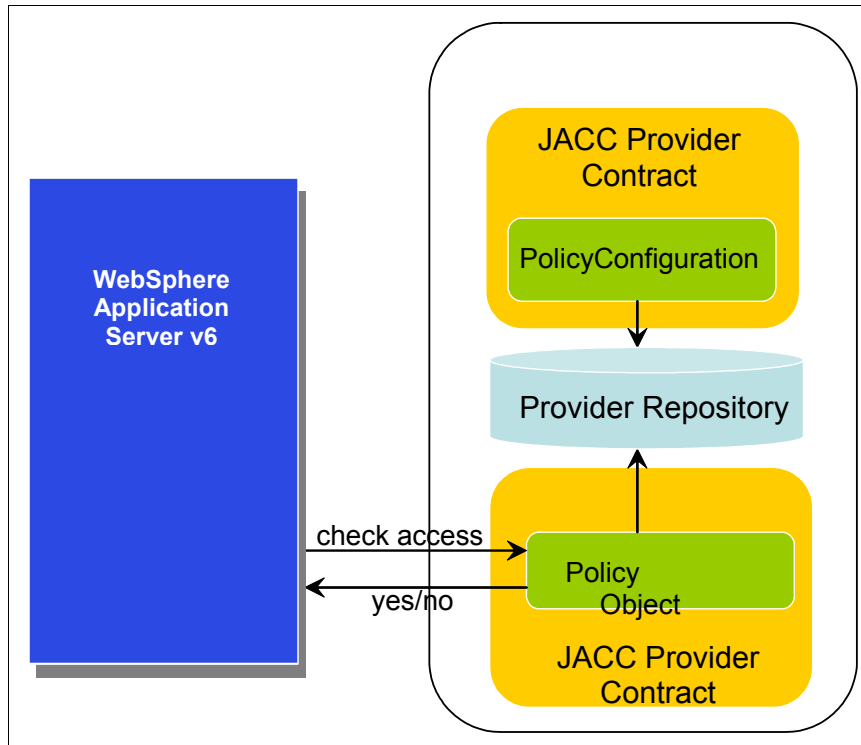


Figure 14-3 Externalized decisions making with JACC in WebSphere V6

Access decisions for enterprise beans

The authenticated user makes a request to the protected EJB resource. WebSphere security run time delegates the authorization check to the security run time.

Figure 14-4 shows the flow of steps that occur when the JACC is enabled for external authorization.

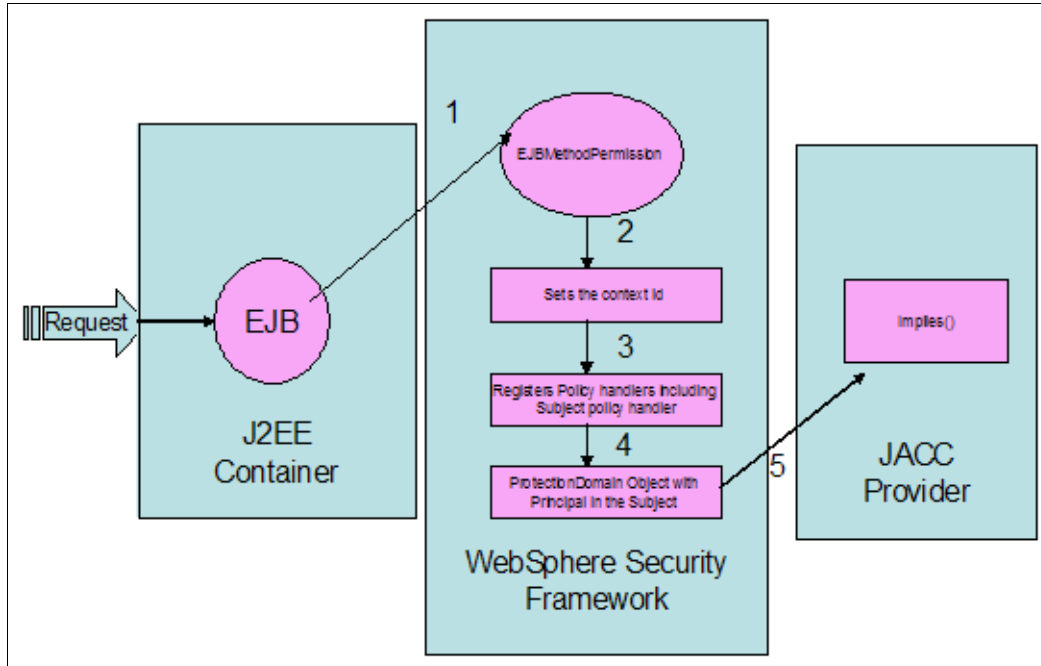


Figure 14-4 Logical steps for decision making

The steps shown in Figure 14-4 are explained as follows:

1. It creates the EJBMethodPermission object using the bean name, method name, interface name, and the method signature.
2. It creates the contextID and sets it on the thread by using the PolicyContext.setContextID(contextID) method.
3. It registers the required policy context handlers, including the Subject policy context handler.
4. It creates the ProtectionDomain object with principal in the subject. If there is no principal, null is passed for the principal name.
5. The access decision is delegated to the JACC provider by calling the implies() method of the Policy object, which is implemented by the provider. The

EJBMethodPermission and the ProtectionDomain objects are passed to this method.

6. The isCallerInRole() access check also follows the same process, except that an EJBRoleRefPermission object is created instead of an EJBMethodPermission.

Access decisions for Web resources

The authenticated user makes a request to the protected Web resource. The WebSphere security run time delegates the authorization check to security run time. Following are the steps that take place when the JACC is enabled for external authorization.

Flow for the subject Everyone

When JACC is enabled for external authorization, the following actions apply to the subject called Everyone:

1. The WebResourcePermission is constructed with urlPattern and the Hypertext Transfer Protocol (HTTP) method accessed.
2. A ProtectionDomain with a null principal name is created.
3. The JACC provider's Policy.implies() method is called with the permission and the protection domain. If the Uniform Resource Identifier (URI) access is unchecked (or given access to the subject called Everyone), the provider must permit access (return true) in the implies() method. Access is then granted without further checks.

Using the HTTPS protocol

When JACC is enabled for external authorization, the following actions apply when using the HTTPS protocol:

1. The WebUserDataPermission is constructed with the urlPattern accessed, along with the HTTP method invoked, and the transport type of the request. If the request is over Hypertext Transfer Protocol Secure (HTTPS), the transport type is set to CONFIDENTIAL, otherwise, null is passed.
2. ProtectionDomain with a null principal name is created.
3. The JACC provider's Policy.implies() method is called with the permission and the protection domain. If the request is using the HTTPS protocol and the implies returns false, the HTTP 403 error is returned to imply excluded/precluded permission and no further checks are performed. If the request is not using the HTTPS protocol, and the implies returns false, the request is redirected over HTTPS.

The provider's implies() method is called using the Permission object and the ProtectionDomain created previously. If the user is granted permission to access

the resource, the `implies()` method must return true. If the user is not granted access, the `implies()` method must return false.

14.5.2 JACC policy context identifiers in WebSphere Application Server V6.1

The JACC specification defines that “It must be possible to define separate authorization policy contexts corresponding to each deployed instance of a J2EE module. This per module scoping of policy context is necessary to provide for the independent administration of policy contexts corresponding to individual application modules (perhaps multiply deployed) within a common Policy provider. Each policy context contains all of the policy statements (as defined by this specification) that affect access to the resources in one or more deployed modules. At policy configuration, a `PolicyConfiguration` object is created for each policy context, and populated with the policy statements (represented by permission objects) corresponding to the context. Each policy context has an associated policy context identifier.”

A policy context identifier is defined as a unique string that represents a policy context. WebSphere Application Server makes the `contextID` unique by using the string `href:cellName/appName/moduleName` as the `contextID` format for the modules. The `href` part of the string indicates that a hierarchical name is passed as the `contextID`.

14.5.3 WebSphere extensions to the JACC specification

WebSphere provides three extension interfaces to the JACC specification: `InitializeJACCProvider`, `RoleConfiguration`, and `RoleConfigurationFactory`.

The JACC specification only specifies a contract to propagate the policy information to the provider. There is no contract specified to propagate the authorization table information to the provider. The provider must present a management interface to handle principals (users/groups) to roles.

To propagate the authorization table information, WebSphere Application Server provides interfaces `RoleConfigurationFactory` and `RoleConfiguration`. The implementation of these interfaces is optional. In some cases, the JACC provider requires initialization during server startup so that it can communicate with the server during startup. WebSphere provides the `InitializeJACCProvider` interface for this reason. When this interface is implemented, it is called during server startup. Any custom properties in the JACC configuration model are propagated to the `initialize` method of this implementation. The custom properties can be entered either by using the administrative console or by scripting.

During server shutdown, the cleanup method is called for any cleanup work that a provider requires. Implementation of this interface is strictly optional, and must be used only if the provider requires initialization during server startup.

The *RoleConfiguration interface* is used to propagate the authorization information to the provider. This interface is similar to the PolicyConfiguration interface found in JACC. The *RoleConfigurationFactory interface* is similar to the PolicyConfigurationFactory interface introduced by JACC and is used to obtain RoleConfiguration objects based on the contextIDs.

14.5.4 JACC policy propagation in WebSphere Application Server V6.1

The policy propagation between the WebSphere Application Server and JACC Provider, as shown in Figure 14-5, is handled in the following ways:

- ▶ A new application is installed and the configuration is saved.
- ▶ An application is uninstalled and the configuration is saved.
- ▶ There is an update to an existing application either with a new module or an update to an existing module with security policy changes.

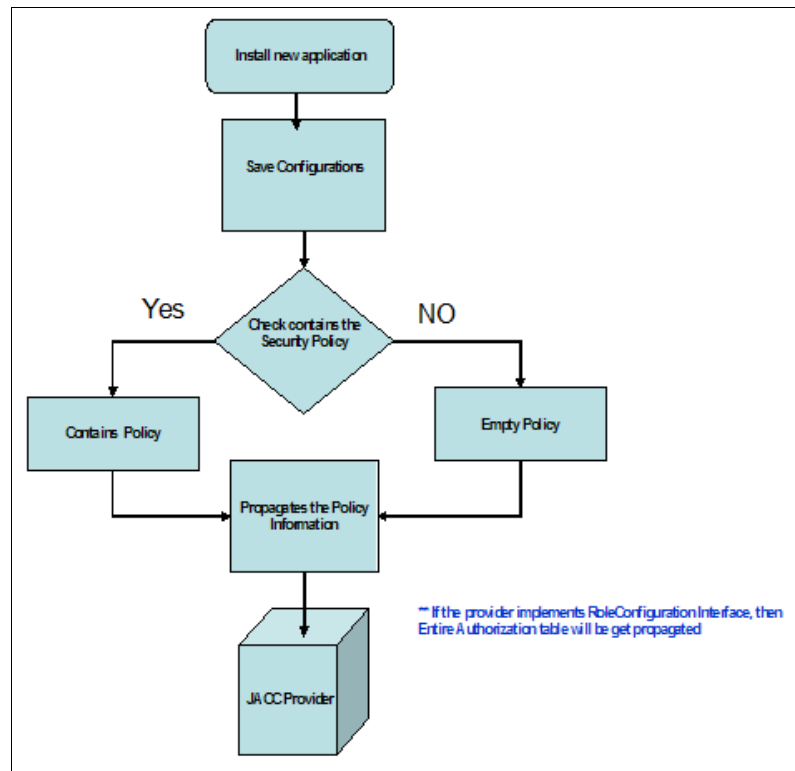


Figure 14-5 JACC policy propagation during application install

When an application is installed or deployed in the WebSphere Application Server, the security policy information in the application is propagated to the provider when the configuration is saved. The contextID for that application is saved in its application.xml file, used for propagating the policy to the JACC provider, and also for access decisions for J2EE resources.

When an application is uninstalled as shown in Figure 14-6, the security policy information in the application is removed from the provider when the configuration is saved.

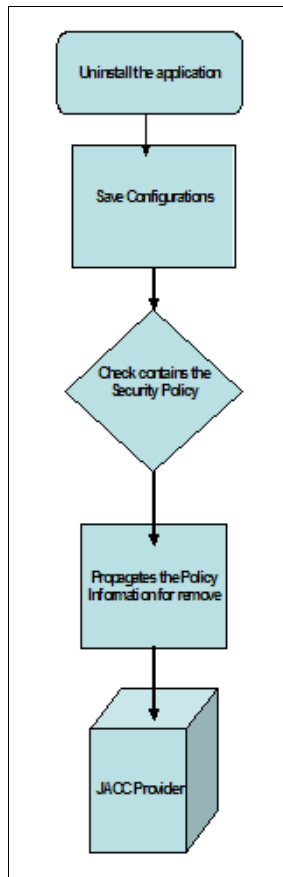


Figure 14-6 JACC policy removal during application uninstall

If you update the existing application or add a new module to an existing application as shown in Figure 14-7, the information in the impacted modules is propagated to the provider by default. A module is impacted when the deployment descriptor of the module has changed as part of the update. If the provider supports the RoleConfiguration interfaces, the entire authorization table for that application is propagated to the provider.

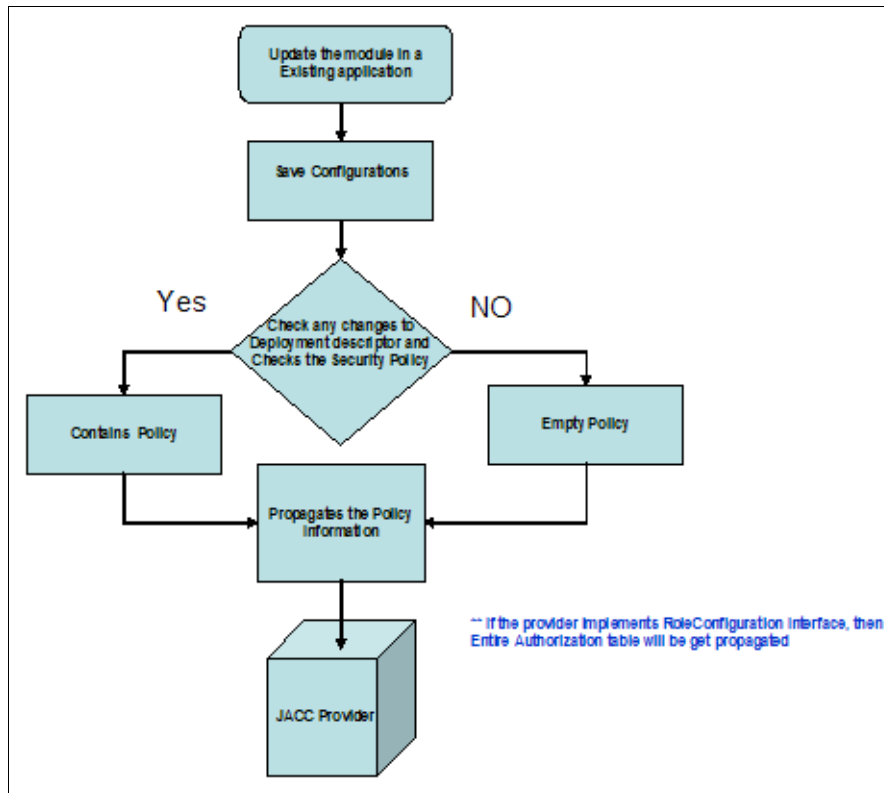


Figure 14-7 JACC policy update during application update

For some reason, if the security information is not to be propagated to the provider during application updates, you can set the Java virtual machine (JVM) property `com.ibm.websphere.security.jacc.propagateonappupdate` to `false` in the deployment manager or the unmanaged base Application Server. If this property is set to `false`, then none of the updates to an existing application in the server is propagated to the provider. Also, you can set this property on a per application basis using the custom properties of an application. You can use the `wsadmin` tool to set the custom property of an application. If this property is set at the application level, none of the updates to that application is propagated to the provider. If the update to an application is a full update, for example, a new

application .ear file is used to replace the existing one, the provider is refreshed with the entire application security policy information.

In the Network Deployment (ND) environment, when an application is installed and saved, the security policy information in that application is updated in the provider from the deployment manager (dmgr or cell). However, the application is not propagated to its respective nodes until the synchronization command is issued and completed.

Also, in the ND setup, when an application is uninstalled and saved at the deployment manager, the policy for that application is removed from the JACC provider. However, unless the synchronization command is issued and completed from the deployment manager to the nodes hosting the application, the applications are still running in the respective nodes. In this instance, any access to this application must be denied because the JACC provider does not contain the required information to make the access decision for that application.

Note that any updates to the application already installed as described previously are also propagated to the provider from the deployment manager. The changes in the provider are not in sync with the applications in the nodes until the synchronization is completed.

14.5.5 Manual policy propagation

It is possible to manually propagate policy information of installed applications to the JACC provider. You can do this if you use **wsadmin** scripting that is shown in Example 14-1 and Example 14-2. You may want to do this in a case where there are network difficulties during the initial JACC policy propagation during application installation and not all JACC providers have the required information. If this happens you have two choices, either reinstall the application or manually propagate the policy and authorization information.

Manual policy propagation uses the `propagatePolicyToJACCProvider(String appNames)` function in the `SecurityAdmin` MBean. To do this the server must be running. This tool propagates the deployment descriptors of the specified applications to the JACC providers. If the JACC provider has implemented the `RoleConfiguration` and `RoleConfigurationFactory` interfaces, then authorization information provided in the binding file of the EAR is also propagated. `appNames` is a colon (:) separated list of application names. If null is specified then the policy information for all deployed applications are propagated.

Example 14-1 shows the use of **wsadmin** scripting to manually propagate policy information for a single server.

Example 14-1 Using wsadmin to manually propagate policy information for single server

```
C:\Program Files\IBM\WebSphere\AppServer\profiles\paul\bin>wsadmin.bat
-username wasadmin -password passwOrd
wsadmin># First of all we need to get the correct SecurityAdmin MBean.
wsadmin>set serverSecAdm [$AdminControl queryNames
type=SecurityAdmin,process=server1,*]
WebSphere:name=SecurityAdmin,process=server1,platform=proxy,node=paulwN
ode01,version=6.1.0.0,type=SecurityAdmin,mbeanIdentifier=SecurityAdmin,
cell=bchhs409Node02Cell,spec=1.0
wsadmin># or for a deployment manager
wsadmin>set serverSecAdm [$AdminControl queryNames
type=SecurityAdmin,process=dmgr,*]
wsadmin># Now we specify the applications we are going to propagate
information on behalf of
wsadmin>set appNames [list ItsohelloEAR:PlantsByWebSphere]
ItsohelloEAR:PlantsByWebSphere
wsadmin># or for all deployed applications
wsadmin>set allApps [list null]
null
wsadmin>$AdminControl invoke $serverSecAdm
propagatePolicyToJACCProvider $appNames
```

Example 14-2 shows the use of **wsadmin** to manually propagate policy information for a cluster.

Example 14-2 Using wsadmin to manually propagate policy information for a cluster

```
C:\Program
Files\IBM\WebSphere\AppServer\profiles\Dmgr01\bin>wsadmin.bat -username
wasadmin -password passwOrd
wsadmin># First of all we need to get the correct SecurityAdmin MBean.
wsadmin>set dmgrSecAdm [$AdminControl queryNames
type=SecurityAdmin,process=dmgr,*]
WebSphere:name=SecurityAdmin,process=dmgr,platform=proxy,node=bchhs409C
ellManager01,version=6.1.0.0,type=SecurityAdmin,mbeanIdentifier=Securit
yAdmin,cell=bchhs409Cell01,spec=1.0
wsadmin># Now we specify the applications we are going to propagate
information on behalf of
wsadmin>set appNames [list ItsohelloEAR:PlantsByWebSphere]
ItsohelloEAR:PlantsByWebSphere
wsadmin># or for all deployed applications
wsadmin>set allApps [list null]
```

```
null
wsadmin>$AdminControl invoke $dmgrSecAdm propagatePolicyToJACCProvider
$allApps
```

14.5.6 Dynamic module updates in WebSphere Application Server V6.1 for JACC

WebSphere handles the dynamic module update with respect to JACC for Web modules. When the Web module is updated, only that application must be restarted in native authorization mode. In the case of JACC enabled, the dynamic module update depends on the provider support to handle such updates specific to the security modules. You must select the **Dynamic module** check box for the change to take effect.

14.6 Integrating Tivoli Access Manager as an external JACC provider

To configure WebSphere Application Server to use Tivoli Access Manager as the external authorization engine:

1. Ensure that WebSphere Application Server and the Tivoli Access Manager Policy Server are sharing the same Lightweight Directory Access Protocol (LDAP). Also ensure that the WebSphere administrative user has a valid account in Tivoli Access Manager.

Memory shortage problems: You might encounter memory shortage problems when WebSphere Application Server runs with Tivoli Access Manager as the JACC provider. To fix this issue prior to configuring Tivoli Access Manager as the JACC provider, set the `com.tivoli.pdas.atcc.ATCCache.enabled` property to `false` in the `amwas.amjacc.template.properties` file. This file is in the `PROFILE_HOME/config/cells/cell_name/` directory. After setting this property to `false`, restart WebSphere Application Server.

2. Start the WebSphere Application Server Administrative Console, and log in.
3. From the left navigation menu, click **Security** → **Secure administration, applications, and infrastructure**.
4. Click **External authorization providers**.

5. On the External authorization providers page, specify whether to use the default authorization provider or an external JACC provider. Select **External authorization using a JACC provider** and click **Apply**.
6. Navigate back to the External authorization providers page, and under General Properties, click **External JACC provider**. If the Tivoli Access Manager properties are not prefilled, specify the following properties as shown in Figure 14-8 on page 422 and apply the changes:
 - For Policy class name, type `com.tivoli.pd.as.jacc.TAMPolicy`.
 - Policy configuration factory class name, type `com.tivoli.pd.as.jacc.TAMPolicyConfigurationFactory`.
 - For Role configuration factory class name, type `com.tivoli.pd.as.jacc.TAMRoleConfigurationFactory`.
 - For JACC provider initialization class name, type `com.tivoli.pd.as.jacc.cfg.TAMConfigInitialize`.
 - For Requires the EJB arguments policy context handler for access decisions, type `false`.
 - For Supports dynamic module updates, type `true`.

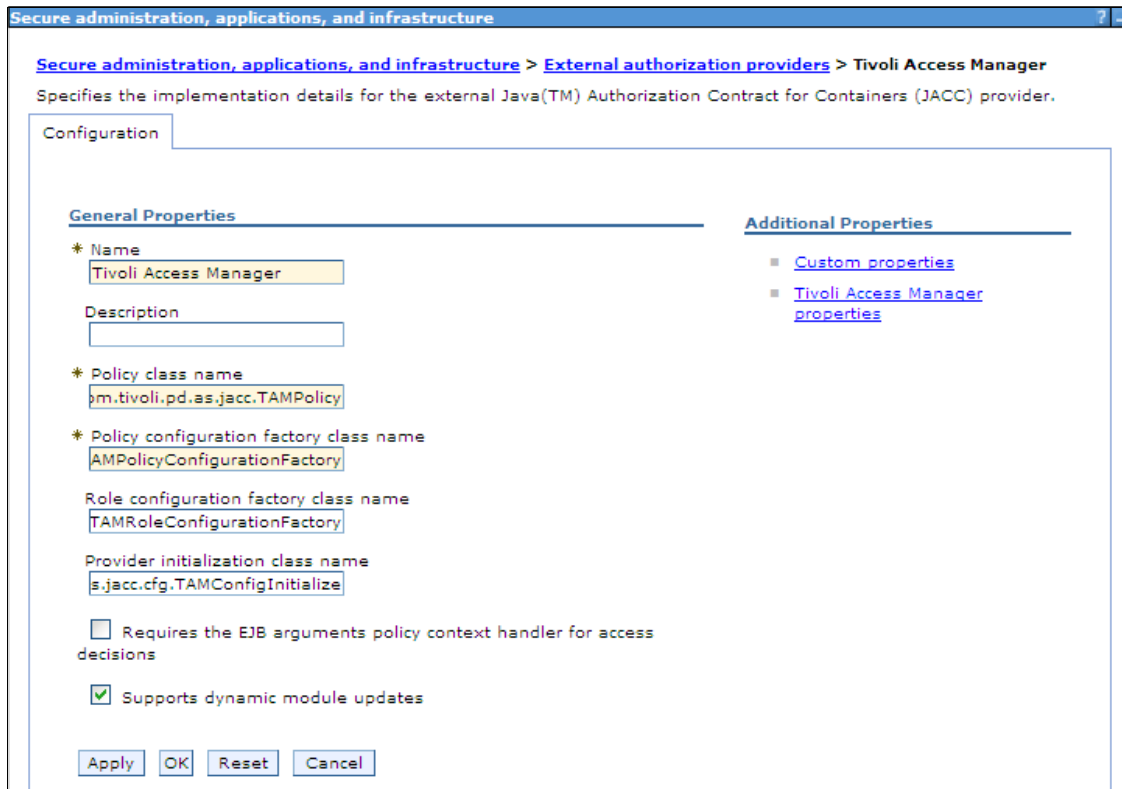


Figure 14-8 Specifying the Tivoli Access Manager JACC classes

7. Under Additional Properties, click **Tivoli Access Manager properties**.
8. Enter the following information:
 - Enable embedded Tivoli Access Manager
Select this option to enable the Tivoli Access Manager.
 - Ignore errors during embedded Tivoli Access Manager Disablement
Select this option when you want to unconfigure the JACC provider. Do not select this option during configuration.
 - Client listening point set
WebSphere Application Server must listen using a Transmission Control Protocol/Internet Protocol (TCP/IP) port for authorization database updates from the policy server. More than one process can run on a particular node or machine. Enter the listening ports used by Tivoli Access Manager Clients, separated by a comma. If a range of ports is specified,

separate the lower and higher values by a colon (for example, 7999, 9990:999).

- Policy server

Enter the name of the Tivoli Access Manager Policy server and the connection port. Use the form `policy_server:port`. The policy communication port is set at the time of the Tivoli Access Manager configuration, and the default is 7135.

- Authorization servers

Enter the name of the Tivoli Access Manager Authorization server. Use the form `auth_server:port:priority`. The authorization server communication port is set at the time of the Tivoli Access Manager configuration, and the default is 7136. More than one authorization server can be specified by separating the entries with commas. Specifying more than one authorization server at a time is useful for reasons of failover. The priority value is determined by the order of the authorization server use (for example, `auth_server1:7136:1` and `auth_server2:7137:2`). A priority value of 1 is required when configuring against a single authorization server.

Important: You must specify a priority or else the configuration fails. For troubleshooting tips, see the WebSphere 6.1 Information Center and then go to **Troubleshooting and support** → **Troubleshooting WebSphere applications** → **Security** → **Troubleshooting security configurations** → **Authorization provider troubleshooting tips**.

- Administrator user name

Enter the Tivoli Access Manager Administrator user name that was created when Tivoli Access Manager was configured (it is usually `sec_master`).

- Administrator user password

Enter the Tivoli Access Manager administrator password.

- User registry distinguished name suffix

Enter the Distinguished Name (DN) suffix for the user registry that is shared between Tivoli Access Manager and WebSphere (for example, `o=ibm,c=us`).

- Security domain

You can create more than one security domain in Tivoli Access Manager, each with its own administrative user. Users, groups, and other objects are created within a specific domain, and are not permitted to access resource in another domain. Enter the name of the Tivoli Access Manager

security domain that is used to store WebSphere Application Server users and groups. If a security domain has not been established at the time of the Tivoli Access Manager configuration, leave the value as Default.

– Administrator user distinguished name

Enter the full Distinguished Name of the WebSphere security administrator ID (for example, cn=wasadmin, o=ibm,c=us). The ID name must match the Primary administrative user name on the LDAP User Registry panel in the administrative console. To access this panel, click **Security** → **Secure administration, applications, and infrastructure**. Under **Available realm definitions**, click **Stand-alone LDAP registry**.

Note: The Embedded Tivoli Access Manager client only supports Stand-alone LDAP Registries or a Federated repository containing a single LDAP registry, which is the equivalent to a Stand-alone LDAP Registry.

Figure 14-9 shows the Tivoli Access Manager client settings.

General Properties

Tivoli Access Manager client settings

- Enable embedded Tivoli Access Manager
- Ignore errors during embedded Tivoli Access Manager disablement

* Client listening port set

8900:8999

Tivoli Access Manager server settings

- * Policy server: bchhs409
- * Authorization servers: bchhs409
- * Administrator user name: sec_master
- * Administrator user password:
- * User registry distinguished name suffix: o=ibm,c=us
- * Security domain: Default

WebSphere Application Server settings

- * Administrator user distinguished name: cn=wasadmin,o=ibm,c=us

Apply OK Reset Cancel

Figure 14-9 Tivoli Access Manager client settings

14.6.1 Disabling the embedded Tivoli Access Manager

In a Network Deployment architecture, ensure all managed servers, including node agents, are started, then perform the following process after you are on the deployment management server. Information from the unconfigure operation is forwarded to managed servers, including node agents, when the server is restarted. The managed servers then require a restart for changes to take effect.

Disabling Tivoli Access Manager by using the Administrative Console

To unconfigure the Tivoli Access Manager JACC provider by using the WebSphere Application Server Administrative Console:

1. Select **Security** → **Secure administration, applications, and infrastructure**.
2. On the right sidebar, click **Authorization provider** (default).
3. Under Related items, click **External JACC provider**.
4. Under Additional properties, click **Tivoli Access Manager Properties**. The configuration screen for the Tivoli Access Manager JACC provider is displayed.
5. Deselect **Enable embedded Tivoli Access Manager** option. If you want to ignore errors when unconfiguring, select **Ignore errors during embedded Tivoli Access Manager Disablement** option. Select this option only when the Tivoli Access Manager domain is in an irreparable state.
6. Click **OK**.
7. Restart all WebSphere Application Server instances for the changes to take effect.

Disabling Tivoli Access Manager by using wsadmin

To disable the Tivoli Access Manager JACC provider:

1. Start the `wsadmin` command line utility.
2. From the `wsadmin` prompt, enter the following command:
`$AdminTask unconfigureTAM -interactive`
3. Tell WebSphere Application Server *not* to use an external JACC provider as shown in Example 14-3.

Example 14-3 Eliminating use of an external JACC

```
wsadmin># Get the Authorization Configuration
wsadmin>set authConfig [ $AdminConfig list AuthorizationConfig ]
(cells/localhostNode01Cell|security.xml#AuthorizationConfig_1)
```

```

wsadmin># Just seeing what we have
wsadmin>$AdminConfig show $authConfig
{authorizationProviders {"Tivoli Access
Manager(cells/localhostNode01Cell|security.xml#AuthorizationProvider
_1)"}}
{useJACCProvider true}
{useNativeAuthorization false}
wsadmin># Stop using any external JACC providers
wsadmin>$AdminConfig modify $authConfig [list { useJACCProvider
false } ]

wsadmin>$AdminConfig show $authConfig
{authorizationProviders {"Tivoli Access
Manager(cells/localhostNode01Cell|security.xml#AuthorizationProvider
_1)"}}
{useJACCProvider false}
{useNativeAuthorization false}
wsadmin>$AdminConfig save

```

4. When all the information is entered, enter F to save the properties (or C to cancel the unconfiguration process and discard entered information).
5. Restart all WebSphere Application Server instances for the changes to take effect.

14.6.2 Reconfiguring the JACC provider by using wsadmin

Reconfigure the JACC provider by using the following **wsadmin** command:

```
$AdminTask reconfigureTAM interactive
```

Enter all new and existing options.

14.7 Sample application for JACC

You can find details about the sample application in “Sample application for testing JACC” on page 513.



Web services security

This chapter discusses Web services security in WebSphere Application Server V6.1.

15.1 Web services security exposures

Web services security is one of the most important Web services subjects. When using Web services, security exposures that exist are similar to other Internet services, middleware-based applications, and communications.

To explain the Web services security exposures, we use a bank teller scenario as an example, as shown in Figure 15-1. The bank teller (Web service consumer) connects over the Internet to the bank's data center (Web service provider). We assume there is no security applied at all, which is not realistic, but necessary for the example.

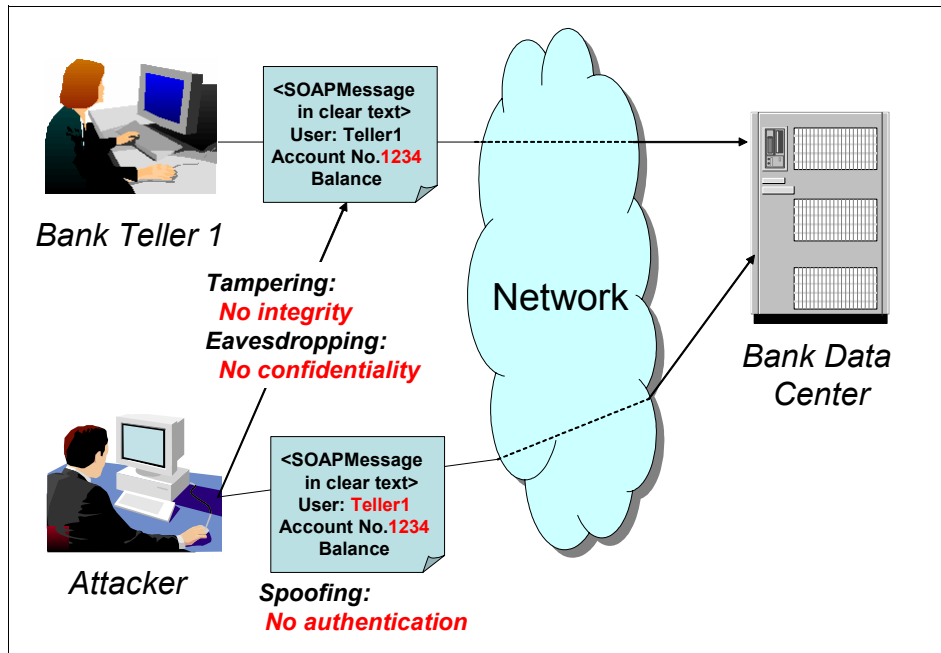


Figure 15-1 Common security exposures in a sample bank teller application based on Web services

This example has the following major risk factors:

- ▶ Spoofing: No authentication

An attacker can send a modified SOAP message to the service provider, pretending to be a bank teller, to get confidential information, or to withdraw money from another customers account. By applying authentication to the Web services, this security exposure can be eliminated.

► Tampering: No integrity

The SOAP message is intercepted between the Web service client and server. An attacker can modify the message, for example, and deposit the money into another account by changing the account number. Because there is no integrity constraint, the Web service server does not check if the message is valid, and accepts the modified transaction.

Applying integrity mechanism to the Web services, this security exposure can be eliminated.

► Eavesdropping: No confidentiality

An attacker can intercept the SOAP message, and read all contained information. Because the message is not encrypted, confidential customer or bank information can end up in the wrong hands.

This exposure exists because the account and balance information is sent over the network in plain text.

Applying a confidentiality mechanism to the Web services, this security exposure can be eliminated.

To prevent the described security exposures, the following mechanisms can be applied to secure a Web services environment as shown in Figure 15-2:

- Message level security: Web services security (WS-Security)
- Transport level security: TLS/SSL

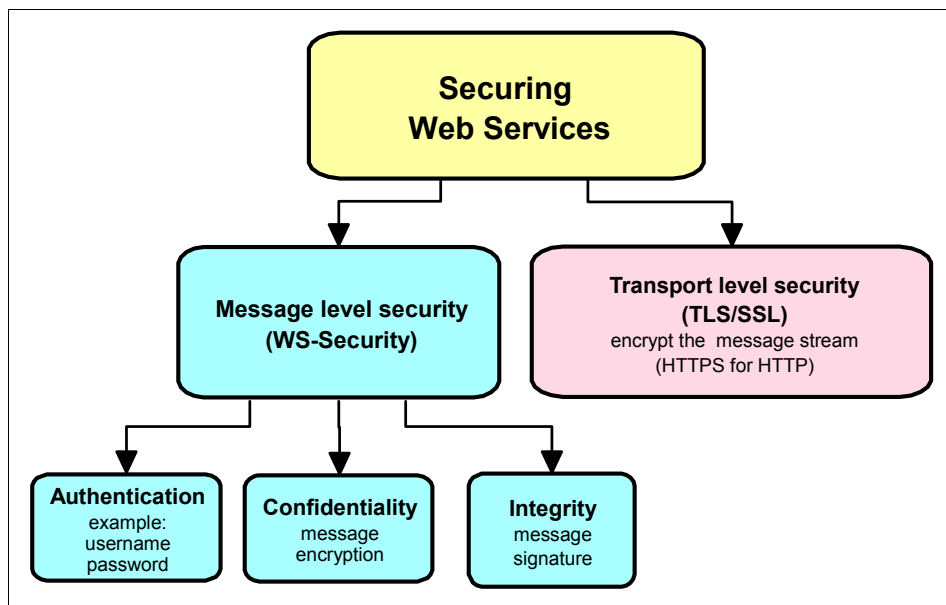


Figure 15-2 Securing Web services

Depending on the demanded level of application security, you can apply one or more of these security mechanisms. Also, a combination of message-level security and transport-level security can be implemented.

The more the security mechanisms are implemented, which increases the security effect, the more influence on other non-functional requirements is given. Therefore, while designing a Web services security solution, it is kept in mind that security has an impact on the following non-functional requirements:

- ▶ System capacity

Any applied security mechanism has an impact on system resource usage (for example CPU and memory usage). Therefore, when planning a Web services environment, the required security *overhead* must be considered in the system capacity and volume planning.

The non-functional requirements, capacity and volume, cover, for example, the number of concurrent users and the number of transactions per second. This has influence on the required system infrastructure (hardware, network).

- ▶ Performance

Security mechanisms and functions also impact the applications response time. When defining the Web services system response time requirements, keep in mind that the response times are affected when you apply security.

The performance requirement for a system defines the response time for a main application operation, for example, less than 1 second for 90% of all transactions.

Note: Applying security is not only a question of feasibility, the additional system resources and the influence on the response time must also be considered.

The WS-Security specification, and Secure Sockets Layer (SSL) mechanism is covered in detail in the next sections.

15.2 WS-Security

This section introduces WS-Security concepts. You can find more information about the various WS-Security specifications in 15.2.3, “WS-Security roadmap” on page 436.

15.2.1 WS-Security concepts

The WS-Security specification provides a message-level security which is used when building secure Web services to implement message content integrity and confidentiality. The advantage of using WS-Security over SSL is that it can provide End-to-End Message Level security. This means that the message security can be protected even if the message goes through multiple services, therefore, called intermediaries. Additionally, WS-Security is independent of the transport layer protocol. It can be used for any SOAP binding, for example Hypertext Transfer Protocol (HTTP) or Java Messaging Service (JMS). Using WS-Security, end-to-end security can be obtained as shown in Figure 15-3.

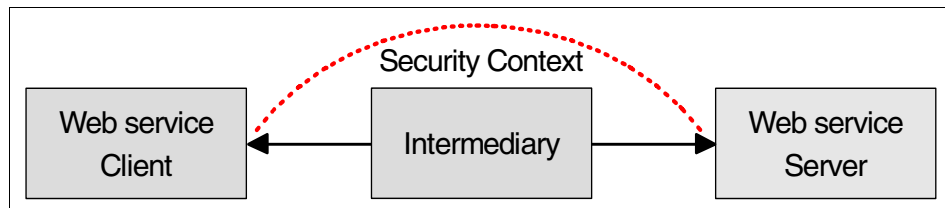


Figure 15-3 End to end security with message level security

The WS-Security specification, which is Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), is proposed by the Organization for the Advancement of Structured Information Standards (OASIS) WebSphere Services Security (WSS) Technical Committee. This specification proposes a standard set of SOAP extensions. This specification is flexible and is designed to be used as the basis for securing Web services within a wide variety of security models including public key infrastructure (PKI), Kerberos, and SSL. It provides support for multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies based on Extensible Markup Language (XML) Signature and XML Encryption to provide integrity or confidentiality.

The specification includes security token propagation, message integrity, and message confidentiality. However, these mechanisms by themselves do not address all the aspects of complete security solution. Therefore, WS-Security represents only one of the layers in a complex secure Web services solution design.

Important: With WS-Security 1.0, the wire format changed in a way which is not compatible with previous WS-Security drafts. Also, interoperability between implementations based on previous drafts and Version 1.0 is not possible.

The WS-Security specification defines the usage of XML Signature and XML Encryption:

- ▶ Message integrity is provided by XML Signature in conjunction with security tokens to ensure that modifications to messages are detected. See:
<http://www.w3c.org/Signature>
- ▶ Message confidentiality leverages XML Encryption in conjunction with security tokens to keep portions of a SOAP message confidential. See:
<http://www.w3c.org/Encryption>

15.2.2 Evolution of the WS-Security specification

The WS-Security support is provided in WebSphere 5.0.2 and later. Each version of WebSphere is based on different versions of the Web services security language.

The first version of the WS-Security specification was proposed by IBM, Microsoft, and Verisign in April 2002. After the formalization of the April 2002 specifications, the specification is transferred to OASIS consortium. For more information, see the following address:

<http://www.oasis-open.org>

In OASIS activities, core specification and many profiles which describe the use of a specific token framework in WS-Security have been discussed. The latest specification and profiles of WS-Security were proposed in March 2004 as the OASIS Standard. The latest core specification, Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) was standardized in March 2004. The two profiles, Web Services Security UsernameToken Profile 1.0 and Web Services Security X.509 Certificate Token Profile 1.0, were standardized at the same time.

There are other token profiles that OASIS is currently working on such as Web Services Security: SAML Token Profile, Web Services Security: Rights Expression@ Language (REL) Token Profile, Web Services Security: Kerberos Token Profile, Web Services Security Minimalist Profile (MProf), and Web Services Security: SOAP Message with Attachments (SwA) Profile.

The support of the April 2002 specification is provided in WebSphere 5.0.2 and 5.1. WebSphere Application Server Version 6.0 and 6.1 support the WS-Security 1.0 specification and two profiles (UserName-Token 1.0, x.509 Token 1.0). Figure 15-4 shows the evolution of WS-Security.

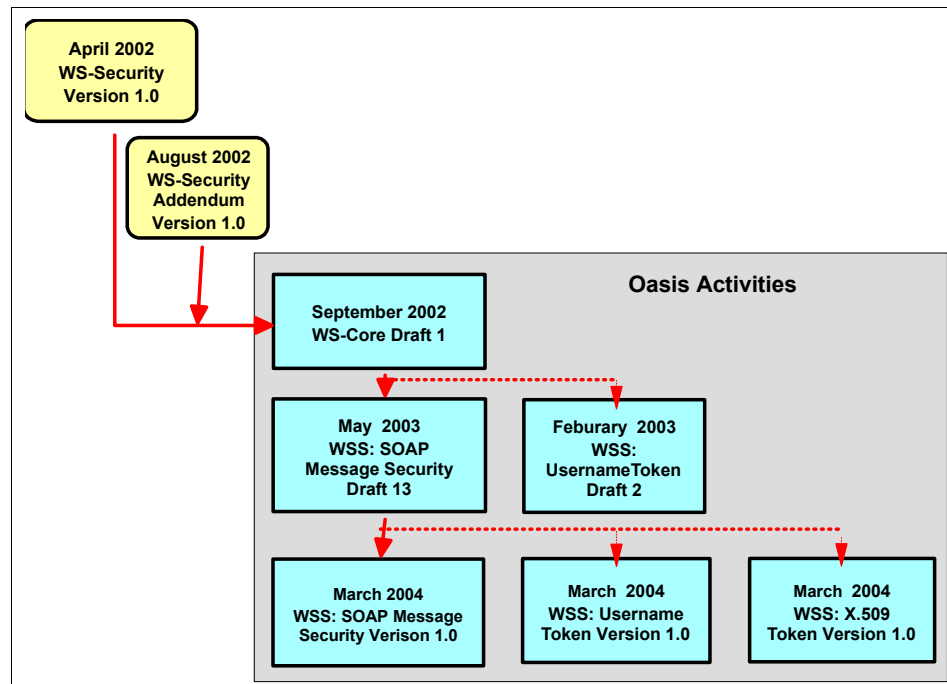


Figure 15-4 Evolution of Web services security

To read more about these standards, refer to the following Web addresses:

- ▶ Specification: Web Services Security (WS-Security) Version 1.0 (April 2002):
<http://www-106.ibm.com/developerworks/webservices/library/ws-secure/>
- ▶ Web Services Security Addendum (August 2002):
<http://www-106.ibm.com/developerworks/webservices/library/ws-secureadd.html>
- ▶ Web Services Security: SOAP Message Security V1.0 (March 2004):
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>

- ▶ Web Services Security: UsernameToken Profile V1.0 (March 2004):
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf>
- ▶ Web Services Security: X.509 Token Profile V1.0 (March 2004):
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf>

15.2.3 WS-Security roadmap

As mentioned, the WS-Security specification addresses only a subset of security services for all security aspects. A more general security model is required to cover other security aspects, such as logging and non-repudiation. The definition of those requirements is given in a common Web services security model framework, a security white paper of Web Services Security Roadmap proposed by IBM and Microsoft. We describe this Roadmap in the following section.

Web services security model framework

The Web services security model introduces a set of individual interrelated specifications to form a layering approach to security. It includes several aspects of security, which are identification, authentication, authorization, integrity, confidentiality, auditing, and non-repudiation. It is based on the WS-Security specification, co-developed by IBM, Microsoft, and VeriSign.

Figure 15-5 shows a schematic version of the Web services security model.

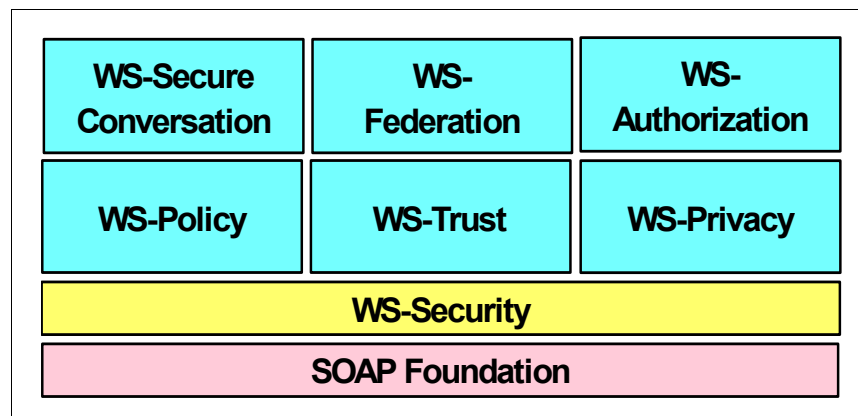


Figure 15-5 WS-Security roadmap

These specifications include different aspects of Web services security:

WS-Policy	Describes the capabilities and constraints of the security policies and other business policies on intermediaries and endpoints, for example, required security tokens, supported encryption algorithms, and privacy rules.
WS-Trust	Describes a framework for trust models that enables Web services to securely interoperate. This specification is responsible for managing trusts and establishing trust relationships.
WS-Privacy	Describes a model for how Web services and requestors state privacy preferences and organizational privacy practice statements.
WS-Federation	Describes how to manage and broker the trust relationships in a heterogeneous federated environment, including support for federated identities.
WS-Authorization	Describes how to manage authorization data and authorization policies.
WS-SecureConversation	Describes how to manage and authenticate message exchanges between parties, including security context exchange and establishing and deriving session keys.

The combination of these security specifications enables many scenarios, which are difficult or impossible to implement with today's more basic security mechanisms, such as transport securing or XML document encryption.

15.2.4 Example of WS-Security

This section provides some examples of SOAP messages with WS-Security. Using WS-Security, authentication mechanism, integrity, and confidentiality can be applied in the message level. In WebSphere Application Server V6.1, there are many options to apply these security mechanisms. In this section, the most typical scenarios of each mechanism are shown as an introduction.

As an overview, Figure 15-6 shows the Web services security elements added to the SOAP message.

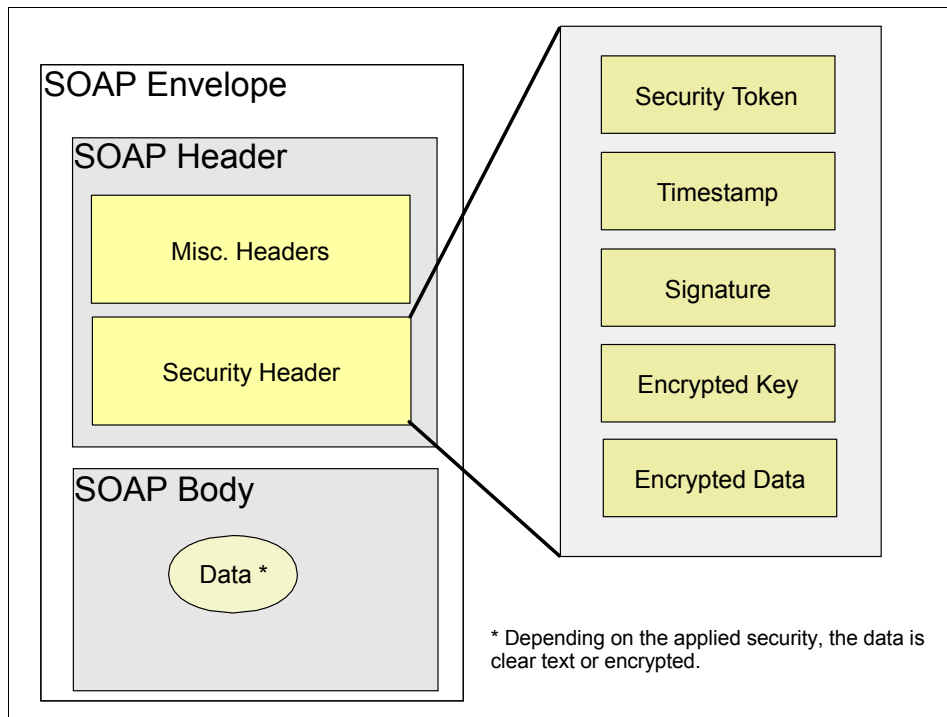


Figure 15-6 SOAP message security with WS-Security

Applying WS-Security, the SOAP security header is inserted under the SOAP envelope.

Authentication

The user name and password information as a Username Token is stored in the message. When the Username Token is received by the Web service server, the user name and password are extracted from the Username Token and they are verified. Only when both the user name and password are valid is the message accepted and processed at the server.

Using Username Token is just one of the ways of implementing authentication. This mechanism is also known as *basic authentication*. Other forms of authentication are digital signature, ID Assertion, Lightweight Third Party Authentication (LTPA), and custom tokens.

Enabling basic authentication in your application

To configure authentication:

1. On the client side, to insert the Username Token to a SOAP message, specify a security token and its token generator in the client's WS-Security configuration:
 - a. Specify a security token at Request Generator configuration. In case you are using *basic authentication*, the security token type must be Username. This security token is sent inside the SOAP message to the server.
 - b. Specify a token generator for Username Token at Request Generator configuration. The role of the token generator is to get the user name and password from the configuration file and generate the Username Token with this user name and password. The token generator class for Username Token, `UsernameTokenGenerator`, is provided by the WebSphere Web services security run time as a default implementation.
2. On the server side, to receive the client's Username Token, specify a security token in the server's WS-Security configuration, which the server and a token consumer require:
 - a. Specify a security token which is required by the server, in case of *basic authentication*, the required security token type is a Username similar to a client's configuration.
 - b. Specify a token consumer at Request Consumer configuration. The token consumer receives a security token in the request message and validates it. The token consumer class for Username Token, `UsernameTokenConsumer`, is provided by the WebSphere Web services security run time as a default implementation.
 - c. Turn on the application security in the WebSphere Application Server where the application is deployed.

Integrity

Integrity is applied to the application to ensure that no one illegally modifies the message while it is in transit. Essentially, integrity is provided by implementing an XML digital signature on the contents of the SOAP message. If the message data changes illegally, the signature is no longer valid.

In WebSphere Application Server V6.1, multiple and arbitrary parts of the message can be signed, for example a message body, security token and time stamp.

A signature is created based on a key that the sender is authorized to have. Unauthorized sniffers do not have this key. When the receiver gets the message, it too creates a signature using the message contents. Only if the two signatures

match does the receiver honor the message. If the signatures are different, an error is returned to the sender.

Enabling integrity in your application

To configure integrity:

1. On the client side, to specify the integrity of part of a SOAP message, specify the part that must be signed and the process of signing in the client's WS-Security configuration:
 - a. Specify the parts of the message that must be signed at Request Generator configuration. The message parts can be specified by the predefined keyword or XPath expression. Also, you can specify multiple parts which require a signature.
 - b. Specify key-related information which includes the location of the client's key, a type of key, and a password for protecting the key.
 - c. Specify signing information which defines how to sign to the specified part. Specify some options for signature such as a signature method algorithm or key-related information.
 - d. In a most typical integrity example, a security token is inserted in the SOAP message, which is used as signature verification by the server. In such an example, a token generator must be specified at Request Generator configuration. This token generator's role is to generate a token for signature verification. In this case, a token generator for X.509 certificate token, X509TokenGenerator, must be specified, which is provided by the WebSphere Web services security run time as a default implementation.
 - e. If a client expects a response that includes integrity information by the server, then the client also has to be configured to validate the integrity of the response message at Response Consumer configuration.
2. On the server side, to specify required integrity for part of a SOAP message, specify the part that must be signed and the process of verifying the signature in the server's WS-Security configuration:
 - a. Specify the parts of the message which require a signature at Request Consumer configuration. The message parts can be specified by the predefined keyword or XPath expression. Also, you can specify multiple parts which require a signature.
 - b. Specify key-related information which includes the location of the server's key, a type of key, and a password for protecting the key.
 - c. Specify signing information which defines how the specified part is to be signed. Specify some options for signature, such as a signature method algorithm or key-related information.

- d. In a most typical integrity example, a security token is inserted in to the SOAP message, which is used as signature verification by the server. In such an example, a token consumer must be specified at Request Consumer configuration. This token consumer's role is to receive the token for signature verification. In this case, a token consumer for X.509 certificate token, X509TokenConsumer, must be specified. It is provided by the WebSphere Web services security run time as a default implementation.
- e. If a server requires a response that includes integrity information by the server, then the server also has to be configured to sign the response message at Response Generator configuration.

Confidentiality

In WebSphere Application Server V6.1, multiple and arbitrary parts of the message can be encrypted, for example, a message body, security token, and so on.

Confidentiality is the process by which a SOAP message is protected so that only authorized recipients can read it. Confidentiality is provided by XML encryption of the contents of the SOAP message. If the SOAP message is encrypted, only one who knows the key for confidentiality can decrypt and read the message.

Enabling confidentiality in your application

The following are the simplified steps to enable confidentiality:

1. On the client side, to specify confidentiality of part of a SOAP message, specify the part that must be encrypted and the manner of encryption in the client's WS-Security configuration.
 - a. Specify the parts of the message that must be encrypted at Request Generator configuration. The message parts can be specified by the predefined keyword or XPath expression. Also, you can specify multiple parts that require encryption.
 - b. Specify key-related information which includes the location of the client's key, type of key, and a password for protecting the key.
 - c. Specify encryption information that defines how to encrypt the specified part. Specify some options for encryption such as an encryption method algorithm and key-related information.
 - d. If a client expects a response that includes confidentiality by the server, then the client also has to be configured to decrypt the server's encryption of the response message at Response Consumer configuration.

2. On the server side, to specify the required confidentiality for part of a SOAP message, specify the part that must be encrypted and the way of decrypting the encryption in the server's WS-Security configuration.
 - a. Specify the parts of the message which require decryption at Request Consumer configuration. The message parts can be specified by the predefined keyword or XPath expression. Also, you can specify multiple parts that require a signature.
 - b. Specify key-related information, including the location of the server's key, a type of key, and a password for protecting the key.
 - c. Specify encryption information which defines how to decrypt the specified part. Specify some options for encryption such as an encryption method algorithm and key-related information.
 - d. A token consumer must be specified at Request Consumer configuration. This token consumer's role is to receive information for message decryption. In this case, a token consumer for X.509 certificate token, X509TokenConsumer, must be specified. It is provided by the WebSphere Web services security run time as a default implementation.
 - e. If a server requires a response that includes confidentiality by the server, then the server also has to be configured to encrypt the response message at Response Generator configuration.

15.2.5 Development of WS-Security

WebSphere Application Server V6.1 supports two development tools, which are Application Server Toolkit and Rational Application Developer. Developing Web services applications with WS-Security is made easier with the new Web Service security wizards provided in Application Server Toolkit V6.1.

To access the wizards, in the Project Explorer of the Java 2 Platform, Enterprise Edition (J2EE) perspective, expand **Web Services** → **Services** and right-click your service and select **Secure Web Service** (Figure 15-7 on page 443).

With the wizards, you can add a stand-alone security token, XML encryption or XML digital signature to a Web service. You may also clone the WS-Security settings from another Web service to your Web service.

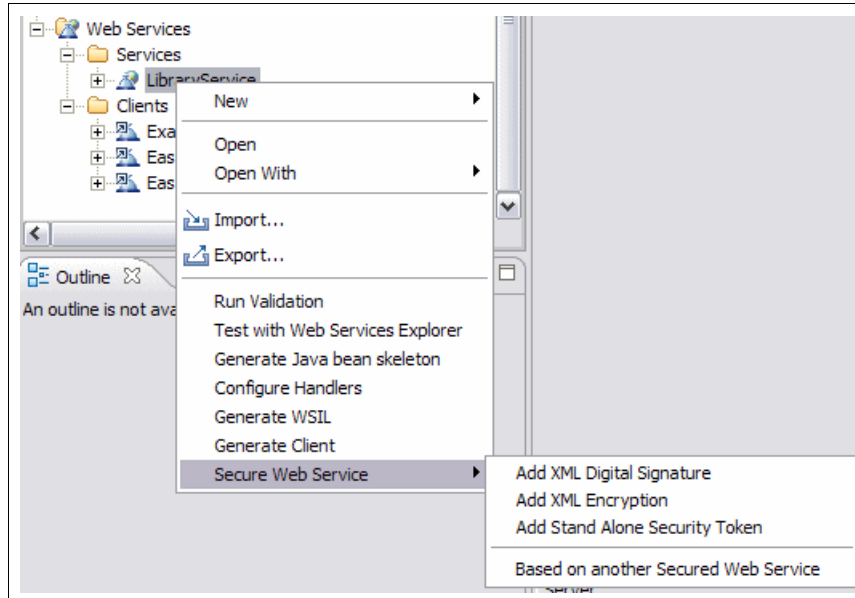


Figure 15-7 Secure Web Service wizards

Another set of wizards (Figure 15-8) is provided to configure WS-Security on your Web service client applications.

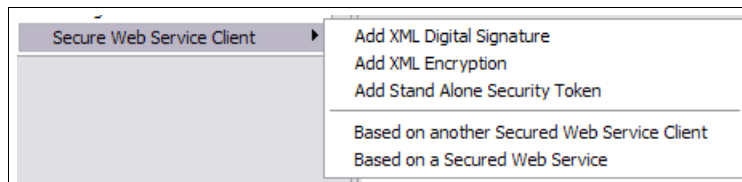


Figure 15-8 Secure Web Service client wizards

For more information about how to use the wizards, see the article *Securing Web services using Web services security wizards* in WebSphere Application Server V6.1 Information Center on the Web at the following address:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp>

However, if the Web Services security wizards do not satisfy your complex security requirement, you still have the option to manually configure WS-Security for your Web services applications. See article “Securing Web services manually based on WS-Security” in WebSphere Application Server V6.1 Information Center for details.

You can also find detailed information about how to define WS-Security configuration manually by using Rational Application Developer in “Securing Web Services” in *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461.

15.2.6 Hardware cryptographic device support for WS-Security

In WebSphere Application Server V6.1, you can configure hardware cryptographic device to be used for Web services security. You can use the hardware cryptographic device in two ways:

- ▶ Accelerate the cryptographic operations for WS-Security.
- ▶ Store the cryptographic keys so that they never leave the device.

Configuring a hardware cryptographic keystore

To enable the previous two functions, define a hardware cryptographic keystore:

1. In the Administrative Console, select **Security** → **SSL certificate and key management**.
2. Under **Related Items**, click **Key stores and certificates**.
3. Click **New**. New keystore page opens, as shown in Figure 15-9 on page 445.
4. Type a name for this hardware keystore.
5. Type the path for this hardware device-specific configuration file.

Note: Two required attributes in this configuration file are name and library. The following is an example:

```
name = SampleAccelerator
library=/opt/sample/lib/libpkcs11.so
```

The *IBMPKCS11Impl Provider Guide* at the following address describes the details of this configuration file:

<http://www-128.ibm.com/developerworks/java/jdk/security/50/secguides/pkcs11implDocs/IBMJavaPKCS11ImplementationProvider.html#ConfigFile>

6. Type and confirm the password. This is optional when the keystore is used purely as a cryptographic accelerator.
7. Select **Cryptographic Token Device (PKCS11)**.
8. Select **Read only**.
9. Optional: Select **Initialize at startup**.
10. Optional: Select **Enable cryptographic operations on hardware device** if this device is used as a cryptographic accelerator.
11. Click **OK**.
12. Click **Save** to save the configuration.

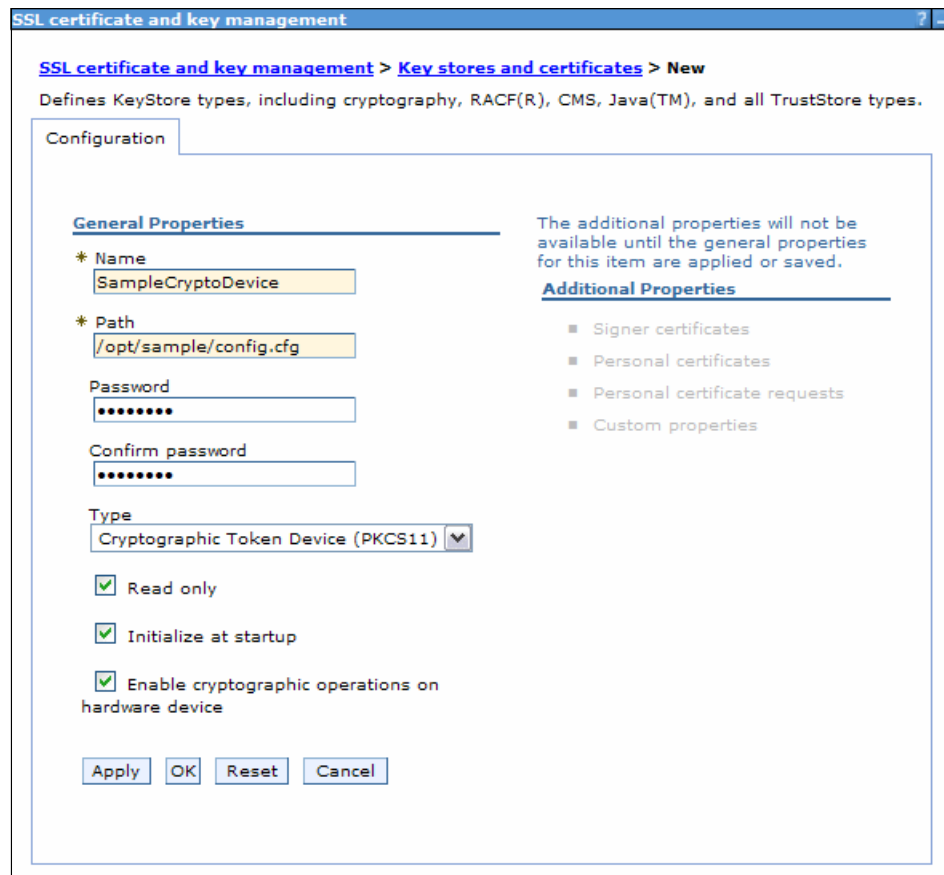


Figure 15-9 New hardware cryptographic keystore

Supported hardware cryptographic devices: For a list of supported hardware cryptographic devices, see the following Web address:

<http://www-128.ibm.com/developerworks/java/jdk/security/50/secguides/pkcs11implDocs/IBMPKCS11SupportList.html>

Enabling cryptographic operations on hardware device

You can use hardware device as a cryptographic accelerator for WS-Security. You can store the key for the cryptographic operations in a regular Java key store file. In this case the device is a pure cryptographic accelerator.

To enable cryptographic operations on hardware device:

1. In the Administrative Console, select **Servers** → **Application servers**.
2. Click the server name that you want to configure.
3. Under Security, click **Web services: Default bindings for Web services security**.
4. Under Cryptographic Hardware, select **Enable cryptographic operations on hardware device** and choose the hardware configuration name (defined in step 4 of “Configuring a hardware cryptographic keystore” on page 444). See Figure 15-10.



Figure 15-10 Cryptographic hardware configuration

5. Click **OK**.
6. Click **Save** to save the configuration.

Note: To further accelerate WS-Security processing for large scale service-oriented architecture (SOA) applications, consider WebSphere DataPower® SOA Appliances:

<http://www.ibm.com/software/integration/datapower/>

15.3 Transport-level security

HTTP, the most used Internet communication protocol, is currently also the most popular protocol for Web services. HTTP is an inherently insecure protocol, because all information is sent in clear text between unauthenticated peers over an insecure network. It belongs to the group of protocols, such as Simple Mail Transfer Protocol (SMTP), telnet, and File Transfer Protocol (FTP), that were designed in the earlier stages of the Internet, when security seemed not to be an issue, and eventually they are to be replaced by transfer protocols that allow authentication and encryption.

To secure HTTP, transport-level security can be applied. Transport-level security is a well-known and often used mechanism to secure HTTPS inter- and intranet communications. Transport-level security is based on SSL or Transport Layer Security (TLS) that runs beneath HTTP.

HTTPS allows client and server-side authentication through certificates, which have been either self-signed or signed by a certification agency. HTTPS can be assigned in any combination with any parts of message-level security (WS-Security).

Unlike message-level security, HTTPS encrypts the *entire* HTTP data packet. There is no option to apply security selectively only on certain parts of the message. SSL and TLS provide security features including authentication, data protection, and cryptographic token support for secure HTTP connections.

This book does not cover HTTPS in more detail. See the points in “More information” on page 449.

15.3.1 SOAP over HTTP transport-level security

Although HTTPS does not cover all aspects of a general security framework, it provides a security level regarding party identification and authentication, message integrity, and confidentiality. It does not provide authentication, auditing, and non-repudiation. To run HTTPS, the Web service port address must be in the form `https://`.

Even with the WS-Security specification, you must consider SSL when you think about Web services security. By using SSL, a so-called point-to-point security can be achieved. See Figure 15-11.

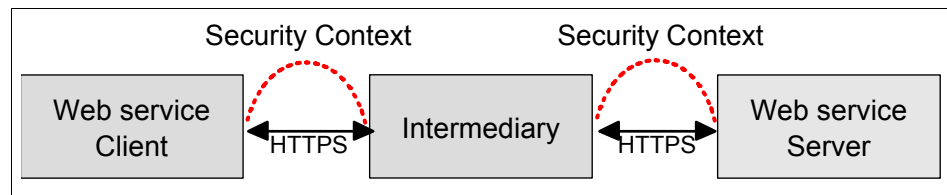


Figure 15-11 Point-to-point security with HTTPS

15.4 WS-I Basic Security Profile

Web Services Interoperability Organization (WS-I) is an open industry effort promoting Web services interoperability across vendors, platforms, programming languages, and applications. One of WS-I's major deliverable to date is WS-I Basic Profile, which provides implementation guidelines for how you can use the profiled Web services specifications together for best interoperability. WS-I Basic Profile V1.1 (BP1.1) is supported in WebSphere V6.0 and later.

WS-I Basic Security Profile (BSP) V1.0 is a Working Group Draft that consists of a set of non-proprietary Web services specifications that clarifies and amplifies those specifications to promote Web services security interoperability across different vendor implementations.

The Basic Security Profile is an extension to the Basic Profile. It describes how OASIS WS-Security specifications must be interpreted by adding constraints and clarifications with the intent to promote interoperability. The scope of BSP includes the following additional specifications:

- ▶ RFC 2818: HTTP over TLS
- ▶ RFC 2246: The Transport Layer Security Protocol V1.0
- ▶ The Secure Sockets Layer Protocol V3.0
- ▶ WS-Security: SOAP Message Security V1.0
- ▶ WS-I Basic Profile V1.0
- ▶ WS-I Basic Profile V1.1
- ▶ Simple SOAP Binding Profile V1.0
- ▶ XML-Signature Syntax and Processing
- ▶ XML Encryption Syntax and Processing

- ▶ WS-Security: UsernameToken Profile V1.0
- ▶ WS-Security: X.509 Certificate Token Profile
- ▶ RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile
- ▶ Information technology “Open Systems Interconnection” The Directory: Public-key and attribute certificate frameworks Technical Corrigendum 1
- ▶ WS-Security: Rights Expression Language (REL) Token Profile V1.0
- ▶ WS-Security: Kerberos Token Profile V1.1
- ▶ WS-Security: SAML Token Profile V1.0
- ▶ WS-I Attachments Profile Version 1.0
- ▶ WS-Security: SOAP Messages with Attachments (SwA) Profile V1.1

WebSphere Application Server V6.1 now supports applications to comply to the WS-I Basic Security Profile V1.0. It provides configuration options to ensure that you can enable the BSP recommendations and security considerations to ensure interoperability.

15.5 Summary

Web services technology enables a loosely coupled, language-neutral, platform-independent way of linking applications within organizations, across enterprises, and across the Internet. To achieve the target, however, it is essential for Web services to provide a sufficient level of security to support business transactions. Ensuring the integrity, confidentiality, and security of Web services through the application of a comprehensive security model is critical, both for organizations and their customers.

In WebSphere Application Server V6.1, Web services security can be applied at transport-level security and at message-level security. Highly secure client-server designs can be architected using these security levels.

15.6 More information

Because Web services security is a quickly evolving field, it is essential for developers and designers to regularly check for recent updates. This following list of Web addresses provides the most important entry points for your exploration:

- ▶ XML Signature Workgroup home page
<http://www.w3.org/Signature/>
- ▶ XML Encryption Workgroup home page
<http://www.w3.org/Encryption/>
- ▶ WS-Security specification 1.0
<http://www.ibm.com/developerworks/library/ws-secure/>
- ▶ The *Web services security Roadmap* white paper
<http://www.ibm.com/developerworks/webservices/library/ws-secmap/>
- ▶ OASIS WS-Security 1.0 and token profiles
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
- ▶ The WS-I Organization Web site, which includes profiles, sample application implementations, and compliance testing tools
<http://www.ws-i.org/>
- ▶ The WS-I Basic Profile V1.1 deliverable
<http://www.ws-i.org/Profiles/BasicProfile-1.1.html>
- ▶ The WS-I Basic Security Profile V1.0 deliverable
<http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html>
- ▶ Information about the IBM representation and contribution to the WS-I Organization
<http://www.ibm.com/developerworks/webservices/wsi/>
- ▶ IBM Tivoli Access Manager for Business Integration
<http://www.tivoli.com/products/index/access-mgr-bus-integration/>
- ▶ IBM WebSphere MQ standard proposition overview on developerWorks
<http://www.ibm.com/software/ts/mqseries/messaging/>
- ▶ Security information about IBM J2SE 5 SDKs, which are used by WebSphere Application Server V6.1
<http://www-128.ibm.com/developerworks/java/jdk/security/50/>

Several commercial and non-commercial information sources are available that cover more general subjects, such as SSL encoding and HTTPS protocol.



Securing access to WebSphere MQ

Java Messaging Service (JMS) is a Java application programming interface (API) that allows applications to create, send, receive, and read messages. As part of the Java 2 Platform, Enterprise Edition (J2EE) 1.4 Specification, WebSphere Application Server V6.1 supports the JMS 1.1 API. J2EE 1.4 platform has the following messaging features:

- ▶ Application clients, Enterprise JavaBeans (EJB) components, and Web components can send or synchronously receive a JMS message. Application clients can also receive JMS messages asynchronously.
Application clients should refer to the application running on the client side in the J2EE client container.
- ▶ The message-driven bean (MDB) enables the asynchronous consumption of messages on the server side.
- ▶ Sending or receiving messages can participate in distributed transactions.

The JMS specifications do not discuss the security and encryption of the message that is getting transferred using the JMS provider. Instead, specifications leave the security implementation to the JMS provider. This chapter discusses about WebSphere MQ and default messaging as JMS providers.

16.1 Application server and WebSphere MQ

If the default messaging provider does not meet your requirements, WebSphere Application Server can use WebSphere MQ as a JMS provider. Applications running on WebSphere Application Server V6.1 can access WebSphere MQ JMS resources through the JMS 1.1 interfaces.

This section discusses the different ways to integrate WebSphere Application Server V6.1 with WebSphere MQ in a secure way. For additional information, see IBM Redbooks *WebSphere MQ Version 6 and Web Services*, SG24-7115 and *WebSphere MQ Security in an Enterprise Environment*, SG24-6814.

16.1.1 WebSphere MQ messaging components

WebSphere applications can communicate with WebSphere MQ in a couple of ways in V6.1. WebSphere MQ can be connected to the service integration bus as a foreign bus and the service integration bus handles communication with WebSphere MQ, or applications can interact directly with WebSphere MQ. When connected to the service integration bus, the WebSphere MQ queue manager appears to be another messaging engine on a foreign bus and all communication is handled by using TCP/IP.

If WebSphere MQ is not linked to the service integration bus, the applications interact directly with the WebSphere MQ server by using inter-process communication if the WebSphere MQ server is local, or via TCP/IP. Regardless of which method is configured, the WebSphere application uses the same JMS API to interact with the JMS provider. Only the underlying implementation changes. Choosing the appropriate architecture depends on application and architectural requirements and are not discussed here.

More information: For more information about configuring Secure Sockets Layer (SSL) between WebSphere Application Server and WebSphere MQ on the JMS provider, review the article “IBM WebSphere Developer Technical Journal: Securing connections between WebSphere Application Server and WebSphere MQ -- Part 1” on developerWorks at the following address:

http://www-128.ibm.com/developerworks/websphere/techjournal/0601_ratnasinghe/0601_ratnasinghe.html

Direct communication with WebSphere MQ

Direct communication between WebSphere applications and IBM WebSphere MQ is handled via a pool of connection objects. The WebSphere MQ “unified”

JMS connection factory specifies how to connect to WebSphere MQ queue manager for both point-to-point and publish/subscribe messaging.

To define this type of JMS resources, click **Resources** → **JMS** → **JMS providers** and select WebSphere MQ messaging provider at the appropriate scope. Connection factories is available in the Additional Properties pane. Also available here are the “domain-specific” Queue connection factories and Topic connection factories. In this configuration, WebSphere Application Server does not handle any of the authentication or authorization tasks other than passing a set of credentials to the WebSphere MQ server. The credentials can be supplied by the application or the container where the request is made. Transport security (confidentiality) with SSL is specified on the connection factory. The WebSphere MQ server handles all of the security tasks.

Alternative: You can also configure the JMS resources by selecting **Resources** → **JMS** and then selecting the appropriate JMS resource type.

Integrating WebSphere MQ onto the service integration bus

Integrating WebSphere MQ onto the service integration bus follows the service-oriented architecture (SOA) model. The WebSphere MQ queue manager opens to the local service integration bus as a *Foreign Bus*. Communication parameters for the foreign bus are defined by an *MQ link* definition on the messaging engine used to communicate with WebSphere MQ.

Figure 16-1 on page 454 shows a WebSphere MQ queue manager connected to the service integration bus via an MQ link. Applications that must send messages to MQ connect directly to their local or remote messaging engine and put messages to a destination. Messages can be routed according to the JMS destination configuration. A foreign, or alias, destination can be configured from a security perspective. The messaging engine reroutes to the messaging engine that has the MQ link definition. Which then routes the messages to the target queue manager via the MQ link. As far as the application is concerned, the destination is local to the messaging engine or can be configured to send messages to the foreign destination.

Note: It is not possible to receive a message from a queue defined on a WebSphere MQ queue manager. Similarly, it is not possible for WebSphere MQ applications to receive messages from a queue defined on the bus.

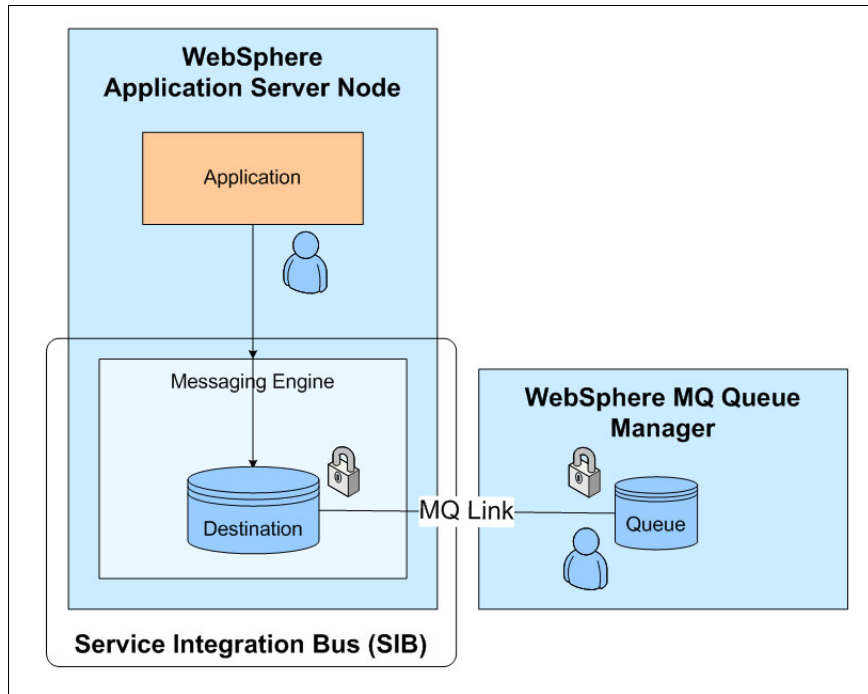


Figure 16-1 WebSphere MQ service integration bus Integration

The following sections discuss the three security points, which are authentication, authorization, and transport security and what roles WebSphere Application Server and WebSphere MQ play.

16.1.2 Authentication

This section describes authentication for WebSphere Application Server and WebSphere MQ.

WebSphere Application Server

When WebSphere MQ is integrated onto the service integration bus as a *Foreign Bus*, being displayed as another messaging engine to WebSphere, WebSphere Application Server can handle a portion of the authentication. To access the Foreign Bus, applications connect to the local messaging engine and then put messages to alias destinations or foreign destinations. To access any resources on the local bus, proper credentials must be supplied to the local messaging engine by either application or the application container if security is enabled for the Application Server and bus. These credentials are validated against the global user registry.

For more information about local service integration bus security, see 10.2, “An overview of service integration bus security” on page 250.

WebSphere MQ

When a JMS client, WebSphere Application Server in this instance, connects to WebSphere MQ, the credentials supplied are checked against the local operating system user registry where the WebSphere MQ server is installed.

Note: For meaningful security to exist between WebSphere Application Server and WebSphere MQ when TCP is used across a host, which is the most common scenario, a custom MQ security exit must be written or mutual SSL must be configured.

16.1.3 Authorization

This section describes authorization for WebSphere Application Server and WebSphere MQ.

WebSphere Application Server

Authorization to access service integration bus resources, including alias destinations, foreign destinations, and connecting to the bus has already been discussed in Authorization in 10.2, “An overview of service integration bus security” on page 250. Alias destinations and foreign destinations are used to access resources on the WebSphere MQ server, therefore, access to these local resources is required.

WebSphere MQ

Access to WebSphere MQ resources is required both in Service Integration Bus (SIB) to send to the foreign destination, and the *Object Authority Manager* (OAM) on the WebSphere MQ server. The OAM is automatically enabled for each queue manager. If authorization checking is not required the OAM may be disabled.

The OAM maintains an access control list (ACL) for each WebSphere MQ object it is controlling access to. On UNIX systems only group IDs can be displayed in an ACL. This means that all members of a group have the same authority. On Windows, both user IDs and group IDs can be displayed in an ACL. This means that authorities can be granted to individual users and also groups. The control command `setmqaut` grants and revokes authorities and is used to maintain the ACLs.

The following authorizations, among others, can be granted or revoked:

- ▶ get
- ▶ browse
- ▶ put
- ▶ connect

To connect to the queue manager the user must have connect authorization. After it is connected, when a request is made to a queue, process or namelist, the OAM checks the users authorization for that resource. For example, if a user wants to put a message on the queue, they would require “put” authorization for the queue. You can find all details about WebSphere MQ messaging security in the IBM WebSphere MQ V6.0 Security product documentation.

More information: For information about System Authorization Facility (SAF) on z/OS for WebSphere Application Server V6.1, visit the Web at:

<http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzas.doc/spladzos.htm>

16.1.4 Transport security

Communication between WebSphere Application Server and WebSphere MQ can be accomplished via TCP/IP, or if the queue manager and Application Server are on the same machine, then via inter-process communication. When communication via a foreign bus, MQ link is used, communication is always done via TCP/IP and the following two sub-sections apply.

If the applications communicate directly with the WebSphere MQ server, not over the service integration bus, then you can use either of the communication path. Transport security is not required when inter-process communication is used as transport of messages is done in memory. Transport level security, SSL parameters, for direct connections is defined on the JMS connection factories. The SSL settings on the WebSphere side and WebSphere MQ side must match.

WebSphere Application Server

When defining the MQ link between a WebSphere Application Server messaging engine and the WebSphere MQ foreign bus, two WebSphere MQ style channels, a sender and receiver, are defined in the local messaging engine. These two channels handle all communication between the messaging engine and the queue manager.

The sender channel, as the name implies, is used when messages are sent from the Application Server to WebSphere MQ. The sender channel has an option for *Transport chain* which determines if SSL is used when transporting messages to WebSphere MQ. The two default options for Transport chain are *OutboundBasicMQLink* and *OutboundSecureMQLink*. *OutboundSecureMQLink* uses SSL when connecting to WebSphere MQ.

When WebSphere MQ has to route a message to the default messaging engine on the Application Server, a connection is opened on the receiver channel. The port, or ports, which the local messaging engine accepts MQ link inbound requests on, are defined by the available transports in *WebSphere MQ link inbound transports* from the Application Server properties page. By default two transports are defined, *InboundBasicMQLink* and *InboundSecureMQLink*. *InboundSecureMQLink* is SSL enabled while *InboundBasicMQLink* is not. If SSL is required then disable the *InboundBasicMQLink* and restart the Application Server or node where the messaging engine is. Additional transports may be defined as required.

Notes:

- ▶ The default *InboundBasicMQLink* port is 5558. The default *InboundSecureMQLink* port is 5578.
- ▶ When communication with WebSphere MQ is done across the service integration bus, the name of the Sender channel defined on the MQ link must match the name of a receiver channel on the WebSphere MQ queue manager. The same is true for the sender channel defined on the WebSphere MQ queue manager matching the name of the receiver channel on the MQ link.

Important: Use special consideration when using lowercase letters and defining the channels on WebSphere MQ. Do *not* use lowercase letters on channel names.

WebSphere MQ

Clients and queue managers communicate with WebSphere MQ over channels defined on the WebSphere MQ server. When WebSphere Application Server applications use direct JMS connections to a WebSphere MQ server, not over the service integration bus, the applications are displayed as clients. When communication is done via the service integration bus, the MQ link is displayed as a queue manager to WebSphere MQ. Regardless of what method is used, communication is done over one or more channels. SSL properties on the channel allow for selection of which Cipher Spec to use and which clients, based on Distinguished Name (DN), to accept connections from. Enabling SSL on the channel is as simple as selecting the Cipher Spec and restarting the channel.

For more information about configuring SSL between WebSphere Application Server and WebSphere MQ on the JMS provider, review the article “IBM WebSphere Developer Technical Journal: Securing connections between WebSphere Application Server and WebSphere MQ -- Part 1” on developerWorks at the following address:

http://www-128.ibm.com/developerworks/websphere/techjournal/0601_ratnasinghe/0601_ratnasinghe.html

16.1.5 Administering foreign service integration bus security

Security on the service integration bus and resources within the service integration bus are checked if both administrative security and bus security are enabled. Access to foreign destinations defined on the local service integration bus first requires access to the local service integration bus. The commands to modify the Bus Connector role (the role to connect to the service integration bus) have already been detailed in 10.3, “Administering service integration bus security” on page 253. After the application connects to the local service integration bus, additional checks are made to verify sender rights on the destination and foreign bus objects.

You can use the following commands to modify the Sender role for a foreign bus:

- ▶ List users in role for a foreign bus:

```
$AdminTask listUsersInForeignBusRole {-bus busName -foreignBus foreignBusName -role rolename}
```

- ▶ List groups in role for a foreign bus:

```
$AdminTask listGroupsInForeignBusRole {-bus busName -foreignBus foreignBusName -role rolename}
```

- ▶ Add a user to a role for a foreign bus:

```
$AdminTask addUserToForeignBusRole {-bus busName -foreignBus foreignBusName -role rolename -user username}
```

- ▶ Add a group to a role for a foreign bus:

```
$AdminTask addGroupToForeignBusRole {-bus busName -foreignBus foreignBusName -role rolename -group groupname}
```

- ▶ Remove a user from a role for a foreign bus:

```
$AdminTask removeUserFromForeignBusRole {-bus busName -foreignBus foreignBusName -role rolename -user username}
```

- ▶ Remove a group from a role for a foreign bus:

```
$AdminTask removeGroupFromForeignBusRole {-bus busName -foreignBus foreignBusName -role rolename -group groupname}
```

You can use the following commands to modify the Sender role for a foreign destination:

- ▶ List users in a destination role for a foreign bus:

```
$AdminTask listUsersInDestinationRole {-type destinationType -bus
busName -foreignBus foreignBusName -destination destinationName
-role roleName}
```

- ▶ List groups in a destination role for a foreign bus:

```
$AdminTask listGroupsInDestinationRole {-type destinationType -bus
busName -foreignBus foreignBusName -destination destinationName
-role roleName}
```

- ▶ Add a user to a destination role for a foreign bus:

```
$AdminTask addUserToDestinationRole {-type destinationType -bus
busName -foreignBus foreignBusName -destination destinationName
-role roleName -user userName}
```

- ▶ Add a group to a destination role for a foreign bus:

```
$AdminTask addGroupToDestinationRole {-type destinationType -bus
busName -foreignBus foreignBusName -destination destinationName
-role roleName -group groupName}
```

- ▶ Remove a user from a destination role for a foreign bus:

```
$AdminTask removeUserFromDestinationRole {-type destinationType -bus
busName -foreignBus foreignBusName -destination destinationName
-role roleName -user userName}
```

- ▶ Remove a group from a destination role for a foreign bus:

```
$AdminTask removeGroupFromDestinationRole {-type destinationType
-bus busName --foreignBus foreignBusName destination destinationName
-role roleName -user userName}
```

16.1.6 Administering WebSphere MQ security

Access to WebSphere MQ objects is controlled by the OAM. The **setmqaut** command is used to grant and revoke authorities and maintain the ACL contained in the OAM. Only a user with administration authority can run this command. Running the setmqaut command without parameters displays the command usage as follows:

```
setmqaut -m QMgrName [-n ObjName] -t ObjType [-p Principal | -g
Group] [-s ServiceName] Authorizations
```

The following authorizations, among others, can be granted or revoked:

- ▶ get
- ▶ browse
- ▶ put
- ▶ connect

Notes:

- ▶ Prepend the authorization with + to grant and - to revoke. For example, use +connect to grant connect authorization and -get to revoke get authorization.

Multiple authorizations can be specified at one time by separating them with a space. For example, +browse -get - put

- ▶ In a UNIX environment, authorizations can only be granted on groups.

For example, to grant browse rights to user janedoe on a queue name default and queue manager named QM_k1chm8p the following command is executed:

```
C:\PROGRA~1>setmqaut -m QM_k1chm8p -n default -t queue -p janedoe  
+browse
```

The result is as follows:

The setmqaut command completed successfully.

You can use the **dspmqaut** command to display authorizations from the OAM. For in depth details on administering WebSphere MQ security see the IBM WebSphere MQ V6.0 Security product documentation.

16.2 Sample application

For more information about configuring messaging for applications, see “Configuring the service integration bus and default messaging provider” on page 516.

Integrating WebSphere MQ onto service integration bus via MQ link is described in “Configuring WebSphere MQ as a foreign bus” on page 526.

You can find the details about a messaging sample application in “Sample application for messaging” on page 530.

16.3 Additional information

Information about the JMS specification is available at the following address:

<http://java.sun.com/products/jms/index.jsp>

For security information about WebSphere MQ, see *WebSphere MQ Security Version 6.0*, SC34-6588-01, in the WebSphere MQ Information Center at the following address:

<http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzas.doc/csqzas03.htm>



J2EE Connector security

This chapter discusses Java 2 Platform, Enterprise Edition (J2EE) Connector security with WebSphere Application Server V6.1.

17.1 The J2EE Connector Architecture

The J2EE Connector Architecture (JCA) specifies a standard architecture that allows J2EE applications to connect to enterprise information systems (EISs). The architecture defines a set of contracts that EIS vendors and Application Server vendors code to. These contracts specify standard application programming interfaces (APIs) to which the resource adapters must adhere and services that the system must provide to support the resource adapter. These system level services include transaction management, connection management, and security management, among others. EIS vendors coding their resource adapters to the connector API take advantage of the system level services on the J2EE server on which they are running. Applications are then free to take advantage of the resource adapter or connector.

After it is coded to the API, the resource adapter is plugged into an Application Server that supports the J2EE Connector Architecture. WebSphere Application Server V6.1 supports the J2EE Connector Architecture Version 1.5 specification. After the resource adapter is installed, applications running on the Application Server can access the EIS without having to handle the system level services. These are called outbound calls. Applications use the Common Client Interface (CCI) for EIS access. This interface provides a common API through which applications can access EISs.

Starting with the Version 1.5 specification, the EIS application can make calls into the Application Server to access application components and perform tasks. Calls initiated by EIS to WebSphere Application Server are called inbound calls.

The specification is beyond the scope of this book, but the architecture defines a standard set of APIs and services for scalable, transaction-oriented, secure connections to back-end EIS, such as enterprise resource planning (ERP) systems, databases, transaction processing, and messaging systems.

Figure 17-1 shows a high-level representation of the J2EE Connector Architecture.

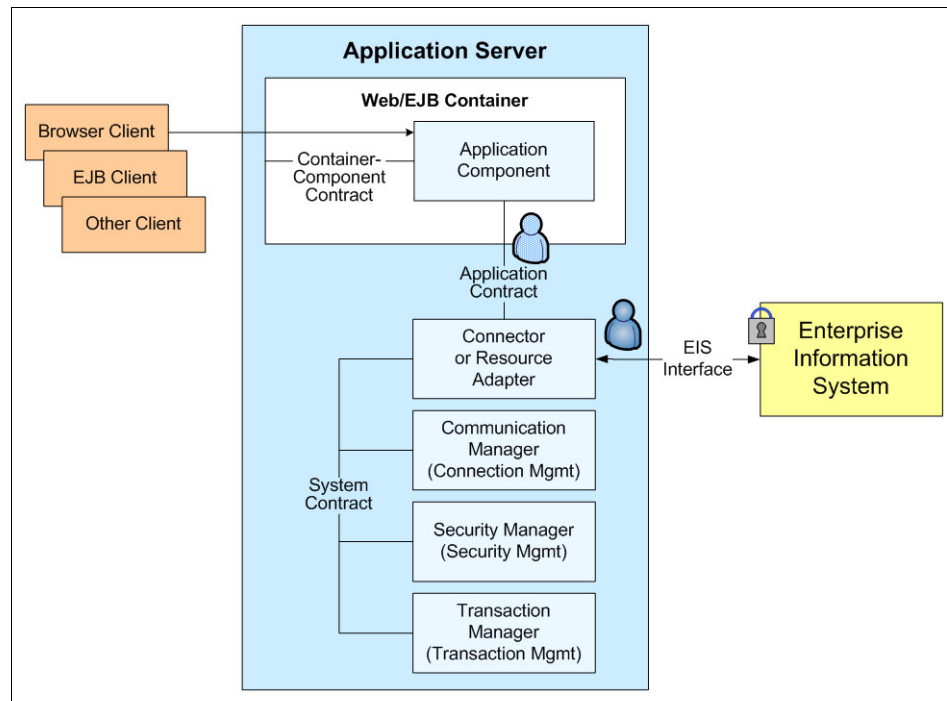


Figure 17-1 J2EE Connector Architecture

You can download the full JCA from Sun Microsystems at the following address:

<http://java.sun.com/j2ee/connector/>

17.1.1 Connector security architecture

The J2EE 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 mechanism as the commonly-supported authentication mechanism:

- ▶ Basic Password: Basic user-password-based authentication mechanism that is specific to an EIS
- ▶ Kerb V5: Kerberos Version 5-based authentication mechanism

Applications define whether to use application-managed sign-on or container-managed sign-on in the resource-ref elements in the deployment descriptor. Each resource-ref element describes a single connection factory reference binding.

The res-auth element in a resource-ref element, whose value is either Application or Container, indicates whether the enterprise bean code performs a sign-on or WebSphere Application Server can sign-on to the resource manager using the principal mapping configuration. The resource-ref element is typically defined at application assembly time with an assembly tool. The resource-ref can also be defined, or redefined, at deployment time.

17.2 Securing the J2EE Connector

By their nature, EIS resources generally require a high level of security to protect the information they contain from unauthorized access. In general, this means that to connect to an EIS, the user must supply a proper set of credentials, usually a user ID and password, before gaining access to the EIS back end. When accessing an EIS back end, the request for access flows from the application to the resource adapter and then to the EIS. In a secure environment, when the request is made to the adapter, the proper credentials are sent to the adapter when a connection is requested. The adapter then uses the credentials to connect to the EIS and performs the requested actions.

J2EE Connectors follow the J2EE security model. Access to EIS resources takes place under the security context of a resource principal. Where the security context comes from depends on the security model in use at the time of access to the resource adapter. The two security models are component-managed authentication (also called application-managed authentication) and container-managed authentication. Each is discussed in the following sections.

Note: You can also find the component-managed authentication referred to as Per Connection Factory. Mostly you see this naming in the Rational Application Developer.

17.2.1 Component-managed authentication

In the case of component-managed authentication, the application component accessing the resource or adapter is responsible for programmatically supplying the credentials, or WebSphere Application Server can supply a default component-managed authentication alias, if available.

After obtaining the connection factory for the resource from Java Naming and Directory Interface (JNDI), the application component creates a connection to the resource using the create method on the connection factory supplying the credentials. If no credentials are supplied when creating a connection and a component-managed authentication alias has been specified on the Java 2 Connector (J2C) connection factory, the credentials from the authentication alias are used. Assuming the credentials are valid, future requests that use the same connection use the same credentials.

Note: Component-managed authentication is specified by setting the `res-auth` entry in the deployment descriptor for the resource reference to the application.

Creating a sample EIS resource adapter is beyond the scope of this book. You can find sample code for looking up a resource adapter connection factory and connecting to the resource in Example 17-1. The code assumes that a Resource Reference has been defined and named `EISResourceName` and maps to a J2EE Resource Adapter connection factory.

The basic steps are as follows:

1. Get the initial JNDI context.
2. Look up the connection factory for the resource adapter.
3. Create a `ConnectionSpec` object holding credentials.
4. Obtain the Connection Object from the Connection Factory by supplying the `ConnectionSpec` object.

After a connection is obtained by using the credentials specified in the `ConnectionSpec` object, all future interactions, through interaction objects, carry the user credentials. The EIS fulfills the request or denies it based on the Authorization properties in EIS.

Example 17-1 Get resource connection

```
try
{
    Context ic = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory)
ic.lookup("java:comp/env/EISResourceName");
    try {
        //Use the following if res-auth=Application
        //This is for Component Managed Authentication with
        //no JAAS Authentication Alias set on the Connection Factory
        WSConnectionSpecImpl conSpec = new WSConnectionSpecImpl();
        conSpec.setUserName("username"); // replace the username with the
value
```

```

        conSpec.setPassword("password"); // replace the password with the
value
        Connection con = cf.getConnection(conSpec);
        //Use the following if res-auth=Container
        //This is for Container Managed Authentication
        //Connection con = cf.getConnection();
    } catch (ResourceException re) {
        System.out.println(re.toString());
    }
}
catch(NamingException ne) {
    System.out.println(ne.toString());
}
}

```

Security of lookups with component-managed authentication

External Java clients (stand-alone clients or server from other cells) with JNDI can look up a J2C resource such as a data source or Java Messaging Service (JMS) queue. However, they are not permitted to take advantage of the component-managed authentication alias defined on the resource. This alias is a default value used when the user and password are not supplied on the `getConnection()` call. Therefore, if an external client requires to get a connection, it must assume responsibility for the authentication by passing it through arguments on the `getConnection()` call.

J2C authentication: J2C authentication alias is per cell. An enterprise bean or servlet in one application server cannot look up a resource in another server process that is in a different cell, because the alias is not resolved.

17.2.2 Container-managed authentication

Container-managed authentication removes the requirement of the component to programmatically supply the credentials for accessing the EIS. Instead of calling the `getConnection()` method with a `ConnectionSpec` object, `getConnection()` is called with no arguments. See Example 17-1 for sample code.

The authentication credentials used for connecting to the EIS are then supplied by the Web container, the application container, or the Enterprise JavaBeans (EJB) container, depending on where the resource is accessed from. WebSphere Application Server V6.1 supports the Java Authentication and Authorization Service (JAAS) specification. Therefore, you can map the credentials for accessing the EIS from any of the configured JAAS Authentication login modules, including any custom JAAS Authentication login module.

When defining the Resource Reference in the deployment descriptor, either Web application deployment descriptor or EJB deployment descriptor, after the Authentication is set to Container and the WebSphere Bindings JNDI Name has been entered, three options become available for the JAAS Login Configuration. The three options are explained in the following sections.

The Container-managed authentication (deprecated) setting

This option uses the container-managed authentication settings that are defined for the resource's connection factory. The credentials can come from a JAAS Authentication Alias when using the DefaultPrincipalMapping Mapping-configuration alias setting, or be mapped from another JAAS Authentication login module.

Note: Selecting this option and specifying DefaultPrincipalMapping and selecting a JAAS Authentication Alias when defining the resource's Connection Factory provides the same functionality as WebSphere Application Server V5.

We no longer recommend this method. We recommend you to use the Use Default Method option, which is discussed next.

The Use Default Method setting

The Use Default Method setting behaves similarly to container-managed authentication using the DefaultPrincipalMapping option. A JAAS authentication alias is linked to the Connection Factory and all container-managed authentication requests using the resource reference use the credentials from the alias when connecting to the EIS. The difference is that the linking from the JAAS authentication alias to Connection Factory is done at the resource reference level within the application. This alleviates a security exposure by limiting the scope of the credentials to the application defining the resource reference. All other applications must supply their own credentials when accessing the Connection Factory directly from JNDI. We recommend this method for mapping JAAS authentication aliases to Connection Factories.

The Use Custom Login Configuration setting

This option allows you to use any defined JAAS authentication module. Enter the name of the JAAS authentication modules as it is defined in **Security → Secure administration, applications, and infrastructure → Java Authentication and Authorization Service → Application logins** and specify any parameters required. When a connection to the resource is required, the specified module is used to obtain the credentials that are passed to the connector.

17.3 JCA authentication mechanism

When bus security is enabled and before users are allowed to connect to a bus, their credentials must first be authenticated. Authentication uses the same user registry as the other parts of WebSphere Application Server. When an authentication failure occurs, a `JMSsecurityException` is thrown. See Chapter 2, “Configuring the user registry” on page 7 for further information about user registry.

You can also use the Java Authorization Container Contract (JACC) provider for Tivoli Access Manager to manage authentication to EISs such as databases, transaction processing systems, and message queue systems which come under WebSphere Application Server Security domain. Global single sign-on (SSO) principal mapper JAAS login module is used to achieve the authentication.

With GSO principal mapping, a special purpose JAAS login module inserts a credential into the subject header. This credential is used by the resource adapter to authenticate to the EIS. The JAAS login module used is configured on a per connection factory basis. The default principal mapping module retrieves the user name and password information from Extensible Markup Language (XML) configuration files. The JACC provider for Tivoli Access Manager bypasses the credential that is stored in the XML configuration files and uses the Tivoli Access Manager global sign-on (GSO) database instead, to provide the authentication information for the EIS security domain.

WebSphere Application Server provides a default principal mapping module that associates user credential information with EIS resources. The default mapping module is defined in the WebSphere Application Server administrative console on the Application login panel. To access the panel, click **Security** → **Secure administration, applications and infrastructure**. Under Java Authentication and Authorization Service, click **Application login**. The mapping module name is `DefaultPrincipalMapping`.

The EIS security domain user ID and password are defined under each connection factory by an `authDataAlias` attribute. The `authDataAlias` attribute does not contain the user name and password. This attribute contains an alias that refers to a user name and password pair that is defined elsewhere.

The Tivoli Access Manager Principal mapping module uses the `authDataAlias` attribute to determine the GSO resource name and the user name that is required to perform the lookup on the Tivoli Access Manager GSO database. The Tivoli Access Manager Policy Server retrieve the GSO data from the user registry. Tivoli Access Manager stores authentication information on the Tivoli Access Manager GSO database against a resource and user name pair.

17.3.1 Role-based authorization

Messaging security uses a simple role model in which a role contains the authorization permission required to perform a given operation. If messaging security is switched on, you must give permission to any users, who connect to a bus, to carry out the operations that they must perform. You do this by assigning them to the appropriate role or roles.

You can assign a user or group to the following types of role:

- ▶ Connector, which contains permission to connect to the local bus.
- ▶ Sender, which contains permission to send (produce) a message to the destination.
- ▶ Receiver, which contains permission to receive (consume) a message from the destination.
- ▶ Browser, which contains permission to browse message on the destination.
- ▶ Creator, which contains permission to create a temporary destination based on this temporary destination prefix. This only applies to prefix destinations.
- ▶ Identity Adopter, which contains permission to send a message using a different user identity.

When messaging security is switched on, all operations on the following objects require authorization:

- ▶ Buses
- ▶ Destinations
- ▶ Topic spaces and Topics

17.3.2 Topic security

When messaging security is switched on, users must be authorized to access topics. Topics are contained in a topic space, which is one of the types of destination.

17.3.3 Messaging security

Messaging security applies to the whole bus. You cannot switch security on for some messaging engines in a bus and off for the others.

When you create a connection to the messaging system, you can specify a user name and password. The user name and password are authenticated using the same user registry that the Application Server uses for its authentication check. To ensure confidentiality and integrity of messages in transit, you can configure a Secure Sockets Layer (SSL) secure transport for the connections between

clients and messaging engines, between messaging engines in the same bus, and between buses. You can configure a bus so that all its connections use a secure transport.

Creating a bus when administrative security is enabled results in a bus that is secure by default. If administrative security is disabled, an insecure bus is created.

17.3.4 Enable bus security

Select this option to inherit the secure administration setting of the cell. Deselect this option if you always wish to disable bus security. Creating a bus when administrative security is enabled results in a bus that is secure, by default. If administrative security is disabled an insecure bus is created.

17.3.5 Inter-engine authentication alias

This is the name of the authentication alias used to authorize communication between messaging engines on the bus. This field can be left blank. If a value is specified, and the bus security is enabled, incoming connections from other messaging engines are controlled to prevent unauthorized messaging engines from establishing a connection.

Permitted transport

You can use the permitted transports group of radio buttons to specify what transports must be used. There are three modes. The first allows the use of any messaging transport chain defined to any bus member. The second allows the use of only messaging transport chains that are protected by an SSL chain. The third only allows the transports in the specified list. The Permitted transport link under Related Items allows you to add and remove permitted transports.

Mediations authentication alias

This is the name of the authentication alias used to authorize the bus to access mediations.

Foreign bus

A foreign bus is a representation of another service integration bus, or a WebSphere MQ queue manager, with which an existing service integration bus, or a WebSphere MQ queue manager, can exchange messages. The purpose of a foreign bus is to extend the network of buses that can exchange messages.

Inbound communication

When a server is created using the default template, transport chains, explained in the following sections, are automatically created to facilitate communication with messaging engines that are hosted by the Application Server.

InboundBasicMessaging

InboundBasicMessaging allows communication using the Transmission Control Protocol (TCP). The default port used by this chain for the first server on the node is 7276. You must verify that the selected port is not already used, for example, if you are configuring a second server with the same name as the first server. Messaging engines hosted in other Application Servers and JMS applications running in a client container can communicate with the messaging engines of the server using this transport chain.

InboundSecureMessaging

InboundSecureMessaging provides secure communication using the SSL-based encryption protocol over a TCP network. The default port used by this chain for the first server on the node is 7286. You must verify that the selected port is not already used, for example, if you are configuring a second server with the same name as the first server. The SSL configuration information for this chain is based on the default SSL repertoire for the Application Server. Messaging engines hosted in other Application Servers and JMS applications running in the client container can communicate using this transport chain.

InboundBasicMQLink

InboundBasicMQLink supports WebSphere MQ queue manager sender channels and applications using the WebSphere MQ JMS provider connecting over a TCP network. The default port used by this chain is 5558, although this can be automatically adjusted to avoid conflicts.

InboundSecureMQLink

InboundSecureMQLink enables WebSphere MQ queue manager sender channels and applications using the WebSphere MQ JMS provider to establish SSL based encrypted connections over a TCP network. The default port used by this chain is 5578, although this is automatically adjusted to avoid conflicts.

You can manage all these chains through the administrative console if you select either **Servers** → **Application server** → **server_name** → **Messaging engine inbound transports** or **Servers** → **Application servers** → **server_name** → **WebSphere MQ link inbound transports**.

17.4 Mediations security

When WebSphere Application Server security is enabled, the messaging engine must be authorized to access the mediation. To specify a mediation authentication alias for the messaging engine, by using the administrative console:

1. In the navigation panel, click **Service Integration** → **Buses**.
2. From the list of buses that is displayed in the content panel, select the name of the bus where the mediation is defined. The properties of the bus are displayed.
3. In the Mediations authentication alias field, select the authentication alias that you want to use to access the bus.
4. Click **OK**.
5. Save your changes to the master configuration.

When an application sends a message to the bus, the identity of the sender application is associated with the message. The message is sent to the next destination in the forward routing path only if the message originator has *Sender* authority for that destination. A mediation can change the identity of the senders to the mediations identity.

When you install a mediation for use when security is enabled, you must ensure that the identity that the messaging engine uses to call mediations can access the mediations. If bus security is enabled, and the mediation sends messages to and receives messages from destinations, the mediation identity requires access to the destination.

17.5 Transport security in service integration bus

The use of permitted transports requires all members of the bus to be at the WebSphere Application Server V6.1 or later version. Configuring with the use of administrative console helps to prevent any error. However, if a bus is configured to use permitted transport and has a previous level bus member, the runtime operation ignores the setting and issues a warning.

The transport security functionality addresses the following issues:

- ▶ If you want to prevent the use of specific protocols to attach to the bus, the administrator must go around each server in the bus, disabling the relevant chains.
- ▶ Adding a new server as a bus member requires the administrator to disable channels on that server.
- ▶ Two different buses that have different requirements on the chain being used cannot share a server.

You can enforce transport encryption with a bus configuration setting on a bus with V6.1 or later bus members. This setting is independent of the bus security setting, therefore chains can be locked down without enabling bus security. The use of permitted chains has three settings:

- ▶ All defined messaging transport chains
- ▶ Any messaging transport chain that contains the SSL channel
- ▶ Only messaging transport chains in a specific list

When the use of permitted chains is enabled, the bus only allows access to the permitted chains. When the bus is set to be secure, by default you can only have chains that use the SSL channel. If the bus is not set to be secure, the default setting allows access for all chains.

When the use of permitted chains is enforced and a protocol is not specified for inter-bus communications then `InboundSecureMessaging` is assumed instead of `InboundBasicMessaging`. You can avoid this assumption if you set the protocol attribute in the bus configuration. If `InboundSecureMessaging` is not a permitted chain then an error occurs.

To configure the list of permitted chains:

1. Open the administrative console, and click **Service Integration Bus Security**.
2. On the content panel, click the name of the bus that you wish to configure the users and groups for. When a new page opens, click **Security** link under Additional Properties.
3. Click **Security selection** link, which is either Enabled or Disabled
4. On the Security Configuration panel, click **Permitted Transport** link under Additional Properties.
5. Click the **New** button on the Permitted Transport Panel.
6. This new panel allows a transport to be added to the list of permitted channel chains. The selection list contains the unique messaging chain names defined to servers that are members of the bus. Click **OK** to confirm selection.

17.5.1 Destination security

When messaging security is switched on, users must have permission to access destinations, including temporary destinations. If a message has to be routed from one destination to one or more other destinations, then the user must have permission to access all the destinations concerned.

To allow a user access to a destination, you must give them the required authorization permissions by assigning them to the appropriate roles, depending on what activity they must perform. Role assignments for a destination are defined on the bus that owns the destination and all messaging engines on the bus have access to them.

The following roles are available for destinations:

- ▶ Sender
- ▶ Receiver
- ▶ Browser
- ▶ Creator (For Temporary destinations only)

17.6 Securing Web services by using service integration technologies

Service integration technologies provide a range of facilities for secure communication between the service requester and the service integration bus (SIB), and between the service integration bus and the destination service. By default, the service integration bus for Web Services configuration works when WebSphere Application Server security is enabled and your service integration buses are secured.

However, this level of security does not impose any security restrictions on the users of your service integration bus/Web services configuration. To control how your service integration bus/Web services configuration is used by each group of your colleagues or customers, use the service integration bus/Web services additional security features to enable working with password-protected components and servers, with WS-Security and with Hypertext Transfer Protocol Secure (HTTPS).

To configure these facilities:

1. Configure secure transmission of SOAP messages using WS-Security.
2. Create user ID and password authentication and authorization for inbound and outbound services.
3. Invoke outbound services over HTTPS.

See the WebSphere Application Server V6.1 Information Center for more information.

17.7 Additional information

For additional information about the JCA specification, see the following Web address:

<http://java.sun.com/j2ee/connector>



Securing the database connection

Database security is typically broken down into two areas. First, securing the connection between the client and server, and secondly, access control. Securing the connection generally is concerned with using encryption between the database client and server so that others on the network cannot sniff data as it flows across the network. The encryption can be using Secure Sockets Layer (SSL) between client and server, or some other proprietary encryption scheme.

18.1 Securing the connection

The IBM DB2 Universal Database allows for the specification of one or more authentication types. These authentication types specify how and where the authentication of the user is verified and what type of encryption is required.

Table 18-1 shows the authentication types that are allowed between DB2 Universal Database V8.2 clients and servers, along with the encryption that is performed. See the “Authentication methods for your server” topic in the IBM DB2 Information Center for further information about the authentication types and how to set them.

Table 18-1 DB2 V8.2 authentication types

Authentication Type	Where Specified	Encryption
SERVER	Client or Server	None. User ID, password, and data sent unencrypted.
SERVER_ENCRYPT	Client or Server	Encrypted user ID and encrypted password are sent to server for authentication.
CLIENT	Client or Server	None. User ID and password are authenticated on the client node only.
DATA_ENCRYPT	Client or Server	Same authentication as SERVER_ENCRYPT in addition to the encryption of user data: <ul style="list-style-type: none"> ▶ Structured Query Language (SQL) statements ▶ SQL program variable data ▶ Output data from the server processing of an SQL statement ▶ Answer set data resulting from a query ▶ Large object (LOB) data streaming ▶ SQLDA descriptors
DATA_ENCRYPT_CMP	Server	Same as DATA_ENCRYPT except for clients that do not support it, they fall back to SERVER_ENCRYPT.
KERBEROS	Client or Server	Authentication is done using Kerberos security system.
KRB_SERVER_ENCRYPT	Server	Same as KERBEROS except for clients that do not support it, they fall back to SERVER_ENCRYPT.
GSSPLUGIN	Client or Server	Authentication is done using GSS-API plug-in.
GSS_SERVER_ENCRYPT	Server	Same as GSSPLUGIN except for clients that do not support it, they fall back to SERVER_ENCRYPT.

Note: When explicitly cataloging a database with a specific authentication type, ensure that the server is set to handle that encryption type. For example, if the client catalogs the database by using KERBEROS and the server is set to SERVER_ENCRYPT, an SQL error results and the connection does not complete.

IBM currently ships two Java Database Connectivity (JDBC) data source provider implementations with DB2 Universal Database V8.2.:

- ▶ DB2 Legacy CLI-based Type 2 JDBC Driver Provider
- ▶ DB2 Universal JDBC Driver Provider

The Universal provider is both a JDBC type 2 and JDBC type 4 driver, while the Legacy provider is a type 2 driver only.

Note: Support for the DB2 Legacy CLI-based Type 2 JDBC Driver provider is deprecated in WebSphere Application Server V6.1. The recommended JDBC driver for DB2 is the Universal JDBC Driver.

To use the Universal driver:

1. Install or copy the db2jcc Java archive (JAR) files to WebSphere Application Server.
2. Set the WebSphere variable DB2UNIVERSAL_JDBC_DRIVER_PATH to point to the directory containing the JAR files.

Encryption of the JDBC connection to the DB2 server depends on which type of JDBC driver is used. Figure 18-1 on page 482 illustrates the communication paths of the type 2 and type 4 drivers.

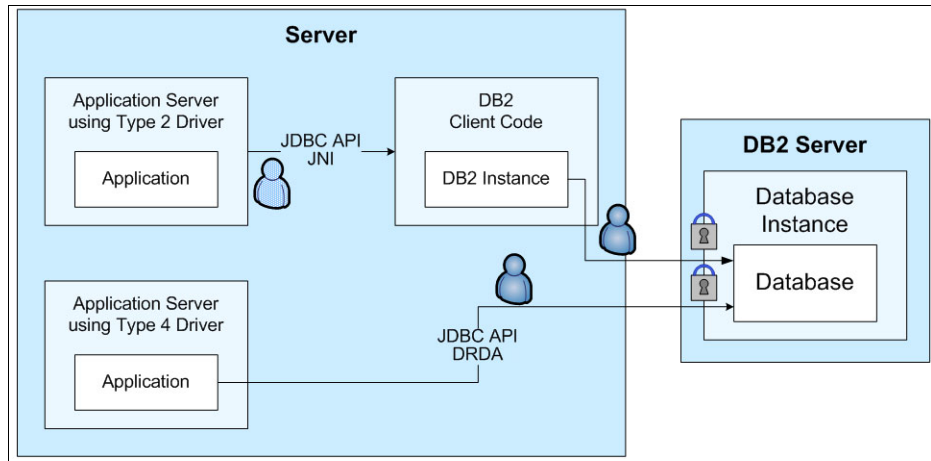


Figure 18-1 IBM DB2 Universal Database JDBC drivers

The security implications of each type of driver is discussed in the following sections.

18.1.1 JDBC type 2 driver

The type 2 driver uses the locally installed native DB2 client libraries. By making Java Native Interface (JNI) calls through the local DB2 instance, as shown in Figure 18-1, the Java application can access databases that have been cataloged on the system. Security over the JNI calls is not required. Security between the DB2 client instance and the DB2 server is determined by the security settings specified during cataloging of the database at the client and the DB2 server's authentication setting.

Example 18-1 shows how to catalog the sample database using the `db2 catalog` command and setting the Authentication mechanism to `SERVER_ENCRYPT`. `SERVER_ENCRYPT` specifies that the encrypted user ID and encrypted password are sent to the server for authentication. All requests and data flow are unencrypted, in the DB2 binary format, between client and server.

Example 18-1 Sample DB2 catalog command

```
db2 catalog database sample at node db2node_name AUTHENTICATION
SERVER_ENCRYPT
```

18.1.2 JDBC type 4 driver

The JDBC type 4 driver is a pure Java, client-side implementation used to connect to an IBM DB2 Universal Database server. The type 4 driver uses the Distributed Relational Database Architecture™ (DRDA®) protocol to connect directly to the database server. When using the type 4 driver, the driver gets a list of the authentication types the server accepts and it uses one of these if possible.

If the client and server have no allowed authentication types in common, an exception is thrown and connection fails. The authentication type chosen determines the level of security between the JDBC client and the DB2 server.

Allow the client and server to negotiate the authentication type of handling authentication and encryption. It is also possible to specify programmatically which authentication type the type 4 driver can use, by setting the *securityMechanism* property on the data source.

Note: When using the type 4 driver, local installation of the DB2 client code is not required. Only the db2jcc JAR files must be copied to the WebSphere Application Server and the WebSphere environment variable DB2UNIVERSAL_JDBC_DRIVER_PATH pointed to the appropriate directory. The JAR files can be found on the DB2 server in the <DB2_root>/java directory.

For details, see the “Vendor-specific data sources minimum required settings” topic in the WebSphere Information Center.

18.2 Securing access to database data

The second important area of database security, access control, is generally broken down into two topics, authentication and authorization. Authentication is validating that a user is who they say they are, and authorization is validating access to particular data within the database. These are processes handled by the database server. When a user supplies his credentials, usually a user ID and password, to the database server during a connect request, the database engine validates these credentials. After the credentials are validated, the database engine then checks to see if the user is authorized to perform the action requested, such as connecting to the database or interacting with data in the database.

The authentication credentials that are used by WebSphere Application Server to access a database can either be programmatically supplied or provided by the various Application Server containers. When the database is accessed from a

resource reference with component/application managed authentication defined, the credentials are either supplied programmatically during the connect request or from a component-managed authentication alias defined on the JDBC connection factory.

For container-managed authentication access, the container supplies the credentials to the connection factory. For further information regarding container-managed authentication, see 17.2.2, “Container-managed authentication” on page 468.

Note: WebSphere Application Server V6.1 provides two types of data sources. WebSphere Application Server V4 data source is provided only to support Java 2 Platform, Enterprise Edition (J2EE) 1.2 applications. The other new data source runs under the J2EE Connector Architecture (JCA) connection manager and the relational resource adapter. J2EE 1.3 and 1.4 applications must use the new data source.

Each database engine has a method for granting, revoking, and storing authentication and authorization information. For example, IBM DB2 Universal Database uses the **grant** and **revoke** commands to specify authorization information. If using a database engine other than DB2, see the documentation specific to your installation. Consider the following general guidelines implementing database security:

- ▶ Require users to supply credentials to access the database. This generally requires revocation of *public* or anonymous access.
- ▶ Do not share user IDs and passwords among users.
- ▶ Specify authorization to tables, stored procedures, functions, and so on, using groups rather than user IDs. While not a true security requirement, this makes administration a lot easier.
- ▶ Grant only the minimum rights required for a user or group to accomplish the tasks assigned to them.



Part 3

Development environment

This part includes Chapter 19, “Development environment security” on page 487.



Development environment security

This chapter discusses the development environment for WebSphere Application Server V6.1.

19.1 Rational Application Developer

Rational Application Developer (formally known as WebSphere Studio Application Developer) is a comprehensive Integrated Development Environment based on Eclipse open source platform to quickly design, develop, analyze, test, profile, and deploy Web, Web Services, Java, Java 2 Platform Enterprise Edition (J2EE), and portal applications. Rational Application Developer is closely integrated with Rational products to leverage the powerful Rational Software technologies to improve the software development life cycle.

The new WebSphere Rapid Deploy feature makes Rational Application Developer easier to system, unit test, and deploy applications to WebSphere Application Server V6.1 and also provides continued and full support to WebSphere Application Server V5.0 and V5.1.

There are a few changes between WebSphere Studio Application Developer and Rational Application Developer from a security point of view. This section discusses the development tool from a security point of view. The application development security issues and WebSphere Application Server security configuration issues are already discussed in various sections of this book.

19.1.1 Securing the workspace

The primary issue from the development tool point of view is to secure the workspace where the actual application code resides. Many organizations use a code repository tool such as Concurrent Versions System (CVS) or Rational ClearCase®. Rational Application Developer uses the security mechanism provided by the repository actual tool. For setting up security for the tool used with the repository, see the appropriate tool documentation.

Regarding the security of the actual workspace, where the copy of the code resides, Rational Application Developer relies more on operating system security. To gain access to the Rational Application Developer, one must be able to log in to the machine on which Rational Application Developer has been installed. There is no separate user ID or password required to access the Rational Application Developer application. Though there is no direct authentication mechanism for Rational Application Developer access, there are a couple of indirect ways to secure from unauthorized users performing unnecessary or unwanted changes to the artifacts.

Operating system access control

Rational Application Developer keeps all the artifacts under the specified workspace directory. One way of securing the workspace is to provide access to only the developer on the machine so that other users cannot access the directory.

Windows hosted development environment

Secure the workspace directory:

1. Open the Windows Explorer, select the workspace directory you want to secure and open the properties.
2. In the Workspace Properties window (Figure 19-1), click the **Security** tab and provide proper access to the right users and groups.

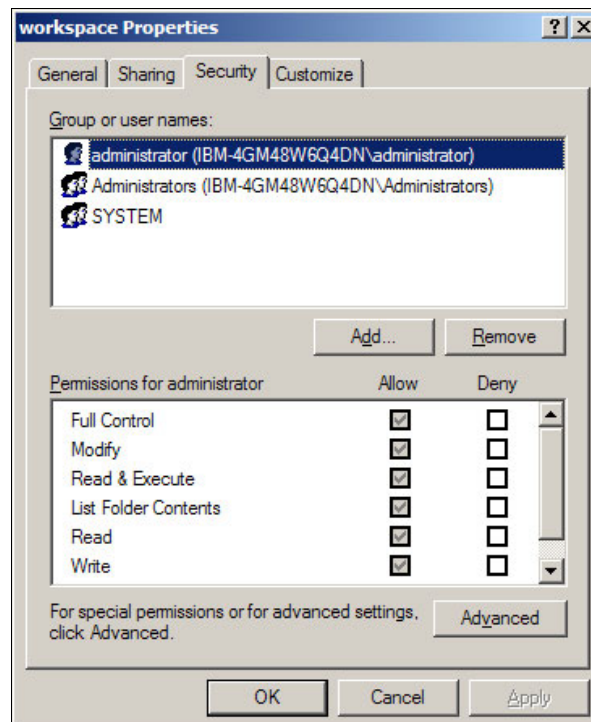


Figure 19-1 Modifying workspace directory access under Windows

If any user tries to open the workspace without proper access, Rational Application Developer is not able to open the workspace because the user ID does not have read access to the workspace directory. The following error message that is shown in Figure 19-2 is displayed.

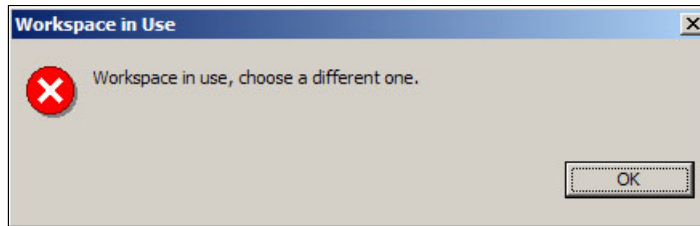


Figure 19-2 WorkSpace directory access error message

Linux hosted development environment

Under a Linux operating system, you can change the access by creating user groups, adding the user to the group, and providing workspace directory read access to the group by using the following command:

```
chmod 770 <workspace_directory>
```

This command provides complete access to the owner and the group and does not give any access to others. Another way to provide access control is at the user level by using the following command:

```
chmod 700 <workspace_directory>
```

This command provides full access to the user and others are not able to access the workspace directory at all.

Network security

Another option is to keep the workspace on the shared network drive instead of the local drive. The access to the directory can be authenticated at mounting time, mapping time, or access time and can be done by providing a user ID and password.

19.2 WebSphere test environment

The creation of a new server, configuration or modifications can be done under the server view.

Note: There is no Server perspective in Rational Application Developer. Only the Server view is available.

The server view displays the list of all the available servers and the configurations associated with each server. The server name and the host name identify a unique server. The server view also displays the status of all the servers. The *Status* can be one of the following settings:

- ▶ Starting
- ▶ Started
- ▶ Started in debug mode
- ▶ Started in profile mode
- ▶ Stopping
- ▶ Stopped

Table 19-1 describes the *State* of the server and defines the action to take based on the server configuration set. For example, when a resource associated to the server has been modified (for example, a JSP file has been updated), the project has to be re-published.

Note: If the server tools detect that a file defined to run on a particular server has changed, and the **Automatically restart servers when necessary** check box has been selected on the Server preferences page (**Window** → **Preferences** → **Server**), the server tools automatically restart that server. The Status column in the Servers view changes from Started to Stopped to Started.

Table 19-1 Possible server status

Server state	Description
Server is synchronized	Both the server configuration and the applications are in sync.
Server must be restarted	The server has to be restarted in order for the changes to take place.
Server must be restarted and republished	Either the server configuration or the applications or both have changed. The changed files must be republished.

Important: If Rational Application Developer is installed with the IBM WebSphere Application Server V6.1 Integrated Test Environment option, a WebSphere Application Server V6.1 server is automatically created when the workbench is started the first time and is viewable under the server perspective.

19.2.1 Creating a new test server

A server is a runtime environment that is used for testing the project resource. The test environment does not have to reside on the same development machine. The WebSphere Application Server V6.1 can be created in a separate machine and configured under Rational Application Developer as a test server. This increases performance because the test environment process and the development tool run on different machines.

To create a new WebSphere test server:

1. Select **File** → **New** → **Other**.
2. In the New window (Figure 19-3), click the **Show all wizards** check box. Expand the server folder and then select **Server**. Click **Next**.

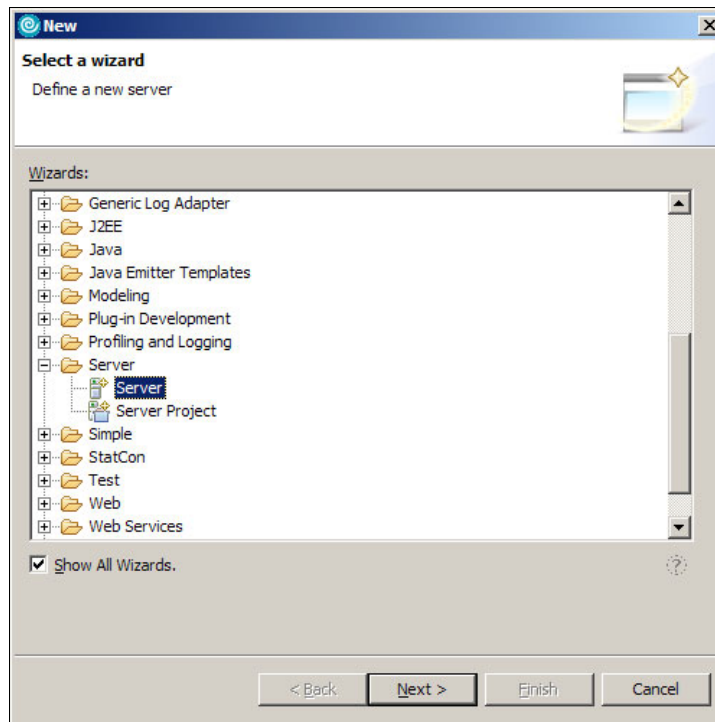


Figure 19-3 Creating a new server

3. The server creation wizard starts. Any existing server can be created and configured as a WebSphere test environment.

In the Host name field shown in Figure 19-4, provide the fully qualified Domain Name Server (DNS) name or Internet Protocol (IP) address of the remote host name that the server is running on. Or use localhost if you are planning to use a local Application Server.

In the *Select the server type* list, select the server or test environment where the resources are to be published. This can be a WebSphere Application Server V6.1 or v5.x server. The next configuration is valid only if WebSphere Application Server V6.1 is selected.

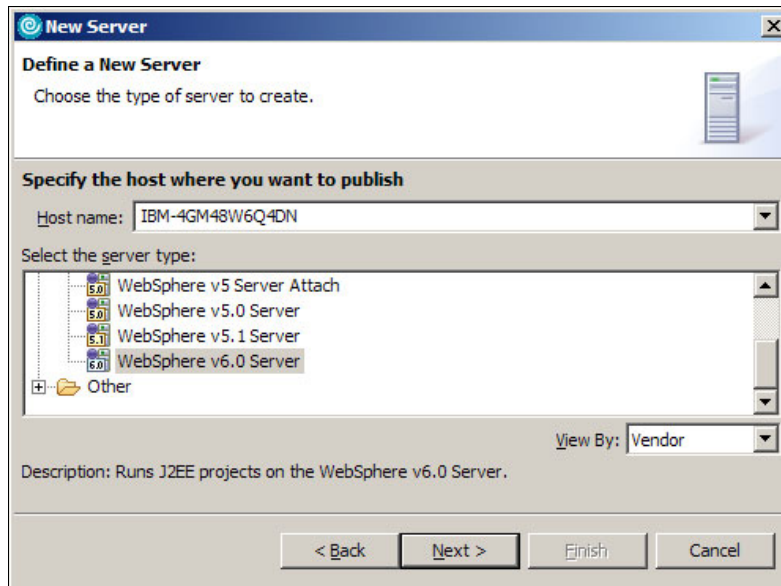


Figure 19-4 Selection of server version

4. If the host name provided has multiple WebSphere profiles running, then the WebSphere profile name field displays all the available profiles and any one can be selected as a WebSphere test environment server profile. With the default Application Server, a default profile is created. You can create new profiles in your Application Server. For more details, see “Creating a new profile” on page 502.

Click the **Detect** button (Figure 19-5) to determine the type of server for the profile under the given host name. You can use this option to check whether the profile selected is Network Deployment profile or Base/Express profile, otherwise you can manually set the type of the server.

You can leave the Enable security option unchecked at this time. You can turn the option on later. See Figure 19-5.

The screenshot shows a 'New Server' dialog box for WebSphere. The title bar says 'New Server' and the main title is 'WebSphere Server Settings'. Below the title is the instruction 'Input settings for the new WebSphere server.' The dialog contains several input fields and options: 'WebSphere profile name' (a dropdown menu), 'Server admin port number (SOAP connector port):' with the value '8880', and 'Server name:' with the value 'ITSO_V6Server'. There is an unchecked checkbox labeled 'Run server with resources within the workspace'. Under the 'Server type:' section, 'BASE or Express server' is selected with a radio button, while 'Network deployment server' is unselected. Below this, there is a text box for 'Network deployment server name:' and a note: 'The server name is in the form of: <cell name>/<node name>/<server name>. For example, localhost/localhost/server1.' A 'Detect' button is present with the text 'Click this button to detect the server type.'. At the bottom, there is an unchecked checkbox for 'Enable security' and two text boxes for 'User ID:' and 'Password:'. At the very bottom are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

Figure 19-5 WebSphere Server settings panel

5. Click **Finish**. The new server is displayed in the Servers view.

19.2.2 Enabling security for the WebSphere Test Server V6.1

The Enable Security option provides the security authentication.

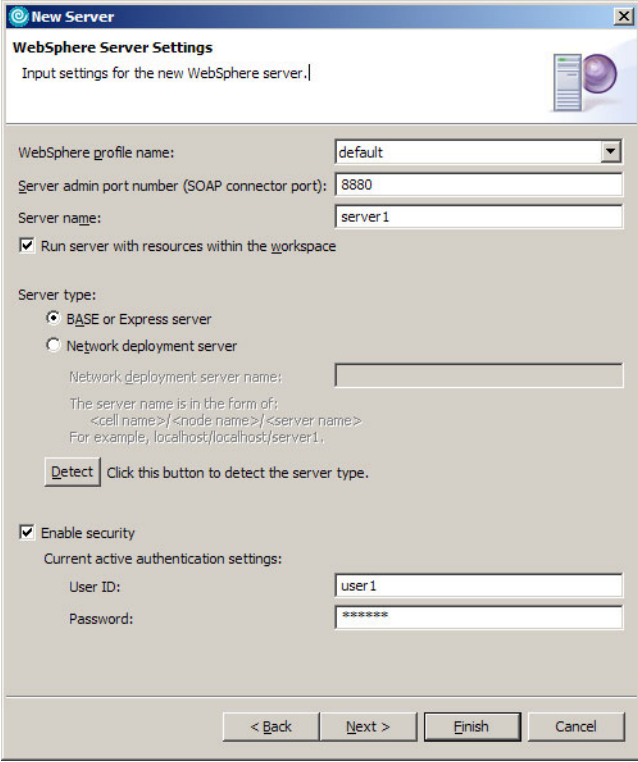
This section applies for both local and also remote WebSphere Application Server V6.1. See Chapter 3, “Administrative security” on page 49 to understand how to configure Administrative Security. Only authorized users can start and access the server. When Rational Application Developer tool starts the server, it uses the user ID and password provided under the Enable Security section.

This security configuration can be done either during the creation of the test server or after the test server has been created.

Enabling security when creating the new server

You can configure security for the test server at the time of creating the configuration. Specify the user name and password in the wizard under the WebSphere Server Settings panel as shown in Figure 19-6.

Restriction: The server must be configured with Administrative Security before configuring a WebSphere test environment under Rational Application Developer.



The screenshot shows the 'New Server' wizard window with the following settings:

- WebSphere profile name: default
- Server admin port number (SOAP connector port): 8880
- Server name: server1
- Run server with resources within the workspace
- Server type:
 - BASE or Express server
 - Network deployment server
- Enable security
 - Current active authentication settings:
 - User ID: user1
 - Password: *****

Navigation buttons at the bottom: < Back, Next >, Finish, Cancel.

Figure 19-6 Test Server startup security setting

Configuring security for an existing test server

On the Server Overview page (Figure 19-7), select the test server you are going to configure.

To configure the startup services security option, expand the Security feature. Then, provide the user name and password for the current active authentication settings defined in the server configuration.

The screenshot shows the 'Server Overview' page with several sections:

- General:** Server name: WebSphere Application Server v6.0, Host name: localhost, Runtime: WebSphere Application Server v6.0.
- Server:** WebSphere profile name: (dropdown), Server admin port number (SOAP connector port): 8880, Update server status interval (in milliseconds): 5000. Includes checkboxes for 'Enable hot method replace in debug mode' and 'Enable universal test client'.
- Publishing:** (collapsed)
- Network Deployment:** (collapsed)
- Security:** 'Enable security' is checked. 'Current active authentication settings' are: User ID: Administrator, Password: *****.

Figure 19-7 Test server startup security update

19.3 Administering and configuring the WebSphere test servers

Administering and configuring the test server can be done either within or outside of the Rational Application Developer.

Inside Rational Application Developer

On the Servers view, right-click the server and select **Run administrative console** from the context menu. This opens the browser view with the Administrative Console within the development environment.

Outside Rational Application Developer

Open your favorite Web browser and type the following URL to open the Administrative Console:

`http://<hostname>:9060/ibm/console`

The port number depends upon the profile configuration.

More information: See the other sections in this book to configure the server from the security point of view.

19.4 Enterprise application security

This section covers some of the security aspects of enterprise applications in the development environment. There are some deployment time tasks that can be performed in the development environment using the features of Rational Application Developer. These topics are discussed in the following sections:

- ▶ Configuring enterprise application security during the development phase
- ▶ JAAS entries in the deployment descriptor

19.4.1 Configuring enterprise application security during the development phase

This section discusses the mapping of application roles defined in the Web module and Enterprise JavaBeans (EJB) module to actual users and groups in the user registry. To learn more about the Web resources security and creation and configuration of the Web deployment descriptor, see Chapter 7, “Securing a Web application” on page 109. To learn more about the EJB resources security and creation and configuration of EJB deployment descriptor, see Chapter 8, “Securing an EJB application” on page 171.

The user and groups mapped under a certain role may or may not match the user and group names in the user registry after deployment. Because of this reason, we recommend you *not* to do role mapping during development or assembly time. There are situations when role mapping before deployment time can be useful and can shorten the deployment time, for example during testing.

If you have role mapping definitions in the enterprise application descriptor and you are going to deploy the application in a different environment where the mappings are not valid, you can remap your roles to the actual user registry during deployment.

To create a new security configuration:

1. Double-click **Deployment Descriptor** under the enterprise application.
2. Select the **Security** tab.

3. Click the **Gather** button shown in Figure 19-8. This gets the security roles defined for Web and EJB modules defined under the enterprise application.

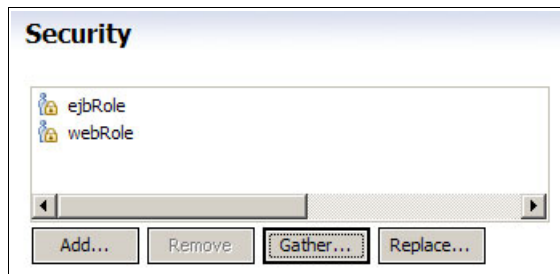


Figure 19-8 Gather the roles defined in Web and EJB deployment descriptor

You can bind the roles in the deployment descriptor to the following subjects:

- ▶ **Everyone**
Literally everyone is mapped to this role. Any user can get access to the resource that is associated with the role.
- ▶ **All authenticated users**
Any user that is authenticated against the user registry can get access to the resource that is associated with the role.
- ▶ **Users/Groups**
Only the users or users from the groups mapped to the role can access the resource that is associated with the role.
 - Mapping a user to a role:
 - i. Check the **Users/Groups** box.
 - ii. Click **Add** next to the Users area.
 - iii. Enter the user name in the field in the new window that opens, then click **Finish**.
 - iv. Later you can remove or edit a mapping that is defined.
 - Mapping a group to a role:
 - i. Check the **Users/Groups** box.
 - ii. Click **Add** next to the Groups area.
 - iii. Enter the user name in the field in the new window that opens, then click **Finish**.
 - iv. Later, remove or edit a mapping that is defined.

You can map multiple users and groups to one role.

The roles for the enterprise application are defined in the `application.xml` file shown in Example 19-1.

Example 19-1 application.xml with role definitions

```
<?xml version="1.0" encoding="UTF-8"?>
<application id="Application_ID" version="1.4" ... >
...
  <security-role id="SecurityRole_1101950918401">
    <role-name>ejbRole</role-name>
  </security-role>
  <security-role id="SecurityRole_1101950918411">
    <role-name>webRole</role-name>
  </security-role>
  <security-role id="SecurityRole_1101950918431">
    <role-name>applicationRole</role-name>
  </security-role>
</application>
```

The security role mapping is stored in the `ibm-application-bnd.xml` file shown in Example 19-2.

Example 19-2 ibm-application-bnd.xml provides the binding information

```
<?xml version="1.0" encoding="UTF-8"?>
<applicationbnd:ApplicationBinding xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:applicationbnd="applicationbnd.xml">
  <authorizationTable>
    <authorizations>
      <users name="ejbUser"/>
      <role
href="META-INF/application.xml#SecurityRole_1101950918401"/>
        <groups name="ejbGroup"/>
      </authorizations>
      <authorizations>
        <specialSubjects xmi:type="applicationbnd:AllAuthenticatedUsers"
name="AllAuthenticatedUsers"/>
        <role
href="META-INF/application.xml#SecurityRole_1101950918411"/>
      </authorizations>
      <authorizations>
        <specialSubjects xmi:type="applicationbnd:Everyone"
name="Everyone"/>
        <role
href="META-INF/application.xml#SecurityRole_1101950918431"/>
      </authorizations>
    </authorizations>
  </authorizationTable>
```

```
</authorizations>
</authorizationTable>
<application href="META-INF/application.xml#Application_ID"/>
</applicationbnd:ApplicationBinding>
```

19.4.2 JAAS entries in the deployment descriptor

WebSphere Application Server V6.1 and Rational Application Developer support extended deployment descriptors. In the extended deployment descriptor, you can configure components, such as the following examples, that you might configure during deployment time:

- ▶ Java Database Connectivity (JDBC) Provider, Data source
- ▶ Classloader mode
- ▶ Substitution variables
- ▶ Shared library
- ▶ Virtual hosts
- ▶ Authentication

Under Authentication, you can define Java Authentication and Authorization Service (JAAS) entries that you can use for the data source defined in the extended deployment descriptor as well.

To define JAAS entries:

1. Open the Deployment Descriptor for your enterprise application.
2. Select the **Deployment** tab.
3. Open the Authentication area (Figure 19-9) and click **Add**.

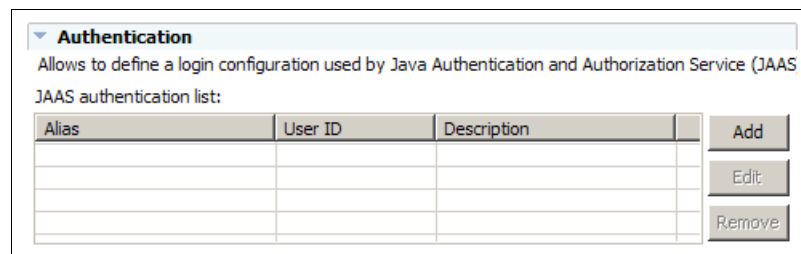


Figure 19-9 Extended deployment descriptor: Authentication

4. In the Add JAAS Authentication Entry window (Figure 19-10), complete the form as required, and then click **OK**.

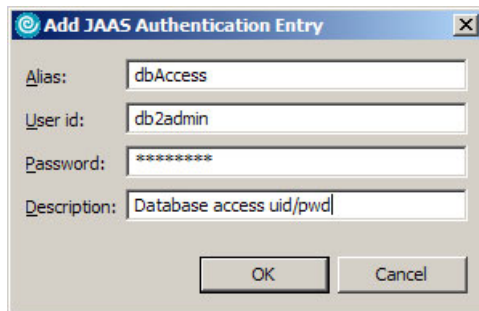


Figure 19-10 Adding a new JAAS entry

The JAAS configuration that you specify is stored in the `security.xml` file under the directory structure `ibmconfig/cells/defaultCell` under the META-INF directory of the enterprise application.

19.5 Creating a new profile for the WebSphere test server

In WebSphere Application Server V6.1, the application server instances are configured under different profiles. If you require to create a new Application Server instance, you cannot simply create a new Application Server. You must create a new profile. The Profile creation wizard can create a new profile under WebSphere Application Server V6.1.

19.5.1 Advantages of multiple profiles

Rational Application Developer creates only one profile, the default, in the test environment for the test server, `server1`. Using the Profile Creation Wizard, you can create more Application Server process. Creating a new server profile has the following advantages:

- ▶ Two different teams can test independently of one another using the same machine.
- ▶ Each application under single Rational Application Developer can use an independent test server.

19.5.2 Creating a new profile

There are different ways to create a new profile. One option is to issue the command directly from the command line. The script is located in `<RAD_home>/runtimes/base_v6/bin/Profile Creator`. The name of the command is `pctWindows.exe` in Windows or `pctLinux.bin` in UNIX.

To create a new profile:

1. Enter the `pctWindows.exe` script in Windows or `pctLinux.bin` script in UNIX to launch the Profile creation wizard.
2. In the Profile name panel (Figure 19-11), enter the profile name and select **Make this profile the default**. Then click **Next**.

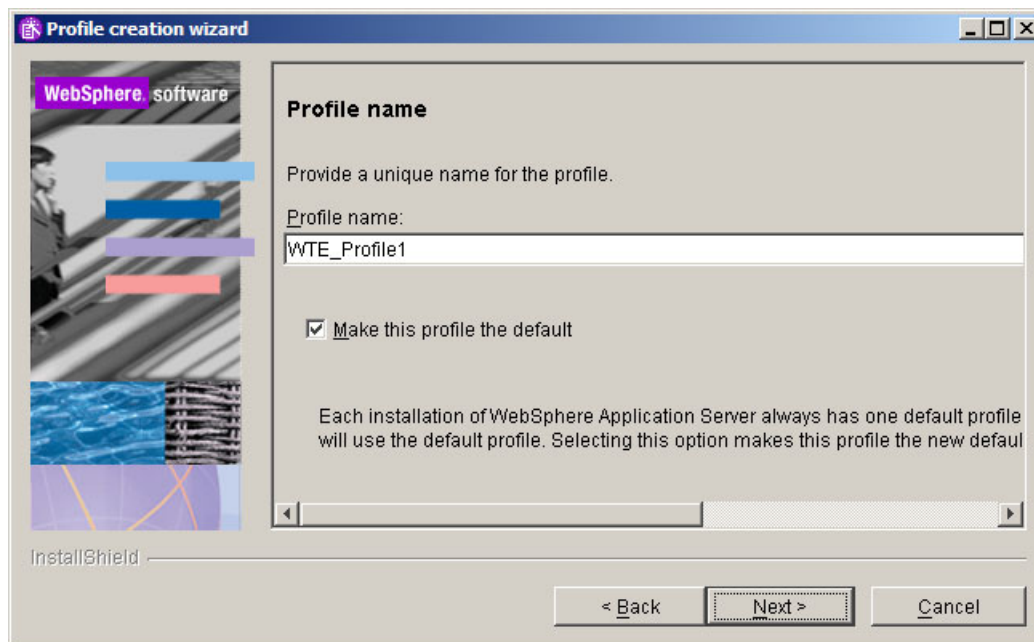


Figure 19-11 Creating new profile using profile creation wizard

3. In the Node and host names panel (Figure 19-12), enter the node name and host name and click **Next**.

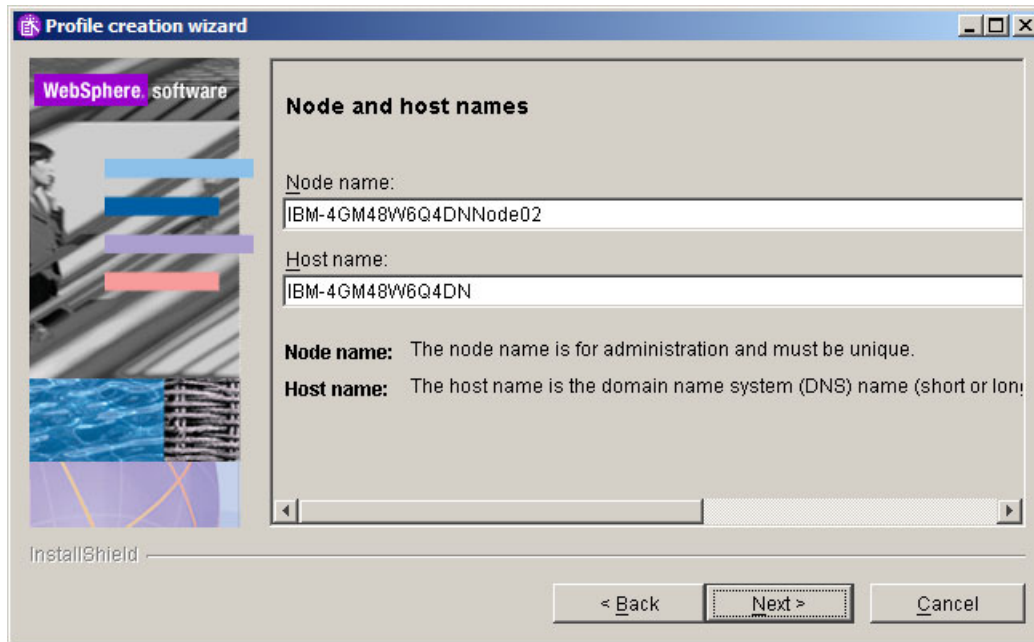


Figure 19-12 Providing node and host name

4. In the Port value assignment panel (Figure 19-13), each new profile is an instance of WebSphere Application Server and share the same runtime executables and libraries. Every time a new profile is created, the port numbers must be different than the existing servers. Click **Next**.

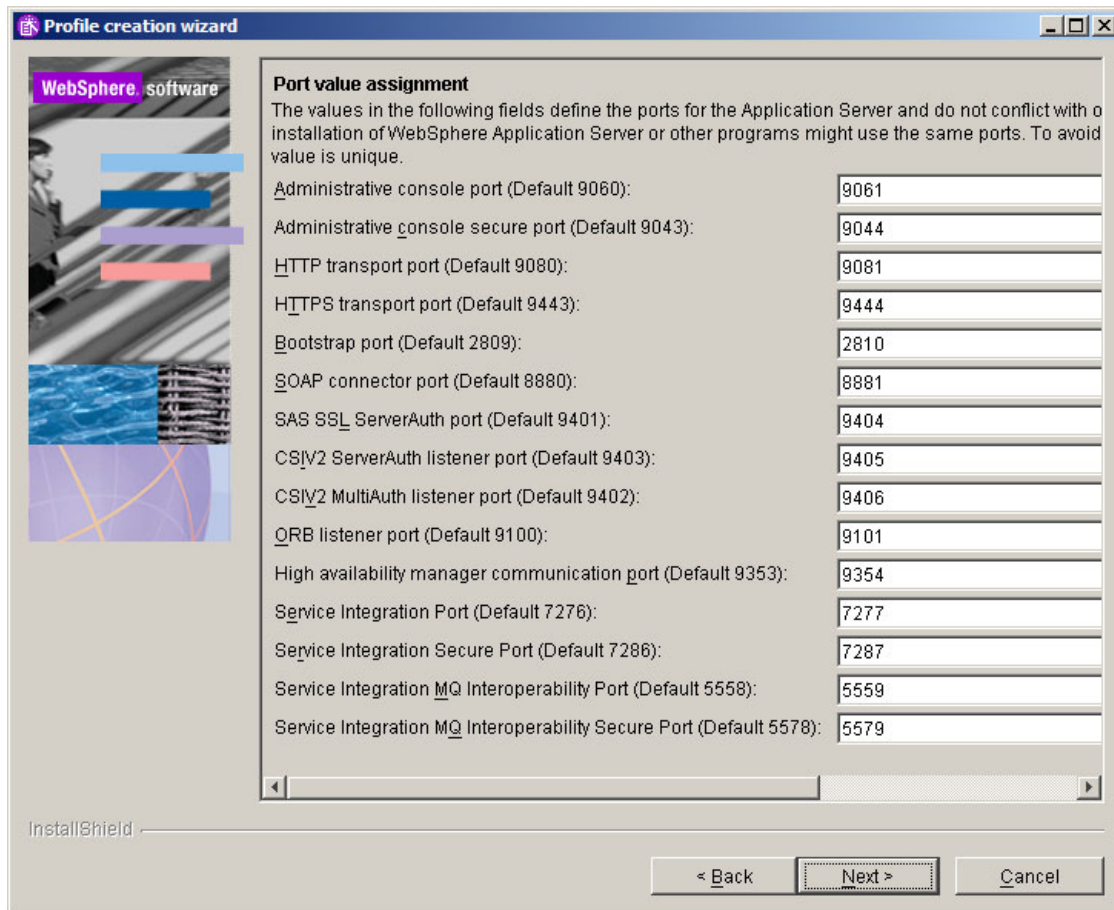


Figure 19-13 Port numbers for the new profile

5. In the Windows service definition panel (Figure 19-14), in the Windows environment, the profile can be registered as Windows services. If the new profile must be registered as services under a Windows operating system, provide the necessary information.

Important: The profile's soap port must be provided while creating WebSphere Test Environment under Rational Application Developer. See 19.2.1, "Creating a new test server" on page 492, for details.

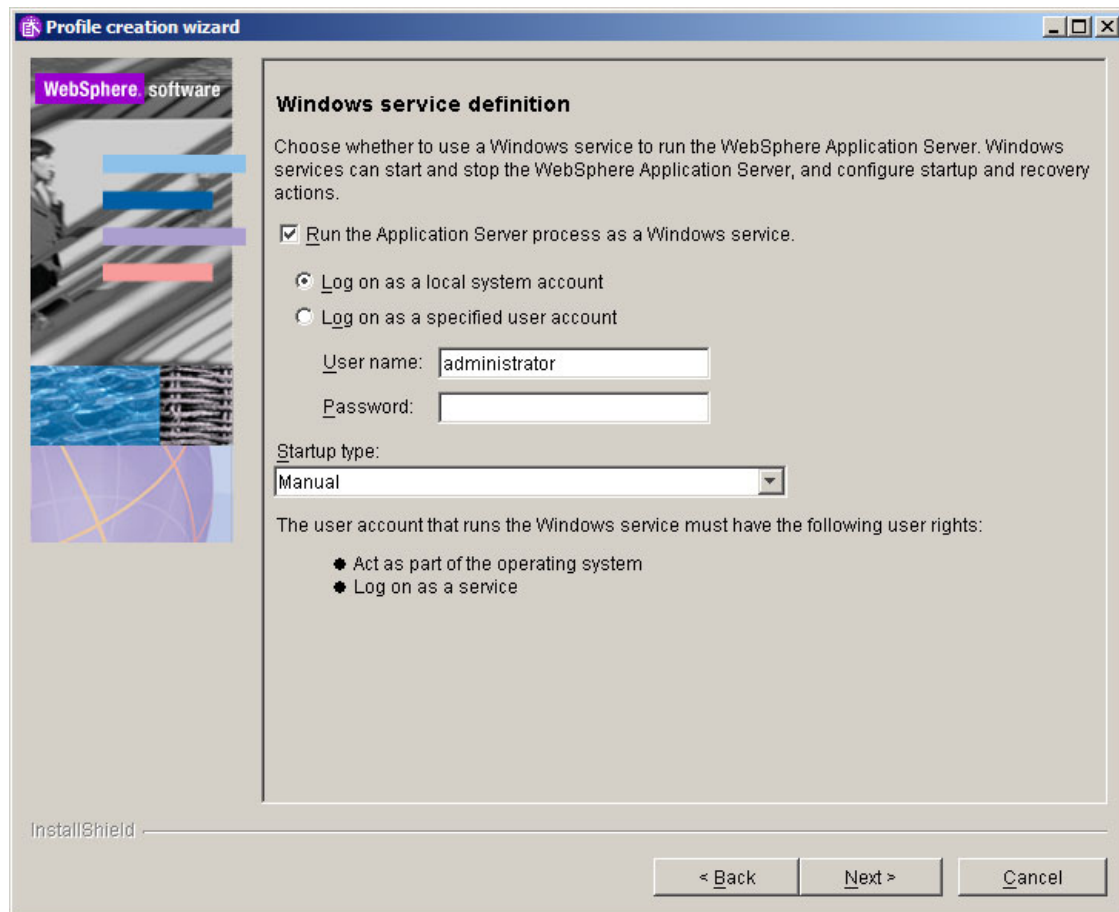


Figure 19-14 Adding profile as Windows services

Important: The specified user must have the following permissions and be logged on as root:

- ▶ Log on as service
- ▶ Act as part of the operating system.

6. On the next panel, when you see that the progress of the profile creation is complete, click **Finish** to close the wizard.

After a new profile is created, you can start the server from either the first step, the command line, or Rational Application Developer after configuring a new WebSphere test server.

19.6 Application Server Toolkit 6.1

WebSphere Application Server Toolkit 6.1 is a GUI tool for assembling a J2EE application from an application module, modifying J2EE deployment descriptor, and installing the application on a WebSphere Application Server. It is also interfaced with Rational Application Developer Tool. Application Server Toolkit extends capabilities of supporting the creation of new application.

The Application Server Toolkit is built using Eclipse V3.1.2 technology with features for team programming, debugging, J2EE application deployment, and more. The Application Server Toolkit also provides a familiar interface for developers experience with the Rational Developer Family of products. Those products extend the capabilities of the Application Server Toolkit, such as supporting the creation of new applications.

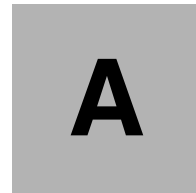
For WebSphere Application Server V6.1, the development environment no longer has the server configuration in the workspace. Server-specific configuration are set in the WebSphere Administrative Console.

Application Server Toolkit 6.1 provides the following features:

- ▶ You can now use the Application Server Toolkit to publish and test an application on any running WebSphere Application Server V6.1, either on a local or on a remote platform.
- ▶ The Application Server Toolkit has improved the WebSphere Enhanced enterprise archive (EAR) file capability, which is used for packaging and preparing applications for publishing to a WebSphere Application Server V6.1. The WebSphere Enhanced EAR file is the deployment page of the Enterprise Application Deployment Descriptor editor. It has been updated to

allow you to add resource adapters and connection factories to an enterprise application targeted for WebSphere Application Server V6.1.

- ▶ The Application Server Toolkit 6.1 support running script against a secured server. You can supply the user ID and password from the workbench to communicate to the secured server. If security exposure is a concern, you can supply authenticated information in the `sas.client.props` or the `soap.client.props` files to communicate with a secured server.
- ▶ The Application Server Toolkit allows you to create Enterprise beans from scratch. You can create, modify, and deploy Enterprise beans using the EJB creation tools.
- ▶ The Application Server Toolkit now provides a comprehensive visual XML development environment. The toolkit includes components for building document type definitions (DTDs), XML schemas, and XML files.
- ▶ The Application Server Toolkit V6.1 supports Bottom-up mapping for container-managed persistence (CMP) enterprise beans.



Additional configurations

This appendix provides additional information about configurations for WebSphere 6 related to security. It also provides = additional information about the sample applications used in this book.

Sample application for client security

This section provides a brief introduction of how to install the Itsohello Web component, the Itsohello Enterprise JavaBeans (EJB) resources and the Itsohello application clients used in this book. It has all the security features built in which are discussed in this book.

No special configuration for the WebSphere Application Server is required. You can use the default WebSphere installation settings where global security is enabled.

Figure A-1 shows the high-level diagram of the Itsohello application. It shows two enterprise beans, which are Hello and SecuredHello, as the core of the Itsohello application, installed in a WebSphere Application Server. These resources are accessible via different remote clients, such as users's browser (by using the HelloServlet servlet), four Java 2 Platform, Enterprise Edition (J2EE) Java application clients, and four thin Java application clients.

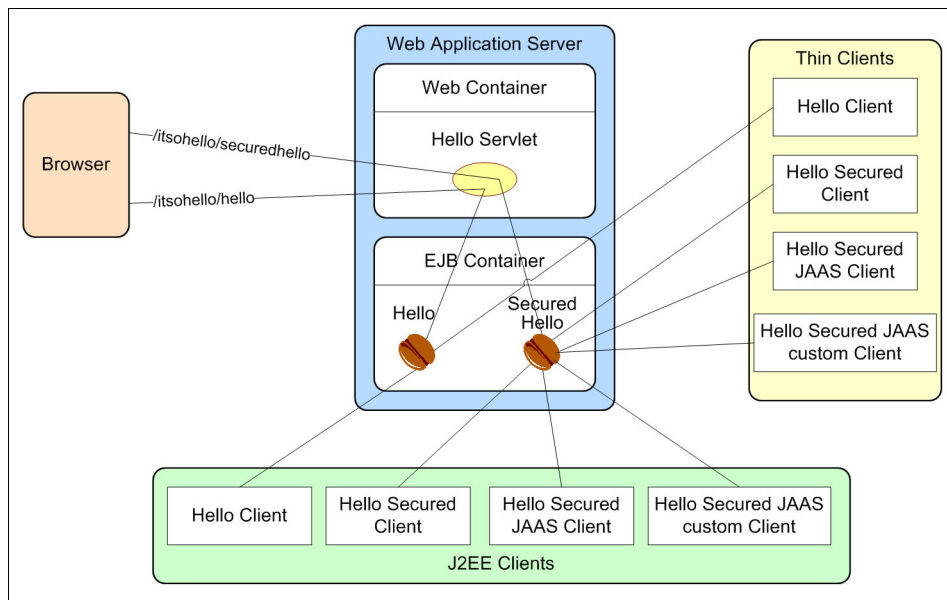


Figure A-1 Itsohello application

Installing and testing Itsohello application

To install and test itsohello:

1. Create a folder AppClient on your workstation, and extract the contents (Example A-1) to the downloaded itsohello.zip file into this folder.

Example: A-1 Contents of itsohelp.zip file

AppClient/ItsohelloEAR.ear
AppClient/runJ2EEClient.bat

AppClient/thinClient/ItsohelloEJB.jar
AppClient/thinClient/ItsohelloTHINCLIENT.jar
AppClient/thinClient/runThinClient.bat

AppClient/thinClient/keys/DummyClientKeyFile.jks
AppClient/thinClient/keys/DummyClientTrustFile.jks
AppClient/thinClient/properties/sas.client.props
AppClient/thinClient/properties/wsjaas_client.conf

2. Deploy the ItsohelloEAR.ear enterprise application into a WebSphere Application Server. It installs the Itsohello servlet and the two enterprise beans. Accept all the default setting values, including the warning for was.policy.
3. Test the installed ItsohelloEAR.ear application by accessing the beans using your browser:

- a. Test the unsecure bean by entering the following URL:

`http://<hostname>:9080/itsohello/hello`

The following reply is returned:

Message from bean: Hello to you UNAUTHENTICATED (roles:
Anonymous)

- b. Test the secure bean by entering the following URL:

`http://<hostname>:9080/itsohello/securedhello.`

A window must open, asking for user ID and password. Fill in *any* user ID and password combination specified in your active WebSphere user registry. This user registry is defined when you enabled the global security. If the user ID and password combination is correctly authenticated.

The following reply is returned:

Message from bean: [Secured] Hello to you viking (roles:
BeanGuest)

4. To test the J2EE Java application clients, edit the `runJ2EEClient.bat` script file. Change the following entries to the correct values:

```
set WAS_HOME=C:\WebSphere\AppServer
set SERVER_HOST=mka0k1my.itso.ra1.ibm.com
set SERVER_PORT=2809
```

`WAS_HOME` specifies the location of your WebSphere Application Client (if it is not installed you can use the location of WebSphere Application Server). `SERVER_HOST` and `SERVER_PORT` are the host name and Internet Inter-ORB Protocol (IIOP) port number of the WebSphere Application Server where your enterprise beans (wrapped in `ItsoHelloEAR.ear`) are located. To run the script file:

- a. Open a command prompt window.
- b. Go to the `AppClient` folder that you previously created.
- c. Run the following modified script as shown in Example A-2:

```
runJ2EEClient.bat
```

The application shows four options that correspond to the four types of J2EE application clients (Figure A-1 on page 510).

Example: A-2 Client application's opening screen

J2EE Itsohello clients:

- a. UNSECURED CLIENT.
Access the unsecured Hello bean. If you still get an authentication challenge window, just click "Cancel". Or you can also change the property "com.ibm.CORBA.loginSource" to "none" in the file "sas.client.props".
- b. SECURED CLIENT.
Access the secured Hello bean. You should be authenticated, otherwise the app will throw an exception. If you don't get an authentication challenge window, you need to change the property "com.ibm.CORBA.loginSource" to "prompt" in the file "sas.client.props".
- c. SECURED CLIENT with JAAS.
Access the secured Hello bean using JAAS. Authentication is done via JAAS.
- d. SECURED CLIENT with JAAS using custom callback handler.
Similar like (c) with custom callback handler

Please enter your choice (a/b/c/d):

- d. Select a J2EE application client you want to test. For example, for SECURED CLIENT with JAAS, press **c** and Enter.
 - e. In the authentication window that opens, enter your user ID and password. When a valid user ID and password is entered (any user ID defined in the user registry), a message similar to the following example is displayed:

```
Accessing SecuredHello bean using JAAS
Message from Hello bean: [Secured] Hello to you viking (roles:
BeanGuest)
```
5. To test the thin Java application clients, edit the `runThinClient.bat` script file. Change the following entries to the correct values:
- ```
set WAS_HOME=C:\WebSphere\AppServer
set SERVER_HOST=mka0k1my.itso.ra1.ibm.com
set SERVER_PORT=2809
```

See step 4 on page 512 for more explanation about these entries.

To run the script file:

- a. Open a command prompt window.
- b. Go to the `AppClient\thinClient` folder that you created previously.
- c. Run the following modified script:

```
runThinClient.bat
```

The rest of the procedure is the same as the procedure to test the J2EE application client.

## Sample application for testing JACC

The sample application consists of one Web module, one EJB Module, and utility jar. There are four roles defined in this application:

- ▶ WebRole
- ▶ First
- ▶ Second
- ▶ Third

WebRole is used to map the Web resource, and other roles are mapped to specific methods in the EJB.

## Web module

The Web module contains three servlets. The JACCTestServlet is the one that invokes the EJB and displays the results. The other two servlets are used to display the deployment descriptor of Web and EJB module. The JACCTestServlet is protected and mapped to WebRole.

## EJB module

The EJB module contains Stateless Session Bean and is exposed with the following four methods:

|                    |                                                                               |
|--------------------|-------------------------------------------------------------------------------|
| <b>getDD()</b>     | Unprotected method to display the deployment descriptor.                      |
| <b>getFirst()</b>  | Protected method and returns the strings. It is mapped to the “First” role.   |
| <b>getSecond()</b> | Protected method and returns the strings. It is mapped to the “Second” roles. |
| <b>getThird()</b>  | Protected method and returns the strings. It is mapped to the “Third” roles.  |

## Deploying the sample application

To deploy this sample application:

1. Open the Administrative Console.
2. Install the ITSOJACC.ear application.
3. During the installation in Step 7, which is to map security roles to Tivoli Access Manager users and groups, perform the mapping for your roles.

**Note:** The assumption here is that you already configured security using Lightweight Directory Access Protocol (LDAP) and Tivoli Access Manager, and enabled the external authorization using Java Authorization Container Contract (JACC) and Tivoli Access Manager.

## Verifying the installation

After the application installs all the policy information propagated to Tivoli Access Manager Object space, verify the installation by using the **pdamin** command interface:

1. Open the **pdadmin** command interface. Log in by using your `sec_master`.
2. Issue the command **object list**, and you see three object trees:

```
/Management
/WebSEAL
/WebAppServer
```

The `/WebAppServer` is the one where all the WebSphere related information is stored. You can drill down to the lower level of the object tree.

3. Enter the following command:

```
Object list /WebAppServer/deployedResources/Roles
```

You see several role definitions depending on your environment. The highlighted ones belong to the sample application shown in Example A-3.

*Example: A-3 Result from Object list command*

---

```
/WebAppServer/deployedResources/Roles/administrator
/WebAppServer/deployedResources/Roles/configurator
/WebAppServer/deployedResources/Roles/CosNamingCreate
/WebAppServer/deployedResources/Roles/CosNamingDelete
/WebAppServer/deployedResources/Roles/CosNamingRead
/WebAppServer/deployedResources/Roles/CosNamingWrite
/WebAppServer/deployedResources/Roles/First
/WebAppServer/deployedResources/Roles/monitor
/WebAppServer/deployedResources/Roles/operator
/WebAppServer/deployedResources/Roles/Second
/WebAppServer/deployedResources/Roles/Thrid
/WebAppServer/deployedResources/Roles/Unchecked
/WebAppServer/deployedResources/Roles/WebRole
```

---

## Testing the application installation

Open the Web browser, and type the following URL:

`http://<yourhostname>:<port>/JACC/JACCTestServlet`

You are asked for authentication. After you provide the necessary credentials, the JACCTestServlet page is displayed (Figure A-2).

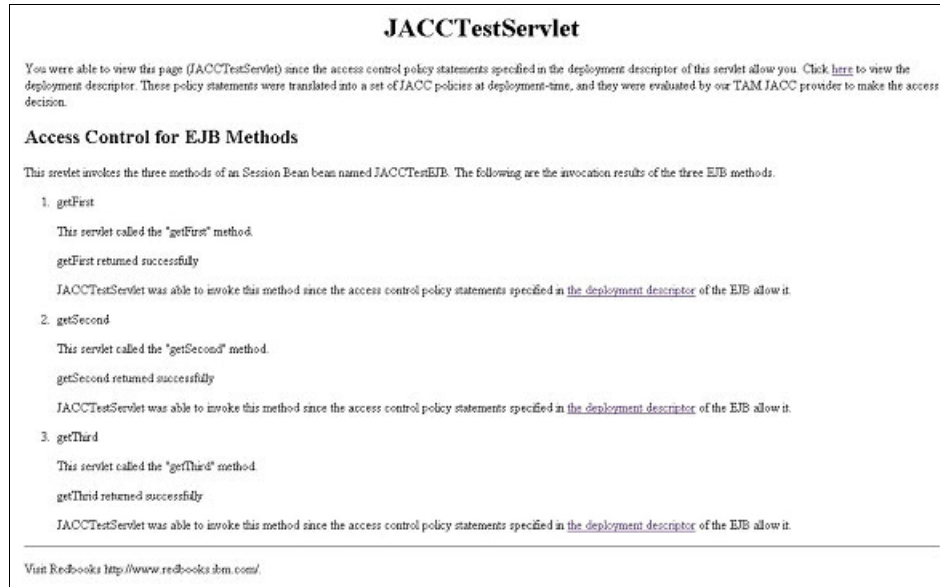


Figure A-2 Results from the test servlet

## Configuring the service integration bus and default messaging provider

This section covers the steps to define a service integration bus and Java Messaging Service (JMS) resources using default messaging provider.

The basic process entails the following tasks, which are explained in the sections that follow:

1. Creating a service integration bus
2. Adding an application server or server cluster to the bus
3. Defining a queue destination on the bus
4. Defining a JMS connection factory
5. Defining a JMS queue

**Note:** A default topic space is defined when the messaging engine is configured (defining a topic space is not demonstrated here). The steps to define a topic space are basically the same as for a queue destination and the only parameters required for the destination are name and description. After it is defined, you can modify the properties to allow send or receive actions and enforce topic access checking, among others.

## Creating a service integration bus

To define a service integration bus:

1. In the Administrative Console, select **Service integration** → **Buses**.
2. In the list of currently defined buses (Figure A-3), click **New** to create a new bus.

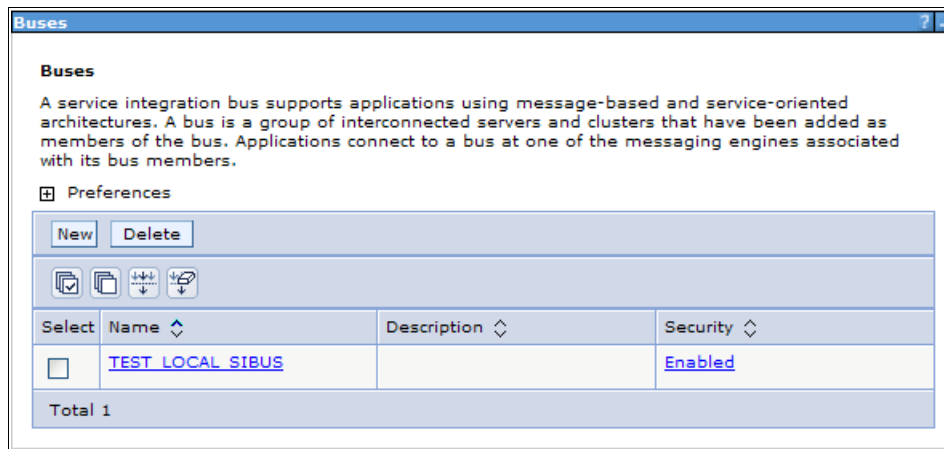


Figure A-3 Defined service integration buses

3. On the Create a new Service Integration Bus page (Figure A-4), enter a name for the bus and select **Bus security** if security is required on the bus.

**Bus security:** Selecting Bus security enables security checking on bus resources only if administrative security is also enabled. If administrative security is not enabled, selecting the Bus security option has no effect until administrative security is enabled.

Then click **Next**.

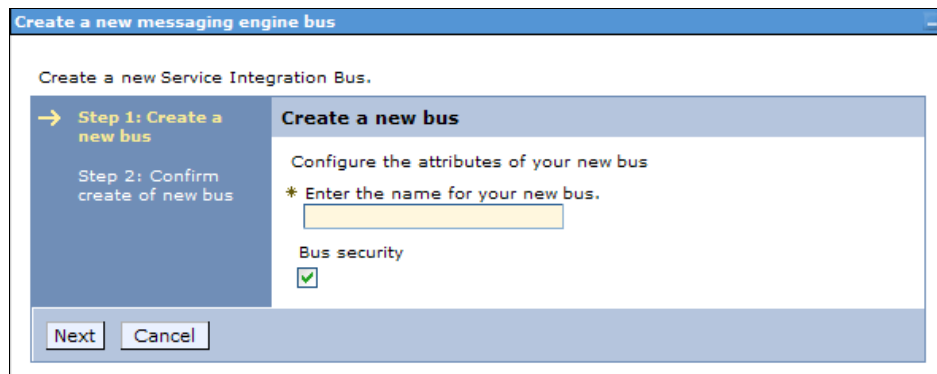


Figure A-4 New service integration bus

4. Click **Finish** to return to the list of defined buses.
5. Click **Save** to save the configuration.

## Adding an application server or server cluster to the bus

To add an application server as a bus member of the bus:

1. In the Administrative Console, select **Service integration** → **Buses**.
2. Click the name of the service integration bus to which you want to add a server.



3. On the bus properties page, under Topology, click **Bus members**.
4. On the Bus members page (Figure A-5), click **Add** to add an Application Server to the bus.

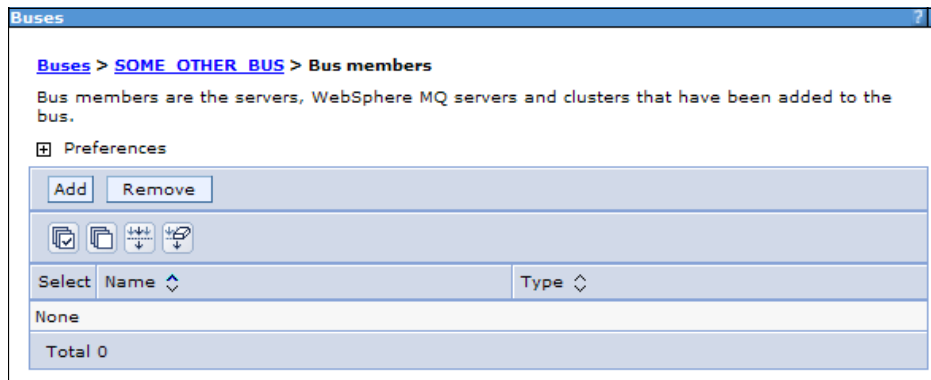


Figure A-5 Bus members list

5. On the Add a new bus member page (Figure A-6), select the server. You can also add a cluster or WebSphere MQ server as a bus member on this page. Click **Next**.

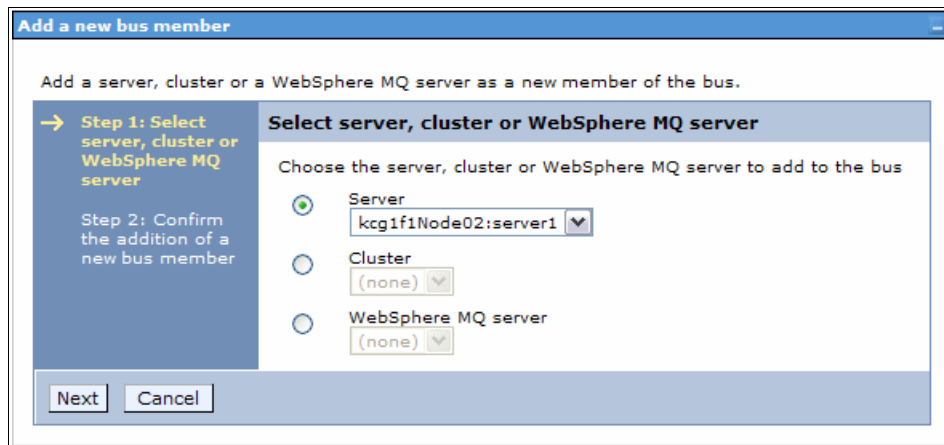


Figure A-6 Add a new bus member

6. Select **File store** or **Data store** for the message persistence. Selecting data store lets you persist messages in database. Click **Next**.
7. Click **Next** to accept the default message store setting.
8. Click **Finish** to return to the bus members page.
9. Click **Save** to save the configuration.

## Defining a queue destination on the bus

To define a queue destination on the bus:

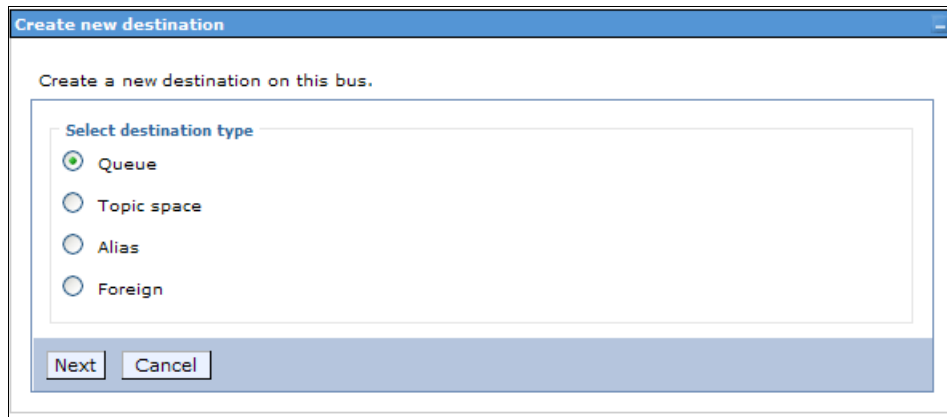
1. In the Administrative Console, select **Service integration** → **Buses**.
2. Click the name of the service integration bus that you want to add a queue destination to.
3. On the bus properties page, under Destination resources, click **Destinations**.
4. On the Destinations page (Figure A-7), click **New** to create a new destination.

| Select                   | Identifier                                                                  | Bus      | Type        | Description |
|--------------------------|-----------------------------------------------------------------------------|----------|-------------|-------------|
| <input type="checkbox"/> | <a href="#">Default.Topic.Space</a>                                         | TEST_SIB | Topic space |             |
| <input type="checkbox"/> | <a href="#">_SYSTEM.Exception.Destination.kcq1f1Node02.server2-TEST_SIB</a> | TEST_SIB | Queue       |             |
| Total 2                  |                                                                             |          |             |             |

Figure A-7 Destinations list

5. On the Create new destination page (Figure A-8), select **Queue** to define a queue destination. Click **Next**.

**Note:** The next few windows gather the information required to define the destination. The basic attributes for a Queue or Topic space are an identifier, description, and the bus member the destination is defined on.



The screenshot shows a dialog box titled "Create new destination". The main text inside the dialog reads "Create a new destination on this bus." Below this text is a section titled "Select destination type" which contains four radio button options: "Queue" (which is selected), "Topic space", "Alias", and "Foreign". At the bottom of the dialog, there are two buttons: "Next" and "Cancel".

Figure A-8 Select destination type

6. Enter the name of the queue in the Identifier field. Optionally, enter a description of the queue destination in the Description field. Click **Next**.
7. From the bus member list, select the bus member where the queue destination must be defined. Click **Next**.
8. Click **Finish** to return to the destinations page.
9. Click **Save** to save the configuration.

## Defining a JMS connection factory

To define a JMS connection factory:

1. In the Administrative Console, select **Resources** → **JMS** → **JMS providers**.
2. From the list of JMS providers (Figure A-9), click **Default messaging provider** at the appropriate scope.

**JMS providers**

A JMS provider enables messaging based on the Java Message Service (JMS). It provides J2EE connection factories to create connections for JMS destinations.

Scope: Cell=**kcg1f1Cell01**, Node=**kcg1f1Node02**, Server=**server2**

Scope specifies the level at which the resource definition is visible. For detailed information on what scope is and how it works, [see the scope settings help](#)

Node=kcg1f1Node02, Server=server2

Preferences

New Delete

| Select | Name                                            | Description                     | Scope                            |
|--------|-------------------------------------------------|---------------------------------|----------------------------------|
|        | <a href="#">Default messaging provider</a>      | Default messaging provider      | Node=kcg1f1Node02,Server=server2 |
|        | <a href="#">V5 default messaging provider</a>   | V5 Default Messaging Provider   | Node=kcg1f1Node02,Server=server2 |
|        | <a href="#">WebSphere MQ messaging provider</a> | WebSphere MQ Messaging Provider | Node=kcg1f1Node02,Server=server2 |

Total 3

Figure A-9 JMS providers list

3. On the Default messaging provider properties page (Figure A-10), under Additional Properties, click **Connection factories** to display connection factories list.

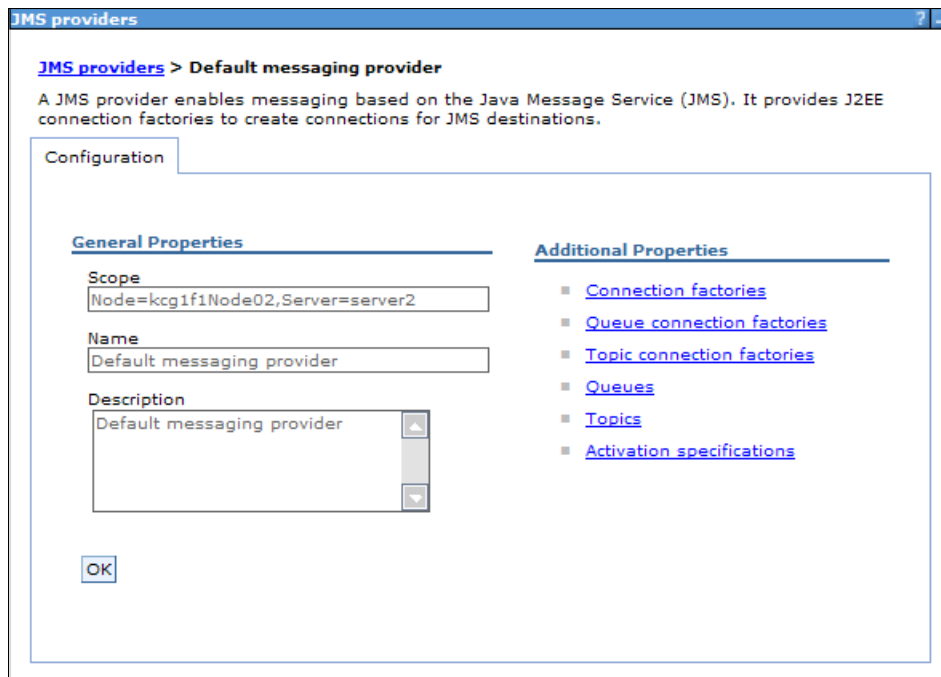


Figure A-10 Default messaging provider properties

4. Click **New** to create a new connection factory.
5. On the New connection factory properties page (Figure A-11):
  - a. Enter the name and Java Naming and Directory Interfaces (JNDI) name for the connection factory.
  - b. From the Bus name drop-down list in the Connection pane, select the local bus hosting the queue destination.
  - c. Modify additional properties on this page as required, including the Component-managed authentication alias.
  - d. Click **OK** to return to the Connection factories list.

**General Properties**

The additional properties will not be available until the general properties for this item are applied or saved.

**Administration**

Scope  
Node=kcg1f1Node02,Server=server2

Provider  
Default messaging provider

\* Name

\* JNDI name

Description

Category

**Connection**

\* Bus name  
Select...

Target

Target type  
Bus member name

Target significance  
Preferred

Target inbound transport chain

**Additional Properties**

- Connection pool properties

**Related Items**

- JAAS - J2C authentication data
- Buses

Figure A-11 New connection factory properties

6. Click **Save** to save the configuration.

## Defining a JMS queue

To define a JMS queue:

1. Click **Default messaging provider** to return to the Default messaging provider properties page.
2. Under Additional Properties, click **Queues** to display the queues list.
3. Click **New** to create a new queue.
4. On the New queue properties page (Figure A-12 on page 526):
  - a. Enter the Name and JNDI name for the queue.
  - b. From the Bus name drop-down list in the Connection pane, select the local bus hosting the queue destination.

**Note:** For Queue, Topic space, and Alias destinations, you must select the local bus where the destination is defined. For Foreign destinations, you must select the foreign bus.

After a bus is selected the page refreshes so that the Queue name field contains a list of destinations on the selected bus.

- c. From the Queue name list in the Connection pane, select the queue destination.
  - d. Click **OK** to return to the queues list.

### General Properties

---

#### Administration

Scope  
Node=kcg1f1Node02,Server=server2

Provider  
Default messaging provider

\* Name

\* JNDI name

Description

#### Connection

Bus name

\* Queue name

Delivery mode

Time to live  
 milliseconds

Priority

### Related Items

- Buses

Figure A-12 New queue properties

5. Click **Save** to save the configuration.

## Configuring WebSphere MQ as a foreign bus

To connect WebSphere MQ to the service integration bus, a local bus must be defined. See “Configuring the service integration bus and default messaging provider” on page 516 for the steps to define a service integration bus and add a server to the bus. This section gives details of the steps required to define a foreign bus and the MQ link for communicating with WebSphere MQ.



## Defining a foreign bus

To define a foreign bus:

1. In the Administrative Console, select **Service integration** → **Buses**.
2. Click the service integration bus where the foreign bus is defined to display the bus properties page.
3. Click **Foreign buses** under Topology to display the list of foreign buses defined on this service integration bus.
4. Click **New** to add a new foreign bus.
5. Enter a name and description for the foreign bus.

**Note:** If foreign destinations are used to communicate with MQ queues, enter the name of the MQ queue manager here. Otherwise any descriptive name is fine.

If applications are not allowed to send messages to the foreign bus, clear the **Send allowed** check box. Click **Next**.

6. In the Routing type list, select **Direct, WebSphere MQ link**. Click **Next**.
7. Enter the name to use to authenticate inbound message flows in the Inbound user ID field.

**Note:** All messages received from the foreign bus appear to come from the ID entered in the Inbound user ID field. This is important when the sender of the message is not authorized to access the service integration bus or puts messages to objects on the local service integration bus.

When a message enters the bus, the user ID that is stored in the message is checked against the local security. Initially the user ID is the user ID assigned to the message by the sending bus. That user ID may not have access to the local service integration bus or resources, and may not even be known to the local User Registry. By entering a locally known user ID in the Inbound user ID field, the message can transmit into the local service integration bus and all messages coming from the foreign bus appear to come from this local ID.

Enter the name to use to authenticate outbound message flows in the Outbound user ID field.

**Note:** All messages sent to the foreign bus appear to come from the ID entered in the Outbound user ID field. On WebSphere MQ, this value is in the UserIdentifier field of the message context.

When a message enters a secure foreign bus, the user ID is stored in the message. The user ID is initially set to the user ID of the message sender. This may not be appropriate, because by entering a user ID here all messages appear to come from the new user ID.

Click **Next**.

8. Click **Finish** to return to the foreign bus list page.
9. Click **Save** to save the configuration.

## Defining an MQ link

To define an MQ link:

1. In the Administrative Console, select **Service integration** → **Buses**.
2. Click the service integration bus where the foreign bus is defined to display the bus properties page.
3. Click **Messaging engines** under Topology to display a list of the messaging engines on the bus.
4. Click the name of the messaging engine where the MQ link is defined.
5. Click **WebSphere MQ links** under Additional Properties.

**Note:** *WebSphere MQ client links* are used to define an MQ link that makes the messaging engine appear as a queue manager to JMS clients connecting to it. Security for client links is handled by the roles discussed for default messaging communication in , “This chapter discusses securing the service integration bus during a WebSphere Application Server V6.1 configuration.” on page 247.

6. Click **New** to create a new WebSphere MQ link.
7. On the next page:
  - a. Enter a name and description for the MQ link.
  - b. From the Foreign bus name list, select the foreign bus, which is the foreign bus that you defined previously.
  - c. Enter a queue manager name, which is the name that the MQ link shows to the MQ queue manager. Do *not* use lowercase letters. This name is

used when defining the sender channel on the WebSphere MQ queue manager.

- d. Modify other properties as required.
  - e. Click **Next**.
8. On the next page:
- a. Enter a name for the sender channel. This must match a receiver channel defined on the MQ server.
  - b. Enter the host name and port of the MQ server queue manager.
  - c. Select the Transport chain to use.

**Note:** The transport chain determines the transport level security. Select *OutboundSecureMQLink* to use Secure Sockets Layer (SSL) when connecting to WebSphere MQ.

- d. Modify other properties as required.
  - e. Click **Next**.
9. On the next page:
- a. Enter a name for the receiver channel. This must match a sender channel defined on the WebSphere MQ queue manager.
  - b. Modify other properties as required.
  - c. Click **Next**.
10. Click **Finish** to return to the WebSphere MQ links list page.
11. Click **Save** to save the configuration.

## Defining a foreign destination

In order for an application to put a message on a queue on the MQ queue manager, the local service integration bus requires a foreign destination definition defining the foreign bus, MQ queue manager, and destination queue.

To define the foreign destination:

1. In the Administrative Console, select **Service integration** → **Buses**.
2. Click the service integration bus where the foreign destination is defined, to display the bus properties page.
3. Click **Destinations** to display the destinations list page.
4. Click **New** to create a new destination.

5. Select **Foreign** as the destination type and click **Next**.
6. On the next page:
  - a. Enter an identifier for the destination.

**Note:** The identifier must match the name of the destination on the foreign bus. In the case of WebSphere MQ, this is the name of the queue defined on the queue manager.

- b. Enter a description for the destination.
  - c. Select the foreign bus from the bus list where this destination links to. For an MQ queue this is the name of the foreign bus defined for the MQ link.
  - d. Modify other properties as required.
  - e. Click **Next**.
7. Click **Finish** to return to the Destinations list page.
8. Click **Save** to save the configuration.

## Defining a JMS queue for a foreign destination

To send messages to a foreign destination, the application must connect to the local messaging engine. The connection factory that was used to connect to the local service integration bus messaging engine can be reused here. The steps to configure the JMS connection factory are explained in “Defining a JMS connection factory” on page 522.

The steps to create a JMS queue for a foreign destination are identical to the steps detailed in “Defining a JMS queue” on page 525 with one exception. Rather than selecting a local service integration bus from the Bus name list, select the name of a foreign bus used to connect to WebSphere MQ. After the page refreshes, select the foreign destination that you just defined from the Queue name list. All other steps are the same.

## Sample application for messaging

The `JMSSampleApplication.ear` file contains a sample application (shown in Figure A-13 on page 531) to demonstrate sending and browsing messages on a service integration bus destination. The sample contains two servlets, one for sending messages to the queue and the second for browsing messages on the queue. This section details the steps to configure an Application Server’s default messaging provider, and optionally WebSphere MQ resources, to demonstrate the sample application.

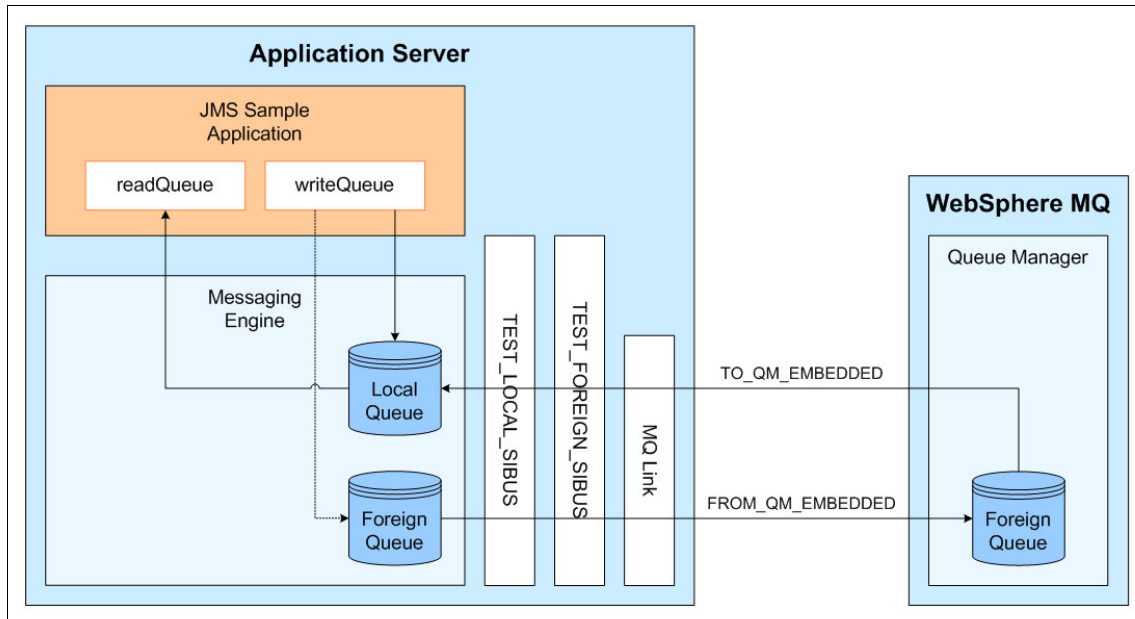


Figure A-13 JMS sample application flow

**Note:** The application is configured to send messages to and browse messages on the same queue if not testing with WebSphere MQ. If WebSphere MQ integration is being tested, then messages are sent to the *ForeignQueue* destination on the local bus. The messages are then routed automatically to the *ForeignQueue* on the WebSphere MQ queue manager. The *ForeignQueue* is defined as a remote queue to WebSphere MQ and routed back to the *LocalQueue* on the Application Server’s messaging engine. Messages cannot be received or browsed from a foreign destination, therefore the application reads them from the *LocalQueue* on the bus.

## Configuring the application server

The following resources are required to test the sample application. The sample application requires administrative security and application security to be enabled. Java 2 security is not required. The resources marked optional are only required if testing with WebSphere MQ.

If you are not familiar with configuring bus and messaging components in WebSphere, see “Configuring the service integration bus and default messaging provider” on page 516.

Defining the resources required for the sample application entail the following tasks as explained in the sections that follow:

1. Creating a service integration bus
2. Adding an application server to the bus
3. Optional: Creating a foreign bus
4. Optional: Defining an MQ link
5. Creating a queue destination
6. Optional: Creating a foreign destination
7. Creating a JMS connection factory
8. Creating a JMS queue for LocalQueue
9. Optional: Creating a JMS queue for ForeignQueue

### Creating a service integration bus

The steps to define a service integration bus are explained in “Creating a service integration bus” on page 517. Use these steps to define a service integration bus with the following properties shown in Table A-1.

Table A-1 Local bus properties

| Field        | Value            |
|--------------|------------------|
| Name         | TEST_LOCAL_SIBUS |
| Bus security | Checked          |

### Adding an application server to the bus

The steps to make an application server a member of the service integration bus are explained in “Adding an application server or server cluster to the bus” on page 518. Use those steps to add the Application Server, where the sample application is installed, to the TEST\_LOCAL\_SIBUS.

### Optional: Creating a foreign bus

The steps to define a foreign bus are explained in “Defining a foreign bus” on page 527. Use the steps, in the section mentioned, to define a foreign bus on the TEST\_LOCAL\_SIBUS with the following properties shown in Table A-2 on page 533, leaving all other properties at their default values.

**Note:** A foreign bus is only required if the sample application is connected to WebSphere MQ.

Table A-2 Foreign bus properties

| Field            | Value                                                                           |
|------------------|---------------------------------------------------------------------------------|
| Name             | <Use the queue manager name of the MQ>                                          |
| Routing type     | Direct, WebSphere MQ link                                                       |
| Inbound user ID  | janedoe                                                                         |
| Outbound user ID | User name that has access to put messages to MQ. Ask MQ administrator for this. |

**Note:** You can use the existing MQ queue manager or create a new queue manager. Make sure the queue manager name is used as the name of this foreign bus.

The user ID janedoe with password janedoe must be defined in the user registry of WebSphere Application Server V6.1.

### Optional: Defining an MQ link

The steps to define an MQ link are explained in “Defining an MQ link” on page 528. Use these steps to define an MQ link on the TEST\_LOCAL\_SIBUS to connect the application server to the WebSphere MQ queue manager by using the properties in Table A-3.

**Note:** An MQ link is only required if the sample application is connected to WebSphere MQ as a foreign bus.

Table A-3 MQ link properties

| Field                  | Value                                                                         |
|------------------------|-------------------------------------------------------------------------------|
| Name                   | TEST_LOCAL_SIBUS To MQ                                                        |
| Foreign bus name       | <name of the foreign bus that you just defined>                               |
| Queue manager name     | QM_EMBEDDED                                                                   |
| Sender MQ channel name | FROM_QM_EMBEDDED                                                              |
| Hostname               | Host name of MQ server                                                        |
| Port                   | Port that MQ queue manager is listening on. Default is 1414.                  |
| Transport chain        | OutboundBasicMQLink (if not using SSL)<br>OutboundSecureMQLink (if using SSL) |

| Field                    | Value          |
|--------------------------|----------------|
| Receiver MQ channel name | TO_QM_EMBEDDED |

**Note:** The channel names, host name, port, and transport chain all depend on values that you get from your MQ administrator.

The sender channel name must be the name of a receiver channel on the MQ server. If the channel is SSL enabled on the MQ server then transport chain must be set to `OutboundSecureMQLink`, otherwise use `OutboundBasicMQLink`.

The receiver channel name must match the name of a sender channel on the MQ server.

## Creating a queue destination

The steps to define a queue destination are explained in “Defining a queue destination on the bus” on page 520. Use these steps to define a queue destination on the `TEST_LOCAL_SIBUS` with the properties shown in Table A-4.

Table A-4 Queue destination properties

| Field            | Value                                                                                     |
|------------------|-------------------------------------------------------------------------------------------|
| Destination Type | Queue                                                                                     |
| Name             | LocalQueue                                                                                |
| Bus member       | Application server that hosts the sample application. There must only be one in the list. |

## Optional: Creating a foreign destination

The steps to define a foreign destination are explained in “Defining a foreign destination” on page 529. Use these steps to define a foreign destination on the `TEST_LOCAL_SIBUS` with the properties shown in Table A-5 on page 535.

**Note:** A foreign destination is only required if the sample application is connected to a WebSphere MQ server via a foreign Service Integration Bus definition.



Table A-5 Foreign destination properties

| Field            | Value                                           |
|------------------|-------------------------------------------------|
| Destination type | Foreign                                         |
| Name             | ForeignQueue                                    |
| Bus              | <name of the foreign bus that you just defined> |

**Note:** The name property must match the queue name on the MQ server to which messages are transmitted.

For the sample application a remote queue named ForeignQueue is defined to route messages sent to MQ back to the LocalQueue on the Application Server. This simplifies the sample application as applications cannot read from foreign destinations.

### Creating a JMS connection factory

The steps to define a JMS connection factory are explained in “Defining a JMS connection factory” on page 522. Use these steps to create a connection factory with the properties shown in Table A-6.

Table A-6 Connection factory properties

| Field     | Value            |
|-----------|------------------|
| Name      | LocalSIB_CF      |
| JNDI name | jms/cf_localSIB  |
| Bus name  | TEST_LOCAL_SIBUS |

### Creating a JMS queue for LocalQueue

The steps to define a JMS queue are explained in “Defining a JMS connection factory” on page 522. Use these steps to create a queue with the properties shown in Table A-7.

Table A-7 JMS properties for LocalQueue

| Field      | Value                |
|------------|----------------------|
| Name       | LocalQueue           |
| JNDI name  | jms/queue_LocalQueue |
| Bus        | TEST_LOCAL_SIBUS     |
| Queue name | LocalQueue           |

## Optional: Creating a JMS queue for ForeignQueue

If you defined a foreign destination for WebSphere MQ, create another JMS queue that points to the Foreign destination. Follow the steps in “Defining a JMS queue for a foreign destination” on page 530 with the properties shown in Table A-8.

Table A-8 JMS properties for Foreign destination

| Field      | Value                                           |
|------------|-------------------------------------------------|
| Name       | ForeignQueue                                    |
| JNDI name  | jms/queue_ForeignQueue                          |
| Bus        | <name of the foreign bus that you just defined> |
| Queue name | ForeignQueue                                    |

## Optional: Configuring WebSphere MQ

In order for WebSphere MQ and a WebSphere Application Server messaging engine to communicate, the following MQ objects must be defined:

- ▶ Transmit queue
- ▶ Sender channel
- ▶ Receiver channel

The sample application also requires a Remote Queue, named ForeignQueue, to be defined on the MQ server. This remote queue routes all messages placed on the queue back to the LocalQueue on the WebSphere Application Server.

After logging on to the WebSphere MQ server as a user with MQ administration privileges, define the channels and queues:

1. From the command line, run **runmqsc** or **runmqsc <queue manager name>**. This queue manager name *must* match the foreign bus name defined in “Optional: Creating a foreign bus” on page 532.
2. From the **runmqsc** prompt, enter the commands shown in Example A-4, line by line. Press Enter where you see the new line symbol (↵).

*Example: A-4 Execution of commands*

```
define qlocal(QM_EMBEDDED) usage (XMITQ) ↵
define channel(FROM_QM_EMBEDDED) CHLTYPE(RCVR) ↵
define channel(TO_QM_EMBEDDED) CHLTYPE(SDR)
conname('localhost(5558)') XMITQ(QM_EMBEDDED) ↵
define qremote('ForeignQueue') RQMNAME(QM_EMBEDDED)
XMITQ(QM_EMBEDDED) RNAME('LocalQueue') Put(ENABLED) ↵
start channel(FROM_QM_EMBEDDED) ↵
```

```
start channel (TO_QM_EMBEDDED) ↵
end ↵
```

---

3. Log out if necessary.

## Installing the sample application

To install the `JMSSampleApplication.ear` file on WebSphere Application Server V6.1:

1. In the Administrative Console, select **Applications** → **Install New Application** to start the application installation process.
2. Click **Browse** and locate the `JMSSampleApplication.ear` file then click **Open**.
3. Click **Next**.
4. On the “Step 1: Select installation options” page, click **Next**.
5. On the “Step 2: Map modules to servers” page, choose an Application Server to install the sample application. Click **Next**.
6. On the “Step 3: Summary” page, click **Finish**.
7. Click **Save** to save the configuration.

Review JMS resource references of the installed sample application.

1. In the Administrative Console, select **Applications** → **Enterprise Applications** to display the installed application list.
2. Click **JMSSampleApplication**.
3. Under References, click **Resource references**.

- On the Resources references page, go to the bottom of the page (Figure A-14). Verify that the target resource JNDI names are the same as the ones you defined in “Configuring the application server” on page 531. Click **Cancel** if no change is required, or click **OK** if changes are made to map to the JNDI names you entered previously in “Configuring the application server” on page 531.

|                            | Resource Reference | Target Resource JNDI Name                                                                  | Login configuration                                                                                  |
|----------------------------|--------------------|--------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| ation_Web.war,WEB-         | jms/cf             | <input type="text" value="jms/cf_localSIB"/> <input type="button" value="Browse..."/>      | Resource authorization:<br>Container<br>Authentication method:<br>DefaultPrincipalMapping<br>janedoe |
|                            |                    |                                                                                            |                                                                                                      |
|                            | Resource Reference | Target Resource JNDI Name                                                                  |                                                                                                      |
| on_Web.war,WEB-INF/web.xml | jms/queue_sender   | <input type="text" value="jms/queue_LocalQueue"/> <input type="button" value="Browse..."/> |                                                                                                      |
| on_Web.war,WEB-INF/web.xml | jms/queue_receiver | <input type="text" value="jms/queue_LocalQueue"/> <input type="button" value="Browse..."/> |                                                                                                      |

Figure A-14 JMS resource references

- Click **Save** to save the configuration changes.

Start the sample application:

- In the Administrative Console, select **Applications** → **Enterprise Applications** to display the installed application list.
- Select the **JMSSampleApplication** entry.
- Click **Start** to start the application. When the application is started, the following message is displayed:

*Application JMSSampleApplication on server <servername> and node <nodename> started successfully.*

## Testing the sample application

To test the sample application:

- Enter the following URL:

`http://localhost:9080/JMSSampleApplication/`

If you are testing the sample application from another machine, replace *localhost* with the host name of the application server machine.

2. Enter a valid user name and password when prompted by the browser.
3. Click **Click here to send a message**.
4. Enter Test Message 0 and click **Post Message**. The following message is displayed:

```
Security Exception occurred.
Your message was not posted to the Queue.
Access to bus was denied
```

**Note:** By default, only the special group Server has bus connector role.

5. Start **wsadmin**:

```
wsadmin -user username -password password
```

**Note:** This user must have administrator access to WebSphere Application Server V6.1 to use **wsadmin** tool.

6. From the **wsadmin** command line, enter the following command to list the users that can access the bus. By default, no user has this role.

```
$AdminTask listUsersInBusConnectorRole {-bus TEST_LOCAL_SIBUS}
```

7. To list groups that can access the bus, enter the following command. Only a special group Server is listed.

```
$AdminTask listGroupsInBusConnectorRole {-bus TEST_LOCAL_SIBUS}
```

8. Add the special group AllAuthenticated to the bus connector role:

```
$AdminTask addGroupToBusConnectorRole {-bus TEST_LOCAL_SIBUS -group
AllAuthenticated}
```

9. Save the changes:

```
$AdminConfig save
```

**Note:** Do not close the **wsadmin** console until instructed to do so, because it is used many times in the following steps.

10. From the sample application in the browser, click **Click here to send a message**.
11. Enter Test Message 1 and click the **Post Message** button. The following message is returned to your browser:

```
Your message has been posted to the Queue
```

**Persistent security exception:** The updated authorization policy might take a few seconds to take effect in a Network Deployment environment.

Restart the messaging engine for the Application Server, which can force the authorization policy to update immediately.

1. In Administration Console, select **Servers** → **Application servers**.
2. Click the name of the application server that is running the messaging engine to open the Application Server properties page.
3. Click **Messaging engines** under **Server messaging**.
4. Select the messaging engine by placing a check mark in the left column and click **Stop**.
5. After it is stopped, select the messaging engine again and click **Start**.

Alternatively, you can also stop and start the messaging engine by selecting **Service integration** → **Buses** → **<your bus name>** → **Messaging engines**.

12. Click **Click here to browse messages on the queue**. The message Test Message 1 must now be displayed.

13. From the **wsadmin** command line, enter the following command to list the groups that are in the default sender role for the bus. By default, AllAuthenticated must be the only group listed.

```
$AdminTask listGroupsInDefaultRole {-bus TEST_LOCAL_SIBUS -role sender}
```

14. From the **wsadmin** command line, enter the following commands to remove the AllAuthenticated group from the default sender role:

```
$AdminTask removeGroupFromDefaultRole {-bus TEST_LOCAL_SIBUS -role sender -group AllAuthenticated}
$AdminConfig save
```

15. Optional: Restart the messaging engine. (See the Persistent security exception note box on page on page 540.)

16. From the sample application, click **Click here to send a message**.

17. Enter the text Test Message 2 and click **Post Message**. The following message is displayed in the browser:

```
Security Exception occurred.
Your message was not posted to the Queue.
Send access to queue was denied.
```

18. From the sample application, click **Click here to browse messages on the queue**. Only the message “Test Message 1” must be displayed.
19. From the **wsadmin** command line, enter the following commands to remove the AllAuthenticated group from the browser default role:

```
$AdminTask removeGroupFromDefaultRole {-bus TEST_LOCAL_SIBUS -role browser -group AllAuthenticated}
$AdminConfig save
```
20. Optional: Restart the messaging engine. (See the Persistent security exception note box on page on page 540.)
21. From the sample application, click **Click here to browse messages on the queue**. The following message is displayed in the browser:

```
Security Exception occurred.
Browse access to queue was denied.
```
22. From the **wsadmin** command line, enter the following commands to add the AllAuthenticated group back to the sender default role:

```
$AdminTask addGroupToDefaultRole {-bus TEST_LOCAL_SIBUS -role sender -group AllAuthenticated}
$AdminConfig save
```
23. From the sample application, click **Click here to send a message**.
24. Enter the text Test Message 3 and click **Post Message**. The message posts successfully.
25. From the sample application, click **Click here to browse messages on the queue**. The following message is displayed in the browser.

```
Security Exception occurred.
Browse access to queue was denied.
```
26. From the **wsadmin** command line, execute the following commands to add the AllAuthenticated group back to the browser default role.

```
$AdminTask addGroupToDefaultRole {-bus TEST_LOCAL_SIBUS -role browser -group AllAuthenticated}
$AdminConfig save
```
27. From the sample application, click **Click here to browse messages on the queue**. The messages Test Message 1 and Test Message 3 is displayed.
28. Exit the **wsadmin** command line by typing quit.

**Roles in destination queue:** Additionally, by using the commands in 10.4, “Administering destination security” on page 257, you can modify and test the roles on the queue destination rather than using the default roles that affect access to all bus destinations not just the LocalQueue destination.

**Testing against WebSphere MQ:** To test against WebSphere MQ, map the resource reference for `jms/queue_sender` to JMS Queue `jms/queue_ForeignQueue` and restart the application. Add the sender role to group `AllAuthenticated` for this foreign destination:

```
$AdminTask addUserToDestinationRole {-bus TEST_LOCAL_SIBUS
-foreignBus <foreign bus name> -type foreignDestination
-destination ForeignQueue -role sender -user AllAuthenticated}
```

Then the previous testing steps also work for the MQ foreign destination on the foreign bus.





## Additional material

This IBM Redbook refers to additional material that you can download from the Internet as the following sections describe.

### Locating the Web material

The Web material associated with this IBM Redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246316>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Select **Additional materials** and open the directory that corresponds with the redbook form number, SG246316.

### Using the Web material

The additional Web material that accompanies this IBM Redbook includes the following files:

| <i>File name</i>    | <i>Description</i>                            |
|---------------------|-----------------------------------------------|
| <b>sg246316.zip</b> | Compressed code samples for ITSO applications |

## System requirements for downloading the Web material

The following system configuration is recommended:

|                          |                      |
|--------------------------|----------------------|
| <b>Hard disk space:</b>  | 20 MB minimum        |
| <b>Operating System:</b> | Windows or Linux     |
| <b>Processor:</b>        | P4 2.x GHz or faster |
| <b>Memory:</b>           | 1 GB or more         |

## How to use the Web material

Create a subdirectory (folder) on your workstation, and extract the contents of the Web material compressed file into this folder.

# Abbreviations and acronyms

|              |                                           |             |                                               |
|--------------|-------------------------------------------|-------------|-----------------------------------------------|
| <b>ACL</b>   | access control list                       | <b>IT</b>   | information technology                        |
| <b>AM</b>    | access manager                            | <b>ITSO</b> | International Technical Support Organization  |
| <b>API</b>   | application programming interface         | <b>JAAS</b> | Java Authentication and Authorization Service |
| <b>BA</b>    | basic authentication                      | <b>JACC</b> | Java Authorization Container Contract         |
| <b>CA</b>    | certificate authority                     | <b>JAR</b>  | Java archive                                  |
| <b>CRL</b>   | certificate revocation list               | <b>JCA</b>  | J2EE Connector Architecture                   |
| <b>CMP</b>   | container-managed persistence             | <b>JDBC</b> | Java Database Connectivity                    |
| <b>CORBA</b> | Common Object Request Broker Architecture | <b>JKS</b>  | Java Key Store                                |
| <b>CPU</b>   | central processing unit                   | <b>JMS</b>  | Java Messaging Service                        |
| <b>DD</b>    | deployment descriptor                     | <b>JMX</b>  | Java Management Extensions                    |
| <b>DNS</b>   | Domain Name Server                        | <b>JNDI</b> | Java Naming and Directory Interface           |
| <b>DRS</b>   | Data Replication Service                  | <b>JNI</b>  | Java Native Interface                         |
| <b>EAR</b>   | enterprise archive                        | <b>JRE</b>  | Java Runtime Environment                      |
| <b>EIS</b>   | enterprise information system             | <b>JSP</b>  | Java ServerPages                              |
| <b>EJB</b>   | Enterprise JavaBeans                      | <b>JSR</b>  | Java Specification Request                    |
| <b>ERP</b>   | enterprise resource planning              | <b>JSSE</b> | Java Secure Socket Extension                  |
| <b>FIPS</b>  | Federal Information Processing Standards  | <b>JVM</b>  | Java virtual machine                          |
| <b>FTP</b>   | File Transfer Protocol                    | <b>KDC</b>  | (Kerberos) Key Distribution Center            |
| <b>GIOP</b>  | General Inter-ORB Protocol                | <b>LDAP</b> | Lightweight Directory Access Protocol         |
| <b>GSO</b>   | global sign-on                            | <b>LDIF</b> | Lightweight Directory Interchange Format      |
| <b>GUI</b>   | graphical user interface                  | <b>LTPA</b> | Lightweight Third Party Authentication        |
| <b>HTML</b>  | Hypertext Markup Language                 | <b>MQ</b>   | Message Queuing                               |
| <b>HTTP</b>  | Hypertext Transfer Protocol               | <b>ND</b>   | Network Deployment                            |
| <b>IBM</b>   | International Business Machines           | <b>NIS</b>  | Network Information Service                   |
| <b>IHS</b>   | IBM HTTP Server                           |             |                                               |
| <b>IIOB</b>  | Internet Inter-ORB Protocol               |             |                                               |
| <b>IOR</b>   | interoperable object reference            |             |                                               |
| <b>IP</b>    | Internet Protocol                         |             |                                               |

|               |                                           |
|---------------|-------------------------------------------|
| <b>OMG</b>    | Object Management Group                   |
| <b>ORB</b>    | Object Request Broker                     |
| <b>OS</b>     | operating system                          |
| <b>OU</b>     | organizational unit                       |
| <b>PAM</b>    | Pluggable Authentication Modules          |
| <b>PKI</b>    | public key infrastructure                 |
| <b>RMI</b>    | Remote Method Invocation                  |
| <b>RSA</b>    | Rivest, Shamir and Adleman (algorithm)    |
| <b>SIP</b>    | Session Initiation Protocol               |
| <b>SIB</b>    | Service Integration Bus                   |
| <b>SMTP</b>   | Simple Mail Transfer Protocol             |
| <b>SOA</b>    | service-oriented architecture             |
| <b>SOAP</b>   | Simple Object Access Protocol             |
| <b>SPNEGO</b> | Simple and Protected Negotiate            |
| <b>SQL</b>    | Structured Query Language                 |
| <b>SSL</b>    | Secure Socket Layer                       |
| <b>SSO</b>    | single sign-on                            |
| <b>SWAM</b>   | Simple WebSphere Authentication Mechanism |
| <b>TAI</b>    | trust association interceptor             |
| <b>URL</b>    | Uniform Resource Locator                  |
| <b>WAR</b>    | Web archive                               |
| <b>WAS</b>    | WebSphere Application Server              |
| <b>WTE</b>    | WebSphere Test Environment                |
| <b>XML</b>    | Extensible Markup Language                |

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this IBM Redbook.

## IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 549. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Enterprise Security Architecture Using IBM Tivoli Security Solutions*, SG24-6014
- ▶ *IBM Tivoli Access Manager for e-business*, REDP-3677
- ▶ *IBM WebSphere V5.0 Security WebSphere Handbook Series*, SG24-6573
- ▶ *WebSphere Security Fundamentals*, REDP-3944

## Other publications

These publications are also relevant as further information sources:

- ▶ *IBM Tivoli Access Manager Administration Guide V6.0*, SC32-1686
- ▶ *IBM Tivoli Access Manager for e-Business Auditing Guide V6.0*, SC32-2202
- ▶ *IBM Tivoli Access Manager for e-Business Installation Guide V6.0*, SC32-1684
- ▶ *IBM Tivoli Access Manager WebSEAL Administration Guide V6.0*, SC32-1687

## Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ WebSphere Application Server, Version 6.1 Information Center  
<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp>

- ▶ WebSphere Application Server V6 Information Center  
<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>
- ▶ WebSphere Application Server - prerequisites  
<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>
- ▶ Tivoli Access Manager V6.0 Information Center  
<http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp>
- ▶ OMG's XMI Web site  
<http://www.omg.org/XMI>
- ▶ Apache Web server documentation: htaccess  
<http://apache-server.com/tutorials/ATusing-htaccess.html>
- ▶ IETF's Web site, RFC2617  
<http://www.ietf.org/rfc/rfc2617.txt>
- ▶ OASIS's Web site  
<http://www.oasis-open.org>
- ▶ Specification: Web Services Security (WS-Security)  
<http://www-106.ibm.com/developerworks/webservices/library/ws-secure>
- ▶ WebSphere MQ Web site  
<http://www.ibm.com/software/ts/mqseries/messaging>
- ▶ Sun Microsystems *Java Authentication and Authorization Service (JAAS) LoginModule Developer's Guide* and other security related APIs and articles  
<http://java.sun.com/javase/6/docs/technotes/guides/security/>
- ▶ Sun's J2EE Web site  
<http://java.sun.com/j2ee>
- ▶ Key Botzum's WebSphere hardening guide  
[http://www-128.ibm.com/developerworks/websphere/techjournal/0512\\_botzum/0512\\_botzum1.html](http://www-128.ibm.com/developerworks/websphere/techjournal/0512_botzum/0512_botzum1.html)

## developerWorks

- ▶ Articles on WebSphere Application Server security  
<http://www.ibm.com/developerworks>
- ▶ IBM WebSphere Developer Technical Journal  
<http://www-128.ibm.com/developerworks/websphere/techjournal/>

## How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)





# Index

## Symbols

.arm file 153

## A

- access check 326
- access control 172, 483
  - directives 116
- access control list 316, 318
- access control list (ACL) 14
- access decisions
  - enterprise beans 412
  - Web resources 413
- Access Manager
  - authorization engine 318
  - authorization server 302
  - aznAPI 345
  - client 323
  - client configuration 325
  - credential information 346
  - external JACC provider 420
  - integration 344
  - J2EE security 345
  - JACC 325
  - lab environment 320
  - management objects 317
  - migration 348
  - plug-in 303
  - policy 308
  - policy database 325
  - policy server 301
  - role 321
  - secure domain 301
  - security model 315
  - user registry 315
  - Web portal manager 303
  - WebSEAL 302
  - WebSphere integration 306
- Access Manager Java Runtime (AMJRTE) 322
- ACE/Server 331
- ACL 316, 318
- active directory 301
- ActiveX application client 208
- add security constraints 137
- additional CORBA configuration 225
- admin SOAP 79
- administrative group members 13
- administrative security 102
- Advanced Encryption Standard (AES) 78
- alias 250
- all authenticated 105
- all authenticated users 498
- AllowOverride 117
- alternate name 250
- AMJRTE (Access Manager Java Runtime) 322
- applet application client 208
- application client run time 208, 225
- application clients 208
- application deployment 105
- application installation 105
- application logins 97
- application programming interface (API) 15, 39
- application scenarios 4
- application security 102
- application server instance 501
- Application Server Toolkit 105
- application.xml 416
- attribute layer 217
- auditing 436
- auth\_module 114
- authentication 251, 436, 454
  - basic authentication 127
  - client certificate 127
  - digital certificate 113
  - form-based 127
- authentication and authorization APIs 302
- authentication challenge 150
- authentication configuration 221
- authentication layer 217
- authentication mechanism 51, 118, 133, 233
- authentication process 9
- authentication protocol 51
- authentication strategy 98
- authentication token 269, 283–284
- authorization 251, 436, 455
  - Web server 116
- authorization API 308
- authorization constraint 134

- authorization constraints
  - configuration 138
- authorization rules 319
- authorization server 302
- authorization service 10
- authorization table support 326
- authorization token 269, 276, 284
- AuthorizationToken 277
- AuthzPropTokenFactory 284
- aznAPI 308, 345

## B

- basic authentication 9, 113, 125, 200, 438
  - logout 132
  - test 115
- bean level delegation 186
- bmPKIX trust manager 79
- bottom-up mapping 507
- buildClientRuntime 230
- Bus Connector role 251
- bus destination 250
- bus destination roles 252
- business logic 171

## C

- callback handlers 88
- CallbackHandler 234, 237
- CCI 464
- certificate
  - authentication with LDAP 156
  - authority 74
  - expirations 74
  - filter 20
  - import 153
  - map mode 20
  - properties 74
- certificate-based client authentication 152
- certification agency 447
- client certificate 9
- client certificate authentication 201
  - options 202
- client example 214
- client ORB 218
- client security 510
- client security enablement 220
- CLIENT\_CERT 157
- client-certificate 118
- ClientContainer 97, 234

- client-side
  - programmatic login 236
- client-side certificate 152, 155
- CMS 111
- CMS keystore 153
- cn=root 19
- Common Auditing and Reporting Service 306
- Common Auditing and Reporting Service (CARS) 305
- common base events (CBE) 306
- common client interface 464
- Common Event Infrastructure (CEI) 306
- common information model (CIM) 12
- Common Secure Interoperability 216, 289
- Common Secure Interoperability Version 2 9
- Common Secure Interoperability Version 2 (CSIV2) 51
- Common Security Interoperability 199, 217
- communication channel 110
- communication types 110
- component-managed authentication 466–467
- confidentiality 436, 441
- configuration
  - custom login module 97
  - exact DN mapping 167
  - local OS user registry 24
  - SSL 118, 121
  - WebSEAL form based authentication 342
- connection factory 467
- connection object 467
- connection-based transport 217
- container contract 407, 410
- container settings 121
- container-authentication 200
- container-authentication configuration 201
- container-managed authentication 468–469
- content integrity 134
- CORBA configuration file 230
- CORBA ConfigURL 220
- CosNaming
  - roles 64
    - with a qualified name 231
    - with unqualified name 232
- creating a new profile 501
- credential list attribute 233
- credential token 9, 233
- credentials 265
- cross domain single sign-on 335
- cryptographic hardware 72

- csec\_localos 24
- CSIV2 9, 199, 216
  - add-on authentication protocol 223
  - inbound 201
  - inbound authentication 293–294
  - inbound transport 243
  - Security Attribute Service 217
- custom attributes 266, 276, 281
- custom authorization token 277
  - implementation 277
- custom callback handler 88
- custom CallbackHandler 240
- custom encryption 271
- custom JAAS login 88
- custom key manager 83
- custom login configuration 469
- custom login form 131
- custom login module 89
  - configuration 97
- custom principal 95
- custom propagation token
  - implementation 282
- custom single sign-on token 279
- custom token 271–272
- custom trust manager 80
- custom user registry 8, 26
  - DB2 33
  - development 29
  - sample 29
- CustomLoginModule.java 90
- CVS 488

## D

- data constraint 134
- Data Encryption Standard (DES) 78
- database
  - authentication types 480
  - connection security 480
  - securing access 483
  - security 479
- DB2
  - custom user registry 33
  - legacy CLI-based Type 2 provider 481
  - libraries 34
  - Universal Database 480
  - Universal JDBC provider 481
- DB2UserRegistrySample 33
- DB2UserRegistrySampleTest 34

- decision-making server 303
- declarative security 102, 132, 174
- default authorization token 276
- default method 469
- default propagation token 280
- default search settings 20
- default single sign-on token 279
- default token 270
- DefaultPrincipalMapping 97, 469
- delegation policy 172, 193
- deployment
  - application 105
- deployment descriptor 128
- deployment descriptor mapping 348
- deployment tools contract 405, 410
- desktop single sign-on 332
- destination security 257
- development environment
  - Linux 490
  - Windows 489
- digital certificate
  - authentication 113
- directory administrator 13
- directory directive 114
- Directory Enabled Network (DEN) 12
- Directory Information Tree (DIT) 11
- Distinguished Name 158
- Distinguished Name (DN) 11, 14, 16
- Distributed Management Task Force (DMTF) 12
- Distributed Relational Database Architecture (DR-DA) 483
- Distributed Replication Service 285
- DN 158
- doAs() 235
- Domain Name Server (DNS) 15
- downstream propagation 289, 293
  - scenario 290
- DRS 285
- DRS Replication Domain 285
- dummy password 337
- dumpNameSpace 229
- dynacache 285
- dynamic module updates 420
- dynamic resources 136
- dynamic Web projects 129

## E

- EAR file 158

- eavesdropping 431
- e-business infrastructure 307
- e-community single sign-on 335
- EIS systems 464
- EJB 171
  - authenticator 9–10
  - container access security 199
  - declarative security 174
  - descriptor file 175
  - method access control 180
  - method level delegation 190
  - method permissions 181
  - modules 174, 514
  - policy context identifier 406
  - programmatic security
    - sample code 198
  - security methods 197
  - security roles 174
- ejb-jar.xml 134, 187
- EJBMethodPermission object 412
- embedded HTTP Server 118
- embedded messaging security 250
- embedded Tivoli Access Manager 346
  - client 323
  - disable 426
  - enable 422
- enable SSL 154
- encryption 431
- enhanced TAI interface 355
- enhanced TAI++ 266
- ensure all unprotected 105
- enterprise application security 497
- enterprise bean clients 9
- Enterprise JavaBeans 171
- Enterprise JavaBeans (EJB) 233
- entity beans 171
- EPAC 346
- Everyone 105, 498
- Everyone role 219
- exact Distinguished Name 167
- exact DN mapping 167
- exchange certificates 119
- exclude 185
- external authorization engine 420
- externalized security 298

**F**

- Federal Information Processing Standard (FIPS)

- 48, 78
- federated repository 39
- file-based
  - registry
    - testing 33
    - repositories 39
    - user registry 29
- FileRegistrySample 29
- filter 339
- foreign bus 250, 472
- foreign destination 529
- foreign service integration bus security 458
- foreign system integration bus 532
- form login
  - authentication mechanism 128
  - configuration 129
- form-based
  - authentication 128, 341
  - login 128
  - logout 131

## G

- General Inter-ORB Protocol (GIOP) 217
- GET method 136
- getCallerPrincipal() 172, 197
- getConnection() 468
- getUniqueID() 271
- GIOP 217
- global administrative group members 13
- global security 102, 127, 494
- global sign-on 335
- global sign-on (GSO) 470
- group 105
- group filter 20
- group ID map 20
- group member ID map 20
- groupDisplayName 27
- groupSecurityName 27
- groupUniqueid 27
- GSO 308, 339

## H

- Health Insurance Portability and Accountability Act (HIPAA) 305
- horizontal propagation 285
  - dynacache 285
  - JMX 286
- htaccess 117

- HTML pages 109, 136
- htpasswd utility 114
- HTTP 76, 447
  - basic authentication 113
  - cookie 128
  - method security 136
  - methods 133
  - plug-in 118, 122
  - transport 125
- http\_plugin.log 125
- httpd.conf 111
- httpd.conf file 116
- HTTPS 111
  - information 162
- Hypertext Transfer Protocol (HTTP) 76

## I

- IBM DB2 Universal Database 480
- IBM HTTP server
  - certificate 152
  - logs 155
  - SSL 111
- IBM Secure Authentication Service 51, 216
- IBM Tivoli Directory server 22
- IBM Tivoli Directory server V5.2 10
- ibm\_security\_logout 131
- ibm\_ssl\_module 111
- ibm-ejb-jar-ext.xmi 191
- ibmX509 trust manager 79
- identification 436
- Identity assertion 267
- Identity propagation
  - definition 267
- ignore 339
- IIOp 76, 126, 216
- IIOp over SSL 242, 245
- IIOp over TCP/IP 244
- iKeyman tool 21, 74, 152
- import certificate 20, 153
- inbound transport 201
- InboundBasicMessaging 473
- InboundBasicMQLink 473
- InboundSecureMessaging 473
- InboundSecureMQLink 473
- inetOrgPerson 335
- initial login 266–267
- installation
  - applications 105

- integrity 431, 436, 439
- internal login 98
- Internet Inter-ORB Protocol 216
- Internet Inter-ORB Protocol (IIOp) 76
- interoperability mode 292
- Interoperable Object Reference 218
- IOR 218
- iPlanet 301
- isCallerInRole() 172, 198
- isolate roles 134
- ITSObank application 127
- Itsohello application 511
- iv-creds 337
- iv-groups 337
- iv-user 336

## J

- j\_password 131
- j\_security\_check 128
- j\_username 131
- J2C 99
  - J2C authentication data 99
  - J2EE 1.3 specification 408
  - J2EE application client 208
  - J2EE Connector Architecture (JCA) 57, 464
  - J2EE Connector security 99, 466
  - J2EE deployment tools 405
  - J2EE programmatic security 142
  - J2RE (Java 2 Runtime Environment) 230
- JAAS 85, 142, 173, 265, 306
  - authentication 468
  - authentication entries 500
  - callback handler 88
  - configuration 87
  - framework 345
  - login module 89, 233, 271
    - authentication alias 100
  - principal 95
  - programmatic login 238
  - subject 233
- JAAS authentication alias 99
- JAAS login 94
- JAAS login sequence 94
- JACC 308, 403
  - access decisions 410
  - policy context identifiers 414
  - policy propagation 415
  - provider 321

- sample 427, 513
- specification 414
- WebSphere 408
- WebSphere extensions 414
- Java 2 Connector (J2C) 468
- Java 2 Platform, Enterprise Edition (J2EE) 306
- Java 2 Runtime Environment (J2RE) 230
- Java 2 security 85
- Java archive (JAR) 30, 36
- Java Authentication and Authorization Service 85, 142, 306
- Java Authentication and Authorization Service (JAAS) 10, 89
- Java Authorization Container Contract (JACC) 408, 420
- Java Authorization Contract for Containers 403
- Java build path 33
- Java client authentication protocol 216
- Java client configuration 220
- Java Cryptography Extension (JCE) 48
- Java Database Connectivity (JDBC) 43
- Java Generic Security Service (JGSS) 369
- Java keytool 74
- Java Management Extensions 287
- Java Messaging Service (JMS) 451, 468
- Java Naming and Directory Interface (JNDI) 468
- Java Native Interface 208
- Java Network Launching Protocols (JNLP) 210
- Java Secure Socket Extension (JSSE) 48
- Java Server Pages
  - JSP 132
- Java Server Pages (JSP) 209
- Java Web Start 209
- java.security.Policy object 403, 408
- JCA (J2EE Connector Architecture) 57, 464
- JDBC
  - data source provider 481
  - type 2 driver 482
  - type 4 driver 483
- JMS 522
  - messaging services 248
  - objects 530
    - define 522
- JMS clients
  - application clients 451
  - Message-Driven Bean 451
- JMX 287
- JMX administration 285
- JNI 208

- JSPs 109
- JSR 115 403
- junction
  - configuration 336

## K

- key configuration 20
- key distribution center (KDC) 362
- key manager 83
- key store 111
- key stores 74
- KeyFile 112
- KeyStore 230

## L

- launchClient 225
- LDAP 14, 27, 76, 325
  - authentication
    - test 150
  - certificate filter 158
  - client 13
  - configuration 14, 149
    - ldap.sth 148
  - keystore 20
  - module trace 151
  - repositories 39
  - server 147
  - server certificate 20
  - test 19
  - test SSL 21
  - user registry 8, 10
    - test 19
  - users 14
- ldap.prop 148
- Lightweight Directory Access Protocol 325
- Lightweight Directory Access Protocol (LDAP) 14, 22, 76
- Lightweight Intranet Person Schema (LIPS) 12
- Lightweight Third Party Authentication 9, 268
- Lightweight Third Party Authentication (LTPA) 15, 51, 326, 366
- LoadModule 111, 114, 149
- local
  - operating system 52
    - registry 15
  - OS 26
  - OS user registry 8
    - test 26

- local OS
  - user registry 23
- local replica 325
- LocalOS 10
- log on as 23
- log on as a service 23
- logical roles 134
- login form 128
- login module 89, 233
- login process 235
- login sequence 94
- login-config 158
- LoginContext 234
- LoginModule 265
- LoginModule interface 89
- logout 163
- lower administration 298
- LTPA 9, 128, 268, 283
  - cache 401
  - cookie 308
  - token 396
- LTPAToken 269
- LtpaToken2 279
- LTPAToken2Factory 284

## M

- manual policy propagation 418
- mapping
  - administrator role to group 62
  - administrator role to user 61
  - CosNaming role to user 65
- marker interfaces 268
- master authorization database 316
- master server DN 14
- message layer authentication 200
  - options 202
- message level security 431
- message-driven beans 172
- messaging
  - engine 249
  - sample application 530
- method access control 180
- method level delegation 190
- method permissions 181
- method-level delegation policies 196
- MQ link 453, 528, 533
- multi-phase negotiation 355
- multiple profiles 501

- mutual SSL 121, 379

## N

- NameServiceServerRoot 230
- naming model 11
- netstat 21
- netstat reports 21
- network identity 298
- network information service 24
- network security 490
- new application login module 98
- new test server 492
- NIS 24
- non-repudiation 436
- non-secure HTTP 124

## O

- OAM 455
- Object Authority Manager 455
- Object Management Group 216, 289
- Object Management Group (OMG) 51
- Object Request Broker 216
- Object Request Broker (ORB) 79
- OMG 216, 289
- operating system access control 489
- ORB 216
- ORB object 229
- OrgContainer entity 42
- OrgContainer.Delimit 42
- outbound transport 201
- out-of-box (OOBE) xv
- overhead 432

## P

- PAM 85
- pctLinux 502
- pctWindows 502
- pdadmin 303
- PDLoginModule 326, 346
- PDPermission 308, 346
- performance 432
- personal certificate 74, 152
- PKCS12 152
- pluggable application client 208
- Pluggable Authentication Module 85
- plug-in configuration 123
- plug-in file 122

- plugin-cfg.xml 123
- policy context 405
  - identifier 405
  - identifier (contextID) 411
- policy propagation 415
- policy server 301, 423
- policy store 303
- POP 316, 318
- port
  - 443 112
- POST method 136
- primary administrative user 54
- principal 95, 146
- programmatically
  - login 233
    - client-side 236
    - server-side 173
  - security 102, 141, 197
  - sample 144
- programmatically J2EE security 197
- programming authentication 99
- programming authorization 99
- propagation login 267–268
- propagation token 269, 280, 284
- PropagationToken 281–282
- property extension repository 44
- protected 133
- protected object policies 316, 318
- protected object space 316
- ProtectionDomain object 412
- protocol 436
- provider contract 408, 410
- proxy LoginModule 88
- public keys 118–119

## Q

- queue 250
- queue destination
  - define 520

## R

- RACF 27
- Rational Application Developer 33, 105, 129, 488
- Rational ClearCase 488
- Redbooks Web site 549
  - Contact us xix
- registry master 13
- registry replica 13

- Relative Distinguished Name (RDN) 11, 42
- Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) 51
  - request consumer 439
  - request generator 439
  - required privileges 23
  - res-auth 467
  - resource collection 134
  - resource name 137
  - resource reference 469
- RMI/IIOP 199
  - RMI/IIOP authentication protocol 221
  - RMI/IIOP transport channel protection 204
- RMI\_INBOUND 272, 290
- RMI\_OUTBOUND 290
- role link 135
- role mapping 105–106
- role name 134
- RoleConfiguration 409
- RoleConfiguration interface 326
- RoleConfigurationFactory 409
- roles 134
- root authority 24
- RSA authorization API 331
- RSA SecurID token authentication server 331
- rser registry
  - LDAP 10
- Run-As
  - caller mode 186
  - delegation policy 186
  - mapping 105, 193
  - mode 186
  - mode mapping 172
  - role mapping 186
  - server 191
- RunAs identity 212
- RunAsRole 187
- RunAsRole security role 187

## S

- sample application 5
- sample configuration 5
- Sarbanes-Oxley (SOX) Act 305
- SAS 289
- sas.client.props 220
- scenario 4
- Secure Association Service 289
- Secure Authentication Service 9, 51



- secure client 227
- secure domain 301
- Secure Sockets Layer 110
- Secure Sockets Layer (SSL) 49
- secure thin client 233
- securing connection 242
- security 430
  - attribute 266
  - attribute propagation 292
  - authentication 9
  - aware 141
  - challenge 228
  - constraints 133
    - configuration 137
  - enterprise 429
  - identity 188
  - methods
    - sample 143
  - role
    - Web module 132
  - role reference 134
  - role references 172, 175
  - token 439
  - transport channel 447
- security and authorization constraint 140
- Security Attribute Service 199
- security constraints 211
- Security Workbench Development Environment for Java (SWORD4J) 56
- securityMechanism 483
- security-role-ref 134, 175
- self-signed certificates 74, 118
  - create 119
- server creation wizard 493
- server ORB 219
- server perspective 491
- server Status 491
- server user identity 54
- service context 217, 219
- Service Integration Bus 248–249
  - integrating with MQ 453
  - security 253
- Service Integration Bus (SIB) 76, 247
- service principal name (SPN) 362
- Service Provider Programming Interface (SPI) 39
- service-oriented architecture (SOA) 453
- Servlet
  - getRemoteUser() 142
  - getUserPrincipal() 142
  - isUserInRole() 142
- servlet policy context identifier 405
- servlet policy enforcement 407
- servlet security 142
- servlet security methods 142
  - sample code 143
- servlets 109
- session beans 171
- Session Initiation Protocol (SIP) 76
- Session Management Server (SMS) 303
- setspn tool 362
- SIB 76
- signer certificates 74
- Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO) xv
- simple junctions 335
- Simple Object Access Protocol (SOAP) 76
- Simple WebSphere Authentication Mechanism (SWAM) 9, 51, 326
- single sign-on 128
- single sign-on (SSO) 16, 51
- SingleSignonToken 279
- SIP 76
- SMTP 447
- snoop 125
- snoop servlet 125
- SOAP 76
  - binding 433
  - message security 433
- SPNEGO protocol 332
- spoofing 430
- SSL 110–111, 124, 216
  - certificate 152
  - configuration 69, 111, 222, 242
  - entry 119
  - handshake 163
  - inbound channel 122
  - module 111
  - repertoire 118
  - settings 204
  - test 112
  - testing 124
  - Web container 121
  - Web server and WebSphere 118
- SSLEnable 112
- SSO 128
- stand-alone
  - custom registry 8, 10
  - LDAP registry 14

- Lightweight Directory Access Protocol (LDAP)
  - 10
- stateful security context 219
- stateless security context 219
- static content 110
- static resources 136
- static Web resources 110
- step-up authentication 333
- strong private key protection 154
- subject 265
- SubjectDN 158
- supply 339
- system capacity 432
- system integration bus
  - define 517
- system logins 98

## T

- TAI 308, 337, 354
- TAIResult 355
- tampering 431
- test
  - LDAP 19
  - user registry 26
- testing
  - client certificate 161
  - SSL 124
- testing SSL 112
- thin application client 208, 228
  - running 230
- thin Java application client 238
- timestamp 285
- Tivoli Access Manager 266, 298, 305
  - for business integration 450
  - GSO database 470
  - policy server 470
  - principal 470
  - Trust Association Interceptor (TAI) 397
- Tivoli Directory Server 12, 301
- Tivoli global sign-on 308
- token factory 283
- token framework 268
- tokens 266
- topic
  - roles 252
  - space 250
  - space roles 259
- trace 167

- transport 123
  - chain 121
  - channel 110, 126
  - channel encryption 199
  - channel security 447
  - guarantee
    - confidential 140
    - integral 140
    - none 140
- transport guarantee
  - Web module
    - transport guarantee 140
- transport layer 134
- Transport Layer Security (TLS) 447
- transport level security 431
- transport security 109, 456, 475
  - confidentiality 253
  - options 204
- Trust Association Interceptor 308, 337, 354, 380
- Trust Association Interceptor (TAI) 86, 354
- trust manager 79
- trust store 73
- trusted connection 379
- trusted relationship 354
- trusted user 378
- TrustStore 230

## U

- unauthenticated credential 219
- unchecked 105, 182, 185–186
- uniqueIdentifier 158
- uniqueUserId 27
- UNIX required privileges 24
- unprotected methods 184
- unsecure client 225
- unsecure thin client 231
- URI-based access control 306
- URL
  - bindings 137
  - definition 138
  - patterns 133
- use identity assigned to specific role 192
- use identity of caller 192
- use identity of EJB server 192
- user 105
- user account repository 52
- user data constraint 134
- user filter 20

- user ID map 20
- user registries 8, 14
- user registry 8, 114, 303
  - custom 8, 26
  - file-based 29
  - LDAP 8
  - local OS 8, 23
- user repository 8
- user role 135
- user Run-As roles 106
- user-defined objects 317
- userDisplayName 27
- UserRegistry interface 8, 26
  - method list 28
- users/groups 498
- userSecurityName 27

## V

- virtual host 112
- Virtual Member Manager (VMM) xv, 50

## W

- WCInboundDefaultSecure 121
- Web applications 109, 132
  - client certificate 157
  - module 129, 137
- Web archive file 136
- Web authenticator 9–10
- Web browser security 111
- Web client 9
- Web clients 9
- Web components 109
- Web container 118, 126, 132
  - authentication 127
  - ORB 10
  - SSL 121
  - testing SSL 124
- Web Deployment Descriptor 137
- Web module 132, 514
  - authentication method 132
  - basic authentication 127
  - client certificate authentication 127
  - form-based authentication 127
  - form-based logout 131
  - security roles 132
- Web objects 317
- Web portal manager 303
- Web proxy

- authentication server 354
- Web resources 133
- Web security
  - options 147
- Web server 124
  - .htaccess 117
  - authentication 113
  - authorization 116
  - basic authentication 114
  - certificate 387
  - client certificates 154
  - configuration 149
  - configuration files 148
  - definition 122
  - LDAP authentication 149
  - ldap.sth 148
  - plug-in file 122
  - security 111
  - SSL 118, 386
- Web services
  - security 430
  - model 436
- Web Services Interoperability Organization (WS-I) xv, 448
- web.xml 134, 405
- WEB\_INBOUND 272
- WebRole 514
- WebSEAL 295, 308
  - authentication 327
  - basic authentication 327, 339
  - certificate 387
  - client certificate-based authentication 329
  - form-based authentication 328
  - HTTP header authentication 332
  - integration 378
  - junctions 334
  - Kerberos authentication 332
  - LTPA 396
  - SPNEGO authentication 332
  - TAI 354
  - token authentication 331
- webseald.conf 327
- WebSphere
  - administration console 74, 83
  - Application Server 4
    - Toolkit 6.1 506
  - Application Server V6.0 51
  - JAAS 86
  - JACC 408

- profiles 493
- SSL 118, 121
- test environment 491
- test server
  - new profile 501
  - security 494
- WebSphere Common Configuration Model (WCCM)  
23
- WebSphere MQ
  - access control list 455
  - configuration 536
  - direct communication 452
  - integration 452
  - messaging components 452
  - security 459
- Windows required privileges 23
- workspace security 488
- WS-Authentication 437
- WS-Federation 437
- WSGUICallbackHandlerImpl 237
- WS-I Basic Security Profile (BSP) 448
- wsjaas\_client.conf 234
- WSLogin 97, 234
- WS-Privacy 437
- WS-SecureConversation 437
- WS-Security 432
  - authentication 438
  - roadmap 436
  - specification 434
  - Web site 450
- WSSecurityHelper 281
- WSStdinCallbackHandlerImpl 237
- WSSubject 88
- WS-Trust 437

## **X**

- XML
  - encryption 433
    - Web site 450
  - signature 433
    - Web site 450
- xtokenauth 331



**Redbooks**

# WebSphere Application Server V6.1 Security Handbook

(1.0" spine)  
0.875" <-> 1.498"  
460 <-> 788 pages







# IBM WebSphere Application Server V6.1 Security Handbook



**Redbooks**

**J2EE application server and enterprise application security**

**Additional security components including Tivoli Access Manager**

**Sample code and applications for security examples**

This IBM Redbooks publication is part of the IBM WebSphere V6.1 series. It focuses on security and related topics, as well as provides technical details for designing and implementing secure solutions with WebSphere. Written for IT Architects, IT Specialists, application designers, application developers, application assemblers, application deployers, and consultants, this book provides information about designing, developing, and deploying secure e-business applications using IBM WebSphere Application Server V6.1. It discusses theory and presents proven exercises performed in our lab by using sample applications.

Part 1 discusses security for the application server and its components, including enterprise applications. It focuses on administrative security and application security, which were previously known as *global security*. It includes essential information about how to secure Web and Enterprise JavaBeans (EJB) applications and how to develop a Java client using security.

Part 2 introduces additional components from the enterprise environment and discusses security beyond the application server. External components include third-party security servers, messaging clients and servers, and database servers.

Part 3 provides a short introduction to development environment security. It includes guidelines and best practices that are applicable to a secure development environment.

**INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION**

**BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)